NASA Contractor Report 189698

Formal Design Specification of a Processor Interface Unit

David A Fura
Boeing Defense & Space Group
Seattle, Washington

Phillip J. Windley
University of Idaho
Moscow, Idaho

G. C. Cohen
Boeing Defense & Space Group
Seattle, Washington

NASA Contract NAS1-18586
November 1992

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665-5525

Preface

This document was generated in support of NASA contract NAS1-18586, Design and Validation of Digital Flight Control Systems Suitable for Fly-By-Wire Applications, Task Assignment 9. Task 9 is concerned with the formal specification of a processor interface unit.

This report describes the formal specification of the design for a processor interface unit using the HOL methodology. The processor interface unit is a single-chip subsystem within a fault-tolerant embedded system under development at the Boeing High Technology Center. It provides the opportunity to investigate the specification and verification of a real-world component within a commercially-developed fault-tolerant computer.

The NASA technical monitor for this work is Sally Johnson of the NASA Langley Research Center, Hampton, Virginia.

The work was accomplished at the Boeing Company, Seattle, Washington and the University of Idaho, Moscow, Idaho. Personnel responsible for the work include:

Boeing Military Airplanes:
D. Gangsaas, Responsible Manager
T. M. Richardson, Program Manager

Boeing High Technology Center:
Gerald C. Cohen, Principal Investigator
David A. Fura, Researcher

University of Idaho:
Dr. Phillip J. Windley, Chief Researcher

Contents

List of Figures

List of Tables

1 Introduction

This report describes work to formally specify the requirements and design of a processor interface unit (PIU), a single-chip subsystem providing memory-interface, bus-interface, and additional support services for a commercial microprocessor within a fault-tolerant computer system. This system, the Fault-Tolerant Embedded Processor (FTEP), is targeted towards applications in avionics and space requiring extremely high levels of mission reliability, extended maintenance-free operation, or both. The need for high-quality design assurance in such applications is an undisputed fact, given the disastrous consequences that even a single design flaw can produce. Thus, the further development and application of formal methods to fault-tolerant systems is of critical importance as these systems see increasing use in modern society.

The work described in this report is but a first step towards developing a provably correct fault-tolerant computing platform for application to real commercial and military systems. Beyond the PIU verification task that follows this work, future formal methods targets include at least two additional application-specific integrated circuits (ASICs) and the operating system software for the FTEP system. It is expected that the lessons learned in this PIU effort will influence the future design and modeling of these components to facilitate their subsequent verification.

This report contains five major sections following this introduction, as well as several appendices containing the PIU design specification in its full detail. Section 2 describes the generic interpreter theory used to formally specify portions of the PIU design. This theory builds on previous NASA-funded work described in [Win90], with important extensions in the handling of interpreter outputs to support subsystem composition.

Section 3 explains the PIU design specification at a high level to facilitate the understanding of the formal models contained in the appendices. The specification itself was written using the HOL theorem-proving system developed at the University of Cambridge, England [Gor88].

Section 4 describes our progress in developing a transaction-based modeling approach for specifying the PIU requirements. A number of modeling candidates were investigated and a preferred approach was identified for formalization in HOL.

Section 5 describes our initial efforts at integrating our hardware design and verification environments into a single framework. A prototype M-to-HOL translator was developed and was used to translate the PIU behavioral specifications initially written in the simulation language M.

Section 6 contains a concluding discussion.

Before leaving this section, we first present an informal description of the PIU, including both its structure and an overview of its behavior. Following this we introduce the specification hierarchy developed for the PIU.

1.1 Informal PIU Description

The PIU is a single-chip subsystem providing memory-interface, bus-interface, and additional support services within the Processor-Memory Module (PMM) of the FTEP system. The PIU's position within the PMM structure is shown in Figure 1.1. A PMM, itself a single block within an FTEP Core, interconnects three internal PMM subsystems: the local processors, the local memory, and the Core Bus (C_Bus) interface.

The PMM processors (CPU0 and CPU1) are arranged in a cold-sparing configuration to enhance long-life operation. Only one processor is active during a given mission, with the choice of active processor determined during initialization. The spare processor is disabled by the PIU through assertion of the processor's *cpu_reset* input. For the first implementation of the PMM, described in this report, Intel 80960MC microprocessors are used for the local processors. They communicate with the PIU using the L_Bus bus protocol of the 80960.

Processor programs and data are stored in local electrically-erasable programmable read-only memory (EEPROM) and static random access memory (SRAM), respectively. Memory accesses are initiated by either the local processor or an external block acting as C_Bus master. In either case the PIU provides the memory interface. The features provided by the PIU include memory error correction, memory locking to implement atomic read-modify-write operations, byte accesses, and block accesses of up to 64 words. EEPROM and SRAM memory capacity in the first implementation is 1 MB (megabyte) of actual information storage each, implemented within seven 256Kx8-bit memory chips each. A (7,4) Hamming code provides single-bit error correction on memory reads.

The PIU also provides processor support features such as timers and interrupt control. Two 64-bit timers can be set by the processor to provide either timekeeping or watchdog functions. Processor interrupts are generated within the PIU under two conditions. One condition is a timer time-out; the other is a write operation to a specially designated PIU register by either the local processor or C_Bus master.

The reset and clock signals shown at the top of Figure 1.1 are produced by the Fault-Tolerant Clock Unit (FTCU) not shown here. The *pmm_reset* signal is sent only to the PIU to allow it greater control over the local processors. For example, the PIU uses this signal to enter its initialization mode, during which it activates the processor reset signals. All of the PIU input signals produced by the FTCU are synchronized with those in the PIUs in redundant PMMs of a fault-tolerant FTEP core.

The structure of the PIU itself is shown in Figure 1.2. The Processor Port (P_Port), C_Bus Port (C_Port), and Memory Port (M_Port) implement the communication protocols for the L_Bus, C_Bus, and M_Bus, respectively. The M_Port also implements (7,4) Hamming encoding and decoding on writes and reads, respectively, to the local memory, and the C_Port implements single-bit parity encoding and decoding for C_Bus transfers.

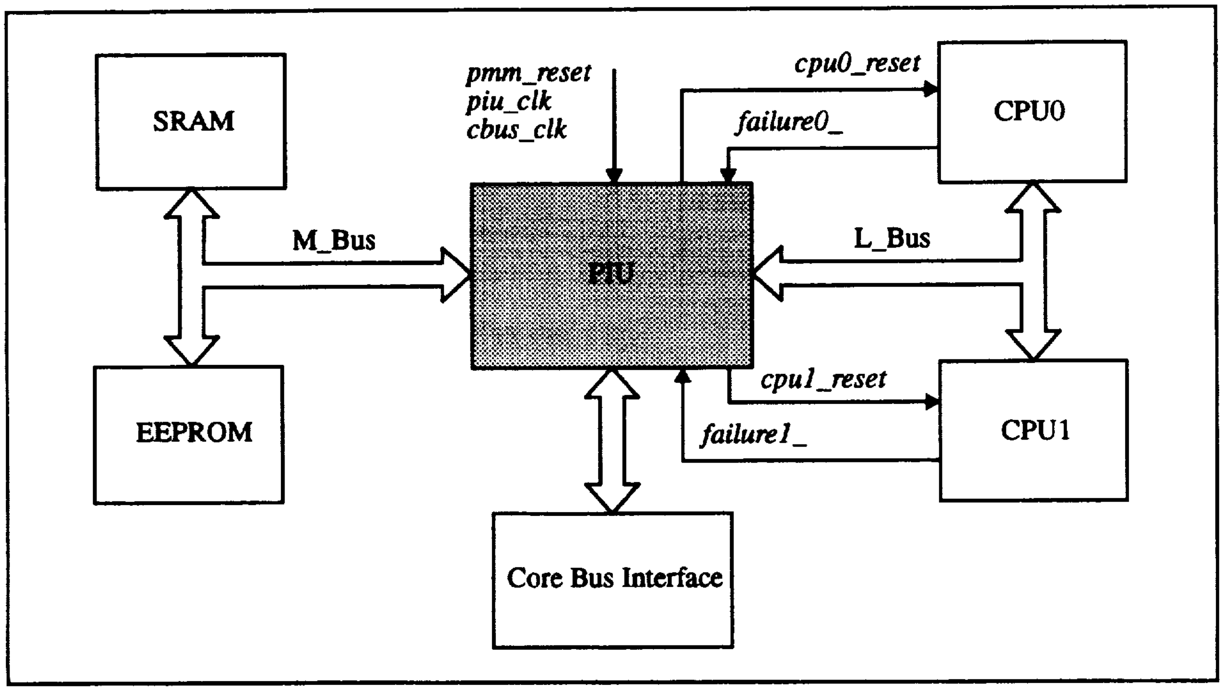


Figure 1.1: Block Diagram of the Processor-Memory Module (PMM).

The Register Port (R_Port) is the fourth, and final, port residing on the PIU's Internal Bus (I_Bus). It contains a state machine, counters, and various command and status registers used by the local processor to implement timers and interrupts.

The Start-up Controller (SU_Cont) implements the PMM initialization sequence. After it has concluded initialization, control is turned over to the other ports with the SU_Cont continuing operation in a background mode. The SU_Cont is not physically located on the I_Bus, however, for convenience, we will sometimes refer to it as one of the five PIU *ports*.

Behaviorally, the PIU functionality can be divided into four categories: (1) PMM initialization, (2) local-processor memory accesses, (3) C_Bus memory accesses, and (4) timers and interrupts.

1.1.1 PMM Initialization

The PIU controls the PMM initialization sequence. After receiving a synchronous *pmm_reset* signal from the FTCU, the PIU initiates the testing of the two local processors (or CPUs). Based on the test results, the PIU selects one of the CPUs to be active for the upcoming mission, while at the same time isolating the

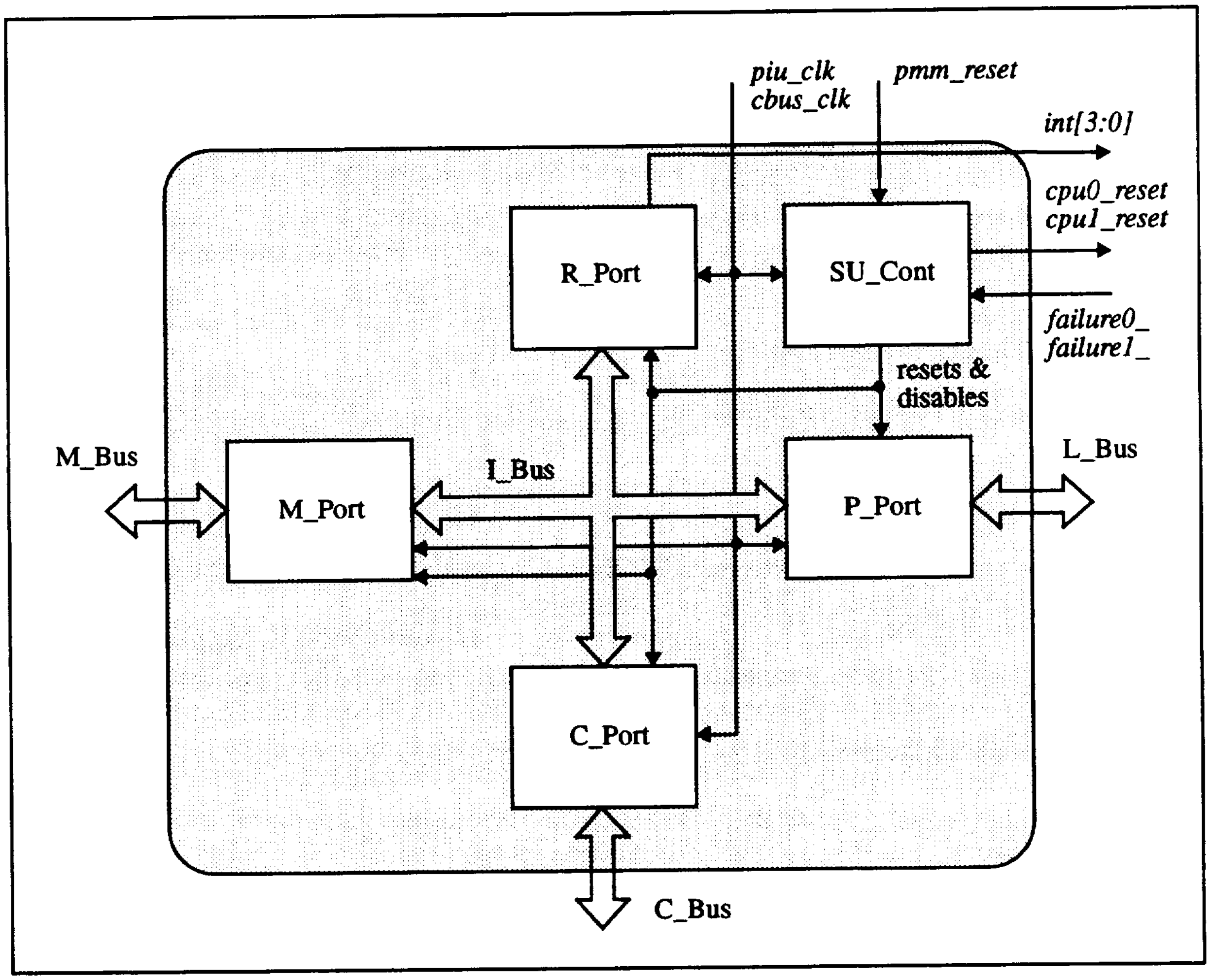


Figure 1.2: Major Blocks of the Processor Interface Unit (PIU).

other CPU. During the initialization, the PIU also maintains the inter-PMM synchronization that is initially established by the FTCUs.

The PIU initiates CPU self-test via the CPU reset signals that it controls. To begin the initialization sequence, the PIU resets CPU0, which then goes through a two-phase (Intel 80960) testing process of its own. In the first phase the CPU executes a 47,000-cycle self-test procedure; in the second phase the CPU reads the first eight words of local memory (via the PIU) and performs a check-sum test. If either of these tests fail, then the CPU's *failure0_* pin remains asserted, otherwise it is deasserted.

After the CPU self-test is completed, the CPU executes a software-based test using a program and the prior-mission fault status stored in local memory. At preselected points in this program the CPU updates PIU registers in a prespecified manner. At the end of this program, the PIU compares the modified PIU register values against their expected values. This acceptance test is the final major test of CPU functionality during initialization.

At the same time that CPU0 is being tested, the PIU isolates CPU1 by asserting its *cpu1_reset* input. Once the testing of CPU0 is completed, the roles are reversed. After both CPUs have been tested, the PIU selects one to be active for the upcoming mission. The selection algorithm makes use of the CPU failure signal outputs and the acceptance-test results: if CPU0 is ok then it is selected, otherwise if CPU1 is ok then it is selected, otherwise neither one is selected. Once the selection is made, the selected CPU is reset again and begins normal operation. The PIU isolates the other CPU by keeping its reset active.

An important PIU requirement is to maintain clock-level synchronization between redundant PMMs, yet accommodate possible nondeterminism within the PMM initialization sequences. Before the PMM initialization begins, the redundant PMM clocks are synchronized by the FTCUs, and *pmm_reset* signals are delivered to the PIUs synchronously across all PMMs. Synchronization is maintained by establishing maximum time durations for each phase of the initialization and having each PMM use the entire duration. The PIUs enforce these phase boundaries and thus guarantee that each PMM leaves its initialization on precisely the same clock cycle.

1.1.2 CPU Accesses to Memory

The PIU controls CPU reads and writes to the local memory, the internal PIU registers, and global memory.

1.1.2.1 To Local Memory

The PIU implements error-correction code (ECC) encoding and decoding and supports atomic memory operations, byte accesses, and 2-, 3-, and 4-word block transfers.

On writes to the local memory, the PIU encodes the 32-bit data words using a single-error-correction (7,4) Hamming code. The 56-bit encoded words are stored such that each 7-bit word (there are eight of these) is spread among the seven 256Kx8-bit memory chips. On reads, the decoding process implemented within the PIU masks all faults affecting one of the seven bits of each code word. Entire memory-chip failures are thus handled.

Atomic memory accesses, the atomic add and atomic modify instructions of the Intel 80960 instruction set, are supported by the PIU. During these operations the PIU prevents the C_Bus from gaining access to the local memory. The PIU uses the lock signal provided by the CPU during these operations.

Byte accesses to the local memory are supported by the PIU. Reads are implemented in a straightforward way. Writes are implemented using a read-modify-write operation that reencodes the entire 32-bit data word.

Byte accesses of up to four words are also supported to implement cache refilling within the CPU.

1.1.2.2 To Internal Register File

The PIU supports atomic accesses and 2-, 3-, and 4-word block transfers to and from its internal registers within the R_Port. Byte accesses are not supported, nor is the data encoded before being stored. Table 1.1 shows the R_Port register definitions.

The Interrupt Control Register (ICR) supports memory-mapped interrupts to the local processor. The register is divided into four fields. The first two contain the interrupt settings and mask bits for *int0_*, in bits 0 through 7 and 8 through 15, respectively. A logic-1 in both a set location and the associated mask location signifies an active interrupt, which if enabled (external to the R_Port) will generate an active *int0_* signal to the processor. Bits 16 through 31 are used in a corresponding way for *int3_*.

The ICR contents are updated in two different ways. A write to register address 0 implements a logical-AND operation on the new value and the old register contents, while a write to address 1 implements a logical-OR operation. These two operations implement the resetting and setting of register bits, respectively. A read to either of these addresses returns the current register value.

The General Control Register (GCR) and Communication Control Register (CCR) provide control bits to the internal PIU and the C_Bus, respectively. The GCR bits include the start-up software counter enable (used for the acceptance test discussed earlier), R_Port counter configuration control bits, and parity-error-latch reset bits. The CCR contains the message header for the next C_Bus transaction. Either of these registers can be written to or read from by the local processor.

The Status Register (SR) holds status information produced internally to the PIU. This includes start-up error-detection status, local-memory and C_Bus error-detection status, start-up controller state, and the last C_Bus slave-status report. This register is read-only.

Register addresses 8 through 11 are used to load new counter values to the 32-bit counters 0 through 3, respectively. These load values can be read by the local processor using the same addresses. Register addresses 12 through 15 are read-only locations containing the current value of the four counters.

The four counters are combined to form two 64-bit counters which can be configured in a variety of ways via control bits in the GCR. The choices include enabled vs. disabled counting, enabled vs. disabled interrupting on overflow, and reloading vs. count-continuation on overflow. Counters 0 and 1 together support timer interrupts using the *int1* interrupt line; counters 2 and 3 use *int2*.

Table 1.1: R_Port Register Definitions.

Register Address	Contents
0	Interrupt Control Register (ICR) reset
1	ICR set
2	General Control Register (GCR)
3	Communication Control Register (CCR)
4	Status Register (SR)
8	Counter 0 in
9	Counter 1 in
10	Counter 2 in

Table 1.1: R_Port Register Definitions.

Register Address	Contents
11	Counter 3 in
12	Counter 0 out
13	Counter 1 out
14	Counter 2 out
15	Counter 3 out

1.1.2.3 To the C_Bus

The upper 2 GB (gigabytes) of the CPU address space is reserved for external memory and input/output (I/O). The PIU routes CPU memory accesses at these addresses to the C_Bus. It implements the C_Bus protocol, parity encoding and decoding of data, and support for atomic memory operations, byte transfers, and 2-, 3-, and 4-word block transfers.

The PIU implements the C_Bus communication protocol. This includes all arbitration actions and necessary handshaking.

On writes to the C_Bus the PIU encodes each byte of data using a single-error-detection parity code. Data arriving over the C_Bus is likewise decoded.

Atomic memory operations are supported by the PIU. Once the PIU acquires the C_Bus it doesn't relinquish it until the atomic operation is completed. The PIU again makes use of the CPU lock signal to know when to do this.

Byte transfers and 2-, 3-, and 4-word transfers are handled in a straightforward manner.

1.1.3 C_Bus Accesses to Memory

The PIU controls C_Bus reads and writes to local memory and the PIU register file. All of the support features described earlier for the CPU-initiated transfers are supported here as well. The C_Bus (i.e., the processing unit of an external block) has priority over the CPU for local memory accesses. The PIU holds off the local CPU using the CPU *hold_* input signal. The PIU supports block transfers as large as 64 words over the C_Bus.

1.1.4 Timers and Interrupts

As explained above, the PIU contains two 64-bit counters and an interrupt control register. The counters can be used to implement timed interrupts as well as a real-time clock. The timed interrupts can be programmed to provide either a single-shot interrupt or repeated, periodic interrupts.

The interrupt register is a memory-mapped register used to implement 16 possible interrupts. These interrupts can be initiated by either the active local processor or an external C_Bus master.

1.2 Specification Overview

Figure 1.3 shows the specification hierarchy developed for the PIU. In constructing this hierarchy much emphasis was placed on maintaining compatibility with existing formal specification methods, particularly the generic interpreter theory described in Section 2. The resulting hierarchy reflects this emphasis, particularly in the lower levels where many of the techniques described in [Win90] are used.

Consistent with established hierarchical specification methods, the levels in the hierarchy of Figure 1.3 are abstractions of the levels below them. Four types of abstraction are used here. Temporal abstraction relates time at a particular level to the time at lower levels; each unit of time at the higher level corresponds to multiple time units at the lower level. Data abstraction relates the states of two levels, with the higher level state being a function (typically a subset) of the state at the lower level. In behavioral abstraction, a structural description at the lower level, defined using the physical interconnection of components or subsystems, is replaced by a purely behavioral description at the higher level. Structural abstraction (or composition) combines subsystems defined at one level to form a higher level.

At the bottom of the PIU specification hierarchy is the gate-level description. This is a structural description derived from the lowest-level detailed design developed by the PIU design team. The chip layout is obtained directly from this level using silicon compilation techniques that are not within the scope of the specification and subsequent verification tasks. Components at the gate level include individual logic gates, latches, counters, and finite-state machines. This level is comparable to the electronic block model (EBM) level of [Win90].

The phase-level behavioral description for each of the five PIU ports is a behavioral abstraction of each corresponding gate level. This level is comparable to the phase level used in [Win90]. The specification at this level consists of an instruction set containing two instructions, one for phase A and one for phase B, defining the state transition and outputs generated during each phase.

The clock-level behavioral description for the PIU ports uses a time interval of an entire clock period rather than a single phase (temporal abstraction), and the state is a subset of the phase-level state (data abstraction). Only a single instruction is defined for each port, specifying the state change and outputs of the port occurring during its execution. This level is comparable to the microinstruction level of [Win90] and elsewhere except that only a subset of the chip design (i.e., a port) is described here rather than the entire chip.

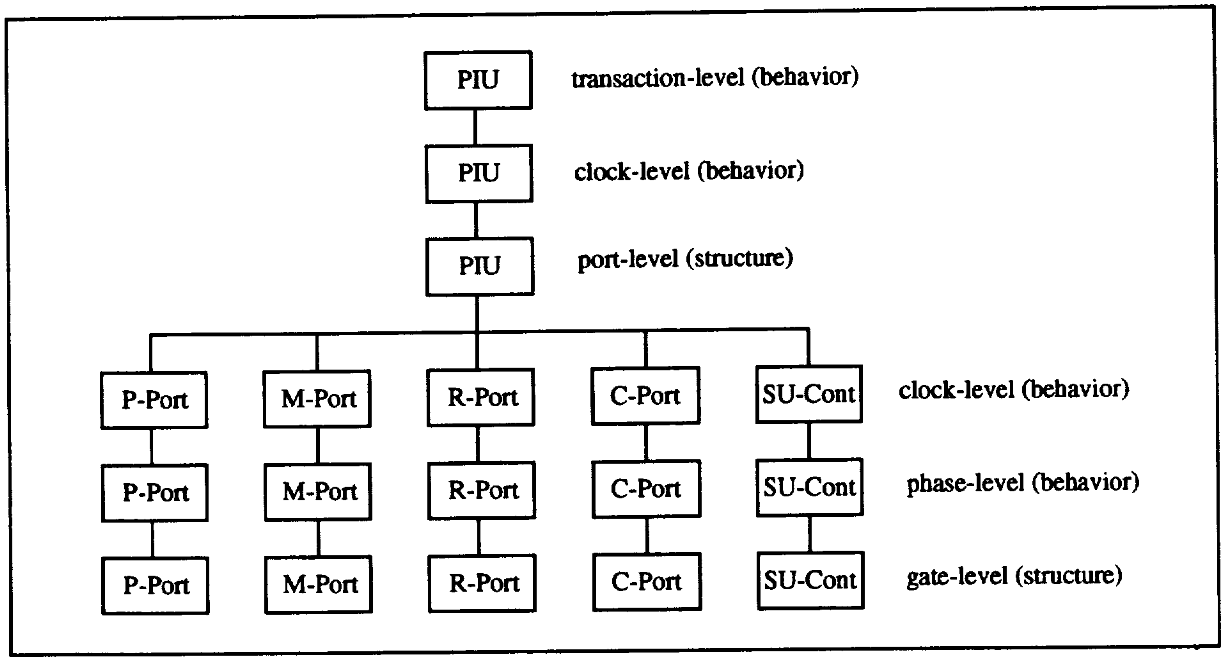


Figure 1.3: PIU Specification Hierarchy.

The port-level structure is a structural composition of the five individual clock-level port specifications. The port composition is based on the established method of forming a logical conjunction of the individual port descriptions.

The clock-level behavioral description for the PIU is a behavioral abstraction of the structural description at the PIU port level, providing a clock-level description for the entire chip. This level is comparable to the microinstruction level referred to above, an important difference being in the approach to instruction decoding: here no decoding is used, resulting in a single instruction compared to the many microinstructions in [Win90], for example.

The transaction-style behavioral description is the topmost level in the PIU hierarchy providing a concise and easy-to-understand definition of PIU behavior. Whereas the lower five levels of the hierarchy represent the PIU *design* and were developed bottom-up, the transaction level specifies the PIU *requirements*. In this role as human interface the transaction level must address modeling problems not faced at the lower levels.

Three important problems unique to the transaction level are: (1) independently-initiated concurrent behavior, (2) multiple sequential outputs, and (3) shared state. Because of these, hardware modeling approaches used within the HOL community to date are inadequate for transaction-level modeling. Section 4 describes these problems in more detail and explains our progress in developing a transaction-level model suitable for the PIU.

2 Generic Interpreter Theory

This section describes the generic interpreter theory used to model portions of the PIU. The work described in this section grew out of efforts to model microprocessors and thus the model discusses microprocessor specification and verification heavily. We have discovered that the model is useful for describing other hardware devices as well, and, in particular, we have found it to be well-suited for specifying the PIU design. The generic interpreter theory is described more fully in [Win90].

2.1 Introduction.

The formal specification and verification of microprocessors has received much attention. Indeed, several verified microprocessors have been presented in the literature. This section presents an abstract model that describes a large class of hardware devices, including microprocessors and other devices with a single major control point. The model is called a generic interpreter and the theory contains important theorems about it.

We have formalized the interpreter model in the HOL theorem proving system [Gor88,Gog88]. The formal model can be instantiated inside the system and serves as a framework for writing device specifications and verifying them. This framework clearly states what definitions must be made to specify the device and which lemmas must be established to complete the verification. After the user has defined the components of the hardware device model and proven the necessary lemmas about them, individual theorems from the abstract theory can be instantiated to provide concrete theorems about the actual device being verified.

The model that we have defined has proven useful in specifying and verifying several microprocessors [Win90,Aro90]. The model is not, however, limited to microprocessors only. Recent work has shown that the model can be used in specifying other hardware devices as well [Win91]. Because the model was originally developed for microprocessor modeling, however, much of the terminology in the model (e.g., instruction set) is influenced by microprocessor terminology. We have kept it even though more general terminology might be better in some cases.

The model we have defined differs from other formal descriptions of state machines (such as Loewenstein's model in [Low89]) by including the data and temporal abstractions that are important in specifying and verifying microprocessors in the formalization.

2.2 Formal Microprocessor Modeling.

There have been numerous efforts to formally model microprocessors. At the time this project was begun the best known of these included Jeff Joyce's Tamarack microprocessor [Joy89], Warren Hunt's FM8501 and FM9001 microprocessors [Hun87, Hun92], and Avra Cohn's verification of VIPER [Coh88]. Tamarack is a simple microprocessor with only 8 instructions. FM8501 is larger (roughly the size of a PDP-11), but has not been implemented; FM9001 is a 32–bit version that is being verified and implemented. VIPER is the first microprocessor intended for commercial use where formal verification was used. However, the verification has not been completed because of the large case explosion that occurred and the size of the proofs in each of the cases. Recent work on hierarchical specification [Win88], coupled with the work presented here, has overcome this problem; microprocessors significantly more complicated than VIPER are now within the realm of formal treatment.

2.2.1 Microprocessor Specification.

The specifications for the microprocessors mentioned above appear very different on the surface; in fact, the specifications of FM8501 and FM9001 are even in a different language. On closer inspection, however, each uses the same implicit behavioral model. In general, the model uses a state transition system to describe the microprocessor. A microprocessor specification has four important parts:

1. A representation of the state, S.
2. A set of state transition functions, J, denoting the behavior of the individual instructions of the microprocessor. Each of these functions takes the state defined in step (1) as an argument and returns the state updated in some meaningful way.
3. A selection function, N, that selects a function from the set J according to the current state.
4. A predicate, I, relating the state at time $t+1$ to the state at time t by means of J and N.

In some cases, the individual state transition functions, J, and the selection function, N, are combined to form one large state transition function. Also, a functional specification would use a function for part (4) instead of a predicate. The general form, however, is the same.

2.2.2 Microprocessor Verification.

Just as most microprocessor specifications are similar, so too are their verifications. After the microprocessor has been specified, we can verify that a machine description, M, implements the specification, I, for some state, s, by showing:

$$\forall s \in S \bullet (M(s) \Rightarrow I(s))$$

That is, we show that I has the same effect on the state, s, as M does. This theorem is typically shown by case analysis on the instructions in J by establishing the following lemma:

$$\forall (j \in J) \bullet M(s) \Rightarrow (\forall t \bullet C(j, s, t) \Rightarrow (s(t+n_j) = j(s(t))))$$

where C is a predicate expressing the conditions for instruction j's selection, $s(t)$ is the state at time t, and n_j is the number of cycles that it takes to execute j. This lemma says that if an instruction j is selected, then applying j to the current state yields the state that results by letting the implementing interpreter M run for n_j cycles. We call this lemma the instruction correctness lemma.

2.3 A Formal Model of Interpreters.

An interpreter is a computing structure with one control point. One of the many available instructions is chosen at this control point based on the current state and inputs. The state is then processed by this instruction and the cycle begins again.

In general, a microprocessor specification can consist of many abstraction levels. Every level except the bottom specification (which is the structural specification) can be modeled as an interpreter. A hierarchical approach to specification and verification has been shown to significantly reduce the amount of effort required to complete the verification of a microprocessor [Win88].

Figure 2.1 shows a generalized hierarchy of interpreters. Note that each communicates with the state and environment, although most interpreters see only an abstraction of the state. An interpreter sends instructions to the interpreter below it and communicates (mostly timing) information to the interpreter above it.

2.3.1 Abstract Theories.

A theory is a set of types, definitions, constants, axioms and parent theories. Logics are extended by defining new theories. An abstract theory is parameterized so that some of the types and constants defined

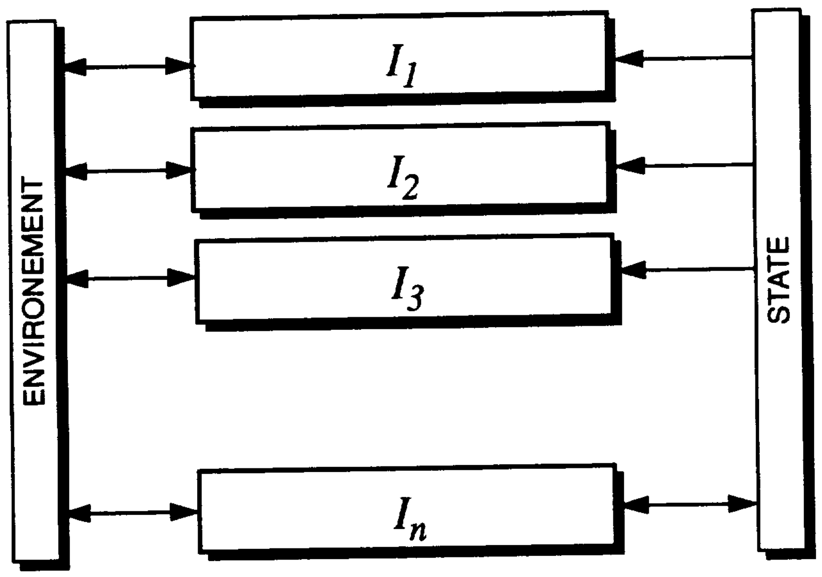


Figure 2.1: A Hierarchy of Interpreters.

in the theory are undefined inside the theory except for their syntax and a loose algebraic specification of their semantics. Group theory is an example of an abstract theory. The multiplication operator is undefined except for its syntax (a binary operator on type "*:group*") and a loose semantics given by the axioms of group theory.

Abstract theories are useful because they provide proofs about abstract structures that can be used to reason about specific instances of the structure. In groups, for example, after showing that addition over the integers satisfies the axioms of group theory, we can use the theorems from group theory to reason about addition on the integers.

An abstract theory consists of three parts:

1. An *abstract representation* of the uninterpreted constants and types in the theory. The *abstract representation* contains a set of abstract operations and a set of abstract objects. (These are sometimes called uninterpreted constants and uninterpreted types.)
2. A set of *theory obligations* defining relationships between members of the abstract representation. Inside the theory, the obligations represent axiomatic knowledge concerning the abstract representation. Outside the theory, the obligations represent the criteria that a concrete representation must meet if it is to be used to instantiate the abstract theory.
3. A collection of *abstract theorems*. The theorems are generally based on the theory obligations and can stand alone only after the theory obligations have been met.

To instantiate an abstract theory, the concrete representation must meet the syntactic requirements of the abstract representation as well as the semantic requirements of the theory obligations. If the syntactic and semantic requirements are met, then the instantiation provides a collection of concrete theorems about the new representation.

There are several specification and verification systems that support abstract theories. Some, such as OBJ [Gog88] and EHDM [SRI88], offer explicit support. HOL, the verification environment used for the research reported here, does not explicitly support abstract theories; however, HOL's metalanguage, ML, combined with higher-order logic, provides a framework for implementing abstract theories [Win90a] in a manner that does not degrade the trustworthiness of the theorem prover.

2.3.2 Temporal Abstraction

Before we can discuss the formal model, we must describe the temporal abstraction that it uses. The development follows that of [Joy89,Mel88,Her88].

In general, different levels in the interpreter hierarchy will have different views of time. We use temporal abstraction to produce a function that maps time at one level to time at another. Figure 2.2 shows a temporal abstraction function *F*. The circles represent clock ticks. The number of clock ticks required at the implementing level to produce one clock tick at the implemented level is irregular.

The predicate, *G*, is true whenever there is a valid abstraction from the lower level to the upper level. We can define a generic temporal abstraction function in terms of *G*. In a microprocessor specification, *G* is usually a predicate indicating when the lower level interpreter is at the beginning of its cycle—a condition that is easy to test.

We will use a function *Temp_Abs* as our temporal abstraction function. The function is defined recursively so that *(Temp_Abs g 0)* is the first time that the predicate *g* is true and *(Temp_Abs g (n+1))* is the next time after time *n* when *g* is true. We will not develop the details of the temporal abstraction function here, but refer the interested reader to the references given above and [Win90].

2.3.3 The Abstract Representation

We specify the abstract representation by defining a list of abstract objects and operations. Table 2.1 shows the operations and their types. We must emphasize that the representation *is* abstract and, therefore, the objects and operations have no definitions. The descriptions that follow are what we *intend* for the representation to mean. The representation is purely syntactic, however.

The following abstract types are used in the representation.

- *:*state* represents the state.
- *:*env* represents the environment.

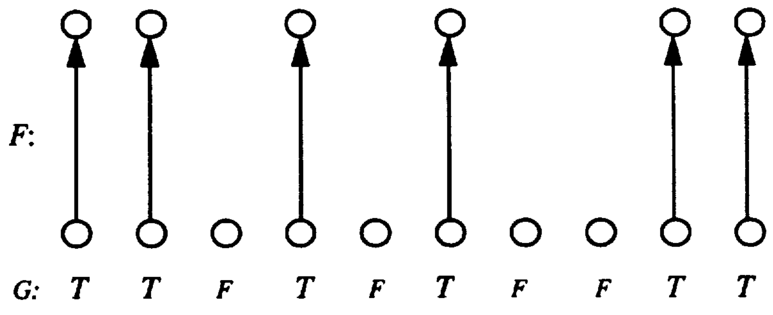


Figure 2.2: The Temporal Abstraction Functions *F* and *G*.

Table 2.1: The abstract functions and their types for the generic interpreter model.

Operation	Type
instructions	*":*key->(*state->*env->*state)"*
select	*":*state->*env->*key"*
output	*":*key->(*state->*env->*out)"*
substate	*":*state'->*state"*
subenv	*":*env'->*env"*
subout	*":*out'->*out*
Impl	*":(time'->*state')->(time'->*env')->bool"*
count	*":*state'->*env'->*key'"*
start	*":*key'"*

- *:*out* represents the outputs.
- *:*key* is type containing all of the keys. Keys are used to select instructions. For example, the opcodes form the keys in the top–level specification of a microprocessor.

We add primes to the types to indicate that they represent state, time, etc. at the implementing rather than the implemented level of the hierarchy.

The abstract representation can be broken into two parts. The first contains those operations concerned with the interpreter.

- *instructions* is the instruction set. The set is represented by a function from a key to a state transition function.
- *select* picks a key based on the present state and environment.
- *output* is a set of output functions. The set is represented by a function from a key to a function that produces output for a given state and environment.
- *substate* is the state abstraction function for the interpreter. The *substate* function is used to hide the visible state in the interpreter.
- *subenv* is the environment abstraction.
- *subout* is the output abstraction.

Because we want to prove correctness results about the interpreter, we must have an implementation. The second part of the abstract representation contains three functions that provide the necessary abstract definitions for the implementation.

- *Impl* is the abstract implementation. We could have chosen to make this function more concrete, but doing so would have required that every implementation have some pre-chosen structure. Thus, we say nothing about it except to define its type.
- *count* is analogous to *select* except it operates at the implementing level.
- *start* denotes the beginning of the implementation clock cycle.

We will ensure that *count* periodically reaches *start* as part of the synchronization process.

2.3.4 The Theory Obligations

Proving that the implementation implies the interpreter definition is typically done by case analysis on the instructions; we show that when the conditions for an instruction's selection are right, the instruction is implied by the implementation. We call this the instruction correctness lemma.

The predicate *INSTRUCTION_CORRECT* expresses the conditions that we require in the instruction correctness lemma:[1]

```
|-def INSTRUCTION_CORRECT gi s' e' inst  =
        (Impl gi s' e') ==>
        (!t:time'.
        let st = (substate gi (s't)) in
        let et = (subenv gi (e't)) in
        let ft = (count gi (s't) (e't) = (start gi)) in
        let k = (select gi (st) (et)) in
        ((inst = (instructions gi k)) ∧ (ft) ==>
        ?c. Next f(t,t+c) ∧ (inst (st) (et) = (s(t+c)))))
```

INSTRUCTION_CORRECT operates on a single instruction *inst*. The implementation implies that for every time, *t*, if *inst* is selected and the implementation's counter is at the beginning, then there is a time *c* cycles in the future such that applying the instruction to the current state yields the same state change that the implementation does in *c* cycles.

INSTRUCTION_CORRECT is a good example of the kind of information that is captured in the generic model. Previous microprocessor verifications created this lemma, or one similar to it, in a largely *ad hoc* manner.

Because our model has outputs as well as inputs (the environment), we must also assume something about the output in order to establish correctness. The predicate *OUTPUT_CORRECT* expresses the conditions that we require in the output correctness lemma:

```
|-def OUTPUT_CORRECT gi s' e' p' k  =
        (Impl gi s' e' p') ==>
        (!t:time'.
              let st = (substate gi (s't)) in
              let et = (subenv gi (e't)) in
              let pt = (subout gi (p't)) in
              let ft = (count gi (s't) (e't) = (start gi)) in
              ((count gi (s't) (e't) = (start gi)) ∧
               (select gi (st (et) = k) ==>
              (pt = (output gi k) (st) (et))))
```

1. The HOL code in this report is shown using the HOL convention of representing universal quantification, existential quantification, implication, conjunction, disjunction, and negation by the symbols !, ?, ==>, ∧, V., and ~, respectively. The form "e1 => e2 | e3" represents "if e1 then e2 else e3."

Using *INSTRUCTION_CORRECT* and *OUTPUT_CORRECT* we can define the theory obligations in our model. The theory obligations are given as a predicate on an abstract representation *gi*:

```
|-defGI gi =
        (!s'e'p'k. INSTRUCTION_CORRECT gi s' e' p' k) ∧
        (!s'e'p'k. OUTPUT_CORRECT gi s' e' p' k)
```

The predicate says that every instruction in the instruction set satisfies the predicate *INSTRUCTION_CORRECT* and every output function satisfies the conditions set forth in *OUTPUT_CORRECT*.

2.3.5 Abstract Theorems

Using the abstract representation and the theory obligations, many useful theorem pertaining to interpreters can be established on the generic structure.

2.3.5.1 Defining the Interpreter

One of the important parts of the collection of abstract theorems is the definition of a generic interpreter. The definition is based on functions from the abstract representation.

```
|-defINTERP gi s e p  =
        !t:time.
        let k = (select gi (s t) (e t)) in
        (s(t+1) = (instructions gi k) (s t) (e t)) ∧
        (p t = (output gi k (s t) (e t))
```

The specification of an interpreter is a predicate relating the contents of the state stream at time *t*+1 to the contents of the state stream at time *t*. The relationship is defined using the functions from the abstract representation. The definition also uses the currently selected output function to denote the current output.

2.3.5.2 Induction on Interpreters

The definition of the interpreter sets up a relation between the state at *t* and *t*+1. Sometimes it is useful to have a more explicit statement regarding induction. The following theorem, which follows from the definition of the interpreter given in Section 2.3.5.1, defines induction on an interpreter:

```
|-!Q. INTERP gi s e p  ==>
    (Q (s0) ∧
        !t. let inst = (instructions gi (select gi (s t) (e t)) in
        Q (s t) ==> Q (inst (s t) (e t))))==>
    !t. Q (s t)
```

The theorem states that for any arbitrary predicate on states, *Q*, if *Q* is true of the state at time 0, and when *Q* is true of the state at time *t*, it follows that it's also true of the state returned by the current instruction, then *Q* is true of every state.

We note that even though this theorem looks fairly simple, and indeed is quite easy to show in the generic theory, the theorem will eventually be instantiated with the entire denotational description of the semantics of a particular instruction set and will be quite involved. The same admonition holds for each of the theorems and definitions presented in this section.

2.3.5.3 The Implementation is Live

Using the theory obligations, we can prove that the implementation is live. By *live* we mean that if the implementation starts at the beginning of its cycle, then there is a time in the future when the implementation will be at the beginning of its cycle again. That is, we show that the device will not go into an infinite loop.

```
|-  Impl gi s' e'  ==>
    (!t. (count gi (s't) (e't) = start gi   ==>
    (?n. Next (\t. count gi (s't) (e't) = start gi) (t, t+n)))
```

Next P (t1, t2) says that *t2* is the next time after *t1* when *P* is true.

2.3.5.4 The Correctness Statement

The correctness result can be proven from the definition of the interpreter and the theory obligations:

```
|-  let st = (substate gi (s't)) and
        et = (subenv gi (e't)) and
        pt = (subout gi (p't)) and
        ft = (count gi (s't) (e't) = (start gi)) in
    let abs = (Temp_ABSf) in
    (Impl gi s' e' p') ∧
    (?t. ft) ==>
    (INTERP gi) (s o abs) (e o abs) (p o abs)
```

In the correctness statement, *s'*, *e'*, and *p'* are the state, environment, and output streams in the implementation. The terms *(s o abs)*, *(e o abs)*, and *(e o abs)* are the state, environment, and output streams for the interpreter defined in the model. They are data and temporal abstractions of *s'*, *e'*, and *p'*. The correctness statement says that if the implementation is valid on its state, environment, and output streams and there is a time when the implementing clock is at the beginning of its cycle, then the interpreter is valid on its state and environment streams.

2.3.5.5 Composing Interpreters Hierarchically

In [Win88], we show that hierarchical decomposition makes the verification of large microprocessors practical. To support this decomposition, the generic interpreter model contains a theorem about composing generic interpreters hierarchically.

```
|-(INTERP gi 1 = Impl gi 2) ∧
    (select gi 1 = count gi 2)  ==>
    !(s":time->*state")(e":time->*env")(p":time->*out").
    let s't = (substate gi 1 (s"t)) and
       e't =(subenv gi 1 (e"t)) and
       p't = (subout gi 1 (p"t)) and
       ft = (count gi 1 (s"t) (e"t) = start gi 1) in
    let st = (substate gi 2 (s't)) and
       et = (subenv gi 2 (e't)) and
       pt = (subout gi 2 (p't)) and
       gt = (select gi 1 (s't) (e't) = start gi 2) in
    let abs1 = (Temp_ABS f) in
    let abs2 = abs1 o (Temp_ABS (g o abs1)) in
    (Impl gi 1 s" e" p") ∧
    (?t. ft) ==>
    (?t. (g o abs1) t) ==>
    INTERP gi 2 (s o abs2) (e o abs2) (p o abs2)
```

This theorem states that if *gi 1* and *gi 2* are generic interpreters and they are connected such that the interpreter definition of *gi 1* is the implementation of *gi 2* then the implementation of *gi 1* implies the interpreter definition of *gi 2*.

This important theorem captures the temporal and data abstraction required to compose two interpreters. This theorem is a good example of the utility of abstract theories in hardware verification. This theorem is tedious to prove and were it not contained in the abstract theory, it would have to be proven numerous times in the course of a single microprocessor verification.

2.4 Parallel Composition

Our eventual goal is to use the work that is described in Section 4 to show how a set of interpreters can be composed with each other in parallel. This goal is significantly different from the theorem described in Section 2.3.5.5. In hierarchical composition, the implementation of one interpreter model is the interpreter from the other. In parallel composition, the two interpreters share a behavioral specification (i.e., interpreter definition), and the implementation is two or more interpreters linked together. The interpreters can be linked by shared state, common input, common output, and connections between the interpreters' inputs and outputs.

Undoubtedly, as our theory of composition matures, the generic interpreter theory will change. The advantage of generic theories is that these changes can be made more easily in the generic theory than they can in a specific definition of a VLSI device.

2.5 Conclusion

This section has described the generic interpreter model. The theory isolates the temporal and data abstractions of the proof inside the abstract theory. The theory also contains several important theorems about the abstract representation. These theorems are true of every instantiation of the abstract representation that meets the theory obligations.

The theory has many important benefits:

- The generic model structures the proof by stating explicitly which definitions must be made (one for each of the members of the abstract representation) and which lemmas need to be proven about these definitions (namely, the theory obligation). This is a substantial improvement over previous microprocessor verifications where these decisions were made on an *ad hoc* basis.
- The generic model insulates users of the model from complex proofs about the data and temporal abstractions. These proofs are done once and then made available to the user by instantiation.
- The use of a generic interpreter model for specifying and verifying microprocessors provides a methodological approach. Making specification and verification methodological is an important step in turning what has been primarily a research activity into an engineering activity.

3 Design Specification

This section describes the lower five levels of the PIU specification hierarchy (Figure 1.3), which constitute the design specification. The discussion proceeds bottom-up, beginning with the gate-level specification of individual ports and finishing up with the clock-level specification for the entire PIU.

The gate-level specification, described in Section 3.1, corresponds to the lowest-level design implemented by the PIU design team. Below this level a silicon compiler provides the translation to the mask layout used for chip fabrication. The specification effort described in this report is not concerned with this translation, which currently falls within the domain of the tool vendor — Mentor Graphics Corporation.

A set of detailed-design schematics was produced by the design team as part of the design process. Unfortunately they are not suitable for this report because, in printed form, many are too small to be understood. Because of this we created our own set of schematics, included in Section 3.1, to accompany the HOL specifications located within the appendices. These schematics are provided as aids to understanding only, since, due to time constraints in developing them, they are not complete nor are they fully accurate.

Sections 3.2 through 3.5 describe, in order, the phase-level specifications for the five ports, the clock-level specifications for the five ports, the port-level structural specification, and the clock-level specification for the entire PIU.

3.1 Gate-Level Structure

The gate-level specifications for the five PIU ports use the structural definition style described in [Gor86] and in use throughout the HOL community. Within each port, each component, or block, has its behavior specified in the form of a predicate; in essence, the block behavior is defined to be the relationship between inputs, outputs, and internal states that results in the predicate's being true. The behavior of the composition of these blocks is defined as the logical conjunction of the individual block predicates. Existentially quantified variables are used for the block interconnections internal to the port-level composition.

The gate-level specification for the PIU is much too unwieldy for a detailed coverage in these pages. This section therefore provides only a high-level explanation of the PIU's operation and the HOL models that represent it. References will be made to the appropriate sections of the appendices for the full details.

We begin in Section 3.1.1 with a description of the components used in the PIU design. Fortunately, the design uses only a small subset of the component types available in the silicon compiler library, ranging in complexity from individual logic gates to medium-scale integration (MSI) datapath elements and finite-state machines. Section 3.1.2 explains how the components are combined to form the five PIU ports.

3.1.1 Component Descriptions

The HOL models for elementary logic gates follow closely the previous work in this area and we say little about this subject. Modeling sequential logic is more interesting however. Previous sequential models generally depict even the most elementary components as edge-sensitive devices — a flip-flop perspective. However, in the design tool used for the PIU, the elementary sequential component is not edge-sensitive, but rather the level-sensitive latch. Flip-flops are higher order components, consisting of two or more latches. As explained below, the level-sensitive components used in the PIU require a different modeling approach.

3.1.1.1 Combinational Logic

The PIU specification requires only a few inverters, AND and OR gates, and buffers from the component library. The specification style used for these components follows that of earlier work and is demonstrated in the AND-gate definition shown here. The theory *gates_def* in Appendix A contains the complete HOL source for these components.

```
|- AND3_SPEC a b c z  =  ∀ t:time . z t = (a t)∧(b t)∧(c t)
```

3.1.1.2 Latches

The HOL definitions for the latches used in the PIU design are contained in the theory *latches_def* in Appendix A. In this section we describe the modeling of a simple D latch as an explanation of the HOL models.

The following definition of a D latch demonstrates the specification style that we use for PIU latches. This specification states that the next state *q_state (t+1)* equals the input *d_in t* if the clock *clk_in t* is active, otherwise it equals its current value *q_state t*. The latch output *q_out t* equals the new state.

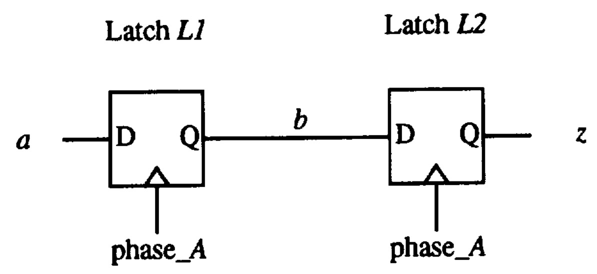
```
|- DLAT_SPEC d_in clk_in q_state q_out  =
     ∀ t:time .
     (q_state (t+1) = (clk_in t) => d_in t | q_state t) ∧
     (q_out t = q_state (t+1))
```

Latch behavior is being expressed here as a finite-state machine (FSM), using both a next-state function and an output function. Previous latch models in HOL, where the next-state function was also used for outputs, failed to faithfully represent true latch behavior. To demonstrate why this is true, Figure 3.1(a) shows an example circuit where two latches, in series, are clocked with the same phase of the system clock. To our knowledge, scenarios such as this have not been considered in prior verification work; however, we cannot dismiss them since they occur within the PIU design. Actually, such combinations might be expected in any standard-cell approach to chip design where designers work with predefined cells containing a multitude of latches in fixed locations. There are places in the PIU design, for example, where avoiding these combinations would actually require a more complicated design.

The circuit in Figure 3.1(a) would be incorrectly modeled if latch models containing only the next-state function of *DLAT_SPEC* were used. This is demonstrated in the HOL code segments of Figure 3.1(b), defining first the behavior of the implementation, including the next state of latch *L2* derived from this behavior, followed by a reasonable specification for its *required* behavior.

The behavior of the implementation *(IMP)* is a standard composition of individual latch behaviors. The key observation here is that the value of *z* at time *t+1* depends on signal values at time *t-1* (e.g., *a (t-1)*). However, as expressed in the model of required behavior *(REQ)*, in reality the circuit of Figure 3.1(a), when viewing the signal *z*, behaves no differently than a single A-clocked latch does (aside from propagation delay differences not expressed at this level). Therefore, the value of *z (t+1)* should be a function of signal values at time *t*, not *t-1*. Note that for the general case of *N* series, same-phase latches, we would have *z (t+1)* as a function of signals at time *(t-N-1)*; clearly this is not what we want. We note that the source of this problem is the level-sensitive nature of latches, which results in cascaded latches behaving very much like combinational logic; this is not true of edge-sensitive components such as flip-flops.

Revisiting fundamental FSM definitions suggests ways to solve this latch modeling problem. In automata theory texts, such as [Koh78], the next-state and present-output of an FSM are said to be functions of



(a) Block diagram.

$$IMP = (b(t+1) = phase\_A\ t \Rightarrow a\ t \mid b\ t) \wedge$$
$$(z(t+1) = phase\_A\ t \Rightarrow b\ t \mid z\ t)$$

⋮ (derived)

$$z(t+1) = phase\_A\ t \Rightarrow (phase\_A\ (t-1) \Rightarrow a\ (t-1) \mid b\ (t-1)) \mid z\ t$$

$$REQ = (b(t+1) = phase\_A\ t \Rightarrow a\ t \mid b\ t) \wedge$$
$$(z(t+1) = phase\_A\ t \Rightarrow a\ t \mid z\ t)$$

(b) Relationship between next z and current values, using standard latch model.

Figure 3.1: Two Series Latches Clocked by the Same Phase.

the present-state and present-inputs. Figure 3.2(a) is a pictoral representation of this where the *present* and *next* times are denoted by *t* and *t+1*, respectively. Figure 3.2(b) shows an alternative approach where the inputs and outputs use the time index of the next-state.

In models of synchronous systems such as FSMs, lower-level issues such as propagation delay are not represented. For a latch, whose time interval is a single clock phase, the present- and next-states correspond to the states at exactly the beginning and end of the phase, respectively. All present-inputs can similarly be assumed to arrive at either the phase beginning or end. Present-outputs are defined in terms of the present-state and -inputs, and are assumed to be transmitted with zero delay. Of course, in reality an input is a present-input only if it satisfies the setup and hold times of the latch with respect to the falling edge (the end) of the clock phase; state changes and output transmissions have propagation delay as well.

With this view of FSM behavior, it is clear that for a formal latch model to be composable in all clocking scenarios it must use the same time index for both its present-inputs and -outputs. This is necessary to permit signal propagation through series-connected, same-phase latches in zero time. In a latch model using only a single FSM next-state function, this function must play the role of the output function as well; thus, the time index of the current-output is *t+1*. If the standard interval representation of Figure 3.2(a) is used, then the input and output time indexes don't match, resulting in the problem explained above. Two obvious solu-

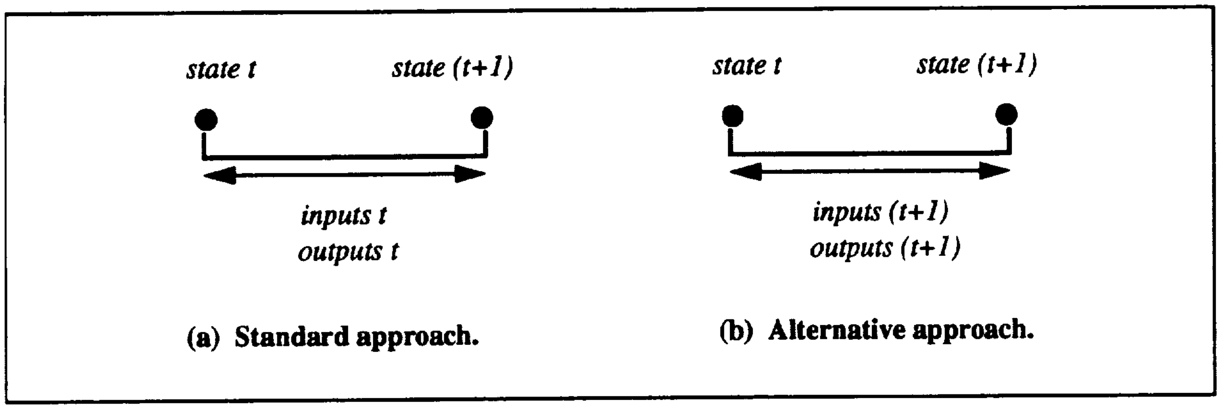


Figure 3.2: Interval Representations.

tions are to either use the alternative interval representation of Figure 3.2(b) or else use a second FSM function for the output, matching its time index to that of the input.

We mention the first solution, using the alternative interval representation, only to point it out as a candidate for future consideration. We currently prefer the second approach, expressed in the model DLAT_SPEC above, since it is consistent with the generic interpreter model described in Section 2.

3.1.1.3 Flip-Flops

HOL definitions for the flip-flops used in the PIU design are contained in the theory *ffs_def* of Appendix A. In this section we describe the modeling of a simple D flip-flop as an explanation of the HOL models.

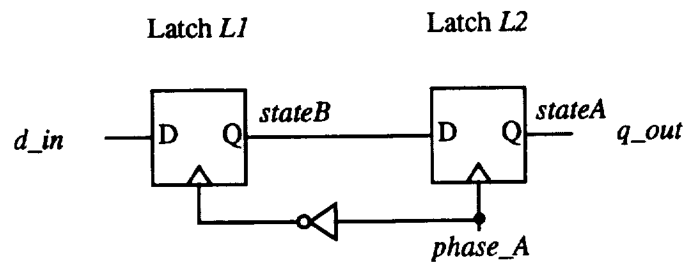
Flip-flops are built out of latches as in the example phase-A-clocked D flip-flop shown in Figure 3.3. In this model inputs arriving at the flip-flop during phase B are latched on the falling edge of B. The new flip-flop output is available at the beginning of phase A and remains stable for an entire clock period. From an edge-triggered point of view this flip-flop is seen to be clocked on the rising edge of phase A.

It is an interesting side note that in discussions with the PIU designers it became clear that their view of flip-flop behavior is somewhat different from the perspective that we employ. For example, if asked to choose which of the two latches in the flip-flop model of Figure 3.3 represents the true state of the flip-flop, the designers say latch *L2* and we say *L1*. This difference is easy to understand given the modeling environments that each group uses, and it turns out that the FSM-based specification approach embodied in Figure 3.3(b) provides a perspective to help reconcile these two viewpoints.

The PIU designers view latch *L2* as the important one because it is the only one directly visible to them during simulation. All flip-flop changes occur on the rising edge of *L2*'s clock (phase A) and the flip-flop is stable otherwise. From this perspective the purpose of latch *L1* is only to ensure the edge-triggered nature of the flip-flop by restricting possible flip-flop output values to those inputs arriving *before* phase A rises.

As formal verifiers we view *L1* as the important latch because it is clocked by phase B, the last phase in the clock cycle. This is important when we make the jump in abstraction from the phase level to the clock level and wish to eliminate one of the two state variables associated with these latches (data abstraction). As a general rule it is best to keep the latch with the most up-to-date state among the candidates for elimination, otherwise updated state will not be carried forward to the next clock cycle when the model is symbolically executed. From this perspective latch *L1* contains the essential state of the flip-flop of Figure 3.3 and *L2* serves only to control the time at which the new flip-flop state is made externally visible.

At the clock level of abstraction we model the state of a flip-flop as the contents of its phase-B latch and



(a) Functional block diagram.

```
|- DFF_SPEC d_in phase_A stateA stateB q_out =
    ∀ t:time.
      (stateB (t+1) = ~(phase_A t) => d_in t | stateB t) ∧
      (stateA (t+1) = (phase_A t) => stateB t | stateA t) ∧
      (q_out t = stateA (t+1))
```

(b) HOL representation.

Figure 3.3: Example D Flip-Flop Constructed With Latches.

embed the behavior of the phase-A latch within the flip-flop output. This FSM-based approach is also compatible with the PIU designer perspective if we take a commonly-used black box view of fundamental components such as flip-flops. In such an approach, only the inputs and *outputs* of these components are visible to an outside observer during simulation — the internal state is hidden.

3.1.1.4 Counters

Counters are implemented as flip-flops surrounded by increment/decrement and selection logic. All of the counters used in the PIU design are functionally of the form of the example in Figure 3.4 — incrementing is performed within the output stage rather than the input stage. The HOL source for all PIU counters is contained in the theory *counters_def* of Appendix A.

The inputs *ld_in* and *up_in* control the operation of this counter. If *ld_in* is active then the input *d_in* is loaded into the counter, otherwise the current value, incremented or nonincremented according to the *up_in* input, is reloaded. The input *up_in* also controls the value output by the counter.

3.1.1.5 CTR Datapath Block

The PIU R_Port contains two 64-bit counters implemented using a total of four 32-bit CTR datapath blocks. The CTR datapath blocks are themselves built from lower-level components of the compiler library, but we treat them as primitives here since they are used directly in the R_Port specification. The HOL source for the CTR datapath block is contained in the theory *datapaths_def* of Appendix A.

Figure 3.5 shows the functionality of the CTR datapath block. It behaves much like the counter of the previous section, but with additional features such as provisions for carry-in and carry-out and multiple output ports.

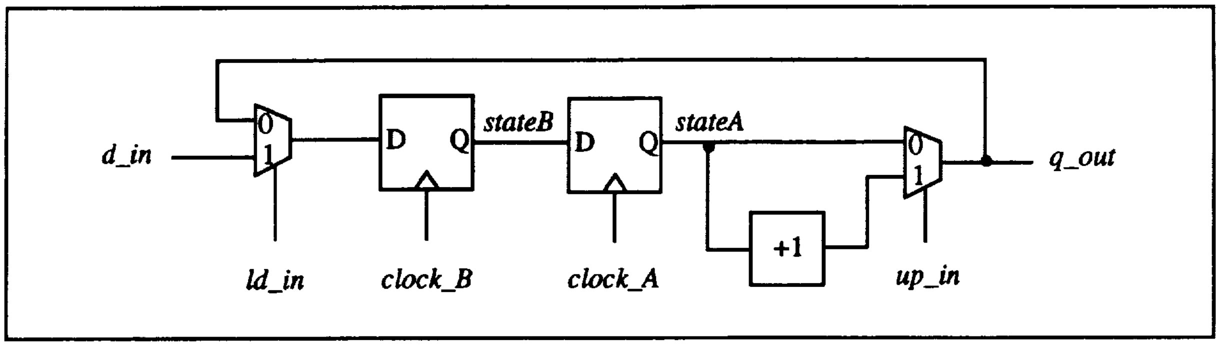


Figure 3.4: Functional Block Diagram of a Counter.

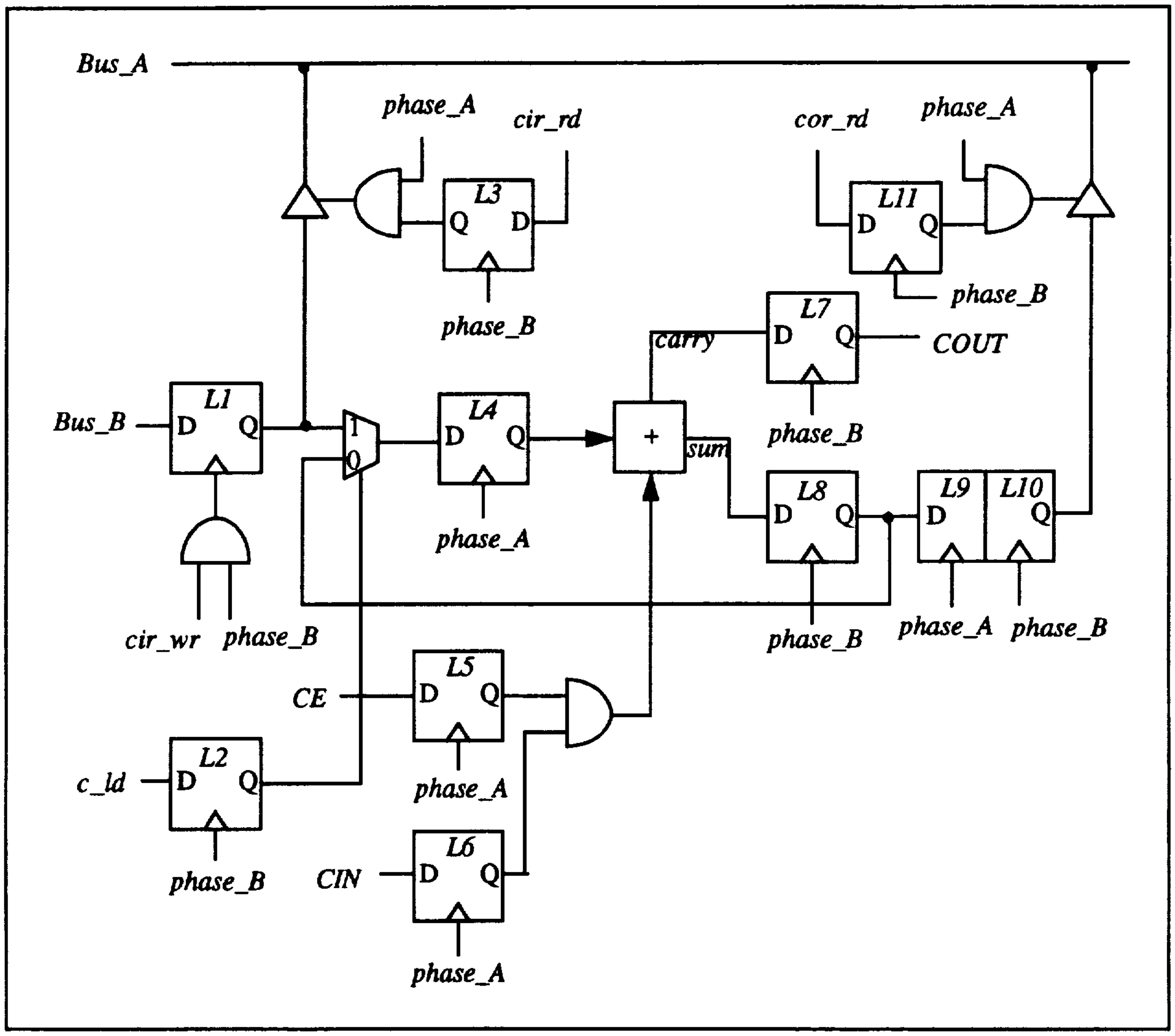


Figure 3.5: Functional Block Diagram of the CTR Datapath Block.

Of the 11 latches in this model, the one best representing the counter value is *L4*, holding the value *ctr*. Latch *L2* contains the load-input, controlling whether a new value is loaded or the updated counter value is reloaded. Latches *L1* and *L8* hold these two values, respectively. Latches *L5* and *L6* hold values controlling the incrementer itself. For the top half of the 64-bit counters, *L6* contains the carry-in from the lower half. Latch *L7* holds the carry-out from the counter. Latches *L9* and *L10* implement a flip-flop holding the updated counter value for possible output. The two latches *L3* and *L11* control the writing of latch values onto *Bus_A*, from the input side and output side, respectively.

3.1.1.6 ICR Datapath Block

The R_Port contains a single Interrupt Control Register (ICR) implementing memory-mapped interrupts for the local processor. The HOL source for this block is located in the theory *datapaths_def* of Appendix A.

Figure 3.6 shows a functional block diagram of this block. The true ICR value is located in the flip-flop implemented by latches *L4* and *L5*. The flip-flop implemented by *L1* and *L2* holds the ICR value fed back using *Bus_A*. Latch *L3* holds a mask-adjustment value that resets or sets individual mask bits according to the value of input *icr_select*. Latch *L6* controls the writing of values onto *Bus_A* either as part of an ICR read by an external processor or the feedback mentioned above.

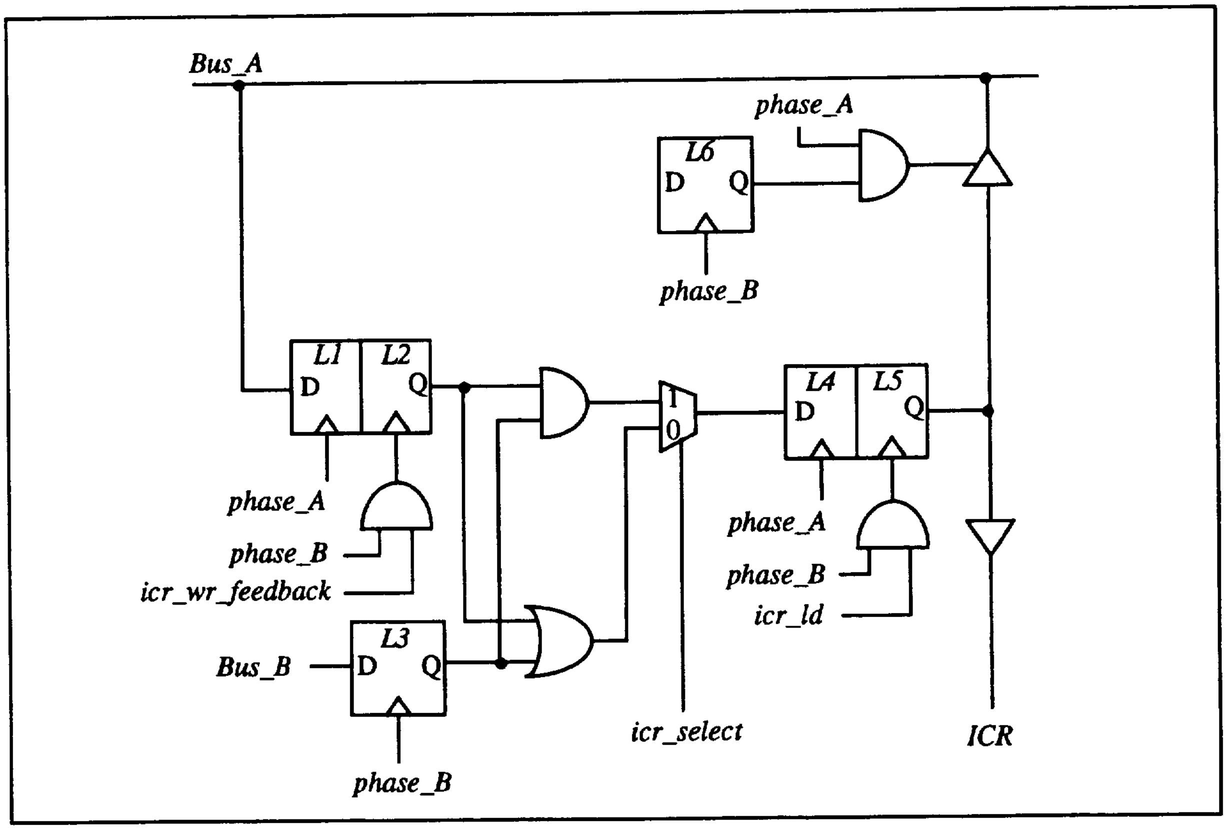


Figure 3.6: Functional Block Diagram of the ICR Datapath Block.

3.1.1.7 CR Datapath Block

The R_Port contains two control registers (CRs), called GCR (for General Control Register) and CCR (for Communications Control Register). The HOL source for the CR datapath block is located in the theory *datapaths_def* of Appendix A.

Figure 3.7 shows a functional block diagram of the CR datapath block. In comparison with the previous two datapath blocks, this one is relatively simple, containing a single latch (*L1*) to hold a loaded 32-bit value and a latch (*L2*) to control the writing of this value onto Bus_A. The second output port, always enabled, provides the CR bits to the PIU subsystems controlled by the control register.

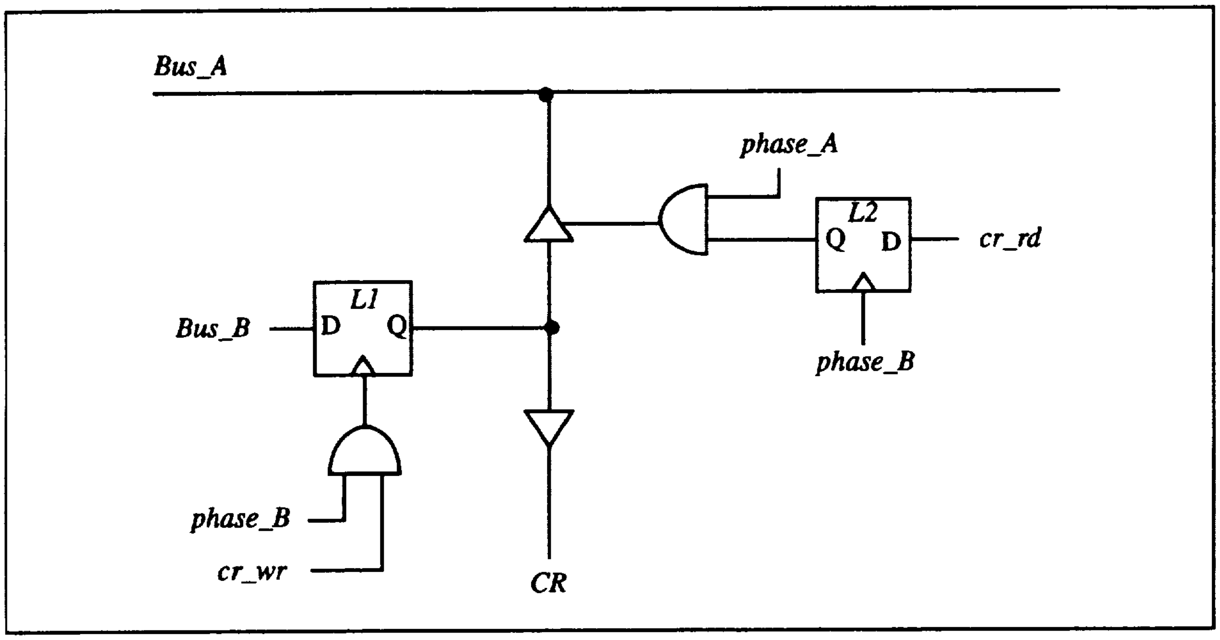


Figure 3.7: Functional Block Diagram of the CR Datapath Block.

3.1.1.8 SR Datapath Block

The R_Port contains a single Status Register (SR) that may be read by an external processor. The HOL source for the SR datapath block is located with the previous datapath blocks in the theory *datapaths_def* of Appendix A.

Figures 3.8 shows a functional block diagram of this datapath block. Inputs provided by several subsystems of the PIU are collected and stored in latch *L1*; latch *L2* controls the writing onto *Bus_A*.

3.1.1.9 Finite-State Machines

Finite-state machine (FSM) modules are used in every PIU port to control the sequencing of port operations. Each FSM module has the structure shown in Figure 3.9. FSM inputs are loaded during phase B, as is the fed back present-state. Combinational logic implements the next-state and output functions, whose results are loaded into the output latches during phase A for transmission to the external system.

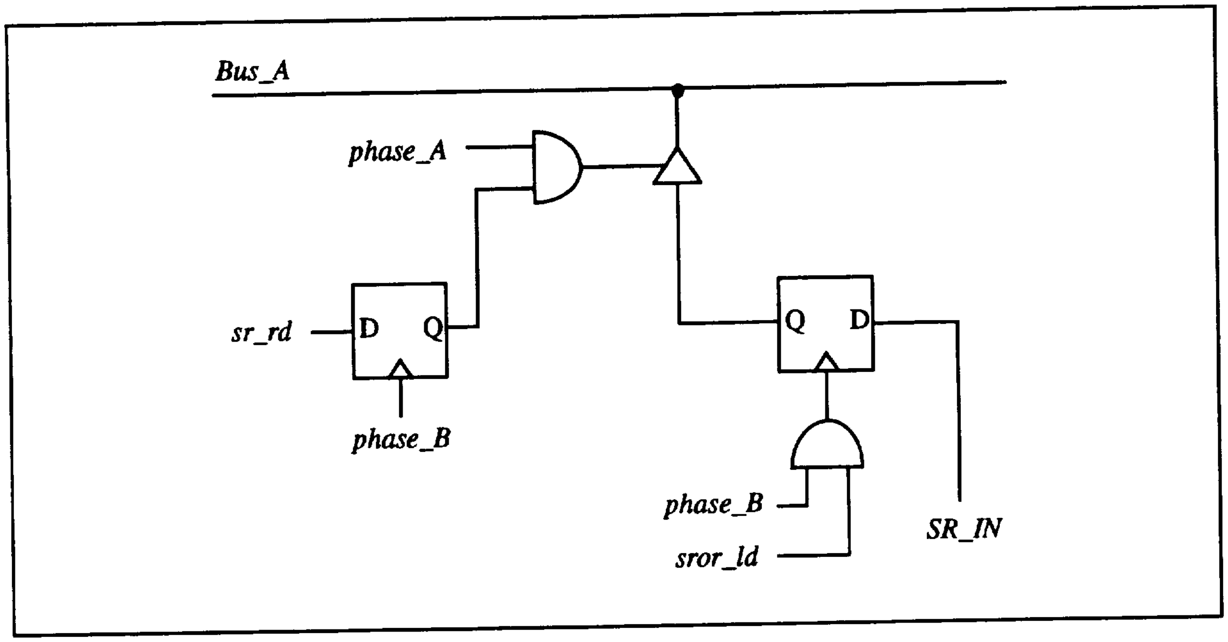


Figure 3.8: Functional Block Diagram for the SR Datapath Block.

3.1.2 Block Diagram Descriptions

To simplify the PIU specification task, we augmented the set of compiler-library components just described with several logic-blocks built of more-primitive components. Two guidelines were followed in constructing these superblocks. First, instances of multilevel logic were converted into equivalent behavioral descriptions. Secondly, memory elements holding multibit words were sometimes grouped into single blocks to facilitate modeling with our array-access functions. Together, these steps greatly decreased the number of components in the gate-level description of the PIU with a risk of introducing modeling error that we consider to be low.

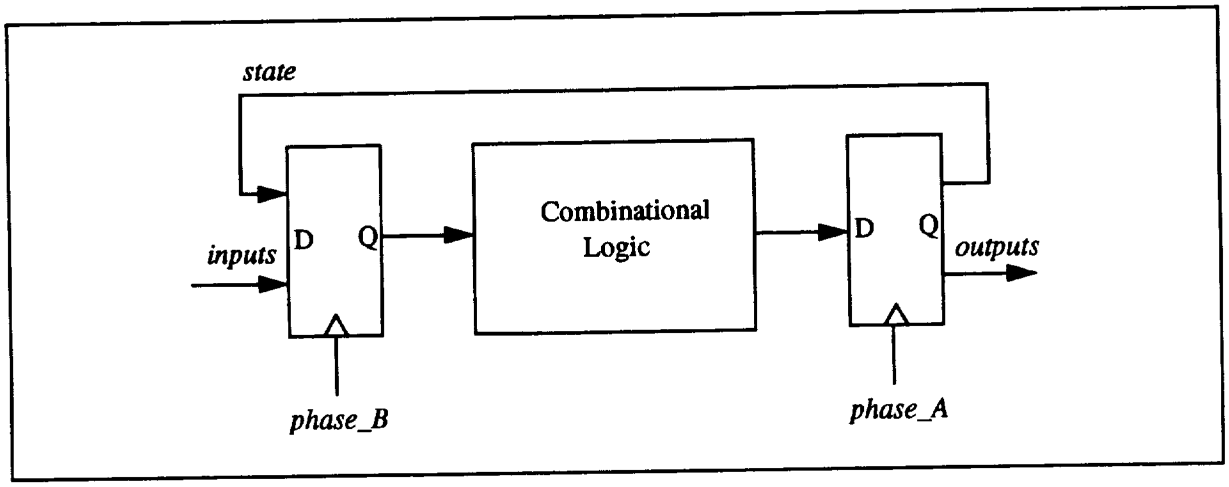


Figure 3.9: Functional Block Diagram for Finite-State Machines.

Creating superblocks also has the beneficial side effect of simplifying our description of the five PIU ports. Even so, the complexity of the resulting specification remains formidable and a fully-detatiled *pictoral* description of the PIU structure is beyond the scope of this report. The HOL descriptions in Appendix B should be considered the gate-level specification for the five PIU ports; the descriptions in this section are intended only to provide insight so that the HOL is more easily understood. Although considerable care has gone into the construction of these descriptions, they are not complete and contain minor inaccuracies as well.

The ports are described in the order: P_Port, M_Port, R_Port, C_Port, and SU_Cont, in the following five subsections.

3.1.2.1 P_Port Structure

The top-level block diagram of the P_Port, shown in Figure 3.10, describes the partitioning of the P_Port into two subblocks: datapath and controller. These are further broken down in the two figures that follow Figure 3.10.

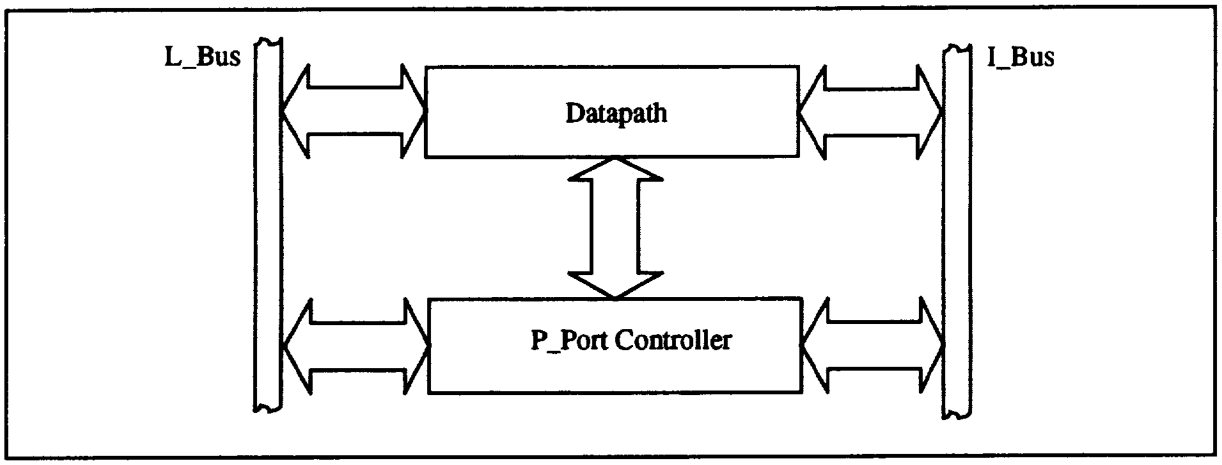


Figure 3.10: P_Port Top-Level Block Diagram.

The P_Port Datapath, shown in Figure 3.11, consists mainly of latches to hold L_Bus-sourced information and tristate buffers for driving the L_Bus and I_Bus. Read from top to bottom, the latch contents are: 32-bit data, the 26 least significant address bits, the most significant address bit, the 4-bit byte enables, and the write/read bit, all sourced by the local processor. All control signals are provided by the P_Port Controller.

The P_Port Controller is shown in Figure 3.12. The *FSM* block implements the I_Bus protocol and supports atomic memory accesses by the local processor. The other blocks support the FSM by encoding information received from the two adjacent buses and by handling some of the control-signal generation.

The *Req_Inputs* block implements the setting and resetting of the *P_rqt* latch, based on new-transaction requests and transaction-completed messages received from the L_Bus and I_Bus, respectively. An active high *P_rqt* indicates a pending or in-progress L_Bus transaction.

The *Ctr_Logic* block keeps track of the number of words remaining in the current transaction so that the slave port can be notified when the last word is being accessed.

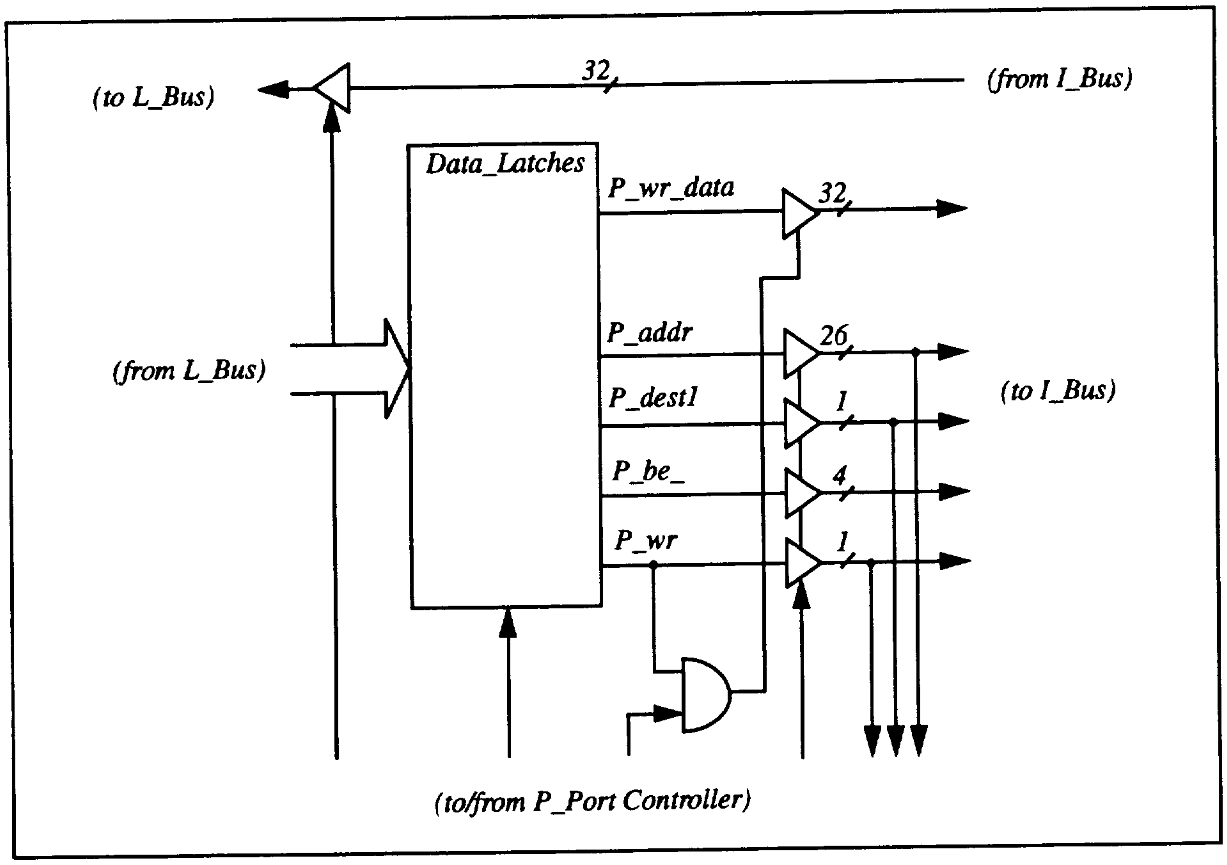


Figure 3.11: Block Diagram of P_Port Datapath.

The *Lock_Inputs* block and associated latches provide support for handling atomic operations. The *P_lock_* latch holds the most recent valid lock signal provided by the local processor. The FSM implements memory locking by locking the I_Bus.

3.1.2.2 M_Port Structure

The top-level structure of the M_Port is shown in Figure 3.13. It has the same form as the P_Port, containing a single datapath block and a single controller block. These are described further in the two figures following Figure 3.13.

Figure 3.14 shows the structure of the M_Port datapath. On the left is the interface to the M_Bus. The *EDAC_Decode_Logic* block performs a Hamming decode on the 56-bit data received from the M_Bus, while the *Enc_Out_Logic* block encodes 32-bit data for writing onto the M_Bus.

The *Read_Latches* block stores the 32-bit decoded data word read from memory. The *Mux_Out_Logic* block selects bytes from this stored value or else the word currently on the I_Bus for writing onto the M_Bus. The stored bytes are written back as part of a read-modify-write implementation of byte-write operations.

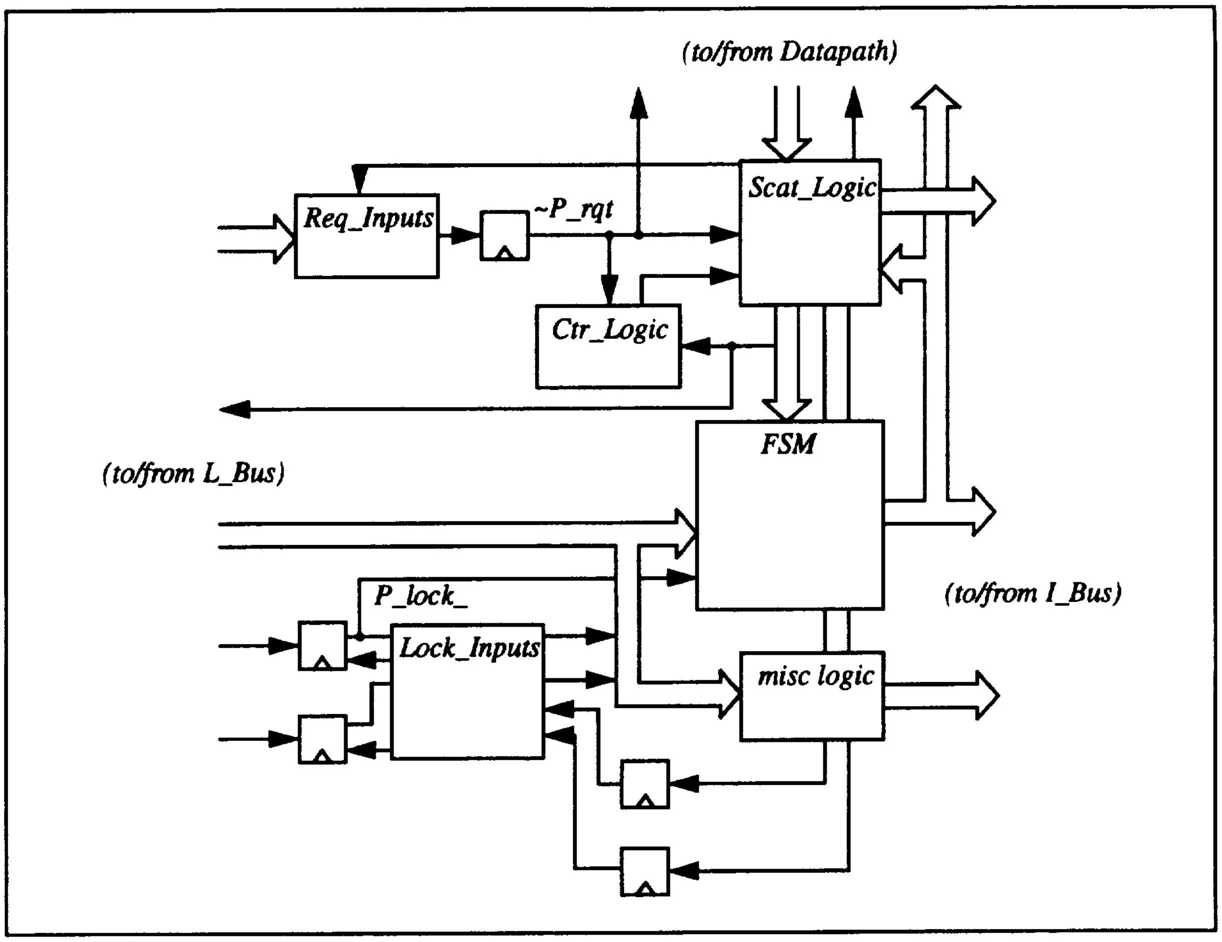


Figure 3.12: Block Diagram of the P_Port Controller.

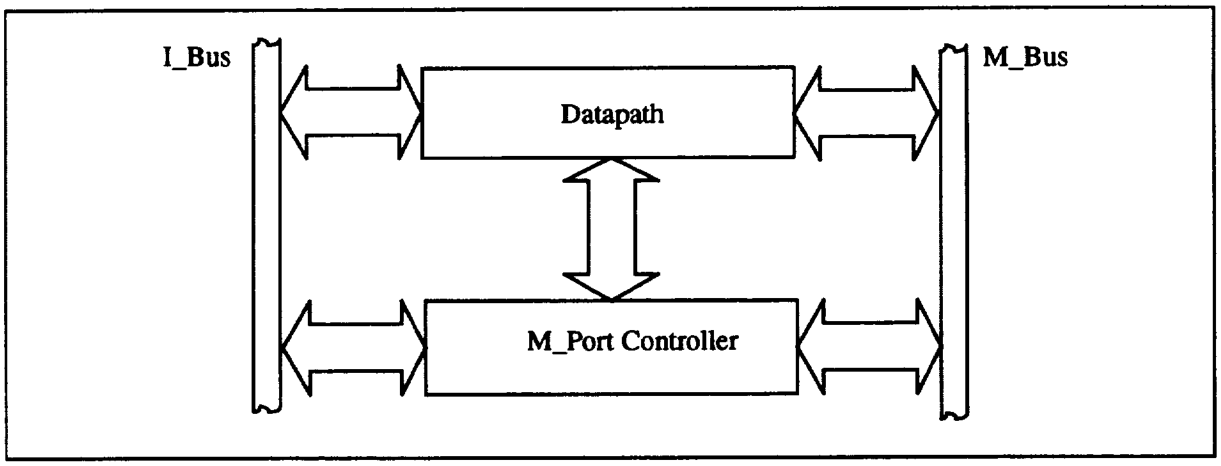


Figure 3.13: M_Port Top-Level Block Diagram.

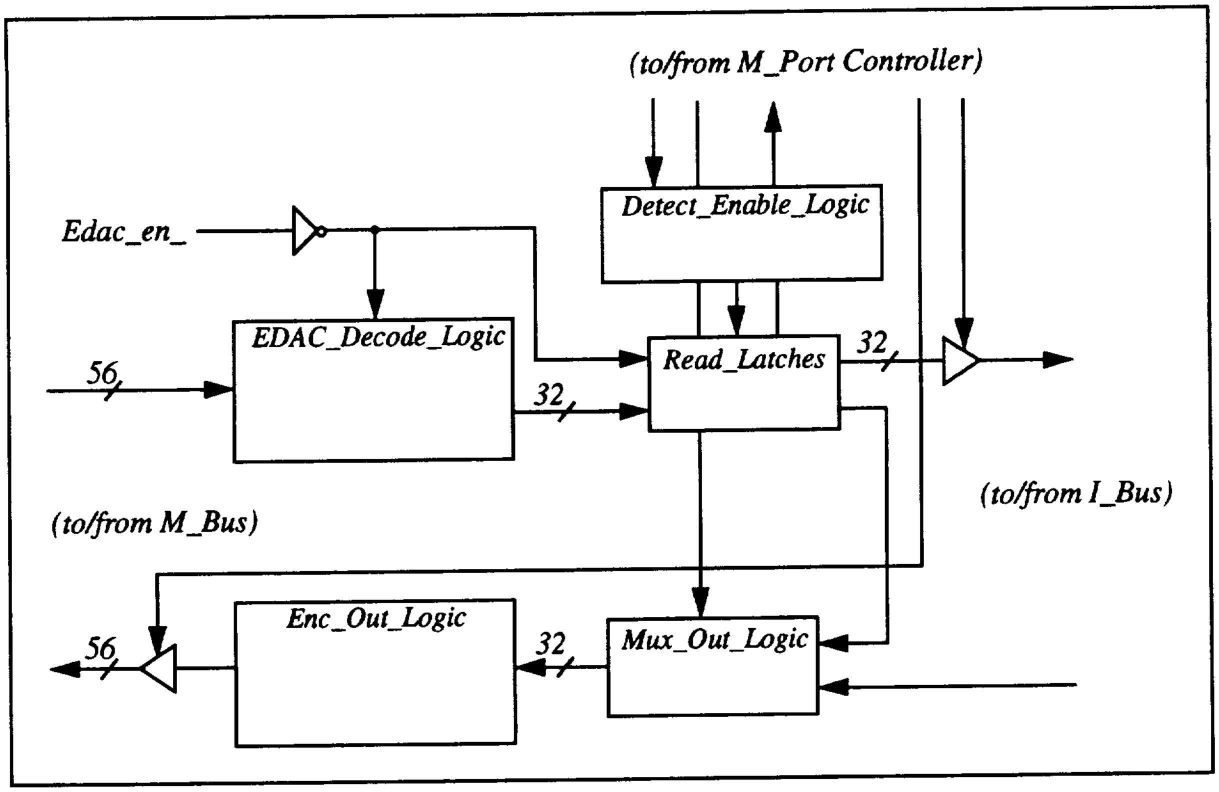


Figure 3.14: Block Diagram of the M_Port Datapath.

The M_Port controller is shown in Figure 3.15. The left side of the figure is the I_Bus interface. The *SE_Logic block* determines whether a memory access is to SRAM memory or to EEPROM memory, based on the memory address. It drives the appropriate chip-select signal based on this determination.

The *WR_Logic* block determines whether a memory access is a read or write and provides this information to the rest of the M_Port. The *Addr_Ctr* block and *BE_Logic* block store the memory address and byte enables, respectively, for the word being accessed.

The *Rdy_Logic*, *Ctr_Logic*, and *Srdy_Logic* blocks together implement most of the I_Bus protocol for the M_Port, which consists mainly of controlling the value of the *I_srdy_* signal transmitted back to the I_Bus master. The 2-bit counter in Ctr_Logic implements variable wait-states for the SRAM and EEPROM memory.

The *FSM* block provides high-level control of the memory interface. It sequences through a series of states, depending on the type of memory transaction, and provides output signals mainly used by the *Enable_Logic* block to implement the control of the M_Port datapath. The FSM also directly controls bus enabling for the I_Bus.

The *Memparity_In_Logic* block and its associated latch store the error status for memory accesses. The output *MB_parity* is transmitted to the R_Port where it is stored in the Status Register.

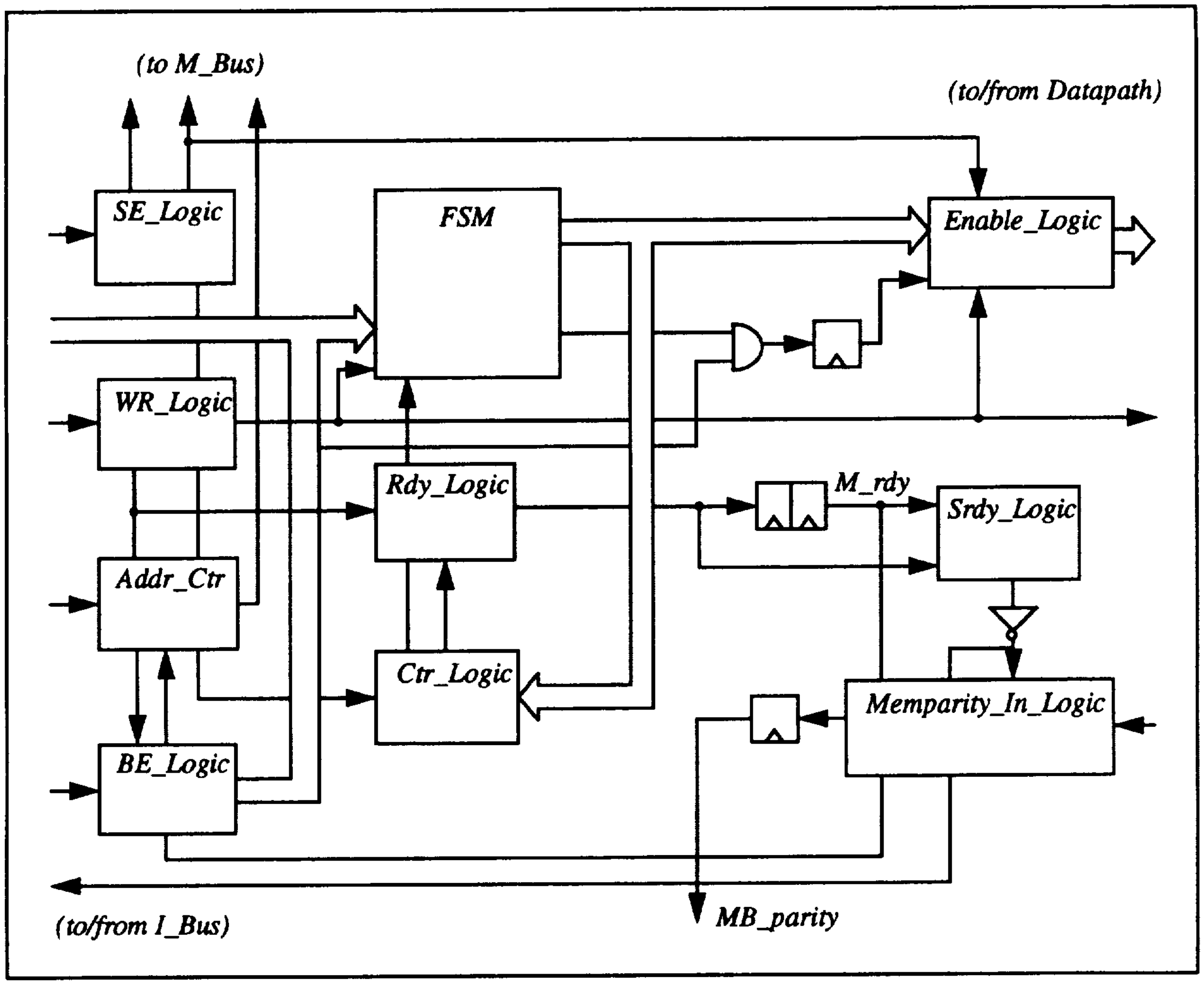


Figure 3.15: Block Diagram of the M_Port Controller.

3.1.2.3 R_Port Structure

The R_Port top-level block diagram is shown in Figure 3.16. Of the five major blocks shown there three are described further in the figures that follow Figure 3.16. The Register File block is not broken down further since it consists entirely of the datapath blocks described in Sections 3.1.1.5 through 3.1.1.8. There are four CTR blocks implementing two 64-bit counters, one ICR block, two CR blocks implementing the GCR and CCR, and one SR block.

The Bus Interface block represents the multiple tristate buffers that potentially drive the Bus_A node of the R_Port. This block is similar to the approach used to model buses described in [Joy90].

The Register File Controller is shown in Figure 3.17. The *Wr_Lat* block determines whether a register access is a read or write and provides this information to the rest of the R_Port. The *FSM* block is a simple 3-state state machine providing high-level control of the register accesses and I_Bus interface. The *RW_Sigs* block encodes the FSM output to implement this control.

The *Reg_Sel_Ctr* block contains a 4-bit counter holding the register number for the current access. The *R_srdy_del_* latch value is used to increment the counter on multiword accesses. The *Reg_File_Ctl* block

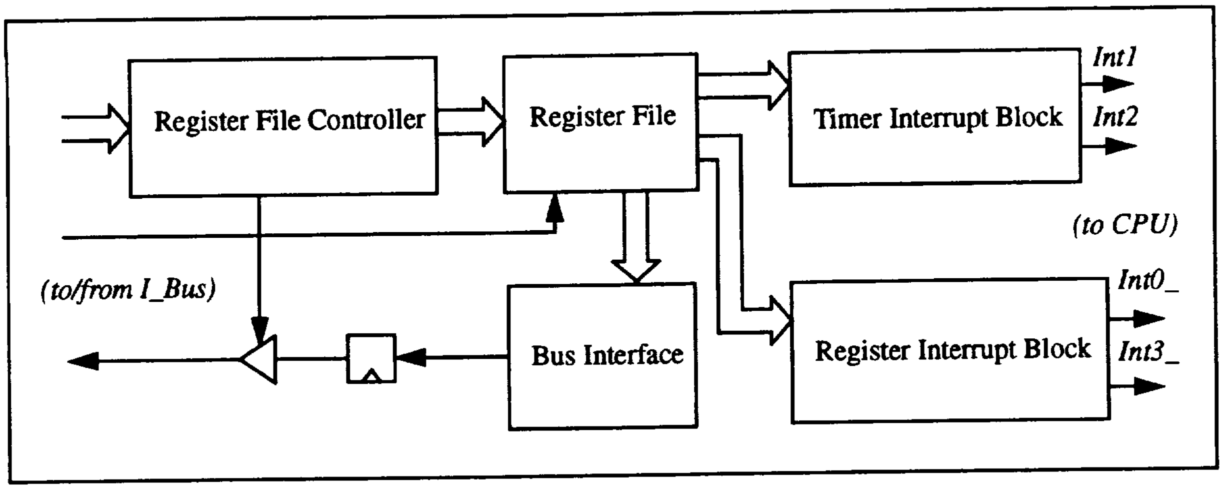


Figure 3.16: R_Port Top-Level Block Diagram

decodes the register address to create most of the control signals needed by the register file.

The Timer Interrupt Block is shown in Figure 3.18. It consists of two identical sub-blocks, each implementing the interrupt logic for one of the two 64-bit counters.

The latches *R_c01_cout* and *R_c23_cout* hold the carry-out values of the two counters. The *Ctr_Int_-Logic* blocks use this information and several bits of the GCR to determine whether the timer interrupts should be enabled or not. The two interrupt outputs, *Int1* and *Int2*, are active-high signals sent to the local processor.

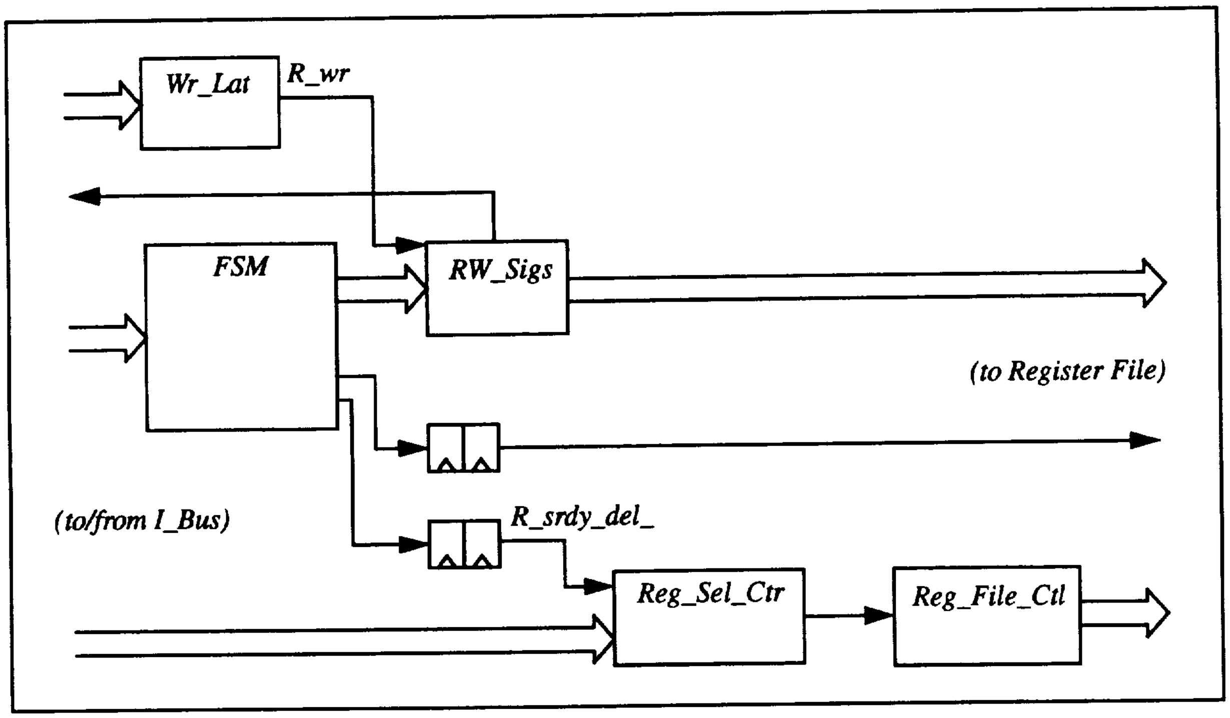


Figure 3.17: Block Diagram of Register File Controller.

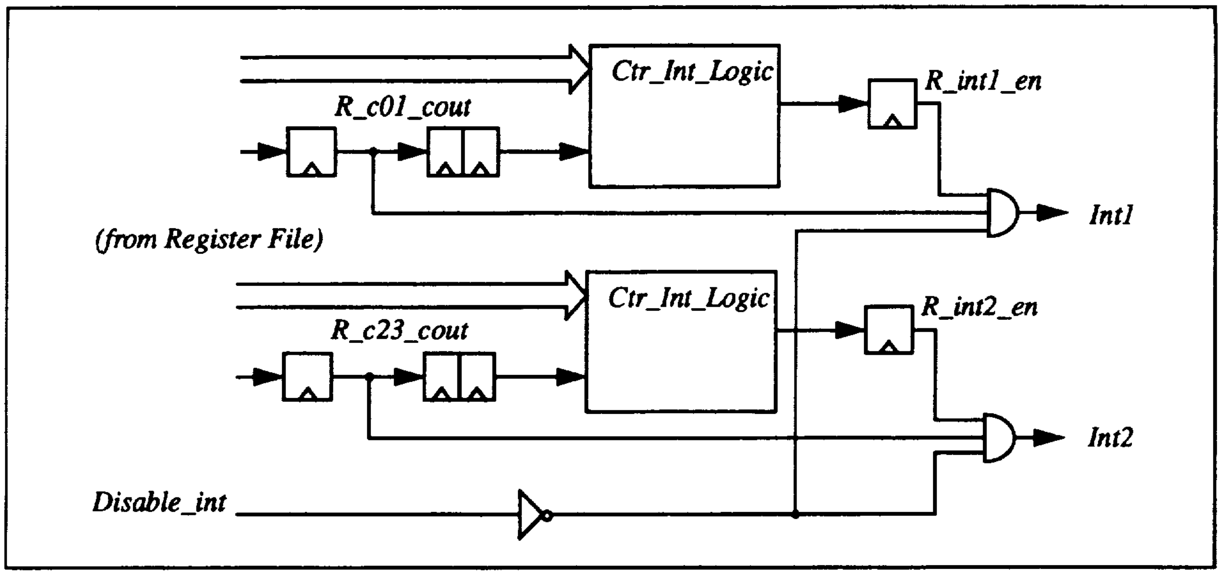


Figure 3.18: Block Diagram of the Timer Interrupt Block.

Figure 3.19 shows the structure of the Register Interrupt Block. The *And_Tree* block receives the 32-bit ICR value, consisting of 16 interrupt-set bits and 16 mask bits. Half of these bits are dedicated to interrupt *Int0_* and half to *Int3_*. If an interrupt-set bit and its associated mask bit are simultaneously active-high, then the appropriate latch, *R_int0_en* or *R_int3_en*, is loaded with a logic-1.

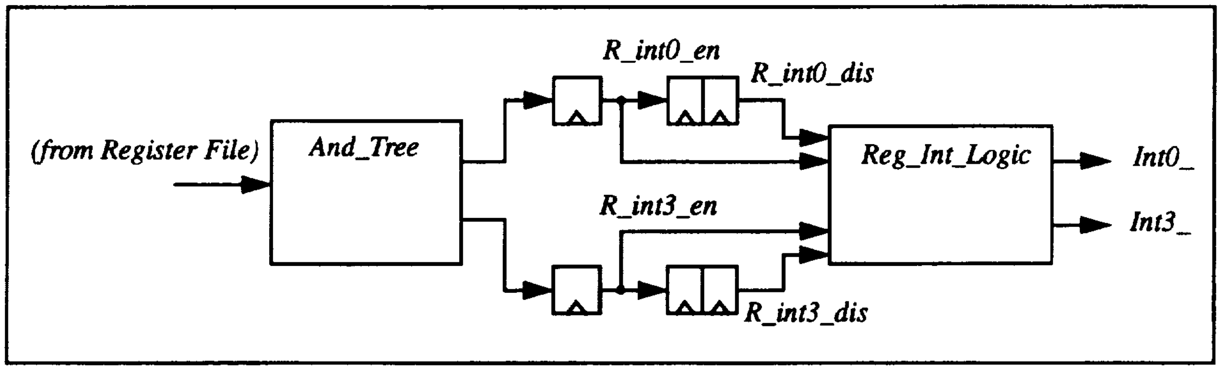


Figure 3.19: Block Diagram of the Register Interrupt Block.

3.1.2.4 C_Port Structure

The C_Port top-level structure is shown in Figure 3.20, minus the complicated external interfaces. The C_Port controller is divided into two subunits because of its large size. Because we could not identify a logical partitioning, we simply divided the existing schematic down the center, creating a left half and a right half, controllers A and B, respectively.

Figure 3.21 shows the C_Port datapath block diagram. The right side of the figure shows the interface

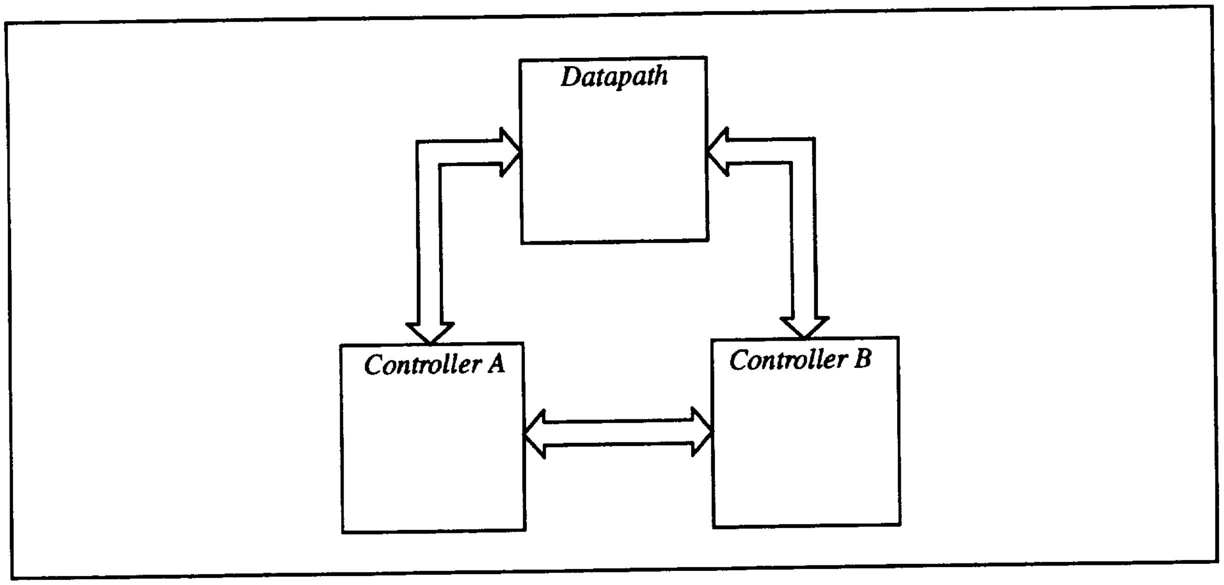


Figure 3.20: C_Port Top-Level Block Diagram.

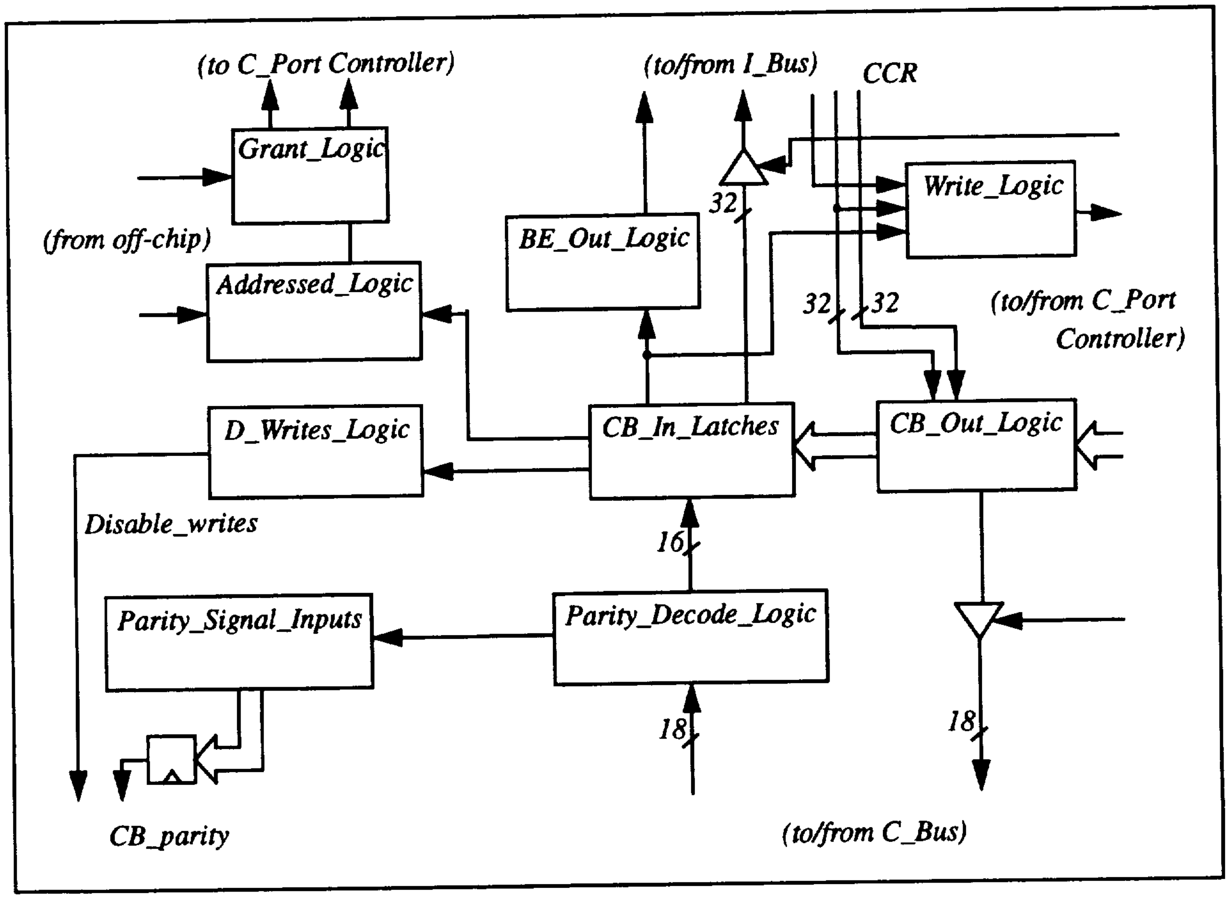


Figure 3.21: Block Diagram of the C_Port Datapath.

between the I_Bus and the C_Bus. The *Parity_Decode_Logic* block decodes the 18-bit parity-encoded data received from the C_Bus data lines. It outputs 16-bit data and a single-bit error-detection flag.

The *CB_In_Latches* block stores the messages received from the C_Bus. This information consists of transaction header information, address, and data. The *BE_Out_Logic* block outputs the byte enables onto the I_Bus. The *CB_Out_Logic* block parity-encodes data for transmission onto the C_Bus.

On the left side of the figure, the *Grant_Logic* block implements the C_Bus arbitration. The *Addressed_Logic* block determines whether this PIU is being addressed by the C_Bus master. The *D_Writes_Logic* block determines whether this PIU is an active channel or not; if not then it prohibits memory accesses using the *Disable_writes* output. The *Parity_Signal_Inputs* block controls the setting and resetting of the *C_parity* latch, whose output, *CB_parity*, is transmitted to the R_Port SR.

Part (A) of the C_Port controller is shown in Figure 3.22. The two state machines: *Master FSM* and *Slave FSM*, implement the C_Bus protocol from the master and slave perspectives, respectively. The *Srdy FSM* controls the enabling of I_Bus slave signals transmitted by the C_Port.

The *Last_Logic* block and the latches holding *C_lock_in_* and *C_last_in_* preprocess the *I_lock_* and *I_last_* I_Bus signals received from the P_Port. The *Hold_Logic* block and the latches holding *C_last_out_* and *C_hold_* process the *I_last_* and *I_hold_* signals transmitted over the I_Bus. The *Cout_Sel_Logic* block determines which 16-bit word is to be transmitted over the C_Bus and provides selection signals to the datapath to control this.

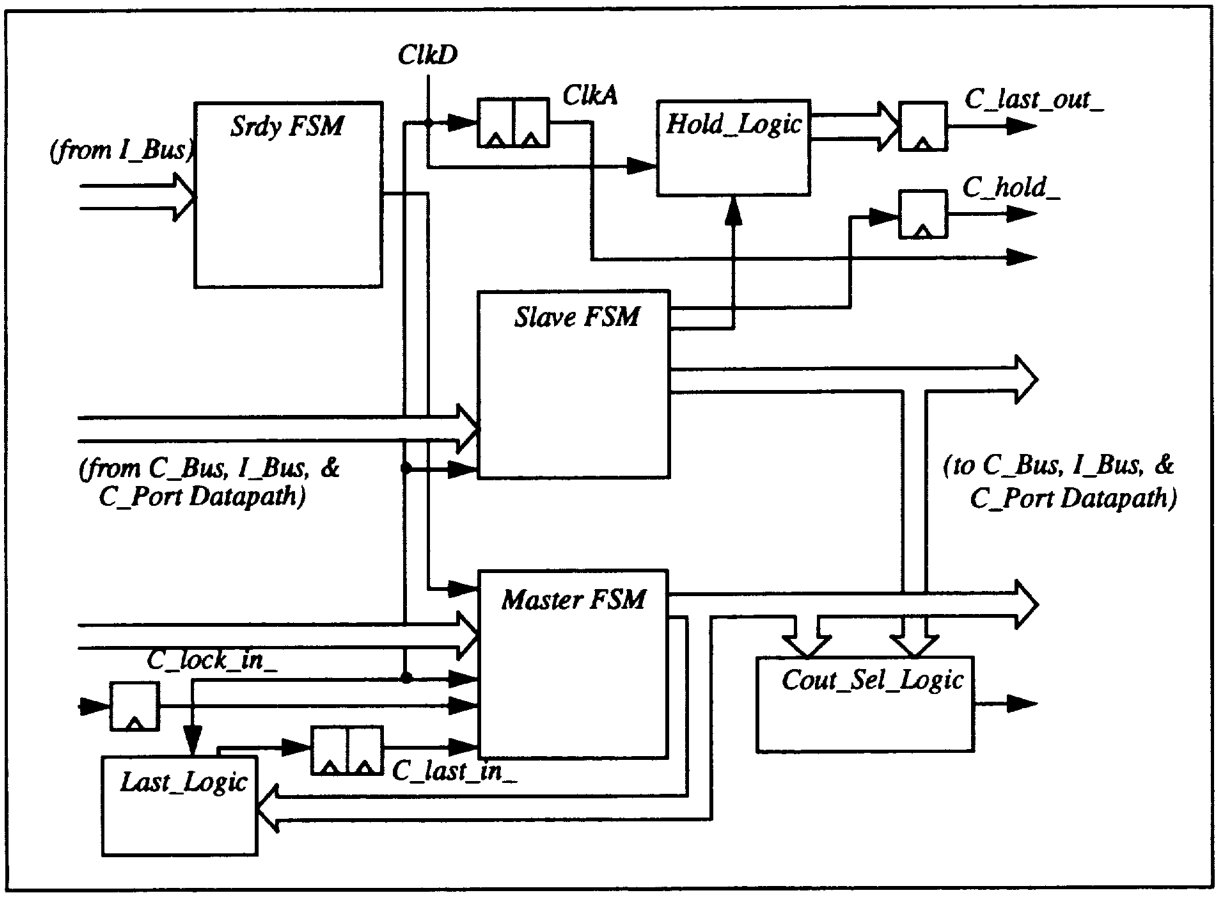


Figure 3.22: Block Diagram of the C_Port Controller (Part A).

Figure 3.23 shows part (B) of the C_Port controller. The *DP_Ctls PLA* block converts output signals from both the master and slave state machines of part (A) into control signals for the datapath. The latches at the output of this block, as well as the *Cout_1_Le_Logic* block, provide further processing for the datapath, primarily to control the enabling of the datapath latches.

The *CBss_Out_Logic* block and the *CBms_Out_Logic* block determine the master-status and slave-status, respectively, for C_Bus transactions. The *Srdy_In_Logic* block decodes the slave-status input from the C_Bus to determine whether the slave is ready for the next transaction.

The *Rdy_Logic* block, the *ISrdy_Out_Logic* block, and intervening latches implement the generation and transmission of the *I_srdy_* signal to the I_Bus. The *Iad_En_Logic* block controls the enabling for address and data transmissions over the I_Bus.

The *Pe_Cnt_Logic* block controls the enabling of parity-error counting within the datapath.

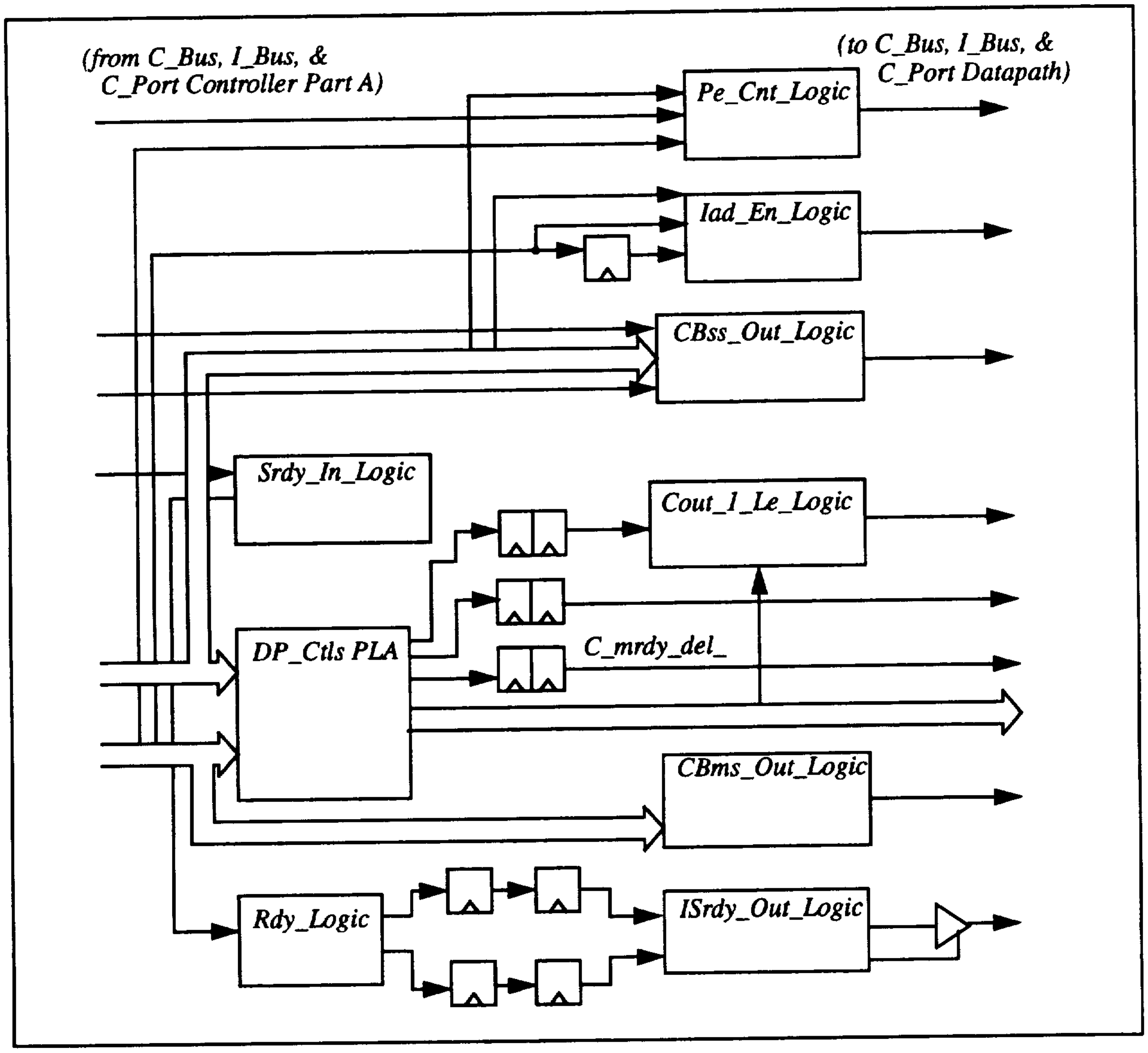


Figure 3.23: Block Diagram of the C_Port Controller (Part B).

3.1.2.5 SU_Cont Structure

The SU_Cont structure is divided into the two subsections shown in Figures 3.24 and 3.25. The first figure shows mainly the blocks that interact with the other ports within the PIU, while the second shows mainly those that interface with the local processor.

The *FSM* block in Figure 3.24 controls the initialization process. It sequences through states that successively reset and test CPU0, reset and test CPU1, then select and initialize the active mission processor. It uses the output of the 18-bit *counter* block, via the *Muxes* block, to control its time duration in many of its states. The *Delay_In* block processes the input signals for the *counter* block.

The *Dis_Int_Out* block determines and then transmits reset signals and various disable signals to the other ports.

The blocks *Scnt_In, Scnt_In1*, the 3-bit *counter* block, and the intervening latches support the software-based acceptance test of each processor. The output *S_Soft_Cnt* contains the number of instances that the local processor writes a specific pattern to the General Control Register in the R_Port. If not equal to a specific bit pattern, this counter value indicates a failed acceptance test.

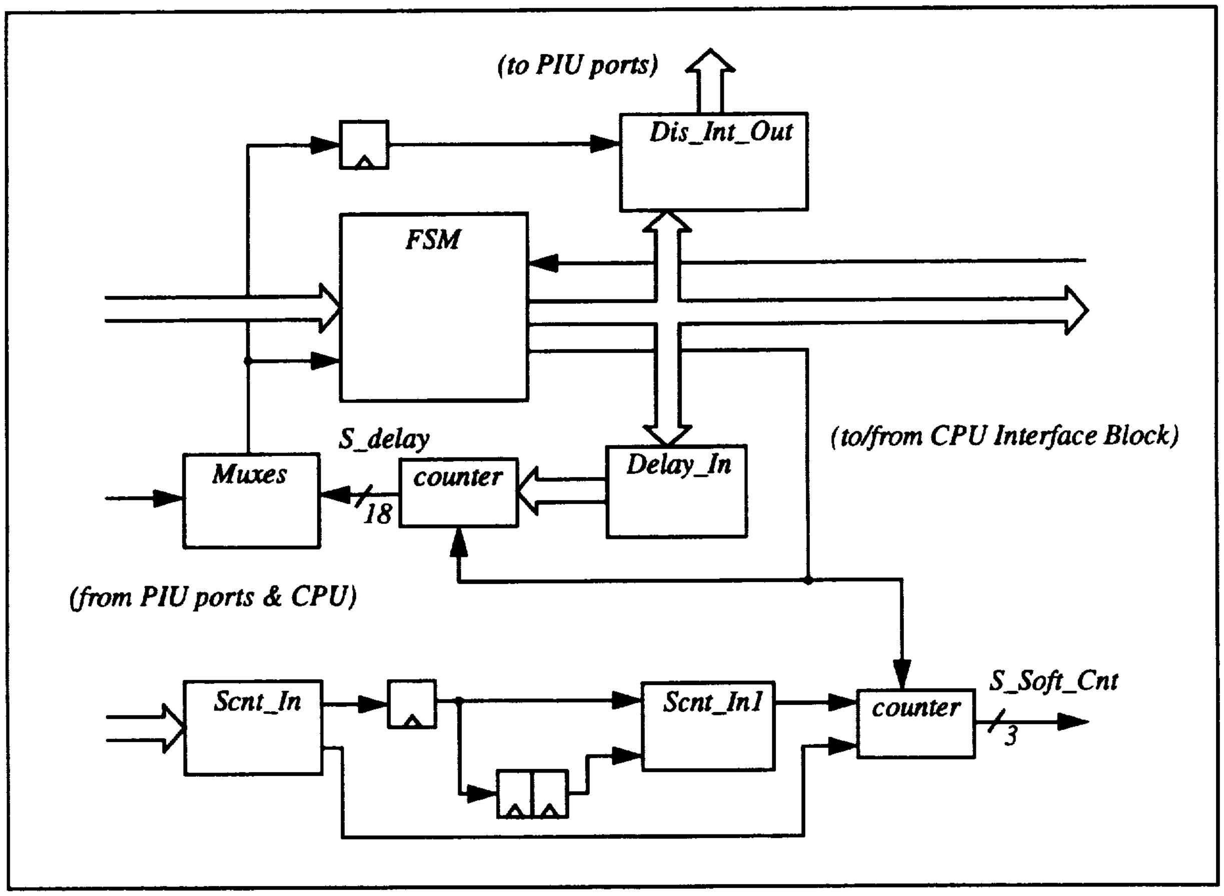


Figure 3.24: Block Diagram of the Startup Controller PIU-Port Interface.

Figure 3.25 shows the SU_Cont blocks that interact mainly with the local processor. The *Cpu_Ok* block and the *Fail_In* block together control the loading of four latches holding failure-status information. The *Cpu_Ok* block uses the *S_Soft_Cnt* signal just discussed and the *Failure_* signals from the local processors. The latch outputs are transmitted to the R_Port where they are stored in the Status Register.

The *Bad_Cpu_In* block controls the loading of two latches holding processed failure status of the two local processors. These latch outputs are used, together with *FSM* block outputs, in the *misc logic* block to control the loading of two other latches. These latch outputs are used to maintain the local processors in a reset or nonreset state, as appropriate.

3.2 Port Phase-Level Behavior

The phase-level specification for each PIU port is a behavioral abstraction of the corresponding gate-level structure. Each port is defined in terms of a 2-instruction instruction set, corresponding to the behavior occurring during each of the two clock phases. Each instruction is itself represented using two functions, defining the next-state transition and the output. Consistent with the generic interpreter model, the states and outputs for the ports are represented as n-tuples.

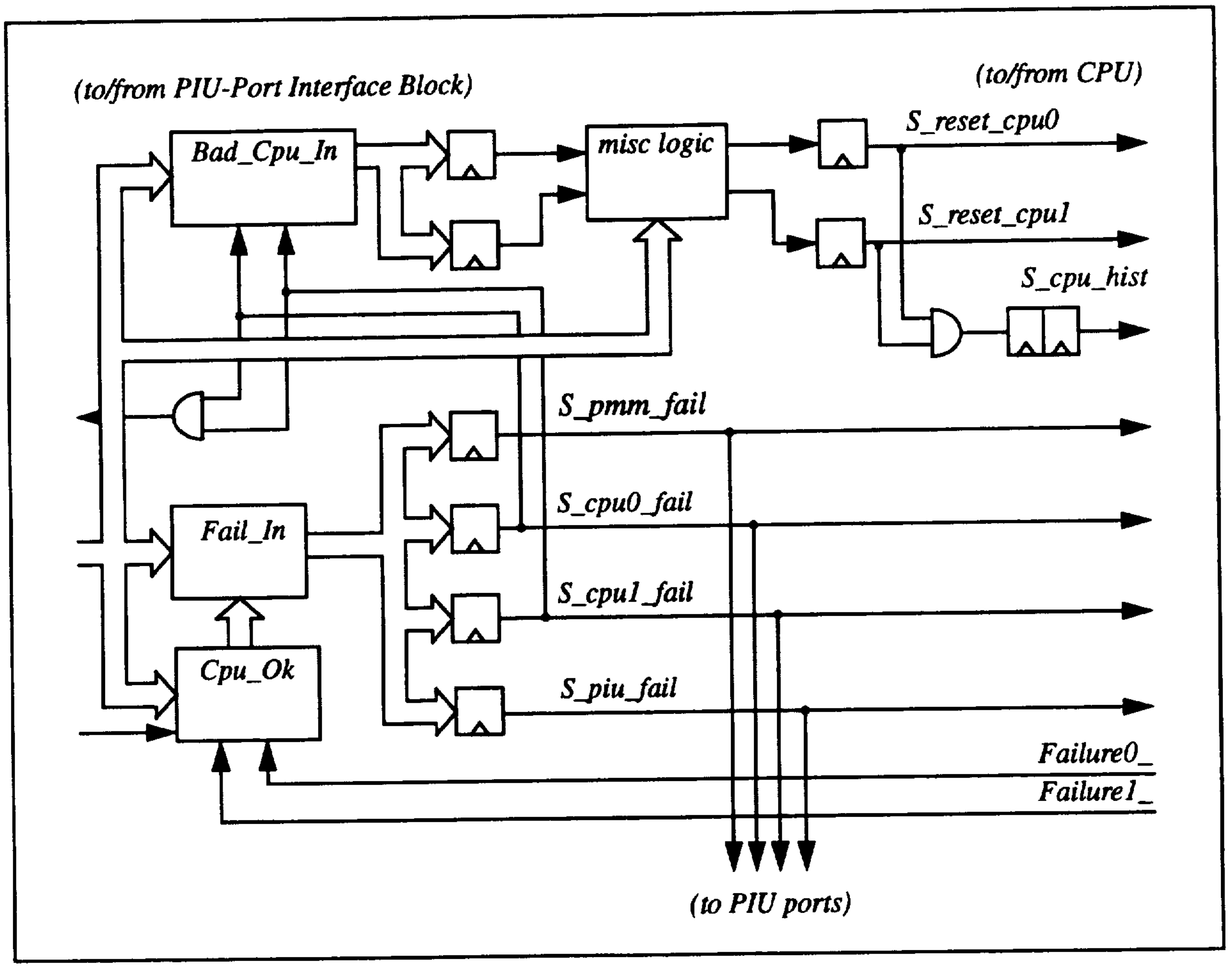


Figure 3.25: Block Diagram of the Startup Controller CPU Interface.

Appendix C contains the HOL phase-level specification. The ports are presented in the order: P Port, M Port, R Port, C Port, and SU_Cont, in Sections C.1 through C.5, respectively. Within each section the next-state function for phase A is presented first, followed by the output function for phase A, and the next-state and output functions for phase B.

3.3 Port Clock-Level Behavior

The clock-level specification for each PIU port is both a temporal abstraction and a data abstraction of the corresponding phase-level specification. Here the unit of time is an entire 2-phase clock period, rather than a single phase. Data abstraction is achieved by eliminating state variables representing certain latch values. Usually the eliminated latches are part of edge-triggered devices, such as flip-flops and counters, and are clocked on phase A.

In contrast to the phase level, where the choice of instruction set is dictated by the number of clock phases, the choice at the clock level is much more subjective. For example, only a single instruction is really necessary to capture the behavior of the ports. This would provide the most concise description of behavior at the cost of providing the least understandable description. At the opposite extreme, the ports could be specified using an instruction set with millions of very simple and easy-to-understand instructions. However, verifying such a large instruction set would be infeasible, as would the mere goal of trying to print their descriptions.

Instruction sets provide the human interface to state-transition system behavior. Their existence implies an instruction selection capability such as that provided by the *select* function of the generic interpreter model. Often this functionality is referred to as instruction *decoding*, and the proper choice of this function (i.e., of the instruction set itself) is important for any specification attempting to provide a human-understandable yet concise description of behavior.

By their very nature, *microprocessor* instruction sets at the macro and microcode levels must be straightforward to specify since they provide the programming interface for the microprocessor. However, since the PIU was never intended to be programmed, nor is it microcoded, (clock-level) instruction set elegance received little consideration from the PIU design team. As a result, a clock-level instruction set for each port in which each instruction specifies a single well-defined action would require many tens of individual port-level instructions. The composition of these port-level instructions would require many tens or hundreds of PIU-level instructions, requiring many thousands of pages to even print; verifying these instructions would be an enormous undertaking.

Based on these considerations, we have abandoned our earlier efforts to define human-friendly instruction sets at the clock level. Instead we have opted for practicality and we specify clock-level behavior using a *single* instruction for each port. Each port instruction has two parts — a next-state function and an output function, defining the next state and output under all operating conditions. Sections D.1 through D.5 of Appendix D contain the HOL specification for this level.

3.4 PIU Port-Level Structure

The PIU port-level structure is a structural composition of the five clock-level port specifications. We have used the standard approach to structural composition in which component-defining predicates are logically ANDed to form the composite behavior. Existentially-quantified variables are used for component outputs remaining internal to the composed system. Appendix E contains the HOL specification for this level.

3.5 PIU Clock-Level Behavior

Appendix F contains the HOL specification for the PIU clock-level behavior. As with the individual ports, the clock-level behavior of the entire PIU is represented using only a single instruction consisting of a next-state function and an output function.

4 Models for Transaction Specification

This section describes the work undertaken to determine the most appropriate model for specifying the top level of the Processor Interface Unit (PIU).

4.1 Introduction.

To complete the specification of the PIU, a top–level specification of the required behavior of the PIU must be written. This behavioral model should describe the actions of the device with respect to its environment and internal state.

The PIU is essentially a bus controller. However, there are some differences: the PIU contains special features for fault tolerance and dependability, such as an encoding of words sent to memory for error correction and the ability to select between two processors depending on the results of a power–on self test.

Our goal is to model each of the concurrent portions of the PIU individually using an interpreter (as discussed in Section 2) and to show that a composition of these interpreters entails the behavior of a more abstract model. At first, we believed that the composite behavior of the PIU could be described using the interpreter model as well. However, we found that the high–level behavior of a device such as the PIU is not easily modeled as an interpreter.

An interpreter is a computational device with one major control point. That is, one of a set of instructions is chosen based on the current state and that instruction is used to process the state; following the execution of the instruction, the process begins anew. While interpreters describe many interesting devices, the model is too restrictive to describe the PIU.

There are at least three aspects of the intended behavior of the PIU that make it difficult to describe using existing techniques:

- The feature of a bus controller that causes the greatest difficulty in using an interpreter model to describe it is its *concurrency*—a bus controller does many things at once. For example, most bus controllers contain timers that, in conjunction with an on–board interrupt controller, can interrupt the CPU. These timers operate concurrently with other portions of the bus controller, such as memory and network operations.
- A typical top–level specification of the PIU might include the memory subsystem because this corresponds to the CPU's view of the PIU (see the next section for a more complete discussion of this). This *shared state* between the PIU and other devices makes description using an interpreter model difficult.
- The outputs of the PIU do not correspond on a one–to–one basis with the inputs; there is a *many–to–one relationship* between the outputs and inputs. The interpreter model assumes that the output at a particular time is described by a function on the current state and environment. The PIU may make several outputs in sequence because of a single input request (a block memory read request is a good example).

In exploring possible models for use in describing the behavior of hardware devices such as bus controllers, we were concerned with the following issues:

- The notation and semantics should be amenable to embedding and automation in an automatic theorem prover such as HOL.
- The model and notation should be sufficiently general to allow a large number of interesting devices to be described.
- The model and notation should be sufficiently defined to allow a rich set of theorems to be proven about it in isolation of any particular application.

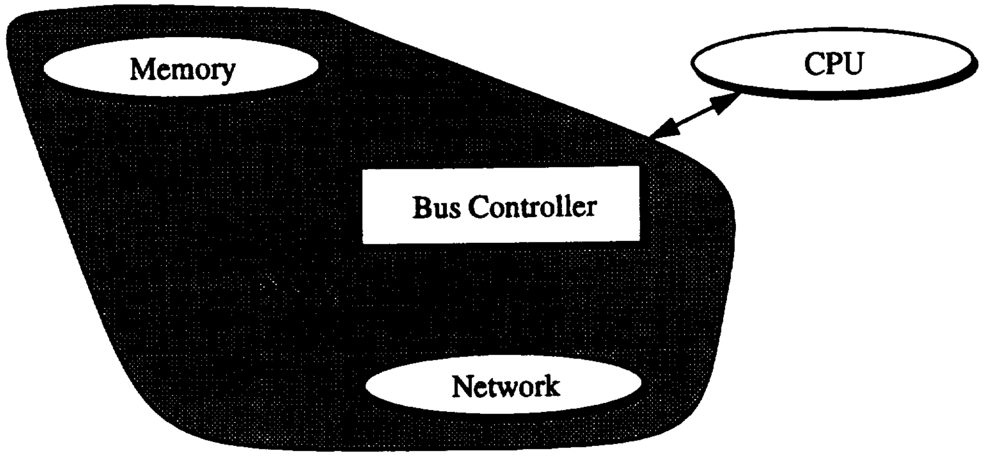


Figure 4.1: The view from the CPU.

4.2 Abstract Views

Before exploring specific notations for describing the PIU, we consider some of the features of the PIU that make its behavioral specification interesting. These abstract views contribute to the understanding necessary to specify its operation. In general, the behavior of the PIU can be looked on as a combination of behaviors from different viewpoints: that of the CPU, the network, and the memory. In order to simplify the discussion that follows, we will ignore certain behaviors of the PIU. In particular, we will assume that the start–up processor is finished and that the PIU is in steady–state operation.

Figure 4.1 shows the abstract view of the PIU from the CPU. In this view, the CPU sees the combination of the PIU, Network, and Memory (PNM) as a monolithic address space. Similarly, interrupt signals can be viewed as coming to the CPU from this abstract object rather than the individual components.

In the CPU view, when the CPU issues a read request to the PNM, the PNM responds with the information located at the virtual address given by the CPU. The actual location of the requested data, that is, whether it resides in local memory, remote memory, or a register in the PIU, is abstracted away. Similarly, when the CPU issues a write request, it does not know whether the request will update local memory, remote memory, or a register in the PIU.

Of course, inside the CPU view, the PIU either responds to requests from the CPU itself, or by issuing other requests to the network or the memory. Specifying what requests the PIU makes to other devices in response to a request from the CPU can be viewed as a specification of the implementation of the PNM. Another way of viewing these requests is that they will be specified in the other views of the system. The latter is the method we employ.

Figure 4.2 shows the view from the memory. The memory can be viewed as a processor, albeit a simple one. In the memory view, the PIU/CPU/Network abstraction (PCN) makes memory read and write requests and the memory responds appropriately. Because the memory device is simple, it makes no requests of the PCN itself, but only responds to requests.

The fact that some of these requests originated with the CPU and others with other hosts on the network is abstracted away. Inside the PCN abstraction, of course, the requests to the memory are originating with the CPU or the network and after some processing by the PIU (such as error correction encoding and decoding) are being passed on. The relationship between requests from the CPU and the network do not necessar-

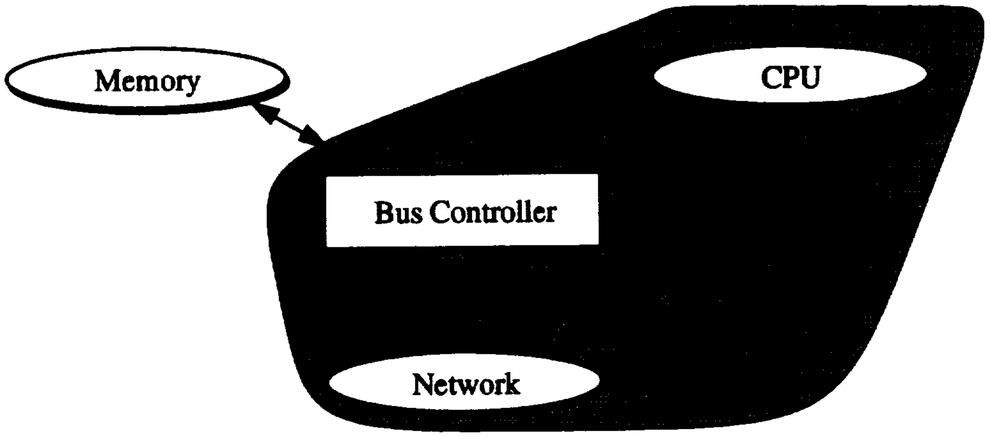


Figure 4.2: View from the Memory.

ily correspond on a one–to–one basis with the requests sent to the memory. A single request from the CPU may result in many requests to the memory.

Figure 4.3 shows the view of the PIU from the perspective of the network. In this view, the PIU, memory, and CPU are abstracted into a single object (PMC). This is, perhaps, the most complex abstraction. The network makes requests of the PMC and the PMC makes requests of the network. These requests are primarily memory read and write requests.

The problem with the views presented in Figures 4.1–4.3 is that the abstractions include the behavior of the CPU, network, and memory. Our goal is to specify the behavior of the PIU independent of the devices to which it is connected. Each of these views can be thought of as a specification of the abstract interface to one portion of the PIU. As Figure 4.4 shows, we can superimpose the specifications on one another. The union of the PNM, PCN, and PMC specify the behavior of the entire unit. Their intersection, denoted by the shaded area, is meant to represent the behavior that is specific to the PIU.

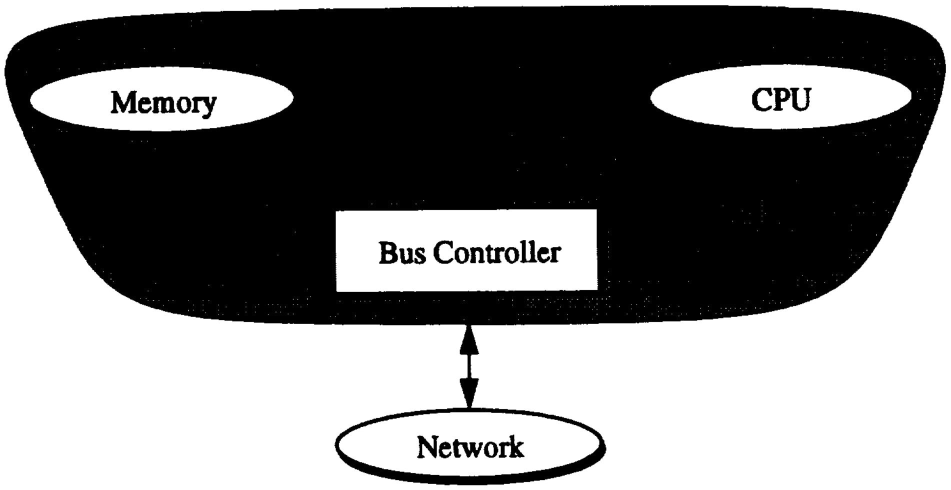


Figure 4.3: View from the Network.

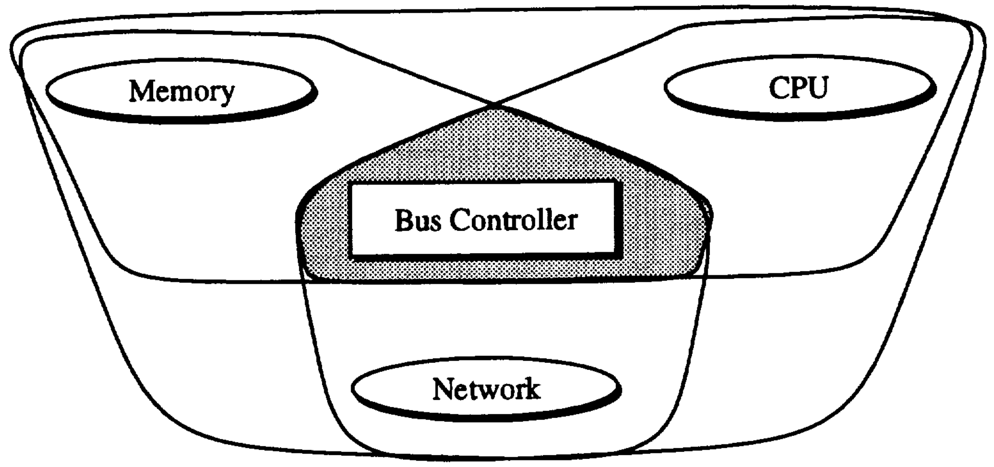


Figure 4.4: Abstraction Views for the PIU.

While we feel that this is a good way to think about the behavior of the PIU in abstract, we are not convinced that it is an appropriate method of specifying the behavior of the PIU. Before such a decision can be made, we will need to do further work. Primarily, we would like to attempt to model the specification of a small device in this way and evaluate the specification for readability and ease of use in verification.

4.3 Representing Transaction Systems

The last section discussed the specification of the abstract interfaces of the PIU, but ignored the details about how those specifications would be written. We talked abstractly about transactions between the PIU and other system components, but the question remains of how to represent those transactions.

One of the difficulties of representing the PIU was touched upon in the last section. If we were only faced with the problem of representing a transaction system such as the PNM (PIU, network, and memory abstraction), the problem would be much simpler. The model would consist of a set of response functions associated with incoming transactions. For each incoming transaction, the response function would update the state of the system and generate an outgoing response based on the current value of the state.

In the model shown in Figure 4.4, the PIU is not a transaction system, but a transaction *translation* system. The PIU cannot generate a response until it issues requests of its own and receives answers to those requests. In addition, there may be state internal to the PIU that needs to be updated and affects the response.

The ultimate goal of the work presented in this report is not to just specify the PIU, but to verify that specification against a lower–level specification. This goal creates several criteria that limit our choice of notation for the behavioral specification:

1. The notation must be capable of specifying concurrent operations of the PIU.
2. The notation must be capable of describing the PIU independent of the other devices to which it might be attached (i.e., the state of those devices should not be a necessary part of the PIU specification.
3. The notation must allow a many–to–one relationship between outputs and inputs.
4. The final specification must be concise and readable. We would like to be able to look at the specification and capture some overall feeling for what it means. Without this level of abstraction, it is very difficult to determine whether the specification is correct or not.
5. The notation must have, or be amenable to building, a collection of theorems about it so that we can reason about the specification and its relationship to the lower–level implementations.

6. The notation must be mechanizable and, since our verification system of choice is HOL, be representable in the HOL logic.

There are a number of candidate notations:

1. We could attempt to represent the transactions in HOL without resorting to any specific notation (i.e., *raw* HOL). We consider the generic interpreter theory (GIT) to be a representation of one kind of computational object in raw HOL. The use of raw HOL to represent transactions implies that we would build a model similar to the GIT, but capturing the abstractions envisioned in the previous section.

   The advantages of this approach are that the model is likely to be tailored to the structure of the PIU more closely than with the other approaches. This means that the meaning of the specification may be clearer. Our experience with the GIT has shown us that abstract models built in HOL can be a fruitful avenue of exploration because they yield a great deal of information to aid in understanding the structure at hand. These models lend a structure to the specification and verification task that is usually not there otherwise; the model states explicitly what definitions must be made to complete the specification and which lemmas need to be proven to complete the verification.

   The disadvantages of using raw HOL are that the model of a transaction system would have to be built and useful theorems about this model would have to be proven. This task is usually more easily done when at least one concrete specification of the type being modeled has been built. This *prototype* specification serves to guide the model development.

2. We could use temporal logic. The primary benefit of temporal logic is that transactions entail describing and reasoning about actions that will occur in the future because of something that occurs now. For example, when the CPU sends a memory read transaction to the PIU, this creates an obligation in the PIU to respond to the request in the future. In between receiving the request and answering it, the PIU would engage in a number of transactions with the network, memory, or both.

   The primary advantage of temporal logic is that there has been much work in the area and it has been successfully used to model hardware devices in other specification efforts.

   The disadvantage is that it is as general as any other general purpose logic and thus, while expressive, would not serve to structure the specification.

3. We could use a well-developed process algebra [Hen88, Hoa85, Mil89a, Mil89b, Mil89c]. Milner [Mil89a] presents a calculus of communicating concurrent processes called CCS; CCS is perhaps the best known process algebra. In process algebras, the specification concentrates on the communication between processes. The specification of the PIU would entail a specification of the events that occur and the events that follow from them.

   There are several advantages to using a process algebra. Process algebras are well understood and there are several popular ones from which to choose. This implies that there are also a great many theories developed and ready for use in a proof effort. To the extent that deduction rules and theorems about the process algebra can be mechanized in HOL, the job of proving properties of the specification will be eased. Indeed, several of the most popular process algebras have been mechanized in HOL and are available for use [Sch91, Cam89, Mel91]. These mechanizations are in various states, so the amount of effort in using one is difficult to predict.

   The disadvantages are similar to those of temporal logics. We fear that the specification will be largely free-form because of the generality of the specification language and thus not structure the problem enough to make the specification and verification methodical.

4. We could use a formal model of a coordination language such as LINDA [But91] to model the actions of the system. In this model, the PIU, CPU, memory, and network are modeled as communicating in a

common area called *tuple space*. Figure 4.5 shows how this would look. In this model, the PIU writes to and reads from tuple space along with the other devices in the system. We can think of tuple space as an abstract model of the bus.

We have given considerable thought to this option. The advantage of this option is that the model is general and seems to be useful for describing ensembles of coordinated processes. The disadvantage is that the model is not yet fully formalized (not to mention mechanized), and thus there would be considerable work before we could begin using the model. Also, we consider this model to be better suited to describing interactions *between* system components (how ever they are specified) rather than specifying the components themselves. Thus, we plan to pursue the formalization of LINDA as a model for *composing* specifications, rather than for the specifications themselves.

Overall, we believe that approach (1) has the most promise and meets the criteria that we outlined above. We do, however, recognize that there is a rich body of research surrounding process algebras and thus will draw on that wherever possible. Indeed, much as the GIT looks similar to a state machine, but has specific features designed to specify and verify microprocessors, our transaction model will look similar to existing process algebras but have features specific to specifying and verifying hardware devices such as the PIU.

4.4 Preliminary Transaction Model Design

This section discusses some preliminary design concepts for the transaction model and gives our development plans.

4.4.1 The Transaction Model

Our preliminary transaction model contains elements common to other behavioral models, augmented by features targeting transaction-level behavior.

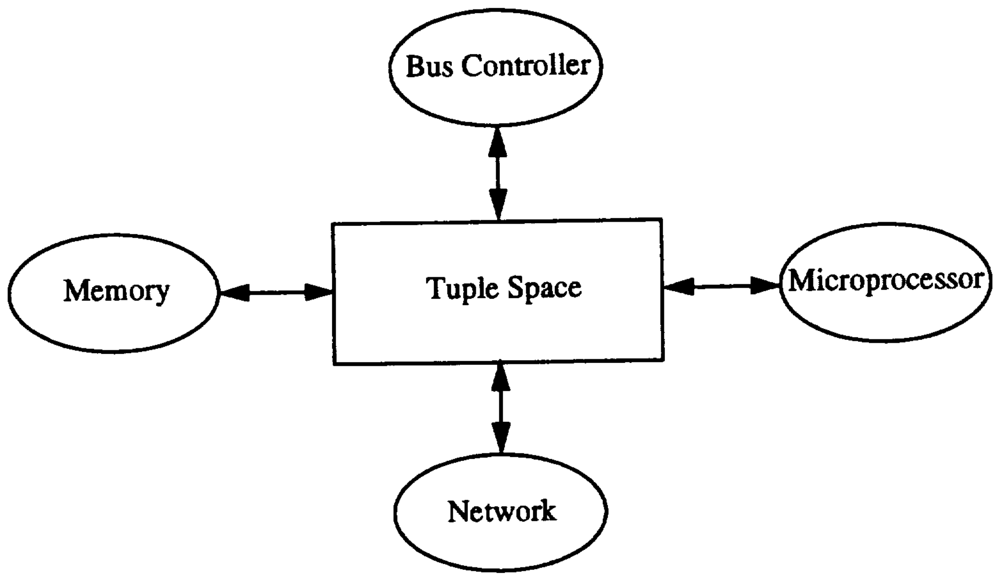


Figure 4.5: Modeling the Buses in a Computer System using Tuple Space.

4.4.1.1 Ports

A transaction system has a number of ports. The system will receive requests on input ports, send requests on output ports and communicate data on data ports. Our model will have an alphabet of port names that can be used to identify ports uniquely.

4.4.1.2 State

The transaction system will have internal state. This state will be represented in a concrete object as a tuple, but in the model will be represented abstractly.

4.4.1.3 Transactions

A transaction will be a triple consisting of an identifying request (taken from an alphabet of possible requests), a state transition function used to update the state, and a set of port–request function pairs representing the requests to be sent and the ports to issue them on in response to the transaction request. The request functions use the current state and values on the data ports to generate a request.

4.4.1.4 Operation

The model will be driven by request events. The model will consist of a set of transactions for each input port. The set represents the legal requests on that port. For each input port, the model will, in parallel, read a request, find the appropriate transaction in its transaction set, and use that transaction to update the state and issue requests on output ports.

4.4.2 Development Plan and Comments

We plan to refine the preliminary concepts outlined above as follows:

1. Build a function program in ML of the behavior of the PIU based on the model present above. The program will allow us to exercise the model and determine where there are problems. We chose ML since it is close to the syntax of HOL and will be readily converted into HOL when we are satisfied with it.
2. The program built in the previous step will be specific to the PIU. Our plan is to generalize that program into an abstract model of transaction systems. We plan to use the results of the experiments in the previous step to guide a formalization of the general model in HOL. Careful design of the abstraction in the program will make this task easier. Provided that the results of the experiments yield favorable results, we do not anticipate formalization to be a large effort.
3. After the model has been formalized, we will need to use it to assess its utility and determine what lemmas need to be proven in the abstract theory to enable effective reasoning in the concrete model. There is no way to determine what these theories will be until the model is used the first time.
4. As the model is used, there will undoubtedly be refinements and extensions. Our experience with the generic interpreter theory has shown that refining and extending abstract theories is not an arduous task and anticipate that the same will be true of the new model.

There are several areas that may lead to difficulties:

- The model specifies each input port separately (in the spirit of the abstract views of Section 4.2). There will have to be coordination between ports due to shared state and output ports. The network port and the CPU port cannot both issue requests of the memory port simultaneously. This, of course, is also a restriction in the design. Our problem is not what coordination to perform, since that exists in the PIU already, but how to represent such coordination in the model. We hope that process algebras will give us some guidance.

- The state is shared and thus may be updated by several ports at once (provided that such updating does not cause interference). We hope that partial specifications of the changes, represented by predicates rather than functions, will solve this problem.
- We have ignored the start–up operation of the PIU in our model. We do not believe that this is a problem since the start–up portion of the chip operates in sequence with the rest of the PIU components. We can model the start–up portion using an interpreter or transaction system (whichever is more appropriate) and choose the behavior of the start–up device or the PIU device depending on the current state.
- The PIU has a number of on–board clocks that serve as interrupt timers. We hope that they can be modeled using the concepts presented in this chapter by looking at the external clock port as another input port with its own set of transactions. One of those transactions will trigger interrupts when the state is correct.

4.5 Conclusions

Hardware devices such as the PIU present a unique challenge for behavioral specification. They differ from interpreters primarily in that there is a large amount of course–grained parallelism and they do not control all the state that they are expected to impact. The overall system (PIU, CPU, network, and memory) could be modeled as an interpreter, but our desire is to model the PIU independently.

One could just make a laundry list of all the actions that occur and use this as the specification, but the result would be nearly unreadable for a complex device such as the PIU. Our goal is to create an abstraction that organizes that behavior so that the specification is readable as well as useful for verification. An unreadable specification is likely to be wrong.

The research presented here is only a start at the top–level specification of the PIU. We plan the following follow–on work:

- The preliminary transaction model must be refined as presented in Section 4.4. The models need to be tested on the PIU design for utility. Furthermore, the model needs to be formalized in HOL.
- Further work must be done on the composition of our abstract–view approach to behavior. We plan a further review of the literature for applicable work and a small test study involving a small device with a simple semantics, but more than one interface, to determine whether composing the abstract behaviors of the interface is sufficient to represent behavior.
- We intend to pursue the formalization of the LINDA coordination language since it seems a likely candidate model for composing the specification of the PIU with the specifications of the CPU, memory, and network. This composition would be used to implement a more abstract view of the system. This work does not have consequences for the top–level specification of the PIU itself but may be important for future compositions.

5 Towards an Integrated Simulation/Verification Environment

This section describes work that links the M hardware description language and the HOL theorem proving system.

The M hardware description language is part of a simulation and synthesis system from Mentor Graphics Corporation. M is a superset of C with extensions for efficiently describing hardware.

The goal of the work presented in this section was to develop a prototype translator for converting M descriptions to the equivalent HOL descriptions. We chose to describe the implementation of the PIU in M for several reasons:

- Engineers working on the project are more comfortable with M descriptions than they are with the logic of HOL. This is probably because of the similarity of M to imperative programming languages in which most engineers are schooled.
- M descriptions can be executed. This allows the specifications to be animated, providing a form of simulation. Engineers cán observe the operation of the specification in an effort to judge its correctness.

The translator described here is a *prototype* tool. We have used the AWK programming language [Aho88] to construct a parser for the subset of M actually used in the description of the PIU. In addition to parsing M, the tool generates HOL statements corresponding to the input. The generation is done on an *ad hoc* basis—no attempt has been made to describe the semantics of M formally.

The translator between M and HOL is important because a hand translation would be tedious and error prone. Using a machine translation, even one done informally, provides *consistent* translations. When an error in a translation is found, the translator can be corrected and the other translations redone to ensure that the error does not affect other specifications as well.

Future work may include a more formal translator between M and HOL if we determine that M descriptions are useful. The more formal translator would include a parser built into the HOL theorem prover as well as a formal semantic description. The translation would be done completely within the theorem prover for added assurance.

The following section will discuss data types developed for use with the model. We will not discuss the actual translation process in detail, but we will give a simple example of an M description of a finite state machine and its equivalent form in HOL as produced by the M-to-HOL translator. The HOL definitions are intended to be used with the generic interpreter model described in Section 2 of this report.

5.1 New Datatypes in HOL

In order to translate M to HOL, we had to make type definitions in HOL that correspond to the types used in the M language. Two of the more involved type definitions were arrays and n–bit words.

5.1.1 Arrays

Since M is a superset of C, M descriptions make heavy use of arrays. HOL does not have a built–in array type, but arrays are easy to model in higher–order logic using functions. In general we treat an array of objects as a function from the natural numbers to the same objects. There are four basic operations on arrays in M that needed to be defined in HOL: array indexing, array assignment, array subsetting, and subarray assignment.

Array Indexing. In M, arrays are indexed using bracket notation. In HOL, since arrays are just functions, arrays are indexed by function application. Thus, the M term *x[i]* is written in HOL as *(x i)*.

Array Assignment. In M, one can use an indexed array variable as the *lvalue* in an assignment statement. Logic does not have assignment, so the corresponding definition is functional. We define a function called *ALTER* that operates on an array, an index, and a value and returns a new array with the value stored in the array at the index given. All other values are unchanged. Thus, the M term *x[i] = y* is written *(ALTER x (i) y)* in HOL.

Array Subsetting. In M, one can use a subarray in an expression. The HOL function *SUBARRAY* serves the same purpose. Thus, the M term *x[15:5]* (which represents an 11–element array with location 0 holding the same value as *x[5]*, location 1 holding the same value as *x[6]*, and so on) would be written in HOL as *SUBARRAY x (15,5)*.

Subarray Assignment. In M, one can assign arrays to portions of an existing array. The HOL function that does this is called *MALTER*. The M term *x[15:5] = y*, would be written in HOL as *MALTER x (15,5) y*.

The theory of arrays also contains theorems pertaining to these definitions that aid in reasoning about arrays.

5.1.2 N–Bit Words

N–bit words are defined in M using arrays of booleans. Since we represent arrays as functions, the natural representation for n–bit words is a function from the natural numbers to the booleans. The theory of n–bit words that we defined uses this representation and makes definitions that allow the representation to be usable. There are four kinds of definitions in the n–bit word theory:

1. Definitions that *interpret* the meaning of an n–bit word.
2. Definitions that *create* n–bit words with special meanings and give them a name.
3. Definitions that *test* an n–bit word for a given property.
4. Definitions that *operate* on n–bit words.

There are two major functions for interpreting n–bit words: *VAL* and *WORDN*. *VAL* returns the numeric value of an n–bit word. *WORDN* returns the n–bit word representing a given number.

There are a number of functions for creating special n–bit words. We will not discuss all of them here, but only give a few examples. *SETN* returns an n–bit word with all of its bits set. Similarly, *RSTN* returns an n–bit word with all of its bits false.

Examples of test predicates include *ONES* which tests if all the bits in a word are true and *ZEROS* which tests if all the bits in a word are false.

Operations on n–bit words implement common boolean and arithmetic operations on n–bit words. For example, *NOTN* returns the n–bit complement of a word. *INCN* returns the n–bit word resulting from adding 1 (modulo n) to its argument.

So far, the theory does not contain many theorems regarding these definitions and their relationship to one another. These theorems will be proven as necessary.

5.2 An Example in M

The following example shows how a finite state machine is described in M. For brevity, the description contains only one state, *S1*; a more realistic description would contain more states, as well as more logic variables. The example does illustrate some of the features of M that required translation such as logic operations, array subranging, and the mixture of output and logical statements in the same context.

```
/************************************************************
   Module:   test.M
   Authors:  David Fura / Phillip Windley
   Date:     13MAR92

   Example of M description for translation.
************************************************************/
#define V1  1
#define V2  2
MODULE test () {
   /*       State variables:*/
   MEMORY   LOGIC  new_A, A;
   MEMORY   LOGIC  new_B, B;
   MEMORY   LOGIC  new_C[32], C[32];

   /*       Output variables:*/
   OUT      I_X[32];

   /*       Input variables:*/
   IN       Clock;
   IN       Rst;

   INITIALIZE {}

   SIMULATE {
       switch (Decode (Clock)) {
            case S1:
            new_A = (C == V1) || (C != V2);
            new_B = (C == V1) && new_A;
            new_C = wr(C,1);

            I_X[31] = new_A
              ? Clock
              : Rst;
            I_X[30:29] = new_C[1:0];
            I_X[28:0] = new_B
              ? new_C[28:0]
              : I_X[28:0];
            break;
            default:
            PRINT ("\nILLEGAL");
            break;
       }
   }
}
```

5.3 An Example in HOL

The following code represents the translation of the M code in the last section into HOL by the prototype translator developed for this project. No substantive changes have been made to the text. Except for indentation and spacing, everything is just as the translator produced it.

```
let V1 = "1";;
let V2 = "2";;

let test_state = ((A, B, C): bool # bool # wordn);;
let test_inputs = ((Rst, Clock): bool # bool);;
let test_outputs = ((I_X): wordn);;

let S1_inst_def = new_definition
    ('S1_inst',
    "S1_inst ^test_state ^test_inputs =
        let new_A = (C = (WORDN ^V1)) \/ (~(C = (WORDN ^V2))) in
        let new_B = (C = (WORDN ^V1)) /\ new_A in
        let new_C = wr(C, (WORDN 1)) in
     (new_A, new_B, new_C)"
    );;

let S1_out_def = new_definition
    ('S1_out',
    "S1_out ^test_state ^test_inputs =
        let new_A = (C = (WORDN ^V1)) \/ (~(C = (WORDN ^V2))) in
        let new_B = (C = (WORDN ^V1)) /\ new_A in
        let new_C = wr(C, (WORDN 1)) in
        let I_X_31_31 = new_A
          =>  Clock
          |   Rst in
        let I_X_30_29 = (SUBARRAY new_C (1,0)) in
        let I_X_28_0 = new_B
          => (SUBARRAY new_C (28,0))
          |  (SUBARRAY I_X (28,0)) in
        let I_X = (MALTER
                     (MALTER
                      (MALTER I_X (31,31) I_X_31_31)
                     (30,29) I_X_30_29)
                    (28,0) I_X_28_0) in
     (I_X)"
    );;
```

The translator does a good job of translating most M programs into HOL. The largest limitation on its use is the simple type analysis that is done. A more thorough type analysis would catch some of the infrequent errors, but would have made the translator much more complicated. If a translator based on formal semantics is constructed, we will overcome this limitation.

6 Conclusions

We have completed the design specification for a processor interface unit (PIU) and identified the modeling approach to be used for the requirements specification. Along the way we have made progress in integrating our hardware design and verification environments into a single unified framework.

In performing this task a number of important conclusions have been reached concerning the state-of-the-art in formal specification, using HOL, with respect to the demands of real-world hardware systems.

The generic interpreter theory, described in Section 2, was shown to work well in a real-world hardware application. It is clear that this theory, which was initially funded by NASA in a previous task [Win90], fits applications well beyond the domain of microprocessors for which it was originally used. Our introduction of outputs into the theory accommodates the composition of subsystems modeled as interpreters, and enhances the theory's applicability to future system modeling problems.

Developing the lower five levels of the PIU specification hierarchy, described in Section 3, stretched existing specification tools and techniques to their limit. To illustrate the size of this modeling problem, the five phase-level specifications together required equations for 280 state variables and 60 output variables. The PIU clock-level model caused overflows in three different stacks in the original Lisp implementation used to build the HOL system.

Because of delays in the PIU design schedule, this task began while the design was still undergoing considerable change. Due to the multiple specification levels and the lack of any significant automation, modifying our models to reflect these changes required much more effort than that required by the design team, for example. As a result, the total effort required to complete the design specification was far greater than necessary. Although previous formal specification and verification efforts appear to have begun only *after* the design was finalized, and therefore didn't face this problem, formal methods will be most useful when they can be applied *before* a chip is initially fabricated, and thus before the design is finished as well. Based on this experience it is clear that major improvements are needed in the tools used to develop future design specifications.

Perhaps our most significant discovery is that current hardware specification approaches, although suitable for the lower levels of the PIU specification hierarchy, are inadequate for the topmost level. This motivated us to investigate the alternative modeling techniques described in Section 4, from which we have defined a preliminary model for use in formalizing a new transaction-based modeling level.

Although not explicitly part of this task's description, we have made progress in integrating our hardware design and verification environments to support this and future work. The M-to-HOL translator, described in Section 5, performs a nearly-complete translation of suitably-formatted M-language models into HOL. The utility of this tool was demonstrated by our translation of all the port-level behavioral models from their definitions in M. Although this translation is not based on a formal semantics for M, it provides a consistent translation capability that is available for use *now*. It should have an immediate impact on productivity for the next chip specification.

The work presented in this report has made a significant contribution to the specification and verification of real–world devices, but much remains to be done. In particular, this report has outlined the following tasks:

1. Before work on the specification of the top level can be completed, the formal model of the transaction level must be completed. Section 4 gives a more detailed plan for completing this work.

2. The specification hierarchy was outlined in Section 3, but this task did not include the completion of the specification. In particular, the PIU top–level specification remains to be written.

In addition to the work that must be completed to finish the specification, there are a number of open questions that have a direct bearing on how this work is used:

1. The proofs of correspondence between levels in the specification hierarchy should be completed. The specification process itself is useful because it gives designers an abstract view of the device and aids understanding. The detailed examination entailed in the specification is useful for finding errors. However, the primary benefit of a formal specification is that it is amenable to analysis.
2. If we intend to use the top–level specification along with specifications of other devices in the PMM, such as the CPU and memory, to write a specification of the PMM, a model of composition must be developed. Section 4 recommended a formalization of LINDA as that model, but no work has been done to explore the feasibility or utility of this method.
3. The translation between M and HOL is being done in a prototype system written in AWK. A more formal approach, with more confidence in its correctness, would be to embed M in HOL. This would involve defining the syntax of M (or a reasonable subset) in HOL and then defining a formal semantics of M for use in the translation. Because the translation would be done by the verification system itself, we could have increased confidence that the HOL model corresponded to the M model.

7 References

[Aho88] A.V. Aho, B.W. Kerninghan, P.J. Weinberger, *The AWK Programming Language*, Addison-Wesley, 1988.

[Aro90] Tejkumar Arora, *The formal verification of the VIPER microprocessor: EBM to microcode level*, Master's thesis, University of California, Davis, 1990.

[But91] P. Butcher, "A Behavioral Semantics for Linda–2," *Software Engineering Journal*, July 1991.

[Cam89] A. J. Camilleri, "Mechanizing CSP Trace Theory in Higher–Order Logic," Hewlett–Packard Laboratories, *Technical Memorandum HPL–ISC–TM–89–131*, August 1989.

[Coh88] Avra Cohn, "Correctness properties of the VIPER block model: The second level," University of Cambridge Computer Laboratory, *Technical Report 134*, May 1988.

[SRI88] SRI International Computer Science Laboratory, *EHDM Specification and Verification System: User's Guide, Version 4.1*, 1988.

[Gor86] M. Gordon, "Why Higher–Order Logic is a good Formalism for Specifying and Verifying Hardware," in G.J. Milne and P.A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, North-Holland, 1986.

[Gor88] Michael J.C. Gordon, "HOL: A proof generating system for higher-order logic," in G. Birtwistle and P.A Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, Kluwer Academic Publishers, 1988.

[Gog88] J. Goguen and T. Winkler, "Introducing OBJ3," SRI International, *Technical Report SRI-CSL-88-9*, August 1988.

[Hen88] M. Hennessy, *Algebraic Theory of Processes*, MIT Press, 1988.

[Her88] John Herbert, "Temporal abstraction of digital designs," in G.J. Milne, editor, *The Fusion of Hardware Design and Verification, Proceedings of the IFIP WG 10.2 International Working Conference*, Glasgow, Scotland, North–Holland, 1988.

[Hoa85] C. A. R. Hoare, "Communicating Sequential Processes," Prentice Hall, 1985.

[Hun87] Warren A. Hunt, Jr., "The mechanical verification of a microprocessor design," in D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*, Elsevier Scientific Publishers, 1987.

[Hun92] Warren A. Hunt, Jr., and Bishop Brock, "A Formal HDL and its use in the FM9001 Verification," in C.A.R. Hoare and M.J.C. Gordon, editors, *Mechanized Reasoning and Hardware Design*, Prentice Hall, 1992.

[Joy89] Jeffrey J. Joyce, *Multi–Level Verification of Microprocessor–Based Systems*, PhD thesis, University of Cambridge, December 1989.

[Koh78] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 1978.

[Low89] Paul Loewenstein, "Reasoning about state machines in higher–order logic," in M. Leeser and G. Brown, editors, *Workshop on Hardware Specification, Verification, and Synthesis: Mathematical Aspects*, Lecture Notes in Computer Science, Springer-Verlag, 1989.

[Mel88] Thomas Melham, "Abstraction mechanisms for hardware verification," in G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, Kluwer Academic Publishers, 1988.

[Mel90] T.F. Melham, "Formalizing Abstraction Mechanisms for Hardware Verification in Higher Order Logic," University of Cambridge Computer Laboratory, *Technical Report 201*, August 1990.

[[Mel91] T. F. Melham, "A Mechanized Theory of the π–Calculus in HOL," in G. Huet, G. Plotkin, and C. Jones, editors, *Second Annual Workshop on Logical Frameworks*, Edinburgh, May 1991.

[Mil89a] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[Mil89b] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes, Part I," University of Edinburgh, Laboratory for Foundations of Computer Science, *Technical Report ECS–LFCS–89–85*, June 1989.

[Mil89c] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes, Part II," University of Edinburgh, Laboratory for Foundations of Computer Science, *Technical Report ECS–LFCS–89–86*, June 1989.

[Sch91] E. T. Schubert, K. Levitt, G.C. Cohen,. "Towards Composition of Verified Hardware Devices," *NASA Contractor Report 187504*, November 1991.

[Win88] Phillip J. Windley, "A hierarchical methodology for the verification of microprogrammed microprocessors," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 1990.

[Win90] Phillip J. Windley, *The Formal Verification of Generic Interpreters*, PhD thesis, University of California, Davis, Division of Computer Science, June 1990.

[Win90a] Phillip J. Windley, "A poor man's implementation of abstract theories," University of California, Davis, Division of Computer Science, " *Technical Report CSE-90-06*, 1990.

[Win91] Phillip J. Windley, "The formal specification of a high–speed CMOS correlator," in *Proceedings of the Third Annual IEEE/NASA Symposium on VLSI Design*, October 1991.

Appendix A ML Source for Component Specifications.

This appendix contains the HOL models for components used in the gate-level specification for the PIU ports, as well as auxiliary definitions for n-bit words implemented as arrays and array accessing functions.

```
%--------------------------------------------------------------------------------------------------------------

   File:          gates_def.ml

   Author:        (c) D.A. Fura 1992

   Date:          31 March 1992

   This file contains the ml source for the combinational logic gates used in the gate-level description of the
   FTEP PIU, an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.

--------------------------------------------------------------------------------------------------------------%

system 'rm gates_def.th';;

new_theory 'gates_def';;

map new_parent ['aux_def'];;

let NOT_SPEC = new_definition
   ('NOT_SPEC',
    "! a z .
     NOT_SPEC a z =
        (! t:time . z t = ~a t)"
   );;

let AND2_SPEC = new_definition
   ('AND2_SPEC',
    "! a b z .
     AND2_SPEC a b z =
        (! t:time . z t = a t /\ b t)"
   );;

let AND3_SPEC = new_definition
   ('AND3_SPEC',
    "! a b c z .
     AND3_SPEC a b c z =
        (! t:time . z t = a t /\ b t /\ c t)"
   );;

let OR2_SPEC = new_definition
   ('OR2_SPEC',
    "! a b z .
     OR2_SPEC a b z =
        (! t:time . z t = a t \/ b t)"
   );;

let OR3_SPEC = new_definition
```

```
    ('OR3_SPEC',
    "! a b c z .
    OR3_SPEC a b c z =
       (! t:time . z t = a t \/ b t \/ c t)"
    );;

let NAND2_SPEC = new_definition
    ('NAND2_SPEC',
    "! a b z .
    NAND2_SPEC a b z =
       (! t:time . z t = ~(a t /\ b t))"
    );;

let NAND3_SPEC = new_definition
    ('NAND3_SPEC',
    "! a b c z .
    NAND3_SPEC a b c z =
       (! t:time . z t = ~(a t /\ b t /\ c t))"
    );;

let BUF_SPEC = new_definition
    ('BUF_SPEC',
    "! (a:time->*) z .
    BUF_SPEC a z =
       (! t:time . z t = a t)"
    );;

let TRIBUF_SPEC = new_definition
    ('TRIBUF_SPEC',
    "! (a:time->*) e z .
    TRIBUF_SPEC a e z =
       (! t:time . (e t) ==> (z t = a t))"
    );;

close_theory();;

%--------------------------------------------------------------------------------------------------------

    File:        latches_def.ml

    Author:      (c) D.A. Fura 1992

    Date:        31 March 1992

    This file contains the ml source for the latches used in the gate-level specification of the FTEP PIU, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center.

--------------------------------------------------------------------------------------------------------%

system 'rm latches_def.th';;

new_theory 'latches_def';;

map new_parent ['aux_def'];;
```

```
%---------------------------------------------------------------------------------------------------------------
    One-bit D-latch, no set, no reset, no enable.
---------------------------------------------------------------------------------------------------------------%

let DLAT_SPEC = new_definition
    ('DLAT_SPEC',
     "! (din:time->bool) clk state qout .
      DLAT_SPEC din clk state qout =
         ! t:time .
             (state (t+1) = (clk t) => din t | state t) ∧
             (qout t = state (t+1))"
    );;

%---------------------------------------------------------------------------------------------------------------
    One-bit D-latch, with set, no reset, no enable.
---------------------------------------------------------------------------------------------------------------%

let DSLAT_SPEC = new_definition
    ('DSLAT_SPEC',
     "! (din:time->bool) set clk state qout .
      DSLAT_SPEC din set clk state qout =
         ! t:time .
             (state (t+1) = (clk t) => ((set t) => T | din t) | state t) ∧
             (qout t = state (t+1))"
    );;

%---------------------------------------------------------------------------------------------------------------
    One-bit D-latch, no set, with reset, no enable.
---------------------------------------------------------------------------------------------------------------%

let DRLAT_SPEC = new_definition
    ('DRLAT_SPEC',
     "! (din:time->bool) rst clk state qout .
      DRLAT_SPEC din rst clk state qout =
         ! t:time .
             (state (t+1) = (clk t) => ((rst t) => F | din t) | state t) ∧
             (qout t = state (t+1))"
    );;

%---------------------------------------------------------------------------------------------------------------
    One-bit D-latch, with set, with reset, no enable.
---------------------------------------------------------------------------------------------------------------%

let DSRLAT_SPEC = new_definition
    ('DSRLAT_SPEC',
     "! (din:time->bool) set rst clk state qout .
      DSRLAT_SPEC din set rst clk state qout =
         ! t:time .
             (state (t+1) = (clk t) => ((set t ∧ ~rst t) => T |
                                        (~set t ∧ rst t) => F |
                                        (~set t ∧ ~rst t) => din t |
                                        ARB) |
                              state t) ∧
```

```
            (qout t = state (t+1))"
    );;

%------------------------------------------------------------------------------------------------
    One-bit D-latch, no set, no reset, with enable.
------------------------------------------------------------------------------------------------%

let DELAT_SPEC = new_definition
    ('DELAT_SPEC',
     "! (din:time->bool) en clk state qout .
      DELAT_SPEC din en clk state qout =
         ! t:time .
            (state (t+1) = (clk t Λ en t) => din t | state t) Λ
            (qout t = state (t+1))"
    );;

%------------------------------------------------------------------------------------------------
    One-bit D-latch, no set, with reset, with enable.
------------------------------------------------------------------------------------------------%

let DRELAT_SPEC = new_definition
    ('DRELAT_SPEC',
     "! (din:time->bool) rst en clk state qout .
      DRELAT_SPEC din rst en clk state qout =
         ! t:time .
            (state (t+1) = (clk t Λ en t) => ((rst t) => F | din t) | state t) Λ
            (qout t = state (t+1))"
    );;

%------------------------------------------------------------------------------------------------
    One-bit D-latch, with set, no reset, with enable.
------------------------------------------------------------------------------------------------%

let DSELAT_SPEC = new_definition
    ('DSELAT_SPEC',
     "! (din:time->bool) set en clk state qout .
      DSELAT_SPEC din set en clk state qout =
         ! t:time .
            (state (t+1) = (clk t Λ en t) => ((set t) => T | din t) | state t) Λ
            (qout t = state (t+1))"
    );;

%------------------------------------------------------------------------------------------------
    One-bit D-latch, with set, with reset, with enable.
------------------------------------------------------------------------------------------------%

let DSRELAT_SPEC = new_definition
    ('DSRELAT_SPEC',
     "! (din:time->bool) set rst en clk state qout .
      DSRELAT_SPEC din set rst en clk state qout =
         ! t:time .
            (state (t+1) = (clk t Λ en t) => (( set t Λ ~rst t) => T |
                                              (~set t Λ rst t) => F |
                                              (~set t Λ ~rst t) => din t |
                                        ARB) |
                              state t) /\
           (qout t = state (t+1))"
   );;

%--------------------------------------------------------------------------------------------------------------------
   Multiple-bit D-latch, no set, no reset, no enable.
--------------------------------------------------------------------------------------------------------------------%

let DLATn_SPEC = new_definition
   ('DLATn_SPEC',
    "! (din:time->wordn) clk state qout .
    DLATn_SPEC din clk state qout =
       ! t:time .
           (state (t+1) = (clk t) => din t | state t) /\
           (qout t = state (t+1))"
   );;

close_theory();;

%--------------------------------------------------------------------------------------------------------------------

   File:          ffs_def.ml

   Author:        (c) D.A. Fura 1992

   Date:          31 March 1992

   This file contains the ml source for the flip-flops used in the gate-level specification of the FTEP PIU, an ASIC
   developed by the Embedded Processing Laboratory, Boeing High Technology Center.

--------------------------------------------------------------------------------------------------------------------%

system 'rm ffs_def.th';;

new_theory 'ffs_def';;

map new_parent ['aux_def'];;

%--------------------------------------------------------------------------------------------------------------------
   One-bit flip-flop, no set, no reset, no enable.
--------------------------------------------------------------------------------------------------------------------%

let DFF_SPEC = new_definition
   ('DFF_SPEC',
    "! (din:time->bool) clk state0 state1 qout .
    DFF_SPEC din clk state0 state1 qout =
       (! t:time . (state0 (t+1) = (~clk t) => din t | state0 t) /\
                   (state1 (t+1) = (clk t) => state0 t | state1 t) /\
                   (qout t = state1 (t+1)))"
   );;

%--------------------------------------------------------------------------------------------------------------------
   One-bit flip-flop, no set, with reset, no enable.
```

```
--------------------------------------------------------------------------------------------------------%

let DRFF_SPEC = new_definition
    (`DRFF_SPEC`,
     "! (din:time->bool) rst clk state0 state1 qout .
     DRFF_SPEC din rst clk state0 state1 qout =
        (! t:time . (state0 (t+1) = (~clk t) => (rst t => F | din t) | state0 t) ∧
                    (state1 (t+1) = (clk t) => state0 t | state1 t) ∧
                    (qout t = state1 (t+1)))"
    );;

%--------------------------------------------------------------------------------------------------------
    One-bit flip-flop, with set, no reset, no enable.
--------------------------------------------------------------------------------------------------------%

let DSFF_SPEC = new_definition
    (`DSFF_SPEC`,
     "! (din:time->bool) set clk state0 state1 qout .
     DSFF_SPEC din set clk state0 state1 qout =
        (! t:time . (state0 (t+1) = (~clk t) => (set t => T | din t) | state0 t) ∧
                    (state1 (t+1) = (clk t) => state0 t | state1 t) ∧
                    (qout t = state1 (t+1)))"
    );;

%--------------------------------------------------------------------------------------------------------
    One-bit flip-flop, with set, with reset, no enable.
--------------------------------------------------------------------------------------------------------%

let DRSFF_SPEC = new_definition
    (`DRSFF_SPEC`,
     "! (din:time->bool) rst set clk state0 state1 qout .
     DRSFF_SPEC din rst set clk state0 state1 qout =
        (! t:time . ((~clk t ∧ set t ∧ ~rst t) ==> state0 (t+1) = T) ∧
                    ((~clk t ∧ ~set t ∧ rst t) ==> state0 (t+1) = F) ∧
                    ((clk t ∨ ~set t ∧ ~rst t) ==> state0 (t+1) = state0 t) ∧
                    (state1 (t+1) = (clk t) => state0 t | state1 t) ∧
                    (qout t = state1 (t+1)))"
    );;

%--------------------------------------------------------------------------------------------------------
    One-bit flip-flop, no set, no reset, with enable.
--------------------------------------------------------------------------------------------------------%

let DEFF_SPEC = new_definition
    (`DEFF_SPEC`,
     "! (din:time->bool) en clk state0 state1 qout .
     DEFF_SPEC din en clk state0 state1 qout =
        (! t:time . (state0 (t+1) = (~clk t) => din t | state0 t) ∧
                    (state1 (t+1) = (clk t ∧ en t) => state0 t | state1 t) ∧
                    (qout t = state1 (t+1)))"
    );;

%--------------------------------------------------------------------------------------------------------
    Multiple-bit flip-flop, no set, no reset, with enable.
```

```
------------------------------------------------------------------------------------------------------------------%

let DEFFn_SPEC = new_definition
    ('DEFFn_SPEC',
     "! (din:time->wordn) en clk state0 state1 qout .
      DEFFn_SPEC din en clk state0 state1 qout =
         (! t:time . (state0 (t+1) = (~clk t) => din t | state0 t) Λ
                     (state1 (t+1) = (clk t Λ en t) => state0 t | state1 t) Λ
                     (qout t = state1 (t+1)))"
    );;


%------------------------------------------------------------------------------------------------------------------
    One-bit flip-flop, no set, with reset, with enable.
------------------------------------------------------------------------------------------------------------------%

let DREFF_SPEC = new_definition
    ('DREFF_SPEC',
     "! (din:time->bool) en rst clk state0 state1 qout .
      DREFF_SPEC din en rst clk state0 state1 qout =
         (! t:time . (state0 (t+1) = (~clk t) => (rst t => F | din t) | state0 t) Λ
                     (state1 (t+1) = (clk t Λ en t) => state0 t | state1 t) Λ
                     (qout t = state1 (t+1)))"
    );;

%------------------------------------------------------------------------------------------------------------------
    One-bit flip-flop, with set, no reset, with enable.
------------------------------------------------------------------------------------------------------------------%

let DSEFF_SPEC = new_definition
    ('DSEFF_SPEC',
     "! (din:time->bool) en set clk state0 state1 qout .
      DSEFF_SPEC din en set clk state0 state1 qout =
         (! t:time . (state0 (t+1) = (~clk t) => (set t => T | din t) | state0 t) Λ
                     (state1 (t+1) = (clk t Λ en t) => state0 t | state1 t) Λ
                     (qout t = state1 (t+1)))"
    );;

%------------------------------------------------------------------------------------------------------------------
    One-bit flip-flop, with set, with reset, with enable.
------------------------------------------------------------------------------------------------------------------%

let DRSEFF_SPEC = new_definition
    ('DRSEFF_SPEC',
     "! (din:time->bool) en rst set clk state0 state1 qout .
      DRSEFF_SPEC din en rst set clk state0 state1 qout =
         (! t:time . ((~clk t Λ set t Λ ~rst t) ==> state0 (t+1) = T) Λ
                     ((~clk t Λ ~set t Λ rst t) ==> state0 (t+1) = F) Λ
                     ((clk t V ~set t Λ ~rst t) ==> state0 (t+1) = state0 t) Λ
                     (state1 (t+1) = (clk t Λ en t) => state0 t | state1 t) Λ
                     (qout t = state1 (t+1)))"
    );;

close_theory();;
```

```
%------------------------------------------------------------------------------------------------------

    File:           counters_def.ml

    Author:         (c) D.A. Fura 1992

    Date:           31 March 1992

    This file contains the ml source for the counters used in the gate-level specification of the FTEP PIU, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center.

------------------------------------------------------------------------------------------------------%

system `rm counters_def.th`;;

new_theory `counters_def`;;

map new_parent [`aux_def`;`array_def`;`wordn_def`];;

%------------------------------------------------------------------------------------------------------
    Up-counter, no reset.
------------------------------------------------------------------------------------------------------%

let UPCNT_SPEC = new_definition
    (`UPCNT_SPEC`,
    "! size (din:time->wordn) ld up clk state0 state1 qout zero .
     UPCNT_SPEC size din ld up clk state0 state1 qout zero =
        !t:time .

        (state0 (t+1) = (~clk t) =>
                        ((ld t) => din t |
                        (up t) => INCN size (state1 t) | state1 t) |
                        state0 t) /\
        (state1 (t+1) = (clk t) => state0 t | state1 t) /\
        (qout t = (up t) => INCN size (state1 (t+1)) | state1 (t+1)) /\
        (zero t = (up t) => (INCN size (state1 (t+1)) = WORDN 0) | (state1 (t+1) = WORDN 0))"
    );;

%------------------------------------------------------------------------------------------------------
    Down-counter, no reset.
------------------------------------------------------------------------------------------------------%

let DOWNCNT_SPEC = new_definition
    (`DOWNCNT_SPEC`,
    "! size (din:time->wordn) ld down clk state0 state1 qout zero .
     DOWNCNT_SPEC size din ld down clk state0 state1 qout zero =
        !t:time .

        (state0 (t+1) = (~clk t) =>
                        ((ld t) => din t |
                        (down t) => DECN size (state1 t) | state1 t) |
                        state0 t) /\
        (state1 (t+1) = (clk t) => state0 t | state1 t) /\
        (qout t = (down t) => DECN size (state1 (t+1)) | state1 (t+1)) ∧
        (zero t = (down t) => (DECN size (state1 (t+1)) = WORDN 0) | (state1 (t+1) = WORDN 0))"
    );;

%-------------------------------------------------------------------------------------------
    Up-counter, with reset.
-------------------------------------------------------------------------------------------%

let UPRCNT_SPEC = new_definition
    ('UPRCNT_SPEC',
     "! size (din:time->wordn) ld up rst clk state0 state1 qout zero .
     UPRCNT_SPEC size din ld up rst clk state0 state1 qout zero =
        !t:time .

        (state0 (t+1) = (~clk t) =>
                        ((ld t) => din t |
                        (up t) => INCN size (state1 t) | state1 t) |
                        state0 t) ∧
        (state1 (t+1) = (clk t) =>
                        ((rst t) => WORDN 0 | state0 t) |
                        state1 t) ∧
        (qout t = (up t) => INCN size (state1 (t+1)) | state1 (t+1)) ∧
        (zero t = (up t) => (INCN size (state1 (t+1)) = WORDN 0) | (state1 (t+1) = WORDN 0))"
    );;

%-------------------------------------------------------------------------------------------
    Down-counter, with reset.
-------------------------------------------------------------------------------------------%

let DOWNRCNT_SPEC = new_definition
    ('DOWNRCNT_SPEC',
     "! size (din:time->wordn) ld down rst clk state0 state1 qout zero .
     DOWNRCNT_SPEC size din ld down rst clk state0 state1 qout zero =
        !t:time .

        (state0 (t+1) = (~clk t) =>
                        ((ld t) => din t |
                        (down t) => DECN size (state1 t) | state1 t) |
                        state0 t) ∧
        (state1 (t+1) = (clk t) =>
                        ((rst t) => WORDN 0 | state0 t) |
                        state1 t) ∧
        (qout t = (down t) => DECN size (state1 (t+1)) | state1 (t+1)) ∧
        (zero t = (down t) => (DECN size (state1 (t+1)) = WORDN 0) | (state1 (t+1) = WORDN 0))"
    );;

close_theory();;

%-------------------------------------------------------------------------------------------

    File:        datapaths_def.ml

    Author:      (c) D.A. Fura 1992
    Date:          31 March 1992

    This file contains the ml source for the datapath blocks of the R-Port of the FTEP PIU, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center.

-----------------------------------------------------------------------------------------------%

system 'rm datapaths_def.th';;

new_theory 'datapaths_def';;

map loadf ['abstract'];;

map new_parent ['aux_def';'array_def';'wordn_def'];;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%-----------------------------------------------------------------------------------------------
    Counter block used to build timers.
-----------------------------------------------------------------------------------------------%

let DP_CTR_SPEC = new_definition
 ('DP_CTR_SPEC',
 "! clkA clkB (busB_in:time->wordn) cir_wr c_ld cir_rd ce cin csror_ld cor_rd
    r_ctr_in r_ctr_mux_sel r_ctr_irden r_ctr r_ctr_ce r_ctr_cin r_ctr_cry
    r_ctr_new r_ctr_outA r_ctr_out r_ctr_orden busA_out1 busA_out2 c_out .
  DP_CTR_SPEC clkA clkB busB_in cir_wr c_ld cir_rd ce cin csror_ld cor_rd
                  r_ctr_in r_ctr_mux_sel r_ctr_irden r_ctr r_ctr_ce r_ctr_cin r_ctr_cry
                  r_ctr_new r_ctr_outA r_ctr_out r_ctr_orden busA_out1 busA_out2 c_out =

    !t:time .

    ((clkA t) ==>
        ((r_ctr_in (t+1) = r_ctr_in t) /\
         (r_ctr_mux_sel (t+1) = r_ctr_mux_sel t) /\
         (r_ctr_irden (t+1) = r_ctr_irden t) /\
         (r_ctr (t+1) = (r_ctr_mux_sel t) => r_ctr_in t | r_ctr_new t) /\
         (r_ctr_ce (t+1) = ce t) /\
         (r_ctr_cin (t+1) = cin t) /\
         (r_ctr_cry (t+1) = r_ctr_cry t) /\
         (r_ctr_new (t+1) = r_ctr_new t) /\
         (r_ctr_outA (t+1) = r_ctr_new t) /\
         (r_ctr_out (t+1) = r_ctr_out t) /\
         (r_ctr_orden (t+1) = r_ctr_orden t))) /\
    ((clkB t) ==>
        ((r_ctr_in (t+1) = (cir_wr t) => busB_in t | r_ctr_in t) /\
         (r_ctr_mux_sel (t+1) = c_ld t) /\
         (r_ctr_irden (t+1) = cir_rd t) /\
         (r_ctr (t+1) = r_ctr t) /\
         (r_ctr_ce (t+1) = r_ctr_ce t) /\
         (r_ctr_cin (t+1) = r_ctr_cin t) /\
         (r_ctr_cry (t+1) = (r_ctr_ce t) /\ (r_ctr_cin t) /\ ONES 31 (r_ctr t)) /\
         (r_ctr_new (t+1) = ((r_ctr_ce t) /\ (r_ctr_cin t)) => INCN 31 (r_ctr t) | r_ctr t) /\
         (r_ctr_outA (t+1) = r_ctr_outA t) /\
        (r_ctr_out (t+1) = (csror_ld t) => r_ctr_outA t | r_ctr_out t) ∧
        (r_ctr_orden (t+1) = cor_rd t))) ∧
       ((busA_out1 t = ((r_ctr_irden (t+1)) ∧ (clkA t)) => r_ctr_in (t+1) | ARBN) ∧
        (busA_out2 t = ((r_ctr_orden (t+1)) ∧ (clkA t)) => r_ctr_out (t+1) | ARBN) ∧
        (c_out t = r_ctr_cry (t+1)))"
   );;

%------------------------------------------------------------------------------------------------------------
   Interrupt Control Register (ICR) block.
------------------------------------------------------------------------------------------------------------%

let DP_ICR_SPEC = new_definition
   ('DP_ICR_SPEC',
    "! (rep:^rep_ty) clkA clkB (busA_in:time->wordn) busB_in icr_wr_feedback icr_wr icr_select icr_ld icr_rd
        r_icr_oldA r_icr_old r_icr_mask r_icrA r_icr r_icr_rden
        busA_out icr_out .
    DP_ICR_SPEC rep clkA clkB busA_in busB_in icr_wr_feedback icr_wr icr_select icr_ld icr_rd
                   r_icr_oldA r_icr_old r_icr_mask r_icrA r_icr r_icr_rden
                   busA_out icr_out =

    !t:time .

       ((clkA t) ==>
           (r_icr_oldA (t+1) = busA_in t) ∧
           (r_icr_old (t+1) = r_icr_old t) ∧
           (r_icr_mask (t+1) = r_icr_mask t) ∧
           (r_icrA (t+1) = (icr_select t) => Andn rep (r_icr_old t, r_icr_mask t)
                                            | Orn rep (r_icr_old t, r_icr_mask t)) ∧
           (r_icr (t+1) = r_icr t) ∧
           (r_icr_rden (t+1) = r_icr_rden t)) ∧
       ((clkB t) ==>
           (r_icr_oldA (t+1) = r_icr_oldA t) ∧
           (r_icr_old (t+1) = (icr_wr_feedback t) => r_icr_oldA t | r_icr_old t) ∧
           (r_icr_mask (t+1) = (icr_wr t) => busB_in t | r_icr_mask t) ∧
           (r_icrA (t+1) = r_icrA t) ∧
           (r_icr (t+1) = (icr_ld t) => r_icrA t | r_icr t) ∧
           (r_icr_rden (t+1) = icr_rd t)) ∧
       ((busA_out t = ((r_icr_rden (t+1) ∧ (clkA t)) => r_icr (t+1) | ARBN)) ∧
        (icr_out t = r_icr (t+1)))"
   );;

%------------------------------------------------------------------------------------------------------------
   Control register used to build General Control Register (GCR) and Communication Control Register (CCR).
------------------------------------------------------------------------------------------------------------%

let DP_CR_SPEC = new_definition
   ('DP_CR_SPEC',
    "! clkA clkB (busB_in:time->wordn) cr_wr cr_rd
        r_cr r_cr_rden
        busA_out cr_out .
    DP_CR_SPEC clkA clkB busB_in cr_wr cr_rd
                   r_cr r_cr_rden
                   busA_out cr_out =
    !t:time .
```

```
        ((clkA t) ==>
            (r_cr (t+1) = r_cr t) ∧
            (r_cr_rden (t+1) = r_cr_rden t)) ∧
        ((clkB t) ==>
            (r_cr (t+1) = (cr_wr t) => busB_in t | r_cr t) ∧
            (r_cr_rden (t+1) = cr_rd t)) ∧
        ((busA_out t = ((r_cr_rden (t+1)) ∧ (clkA t)) => r_cr (t+1) | ARBN) ∧
         (cr_out t = r_cr (t+1)))"
    );;

%-----------------------------------------------------------------------------------------------------------
    Status Register Block.
-----------------------------------------------------------------------------------------------------------%

let DP_SR_SPEC = new_definition
    ('DP_SR_SPEC',
    "! clkA clkB (inp:time->wordn) sror_ld sr_rd
        r_sr r_sr_rden
        busA_out .
     DP_SR_SPEC clkA clkB inp sror_ld sr_rd
                    r_sr r_sr_rden
                    busA_out =
     !t:time .
        ((clkA t) ==>
            (r_sr (t+1) = r_sr t) ∧
            (r_sr_rden (t+1) = r_sr_rden t)) ∧
        ((clkB t) ==>
            (r_sr (t+1) = (sror_ld t) => inp t | r_sr t) ∧
            (r_sr_rden (t+1) = sr_rd t)) ∧
        (busA_out t = ((r_sr_rden (t+1)) ∧ (clkA t)) => r_sr (t+1) | ARBN)"
    );;

close_theory();;

%-----------------------------------------------------------------------------------------------------------

    File:       buses_def.ml

    Author:     (c) D.A. Fura 1992

    Date:       31 March 1992

    This file contains the ml source for the buses used in the gate-level specification of the FTEP PIU, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center.

-----------------------------------------------------------------------------------------------------------%


system 'rm buses_def.th';;

new_theory 'buses_def';;

map new_parent ['aux_def'];;
```

```
new_type_abbrev('time', ":num");;

%------------------------------------------------------------------------------------------------
   Specification for a conflict-free bus.
------------------------------------------------------------------------------------------------%

let Bus_CF_12_SPEC = new_definition
   ('Bus_CF_12_SPEC',
   "! inE1 inE2 inE3 inE4 inE5 inE6 inE7 inE8 inE9 inE10 inE11 inE12 .
    Bus_CF_12_SPEC inE1 inE2 inE3 inE4 inE5 inE6 inE7 inE8 inE9 inE10 inE11 inE12 =
    !t:time .

    (inE1 t) => ~((inE2 t) V (inE3 t) V (inE4 t) V (inE5 t) V (inE6 t) V (inE7 t) V (inE8 t) V
                  (inE9 t) V (inE10 t) V (inE11 t) V (inE12 t)) |
    (inE2 t) => ~((inE3 t) V (inE4 t) V (inE5 t) V (inE6 t) V (inE7 t) V (inE8 t) V (inE9 t) V
                  (inE10 t) V (inE11 t) V (inE12 t)) |
    (inE3 t) => ~((inE4 t) V (inE5 t) V (inE6 t) V (inE7 t) V (inE8 t) V (inE9 t) V (inE10 t) V
                  (inE11 t) V (inE12 t)) |
    (inE4 t) => ~((inE5 t) V (inE6 t) V (inE7 t) V (inE8 t) V (inE9 t) V (inE10 t) V (inE11 t) V
                  (inE12 t)) |
    (inE5 t) => ~((inE6 t) V (inE7 t) V (inE8 t) V (inE9 t) V (inE10 t) V (inE11 t) V (inE12 t)) |
    (inE6 t) => ~((inE7 t) V (inE8 t) V (inE9 t) V (inE10 t) V (inE11 t) V (inE12 t)) |
    (inE7 t) => ~((inE8 t) V (inE9 t) V (inE10 t) V (inE11 t) V (inE12 t)) |
    (inE8 t) => ~((inE9 t) V (inE10 t) V (inE11 t) V (inE12 t)) |
    (inE9 t) => ~((inE10 t) V (inE11 t) V (inE12 t)) |
    (inE10 t) => ~((inE11 t) V (inE12 t)) |
    (inE11 t) => ~(inE12 t) | T"
   );;

%------------------------------------------------------------------------------------------------
   Specification for a 12-input bus component.
------------------------------------------------------------------------------------------------%

let Bus_12_1_SPEC = new_definition
   ('Bus_12_1_SPEC',
   "! (inD1:time->*) inD2 inD3 inD4 inD5 inD6 inD7 inD8 inD9 inD10 inD11 inD12
    inE1 inE2 inE3 inE4 inE5 inE6 inE7 inE8 inE9 inE10 inE11 inE12 out .
    Bus_12_1_SPEC inD1 inD2 inD3 inD4 inD5 inD6 inD7 inD8 inD9 inD10 inD11 inD12
                  inE1 inE2 inE3 inE4 inE5 inE6 inE7 inE8 inE9 inE10 inE11 inE12 out =
    !t:time .

    (Bus_CF_12_SPEC inE1 inE2 inE3 inE4 inE5 inE6 inE7 inE8 inE9 inE10 inE11 inE12) ==>
        ((inE1 t ==> (out t = inD1 t)) Λ
         (inE2 t ==> (out t = inD2 t)) Λ
         (inE3 t ==> (out t = inD3 t)) Λ
         (inE4 t ==> (out t = inD4 t)) Λ
         (inE5 t ==> (out t = inD5 t)) Λ
         (inE6 t ==> (out t = inD6 t)) Λ
         (inE7 t ==> (out t = inD7 t)) Λ
         (inE8 t ==> (out t = inD8 t)) Λ
         (inE9 t ==> (out t = inD9 t)) Λ
         (inE10 t ==> (out t = inD10 t)) Λ
         (inE11 t ==> (out t = inD11 t)) Λ
         (inE12 t ==> (out t = inD12 t)))"
    );;

%-----------------------------------------------------------------------------------------------------------
    Specification for a single-input bus component where the input is sourced by an A-clocked latch.
-----------------------------------------------------------------------------------------------------------%

let Bus1A_SPEC = new_definition
    ('Bus1A_SPEC',
     "! (in_A:time->*) out_A out_B .
     Bus1A_SPEC in_A out_A out_B =
     !t:time .

     (out_A t = in_A t) /\
     (out_B t = in_A t)"
    );;

%-----------------------------------------------------------------------------------------------------------
    Specification for a single-input bus component where the input is sourced by a B-clocked latch.
-----------------------------------------------------------------------------------------------------------%

let Bus1B_SPEC = new_definition
    ('Bus1B_SPEC',
     "! (in_B:time->*) out_A out_B .
     Bus1B_SPEC in_B out_A out_B =
     !t:time .

     (out_A t = in_B (t-1)) /\
     (out_B t = in_B t)"
    );;


close_theory();;

%-----------------------------------------------------------------------------------------------------------

    File:           aux_def.ml

    Author:         (c) D.A. Fura 1992

    Date:           31 March 1992

    This file contains auxiliary definitions needed for the gate-level specification of the FTEP PIU, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center.

-----------------------------------------------------------------------------------------------------------%

system 'rm aux_def.th';;

new_theory 'aux_def';;

loadf 'abstract';;

new_type_abbrev('time', ":num");;
new_type_abbrev('wordn', ":(num->bool)");;
```

```
let pfsm_ty_Axiom =
    define_type 'pfsm_ty_Axiom'
        'pfsm_ty = PH | PA | PD | P_ILL';;
let pc_state_ty = ":(wordn#bool#wordn#bool#pfsm_ty#bool#bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool)";;
let pc_env_ty = ":(bool#bool#bool#wordn#bool#bool#wordn#bool#bool#wordn#bool#bool#bool)";;
let pc_out_ty = ":(wordn#bool#wordn#wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool)";;

let cmfsm_ty_Axiom =
    define_type 'cmfsm_ty_Axiom'
        'cmfsm_ty = CMI | CMR | CMA3 | CMA1 | CMA0 | CMA2 | CMD1 | CMD0
                                    | CMW | CMABT';;
let csfsm_ty_Axiom =
    define_type 'csfsm_ty_Axiom'
        'csfsm_ty = CSI | CSL | CSA1 | CSA0 | CSA0W | CSALE | CSRR | CSD1 | CSD0 | CSACK | CSABT';;
let cefsm_ty_Axiom =
    define_type 'cefsm_ty_Axiom'
        'cefsm_ty = CEI | CEE';;
let cc_state_ty = ":(cmfsm_ty#bool#bool#bool#bool#wordn#bool#
                     csfsm_ty#bool#bool#bool#wordn#
                     cefsm_ty#bool#bool#bool#bool#bool#bool#
                     bool#wordn#bool#bool#bool#wordn#bool#
                     bool#bool#bool#bool#bool#bool#bool#
                     bool#bool#bool#wordn#wordn#wordn#wordn#wordn#wordn)";;
let cc_env_ty = ":(wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                    wordn#wordn#wordn#wordn#bool#bool#bool#bool#wordn#wordn#bool#bool#wordn#bool)";;
let cc_out_ty = ":(bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
                   bool#wordn#wordn#wordn#wordn#bool#bool)";;

let mfsm_ty_Axiom =
    define_type 'mfsm_ty_Axiom'
        'mfsm_ty = MI | MA | MW | MRR | MR | MBW | M_ILL';;
let mc_state_ty = ":(mfsm_ty#bool#bool#bool#bool#wordn#bool#bool#wordn#wordn#bool#bool#bool#wordn#wordn)";;
let mc_env_ty = ":(bool#bool#bool#bool#bool#wordn#bool#bool#wordn#bool#wordn#bool#bool)";;
let mc_out_ty = ":(wordn#bool#wordn#wordn#bool#bool#bool#bool#bool)";;

let rfsm_ty_Axiom =
    define_type 'rfsm_ty_Axiom'
        'rfsm_ty = RI | RA | RD';;
let rc_state_ty = ":(rfsm_ty#bool#bool#bool#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#
                    wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#
                    wordn#bool#wordn#bool#wordn#bool#bool#wordn#wordn#bool#wordn#wordn#bool#wordn#bool#wordn#
                    bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn)";;
let rc_env_ty = ":(bool#bool#wordn#bool#bool#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                   wordn#wordn#wordn#bool#bool#wordn)";;
let rc_out_ty = ":(wordn#bool#bool#bool#bool#bool#wordn#wordn#bool#bool)";;

let sfsm_ty_Axiom =
    define_type 'sfsm_ty_Axiom'
        'sfsm_ty = SSTART | SRA | SPF | SC0I | SC0F | ST | SC1I |
                   SC1F | SS | SSTOP | SCS | SN | SO | S_ILL';;
let sc_state_ty = ":(sfsm_ty#bool#bool#bool#bool#bool#bool#wordn#wordn#
                     bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let sc_env_ty = ":(bool#bool#bool#bool#bool#wordn#bool#bool)";;
let sc_out_ty = ":(wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;

let VDD = new_definition
    ('VDD',
    "! t:time . VDD t = T"
);;

let GND = new_definition
    ('GND',
    "! t:time . GND t = F"
);;

let abs_rep = new_abstract_representation [
    ('Andn', ":(wordn#wordn->wordn)");
    ('Orn', ":(wordn#wordn->wordn)");
    ('Ham_Dec', ":(wordn->wordn)");
    ('Ham_Det1', ":(wordn->wordn)");
    ('Ham_Det2', ":(wordn#bool->bool)");
    ('Ham_Enc', ":(wordn->wordn)");
    ('Par_Dec', ":(wordn->wordn)");
    ('Par_Det', ":(wordn->bool)");
    ('Par_Enc', ":(wordn->wordn)");
    ('p_interp', ":(^pc_state_ty#^pc_env_ty#^pc_out_ty->bool)");
    ('c_interp', ":(^cc_state_ty#^cc_env_ty#^cc_out_ty->bool)");
    ('m_interp', ":(^mc_state_ty#^mc_env_ty#^mc_out_ty->bool)");
    ('r_interp', ":(^rc_state_ty#^rc_env_ty#^rc_out_ty->bool)");
    ('s_interp', ":(^sc_state_ty#^sc_env_ty#^sc_out_ty->bool)")];;

make_inst_thms abs_rep;;

let rep_ty = abstract_type 'aux_def' 'Andn';;

close_theory();;

%----------------------------------------------------------------

 File: array_def.ml

 Author: (c) P. J. Windley 1992

 Description:

 Prove auxilliary theorems about functions so that functions
 can be easily used to represent arrays.

 Modification History:

 24FEB92 -- Original file. Many of the theorems included were
 motivated by theorems defined on lists in
 list_aux.ml.

    26FEB92 -- [DAF] Modified order of parameters in calls to
                     ALTER, MALTER, SUBARRAY to match simulation
                     language syntax. Added definition of ELEMENT.
```

```
-----------------------------------------------------------------%

% Removed 26FEB92. [DAF]
loadf 'libs_aux';;

system '/bin/rm array_def.th';;
%

system 'rm array_def.th';;

new_theory 'array_def';;

% Added 26FEB92 (from PJW). [DAF] %

let SYM_RULE =
    (CONV_RULE (ONCE_DEPTH_CONV SYM_CONV))
    ? failwith 'SYM_RULE';;

%-----------------------------------------------------------------
 Auxilliary array definitions and theorems.

 We will use functions to represent arrays. The definition
 that follows defines a ALTER function that can be used to set
 the nth member of an array. The following lemmas are useful
 in reasoning about array operations.
-----------------------------------------------------------------%

let ALTER_DEF = new_definition
    ('ALTER_DEF',
    "ALTER (f:*->**) n x = (\m. (m = n) => x | (f m))"
    );;

let ALTER_THM = prove_thm
    ('ALTER_THM',
    "ALTER (f:*->**) n x y = (y = n) => x | (f y)",
     REWRITE_TAC [ALTER_DEF]
     THEN BETA_TAC
     THEN REFL_TAC
    );;

%-----------------------------------------------------------------
 ALTER_EQUAL is simlar to the EL_SET_EL lemma for lists.
-----------------------------------------------------------------%
let ALTER_EQUAL = prove_thm
    (' ALTER_EQUAL',
    "! x n (f:*->**) . (ALTER f n x) n = x",
     REPEAT GEN_TAC
     THEN REWRITE_TAC [ALTER_DEF]
     THEN BETA_TAC
     THEN REWRITE_TAC []
    );;
```

```
%--------------------------------------------------------------
 ALTER_NON_EQUAL is similar to NOT_EL_SET_EL for lists.
--------------------------------------------------------------%
let ALTER_NON_EQUAL = prove_thm
    'ALTER_NON_EQUAL',
    "! n m (f:*->**) x .
     ~(n = m) ==>
     (f n = (ALTER f m x) n)",
     REPEAT GEN_TAC
     THEN REWRITE_TAC [ALTER_THM]
     THEN STRIP_TAC
     THEN ASM_REWRITE_TAC []
    );;

%--------------------------------------------------------------
 ALTER_COMMUTES is similar to SET_EL_SET_EL for lists.
--------------------------------------------------------------%
let ALTER_COMMUTE = prove_thm
    ('ALTER_COMMUTE',
    "! (d1:*) d2 (f:*->**) (x:**) y .
     ~(d1 = d2) ==>
     ((ALTER (ALTER f d2 x) d1 y) =
     (ALTER (ALTER f d1 y) d2 x))",
     REPEAT GEN_TAC
     THEN CONV_TAC (ONCE_DEPTH_CONV FUN_EQ_CONV)
     THEN REWRITE_TAC [ALTER_THM]
     THEN STRIP_TAC
     THEN GEN_TAC
     THEN REPEAT COND_CASES_TAC
     THEN ASM_REWRITE_TAC []
     THEN UNDISCH_TAC "~((d1:*) = d2)"
     THEN ASSUM_LIST (\thl . REWRITE_TAC (map SYM_RULE thl))
     );;

%--------------------------------------------------------------

 Until now, it hasn't mattered what the type of the subscript is
 and so the previous lemmas were all general, even though
 someone using them to representa arrays, would probably be
 using numbers as subscripts.

 Now, we want to reason about subarrays given as a sequence from
 a starting value to an ending value. This presupposes that the
 subscripts can be totally ordered. To make life easy, we won't
 be that general, but will use numbers as subscripts.

--------------------------------------------------------------%

let SUBARRAY_DEF = new_definition
    ('SUBARRAY_DEF',
    "! n m (f:num->*) .
     SUBARRAY f (m,n) = \x. ((x+n) <= m) => f(x+n) | ARB"
    );;
```

```
let SUBARRAY_THM = prove_thm
    ('SUBARRAY_THM',
    "! n m (f:num->*) .
     SUBARRAY f (m,n) x = ((x+n) <= m) => f(x+n) | ARB",
     REPEAT GEN_TAC
     THEN REWRITE_TAC [SUBARRAY_DEF]
     THEN BETA_TAC
     THEN REFL_TAC
    );;

let ELEMENT_DEF = new_definition
    ('ELEMENT_DEF',
     "! m (f:num->*) .
     ELEMENT f (m) = f m"
    );;

%----------------------------------------------------------------
 MALTER alters multiple values in an array.
----------------------------------------------------------------%

let MALTER_DEF = new_definition
     ('MALTER_DEF',
    "! n m f (g:num->*) .
     MALTER f (m,n) g =
     \x. (n <= x ∧ x <= m) => g (x-n) | f x"
    );;

let MALTER_THM = prove_thm
    ('MALTER_THM',
    "! n m (x:num) g (f:num->*) .
     MALTER f (m,n) g x = (n <= x ∧ x <= m) => g (x-n) | f x",
     REPEAT GEN_TAC
     THEN REWRITE_TAC [MALTER_DEF]
     THEN BETA_TAC
     THEN REFL_TAC
    );;


let MALTER_SUBARRAY_IDENT = prove_thm
    ('MALTER_SUBARRAY_IDENT',
    "!n m (f:num->*) . MALTER f (m,n) (SUBARRAY f (m,n)) = f",
     REPEAT GEN_TAC
     THEN CONV_TAC (ONCE_DEPTH_CONV FUN_EQ_CONV)
     THEN REWRITE_TAC [MALTER_THM;SUBARRAY_THM]
     THEN GEN_TAC
     THEN REPEAT COND_CASES_TAC
     THEN ASM_REWRITE_TAC []
     THEN ASSUM_LIST (\thl . MAP_EVERY ASSUME_TAC
     (flat (map CONJUNCTS (filter (is_conj o concl) thl))))
     THEN IMP_RES_TAC SUB_ADD
     THEN TRY (UNDISCH_TAC "~((n' - n) + n) <= m")
     THEN ASM_REWRITE_TAC []
    );;
```

```
let MALTER_SUBARRAY_SUBSCRIPTS = prove_thm
   (`MALTER_SUBARRAY_SUBSCRIPT`,
   "!n m x (f:num->*) g .
    MALTER f (m,n) (SUBARRAY g (m,n)) x =
    (n <= x /\ x <= m) => g x | f x",
    REPEAT GEN_TAC
    THEN CONV_TAC (ONCE_DEPTH_CONV FUN_EQ_CONV)
    THEN REWRITE_TAC [MALTER_THM;SUBARRAY_THM]
    THEN REPEAT COND_CASES_TAC
    THEN ASM_REWRITE_TAC []
    THEN ASSUM_LIST (\thl . MAP_EVERY ASSUME_TAC
    (flat (map CONJUNCTS (filter (is_conj o concl) thl))))
    THEN IMP_RES_TAC SUB_ADD
    THEN TRY (UNDISCH_TAC "~((x - n) + n) <= m")
    THEN ASM_REWRITE_TAC []
   );;

close_theory();;

%---------------------------------------------------------------

 File: wordn_def.ml

 Description:

 Defines a theory of words which contains a definition for
 converting between functions from numbers to booleans and
 natural numbers and proves various useful theorems about
 this definition. This file is based on a theory that was
 orginally authored by Graham Birtwhistle of the University
 of Calgary in 1988.

 Authors: (c) Graham Birtwhistle, Phillip Windley, 1988, 1992

 Modification History:

    28FEB92 -- [PJW] Original file from words.ml

    10MAR92 -- [PJW] Added definition of WORDN.

    13MAR92 -- [DAF] Added definitions of bv, SETN, RSTN, GNDN,
                     NOTN, INCN, DECN, ARBN.

 ---------------------------------------------------------------%

 % Removed 13MAR92. [DAF]
 let add_root s = `/users/staff/windley/hol/Library/` ^ s;;

 set_search_path(search_path() @
    (map add_root
    [`bits/`;
    `numbers/`;
    `array/`]));;
```

```
%

system '/bin/rm wordn_def.th';;

new_theory 'wordn_def';;

% Replaced 13MAR92. [DAF]
map load_parent [ 'bits'; 'num_thms' ; 'exp' ; 'array_def'];;
%
map new_parent ['aux_def'; 'array_def'];;

new_type_abbrev ('wordn',":num->bool");;

%----------------------------------------------------------------
 Definitions
----------------------------------------------------------------%

let bv = new_definition
    ('bv',
     "! (b:bool) .
     bv b = (b) => 1 | 0"
    );;

let VAL = new_prim_rec_definition
    ('VAL',
    "(VAL 0 (f:wordn) = bv (f 0))
     /\
     (VAL (SUC n) f = ((2 EXP (SUC n)) * (bv (f (SUC n)))) + VAL n f)"
    );;

let pos_val = new_definition
     ('pos_val',
    "! (x:wordn) (y:num) .
     pos_val x y = (bv(x y)) * (2 EXP y)"
    );;

let ONES = new_prim_rec_definition
    ('ONES',
    "(ONES 0 a = (a 0))
     /\
     (ONES (SUC n) a = (a(SUC n)) /\ (ONES n a))
    ");;

let ZEROS = new_prim_rec_definition
    ('ZEROS',
    "(ZEROS 0 a = ~(a 0))
     /\
     (ZEROS (SUC n) a = ~(a(SUC n)) /\ (ZEROS n a))
    ");;

% Modified 13MAR92. [DAF]
let WORDN = new_definition
    ('WORDN',
    "! (x:num) . WORDN x = \n. (x DIV (2 EXP n)) MOD 2"
    );;
%
let WORDN = new_definition
    ('WORDN',
     "! (x:num) . WORDN x = \n. ((x DIV (2 EXP n)) MOD 2 = 1)"
    );;

let SETN = new_definition
    ('SETN',
     "! (x:num) . SETN x = \(n:num). (n <= x) => T | ARB"
    );;

% Equivalent to "WORDN 0" but perhaps more convenient %
let RSTN = new_definition
    ('RSTN',
     "! (x:num) . RSTN x = \(n:num). (n <= x) => F | ARB"
    );;

let GNDN = new_definition
    ('GNDN',
     "! (x:num) (t:time) . GNDN x t = \(n:num). (n <= x) => F | ARB"
    );;

let NOTN = new_definition
    ('NOTN',
     "! (x:num) (f:wordn) . NOTN x f = \(n:num) . (n <= x) => ~(f n) | ARB"
    );;

let INCN = new_definition
    ('INCN',
     "! n f .
     INCN n f = (ONES n f) => RSTN n | WORDN ((VAL n f) + 1)"
    );;

let DECN = new_definition
    ('DECN',
     "! n f .
     DECN n f = (ZEROS n f) => SETN n | WORDN ((VAL n f) - 1)"
    );;

let ARBN = new_definition
    ('ARBN',
     "(ARBN:num->bool) = \n. ARB"
    );;

%----------------------------------------------------------------
 Theorems
----------------------------------------------------------------%

% Removed theorems for now 13MAR92. [DAF]

close_theory();;
```

Appendix B ML Source for the Gate-Level Specification of the PIU Ports.

This appendix contains the HOL models for the gate-level specification for the PIU ports. The ports are listed in the order: P_Port, M_Port, R_Port, C_Port, and SU_Cont.

B.1 P Port Specification

```
%----------------------------------------------------------------------------------------------------------

    File:          p_block.ml

    Author:        (c) D.A. Fura 1992

    Date:          31 March 1992

    This file contains the ml source for the gate-level specification of the PIU P-Port, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center.

----------------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm p_block.th';;

new_theory 'p_block';;

map new_parent ['gates_def';'latches_def';'ffs_def';'counters_def';'aux_def';'array_def';'paux_def'];;

let p_state_ty = ":(pfsm_ty#bool#bool#bool#wordn#wordn#bool#wordn#bool#wordn#num#bool#bool#
                    pfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#num#bool#bool#bool#bool#bool#bool)";;
let p_state = "((P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P_dest1, P_be_,
                P_wr, P_be_n_, P_sizeA, P_loadA, P_downA, P_fsm_state, P_fsm_rst, P_fsm_mrqt, P_fsm_sack,
                P_fsm_cgnt_, P_fsm_crqt_, P_fsm_hold_, P_fsm_lock_, P_rqt, P_size, P_load, P_down, P_lock_,
                P_lock_inh_, P_male_, P_rale_)
                :^p_state_ty)";;

let p_env_ty = ":(bool#bool#bool#wordn#bool#bool#wordn#bool#bool#wordn#bool#bool#bool)";;
let p_env = "((ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_, I_srdy_)
              :^p_env_ty)";;

let p_out_ty = ":(wordn#bool#wordn#wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool)";;
let p_out = "((L_ad_out, L_ready_, I_ad_data_out, I_ad_addr_out, I_be_, I_rale_, I_male_, I_crqt_, I_cale_,
              I_mrdy_, I_last_, I_hlda_, I_lock_)
              :^p_out_ty)";;

%----------------------------------------------------------------------------------------------------------
    P-Port data latches.
----------------------------------------------------------------------------------------------------------%

let Data_Latches_SPEC = new_definition
    ('Data_Latches_SPEC',
    "! clkA clkB (lad_in:time->(num->bool)) (lbe_in:time->(num->bool)) (lwr_in:time->bool) en_in be_sel
      wr_data addr dest1 be wr be_n
      data_out addr_out be_out .
    Data_Latches_SPEC clkA clkB lad_in lbe_in lwr_in en_in be_sel
                          wr_data addr dest1 be wr be_n
                          data_out addr_out be_out =
    !t:time .

    ((clkA t) ==>
          ((wr_data (t+1) = lad_in t) ∧
           (addr (t+1) = (en_in t) => (lad_in t) | (addr t)) ∧
           (dest1 (t+1) = (en_in t) => (ELEMENT (lad_in t) (31)) | (dest1 t)) ∧
           (be (t+1) = (en_in t) => (lbe_in t) | (be t)) ∧
           (wr (t+1) = (en_in t) => (lwr_in t) | (wr t)) ∧
           (be_n (t+1) = lbe_in t))) ∧
    ((clkB t) ==>
          ((wr_data (t+1) = wr_data t) ∧
           (addr (t+1) = addr t) ∧
           (dest1 (t+1) = dest1 t) ∧
           (be (t+1) = be t) ∧
           (wr (t+1) = wr t) ∧
           (be_n (t+1) = be_n t))) ∧
    ((data_out t = wr_data (t+1)) ∧
       (let od1 = MALTER (addr_out t) (31,27) (be (t+1)) in
       (let od2 = ALTER od1 (26) F in
       (let od3 = MALTER od2 (25,24) (SUBARRAY (addr (t+1)) (1,0)) in
       (let od4 = MALTER od3 (23,0) (SUBARRAY (addr (t+1)) (25,2)) in
       (addr_out t = od4))))) ∧
       (be_out t = (be_sel t) => (be (t+1)) | (be_n (t+1))))"
    );;

%------------------------------------------------------------------------------------------------------------
    Input logic for P_rqt latch.
------------------------------------------------------------------------------------------------------------%

let Req_Inputs_SPEC = new_definition
    ('Req_Inputs_SPEC',
    "! l_ads_ l_den_ (reset_rqt:time->bool) rqt_inS rqt_inR rqt_inE .
     Req_Inputs_SPEC l_ads_ l_den_ reset_rqt rqt_inS rqt_inR rqt_inE =
        !t:time .
            (rqt_inS t = ~(l_ads_ t) ∧ (l_den_ t)) ∧
            (rqt_inR t = reset_rqt t) ∧
            (rqt_inE t = (rqt_inS t) ∨ (rqt_inR t))"
    );;

%------------------------------------------------------------------------------------------------------------
    Input logic for P_size counter.
------------------------------------------------------------------------------------------------------------%

let Ctr_Logic_SPEC = new_definition
    ('Ctr_Logic_SPEC',
     "! clkA clkB l_ad_in load_in down_in zero_cnt
         p_size p_sizeA p_load p_loadA p_down p_downA .
     Ctr_Logic_SPEC clkA clkB l_ad_in load_in down_in zero_cnt
                        p_size p_sizeA p_load p_loadA p_down p_downA =
        !t:time .
```

```
          ((clkA t) ==>
             ((p_sizeA (t+1) = p_size t) ∧
              (p_loadA (t+1) = p_load t) ∧
              (p_downA (t+1) = p_down t) ∧
              (p_size (t+1) = p_size t) ∧
              (p_load (t+1) = p_load t) ∧
              (p_down (t+1) = p_down t))) ∧
          ((clkB t) ==>
             ((p_sizeA (t+1) = p_sizeA t) ∧
              (p_loadA (t+1) = p_loadA t) ∧
              (p_downA (t+1) = p_downA t) ∧
              (p_size (t+1) = (p_loadA t) => SUBARRAY (l_ad_in t) (1,0) |
                              (p_downA t) => DECN 2 (p_sizeA t) |
                              p_sizeA t) ∧
              (p_load (t+1) = load_in t) ∧
              (p_down (t+1) = down_in t))) ∧
          (zero_cnt t = (p_downA t) => (DECN 2 (p_sizeA (t+1)) = (WORDN 0)) | (p_sizeA (t+1) = (WORDN 0)))"
   );;

%------------------------------------------------------------------------------------------------------------
   Accumulated random logic.
------------------------------------------------------------------------------------------------------------%

let Scat_Logic_SPEC = new_definition
   ('Scat_Logic_SPEC',
   "! rst fsm_astate fsm_dstate fsm_hlda_ p_addr p_wr p_rqt zero_cnt i_srdy_
      i_ad_data_out_en l_ad_out_en_ i_rale_ i_male_ i_crqt_
      fsm_mrqt fsm_rst fsm_sack reset_rqt l_ready .
    Scat_Logic_SPEC rst fsm_astate fsm_dstate fsm_hlda_ p_addr p_wr p_rqt zero_cnt i_srdy_
                    i_ad_data_out_en l_ad_out_en_ i_rale_ i_male_ i_crqt_
                    fsm_mrqt fsm_rst fsm_sack reset_rqt l_ready =
      !t:time.
         (i_ad_data_out_en t = (p_wr t) ∧ (fsm_dstate t)) ∧
         (l_ad_out_en_ t = (p_wr t) ∧ (fsm_dstate t) V ~(fsm_hlda_ t) V (fsm_astate t)) ∧
         (i_rale_ t = ~(~(ELEMENT (p_addr t) (31)) ∧
                        (VAL 26 (SUBARRAY (p_addr t) (25,24)) = 3) ∧
                        (fsm_astate t) ∧
                        (p_rqt t))) ∧
         (i_male_ t = ~(~(ELEMENT (p_addr t) (31)) ∧
                         ~(VAL 26 (SUBARRAY (p_addr t) (25,24)) = 3) ∧
                         (fsm_astate t) ∧
                         (p_rqt t))) ∧
         (i_crqt_ t = ~((ELEMENT (p_addr t) (31)) ∧ (p_rqt t))) ∧
         (fsm_mrqt t = ~(ELEMENT (p_addr t) (31)) ∧ (p_rqt t)) ∧
         (fsm_rst t = rst t) ∧
         (fsm_sack t = (zero_cnt t) ∧ ~(i_srdy_ t) ∧ (fsm_dstate t)) ∧
         (reset_rqt t = (rst t) V (fsm_sack t)) ∧
         (l_ready t = ~(i_srdy_ t) ∧ (fsm_dstate t))"
   );;

%------------------------------------------------------------------------------------------------------------
   Input logic for P_lock_ latch.
------------------------------------------------------------------------------------------------------------%
```

```
let Lock_Inputs_SPEC = new_definition
    ('Lock_Inputs_SPEC',
     "! rst fsm_dstate p_male_ p_rale_ lock_inE lock_inh_inE .
     Lock_Inputs_SPEC rst fsm_dstate p_male_ p_rale_ lock_inE lock_inh_inE =
        !t:time .
            (lock_inE t = (rst t) V (fsm_dstate t)) Λ
            (lock_inh_inE t = (rst t) V ~(p_male_ t) V ~(p_rale_ t))"
    );;

%------------------------------------------------------------------------------------------------
    P-Port controller state machine.
------------------------------------------------------------------------------------------------%

let FSM_SPEC = new_definition
    ('FSM_SPEC',
     "! clkA clkB rst_in mrqt_in sack_in cgnt_in_ crqt_in_ hold_in_ lock_in_
        state rst mrqt sack cgnt_ crqt_ hold_ lock_
        stateA astate dstate hlda_
        astate_out dstate_out hlda_out_ .
      FSM_SPEC clkA clkB rst_in mrqt_in sack_in cgnt_in_ crqt_in_ hold_in_ lock_in_
                    state rst mrqt sack cgnt_ crqt_ hold_ lock_
                    stateA astate dstate hlda_
                    astate_out dstate_out hlda_out_ =
        !t:time .
            ((clkA t) ==>
                ((state (t+1) = state t) Λ
                 (rst (t+1) = rst t) Λ
                 (mrqt (t+1) = mrqt t) Λ
                 (sack (t+1) = sack t) Λ
                 (cgnt_ (t+1) = cgnt_ t) Λ
                 (crqt_ (t+1) = crqt_ t) Λ
                 (hold_ (t+1) = hold_ t) Λ
                 (lock_ (t+1) = lock_ t) Λ
                 (stateA (t+1) =
                    ((rst t) => PA |
                     (state t = PH) => ((hold_ t) => PA | PH) |
                     (state t = PA) => (((mrqt t) V ~(cgnt_ t) Λ ~(crqt_ t)) => PD |
                                        (((lock_ t) Λ ~(hold_ t)) => PH | PA)) |
                     (((sack t) Λ (hold_ t)) => PA |
                     ((sack t) Λ ~(hold_ t) Λ ~(lock_ t)) => PA |
                     ((sack t) Λ ~(hold_ t) Λ (lock_ t)) => PH | PD))) Λ
                 (astate (t+1) = (stateA (t+1) = PA)) Λ
                 (dstate (t+1) = (stateA (t+1) = PD)) Λ
                 (hlda_ (t+1) = ~(stateA (t+1) = PA)))) Λ
            ((clkB t) ==>
                ((state (t+1) = stateA t) Λ
                 (rst (t+1) = rst_in t) Λ
                 (mrqt (t+1) = mrqt_in t) Λ
                 (sack (t+1) = sack_in t) Λ
                 (cgnt_ (t+1) = cgnt_in_ t) Λ
                 (crqt_ (t+1) = crqt_in_ t) Λ
                 (hold_ (t+1) = hold_in_ t) Λ
                 (lock_ (t+1) = lock_in_ t) Λ
                 (stateA (t+1) = stateA t) Λ
                (astate (t+1) = astate t) ∧
                (dstate (t+1) = dstate t) ∧
                (hlda_ (t+1) = hlda_ t))) ∧
            ((astate_out t = astate (t+1)) ∧
             (dstate_out t = dstate (t+1)) ∧
             (hlda_out_ t = hlda_ (t+1)))"
   );;

%-------------------------------------------------------------------------------------------------------------------
   P-Port Block.
-------------------------------------------------------------------------------------------------------------------%

let P_Block_SPEC = new_definition
   ('P_Block_SPEC',
    "! (P_fsm_stateA P_fsm_state :time->pfsm_ty)
       (P_wr_data P_addr P_be_ P_be_n_ P_sizeA P_size :time->wordn)
       (P_fsm_astate P_fsm_dstate P_fsm_hlda_ P_dest1 P_wr P_loadA P_downA P_fsm_rst P_fsm_mrqt
        P_fsm_sack P_fsm_cgnt_ P_fsm_crqt_ P_fsm_hold_ P_fsm_lock_ P_rqt P_load P_down P_lock_
        P_lock_inh_ P_male_ P_rale_ :time->bool)
       (L_ad_in L_be_ I_ad_in :time->wordn)
       (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ I_cgnt_ I_hold_ I_srdy_ :time->bool)
       (L_ad_out I_ad_data_out I_ad_addr_out I_be_ :time->wordn)
       (L_ready_ I_rale_ I_male_ I_crqt_ I_cale_ I_mrdy_ I_last_ I_hlda_ I_lock_ :time->bool) .
    P_Block_SPEC (P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P_dest1, P_be_,
                  P_wr, P_be_n_, P_sizeA, P_loadA, P_downA, P_fsm_state, P_fsm_rst, P_fsm_mrqt, P_fsm_sack,
                  P_fsm_cgnt_, P_fsm_crqt_, P_fsm_hold_, P_fsm_lock_, P_rqt, P_size, P_load, P_down, P_lock_,
                  P_lock_inh_, P_male_, P_rale_)
                 (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_, I_srdy_)
                 (L_ad_out, L_ready_, I_ad_data_out, I_ad_addr_out, I_be_, I_rale_, I_male_, I_crqt_, I_cale_,
                  I_mrdy_, I_last_, I_hlda_, I_lock_) =

    ? fsm_astate fsm_dstate rqt data_out addr_out be_out data_out_en reset_rqt
      rqt_inS rqt_inR rqt_inE rqt_outQ load_in down_in zero_cnt zero_cnt_
      l_ad_out_en_ rale_ male_ fsm_mrqt fsm_rst fsm_sack l_ready i_cgnt
      lock_inE lock_outQ lock_inh_inE lock_inh_outQ p_male_outQ p_rale_outQ lock_outQ_ .

    (Data_Latches_SPEC ClkA ClkB L_ad_in L_be_ L_wr rqt fsm_astate
                       P_wr_data P_addr P_dest1 P_be_ P_wr P_be_n_
                       data_out addr_out be_out) ∧
    (TRIBUF_SPEC data_out data_out_en I_ad_data_out) ∧
    (TRIBUF_SPEC addr_out fsm_astate I_ad_addr_out) ∧
    (TRIBUF_SPEC be_out I_hlda_ I_be_) ∧
    (Req_Inputs_SPEC L_ads_ L_den_ reset_rqt rqt_inS rqt_inR rqt_inE) ∧
    (DSRELAT_SPEC GND rqt_inS rqt_inR rqt_inE ClkB P_rqt rqt_outQ) ∧
    (NOT_SPEC rqt_outQ reset_rqt) ∧
    (Ctr_Logic_SPEC ClkA ClkB L_ad_in load_in down_in zero_cnt
                    P_size P_sizeA P_load P_loadA P_down P_downA) ∧
    (Scat_Logic_SPEC Rst fsm_astate fsm_dstate I_hlda_ P_addr P_wr P_rqt zero_cnt I_srdy_
                     data_out_en l_ad_out_en_ rale_ male_ I_crqt_
                     fsm_mrqt fsm_rst fsm_sack reset_rqt l_ready) ∧
    (TRIBUF_SPEC rale_ I_hlda_ I_rale_) ∧
    (TRIBUF_SPEC male_ I_hlda_ I_male_) ∧
    (TRIBUF_SPEC GND I_hlda_ I_mrdy_) ∧
    (NOT_SPEC zero_cnt zero_cnt_) ∧
    (TRIBUF_SPEC zero_cnt_ I_hlda_ I_last_) ∧
    (NOT_SPEC I_ready L_ready_) ∧
    (DSELAT_SPEC L_lock_ Rst lock_inE ClkB P_lock_ lock_outQ) ∧
    (DSELAT_SPEC L_lock_ Rst lock_inh_inE ClkB P_lock_inh_ lock_inh_outQ) ∧
    (Lock_Inputs_SPEC Rst fsm_dstate p_male_outQ p_rale_outQ lock_inE lock_inh_inE) ∧
    (DELAT_SPEC male_ fsm_astate ClkB P_male_ p_male_outQ) ∧
    (DELAT_SPEC rale_ fsm_astate ClkB P_rale_ p_rale_outQ) ∧
    (NOT_SPEC lock_outQ lock_outQ_) ∧
    (NAND2_SPEC lock_outQ_ lock_inh_outQ I_lock_) ∧
    (NOT_SPEC I_cgnt_ i_cgnt) ∧
    (NAND3_SPEC i_cgnt fsm_astate I_hold_ I_cale_) ∧
    (BUF_SPEC I_ad_in L_ad_out) ∧
    (FSM_SPEC ClkA ClkB fsm_rst fsm_mrqt fsm_sack I_cgnt_ I_crqt_ I_hold_ lock_outQ
                P_fsm_state P_fsm_rst P_fsm_mrqt P_fsm_sack P_fsm_cgnt_ P_fsm_crqt_
                P_fsm_hold_
                P_fsm_lock_ P_fsm_stateA P_fsm_astate P_fsm_dstate P_fsm_hlda_
                fsm_astate fsm_dstate I_hlda_)"
    );;

close_theory();;
```

B.2 M Port Specification

```
%-----------------------------------------------------------------------------------------------

   File:        m_block.ml

   Author:      (c) D.A. Fura 1992

   Date:        31 March 1992

   This file contains the ml source for the gate-level specification of the P-Port of the FTEP PIU,
   an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.

-----------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm m_block.th';;

new_theory 'm_block';;

loadf 'abstract';;

map new_parent ['gates_def';'latches_def';'ffs_def';'counters_def';'maux_def';'aux_def';'array_def';'wordn_def'];;

let m_state_ty = ":(mfsm_ty#bool#bool#bool#bool#bool#wordn#wordn#wordn#bool#wordn#
                    mfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#
                    bool#bool#wordn#wordn#wordn#bool#bool#bool#wordn#wordn)";;
let m_state = "((M_fsm_stateA, M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable,
                 M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male_, M_fsm_rd,
                 M_fsm_bw, M_fsm_ww, M_fsm_last_, M_fsm_mrdy_, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr,
                 M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
                :^m_state_ty)";;

let m_env_ty = ":(bool#bool#bool#bool#bool#wordn#bool#bool#wordn#bool#wordn#bool#bool)";;
let m_env = "((ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
               I_mrdy_, MB_data_in, Edac_en_, Reset_parity)
              :^m_env_ty)";;

let m_out_ty = ":(wordn#bool#wordn#wordn#bool#bool#bool#bool#bool)";;
let m_out = "((I_ad_out, I_srdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_,
              MB_parity)
             :^m_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%-----------------------------------------------------------------------------------------------
   SRAM/EEPROM selection logic.
-----------------------------------------------------------------------------------------------%

let SE_Logic_SPEC = new_definition
   ('SE_Logic_SPEC',
    "! clkA clkB (i_ad:time->wordn) male mem_enable M_se cs_e_ cs_s_ .
    SE_Logic_SPEC clkA clkB i_ad male mem_enable M_se cs_e_ cs_s_ =
        !t:time .
            ((clkA t) ==> ((M_se (t+1) = M_se t))) /\
            ((clkB t) ==> ((M_se (t+1) = (male t) => ELEMENT (i_ad t) (23) | M_se t))) /\
            ((cs_e_ t = ~((mem_enable t) /\ ~(M_se (t+1)))) /\
             (cs_s_ t = ~((mem_enable t) /\ (M_se (t+1)))))"
    );;

%-----------------------------------------------------------------------------------------------------
    Read/write selection logic.
-----------------------------------------------------------------------------------------------------%

let WR_Logic_SPEC = new_definition
    (`WR_Logic_SPEC`,
    "! clkA clkB i_ad male mem_enable M_wr wr rd_mem wr_mem .
     WR_Logic_SPEC clkA clkB i_ad male mem_enable M_wr wr rd_mem wr_mem =
        !t:time .
            ((clkA t) ==> ((M_wr (t+1) = M_wr t))) /\
            ((clkB t) ==> ((M_wr (t+1) = (male t) => ELEMENT (i_ad t) (27) | M_wr t))) /\
            ((wr t = M_wr (t+1)) /\
             (rd_mem t = (mem_enable t) /\ ~(M_wr (t+1))) /\
             (wr_mem t = (mem_enable t) /\ (M_wr (t+1))))"
    );;

%-----------------------------------------------------------------------------------------------------
    Address counter logic.
-----------------------------------------------------------------------------------------------------%

let Addr_Ctr_SPEC = new_definition
    (`Addr_Ctr_SPEC`,
    "! clkA clkB (i_ad:time->wordn) male rdyA M_addr M_addrA addr_out .
     Addr_Ctr_SPEC clkA clkB i_ad male rdyA M_addr M_addrA addr_out =
        !t:time .
            ((clkA t) ==>
                ((M_addr (t+1) = M_addr t) /\
                 (M_addrA (t+1) = M_addr t))) /\
            ((clkB t) ==>
                ((M_addr (t+1) = (male t) => (SUBARRAY (i_ad t) (18,0)) |
                                  (rdyA t) => (INCN 18 (M_addrA t)) | (M_addrA t)) /\
                 (M_addrA (t+1) = M_addrA t))) /\
            (addr_out t = (rdyA t) => (INCN 18 (M_addrA (t+1))) | M_addrA (t+1))"
    );;

%-----------------------------------------------------------------------------------------------------
    Byte enable logic.
-----------------------------------------------------------------------------------------------------%

let BE_Logic_SPEC = new_definition
    (`BE_Logic_SPEC`,
    "! clkA clkB (i_be:time->wordn) male srdy wr_mem M_be M_beA be_out ww bw .
     BE_Logic_SPEC clkA clkB i_be male srdy wr_mem M_be M_beA be_out ww bw =
        !t:time .
            ((clkA t) ==>
                ((M_be (t+1) = M_be t) /\
              (M_beA (t+1) = M_be t))) ∧
          ((clkB t) ==>
             ((M_be (t+1) = ((male t) ∨ (srdy t)) => (i_be t) | (M_be t)) ∧
              (M_beA (t+1) = M_beA t))) ∧
          ((be_out t = M_beA (t+1)) ∧
           (ww t = (wr_mem t) ∧ (VAL 3 (M_be (t+1)) = 15)) ∧
           (bw t = (wr_mem t) ∧ ~(VAL 3 (M_be (t+1)) = 15)))"
   );;

%---------------------------------------------------------------------------------------------------------------
   Input logic for M_rdy latch.
---------------------------------------------------------------------------------------------------------------%

let Rdy_Logic_SPEC = new_definition
   ('Rdy_Logic_SPEC',
    "! write read zero_cnt wr_mem rdy .
    Rdy_Logic_SPEC write read zero_cnt wr_mem rdy =
        !t:time .
            (rdy t = (write t) ∧ (zero_cnt t) ∨ (read t) ∧ (zero_cnt t) ∧ ~(wr_mem t))"
   );;

%---------------------------------------------------------------------------------------------------------------
   Wait state counter logic.
---------------------------------------------------------------------------------------------------------------%

let Ctr_Logic_SPEC = new_definition
   ('Ctr_Logic_SPEC',
    "! clkA clkB in dn ld M_count M_countA zero_cnt .
    Ctr_Logic_SPEC clkA clkB in dn ld M_count M_countA zero_cnt =
        !t:time .
            ((clkA t) ==>
                ((M_count (t+1) = M_count t) ∧
                 (M_countA (t+1) = M_count t))) ∧
            ((clkB t) ==>
                ((M_count (t+1) = (ld t) => ((in t) => (WORDN 1) | (WORDN 2)) |
                                             (dn t) => (DECN 1 (M_countA t)) | (M_countA t)) ∧
                 (M_countA (t+1) = M_countA t))) ∧
            (zero_cnt t = (M_countA (t+1) = ((dn t) => (WORDN 1) | (WORDN 0))))"
   );;

%---------------------------------------------------------------------------------------------------------------
   Memory control signal logic.
---------------------------------------------------------------------------------------------------------------%

let Enable_Logic_SPEC = new_definition
   ('Enable_Logic_SPEC',
    "! cs_eeprom_ rd_mem address read write byte_write wwdel
       disable_eeprom disable_writes oe_ edac_le we_ mb_wr_en_ .
    Enable_Logic_SPEC cs_eeprom_ rd_mem address read write byte_write wwdel
                            disable_eeprom disable_writes oe_ edac_le we_ mb_wr_en_ =
        !t:time .
            (oe_ t = ~((rd_mem t) ∧ (address t) ∨ (read t))) ∧
            (we_ t = ~(cs_eeprom_ t) ∧ (disable_eeprom t) ∨
                     (disable_writes t) ∨
                ~((write t) V (byte_write t) V (wwdel t))) Λ
        (edac_le t = read t) Λ
        (mb_wr_en_ t = ~(write t))"
  );;

%-----------------------------------------------------------------------------------------------
   Generation logic for I_srdy_.
-----------------------------------------------------------------------------------------------%

let Srdy_Logic_SPEC = new_definition
   ('Srdy_Logic_SPEC',
    "! wr rdy rdy_outQ srdy_ .
    Srdy_Logic_SPEC wr rdy rdy_outQ srdy_ =
       !t:time .
          srdy_ t = ~((rdy_outQ t) Λ ~(wr t) V (rdy t) Λ (wr t))"
   );;

%-----------------------------------------------------------------------------------------------
   Memory decode logic.
-----------------------------------------------------------------------------------------------%

let EDAC_Decode_Logic_SPEC = new_definition
   ('EDAC_Decode_Logic_SPEC',
    "! (rep:^rep_ty) (mb_data_in:time->wordn) edac_en data_out detect_out .
    EDAC_Decode_Logic_SPEC rep mb_data_in edac_en data_out detect_out =
       !t:time .
          (data_out t = (edac_en t) => (Ham_Dec rep (mb_data_in t)) | (mb_data_in t)) Λ
          (detect_out t = (edac_en t) => (Ham_Det1 rep (mb_data_in t)) | (WORDN 0))"
   );;

%-----------------------------------------------------------------------------------------------
   Memory read latches.
-----------------------------------------------------------------------------------------------%

let Read_Latches_SPEC = new_definition
   ('Read_Latches_SPEC',
    "! (rep:^rep_ty) clkA clkB (data_inD:time->wordn) edac_en edac_le detect_inD detect_inE
       M_rd_data M_rd_dataA M_detect m_data_outQ m_detect_outQ .
    Read_Latches_SPEC rep clkA clkB data_inD edac_en edac_le detect_inD detect_inE
                          M_rd_data M_rd_dataA M_detect m_data_outQ m_detect_outQ =
      !t:time .
         ((clkA t) ==>
            ((M_rd_data (t+1) = M_rd_data t) Λ
             (M_rd_dataA (t+1) = M_rd_data t) Λ
             (M_detect (t+1) = (detect_inE t) => (detect_inD t) | (M_detect t)))) Λ
         ((clkB t) ==>
            ((M_rd_data (t+1) = (edac_le t) => (data_inD t) | (M_rd_data t)) Λ
             (M_rd_dataA (t+1) = M_rd_data t) Λ
             (M_detect (t+1) = M_detect t))) Λ
         ((m_data_outQ t = M_rd_dataA (t+1)) Λ
          (m_detect_outQ t = Ham_Det2 rep ((M_detect (t+1)), (edac_en t))))"
   );;

%-----------------------------------------------------------------------------------------------
```

```
    Enable input logic for EDAC correction reporting.
-------------------------------------------------------------------------------------------%

let Detect_Enable_Logic_SPEC = new_definition
    (`Detect_Enable_Logic_SPEC`,
     "! edac_en edac_rd detect_inE .
     Detect_Enable_Logic_SPEC edac_en edac_rd detect_inE =
        !t:time .
            (detect_inE t = (edac_en t) ∧ (edac_rd t) ∨ ~(edac_rd t))"
    );;

%-------------------------------------------------------------------------------------------
    Memory write data multiplexer.
-------------------------------------------------------------------------------------------%

let Mux_Out_Logic_SPEC = new_definition
    (`Mux_Out_Logic_SPEC`,
     "! (m_data_outQ:time->wordn) i_ad be mb_data_out .
     Mux_Out_Logic_SPEC m_data_outQ i_ad be mb_data_out =
        !t:time .

        let od1 =
            (MALTER (mb_data_out t) (7,0) ((ELEMENT (be t) (0)) => (SUBARRAY (i_ad t) (7,0))
                                                                  | (SUBARRAY (m_data_outQ t) (7,0))))
        in
        (let od2 =
            (MALTER od1 (15,8) ((ELEMENT (be t) (1)) => (SUBARRAY (i_ad t) (15,8))
                                                       | (SUBARRAY (m_data_outQ t) (15,8))))
         in
        (let od3 =
            (MALTER od2 (23,16) ((ELEMENT (be t) (2)) => (SUBARRAY (i_ad t) (23,16))
                                                              | (SUBARRAY (m_data_outQ t) (23,16))))
         in
        (let od4 =
            (MALTER od3 (31,24) ((ELEMENT (be t) (3)) => (SUBARRAY (i_ad t) (31,24))
                                                              | (SUBARRAY (m_data_outQ t) (31,24))))
         in (mb_data_out t = od4))))"
    );;

%-------------------------------------------------------------------------------------------
    Data encoding logic.
-------------------------------------------------------------------------------------------%

let Enc_Out_Logic_SPEC = new_definition
    (`Enc_Out_Logic_SPEC`,
     "! (rep:^rep_ty) (mb_data_out:time->wordn) mb_edata_out .
     Enc_Out_Logic_SPEC rep mb_data_out mb_edata_out =
        !t:time .
            (mb_edata_out t = Ham_Enc rep (mb_data_out t))"
    );;

%-------------------------------------------------------------------------------------------
    Input logic for M_parity latch.
-------------------------------------------------------------------------------------------%
```

```
let Memparity_In_Logic_SPEC = new_definition
   (`Memparity_In_Logic_SPEC`,
    "! srdy mem_enable detect_outQ rst reset_parity memparity_inS memparity_inR memparity_inE .
    Memparity_In_Logic_SPEC srdy mem_enable detect_outQ rst reset_parity
                                        memparity_inS memparity_inR memparity_inE =
      !t:time .
         (memparity_inS t = (srdy t) ∧ (mem_enable t) ∧ (detect_outQ t)) ∧
         (memparity_inR t = (rst t) ∨ (reset_parity t)) ∧
         (memparity_inE t = (memparity_inS t) ∨ (memparity_inR t))"
   );;

%----------------------------------------------------------------------------------------------------------------
   M-Port controller state machine.
----------------------------------------------------------------------------------------------------------------%

let FSM_SPEC = new_definition
   (`FSM_SPEC`,
    "! clkA clkB male_in_ rd_in bw_in ww_in last_in_ mrdy_in_ zero_cnt_in rst_in
      state male_ rd bw ww last_ mrdy_ zero_cnt rst
      stateA address read write byte_write mem_enable
      address_out read_out write_out byte_write_out mem_enable_out .
    FSM_SPEC clkA clkB male_in_ rd_in bw_in ww_in last_in_ mrdy_in_ zero_cnt_in rst_in
              state male_ rd bw ww last_ mrdy_ zero_cnt rst
              stateA address read write byte_write mem_enable
              address_out read_out write_out byte_write_out mem_enable_out =
      !t:time.
         ((clkA t) ==>
            ((state (t+1) = state t) ∧
             (male_ (t+1) = male_ t) ∧
             (rd (t+1) = rd t) ∧
             (bw (t+1) = bw t) ∧
             (ww (t+1) = ww t) ∧
             (last_ (t+1) = last_ t) ∧
             (mrdy_ (t+1) = mrdy_ t) ∧
             (zero_cnt (t+1) = zero_cnt t) ∧
             (rst (t+1) = rst t) ∧
             (stateA (t+1) =
                ((rst t) => MI |
                 (state t = MI) => ((~(male_ t)) => MA | MI)|
                 (state t = MA) => ((~(mrdy_ t) ∧ (ww t)) => MW |
                                    (~(mrdy_ t) ∧ ((rd t) ∨ (bw t))) => MR | MA) |
                 (state t = MR) => (((bw t) ∧ (zero_cnt t)) => MBW |
                                    ((last_ t) ∧ (rd t) ∧ (zero_cnt t)) => MA |
                                    (~(last_ t) ∧ (rd t) ∧ (zero_cnt t)) => MRR | MR) |
                 (state t = MRR) => MI |
                 (state t = MW) => (((zero_cnt t) ∧ ~(last_ t)) => MI |
                                     ((zero_cnt t) ∧ (last_ t)) => MA | MW) |
                                     MW)) ∧
             (address (t+1) = (stateA (t+1) = MA)) ∧
             (read (t+1) = (stateA (t+1) = MR)) ∧
             (write (t+1) = (stateA (t+1) = MW)) ∧
             (byte_write (t+1) = (stateA (t+1) = MBW)) ∧
             (mem_enable (t+1) = ~(stateA (t+1) = MI)))) ∧
```

```
        ((clkB t) ==>
          ((state (t+1) = stateA t) ∧
           (male_ (t+1) = male_in_ t) ∧
           (rd (t+1) = rd_in t) ∧
           (bw (t+1) = bw_in t) ∧
           (ww (t+1) = ww_in t) ∧
           (last_ (t+1) = last_in_ t) ∧
           (mrdy_ (t+1) = mrdy_in_ t) ∧
           (zero_cnt (t+1) = zero_cnt_in t) ∧
           (rst (t+1) = rst_in t) ∧
           (stateA (t+1) = stateA t) ∧
           (address (t+1) = address t) ∧
           (read (t+1) = read t) ∧
           (write (t+1) = write t) ∧
           (byte_write (t+1) = byte_write t) ∧
           (mem_enable (t+1) = mem_enable t))) ∧
        ((address_out t = address (t+1)) ∧
         (read_out t = read (t+1)) ∧
         (write_out t = write (t+1)) ∧
         (byte_write_out t = byte_write (t+1)) ∧
         (mem_enable_out t = mem_enable (t+1)))
");;

%------------------------------------------------------------------------------------------------------------------
   M-Port Block.
------------------------------------------------------------------------------------------------------------------%

let M_Block_SPEC = new_definition
   ('M_Block_SPEC',
    "! (M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_rdyA
       M_fsm_male_ M_fsm_rd M_fsm_bw M_fsm_ww M_fsm_last_ M_fsm_mrdy_ M_fsm_zero_cnt M_fsm_rst M_se
       M_wr M_rdy M_wwdel M_parity :(time->bool))
      (M_addrA M_beA M_countA M_rd_dataA M_addr M_be M_count M_rd_data M_detect :(time->wordn))
      (M_fsm_stateA M_fsm_state :(time->mfsm_ty))
      (ClkA ClkB Rst Disable_eeprom Disable_writes I_male_ I_last_ I_mrdy_ Edac_en_ Reset_parity :(time->bool))
      (I_ad_in I_be_ MB_data_in :(time->wordn))
      (I_srdy_ MB_cs_eeprom_ MB_cs_sram_ MB_we_ MB_oe_ MB_parity :(time->bool))
      (I_ad_out MB_addr MB_data_out :(time->wordn))
      (rep:^rep_ty) .
    M_Block_SPEC (M_fsm_stateA, M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable,
                  M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male_, M_fsm_rd,
                  M_fsm_bw, M_fsm_ww, M_fsm_last_, M_fsm_mrdy_, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr,
                  M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
                 (ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
                  I_mrdy_, MB_data_in, Edac_en_, Reset_parity)
                 (I_ad_out, I_srdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_,
                  MB_parity)
                 rep =

      ? male address read write byte_write mem_enable wr rd_mem wr_mem rdy_outQ srdy
        be ww bw zero_cnt rdy count_inDN count_inLD wwdel_inD wwdel_outQ edac_le
        rdy_outQ srdy_ edac_en data_out detect_out data_inD detect_inD detect_inE
        m_data_outQ m_detect_outQ mb_data_out mb_edata_out mb_wr_en_ mb_wr_en
        memparity_inS memparity_inR memparity_inE .
    (NOT_SPEC I_male_ male) ∧
    (SE_Logic_SPEC ClkA ClkB I_ad_in male mem_enable M_se MB_cs_eeprom_ MB_cs_sram_) ∧
    (WR_Logic_SPEC ClkA ClkB I_ad_in male mem_enable M_wr wr rd_mem wr_mem) ∧
    (Addr_Ctr_SPEC ClkA ClkB I_ad_in male rdy_outQ M_addr M_addrA MB_addr) ∧
    (BE_Logic_SPEC ClkA ClkB I_be_ male srdy wr_mem M_be M_beA be ww bw) ∧
    (Rdy_Logic_SPEC write read zero_cnt wr_mem rdy) ∧
    (Ctr_Logic_SPEC ClkA ClkB MB_cs_eeprom_ count_inDN count_inLD M_count M_countA zero_cnt) ∧
    (OR2_SPEC write read count_inDN) ∧
    (OR2_SPEC address byte_write count_inLD) ∧
    (AND2_SPEC ww address wwdel_inD) ∧
    (DLAT_SPEC wwdel_inD ClkB M_wwdel wwdel_outQ) ∧
    (Enable_Logic_SPEC MB_cs_eeprom_ rd_mem address read write byte_write wwdel_outQ
                          Disable_eeprom Disable_writes MB_oe_ edac_le MB_we_ mb_wr_en_) ∧
    (DFF_SPEC rdy ClkA M_rdy M_rdyA rdy_outQ) ∧
    (Srdy_Logic_SPEC wr rdy rdy_outQ srdy_) ∧
    (TRIBUF_SPEC srdy_ mem_enable I_srdy_) ∧
    (NOT_SPEC srdy_ srdy) ∧
    (NOT_SPEC Edac_en_ edac_en) ∧
    (EDAC_Decode_Logic_SPEC rep MB_data_in edac_en data_out detect_out) ∧
    (Read_Latches_SPEC rep ClkA ClkB data_inD edac_en edac_le detect_inD detect_inE
                          M_rd_data M_rd_dataA M_detect m_data_outQ m_detect_outQ) ∧
    (TRIBUF_SPEC m_data_outQ rd_mem I_ad_out) ∧
    (Detect_Enable_Logic_SPEC edac_en rd_mem detect_inE) ∧
    (Mux_Out_Logic_SPEC m_data_outQ I_ad_in be mb_data_out) ∧
    (Enc_Out_Logic_SPEC rep mb_data_out mb_edata_out) ∧
    (NOT_SPEC mb_wr_en_ mb_wr_en) ∧
    (TRIBUF_SPEC mb_edata_out mb_wr_en MB_data_out) ∧
    (Memparity_In_Logic_SPEC srdy mem_enable m_detect_outQ Rst Reset_parity
                               memparity_inS memparity_inR memparity_inE) ∧
    (DSRELAT_SPEC GND memparity_inS memparity_inR memparity_inE ClkB
                        M_parity MB_parity) ∧
    (FSM_SPEC ClkA ClkB I_male_ rd_mem bw ww I_last_ I_mrdy_ zero_cnt Rst
                   M_fsm_state M_fsm_male_ M_fsm_rd M_fsm_bw M_fsm_ww M_fsm_last_ M_fsm_mrdy_
                   M_fsm_zero_cnt M_fsm_rst
                   M_fsm_stateA M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable
                   address read write byte_write mem_enable)"
    );;

close_theory();;
```

B.3 R Port Specification

```
%------------------------------------------------------------------------------------------------------------

    File:       r_block.ml

    Author:     (c) D.A. Fura 1992

    Date:       31 March 1992

    This file contains the ml source for the gate-level specification of the R-Port of the FTEP PIU, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center.

------------------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm r_block.th';;

new_theory 'r_block';;

map loadf ['abstract';'buses_def'];;

map new_parent ['gates_def';'latches_def';'ffs_def';'counters_def';'datapaths_def';'raux_def'; 'aux_def';
                'array_def';'wordn_def'];;

let r_state_ty = ":(rfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
                    bool#bool#wordn#wordn#bool#bool#wordn#wordn#bool#bool#wordn#wordn#bool#bool#
                    wordn#bool#wordn#wordn#wordn#
                    rfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
                    bool#bool#bool#wordn#wordn#bool#wordn#bool#bool#bool#wordn#wordn#bool#wordn#
                    bool#bool#bool#wordn#wordn#bool#wordn#bool#bool#bool#wordn#wordn#bool#bool#
                    wordn#wordn#wordn#bool#wordn#bool#wordn#bool#wordn#bool)";;
let r_state = "((R_fsm_stateA, R_fsm_cntlatch, R_fsm_srdy_, R_int0_en, R_int0_disA, R_int3_en, R_int3_disA,
               R_c01_cout, R_c01_cout_delA, R_c23_cout, R_c23_cout_delA, R_cntlatch_delA, R_srdy_delA_,
               R_reg_selA, R_ctr0, R_ctr0_ce, R_ctr0_cin, R_ctr0_outA, R_ctr1, R_ctr1_ce, R_ctr1_cin,
               R_ctr1_outA, R_ctr2, R_ctr2_ce, R_ctr2_cin, R_ctr2_outA, R_ctr3, R_ctr3_ce, R_ctr3_cin,
               R_ctr3_outA, R_icr_loadA, R_icr_oldA, R_icrA, R_busA_latch, R_fsm_state, R_fsm_ale_,
               R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_int0_dis, R_int3_dis, R_c01_cout_del, R_int1_en,
               R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_, R_reg_sel, R_ctr0_in,
               R_ctr0_mux_sel, R_ctr0_irden, R_ctr0_cry, R_ctr0_new, R_ctr0_out, R_ctr0_orden, R_ctr1_in,
               R_ctr1_mux_sel, R_ctr1_irden, R_ctr1_cry, R_ctr1_new, R_ctr1_out, R_ctr1_orden, R_ctr2_in,
               R_ctr2_mux_sel, R_ctr2_irden, R_ctr2_cry, R_ctr2_new, R_ctr2_out, R_ctr2_orden, R_ctr3_in,
               R_ctr3_mux_sel, R_ctr3_irden, R_ctr3_cry, R_ctr3_new, R_ctr3_out, R_ctr3_orden, R_icr_load,
               R_icr_old, R_icr_mask, R_icr, R_icr_rden, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr,
               R_sr_rden)
               :^r_state_ty)";;

let r_env_ty = ":(bool#bool#bool#wordn#bool#bool#wordn#bool#bool#bool#wordn#wordn#bool#bool#
                  wordn#wordn#wordn#bool#bool#wordn)";;
let r_env = "((ClkA, ClkB, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
             Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss)
             :^r_env_ty)";;
```

```
let r_out_ty = ":(wordn#bool#bool#bool#bool#bool#wordn#wordn#bool#bool)";;
let r_out = "((I_ad_out, I_srdy_, Int0_, Int1, Int2, Int3_, Ccr, Led, Reset_error, Pmm_invalid)
                :^r_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%------------------------------------------------------------------------------------------------------
    R-Port controller state machine.
------------------------------------------------------------------------------------------------------%

let FSM_SPEC = new_definition
    ('FSM_SPEC',
    "! (ClkA:time->bool) ClkB ale_in_ mrdy_in_ last_in_ rst_in
        ale_ mrdy_ last_ rst state
        cntlatch srdy_ (stateA:time->rfsm_ty)
        s0_out s1_out cntlatch_out srdy_out_ .
     FSM_SPEC ClkA ClkB ale_in_ mrdy_in_ last_in_ rst_in
                  ale_ mrdy_ last_ rst state
                  cntlatch srdy_ stateA
                  s0_out s1_out cntlatch_out srdy_out_ =

        !t:time .

        ((ClkA t) ==>
            ((stateA (t+1) = ((rst t) => RI |
                               ((state t) = RI) => ((~ale_ t) => RA | RI) |
                               ((state t) = RA) => ((~mrdy_ t) => RD | RA) |
                               ((~last_ t) => RI | RA))) ∧
 (cntlatch (t+1) = ((state t = RI) ∧ ~ale_ t)) ∧
 (srdy_ (t+1) = ~((state t = RA) ∧ ~mrdy_ t)) ∧
                  (state (t+1) = state t) ∧
                  (ale_ (t+1) = ale_ t) ∧
                  (mrdy_ (t+1) = mrdy_ t) ∧
                  (last_ (t+1) = last_ t) ∧
                  (rst (t+1) = rst t))) ∧
        ((ClkB t) ==>
            ((stateA (t+1) = stateA t) ∧
            (cntlatch (t+1) = cntlatch t) ∧
            (srdy_ (t+1) = srdy_ t) ∧
            (state (t+1) = stateA t) ∧
            (ale_ (t+1) = ale_in_ t) ∧
            (mrdy_ (t+1) = mrdy_in_ t) ∧
            (last_ (t+1) = last_in_ t) ∧
            (rst (t+1) = rst_in t))) ∧
 ((s0_out (t+1) = (stateA (t+1) = RD)) ∧
  (s1_out (t+1) = ((stateA (t+1) = RA) ∨ (stateA (t+1) = RD))) ∧
  (cntlatch_out t = cntlatch (t+1)) ∧
  (srdy_out_ t = srdy_ (t+1)))"
    );;

%------------------------------------------------------------------------------------------------------
    R_wr latch definition.
------------------------------------------------------------------------------------------------------%
```

```
let Wr_Lat_SPEC = new_definition
   ('Wr_Lat_SPEC',
    "! clkB (iad_in:time->wordn) wr_inE r_wr wr_outQ .
    Wr_Lat_SPEC clkB iad_in wr_inE r_wr wr_outQ =
        !t:time .
            ((~(clkB t)) ==> (r_wr (t+1) = r_wr t)) /\
            ((clkB t) ==> (r_wr (t+1) = (wr_inE t) => (ELEMENT (iad_in t) (27)) | r_wr t)) /\
            (wr_outQ t = r_wr (t+1))"
   );;

%------------------------------------------------------------------------------------------------
   Generation logic for control signals dp_read, r_write, r_read, icr_rd_en, srdy_en.
------------------------------------------------------------------------------------------------%

let RW_Sigs_SPEC = new_definition
   ('RW_Sigs_SPEC',
    "! r_wr s0 s1 disable_writes dp_read r_write r_read icr_rd_en srdy_en .
    RW_Sigs_SPEC r_wr s0 s1 disable_writes dp_read r_write r_read icr_rd_en srdy_en =
    (!t:time .
        (dp_read t = (~r_wr t) /\ ((s0 t) \/ (s1 t))) /\
        (r_write t = (~disable_writes t) /\ (r_wr t) /\ (s0 t) /\ (s1 t)) /\
        (r_read t = (~r_wr t) /\ (~s0 t) /\ (s1 t)) /\
        (icr_rd_en t = (~s0 t) /\ (s1 t)) /\
        (srdy_en t = (s0 t) \/ (s1 t)))"
   );;

%------------------------------------------------------------------------------------------------
   R_reg_sel counter and logic.
------------------------------------------------------------------------------------------------%

let Reg_Sel_Ctr_SPEC = new_definition
   ('Reg_Sel_Ctr_SPEC',
    "! clkA iad_in inL inU_ r_reg_sel r_reg_selA outQ .
    Reg_Sel_Ctr_SPEC clkA iad_in inL inU_ r_reg_sel r_reg_selA outQ =
        !t:time .
            ((clkA t) ==>
                ((r_reg_sel (t+1) = r_reg_sel t) /\
                 (r_reg_selA (t+1) = r_reg_sel t))) /\
            ((~(clkA t)) ==>
                ((r_reg_sel (t+1) =
                    (inL t) => SUBARRAY (iad_in t) (3,0) |
                    (~inU_ t) => INCN 3 (r_reg_selA t) | r_reg_selA t) /\
                 (r_reg_selA (t+1) = r_reg_selA t))) /\
            (outQ t = (~inU_ t) => INCN 3 (r_reg_selA (t+1)) | r_reg_selA (t+1))"
   );;

%------------------------------------------------------------------------------------------------
   Generation logic for register file control signals.
------------------------------------------------------------------------------------------------%

let Reg_File_Ctl_SPEC = new_definition
   ('Reg_File_Ctl_SPEC',
    "! (reg_sel:time->wordn) write read icr_rd_en
```

```
        cir_wr01 cir_wr23
        c0ir_wr c0ir_rd c0or_rd c1ir_wr c1ir_rd c1or_rd
        c2ir_wr c2ir_rd c2or_rd c3ir_wr c3ir_rd c3or_rd
        icr_wr_feedback icr_select icr_rd
        ccr_wr ccr_rd gcr_wr gcr_rd sr_rd .
    Reg_File_Ctl_SPEC reg_sel write read icr_rd_en
                        cir_wr01 cir_wr23
                        c0ir_wr c0ir_rd c0or_rd c1ir_wr c1ir_rd c1or_rd
                        c2ir_wr c2ir_rd c2or_rd c3ir_wr c3ir_rd c3or_rd
                        icr_wr_feedback icr_select icr_rd
                        ccr_wr ccr_rd gcr_wr gcr_rd sr_rd =
    (!t:time .
        (cir_wr01 t = (write t) ∧ (((reg_sel t) = WORDN 8) ∨ ((reg_sel t) = WORDN 9))) ∧
        (cir_wr23 t = (write t) ∧ (((reg_sel t) = WORDN 10) ∨ ((reg_sel t) = WORDN 11))) ∧
        (c0ir_wr t = (write t) ∧ ((reg_sel t) = WORDN 8)) ∧
        (c0ir_rd t = (read t) ∧ ((reg_sel t) = WORDN 8)) ∧
        (c0or_rd t = (read t) ∧ ((reg_sel t) = WORDN 12)) ∧
        (c1ir_wr t = (write t) ∧ ((reg_sel t) = WORDN 9)) ∧
        (c1ir_rd t = (read t) ∧ ((reg_sel t) = WORDN 9)) ∧
        (c1or_rd t = (read t) ∧ ((reg_sel t) = WORDN 13)) ∧
        (c2ir_wr t = (write t) ∧ ((reg_sel t) = WORDN 10)) ∧
        (c2ir_rd t = (read t) ∧ ((reg_sel t) = WORDN 10)) ∧
        (c2or_rd t = (read t) ∧ ((reg_sel t) = WORDN 14)) ∧
        (c3ir_wr t = (write t) ∧ ((reg_sel t) = WORDN 11)) ∧
        (c3ir_rd t = (read t) ∧ ((reg_sel t) = WORDN 11)) ∧
        (c3or_rd t = (read t) ∧ ((reg_sel t) = WORDN 15)) ∧
        (icr_wr_feedback t = (write t) ∧ (((reg_sel t) = WORDN 0) ∨ ((reg_sel t) = WORDN 1))) ∧
        (icr_select t = ~((reg_sel t) = WORDN 1)) ∧
        (icr_rd t = (icr_rd_en t) ∧ (((reg_sel t) = WORDN 0) ∨ ((reg_sel t) = WORDN 1))) ∧
        (ccr_wr t = (write t) ∧ ((reg_sel t) = WORDN 3)) ∧
        (ccr_rd t = (read t) ∧ ((reg_sel t) = WORDN 3)) ∧
        (gcr_wr t = (write t) ∧ ((reg_sel t) = WORDN 2)) ∧
        (gcr_rd t = (read t) ∧ ((reg_sel t) = WORDN 2)) ∧
        (sr_rd t = (read t) ∧ ((reg_sel t) = WORDN 4)))"
    );;

%-------------------------------------------------------------------------------------------------------------
    Input logic for R_int1_en, R_int2_en latches.
-------------------------------------------------------------------------------------------------------------%

let Ctr_Int_Logic_SPEC = new_definition
    ('Ctr_Int_Logic_SPEC',
     "! one_shot interrupt reload cout cout_del cir_wr
        int_en_inR int_en_inS int_en_inE c_ld .
     Ctr_Int_Logic_SPEC one_shot interrupt reload cout cout_del cir_wr
                            int_en_inR int_en_inS int_en_inE c_ld =
     (!t:time .
        (int_en_inR t = (one_shot t) ∧ (cout_del t) ∨ (~interrupt t)) ∧
        (int_en_inS t = (interrupt t) ∧ ((cout t) ∧ (reload t) ∨ (cir_wr t))) ∧
        (int_en_inE t = (one_shot t) ∧ (cout_del t) ∨ (~interrupt t) ∨
                        (interrupt t) ∧ ((cout t) ∧ (reload t) ∨ (cir_wr t))) ∧
        (c_ld t = (cout t) ∧ (reload t) ∨ (cir_wr t)))"
    );;
```

```
%------------------------------------------------------------------------------------------------------------
    Input logic for R_int0_en, R_int3_en latches.
------------------------------------------------------------------------------------------------------------%

let And_Tree_SPEC = new_definition
    ('And_Tree_SPEC',
    "! icr out0 out3 .
     And_Tree_SPEC icr out0 out3 =
     (!t:time .
        (out0 t = (ELEMENT (icr t) (0)) ∧ (ELEMENT (icr t) (8)) ∨
                  (ELEMENT (icr t) (1)) ∧ (ELEMENT (icr t) (9)) ∨
                  (ELEMENT (icr t) (2)) ∧ (ELEMENT (icr t) (10)) ∨
                  (ELEMENT (icr t) (3)) ∧ (ELEMENT (icr t) (11)) ∨
                  (ELEMENT (icr t) (4)) ∧ (ELEMENT (icr t) (12)) ∨
                  (ELEMENT (icr t) (5)) ∧ (ELEMENT (icr t) (13)) ∨
                  (ELEMENT (icr t) (6)) ∧ (ELEMENT (icr t) (14)) ∨
                  (ELEMENT (icr t) (7)) ∧ (ELEMENT (icr t) (15))) ∧
        (out3 t = (ELEMENT (icr t) (16)) ∧ (ELEMENT (icr t) (24)) ∨
                  (ELEMENT (icr t) (17)) ∧ (ELEMENT (icr t) (25)) ∨
                  (ELEMENT (icr t) (18)) ∧ (ELEMENT (icr t) (26)) ∨
                  (ELEMENT (icr t) (19)) ∧ (ELEMENT (icr t) (27)) ∨
                  (ELEMENT (icr t) (20)) ∧ (ELEMENT (icr t) (28)) ∨
                  (ELEMENT (icr t) (21)) ∧ (ELEMENT (icr t) (29)) ∨
                  (ELEMENT (icr t) (22)) ∧ (ELEMENT (icr t) (30)) ∨
                  (ELEMENT (icr t) (23)) ∧ (ELEMENT (icr t) (31))))"
    );;

%------------------------------------------------------------------------------------------------------------
    Generation logic for Int0_, Int3_ signals.
------------------------------------------------------------------------------------------------------------%

let Reg_Int_Logic_SPEC = new_definition
    ('Reg_Int_Logic_SPEC',
    "! int0_en int0_dis int3_en int3_dis disable_int int0_ int3_ .
     Reg_Int_Logic_SPEC int0_en int0_dis int3_en int3_dis disable_int int0_ int3_ =
     (!t:time .
        (int0_ t = ~((int0_en t) ∧ (~int0_dis t) ∧ (~disable_int t))) ∧
        (int3_ t = ~((int3_en t) ∧ (~int3_dis t) ∧ (~disable_int t))))"
    );;

%------------------------------------------------------------------------------------------------------------
    Virtual logic to package several R-Port inputs into single SR input word.
------------------------------------------------------------------------------------------------------------%

let SR_Inputs_SPEC = new_definition
    ('SR_Inputs_SPEC',
    "! cpu_fail reset_cpu piu_fail pmm_fail s_state
       id channelID cb_parity c_ss mb_parity (sr_inp:time->wordn) .
     SR_Inputs_SPEC cpu_fail reset_cpu piu_fail pmm_fail s_state
                         id channelID cb_parity c_ss mb_parity sr_inp =
        !t:time .
           let a1 = (MALTER ARBN (1,0) (cpu_fail t)) in
           let a3 = (MALTER a1 (3,2) (reset_cpu t)) in
           let a5 = (ALTER a3 (8) (piu_fail t)) in
        let a6 = (ALTER a5 (9) (pmm_fail t)) in
        let a7 = (MALTER a6 (15,12) (s_state t)) in
        let a8 = (MALTER a7 (21,16) (id t)) in
        let a9 = (MALTER a8 (23,22) (channelID t)) in
        let a10 = (ALTER a9 (24) (cb_parity t)) in
        let a11 = (MALTER a10 (27,25) (c_ss t)) in
        let a12 = (ALTER a11 (28) (mb_parity t)) in
        (sr_inp t = a12)"
  );;

%-----------------------------------------------------------------------------------------------------------
  Virtual logic to distribute single GCR output word as several pieces.
-----------------------------------------------------------------------------------------------------------%

let GCR_Outputs_SPEC = new_definition
  ('GCR_Outputs_SPEC',
   "! (gcr_out:time->wordn)
      led reload01 oneshot01 interrupt01 enable01
      reload23 oneshot23 interrupt23 enable23 reset_error pmm_invalid .
   GCR_Outputs_SPEC gcr_out led reload01 oneshot01 interrupt01
                       enable01 reload23 oneshot23 interrupt23 enable23 reset_error pmm_invalid =
     !t:time .
       (led t = SUBARRAY (gcr_out t) (3,0)) ∧
       (reload01 t = ELEMENT (gcr_out t) (16)) ∧
       (oneshot01 t = ELEMENT (gcr_out t) (17)) ∧
       (interrupt01 t = ELEMENT (gcr_out t) (18)) ∧
       (enable01 t = ELEMENT (gcr_out t) (19)) ∧
       (reload23 t = ELEMENT (gcr_out t) (20)) ∧
       (oneshot23 t = ELEMENT (gcr_out t) (21)) ∧
       (interrupt23 t = ELEMENT (gcr_out t) (22)) ∧
       (enable23 t = ELEMENT (gcr_out t) (23)) ∧
       (reset_error t = ELEMENT (gcr_out t) (24)) ∧
       (pmm_invalid t = ELEMENT (gcr_out t) (28))"
  );;

%-----------------------------------------------------------------------------------------------------------
  Virtual logic to generate the 12 tristate driver enables for datapath Bus A.
-----------------------------------------------------------------------------------------------------------%

let Bus_Enab_SPEC = new_definition
  ('Bus_Enab_SPEC',
   "! clkA r_ctr0_irden r_ctr0_orden r_ctr1_irden r_ctr1_orden r_ctr2_irden r_ctr2_orden
      r_ctr3_irden r_ctr3_orden r_icr_rden r_ccr_rden r_gcr_rden r_sr_rden
      busA_c0_en1 busA_c0_en2 busA_c1_en1 busA_c1_en2 busA_c2_en1 busA_c2_en2
      busA_c3_en1 busA_c3_en2 busA_icr_en busA_ccr_en busA_gcr_en busA_sr_en .
    Bus_Enab_SPEC clkA r_ctr0_irden r_ctr0_orden r_ctr1_irden r_ctr1_orden r_ctr2_irden r_ctr2_orden
                  r_ctr3_irden r_ctr3_orden r_icr_rden r_ccr_rden r_gcr_rden r_sr_rden
                  busA_c0_en1 busA_c0_en2 busA_c1_en1 busA_c1_en2 busA_c2_en1 busA_c2_en2
                  busA_c3_en1 busA_c3_en2 busA_icr_en busA_ccr_en busA_gcr_en busA_sr_en =
      !t:time .
        (busA_c0_en1 t = (clkA t) ∧ (r_ctr0_irden t)) ∧
        (busA_c0_en2 t = (clkA t) ∧ (r_ctr0_orden t)) ∧
        (busA_c1_en1 t = (clkA t) ∧ (r_ctr1_irden t)) ∧
        (busA_c1_en2 t = (clkA t) ∧ (r_ctr1_orden t)) ∧
        (busA_c2_en1 t = (clkA t) ∧ (r_ctr2_irden t)) ∧
        (busA_c2_en2 t = (clkA t) ∧ (r_ctr2_orden t)) ∧
        (busA_c3_en1 t = (clkA t) ∧ (r_ctr3_irden t)) ∧
        (busA_c3_en2 t = (clkA t) ∧ (r_ctr3_orden t)) ∧
        (busA_icr_en t = (clkA t) ∧ (r_icr_rden t)) ∧
        (busA_ccr_en t = (clkA t) ∧ (r_ccr_rden t)) ∧
        (busA_gcr_en t = (clkA t) ∧ (r_gcr_rden t)) ∧
        (busA_sr_en t = (clkA t) ∧ (r_sr_rden t))"
  );;

%------------------------------------------------------------------------------------------------------------
  R-Port block.
------------------------------------------------------------------------------------------------------------%

let R_Block_SPEC = new_definition
  ('R_Block_SPEC',
  "! (rep:^rep_ty)
     (R_fsm_stateA R_fsm_state :time->rfsm_ty)
     (R_reg_selA R_ctr0 R_ctr0_outA R_ctr1 R_ctr1_outA R_ctr2 R_ctr2_outA R_ctr3 R_ctr3_outA R_icr_oldA
      R_icrA R_busA_latch R_reg_sel R_ctr0_in R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1_new R_ctr1_out
      R_ctr2_in R_ctr2_new R_ctr2_out R_ctr3_in R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr
      R_ccr R_gcr R_sr :time->wordn)
     (R_fsm_cntlatch R_fsm_srdy_ R_int0_en R_int0_disA R_int3_en R_int3_disA R_c01_cout R_c01_cout_delA
      R_c23_cout R_c23_cout_delA R_cntlatch_delA R_srdy_delA_ R_ctr0_ce R_ctr0_cin R_ctr1_ce R_ctr1_cin
      R_ctr2_ce R_ctr2_cin R_ctr3_ce R_ctr3_cin R_icr_loadA R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst
      R_int0_dis R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del
      R_srdy_del_ R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden R_ctr1_mux_sel R_ctr1_irden
      R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden R_ctr3_mux_sel
      R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden
      R_sr_rden :time->bool)
     (I_ad_in I_be_ Cpu_fail Reset_cpu S_state Id ChannelID C_ss :time->wordn)
     (ClkA ClkB Rst I_rale_ I_last_ I_mrdy_ Disable_int Disable_writes Piu_fail Pmm_fail
      CB_parity MB_parity :time->bool)
     (I_ad_out Ccr Led :time->wordn)
     (I_srdy_ Int0_ Int1 Int2 Int3_ Reset_error Pmm_invalid :time->bool) .
  R_Block_SPEC rep
                    (R_fsm_stateA, R_fsm_cntlatch, R_fsm_srdy_, R_int0_en, R_int0_disA, R_int3_en, R_int3_disA,
                     R_c01_cout, R_c01_cout_delA, R_c23_cout, R_c23_cout_delA, R_cntlatch_delA, R_srdy_delA_,
                     R_reg_selA, R_ctr0, R_ctr0_ce, R_ctr0_cin, R_ctr0_outA, R_ctr1, R_ctr1_ce, R_ctr1_cin,
                     R_ctr1_outA, R_ctr2, R_ctr2_ce, R_ctr2_cin, R_ctr2_outA, R_ctr3, R_ctr3_ce, R_ctr3_cin,
                     R_ctr3_outA, R_icr_loadA, R_icr_oldA, R_icrA, R_busA_latch, R_fsm_state, R_fsm_ale_,
                     R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_int0_dis, R_int3_dis, R_c01_cout_del, R_int1_en,
                     R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_, R_reg_sel, R_ctr0_in,
                     R_ctr0_mux_sel, R_ctr0_irden, R_ctr0_cry, R_ctr0_new, R_ctr0_out, R_ctr0_orden, R_ctr1_in,
                     R_ctr1_mux_sel, R_ctr1_irden, R_ctr1_cry, R_ctr1_new, R_ctr1_out, R_ctr1_orden, R_ctr2_in,
                     R_ctr2_mux_sel, R_ctr2_irden, R_ctr2_cry, R_ctr2_new, R_ctr2_out, R_ctr2_orden, R_ctr3_in,
                     R_ctr3_mux_sel, R_ctr3_irden, R_ctr3_cry, R_ctr3_new, R_ctr3_out, R_ctr3_orden, R_icr_load,
                     R_icr_old, R_icr_mask, R_icr, R_icr_rden, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr,
                     R_sr_rden)
                    (ClkA, ClkB, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
                     Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss)
                    (I_ad_out, I_srdy_, Int0_, Int1, Int2, Int3_, Ccr, Led, Reset_error, Pmm_invalid) =

     ? fsm_s0 fsm_s1 fsm_cntlatch fsm_srdy_ srdy_en wr_inE wr_outQ
```

```
    dp_read r_write r_read icr_rd_en c13or_ld srdy_del_outQ_ reg_sel
    icr_rd_en r_cir_wr01 r_cir_wr23 c0ir_wr c0ir_rd c0or_rd c1ir_wr c1ir_rd c1or_rd
    c2ir_wr c2ir_rd c2or_rd c3ir_wr c3ir_rd c3or_rd icr_wr_feedback icr_select icr_rd
    ccr_wr ccr_rd gcr_wr gcr_rd sr_rd icr_ld c01_cout c01_cout_outQ c01_cout_delA_outQ
    c23_cout c23_cout_outQ c23_cout_delA_outQ
    oneshot01 interrupt01 reload01 int1_en_inR int1_en_inS int1_en_inE int1_en_outQ c01_ld
    oneshot23 interrupt23 reload23 int2_en_inR int2_en_inS int2_en_inE int2_en_outQ c23_ld
    enable01 enable23 c0_cout c2_cout ccr_out gcr_out sr_inp
    disable_int_ int0_en_inD int0_en_outQ int0_dis_outQ int3_en_inD int3_en_outQ int3_dis_outQ
    icr_out BusA BusB_in busA_latch_out
    (BusA_c0_out1 BusA_c0_out2 BusA_c1_out1 BusA_c1_out2 BusA_c2_out1 BusA_c2_out2
     BusA_c3_out1 BusA_c3_out2 BusA_icr_out BusA_ccr_out BusA_gcr_out BusA_sr_out :time->wordn)
    (BusA_c0_en1 BusA_c0_en2 BusA_c1_en1 BusA_c1_en2 BusA_c2_en1 BusA_c2_en2
     BusA_c3_en1 BusA_c3_en2 BusA_icr_en BusA_ccr_en BusA_gcr_en BusA_sr_en :time->bool)
.

(FSM_SPEC ClkA ClkB I_rale_ I_mrdy_ I_last_ Rst
          R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst R_fsm_state
          R_fsm_cntlatch R_fsm_srdy_ R_fsm_stateA
          fsm_s0 fsm_s1 fsm_cntlatch fsm_srdy_) ∧
(TRIBUF_SPEC fsm_srdy_ srdy_en I_srdy_) ∧
(NOT_SPEC I_rale_ wr_inE) ∧
(Wr_Lat_SPEC ClkB I_ad_in wr_inE R_wr wr_outQ) ∧
(RW_Sigs_SPEC wr_outQ fsm_s0 fsm_s1 Disable_writes dp_read r_write r_read icr_rd_en srdy_en) ∧
(DFF_SPEC fsm_cntlatch ClkA R_cntlatch_del R_cntlatch_delA c13or_ld) ∧
(DFF_SPEC fsm_srdy_ ClkA R_srdy_del_ R_srdy_delA_ srdy_del_outQ_) ∧
(Reg_Sel_Ctr_SPEC ClkA I_ad_in wr_inE srdy_del_outQ_ R_reg_sel R_reg_selA reg_sel) ∧
(Reg_File_Ctl_SPEC reg_sel r_write r_read icr_rd_en
                   r_cir_wr01 r_cir_wr23
                   c0ir_wr c0ir_rd c0or_rd c1ir_wr c1ir_rd c1or_rd
                   c2ir_wr c2ir_rd c2or_rd c3ir_wr c3ir_rd c3or_rd
                   icr_wr_feedback icr_select icr_rd
                   ccr_wr ccr_rd gcr_wr gcr_rd sr_rd) ∧
(DFF_SPEC icr_wr_feedback ClkA R_icr_load R_icr_loadA icr_ld) ∧
(DLAT_SPEC c01_cout ClkA R_c01_cout c01_cout_outQ) ∧
(DLAT_SPEC c23_cout ClkA R_c23_cout c23_cout_outQ) ∧
(DFF_SPEC c01_cout_outQ ClkA R_c01_cout_del R_c01_cout_delA c01_cout_delA_outQ) ∧
(DFF_SPEC c23_cout_outQ ClkA R_c23_cout_del R_c23_cout_delA c23_cout_delA_outQ) ∧
(Ctr_Int_Logic_SPEC oneshot01 interrupt01 reload01 c01_cout_outQ c01_cout_delA_outQ
                    r_cir_wr01 int1_en_inR int1_en_inS int1_en_inE c01_ld) ∧
(Ctr_Int_Logic_SPEC oneshot23 interrupt23 reload23 c23_cout_outQ c23_cout_delA_outQ
                    r_cir_wr23 int2_en_inR int2_en_inS int2_en_inE c23_ld) ∧
(DSRELAT_SPEC GND int1_en_inS int1_en_inR int1_en_inE ClkB R_int1_en int1_en_outQ) ∧
(DSRELAT_SPEC GND int2_en_inS int2_en_inR int2_en_inE ClkB R_int2_en int2_en_outQ) ∧
(NOT_SPEC Disable_int disable_int_) ∧
(AND3_SPEC c01_cout_outQ int1_en_outQ disable_int_ Int1) ∧
(AND3_SPEC c23_cout_outQ int2_en_outQ disable_int_ Int2) ∧
(And_Tree_SPEC icr_out int0_en_inD int3_en_inD) ∧
(DLAT_SPEC int0_en_inD ClkA R_int0_en int0_en_outQ) ∧
(DLAT_SPEC int3_en_inD ClkA R_int3_en int3_en_outQ) ∧
(DFF_SPEC int0_en_outQ ClkA R_int0_dis R_int0_disA int0_dis_outQ) ∧
(DFF_SPEC int3_en_outQ ClkA R_int3_dis R_int3_disA int3_dis_outQ) ∧
(Reg_Int_Logic_SPEC int0_en_outQ int0_dis_outQ int3_en_outQ int3_dis_outQ
                    Disable_int Int0_ Int3_) ∧
    (DLATn_SPEC BusA ClkA R_busA_latch busA_latch_out) ∧
    (TRIBUF_SPEC busA_latch_out dp_read I_ad_out) ∧
    (BUF_SPEC I_ad_in BusB_in) ∧
    (DP_CTR_SPEC ClkA ClkB BusB_in c0ir_wr c01_ld c0ir_rd enable01 VDD fsm_cntlatch
                        c0or_rd R_ctr0_in R_ctr0_mux_sel R_ctr0_irden R_ctr0 R_ctr0_ce R_ctr0_cin
                        R_ctr0_cry R_ctr0_new R_ctr0_outA R_ctr0_out R_ctr0_orden
                        BusA_c0_out1 BusA_c0_out2 c0_cout) ∧
    (DP_CTR_SPEC ClkA ClkB BusB_in c1ir_wr c01_ld c1ir_rd VDD c0_cout c13or_ld
                        c1or_rd R_ctr1_in R_ctr1_mux_sel R_ctr1_irden R_ctr1 R_ctr1_ce R_ctr1_cin
                        R_ctr1_cry R_ctr1_new R_ctr1_outA R_ctr1_out R_ctr1_orden
                        BusA_c1_out1 BusA_c1_out2 c01_cout) ∧
    (DP_CTR_SPEC ClkA ClkB BusB_in c2ir_wr c23_ld c2ir_rd enable23 VDD fsm_cntlatch
                        c2or_rd R_ctr2_in R_ctr2_mux_sel R_ctr2_irden R_ctr2 R_ctr2_ce R_ctr2_cin
                        R_ctr2_cry R_ctr2_new R_ctr2_outA R_ctr2_out R_ctr2_orden
                        BusA_c2_out1 BusA_c2_out2 c2_cout) ∧
    (DP_CTR_SPEC ClkA ClkB BusB_in c3ir_wr c23_ld c3ir_rd VDD c2_cout c13or_ld
                        c3or_rd R_ctr3_in R_ctr3_mux_sel R_ctr3_irden R_ctr3 R_ctr3_ce R_ctr3_cin
                        R_ctr3_cry R_ctr3_new R_ctr3_outA R_ctr3_out R_ctr3_orden
                        BusA_c3_out1 BusA_c3_out2 c23_cout) ∧
    (DP_ICR_SPEC rep ClkA ClkB BusA BusB_in icr_wr_feedback icr_rd icr_select R_icr_loadA icr_rd
                       R_icr_oldA R_icr_old R_icr_mask R_icrA R_icr R_icr_rden
                       BusA_icr_out icr_out) ∧
    (DP_CR_SPEC ClkA ClkB BusB_in ccr_wr ccr_rd R_ccr R_ccr_rden BusA_ccr_out ccr_out) ∧
    (DP_CR_SPEC ClkA ClkB BusB_in gcr_wr gcr_rd R_gcr R_gcr_rden BusA_gcr_out gcr_out) ∧
    (GCR_Outputs_SPEC gcr_out Led reload01 oneshot01 interrupt01
                            enable01 reload23 oneshot23 interrupt23 enable23 Reset_error Pmm_invalid) ∧
    (SR_Inputs_SPEC Cpu_fail Reset_cpu Piu_fail Pmm_fail S_state
                         Id ChannelID CB_parity C_ss MB_parity sr_inp) ∧
    (DP_SR_SPEC ClkA ClkB sr_inp fsm_cntlatch sr_rd R_sr R_sr_rden BusA_sr_out) ∧
    (Bus_Enab_SPEC ClkA R_ctr0_irden R_ctr0_orden R_ctr1_irden R_ctr1_orden R_ctr2_irden R_ctr2_orden
                         R_ctr3_irden R_ctr3_orden R_icr_rden R_ccr_rden R_gcr_rden R_sr_rden
                         BusA_c0_en1 BusA_c0_en2 BusA_c1_en1 BusA_c1_en2 BusA_c2_en1 BusA_c2_en2
                         BusA_c3_en1 BusA_c3_en2 BusA_icr_en BusA_ccr_en BusA_gcr_en BusA_sr_en) ∧
    (Bus_12_1_SPEC BusA_c0_out1 BusA_c0_out2 BusA_c1_out1 BusA_c1_out2 BusA_c2_out1 BusA_c2_out2
                        BusA_c3_out1 BusA_c3_out2 BusA_icr_out BusA_ccr_out BusA_gcr_out BusA_sr_out
                        BusA_c0_en1 BusA_c0_en2 BusA_c1_en1 BusA_c1_en2 BusA_c2_en1 BusA_c2_en2
                        BusA_c3_en1 BusA_c3_en2 BusA_icr_en BusA_ccr_en BusA_gcr_en BusA_sr_en BusA)"
  );;

close_theory();;
```

B.4 C Port Specification

```
%------------------------------------------------------------------------------------------------------------------

    File:       c_block.ml

    Author:     (c) D.A. Fura 1992

    Date:       31 March 1992

    This file contains the ml source for the gate-level specification of the C-Port of the FTEP PIU, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center.

------------------------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm c_block.th';;

new_theory 'c_block';;

loadf 'abstract';;

map new_parent ['gates_def';'latches_def';'ffs_def';'counters_def';'caux_def';'aux_def';'array_def';'wordn_def'];;

let MSTART = "WORDN 4";;
let MEND = "WORDN 5";;
let MRDY = "WORDN 6";;
let MWAIT = "WORDN 7";;
let MABORT = "WORDN 0";;

let SACK = "WORDN 5";;
let SRDY = "WORDN 6";;
let SWAIT = "WORDN 7";;
let SABORT = "WORDN 0";;

let c_state_ty = ":(cmfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                    wordn#bool#bool#bool#bool#bool#
                    csfsm_ty#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                    cefsm_ty#bool#
                    bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn#wordn#
                    cmfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#bool#
                    csfsm_ty#bool#bool#bool#bool#bool#bool#wordn#
                    cefsm_ty#bool#bool#bool#bool#bool#bool#
                    bool#wordn#bool#bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool#bool#bool#
                    bool#bool#wordn#wordn#wordn)";;
let c_state = "((C_mfsm_stateA,C_mfsm_mabort,C_mfsm_midle,C_mfsm_mrequest,C_mfsm_ma3,C_mfsm_ma2,C_mfsm_ma1,
                C_mfsm_ma0,C_mfsm_md1,C_mfsm_md0,C_mfsm_iad_en_m,C_mfsm_m_cout_sel1,C_mfsm_m_cout_sel0,
                C_mfsm_ms,C_mfsm_rqt_,C_mfsm_cgnt_,C_mfsm_cm_en,C_mfsm_abort_le_en_,C_mfsm_mparity,
                C_sfsm_stateA,C_sfsm_ss,C_sfsm_iad_en_s,C_sfsm_sidle,C_sfsm_slock,C_sfsm_sa1,C_sfsm_sa0,
                C_sfsm_sale,C_sfsm_sd1,C_sfsm_sd0,C_sfsm_sack,C_sfsm_sabort,C_sfsm_s_cout_sel0,C_sfsm_sparity,
                C_efsm_stateA,C_efsm_srdy_en,
                C_clkAA,C_sidle_delA,C_mrqt_delA,C_last_inA_,C_ssA,C_holdA_,C_rd_srdy,C_cout_0_le_delA,
                C_cin_2_leA,C_mrdy_delA_,C_iad_en_s_delA,C_wrdyA,C_rrdyA,C_iad_out,C_a1a0,C_a3a2,
                C_mfsm_state,C_mfsm_srdy_en,C_mfsm_D,C_mfsm_grant,C_mfsm_rst,C_mfsm_busy,C_mfsm_write,
                C_mfsm_crqt_,C_mfsm_hold_,C_mfsm_last_,C_mfsm_lock_,C_mfsm_ss,C_mfsm_invalid,
                C_sfsm_state,C_sfsm_D,C_sfsm_grant,C_sfsm_rst,C_sfsm_write,C_sfsm_addressed,C_sfsm_hlda_,C_sfsm_ms,
                C_efsm_state,C_efsm_cale_,C_efsm_last_,C_efsm_male_,C_efsm_rale_,C_efsm_srdy_,C_efsm_rst,
                C_wr,C_sizewrbe,C_clkA,C_sidle_del,C_mrqt_del,C_last_in_,C_lock_in_,C_ss,C_last_out_,
                C_hold_,C_cout_0_le_del,C_cin_2_le,C_mrdy_del_,C_iad_en_s_del,C_wrdy,
                C_rrdy,C_parity,C_source,C_data_in,C_iad_in)
               :^c_state_ty)";;

let c_env_ty = ":(wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                  wordn#wordn#wordn#wordn#bool#bool#bool#bool#wordn#wordn#bool#bool#wordn#bool)";;
let c_env = "((I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_,
               I_lock_, I_cale_, I_hlda_, I_crqt_,
               CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in,
               Rst, ClkA, ClkB, ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr, Reset_error)
             :^c_env_ty)";;

let c_out_ty = ":(bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
                  bool#wordn#wordn#wordn#wordn#bool#bool)";;
let c_out = "((I_cgnt_, I_mrdy_out_, I_hold_, I_rale_out_, I_male_out_, I_last_out_, I_srdy_out_,
               I_ad_out, I_be_out_,
               CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out, C_ss_out, Disable_writes, CB_parity)
             :^c_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%-----------------------------------------------------------------------------------------------
   Input logic for C_last_in_ flip-flop.
-----------------------------------------------------------------------------------------------%

let Last_Logic = new_definition
   ('Last_Logic',
   "! rst clkD mfsm_md1 mfsm_mabort last_in_inE .
    Last_Logic rst clkD mfsm_md1 mfsm_mabort last_in_inE =
       !t:time .
          (last_in_inE t = (rst t) \/ ((clkD t) /\ (mfsm_md1 t)) \/ (mfsm_mabort t))"
   );;

%-----------------------------------------------------------------------------------------------
   Input logic for C_last_out_ latch.
-----------------------------------------------------------------------------------------------%

let Hold_Logic = new_definition
   ('Hold_Logic',
   "! (cb_ms:time->wordn) clkD sfsm_sa1 last_out_inS last_out_inR last_out_inE .
    Hold_Logic cb_ms clkD sfsm_sa1 last_out_inS last_out_inR last_out_inE =
       !t:time .
          (last_out_inS t = sfsm_sa1 t) /\
          (last_out_inR t = (clkD t) /\ ((cb_ms t = ^MEND) \/ (cb_ms t = ^MABORT))) /\
          (last_out_inE t = (last_out_inS t) \/ (last_out_inR t))"
   );;

%-----------------------------------------------------------------------------------------------
```

```
    Generation logic for cout_sel signal.
-----------------------------------------------------------------------------------------------------------%

let Cout_Sel_Logic_SPEC = new_definition
    (`Cout_Sel_Logic_SPEC`,
     "! sfsm_s_cout_sel0 mfsm_m_cout_sel1 mfsm_m_cout_sel0 sfsm_sd0 sfsm_sd1 (cout_sel:time->wordn) .
     Cout_Sel_Logic_SPEC sfsm_s_cout_sel0 mfsm_m_cout_sel1 mfsm_m_cout_sel0 sfsm_sd0 sfsm_sd1 cout_sel =
        !t:time .
            (cout_sel t = ((sfsm_sd0 t) V (sfsm_sd1 t))
                => (let a1 = (ALTER (cout_sel t) 0 (sfsm_s_cout_sel0 t))
                      in (ALTER a1 1 F))
                 | (let a1 = (ALTER (cout_sel t) 0 (mfsm_m_cout_sel0 t))
                    in (ALTER a1 1 (mfsm_m_cout_sel1 t))))"
    );;

%-----------------------------------------------------------------------------------------------------------
    Generation logic for srdy signal.
-----------------------------------------------------------------------------------------------------------%

let Srdy_In_Logic_SPEC = new_definition
    (`Srdy_In_Logic_SPEC`,
     "! (cb_ss:time->wordn) dfsm_srdy .
     Srdy_In_Logic_SPEC cb_ss dfsm_srdy =
        !t:time. (dfsm_srdy t = (cb_ss t = ^SRDY))"
    );;

%-----------------------------------------------------------------------------------------------------------
    Input logic for C_wrdy, C_rrdy latches.
-----------------------------------------------------------------------------------------------------------%

let Rdy_Logic_SPEC = new_definition
    (`Rdy_Logic_SPEC`,
     "! mfsm_md0 mfsm_md1 clkD write srdy wrdy_inD rrdy_inD .
     Rdy_Logic_SPEC mfsm_md0 mfsm_md1 clkD write srdy wrdy_inD rrdy_inD =
        !t:time .
            (wrdy_inD t = (srdy t) Λ (write t) Λ (mfsm_md1 t) Λ (clkD t)) Λ
            (rrdy_inD t = (srdy t) Λ ~(write t) Λ (mfsm_md0 t) Λ (clkD t))"
    );;

%-----------------------------------------------------------------------------------------------------------
    Generation logic for I_srdy_out_ signal.
-----------------------------------------------------------------------------------------------------------%

let ISrdy_Out_Logic_SPEC = new_definition
    (`ISrdy_Out_Logic_SPEC`,
     "! wrdyA_outQ rrdyA_outQ fsm_mabort cale_ srdy_en isrdy_inD isrdy_inE .
     ISrdy_Out_Logic_SPEC wrdyA_outQ rrdyA_outQ fsm_mabort cale_ srdy_en isrdy_inD isrdy_inE =
        !t:time .
            (isrdy_inD t = ~((wrdyA_outQ t) V (rrdyA_outQ t) V (fsm_mabort t))) Λ
            (isrdy_inE t = ~(cale_ t) V (srdy_en t))"
    );;

%-----------------------------------------------------------------------------------------------------------
    Generation logic for CBss_out signal.
```

```
-------------------------------------------------------------------------------------------------------------%

let CBss_Out_Logic_SPEC = new_definition
    ('CBss_Out_Logic_SPEC',
     "! (sfsm_ss:time->wordn) pmm_failure piu_valid cbss_out .
     CBss_Out_Logic_SPEC sfsm_ss pmm_failure piu_valid cbss_out =
         !t:time .
             (cbss_out t = (let a1 = (MALTER (cbss_out t) (1,0) (SUBARRAY (sfsm_ss t) (1,0)))
                            in (ALTER a1 (2) ((ELEMENT (sfsm_ss t) (2)) ∧ (pmm_failure t) ∧ (piu_valid t) ))))"
    );;

%-------------------------------------------------------------------------------------------------------------
    Generation logic for CBms_out signal.
-------------------------------------------------------------------------------------------------------------%

let CBms_Out_Logic_SPEC = new_definition
    ('CBms_Out_Logic_SPEC',
     "! (mfsm_ms:time->wordn) pmm_failure piu_valid cbms_out .
     CBms_Out_Logic_SPEC mfsm_ms pmm_failure piu_valid cbms_out =
         !t:time .
             (cbms_out t = (let a1 = (MALTER (cbms_out t) (1,0) (SUBARRAY (mfsm_ms t) (1,0)))
                             in (ALTER a1 (2) ((ELEMENT (mfsm_ms t) (2)) ∧ ~(pmm_failure t) ∧ ~(piu_valid t)))))"
    );;

%-------------------------------------------------------------------------------------------------------------
    Generation logic for cout_1_le signal.
-------------------------------------------------------------------------------------------------------------%

let Cout_1_Le_Logic_SPEC = new_definition
    ('Cout_1_Le_Logic_SPEC',
     "! dfsm_master cout_0_le_del dfsm_cout_1_le cout_1_le .
     Cout_1_Le_Logic_SPEC dfsm_master cout_0_le_del dfsm_cout_1_le cout_1_le =
         !t:time .
             (cout_1_le t = ~(dfsm_master t) ∧ (dfsm_cout_1_le t) ∨ (dfsm_master t) ∧ (cout_0_le_del t))"
    );;

%-------------------------------------------------------------------------------------------------------------
    Generation logic for iad_en signal.
-------------------------------------------------------------------------------------------------------------%

let Iad_En_Logic_SPEC = new_definition
    ('Iad_En_Logic_SPEC',
     "! mfsm_iad_en_m sfsm_iad_en_s iad_en_s_del iad_en .
     Iad_En_Logic_SPEC mfsm_iad_en_m sfsm_iad_en_s iad_en_s_del iad_en =
         !t:time .
             (iad_en t = (mfsm_iad_en_m t) ∨ (sfsm_iad_en_s t) ∨ (iad_en_s_del t))"
    );;

%-------------------------------------------------------------------------------------------------------------
    Generation logic for c_pe_cnt signal.
-------------------------------------------------------------------------------------------------------------%

let Pe_Cnt_Logic_SPEC = new_definition
    ('Pe_Cnt_Logic_SPEC',
    "! clkD (sfsm_sparity:time->bool) mfsm_mparity (cb_ss_in:time->wordn) c_pe_cnt .
    Pe_Cnt_Logic_SPEC clkD sfsm_sparity mfsm_mparity cb_ss_in c_pe_cnt =
       !t:time .
          (c_pe_cnt t = (clkD t) ∧
                         (~((sfsm_sparity t) = (mfsm_mparity t)) ∨ ((SUBARRAY (cb_ss_in t) (1,0)) = WORDN 0)))"
   );;

%-----------------------------------------------------------------------------------------------------------
   Generation logic for c_grant, c_busy signals.
-----------------------------------------------------------------------------------------------------------%

let Grant_Logic_SPEC = new_definition
   ('Grant_Logic_SPEC',
    "! (id:time->wordn) (rqt_:time->wordn) busy grant .
    Grant_Logic_SPEC id rqt_ busy grant =
       !t:time .
          (busy t = ~(ELEMENT (rqt_ t) (3)) ∨ ~(ELEMENT (rqt_ t) (2)) ∨ ~(ELEMENT (rqt_ t) (1))) ∧
          (grant t = ((SUBARRAY (id t) (1,0)) = WORDN 0) ∧ ~(ELEMENT (rqt_ t) (0)) ∨
                     ((SUBARRAY (id t) (1,0)) = WORDN 1) ∧ ~(ELEMENT (rqt_ t) (0)) ∧ (ELEMENT (rqt_ t) (1)) ∨
                     ((SUBARRAY (id t) (1,0)) = WORDN 2) ∧ ~(ELEMENT (rqt_ t) (0)) ∧ (ELEMENT (rqt_ t) (1)) ∧
                                                                               (ELEMENT (rqt_ t) (2)) ∨
                     ((SUBARRAY (id t) (1,0)) = WORDN 3) ∧ ~(ELEMENT (rqt_ t) (0)) ∧ (ELEMENT (rqt_ t) (1)) ∧
                                                     (ELEMENT (rqt_ t) (2)) ∧ (ELEMENT (rqt_ t) (3)))"
   );;

%-----------------------------------------------------------------------------------------------------------
   Generation logic for addressed signal.
-----------------------------------------------------------------------------------------------------------%

let Addressed_Logic_SPEC = new_definition
   ('Addressed_Logic_SPEC',
    "! (id:time->wordn) (source:time->wordn) addressed .
    Addressed_Logic_SPEC id source addressed =
       !t:time .
          (addressed t = ((ELEMENT (id t) (0)) = (ELEMENT (source t) (10))) ∧
                         ((ELEMENT (id t) (1)) = (ELEMENT (source t) (11))) ∧
                         ((ELEMENT (id t) (2)) = (ELEMENT (source t) (12))) ∧
                         ((ELEMENT (id t) (3)) = (ELEMENT (source t) (13))) ∧
                         ((ELEMENT (id t) (4)) = (ELEMENT (source t) (14))) ∧
                         ((ELEMENT (id t) (5)) = (ELEMENT (source t) (15))))"
   );;

%-----------------------------------------------------------------------------------------------------------
   Generation logic for Disable_writes signal.
-----------------------------------------------------------------------------------------------------------%

let D_Writes_Logic_SPEC = new_definition
   ('D_Writes_Logic_SPEC',
    "! dfsm_slave (chan_id:time->wordn) (source:time->wordn) disable_writes .
    D_Writes_Logic_SPEC dfsm_slave chan_id source disable_writes =
       !t:time .
          (disable_writes t = (dfsm_slave t) ∧ ~((ELEMENT (chan_id t) (0)) ∧ (ELEMENT (source t) (6)))
                                             ∧ ~((ELEMENT (chan_id t) (1)) ∧ (ELEMENT (source t) (7)))
                                             ∧ ~((ELEMENT (chan_id t) (2)) ∧ (ELEMENT (source t) (8)))
```

```
                                        ∧ ~((ELEMENT (chan_id t) (3)) ∧ (ELEMENT (source t) (9))))"
  );;

%----------------------------------------------------------------------------------------------------------
  Generation logic for c_pe signal.
----------------------------------------------------------------------------------------------------------%

let Parity_Decode_Logic_SPEC = new_definition
  ('Parity_Decode_Logic_SPEC',
   "!rep cad_in cad_in_dec cad_in_det .
    Parity_Decode_Logic_SPEC rep cad_in cad_in_dec cad_in_det =
      !t:time .
        (cad_in_dec t = (Par_Dec rep (cad_in t))) ∧
        (cad_in_det t = (Par_Det rep (cad_in t)))"
  );;

%----------------------------------------------------------------------------------------------------------
  Input logic for C_parity latch.
----------------------------------------------------------------------------------------------------------%

let Parity_Signal_Inputs_SPEC = new_definition
  ('Parity_Signal_Inputs_SPEC',
   "! rst cad_in_det clkD c_pe_cnt reset_parity
      c_parity_inS c_parity_inR c_parity_inE .
    Parity_Signal_Inputs_SPEC rst cad_in_det clkD c_pe_cnt reset_parity
                                  c_parity_inS c_parity_inR c_parity_inE =
      !t:time .
        (c_parity_inS t = (cad_in_det t) ∧ (clkD t) ∧ (c_pe_cnt t)) ∧
        (c_parity_inR t = (rst t) V (reset_parity t)) ∧
        (c_parity_inE t = (c_parity_inS t) V (c_parity_inR t))"
  );;

%----------------------------------------------------------------------------------------------------------
  C-Bus input latches.
----------------------------------------------------------------------------------------------------------%

let CB_In_Latches_SPEC = new_definition
  ('CB_In_Latches_SPEC',
   "! clkA clkB rst (cad_in_dec:time->wordn) cin_0_le cin_1_le cin_2_le cin_3_le cin_4_le
      (source:time->wordn) (sizewrbe:time->wordn) iad_preout
      c_source c_data_in c_sizewrbe c_iad_preout .
   CB_In_Latches_SPEC clkA clkB rst cad_in_dec cin_0_le cin_1_le cin_2_le cin_3_le cin_4_le
                         source sizewrbe iad_preout
                         c_source c_data_in c_sizewrbe c_iad_preout =
     !t:time .
       ((clkA t) ==>
          ((c_source (t+1) = c_source t) ∧
           (c_data_in (t+1) = c_data_in t) ∧
           (c_sizewrbe (t+1) = c_sizewrbe t) ∧
           (c_iad_preout (t+1) = (cin_2_le t) => (c_data_in t) | (c_iad_preout t)))) ∧
       ((clkB t) ==>
          ((c_source (t+1) = (rst t) => WORDN 0 |
                             (cin_3_le t) => (cad_in_dec t) |
                             (c_source t)) ∧
```

```
              (c_data_in (t+1) = (rst t) => MALTER (c_data_in t) (31,16) (WORDN 0) |
                                    ((cin_1_le t) ∧ (~cin_0_le t)) => MALTER (c_data_in t) (31,16) (cad_in_dec t) |
                                    (c_data_in (t+1))) ∧
              (c_data_in (t+1) = (rst t) => WORDN 0 |
                                    ((cin_0_le t) ∧ (~cin_1_le t)) => MALTER (c_data_in t) (15,0) (cad_in_dec t) |
                                    (c_data_in (t+1))) ∧
              (c_sizewrbe (t+1) = (rst t) => WORDN 0 |
                                     (cin_4_le t) => SUBARRAY (c_data_in t) (31,22) |
                                     (c_sizewrbe t)) ∧
              (c_iad_preout (t+1) = (c_iad_preout t)))) ∧
           ((source t = c_source (t+1)) ∧
            (sizewrbe t = c_sizewrbe (t+1)) ∧
            (iad_preout t = c_iad_preout (t+1)))"
);;

%-----------------------------------------------------------------------------------------------------------
   Generation logic for I_be_out_ signal.
-----------------------------------------------------------------------------------------------------------%

let BE_Out_Logic_SPEC = new_definition
   ('BE_Out_Logic_SPEC',
    "! (sizewrbe:time->wordn) hlda be_out .
    BE_Out_Logic_SPEC sizewrbe hlda be_out =
       !t:time .
          ((hlda t) ==> (be_out t = SUBARRAY (sizewrbe t) (9,6)))"
   );;

%-----------------------------------------------------------------------------------------------------------
   Generation logic for write signal.
-----------------------------------------------------------------------------------------------------------%

let Write_Logic_SPEC = new_definition
   ('Write_Logic_SPEC',
    "! clkA clkB (iad_in:time->wordn) sizewrbe cale_ master_tran C_wr write .
    Write_Logic_SPEC clkA clkB iad_in sizewrbe cale_ master_tran C_wr write =
       !t:time .
          ((clkA t) ==> C_wr (t+1) = C_wr t) ∧
          ((clkB t) ==> C_wr (t+1) = (~cale_ t) => (ELEMENT (iad_in t) (27)) | C_wr t) ∧
          (write t = (master_tran t) => (C_wr (t+1)) | (ELEMENT (sizewrbe t) (5)))"
   );;

%-----------------------------------------------------------------------------------------------------------
   C-Bus output latches.
-----------------------------------------------------------------------------------------------------------%

let CB_Out_Logic_SPEC = new_definition
   ('CB_Out_Logic_SPEC',
    "! rep clkA clkB (iad_in:time->wordn) (ccr:time->wordn) dfsm_cout_0_le cout_1_le mfsm_mrequest cout_sel cad_preout
       C_iad_in C_a1a0 C_a3a2 .
    CB_Out_Logic_SPEC rep clkA clkB iad_in ccr dfsm_cout_0_le cout_1_le mfsm_mrequest cout_sel cad_preout
                              C_iad_in C_a1a0 C_a3a2 =
       !t:time .
          ((clkA t) ==>
             ((C_iad_in (t+1) = C_iad_in t) ∧
            (C_a1a0 (t+1) = (cout_1_le t) => (C_iad_in t) | (C_a1a0 t)) Λ
            (C_a3a2 (t+1) = (mfsm_mrequest t) => (ccr t) | (C_a3a2 t)))) Λ
        ((clkB t) ==>
           ((C_iad_in (t+1) = (dfsm_cout_0_le t) => (iad_in t) | (C_iad_in t)) Λ
            (C_a1a0 (t+1) = C_a1a0 t) Λ
            (C_a3a2 (t+1) = C_a3a2 t))) Λ
        (cad_preout t = ((cout_sel (t+1)) = WORDN 0) => (Par_Enc rep (SUBARRAY (C_a1a0 (t+1)) (15,0))) |
                        ((cout_sel (t+1)) = WORDN 1) => (Par_Enc rep (SUBARRAY (C_a1a0 (t+1)) (31,16))) |
                        ((cout_sel (t+1)) = WORDN 2) => (Par_Enc rep (SUBARRAY (C_a3a2 (t+1)) (15,0))) |
                                                        (Par_Enc rep (SUBARRAY (C_a3a2 (t+1)) (31,16))))"
   );;

%---------------------------------------------------------------------------------------------------------
   C-Port Block.
---------------------------------------------------------------------------------------------------------%

let C_Block_SPEC = new_definition
   ('C_Block_SPEC',
    "! (C_mfsm_stateA C_mfsm_state :time->cmfsm_ty)
       (C_sfsm_stateA C_sfsm_state :time->csfsm_ty)
       (C_efsm_stateA C_efsm_state :time->cefsm_ty)
       (C_mfsm_ms C_sfsm_ss C_ssA C_iad_out C_a1a0 C_a3a2 C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss
        C_source C_data_in C_iad_in :time->wordn)
       (C_mfsm_mabort C_mfsm_midle C_mfsm_mrequest C_mfsm_ma3 C_mfsm_ma2 C_mfsm_ma1
        C_mfsm_ma0 C_mfsm_md1 C_mfsm_md0 C_mfsm_iad_en_m C_mfsm_m_cout_sel1 C_mfsm_m_cout_sel0
        C_mfsm_rqt_ C_mfsm_cgnt_ C_mfsm_cm_en C_mfsm_abort_le_en_ C_mfsm_mparity
        C_sfsm_iad_en_s C_sfsm_sidle C_sfsm_slock C_sfsm_sa1 C_sfsm_sa0
        C_sfsm_sale C_sfsm_sd1 C_sfsm_sd0 C_sfsm_sack C_sfsm_sabort C_sfsm_s_cout_sel0 C_sfsm_sparity
        C_efsm_srdy_en
        C_clkAA C_sidle_delA C_mrqt_delA C_last_inA_ C_holdA_ C_rd_srdy C_cout_0_le_delA
        C_cin_2_leA C_mrdy_delA_ C_iad_en_s_delA C_wrdyA C_rrdyA
        C_mfsm_srdy_en C_mfsm_D C_mfsm_grant C_mfsm_rst C_mfsm_busy C_mfsm_write
        C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_last_ C_mfsm_lock_ C_mfsm_invalid
        C_sfsm_D C_sfsm_grant C_sfsm_rst C_sfsm_write C_sfsm_addressed C_sfsm_hlda_
        C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
        C_wr C_clkA C_sidle_del C_mrqt_del C_last_in_ C_lock_in_ C_last_out_
        C_hold_ C_cout_0_le_del C_cin_2_le C_mrdy_del_ C_iad_en_s_del C_wrdy
        C_rrdy C_parity :time->bool)
       (I_mrdy_in_ I_rale_in_ I_male_in_ I_last_in_ I_srdy_in_ I_lock_ I_cale_ I_hlda_ I_crqt_
        Rst ClkA ClkB ClkD Pmm_failure Piu_invalid Reset_error :time->bool)
       (I_ad_in I_be_in_ CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID Ccr :time->wordn)
       (I_cgnt_ I_mrdy_out_ I_hold_ I_rale_out_ I_male_out_ I_last_out_ I_srdy_out_ CB_rqt_out_
        Disable_writes CB_parity :time->bool)
       (I_ad_out I_be_out_ CB_ms_out CB_ss_out CB_ad_out C_ss_out :time->wordn)
       (rep:^rep_ty) .
     C_Block_SPEC (C_mfsm_stateA, C_mfsm_mabort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
                   C_mfsm_ma1, C_mfsm_ma0, C_mfsm_md1, C_mfsm_md0, C_mfsm_iad_en_m, C_mfsm_m_cout_sel1,
                   C_mfsm_m_cout_sel0, C_mfsm_ms, C_mfsm_rqt_, C_mfsm_cgnt_, C_mfsm_cm_en,
                   C_mfsm_abort_le_en_, C_mfsm_mparity,
                   C_sfsm_stateA, C_sfsm_ss, C_sfsm_iad_en_s, C_sfsm_sidle, C_sfsm_slock, C_sfsm_sa1,
                   C_sfsm_sa0, C_sfsm_sale, C_sfsm_sd1, C_sfsm_sd0, C_sfsm_sack, C_sfsm_sabort,
                   C_sfsm_s_cout_sel0, C_sfsm_sparity, C_efsm_stateA, C_efsm_srdy_en,
                   C_clkAA, C_sidle_delA, C_mrqt_delA, C_last_inA_, C_ssA, C_holdA_, C_rd_srdy,
                   C_cout_0_le_delA, C_cin_2_leA, C_mrdy_delA_, C_iad_en_s_delA, C_wrdyA, C_rrdyA, C_iad_out,
            C_a1a0, C_a3a2,
            C_mfsm_state, C_mfsm_srdy_en, C_mfsm_D, C_mfsm_grant, C_mfsm_rst, C_mfsm_busy,
            C_mfsm_write, C_mfsm_crqt_, C_mfsm_hold_, C_mfsm_last_, C_mfsm_lock_, C_mfsm_ss,
            C_mfsm_invalid,
            C_sfsm_state, C_sfsm_D, C_sfsm_grant, C_sfsm_rst, C_sfsm_write, C_sfsm_addressed,
            C_sfsm_hlda_, C_sfsm_ms,
            C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_,
            C_efsm_rst,
            C_wr, C_sizewrbe, C_clkA, C_sidle_del, C_mrqt_del, C_last_in_, C_lock_in_, C_ss,
            C_last_out_, C_hold_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_wrdy,
            C_rrdy, C_parity, C_source, C_data_in, C_iad_in)

            (I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_,
            I_lock_, I_cale_, I_hlda_, I_crqt_,
            CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in,
            Rst, ClkA, ClkB, ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr, Reset_error)

            (I_cgnt_, I_mrdy_out_, I_hold_, I_rale_out_, I_male_out_, I_last_out_, I_srdy_out_,
            I_ad_out, I_be_out_,
            CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out, C_ss_out, Disable_writes, CB_parity)

            rep =

? (grant busy mfsm_mabort mfsm_midle mfsm_mrequest mfsm_ma3 mfsm_ma2 mfsm_ma1 mfsm_ma0
  mfsm_md1 mfsm_md0 mfsm_iad_en_m mfsm_m_cout_sel1 mfsm_m_cout_sel0 mfsm_cm_en
  mfsm_abort_le_en_ mfsm_mparity sfsm_iad_en_s sfsm_sidle sfsm_slock sfsm_sa1 sfsm_sa0
  sfsm_sale sfsm_sd1 sfsm_sd0 sfsm_sack sfsm_sabort sfsm_s_cout_sel0 sfsm_sparity
  efsm_srdy_en dfsm_master dfsm_slave dfsm_cin_0_le dfsm_cin_1_le dfsm_cin_3_le
  dfsm_cin_4_le dfsm_cout_0_le dfsm_cout_1_le dfsm_cad_en_ dfsm_male_ dfsm_rale_
  dfsm_mrdy_ last_in_inE last_in_outQ lock_in_inE lock_in_outQ clkA_outQ
  last_out_inS last_out_inR last_out_inE last_out_outQ sstatus_en_ sidle_del_outQ
  mrqt_del_outQ mstatus_en_ dfsm_srdy write wrdy_inD wrdy_outQ rrdy_inD rrdy_outQ
  wrdyA_outQ rrdyA_outQ i_srdy_en isrdy_inD isrdy_inE cout_0_le_del_out cin_2_le_out
  cout_1_le mrdy_del_out iad_en_s_del_outQ iad_en c_pe_cnt addressed cin_2_le
  cad_in_det c_parity_inS c_parity_inR c_parity_inE hlda :time->bool)
 (mfsm_ss mfsm_ms sfsm_ss cout_sel cad_in_dec source sizewrbe iad_preout cad_preout :time->wordn) .

(OR2_SPEC Rst mfsm_ma1 lock_in_inE) ∧
(DRELAT_SPEC I_lock_ Rst lock_in_inE ClkB C_lock_in_ lock_in_outQ) ∧
(Last_Logic Rst ClkD mfsm_md1 mfsm_mabort last_in_inE) ∧
(DREFF_SPEC I_last_in_ last_in_inE Rst ClkB C_last_inA_ C_last_in_ last_in_outQ) ∧
(DEFFn_SPEC mfsm_ss mfsm_abort_le_en_ ClkB C_ssA C_ss C_ss_out) ∧
(DFF_SPEC ClkD ClkA C_clkA C_clkAA clkA_outQ) ∧
(Hold_Logic CB_ms_in ClkD sfsm_sa1 last_out_inS last_out_inR last_out_inE) ∧
(DSRELAT_SPEC GND last_out_inS last_out_inR last_out_inE ClkB C_last_out_ last_out_outQ) ∧
(TRIBUF_SPEC last_out_outQ hlda I_last_out_) ∧
(OR2_SPEC sfsm_sidle sfsm_sabort sstatus_en_) ∧
(DFF_SPEC sfsm_sidle ClkA C_sidle_del C_sidle_delA sidle_del_outQ) ∧
(DFF_SPEC mfsm_mrequest ClkA C_mrqt_del C_mrqt_delA mrqt_del_outQ) ∧
(Cout_Sel_Logic_SPEC sfsm_s_cout_sel0 mfsm_m_cout_sel1 mfsm_m_cout_sel0 sfsm_sd0 sfsm_sd1 cout_sel) ∧
(NOT_SPEC mfsm_cm_en mstatus_en_) ∧
(DEFF_SPEC sfsm_sidle ClkD ClkA C_hold_ C_holdA_ I_hold_) ∧
(Srdy_In_Logic_SPEC CB_ss_in dfsm_srdy) ∧
(Rdy_Logic_SPEC mfsm_md0 mfsm_md1 ClkD write dfsm_srdy wrdy_inD rrdy_inD) ∧
```

```
(DLAT_SPEC wrdy_inD ClkB C_wrdy wrdy_outQ) ∧
(DLAT_SPEC rrdy_inD ClkB C_rrdy rrdy_outQ) ∧
(DLAT_SPEC wrdy_outQ ClkA C_wrdyA wrdyA_outQ) ∧
(DLAT_SPEC rrdy_outQ ClkA C_rrdyA rrdyA_outQ) ∧
(ISrdy_Out_Logic_SPEC wrdyA_outQ rrdyA_outQ mfsm_mabort I_cale_ i_srdy_en isrdy_inD isrdy_inE) ∧
(TRIBUF_SPEC isrdy_inD isrdy_inE I_srdy_out_) ∧
(CBss_Out_Logic_SPEC sfsm_ss Pmm_failure Piu_invalid CB_ss_out) ∧
(DFF_SPEC dfsm_cout_0_le ClkA C_cout_0_le_del C_cout_0_le_delA cout_0_le_del_out) ∧
(DFF_SPEC dfsm_cin_0_le ClkA C_cin_2_le C_cin_2_leA cin_2_le_out) ∧
(Cout_1_Le_Logic_SPEC dfsm_master cout_0_le_del_out dfsm_cout_1_le cout_1_le) ∧
(DFF_SPEC dfsm_mrdy_ ClkA C_mrdy_del_ C_mrdy_delA_ mrdy_del_out) ∧
(NOT_SPEC I_hlda_ hlda) ∧
(TRIBUF_SPEC dfsm_male_ hlda I_male_out_) ∧
(TRIBUF_SPEC dfsm_rale_ hlda I_rale_out_) ∧
(TRIBUF_SPEC mrdy_del_out hlda I_mrdy_out_) ∧
(DEFF_SPEC sfsm_iad_en_s ClkD ClkA C_iad_en_s_del C_iad_en_s_delA iad_en_s_del_outQ) ∧
(Iad_En_Logic_SPEC mfsm_iad_en_m sfsm_iad_en_s iad_en_s_del_outQ iad_en) ∧
(CBms_Out_Logic_SPEC mfsm_ms Pmm_failure Piu_invalid CB_ms_out) ∧
(Pe_Cnt_Logic_SPEC ClkD sfsm_sparity mfsm_mparity CB_ss_in c_pe_cnt) ∧
(Grant_Logic_SPEC Id CB_rqt_in_ busy grant) ∧
(Addressed_Logic_SPEC Id C_source addressed) ∧
(D_Writes_Logic_SPEC dfsm_slave ChannelID C_source Disable_writes) ∧
(Parity_Decode_Logic_SPEC rep CB_ad_in cad_in_dec cad_in_det) ∧
(Parity_Signal_Inputs_SPEC Rst cad_in_det ClkD c_pe_cnt Reset_error
                        c_parity_inS c_parity_inR c_parity_inE) ∧
(DSRELAT_SPEC GND c_parity_inS c_parity_inR c_parity_inE ClkB C_parity CB_parity) ∧
(CB_In_Latches_SPEC ClkA ClkB Rst cad_in_dec dfsm_cin_0_le dfsm_cin_1_le cin_2_le dfsm_cin_3_le
                    dfsm_cin_4_le source sizewrbe iad_preout
                    C_source C_data_in C_sizewrbe C_iad_out) ∧
(BE_Out_Logic_SPEC sizewrbe hlda I_be_out_) ∧
(TRIBUF_SPEC iad_preout iad_en I_ad_out) ∧
(Write_Logic_SPEC ClkA ClkB I_ad_in sizewrbe I_cale_ mfsm_cm_en C_wr write) ∧
(CB_Out_Logic_SPEC rep ClkA ClkB I_ad_in Ccr dfsm_cout_0_le cout_1_le mfsm_mrequest cout_sel cad_preout
                    C_iad_in C_a1a0 C_a3a2 ) ∧
(TRIBUF_SPEC cad_preout dfsm_cad_en_ CB_ad_out) ∧
(CMFSM_SPEC ClkA ClkB efsm_srdy_en ClkD grant Rst busy write
            I_crqt_ I_hold_ last_in_outQ lock_in_outQ CB_ss_in Piu_invalid
            C_mfsm_state C_mfsm_srdy_en C_mfsm_D C_mfsm_grant C_mfsm_rst C_mfsm_busy C_mfsm_write
            C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_last_ C_mfsm_lock_ C_mfsm_ss C_mfsm_invalid
            C_mfsm_stateA C_mfsm_mabort C_mfsm_midle C_mfsm_mrequest C_mfsm_ma3 C_mfsm_ma2
            C_mfsm_ma1 C_mfsm_ma0 C_mfsm_md1 C_mfsm_md0 C_mfsm_iad_en_m C_mfsm_m_cout_sel1
            C_mfsm_m_cout_sel0 C_mfsm_ms C_mfsm_rqt_ C_mfsm_cgnt_ C_mfsm_cm_en
            C_mfsm_abort_le_en_ C_mfsm_mparity
            mfsm_mabort mfsm_midle mfsm_mrequest mfsm_ma3 mfsm_ma2 mfsm_ma1 mfsm_ma0
            mfsm_md1 mfsm_md0 mfsm_iad_en_m mfsm_m_cout_sel1 mfsm_m_cout_sel0 mfsm_ms
            CB_rqt_out_ I_cgnt_ mfsm_cm_en mfsm_abort_le_en_ mfsm_mparity) ∧
(CSFSM_SPEC ClkA ClkB ClkD grant Rst write addressed I_hlda_ CB_ms_in
           C_sfsm_state C_sfsm_D C_sfsm_grant C_sfsm_rst C_sfsm_write C_sfsm_addressed
           C_sfsm_hlda_ C_sfsm_ms C_sfsm_stateA C_sfsm_ss C_sfsm_iad_en_s C_sfsm_sidle
           C_sfsm_slock C_sfsm_sa1 C_sfsm_sa0 C_sfsm_sale C_sfsm_sd1 C_sfsm_sd0 C_sfsm_sack
           C_sfsm_sabort C_sfsm_s_cout_sel0 C_sfsm_sparity
           sfsm_ss sfsm_iad_en_s sfsm_sidle sfsm_slock sfsm_sa1 sfsm_sa0 sfsm_sale
           sfsm_sd1 sfsm_sd0 sfsm_sack sfsm_sabort sfsm_s_cout_sel0 sfsm_sparity) ∧
(CEFSM_SPEC ClkA ClkB I_cale_ I_last_in_ I_male_in_ I_rale_in_ I_srdy_in_ Rst
    C_efsm_state C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
                 C_efsm_stateA C_efsm_srdy_en efsm_srdy_en) /\
  (CDFSM_SPEC dfsm_srdy ClkD clkA_outQ write sizewrbe sfsm_sidle sidle_del_outQ sfsm_slock
                 sfsm_sa1 sfsm_sa0 sfsm_sale sfsm_sd1 sfsm_sd0 sfsm_sack mfsm_midle mrqt_del_outQ
                 mfsm_ma3 mfsm_ma2 mfsm_ma1 mfsm_ma0 mfsm_md1 mfsm_md0 I_cale_ I_srdy_in_
                 dfsm_master dfsm_slave dfsm_cin_0_le dfsm_cin_1_le dfsm_cin_3_le dfsm_cin_4_le
                 dfsm_cout_0_le dfsm_cout_1_le dfsm_cad_en_ dfsm_male_ dfsm_rale_ dfsm_mrdy_)"
    );;

close_theory();;
```

B.5 SU_Cont Specification

```
%-----------------------------------------------------------------------------------------------------------

    File:         s_block.ml

    Author:       (c) D.A. Fura 1992

    Date:         31 March 1992

    This file contains the ml source for the gate-level specification of the startup controller of
    the FTEP PIU, an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.

-----------------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm s_block.th';;

new_theory 's_block';;

map new_parent ['gates_def';'latches_def';'ffs_def';'counters_def';'saux_def';'aux_def';'array_def';'wordn_def'];;

let s_state_ty = ":(sfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                    bool#bool#wordn#wordn#bool#bool#
                    sfsm_ty#bool#bool#bool#bool#bool#
                    bool#wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let s_state = "((S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_srcp, S_fsm_sdi, S_fsm_srp, S_fsm_src0, S_fsm_src1,
                 S_fsm_spf, S_fsm_sc0f, S_fsm_sc1f, S_fsm_spmf, S_fsm_sb, S_fsm_src, S_fsm_sec, S_fsm_srs,
                 S_fsm_scs, S_soft_shot, S_soft_shot_delA, S_soft_cntA, S_delayA, S_instart, S_cpu_histA,
                 S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
                 S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
                 S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_cpu_hist, S_piu_fail)
               :^s_state_ty)";;

let s_env_ty = ":(bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let s_env = "((ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_)
             :^s_env_ty)";;

let s_out_ty = ":(wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let s_out = "((S_state, Reset_cport, Disable_int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist,
             Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail)
            :^s_out_ty)";;

%-----------------------------------------------------------------------------------------------------------
    Input logic for S_soft_shot latch.
-----------------------------------------------------------------------------------------------------------%

let Scnt_In_SPEC = new_definition
    ('Scnt_In_SPEC',
    "! gcrh gcrl soft_shot_inD soft_cnt_inL .
    Scnt_In_SPEC gcrh gcrl soft_shot_inD soft_cnt_inL =
       (! t:time . (soft_shot_inD t = ~gcrh t /\ gcrl t) /\
                    (soft_cnt_inL t = ~gcrh t ∧ ~gcrl t))"
);;

%------------------------------------------------------------------------------------------------------
   Input logic for S_soft_cnt counter.
------------------------------------------------------------------------------------------------------%

let Scnt_In1_SPEC = new_definition
   ('Scnt_In1_SPEC',
    "! soft_shot_outQ soft_shot_del_outQ soft_cnt_inU .
    Scnt_In1_SPEC soft_shot_outQ soft_shot_del_outQ soft_cnt_inU =
       (! t:time . (soft_cnt_inU t = soft_shot_outQ t ∧ ~soft_shot_del_outQ t))"
   );;

%------------------------------------------------------------------------------------------------------
   Input logic for S_delay counter.
------------------------------------------------------------------------------------------------------%

let Delay_In_SPEC = new_definition
   ('Delay_In_SPEC',
    "! scpustart delay reset_cnt delay_inR .
    Delay_In_SPEC scpustart delay reset_cnt delay_inR =
       (! t:time . (delay_inR t = scpustart t ∧ (ELEMENT (delay t) (6)) ∨ reset_cnt t))"
   );;

%------------------------------------------------------------------------------------------------------
   Delay counter output multiplexers.
------------------------------------------------------------------------------------------------------%

let Muxes_SPEC = new_definition
   ('Muxes_SPEC',
    "! (delay:time->wordn) test instart_inD delay17 .
    Muxes_SPEC delay test instart_inD delay17 =
       (!t:time . (instart_inD t = (test t) => ELEMENT (delay t) (5) | ELEMENT (delay t) (16)) ∧
                  (delay17 t = (test t) => ELEMENT (delay t) (6) | ELEMENT (delay t) (17)))"
   );;

%------------------------------------------------------------------------------------------------------
   Generation logic for Disable_int output.
------------------------------------------------------------------------------------------------------%

let Dis_Int_Out_SPEC = new_definition
   ('Dis_Int_Out_SPEC',
    "! instart normal delay disable_int_in disable_int_out .
    Dis_Int_Out_SPEC instart normal delay disable_int_in disable_int_out =
       (! t:time . (disable_int_out t = ~instart t ∧ ~(normal t ∧ (ELEMENT (delay t) (6)) ∧ disable_int_in t)))"
   );;

%------------------------------------------------------------------------------------------------------
   Input logic for S_bad_cpu0, S_bad_cpu1 latches.
------------------------------------------------------------------------------------------------------%

let Bad_Cpu_In_SPEC = new_definition
   ('Bad_Cpu_In_SPEC',
    "! normal operation cpu0_fail cpu1_fail begin
      bad_cpu0_inS bad_cpu0_inR bad_cpu0_inE
      bad_cpu1_inS bad_cpu1_inR bad_cpu1_inE .
    Bad_Cpu_In_SPEC normal operation cpu0_fail cpu1_fail begin
                        bad_cpu0_inS bad_cpu0_inR bad_cpu0_inE
                        bad_cpu1_inS bad_cpu1_inR bad_cpu1_inE =
       (! t:time . (bad_cpu0_inS t = begin t) ∧
                   (bad_cpu0_inR t = (normal t ∨ operation t) ∧ ~cpu0_fail t) ∧
                   (bad_cpu0_inE t = begin t ∨ (normal t ∨ operation t) ∧ ~cpu0_fail t) ∧
                   (bad_cpu1_inS t = begin t) ∧
                   (bad_cpu1_inR t = (normal t ∨ operation t) ∧ cpu0_fail t ∧ ~cpu1_fail t) ∧
                   (bad_cpu1_inE t = begin t ∨ (normal t ∨ operation t) ∧ cpu0_fail t ∧ ~cpu1_fail t))"
    );;

%------------------------------------------------------------------------------------------------
    Generation logic for local signals cpu0_ok, cpu1_ok.
------------------------------------------------------------------------------------------------%

let Cpu_Ok_SPEC = new_definition
    ('Cpu_Ok_SPEC',
    "! soft_cnt cpu0_fail cpu1_fail failure0_ failure1_ cpu0_ok cpu1_ok .
    Cpu_Ok_SPEC soft_cnt cpu0_fail cpu1_fail failure0_ failure1_ cpu0_ok cpu1_ok =
       (! t:time . (cpu0_ok t = ((soft_cnt t) = WORDN 5) ∧ cpu0_fail t ∧ failure0_ t) ∧
                   (cpu1_ok t = ((soft_cnt t) = WORDN 5) ∧ cpu1_fail t ∧ failure1_ t))"
    );;

%------------------------------------------------------------------------------------------------
    Input logic for S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_piu_fail latches.
------------------------------------------------------------------------------------------------%

let Fail_In_SPEC = new_definition
    ('Fail_In_SPEC',
    "! begin pmm_fail piu_fail bypass cpu0_ok cpu1_ok
      pmm_fail_inS pmm_fail_inR pmm_fail_inE cpu0_fail_inS cpu0_fail_inR cpu0_fail_inE
      cpu1_fail_inS cpu1_fail_inR cpu1_fail_inE piu_fail_inS piu_fail_inR piu_fail_inE .
    Fail_In_SPEC begin pmm_fail piu_fail bypass cpu0_ok cpu1_ok
                   pmm_fail_inS pmm_fail_inR pmm_fail_inE cpu0_fail_inS cpu0_fail_inR cpu0_fail_inE
                   cpu1_fail_inS cpu1_fail_inR cpu1_fail_inE piu_fail_inS piu_fail_inR piu_fail_inE =
       (! t:time . (pmm_fail_inS t = begin t) ∧
                   (pmm_fail_inR t = pmm_fail t) ∧
                   (pmm_fail_inE t = begin t ∨ pmm_fail t) ∧
                   (cpu0_fail_inS t = begin t) ∧
                   (cpu0_fail_inR t = bypass t ∨ cpu0_ok t) ∧
                   (cpu0_fail_inE t = begin t ∨ bypass t ∨ cpu0_ok t) ∧
                   (cpu1_fail_inS t = begin t) ∧
                   (cpu1_fail_inR t = bypass t ∨ cpu1_ok t) ∧
                   (cpu1_fail_inE t = begin t ∨ bypass t ∨ cpu1_ok t) ∧
                   (piu_fail_inS t = begin t) ∧
                   (piu_fail_inR t = bypass t ∨ piu_fail t) ∧
                   (piu_fail_inE t = begin t ∨ bypass t ∨ piu_fail t))"
    );;

%------------------------------------------------------------------------------------------------
    Startup-controller controller state machine.
```

```
--------------------------------------------------------------------------------------------------------------%

let FSM_SPEC = new_definition
    ('FSM_SPEC',
    "! clkA clkB rst_in delay_in delay17_in bothbad_in bypass_in
       state rst delay6 delay17 bothbad bypass
       stateA sn so srcp sdi srp src0 src1 spf sc0f sc1f spmf sb src sec srs scs
       stateA_out sn_out so_out srcp_out sdi_out srp_out src0_out src1_out spf_out
       sc0f_out sc1f_out spmf_out sb_out src_out sec_out srs_out scs_out .
    FSM_SPEC clkA clkB rst_in delay_in delay17_in bothbad_in bypass_in
                state rst delay6 delay17 bothbad bypass
                stateA sn so srcp sdi srp src0 src1 spf sc0f sc1f spmf sb src sec srs scs
                stateA_out sn_out so_out srcp_out sdi_out srp_out src0_out src1_out spf_out
                sc0f_out sc1f_out spmf_out sb_out src_out sec_out srs_out scs_out =

       !t:time.

       ((clkA t) ==>
        ((state (t+1) = state t) ∧
         (rst (t+1) = rst t) ∧
         (delay6 (t+1) = delay6 t) ∧
         (delay17 (t+1) = delay17 t) ∧
         (bothbad (t+1) = bothbad t) ∧
         (bypass (t+1) = bypass t) ∧
         (stateA (t+1) =
               ((rst t) => SSTART |
               ((state t) = SSTART) => SRA |
               ((state t) = SRA) => ((delay6 t) => ((bypass t) => SO | SPF) | SRA) |
               ((state t) = SPF) => SC0I |
               ((state t) = SC0I) => ((delay17 t) => SC0F | SC0I) |
               ((state t) = SC0F) => ST |
               ((state t) = ST) => SC1I |
               ((state t) = SC1I) => ((delay17 t) => SC1F | SC1I) |
               ((state t) = SC1F) => SS |
               ((state t)= SS) => ((bothbad t) => SSTOP | SCS) |
               ((state t) = SSTOP) => SSTOP |
               ((state t) = SCS) => ((delay6 t) => SN | SCS) |
               ((state t) = SN) => ((delay17 t) => SO | SN) | SO)) ∧
         (sn (t+1) = (stateA (t+1) = SN)) ∧
         (so (t+1) = (stateA (t+1) = SO)) ∧
         (srcp (t+1) = ((~(stateA (t+1) = SO) ∧ ~((state t) = SSTOP)) ∨ ((state t) = SRA))) ∧
         (sdi (t+1) = ((~(stateA (t+1) = SO) ∧ ~((state t) = SSTOP)) ∨ ((state t) = SRA))) ∧
         (srp (t+1) = ((stateA (t+1) = SSTART) ∨ (stateA (t+1) = SRA) ∨ (stateA (t+1) = SC0F) ∨
                       (stateA (t+1) = ST) ∨ (stateA (t+1) = SC1F) ∨ (stateA (t+1) = SS) ∨
                       (stateA (t+1) = SCS))) ∧
         (src0 (t+1) = (~(stateA (t+1) = SPF) ∧ ~(stateA (t+1) = SC0I))) ∧
         (src1 (t+1) = (~(stateA (t+1) = ST) ∧ ~(stateA (t+1) = SC1I))) ∧
         (spf (t+1) = (((state t) = SRA) ∧ (delay6 t) ∧ ~(rst t))) ∧
         (sc0f (t+1) = (stateA (t+1) = SC0F)) ∧
         (sc1f (t+1) = (stateA (t+1) = SC1F)) ∧
         (spmf (t+1) = (stateA (t+1) = SO)) ∧
         (sb (t+1) = (stateA (t+1) = SSTART)) ∧
         (src (t+1) = ((stateA (t+1) = SSTART) ∨ (((state t) = SRA) ∧ (delay6 t)) ∨
                       (stateA (t+1) = SC0F) ∨ (stateA (t+1) = ST) ∨ (stateA (t+1) = SC1F) ∨
                (stateA (t+1) = SS) ∨ (((state t) = SCS) ∧ (delay6 t)))) ∧
   (sec (t+1) = ((~(stateA (t+1) = SSTOP) ∧ ~(stateA (t+1) = SO)) ∨ ((state t) = SN))) ∧
   (srs (t+1) = ((((state t) = SPF) ∧ ~(rst t)) ∨ (((state t) = ST) ∧ ~(rst t)))) ∧
   (scs (t+1) = (stateA (t+1) = SCS)))) ∧
  ((clkB t) ==>
  ((state (t+1) = stateA t) ∧
   (rst (t+1) = rst_in t) ∧
   (delay6 (t+1) = ELEMENT (delay_in t) (6)) ∧
   (delay17 (t+1) = delay17_in t) ∧
   (bothbad (t+1) = bothbad_in t) ∧
   (bypass (t+1) = bypass_in t) ∧
   (sn (t+1) = sn t) ∧
   (so (t+1) = so t) ∧
   (srcp (t+1) = srcp t) ∧
   (sdi (t+1) = sdi t) ∧
   (srp (t+1) = srp t) ∧
   (src0 (t+1) = src0 t) ∧
   (src1 (t+1) = src1 t) ∧
   (spf (t+1) = spf t) ∧
   (sc0f (t+1) = sc0f t) ∧
   (sc1f (t+1) = sc1f t) ∧
   (spmf (t+1) = spmf t) ∧
   (sb (t+1) = sb t) ∧
   (src (t+1) = src t) ∧
   (sec (t+1) = sec t) ∧
   (srs (t+1) = srs t) ∧
   (scs (t+1) = scs t))) ∧
   ((let a0 = (ALTER (stateA_out t) (0)
                    ((stateA (t+1) = SRA) ∨ (stateA (t+1) = SPF) ∨ (stateA (t+1) = ST) ∨
                     (stateA (t+1) = SC1I) ∨ (stateA (t+1) = SCS) ∨ (stateA (t+1) = SN) ∨
                     (stateA (t+1) = SO)))
    in
    (let a1 = (ALTER a0 (1)
                    ((stateA (t+1) = SPF) ∨ (stateA (t+1) = SC0I) ∨ (stateA (t+1) = SC0F) ∨
                     (stateA (t+1) = ST) ∨ (stateA (t+1) = SSTOP) ∨ (stateA (t+1) = SO)))
    in
    (let a2 = (ALTER a1 (2)
                    ((stateA (t+1) = SC0F) ∨ (stateA (t+1) = ST) ∨ (stateA (t+1) = SC1I) ∨
                     (stateA (t+1) = SC1F) ∨ (stateA (t+1) = SS) ∨ (stateA (t+1) = SSTOP) ∨
                     (stateA (t+1) = SCS)))
    in
    (let a3 = (ALTER a2 (3)
                    ((stateA (t+1) = SS) ∨ (stateA (t+1) = SSTOP) ∨ (stateA (t+1) = SCS) ∨
                     (stateA (t+1) = SN) ∨ (stateA (t+1) = SO)))
    in
   (stateA_out t = a3))))) ∧
    (sn_out t = sn (t+1)) ∧
    (so_out t = so (t+1)) ∧
    (srcp_out t = srcp (t+1)) ∧
    (sdi_out t = sdi (t+1)) ∧
    (srp_out t = srp (t+1)) ∧
    (src0_out t = src0 (t+1)) ∧
    (src1_out t = src1 (t+1)) ∧
    (spf_out t = spf (t+1)) ∧
```

```
        (sc0f_out t = sc0f (t+1)) ∧
        (sc1f_out t = sc1f (t+1)) ∧
        (spmf_out t = spmf (t+1)) ∧
        (sb_out t = sb (t+1)) ∧
        (src_out t = src (t+1)) ∧
        (sec_out t = sec (t+1)) ∧
        (srs_out t = srs (t+1)) ∧
        (scs_out t = scs (t+1)))
");;

%------------------------------------------------------------------------------------------------
   Startup controller block.
------------------------------------------------------------------------------------------------%

let S_Block_SPEC = new_definition
   ('S_Block_SPEC',
   "! (S_fsm_stateA S_fsm_state :(time->sfsm_ty))
     (S_soft_cntA S_delayA S_soft_cnt S_delay :(time->wordn))
     (S_fsm_sn S_fsm_so S_fsm_srcp S_fsm_sdi S_fsm_srp S_fsm_src0 S_fsm_src1 S_fsm_spf S_fsm_sc0f
      S_fsm_sc1f S_fsm_spmf S_fsm_sb S_fsm_src S_fsm_sec S_fsm_srs S_fsm_scs
      S_soft_shot S_soft_shot_delA S_instart S_cpu_histA
      S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass
      S_soft_shot_del S_bad_cpu0 S_bad_cpu1 S_reset_cpu0 S_reset_cpu1 S_pmm_fail S_cpu0_fail S_cpu1_fail
      S_piu_fail S_cpu_hist :(time->bool))
     (ClkA ClkB Rst Bypass Test Gcrh Gcrl Failure0_ Failure1_ :(time->bool))
     (S_state :(time->wordn))
     (Reset_cport Disable_int Reset_piu Reset_cpu0 Reset_cpu1 Cpu_hist Piu_fail Cpu0_fail Cpu1_fail
      Pmm_fail :(time->bool)) .
    S_Block_SPEC (S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_srcp, S_fsm_sdi, S_fsm_srp, S_fsm_src0, S_fsm_src1,
                  S_fsm_spf, S_fsm_sc0f, S_fsm_sc1f, S_fsm_spmf, S_fsm_sb, S_fsm_src, S_fsm_sec, S_fsm_srs,
                  S_fsm_scs, S_soft_shot, S_soft_shot_delA, S_soft_cntA, S_delayA, S_instart, S_cpu_histA,
                  S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
                  S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
                  S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_cpu_hist, S_piu_fail)
                 (ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_)
                 (S_state, Reset_cport, Disable_int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist,
                  Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail) =
    (!t:time .
       ? fsm_delay17 fsm_bothbad
         fsm_sn fsm_so fsm_sdi fsm_src0 fsm_src1 fsm_spf fsm_sc0f fsm_sc1f fsm_spmf fsm_sb
         fsm_src fsm_sec fsm_srs fsm_scs NC
         soft_shot_inD soft_shot_outQ soft_shot_del_outQ
         soft_cnt_inL soft_cnt_inU soft_cnt_inR soft_cnt_outQ
         delay_inL delay_inU delay_inR delay_outQ instart_inD instart_outQ
         bad_cpu0_inS bad_cpu0_inR bad_cpu0_inE bad_cpu0_outQ reset_cpu0_inD
         bad_cpu1_inS bad_cpu1_inR bad_cpu1_inE bad_cpu1_outQ reset_cpu1_inD cpu_hist_inD
         cpu0_ok cpu1_ok
         pmm_fail_inS pmm_fail_inR pmm_fail_inE cpu0_fail_inS cpu0_fail_inR cpu0_fail_inE
         cpu1_fail_inS cpu1_fail_inR cpu1_fail_inE piu_fail_inS piu_fail_inR piu_fail_inE.
     (Scnt_In_SPEC Gcrh Gcrl soft_shot_inD soft_cnt_inL) ∧
     (DLAT_SPEC soft_shot_inD ClkA S_soft_shot soft_shot_outQ) ∧
     (DFF_SPEC soft_shot_outQ ClkA S_soft_shot_del S_soft_shot_delA soft_shot_del_outQ) ∧
     (Scnt_In1_SPEC soft_shot_outQ soft_shot_del_outQ soft_cnt_inU) ∧
     (UPRCNT_SPEC 2 (GNDN 2) soft_cnt_inL soft_cnt_inU soft_cnt_inR ClkA S_soft_cnt S_soft_cntA
                    soft_cnt_outQ NC) ∧
        (Delay_In_SPEC fsm_scs delay_outQ fsm_src delay_inR) ∧
        (UPRCNT_SPEC 17 (GNDN 17) delay_inL delay_inU delay_inR ClkA S_delay S_delayA delay_outQ NC) ∧
        (Muxes_SPEC delay_outQ Test instart_inD fsm_delay17) ∧
        (DLAT_SPEC instart_inD ClkA S_instart instart_outQ) ∧
        (Dis_Int_Out_SPEC instart_outQ fsm_sn delay_outQ fsm_sdi Disable_int) ∧
        (AND2_SPEC Cpu0_fail Cpu1_fail fsm_bothbad) ∧
        (Bad_Cpu_In_SPEC fsm_sn fsm_so Cpu0_fail Cpu1_fail fsm_sb
                    bad_cpu0_inS bad_cpu0_inR bad_cpu0_inE
                    bad_cpu1_inS bad_cpu1_inR bad_cpu1_inE) ∧
        (DSRELAT_SPEC GND bad_cpu0_inS bad_cpu0_inR bad_cpu0_inE ClkB S_bad_cpu0 bad_cpu0_outQ) ∧
        (DSRELAT_SPEC GND bad_cpu1_inS bad_cpu1_inR bad_cpu1_inE ClkB S_bad_cpu1 bad_cpu1_outQ) ∧
        (AND2_SPEC bad_cpu0_outQ fsm_src0 reset_cpu0_inD) ∧
        (AND2_SPEC bad_cpu1_outQ fsm_src1 reset_cpu1_inD) ∧
        (DLAT_SPEC reset_cpu0_inD ClkB S_reset_cpu0 Reset_cpu0) ∧
        (DLAT_SPEC reset_cpu1_inD ClkB S_reset_cpu1 Reset_cpu1) ∧
        (AND3_SPEC Reset_cpu0 Reset_cpu1 Bypass cpu_hist_inD) ∧
        (DFF_SPEC cpu_hist_inD ClkB S_cpu_histA S_cpu_hist Cpu_hist) ∧
        (Fail_In_SPEC fsm_sb fsm_spmf fsm_spf Bypass cpu0_ok cpu1_ok
                pmm_fail_inS pmm_fail_inR pmm_fail_inE cpu0_fail_inS cpu0_fail_inR cpu0_fail_inE
                cpu1_fail_inS cpu1_fail_inR cpu1_fail_inE piu_fail_inS piu_fail_inR piu_fail_inE) ∧
        (DSRELAT_SPEC GND pmm_fail_inS pmm_fail_inR pmm_fail_inE ClkB S_pmm_fail Pmm_fail) ∧
        (DSRELAT_SPEC GND cpu0_fail_inS cpu0_fail_inR cpu0_fail_inE ClkB S_cpu0_fail Cpu0_fail) ∧
        (DSRELAT_SPEC GND cpu1_fail_inS cpu1_fail_inR cpu1_fail_inE ClkB S_cpu1_fail Cpu1_fail) ∧
        (DSRELAT_SPEC GND piu_fail_inS piu_fail_inR piu_fail_inE ClkB S_piu_fail Piu_fail) ∧
        (Cpu_Ok_SPEC soft_cnt_outQ fsm_sc0f fsm_sc1f Failure0_ Failure1_ cpu0_ok cpu1_ok) ∧
        (FSM_SPEC ClkA ClkB Rst delay_outQ fsm_delay17 fsm_bothbad Bypass
                S_fsm_state S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass
                S_fsm_stateA S_fsm_sn S_fsm_so S_fsm_srcp S_fsm_sdi S_fsm_srp S_fsm_src0 S_fsm_src1
                S_fsm_spf S_fsm_sc0f S_fsm_sc1f S_fsm_spmf S_fsm_sb S_fsm_src S_fsm_sec S_fsm_srs
                S_fsm_scs
                S_state fsm_sn fsm_so Reset_cport fsm_sdi Reset_piu fsm_src0 fsm_src1 fsm_spf
                fsm_sc0f fsm_sc1f fsm_spmf fsm_sb fsm_src fsm_sec fsm_srs fsm_scs))"
    );;

close_theory();;
```

Appendix C ML Source for the Phase-Level Specification of the PIU Ports.

This appendix contains the HOL models used in the phase-level specification for the PIU ports. They are listed in the order: P_Port, M_Port, R_Port, C_Port, and SU_Cont.

C.1 P Port Specification

```
%----------------------------------------------------------------------------------------------------------
   File:      p_phase.ml

   Author:    (c) D.A. Fura 1992

   Date:      31 March 1992

   This file contains the ml source for the phase-level specification of the P-Port of the FTEP BIU,
   an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
   The bulk of this code was translated from an M-language simulation program using a translator
   written by P.J. Windley at the University of Idaho.

----------------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm p_phase.th';;

new_theory 'p_phase';;

map new_parent ['paux_def';'aux_def';'array_def';'wordn_def'];;

let p_state_ty = ":(pfsm_ty#bool#bool#bool#wordn#wordn#bool#wordn#bool#wordn#wordn#bool#bool#
                    pfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool#bool)";;
let p_state = "((P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P_dest1, P_be_,
                P_wr, P_be_n_, P_sizeA, P_loadA, P_downA, P_fsm_state, P_fsm_rst, P_fsm_mrqt, P_fsm_sack,
                P_fsm_cgnt_, P_fsm_crqt_, P_fsm_hold_, P_fsm_lock_, P_rqt, P_size, P_load, P_down, P_lock_,
                P_lock_inh_, P_male_, P_rale_)
                :(pfsm_ty#bool#bool#bool#wordn#wordn#bool#wordn#bool#wordn#wordn#bool#bool#
                  pfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool#bool))";;

let p_env_ty = ":(bool#bool#bool#wordn#bool#bool#wordn#bool#bool#wordn#bool#bool#bool)";;
let p_env = "((ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_, I_srdy_)
              :(bool#bool#bool#wordn#bool#bool#wordn#bool#bool#wordn#bool#bool#bool))";;

let p_out_ty = ":(wordn#bool#wordn#wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool)";;
let p_out = "((L_ad_out, L_ready_, I_ad_data_out, I_ad_addr_out, I_be_, I_rale_, I_male_, I_crqt_, I_cale_,
              I_mrdy_, I_last_, I_hlda_, I_lock_)
              :(wordn#bool#wordn#wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool))";;

%----------------------------------------------------------------------------------------------------------
   Next_state definition for Phase-A instruction.
----------------------------------------------------------------------------------------------------------%

let PH_A_inst_def = new_definition
```

```
('PH_A_inst',
"! (P_fsm_state P_fsm_stateA :pfsm_ty)
   (P_fsm_astate P_fsm_dstate P_fsm_hlda_ P_dest1 P_wr P_loadA P_downA :bool)
   (P_fsm_rst P_fsm_mrqt P_fsm_sack P_fsm_cgnt_ P_fsm_crqt_ P_fsm_hold_ P_fsm_lock_ P_rqt P_load :bool)
   (P_down P_lock_ P_lock_inh_ P_male_ P_rale_ :bool)
   (P_wr_data P_addr P_be_ P_be_n_ P_sizeA P_size :wordn)

   (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ I_cgnt_ I_hold_ I_srdy_ :bool) (L_ad_in L_be_ I_ad_in :wordn) .

PH_A_inst (P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P_dest1, P_be_,
           P_wr, P_be_n_, P_sizeA, P_loadA, P_downA, P_fsm_state, P_fsm_rst, P_fsm_mrqt, P_fsm_sack,
           P_fsm_cgnt_, P_fsm_crqt_, P_fsm_hold_, P_fsm_lock_, P_rqt, P_size, P_load, P_down, P_lock_,
           P_lock_inh_, P_male_, P_rale_)

          (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_, I_srdy_) =

let new_P_fsm_stateA =
      ((P_fsm_rst) => PA |
       ((P_fsm_state = PH) => ((P_fsm_hold_) => PA | PH) |
       ((P_fsm_state = PA) =>
            ((P_fsm_mrqt V (~P_fsm_crqt_ Λ ~P_fsm_cgnt_)) => PD |
            ((P_fsm_lock_ Λ ~P_fsm_hold_) => PH | PA)) |
       ((P_fsm_state = PD) =>
            (((P_fsm_sack Λ P_fsm_hold_) V (P_fsm_sack Λ ~P_fsm_hold_ Λ ~P_fsm_lock_)) => PA |
            ((P_fsm_sack Λ ~P_fsm_hold_ Λ P_fsm_lock_) => PH | PD)) | P_ILL)))) in
let new_P_fsm_astate = (new_P_fsm_stateA = PA) in
let new_P_fsm_dstate = (new_P_fsm_stateA = PD) in
let new_P_fsm_hlda_ = ~(new_P_fsm_stateA = PH) in
let new_P_wr_data = L_ad_in in
let new_P_addr = ((~P_rqt) => (SUBARRAY L_ad_in (25,0)) | P_addr) in
let new_P_dest1 = ((~P_rqt) => (ELEMENT L_ad_in (31)) | P_dest1) in
let new_P_be_ = ((~P_rqt) => L_be_ | P_be_) in
let new_P_wr = ((~P_rqt) => L_wr | P_wr) in
let new_P_be_n_ = L_be_ in
let new_P_loadA = P_load in
let new_P_downA = P_down in
let new_P_sizeA = P_size in
let new_P_fsm_state = P_fsm_state in
let new_P_fsm_rst = P_fsm_rst in
let new_P_fsm_mrqt = P_fsm_mrqt in
let new_P_fsm_sack = P_fsm_sack in
let new_P_fsm_cgnt_ = P_fsm_cgnt_ in
let new_P_fsm_crqt_ = P_fsm_crqt_ in
let new_P_fsm_hold_ = P_fsm_hold_ in
let new_P_fsm_lock_ = P_fsm_lock_ in
let new_P_rqt = P_rqt in
let new_P_size = P_size in
let new_P_load = P_load in
let new_P_down = P_down in
let new_P_lock_ = P_lock_ in
let new_P_lock_inh_ = P_lock_inh_ in
let new_P_male_ = P_male_ in
let new_P_rale_ = P_rale_ in
```

```
    (new_P_fsm_stateA, new_P_fsm_astate, new_P_fsm_dstate, new_P_fsm_hlda_, new_P_wr_data, new_P_addr, new_P_dest1,
     new_P_be_, new_P_wr, new_P_be_n_, new_P_sizeA, new_P_loadA, new_P_downA, new_P_fsm_state, new_P_fsm_rst,
     new_P_fsm_mrqt, new_P_fsm_sack, new_P_fsm_cgnt_, new_P_fsm_crqt_, new_P_fsm_hold_, new_P_fsm_lock_,
     new_P_rqt, new_P_size, new_P_load, new_P_down, new_P_lock_, new_P_lock_inh_, new_P_male_, new_P_rale_)"
    );;

%------------------------------------------------------------------------------------------------
    Output definition for Phase-A instruction.
------------------------------------------------------------------------------------------------%

let PH_A_out_def = new_definition
    ('PH_A_out',
    "! (P_fsm_state P_fsm_stateA :pfsm_ty)
       (P_fsm_astate P_fsm_dstate P_fsm_hlda_ P_dest1 P_wr P_loadA P_downA :bool)
       (P_fsm_rst P_fsm_mrqt P_fsm_sack P_fsm_cgnt_ P_fsm_crqt_ P_fsm_hold_ P_fsm_lock_ P_rqt P_load :bool)
       (P_down P_lock_ P_lock_inh_ P_male_ P_rale_ :bool)
       (P_wr_data P_addr P_be_ P_be_n_ P_sizeA P_size :wordn)
       (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ I_cgnt_ I_hold_ I_srdy_ :bool) (L_ad_in L_be_ I_ad_in :wordn) .
     PH_A_out (P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P_dest1, P_be_,
                P_wr, P_be_n_, P_sizeA, P_loadA, P_downA, P_fsm_state, P_fsm_rst, P_fsm_mrqt, P_fsm_sack,
                P_fsm_cgnt_, P_fsm_crqt_, P_fsm_hold_, P_fsm_lock_, P_rqt, P_size, P_load, P_down, P_lock_,
                P_lock_inh_, P_male_, P_rale_)
               (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_, I_srdy_) =

     let new_P_fsm_stateA =
           ((P_fsm_rst) => PA |
            ((P_fsm_state = PH) => ((P_fsm_hold_) => PA | PH) |
            ((P_fsm_state = PA) =>
                 ((P_fsm_mrqt V (~P_fsm_crqt_ ∧ ~P_fsm_cgnt_)) => PD |
                 ((P_fsm_lock_ ∧ ~P_fsm_hold_) => PH | PA)) |
            ((P_fsm_state = PD) =>
                 (((P_fsm_sack ∧ P_fsm_hold_) V (P_fsm_sack ∧ ~P_fsm_hold_ ∧ ~P_fsm_lock_)) => PA |
                 ((P_fsm_sack ∧ ~P_fsm_hold_ ∧ P_fsm_lock_) => PH | PD)) | P_ILL)))) in
     let new_P_fsm_astate = (new_P_fsm_stateA = PA) in
     let new_P_fsm_dstate = (new_P_fsm_stateA = PD) in
     let new_P_fsm_hlda_ = ~(new_P_fsm_stateA = PH) in
     let new_P_wr_data = L_ad_in in
     let new_P_addr = ((~P_rqt) => (SUBARRAY L_ad_in (25,0)) | P_addr) in
     let new_P_dest1 = ((~P_rqt) => (ELEMENT L_ad_in (31)) | P_dest1) in
     let new_P_be_ = ((~P_rqt) => L_be_ | P_be_) in
     let new_P_wr = ((~P_rqt) => L_wr | P_wr) in
     let new_P_be_n_ = L_be_ in
     let new_P_loadA = P_load in
     let new_P_downA = P_down in
     let new_P_sizeA = P_size in
     let new_P_fsm_state = P_fsm_state in
     let new_P_fsm_rst = P_fsm_rst in
     let new_P_fsm_mrqt = P_fsm_mrqt in
     let new_P_fsm_sack = P_fsm_sack in
     let new_P_fsm_cgnt_ = P_fsm_cgnt_ in
     let new_P_fsm_crqt_ = P_fsm_crqt_ in
     let new_P_fsm_hold_ = P_fsm_hold_ in
     let new_P_fsm_lock_ = P_fsm_lock_ in
     let new_P_rqt = P_rqt in
    let new_P_size = P_size in
    let new_P_load = P_load in
    let new_P_down = P_down in
    let new_P_lock_ = P_lock_ in
    let new_P_lock_inh_ = P_lock_inh_ in
    let new_P_male_ = P_male_ in
    let new_P_rale_ = P_rale_ in
    let p_ale = (~L_ads_ ∧ L_den_) in
    let p_sack = ((new_P_sizeA = ((new_P_downA) => WORDN 1 | WORDN 0)) ∧ ~I_srdy_ ∧ new_P_fsm_dstate) in

    let L_ad_out = ((~new_P_fsm_astate ∧ new_P_fsm_hlda_ ∧ ~(new_P_fsm_dstate ∧ new_P_wr)) => I_ad_in | ARBN) in
    let L_ready_ = (~(~I_srdy_ ∧ new_P_fsm_dstate)) in
    let od0 = ARBN in
    let od1 = (MALTER od0 (31,27) new_P_be_) in
    let od2 = (ALTER od1 (26) F) in
    let od3 = (MALTER od2 (25,24) (SUBARRAY new_P_addr (1,0))) in
    let od4 = (MALTER od3 (23,0) (SUBARRAY new_P_addr (25,2))) in
    let I_ad_addr_out = ((new_P_fsm_astate) => od4 | ARBN) in
    let I_ad_data_out = ((new_P_fsm_dstate ∧ new_P_wr) => new_P_wr_data | ARBN) in
    let I_be_ = ((new_P_fsm_hlda_) => ((new_P_fsm_astate) => new_P_be_ | new_P_be_n_) | ARBN) in
    let I_rale_ = ((new_P_fsm_hlda_) =>
    ~(~new_P_destl ∧ ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) ∧ new_P_fsm_astate ∧ new_P_rqt) | ARB) in
    let I_male_ = ((new_P_fsm_hlda_) =>
    ~(~new_P_destl ∧ (~((SUBARRAY new_P_addr (25,24)) = (WORDN 3))) ∧ new_P_fsm_astate ∧ new_P_rqt) | ARB) in
    let I_crqt_ = ~(new_P_destl ∧ new_P_rqt) in
    let I_cale_ = ~(~I_cgnt_ ∧ new_P_fsm_astate ∧ I_hold_) in
    let I_mrdy_ = ((new_P_fsm_hlda_) => F | ARB) in
    let I_last_ = ((new_P_fsm_hlda_) => (new_P_sizeA = ((new_P_downA) => WORDN 1 | WORDN 0)) | ARB) in
    let I_hlda_ = new_P_fsm_hlda_ in
    let I_lock_ = ~(~new_P_lock_ ∧ new_P_lock_inh_) in

    (L_ready_, I_last_, I_be_, I_mrdy_, I_ad_data_out, I_ad_addr_out, I_hlda_, I_lock_, I_cale_, I_male_, I_rale_,
     I_crqt_, L_ad_out)"
   );;

%------------------------------------------------------------------------------------------------------------------
   Next-state definition for Phase-B instruction.
------------------------------------------------------------------------------------------------------------------%

let PH_B_inst_def = new_definition
   ('PH_B_inst',
    "! (P_fsm_state P_fsm_stateA :pfsm_ty)
       (P_fsm_astate P_fsm_dstate P_fsm_hlda_ P_destl P_wr P_loadA P_downA :bool)
       (P_fsm_rst P_fsm_mrqt P_fsm_sack P_fsm_cgnt_ P_fsm_crqt_ P_fsm_hold_ P_fsm_lock_ P_rqt P_load :bool)
       (P_down P_lock_ P_lock_inh_ P_male_ P_rale_ :bool)
       (P_wr_data P_addr P_be_ P_be_n_ P_sizeA P_size :wordn)
       (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ I_cgnt_ I_hold_ I_srdy_ :bool) (L_ad_in L_be_ I_ad_in :wordn) .
    PH_B_inst (P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P_destl, P_be_,
                 P_wr, P_be_n_, P_sizeA, P_loadA, P_downA, P_fsm_state, P_fsm_rst, P_fsm_mrqt, P_fsm_sack,
                 P_fsm_cgnt_, P_fsm_crqt_, P_fsm_hold_, P_fsm_lock_, P_rqt, P_size, P_load, P_down, P_lock_,
                 P_lock_inh_, P_male_, P_rale_)
                (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_, I_srdy_) =

    let p_ale = (~L_ads_ ∧ L_den_) in
let p_sack = ((P_sizeA = ((P_downA) => WORDN 1 | WORDN 0)) ∧ ~I_srdy_ ∧ P_fsm_dstate) in
let new_P_rqt = ((p_ale ∧ ~(p_sack ∨ Rst)) => T |
                ((~p_ale ∧ (p_sack ∨ Rst)) => F |
                ((~p_ale ∧ ~(p_sack ∨ Rst)) => P_rqt | ARB))) in
let new_P_load = ~new_P_rqt in
let new_P_down = (~I_srdy_ ∧ P_fsm_dstate) in
let new_P_size = ((P_loadA) => (SUBARRAY L_ad_in (1,0)) |
                 ((P_downA) => DECN 1 P_sizeA | P_sizeA)) in
let new_P_male_ = ((P_fsm_astate) =>
      ~(~P_dest1 ∧ (~((SUBARRAY P_addr (25,24)) = (WORDN 3))) ∧ new_P_rqt) | P_male_) in
let new_P_rale_ = ((P_fsm_astate) =>
      ~(~P_dest1 ∧ ((SUBARRAY P_addr (25,24)) = (WORDN 3)) ∧ new_P_rqt) | P_rale_) in
let new_P_lock_ = ((Rst) => T |
                  ((P_fsm_dstate) => L_lock_ | P_lock_)) in
let new_P_lock_inh_ = ((Rst) => T |
                      ((~new_P_male_ ∨ ~new_P_rale_) => L_lock_ | P_lock_inh_)) in
let new_P_fsm_state = P_fsm_stateA in
let new_P_fsm_rst = Rst in
let new_P_fsm_mrqt = (~P_dest1 ∧ new_P_rqt) in
let new_P_fsm_sack = p_sack in
let new_P_fsm_cgnt_ = I_cgnt_ in
let new_P_fsm_crqt_ = ~(P_dest1 ∧ new_P_rqt) in
let new_P_fsm_hold_ = I_hold_ in
let new_P_fsm_lock_ = new_P_lock_ in
let new_P_fsm_stateA = P_fsm_stateA in
let new_P_fsm_astate = P_fsm_astate in
let new_P_fsm_dstate = P_fsm_dstate in
let new_P_fsm_hlda_ = P_fsm_hlda_ in
let new_P_wr_data = P_wr_data in
let new_P_addr = P_addr in
let new_P_dest1 = P_dest1 in
let new_P_be_ = P_be_ in
let new_P_wr = P_wr in
let new_P_be_n_ = P_be_n_ in
let new_P_sizeA = P_sizeA in
let new_P_loadA = P_loadA in
let new_P_downA = P_downA in

(new_P_fsm_stateA, new_P_fsm_astate, new_P_fsm_dstate, new_P_fsm_hlda_, new_P_wr_data, new_P_addr, new_P_dest1,
 new_P_be_, new_P_wr, new_P_be_n_, new_P_sizeA, new_P_loadA, new_P_downA, new_P_fsm_state, new_P_fsm_rst,
 new_P_fsm_mrqt, new_P_fsm_sack, new_P_fsm_cgnt_, new_P_fsm_crqt_, new_P_fsm_hold_, new_P_fsm_lock_,
 new_P_rqt, new_P_size, new_P_load, new_P_down, new_P_lock_, new_P_lock_inh_, new_P_male_, new_P_rale_)"
);;


%-----------------------------------------------------------------------------------------------------------
  Output definition for Phase-B instruction.
-----------------------------------------------------------------------------------------------------------%

let PH_B_out_def = new_definition
  ('PH_B_out',
   "! (P_fsm_state P_fsm_stateA :pfsm_ty)
      (P_fsm_astate P_fsm_dstate P_fsm_hlda_ P_dest1 P_wr P_loadA P_downA :bool)
      (P_fsm_rst P_fsm_mrqt P_fsm_sack P_fsm_cgnt_ P_fsm_crqt_ P_fsm_hold_ P_fsm_lock_ P_rqt P_load :bool)
```

```
    (P_down P_lock_ P_lock_inh_ P_male_ P_rale_ :bool)
    (P_wr_data P_addr P_be_ P_be_n_ P_sizeA P_size :wordn)
    (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ I_cgnt_ I_hold_ I_srdy_ :bool) (L_ad_in L_be_ I_ad_in :wordn) .
PH_B_out (P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P_dest1, P_be_,
            P_wr, P_be_n_, P_sizeA, P_loadA, P_downA, P_fsm_state, P_fsm_rst, P_fsm_mrqt, P_fsm_sack,
            P_fsm_cgnt_, P_fsm_crqt_, P_fsm_hold_, P_fsm_lock_, P_rqt, P_size, P_load, P_down, P_lock_,
            P_lock_inh_, P_male_, P_rale_)
          (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_, I_srdy_) =


let p_ale = (~L_ads_ ∧ L_den_) in
let p_sack = ((P_sizeA = ((P_downA) => WORDN 1 | WORDN 0)) ∧ ~I_srdy_ ∧ P_fsm_dstate) in
let new_P_rqt = ((p_ale ∧ ~(p_sack ∨ Rst)) => T |
                    ((~p_ale ∧ (p_sack ∨ Rst)) => F |
                    ((~p_ale ∧ ~(p_sack ∨ Rst)) => P_rqt | ARB))) in
let new_P_load = ~new_P_rqt in
let new_P_down = (~I_srdy_ ∧ P_fsm_dstate) in
let new_P_size = ((P_loadA) => (SUBARRAY L_ad_in (1,0)) |
                    ((P_downA) => DECN 1 P_sizeA | P_sizeA)) in
let new_P_male_ = ((P_fsm_astate) =>
        ~(~P_dest1 ∧ (~((SUBARRAY P_addr (25,24)) = (WORDN 3))) ∧ new_P_rqt) | P_male_) in
let new_P_rale_ = ((P_fsm_astate) =>
        ~(~P_dest1 ∧ ((SUBARRAY P_addr (25,24)) = (WORDN 3)) ∧ new_P_rqt) | P_rale_) in
let new_P_lock_ = ((Rst) => T |
                      ((P_fsm_dstate) => L_lock_ | P_lock_)) in
let new_P_lock_inh_ = ((Rst) => T |
                          ((~new_P_male_ ∨ ~new_P_rale_) => L_lock_ | P_lock_inh_)) in
let new_P_fsm_state = P_fsm_stateA in
let new_P_fsm_rst = Rst in
let new_P_fsm_mrqt = (~P_dest1 ∧ new_P_rqt) in
let new_P_fsm_sack = p_sack in
let new_P_fsm_cgnt_ = I_cgnt_ in
let new_P_fsm_crqt_ = ~(P_dest1 ∧ new_P_rqt) in
let new_P_fsm_hold_ = I_hold_ in
let new_P_fsm_lock_ = new_P_lock_ in
let new_P_fsm_stateA = P_fsm_stateA in
let new_P_fsm_astate = P_fsm_astate in
let new_P_fsm_dstate = P_fsm_dstate in
let new_P_fsm_hlda_ = P_fsm_hlda_ in
let new_P_wr_data = P_wr_data in
let new_P_addr = P_addr in
let new_P_dest1 = P_dest1 in
let new_P_be_ = P_be_ in
let new_P_wr = P_wr in
let new_P_be_n_ = P_be_n_ in
let new_P_sizeA = P_sizeA in
let new_P_loadA = P_loadA in
let new_P_downA = P_downA in

let L_ad_out = ((~new_P_fsm_astate ∧ new_P_fsm_hlda_ ∧ ~(new_P_fsm_dstate ∧ new_P_wr)) => I_ad_in | ARBN) in
let L_ready_ = (~(~I_srdy_ ∧ new_P_fsm_dstate)) in
let od0 = ARBN in
let od1 = MALTER od0 (31,27) new_P_be_ in
let od2 = ALTER od1 (26) F in
```

```
    let od3 = MALTER od2 (25,24) (SUBARRAY new_P_addr (1,0)) in
    let od4 = MALTER od3 (23,0) (SUBARRAY new_P_addr (25,2)) in
    let I_ad_addr_out = ((new_P_fsm_astate) => od4 | ARBN) in
    let I_ad_data_out = ((new_P_fsm_dstate ∧ new_P_wr) => new_P_wr_data | ARBN) in
    let I_be_ = ((new_P_fsm_hlda_) => ((new_P_fsm_astate) => new_P_be_ | new_P_be_n_) | ARBN) in
    let I_rale_ = ((new_P_fsm_hlda_) =>
    ~(~new_P_destl ∧ ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) ∧ new_P_fsm_astate ∧ new_P_rqt) | ARB) in
    let I_male_ = ((new_P_fsm_hlda_) =>
    ~(~new_P_destl ∧ (~((SUBARRAY new_P_addr (25,24)) = (WORDN 3))) ∧ new_P_fsm_astate ∧ new_P_rqt) | ARB) in
    let I_crqt_ = ~(new_P_destl ∧ new_P_rqt) in
    let I_cale_ = ~(~I_cgnt_ ∧ new_P_fsm_astate ∧ I_hold_) in
    let I_mrdy_ = ((new_P_fsm_hlda_) => F | ARB) in
    let I_last_ = ((new_P_fsm_hlda_) => (new_P_sizeA = ((new_P_downA) => WORDN 1 | WORDN 0)) | ARB) in
    let I_hlda_ = new_P_fsm_hlda_ in
    let I_lock_ = ~(~new_P_lock_ ∧ new_P_lock_inh_) in

    (L_ready_, I_last_, I_be_, I_mrdy_, I_ad_data_out, I_ad_addr_out, I_hlda_, I_lock_, I_cale_, I_male_, I_rale_,
     I_crqt_, L_ad_out)"
    );;

close_theory();;
```

C.2 M Port Specification

```
%------------------------------------------------------------------------------------------

    File:        m_phase.ml

    Author:      (c) D.A. Fura 1992

    Date:        31 March 1992

    This file contains the ml source for the phase-level specification of the M-Port of the FTEP PIU,
    an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
    The bulk of this code was translated from an M-language simulation program using a translator
    written by P.J. Windley at the University of Idaho.

------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm m_phase.th';;

new_theory 'm_phase';;

loadf 'abstract';;

map new_parent ['maux_def';'aux_def';'array_def';'wordn_def'];;

let m_state_ty = ":(mfsm_ty#bool#bool#bool#bool#bool#wordn#wordn#wordn#bool#wordn#
                   mfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#
                   bool#bool#wordn#wordn#wordn#bool#bool#bool#wordn#wordn)";;
let m_state = "((M_fsm_stateA, M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable,
                M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male_, M_fsm_rd,
                M_fsm_bw, M_fsm_ww, M_fsm_last_, M_fsm_mrdy_, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr,
                M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
               :^m_state_ty)";;

let m_env_ty = ":(bool#bool#bool#bool#bool#wordn#bool#bool#wordn#bool#wordn#bool#bool)";;
let m_env = "((ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
              I_mrdy_, MB_data_in, Edac_en_, Reset_parity)
             :^m_env_ty)";;

let m_out_ty = ":(wordn#bool#wordn#wordn#bool#bool#bool#bool#bool)";;
let m_out = "((I_ad_out, I_srdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_,
              MB_parity)
             :^m_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%------------------------------------------------------------------------------------------
    Next-state definition for Phase-A instruction.
------------------------------------------------------------------------------------------%

let PH_A_inst_def = new_definition
```

```
('PH_A_inst',
    "! (M_fsm_stateA M_fsm_state :mfsm_ty)
      (M_addrA M_beA M_countA M_rd_dataA M_addr M_be M_count M_rd_data M_detect :wordn)
      (M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_rdyA
       M_fsm_male_ M_fsm_rd M_fsm_bw M_fsm_ww M_fsm_last_ M_fsm_mrdy_ M_fsm_zero_cnt M_fsm_rst
       M_se M_wr M_rdy M_wwdel M_parity :bool)
      (I_ad_in I_be_ MB_data_in :wordn)
      (ClkA ClkB Rst Disable_eeprom Disable_writes I_male_ I_last_ I_mrdy_ Edac_en_ Reset_parity :bool) .

PH_A_inst (M_fsm_stateA, M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable,
           M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male_, M_fsm_rd,
           M_fsm_bw, M_fsm_ww, M_fsm_last_, M_fsm_mrdy_, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr,
           M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
          (ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
           I_mrdy_, MB_data_in, Edac_en_, Reset_parity) =

    let new_M_fsm_stateA =
      ((M_fsm_rst) => MI |
       ((M_fsm_state = MI) => ((~M_fsm_male_) => MA | MI) |
       ((M_fsm_state = MA) =>
            ((~M_fsm_mrdy_ ∧ M_fsm_ww) => MW |
            ((~M_fsm_mrdy_ ∧ (M_fsm_rd ∨ M_fsm_bw)) => MR | MA)) |
       ((M_fsm_state = MR) =>
            ((M_fsm_bw ∧ M_fsm_zero_cnt) => MBW |
            ((M_fsm_last_ ∧ M_fsm_rd ∧ M_fsm_zero_cnt) => MA |
            ((~M_fsm_last_ ∧ M_fsm_rd ∧ M_fsm_zero_cnt) => MRR | MR))) |
       ((M_fsm_state = MRR) => MI |
       ((M_fsm_state = MW) =>
            ((~M_fsm_last_ ∧ M_fsm_zero_cnt) => MI |
            ((M_fsm_last_ ∧ M_fsm_zero_cnt) => MA | MW)) |
       ((M_fsm_state = MBW) => MW | M_ILL))))))) in
    let new_M_fsm_address = (new_M_fsm_stateA = MA) in
    let new_M_fsm_read = (new_M_fsm_stateA = MR) in
    let new_M_fsm_write = (new_M_fsm_stateA = MW) in
    let new_M_fsm_byte_write = (new_M_fsm_stateA = MBW) in
    let new_M_fsm_mem_enable = (~(new_M_fsm_stateA = MI)) in
    let new_M_addrA = M_addr in
    let new_M_beA = M_be in
    let new_M_countA = M_count in
    let new_M_rdyA = M_rdy in
    let new_M_rd_dataA = M_rd_data in
    let new_M_fsm_state = M_fsm_state in
    let new_M_fsm_male_ = M_fsm_male_ in
    let new_M_fsm_rd = M_fsm_rd in
    let new_M_fsm_bw = M_fsm_bw in
    let new_M_fsm_ww = M_fsm_ww in
    let new_M_fsm_last_ = M_fsm_last_ in
    let new_M_fsm_mrdy_ = M_fsm_mrdy_ in
    let new_M_fsm_zero_cnt = M_fsm_zero_cnt in
    let new_M_fsm_rst = M_fsm_rst in
    let new_M_se = M_se in
    let new_M_wr = M_wr in
    let new_M_addr = M_addr in
    let new_M_be = M_be in
    let new_M_count = M_count in
    let new_M_rdy = M_rdy in
    let new_M_wwdel = M_wwdel in
    let new_M_parity = M_parity in
    let new_M_rd_data = M_rd_data in
    let new_M_detect = M_detect in

   (new_M_fsm_stateA, new_M_fsm_address, new_M_fsm_read, new_M_fsm_write, new_M_fsm_byte_write,
    new_M_fsm_mem_enable, new_M_addrA, new_M_beA, new_M_countA, new_M_rdyA, new_M_rd_dataA,
    new_M_fsm_state, new_M_fsm_male_, new_M_fsm_rd, new_M_fsm_bw, new_M_fsm_ww, new_M_fsm_last_,
    new_M_fsm_mrdy_, new_M_fsm_zero_cnt, new_M_fsm_rst, new_M_se, new_M_wr, new_M_addr, new_M_be,
    new_M_count, new_M_rdy, new_M_wwdel, new_M_parity, new_M_rd_data, new_M_detect)"
   );;

%----------------------------------------------------------------------------------------------------
   Output definition for Phase-A instruction.
----------------------------------------------------------------------------------------------------%

let PH_A_out_def = new_definition
 ('PH_A_out',
    "! (M_fsm_stateA M_fsm_state :mfsm_ty)
       (M_addrA M_beA M_countA M_rd_dataA M_addr M_be M_count M_rd_data M_detect :wordn)
       (M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_rdyA
        M_fsm_male_ M_fsm_rd M_fsm_bw M_fsm_ww M_fsm_last_ M_fsm_mrdy_ M_fsm_zero_cnt M_fsm_rst
        M_se M_wr M_rdy M_wwdel M_parity :bool)
       (I_ad_in I_be_ MB_data_in :wordn)
       (ClkA ClkB Rst Disable_eeprom Disable_writes I_male_ I_last_ I_mrdy_ Edac_en_ Reset_parity :bool)
       (rep:^rep_ty) .

   PH_A_out (M_fsm_stateA, M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable,
             M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male_, M_fsm_rd,
             M_fsm_bw, M_fsm_ww, M_fsm_last_, M_fsm_mrdy_, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr,
             M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
            (ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
             I_mrdy_, MB_data_in, Edac_en_, Reset_parity)
            rep =

   let new_M_fsm_stateA =
      ((M_fsm_rst) => MI |
       ((M_fsm_state = MI) => ((~M_fsm_male_) => MA | MI) |
       ((M_fsm_state = MA) =>
             ((~M_fsm_mrdy_ ∧ M_fsm_ww) => MW |
             ((~M_fsm_mrdy_ ∧ (M_fsm_rd ∨ M_fsm_bw)) => MR | MA)) |
       ((M_fsm_state = MR) =>
             ((M_fsm_bw ∧ M_fsm_zero_cnt) => MBW |
             ((M_fsm_last_ ∧ M_fsm_rd ∧ M_fsm_zero_cnt) => MA |
             ((~M_fsm_last_ ∧ M_fsm_rd ∧ M_fsm_zero_cnt) => MRR | MR))) |
       ((M_fsm_state = MRR) => MI |
       ((M_fsm_state = MW) =>
             ((~M_fsm_last_ ∧ M_fsm_zero_cnt) => MI |
             ((M_fsm_last_ ∧ M_fsm_zero_cnt) => MA | MW)) |
       ((M_fsm_state = MBW) => MW | M_ILL))))))) in
   let new_M_fsm_address = (new_M_fsm_stateA = MA) in
   let new_M_fsm_read = (new_M_fsm_stateA = MR) in
let new_M_fsm_write = (new_M_fsm_stateA = MW) in
let new_M_fsm_byte_write = (new_M_fsm_stateA = MBW) in
let new_M_fsm_mem_enable = (~(new_M_fsm_stateA = MI)) in
let new_M_addrA = M_addr in
let new_M_beA = M_be in
let new_M_countA = M_count in
let new_M_rdyA = M_rdy in
let new_M_rd_dataA = M_rd_data in
let new_M_fsm_state = M_fsm_state in
let new_M_fsm_male_ = M_fsm_male_ in
let new_M_fsm_rd = M_fsm_rd in
let new_M_fsm_bw = M_fsm_bw in
let new_M_fsm_ww = M_fsm_ww in
let new_M_fsm_last_ = M_fsm_last_ in
let new_M_fsm_mrdy_ = M_fsm_mrdy_ in
let new_M_fsm_zero_cnt = M_fsm_zero_cnt in
let new_M_fsm_rst = M_fsm_rst in
let new_M_se = M_se in
let new_M_wr = M_wr in
let new_M_addr = M_addr in
let new_M_be = M_be in
let new_M_count = M_count in
let new_M_rdy = M_rdy in
let new_M_wwdel = M_wwdel in
let new_M_parity = M_parity in
let new_M_rd_data = M_rd_data in
let new_M_detect = M_detect in
let m_rdy = ((new_M_fsm_write ∧ (new_M_countA = (WORDN 1)))
            ∨ (new_M_fsm_read ∧ (new_M_countA = (WORDN 1)) ∧ ~new_M_wr)) in
let m_srdy_ = ~((new_M_rdyA ∧ ~new_M_wr) ∨ (m_rdy ∧ new_M_wr)) in
let mb_data_7_0 = ((ELEMENT new_M_beA (0)) => (SUBARRAY I_ad_in (7,0)) | (SUBARRAY new_M_rd_dataA (7,0))) in
let mb_data_15_8 = ((ELEMENT new_M_beA (1)) => (SUBARRAY I_ad_in (15,8)) | (SUBARRAY new_M_rd_dataA
(15,8))) in
let mb_data_23_16 = ((ELEMENT new_M_beA (2)) => (SUBARRAY I_ad_in (23,16)) | (SUBARRAY new_M_rd_dataA
(23,16))) in
let mb_data_31_24 = ((ELEMENT new_M_beA (3)) => (SUBARRAY I_ad_in (31,24)) | (SUBARRAY new_M_rd_dataA
(31,24))) in
let mb_data = ((MALTER (MALTER (MALTER (MALTER ARBN (7,0) mb_data_7_0)
                                                    (15,8) mb_data_15_8)
                                                    (23,16) mb_data_23_16)
                                                    (31,24) mb_data_31_24)) in
let I_ad_out = ((~new_M_wr ∧ new_M_fsm_mem_enable) => new_M_rd_dataA | ARBN) in
let I_srdy_ = ((new_M_fsm_mem_enable) => m_srdy_ | ARB) in
let MB_addr = ((new_M_rdyA) => (INCN 18 new_M_addrA) | new_M_addrA) in
let MB_data_out = ((new_M_fsm_write) => (Ham_Enc rep mb_data) | ARBN) in
let MB_cs_eeprom_ = ~(new_M_fsm_mem_enable ∧ ~new_M_se) in
let MB_cs_sram_ = ~(new_M_fsm_mem_enable ∧ new_M_se) in
let MB_we_ = ~((new_M_se ∨ ~new_M_fsm_mem_enable ∨ ~Disable_eeprom)
              ∧ ~Disable_writes
              ∧ (new_M_fsm_byte_write ∨ new_M_fsm_write ∨ new_M_wwdel)) in
let MB_oe_ = ~((~new_M_wr ∧ new_M_fsm_address) ∨ new_M_fsm_read) in
let MB_parity = new_M_parity in

(I_ad_out, I_srdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_, MB_parity)"
```

```
);;

%-----------------------------------------------------------------------------------------------------------
   Next-state definition for Phase-B instruction.
-----------------------------------------------------------------------------------------------------------%

let PH_B_inst_def = new_definition
('PH_B_inst',
   "! (M_fsm_stateA M_fsm_state :mfsm_ty)
      (M_addrA M_beA M_countA M_rd_dataA M_addr M_be M_count M_rd_data M_detect :wordn)
      (M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_rdyA
         M_fsm_male_ M_fsm_rd M_fsm_bw M_fsm_ww M_fsm_last_ M_fsm_mrdy_ M_fsm_zero_cnt M_fsm_rst
         M_se M_wr M_rdy M_wwdel M_parity :bool)
      (I_ad_in I_be_ MB_data_in :wordn)
      (ClkA ClkB Rst Disable_eeprom Disable_writes I_male_ I_last_ I_mrdy_ Edac_en_ Reset_parity :bool)
      (rep:^rep_ty) .

PH_B_inst (M_fsm_stateA, M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable,
M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male_, M_fsm_rd,
M_fsm_bw, M_fsm_ww, M_fsm_last_, M_fsm_mrdy_, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr,
M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
                  (ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
I_mrdy_, MB_data_in, Edac_en_, Reset_parity)
                  rep =

   let new_M_se = ((~I_male_) => (ELEMENT I_ad_in (23)) | M_se) in
   let new_M_wr = ((~I_male_) => (ELEMENT I_ad_in (27)) | M_wr) in
   let new_M_addr =
         ((~I_male_) => (SUBARRAY I_ad_in (18,0)) |
         ((M_rdyA) => (INCN 18 M_addrA) | M_addrA)) in
   let new_M_count =
         ((M_fsm_address ∨ M_fsm_byte_write) => ((new_M_se) => (WORDN 1) | (WORDN 2)) |
         ((M_fsm_write ∨ M_fsm_read) => (DECN 1 M_countA) | M_countA)) in
   let m_rdy = ((M_fsm_write ∧ (new_M_count = (WORDN 0)))
                  ∨ (M_fsm_read ∧ (new_M_count = (WORDN 0)) ∧ ~new_M_wr)) in
   let m_srdy_ = ~((M_rdyA ∧ ~new_M_wr) ∨ (m_rdy ∧ new_M_wr)) in
   let new_M_be = ((~I_male_ ∨ ~m_srdy_) => (NOTN 3 I_be_) | M_be) in
   let new_M_rdy = m_rdy in
   let new_M_wwdel = (M_fsm_address ∧ new_M_wr ∧ (new_M_be = (WORDN 15))) in
   let new_M_rd_data = ((M_fsm_read) => (Ham_Dec rep MB_data_in) | M_rd_data) in
   let new_M_detect =
         (((M_fsm_read ∧ ~new_M_wr) ∨ new_M_wr ∨ ~M_fsm_mem_enable) =>
                  ((~Edac_en_) => (Ham_Det1 rep MB_data_in) | (WORDN 0)) | M_detect) in
   let m_error = (~m_srdy_ ∧ M_fsm_mem_enable ∧ (Ham_Det2 rep (new_M_detect, ~Edac_en_))) in
   let new_M_parity =
         ((m_error ∧ ~(Rst ∨ Reset_parity)) => T |
         ((~m_error ∧ (Rst ∨ Reset_parity)) => F |
         ((~m_error ∧ ~(Rst ∨ Reset_parity)) => M_parity | ARB))) in
   let new_M_fsm_state = M_fsm_stateA in
   let new_M_fsm_male_ = I_male_ in
   let new_M_fsm_rd = (~new_M_wr ∧ M_fsm_mem_enable) in
   let new_M_fsm_bw = ((~(new_M_be = (WORDN 15))) ∧ new_M_wr ∧ M_fsm_mem_enable) in
   let new_M_fsm_ww = ((new_M_be = (WORDN 15)) ∧ new_M_wr ∧ M_fsm_mem_enable) in
   let new_M_fsm_last_ = I_last_ in
```

```
    let new_M_fsm_mrdy_ = I_mrdy_ in
    let new_M_fsm_zero_cnt = (new_M_count = (WORDN 0)) in
    let new_M_fsm_rst = Rst in
    let new_M_fsm_stateA = M_fsm_stateA in
    let new_M_fsm_address = M_fsm_address in
    let new_M_fsm_read = M_fsm_read in
    let new_M_fsm_write = M_fsm_write in
    let new_M_fsm_byte_write = M_fsm_byte_write in
    let new_M_fsm_mem_enable = M_fsm_mem_enable in
    let new_M_addrA = M_addrA in
    let new_M_beA = M_beA in
    let new_M_countA = M_countA in
    let new_M_rdyA = M_rdyA in
    let new_M_rd_dataA = M_rd_dataA in

  (new_M_fsm_stateA, new_M_fsm_address, new_M_fsm_read, new_M_fsm_write, new_M_fsm_byte_write,
        new_M_fsm_mem_enable, new_M_addrA, new_M_beA, new_M_countA, new_M_rdyA, new_M_rd_dataA,
        new_M_fsm_state, new_M_fsm_male_, new_M_fsm_rd, new_M_fsm_bw, new_M_fsm_ww, new_M_fsm_last_,
        new_M_fsm_mrdy_, new_M_fsm_zero_cnt, new_M_fsm_rst, new_M_se, new_M_wr, new_M_addr, new_M_be,
        new_M_count, new_M_rdy, new_M_wwdel, new_M_parity, new_M_rd_data, new_M_detect)"

  );;

  %------------------------------------------------------------------------------------------------
    Output definition for Phase-B instruction.
  ------------------------------------------------------------------------------------------------%

  let PH_B_out_def = new_definition
   ('PH_B_out',
     "! (M_fsm_stateA M_fsm_state :mfsm_ty)
        (M_addrA M_beA M_countA M_rd_dataA M_addr M_be M_count M_rd_data M_detect :wordn)
        (M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_rdyA
           M_fsm_male_ M_fsm_rd M_fsm_bw M_fsm_ww M_fsm_last_ M_fsm_mrdy_ M_fsm_zero_cnt M_fsm_rst
           M_se M_wr M_rdy M_wwdel M_parity :bool)
        (I_ad_in I_be_ MB_data_in :wordn)
        (ClkA ClkB Rst Disable_eeprom Disable_writes I_male_ I_last_ I_mrdy_ Edac_en_ Reset_parity :bool)
        (rep:^rep_ty) .
    PH_B_out (M_fsm_stateA, M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable,
              M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male_, M_fsm_rd,
              M_fsm_bw, M_fsm_ww, M_fsm_last_, M_fsm_mrdy_, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr,
              M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
             (ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
              I_mrdy_, MB_data_in, Edac_en_, Reset_parity)
             rep =

     let new_M_se = ((~I_male_) => (ELEMENT I_ad_in (23)) | M_se) in
     let new_M_wr = ((~I_male_) => (ELEMENT I_ad_in (27)) | M_wr) in
     let new_M_addr =
           ((~I_male_) => (SUBARRAY I_ad_in (18,0)) |
           ((M_rdyA) => (INCN 18 M_addrA) | M_addrA)) in
     let new_M_count =
           ((M_fsm_address \/ M_fsm_byte_write) => ((new_M_se) => (WORDN 1) | (WORDN 2)) |
           ((M_fsm_write \/ M_fsm_read) => (DECN 1 M_countA) | M_countA)) in
     let m_rdy = ((M_fsm_write /\ (new_M_count = (WORDN 0)))
            ∨ (M_fsm_read ∧ (new_M_count = (WORDN 0)) ∧ ~new_M_wr)) in
let m_srdy_ = ~((M_rdyA ∧ ~new_M_wr) ∨ (m_rdy ∧ new_M_wr)) in
let new_M_be = ((~I_male_ ∨ ~m_srdy_) => (NOTN 3 I_be_) | M_be) in
let new_M_rdy = m_rdy in
let new_M_wwdel = (M_fsm_address ∧ new_M_wr ∧ (new_M_be = (WORDN 15))) in
let new_M_rd_data = ((M_fsm_read) => (Ham_Dec rep MB_data_in) | M_rd_data) in
let new_M_detect =
      (((M_fsm_read ∧ ~new_M_wr) ∨ new_M_wr ∨ ~M_fsm_mem_enable) =>
            ((~Edac_en_) => (Ham_Det1 rep MB_data_in) | (WORDN 0)) | M_detect) in
let m_error = (~m_srdy_ ∧ M_fsm_mem_enable ∧ (Ham_Det2 rep (new_M_detect, ~Edac_en_))) in
let new_M_parity =
      ((m_error ∧ ~(Rst ∨ Reset_parity)) => T |
      ((~m_error ∧ (Rst ∨ Reset_parity)) => F |
      ((~m_error ∧ ~(Rst ∨ Reset_parity)) => M_parity | ARB))) in
let new_M_fsm_state = M_fsm_stateA in
let new_M_fsm_male_ = I_male_ in
let new_M_fsm_rd = (~new_M_wr ∧ M_fsm_mem_enable) in
let new_M_fsm_bw = ((~(new_M_be = (WORDN 15))) ∧ new_M_wr ∧ M_fsm_mem_enable) in
let new_M_fsm_ww = ((new_M_be = (WORDN 15)) ∧ new_M_wr ∧ M_fsm_mem_enable) in
let new_M_fsm_last_ = I_last_ in
let new_M_fsm_mrdy_ = I_mrdy_ in
let new_M_fsm_zero_cnt = (new_M_count = (WORDN 0)) in
let new_M_fsm_rst = Rst in
let new_M_fsm_stateA = M_fsm_stateA in
let new_M_fsm_address = M_fsm_address in
let new_M_fsm_read = M_fsm_read in
let new_M_fsm_write = M_fsm_write in
let new_M_fsm_byte_write = M_fsm_byte_write in
let new_M_fsm_mem_enable = M_fsm_mem_enable in
let new_M_addrA = M_addrA in
let new_M_beA = M_beA in
let new_M_countA = M_countA in
let new_M_rdyA = M_rdyA in
let new_M_rd_dataA = M_rd_dataA in
let m_rdy = ((new_M_fsm_write ∧ (new_M_countA = (WORDN 1)))
            ∨ (new_M_fsm_read ∧ (new_M_countA = (WORDN 1)) ∧ ~new_M_wr)) in
let m_srdy_ = ~((new_M_rdyA ∧ ~new_M_wr) ∨ (m_rdy ∧ new_M_wr)) in
let mb_data_7_0 = ((ELEMENT new_M_beA (0)) => (SUBARRAY I_ad_in (7,0)) | (SUBARRAY new_M_rd_dataA (7,0))) in
let mb_data_15_8 =
   ((ELEMENT new_M_beA (1)) => (SUBARRAY I_ad_in (15,8)) | (SUBARRAY new_M_rd_dataA (15,8))) in
let mb_data_23_16 =
   ((ELEMENT new_M_beA (2)) => (SUBARRAY I_ad_in (23,16)) | (SUBARRAY new_M_rd_dataA (23,16))) in
let mb_data_31_24 =
   ((ELEMENT new_M_beA (3)) => (SUBARRAY I_ad_in (31,24)) | (SUBARRAY new_M_rd_dataA (31,24))) in
let mb_data = ((MALTER (MALTER (MALTER (MALTER ARBN (7,0) mb_data_7_0)
                                                   (15,8) mb_data_15_8)
                                                   (23,16) mb_data_23_16)
                                                   (31,24) mb_data_31_24)) in

let I_ad_out = ((~new_M_wr ∧ new_M_fsm_mem_enable) => new_M_rd_dataA | ARBN) in
let I_srdy_ = ((new_M_fsm_mem_enable) => m_srdy_ | ARB) in
let MB_addr = ((new_M_rdyA) => (INCN 18 new_M_addrA) | new_M_addrA) in
let MB_data_out = ((new_M_fsm_write) => (Ham_Enc rep mb_data) | ARBN) in
let MB_cs_eeprom_ = ~(new_M_fsm_mem_enable ∧ ~new_M_se) in
let MB_cs_sram_ = ~(new_M_fsm_mem_enable ∧ new_M_se) in
let MB_we_ = ~((new_M_se ∨ ~new_M_fsm_mem_enable ∨ ~Disable_eeprom)
                ∧ ~Disable_writes
                ∧ (new_M_fsm_byte_write ∨ new_M_fsm_write ∨ new_M_wwdel)) in
let MB_oe_ = ~((~new_M_wr ∧ new_M_fsm_address) ∨ new_M_fsm_read) in
let MB_parity = new_M_parity in

(I_ad_out, I_srdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_, MB_parity)"
);;
```

C.3 R Port Specification

```
%-----------------------------------------------------------------------------------------------------------

    File:       r_phase.ml

    Author:     (c) D.A. Fura 1992

    Date:       31 March 1992

    This file contains the ml source for the phase-level specification of the R-Port of the FTEP PIU,
    an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
    The bulk of this code was translated from an M-language simulation program using a translator
    written by P.J. Windley at the University of Idaho.

-----------------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm r_phase.th';;

new_theory 'r_phase';;

loadf 'abstract';;

map new_parent ['raux_def';'aux_def';'array_def';'wordn_def'];;

let r_state_ty = ":(rfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
                    bool#bool#wordn#wordn#bool#bool#wordn#wordn#bool#bool#wordn#wordn#bool#bool#
                    wordn#bool#wordn#wordn#wordn#
                    rfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
                    bool#bool#bool#wordn#wordn#bool#wordn#bool#bool#bool#wordn#wordn#bool#wordn#
                    bool#bool#bool#wordn#wordn#bool#wordn#bool#bool#bool#wordn#wordn#bool#bool#
                    wordn#wordn#wordn#bool#wordn#bool#wordn#bool#wordn#bool)";;
let r_state = "((R_fsm_stateA, R_fsm_cntlatch, R_fsm_srdy_, R_int0_en, R_int0_disA, R_int3_en, R_int3_disA,
                R_c01_cout, R_c01_cout_delA, R_c23_cout, R_c23_cout_delA, R_cntlatch_delA, R_srdy_delA_,
                R_reg_selA, R_ctr0, R_ctr0_ce, R_ctr0_cin, R_ctr0_outA, R_ctr1, R_ctr1_ce, R_ctr1_cin,
                R_ctr1_outA, R_ctr2, R_ctr2_ce, R_ctr2_cin, R_ctr2_outA, R_ctr3, R_ctr3_ce, R_ctr3_cin,
                R_ctr3_outA, R_icr_loadA, R_icr_oldA, R_icrA, R_busA_latch, R_fsm_state, R_fsm_ale_,
                R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_int0_dis, R_int3_dis, R_c01_cout_del, R_int1_en,
                R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_, R_reg_sel, R_ctr0_in,
                R_ctr0_mux_sel, R_ctr0_irden, R_ctr0_cry, R_ctr0_new, R_ctr0_out, R_ctr0_orden, R_ctr1_in,
                R_ctr1_mux_sel, R_ctr1_irden, R_ctr1_cry, R_ctr1_new, R_ctr1_out, R_ctr1_orden, R_ctr2_in,
                R_ctr2_mux_sel, R_ctr2_irden, R_ctr2_cry, R_ctr2_new, R_ctr2_out, R_ctr2_orden, R_ctr3_in,
                R_ctr3_mux_sel, R_ctr3_irden, R_ctr3_cry, R_ctr3_new, R_ctr3_out, R_ctr3_orden, R_icr_load,
                R_icr_old, R_icr_mask, R_icr, R_icr_rden, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr,
                R_sr_rden)
                :^r_state_ty)";;

let r_env_ty = ":(bool#bool#bool#wordn#bool#bool#wordn#bool#bool#bool#wordn#wordn#bool#bool#
                  wordn#wordn#wordn#bool#bool#wordn)";;
let r_env = "((ClkA, ClkB, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
              Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss)
```

```
            :^r_env_ty)";;

let r_out_ty = ":(wordn#bool#bool#bool#bool#bool#wordn#wordn#bool#bool)";;
let r_out = "((I_ad_out, I_srdy_, Int0_, Int1, Int2, Int3_, Ccr, Led, Reset_error, Pmm_invalid)
            :^r_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%------------------------------------------------------------------------------------------------
   Next-state definition for Phase-A instruction.
------------------------------------------------------------------------------------------------%

let PH_A_inst_def = new_definition
 ('PH_A_inst',
   "! (rep:^rep_ty)
     (R_fsm_stateA R_fsm_state :rfsm_ty)
     (R_reg_selA R_ctr0 R_ctr0_outA R_ctr1 R_ctr1_outA R_ctr2 R_ctr2_outA R_ctr3 R_ctr3_outA R_icr_oldA
       R_icrA R_busA_latch R_reg_sel R_ctr0_in R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1_new R_ctr1_out
       R_ctr2_in R_ctr2_new R_ctr2_out R_ctr3_in R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr
       R_ccr R_gcr R_sr :wordn)
     (R_fsm_cntlatch R_fsm_srdy_ R_int0_en R_int0_disA R_int3_en R_int3_disA R_c01_cout R_c01_cout_delA
       R_c23_cout R_c23_cout_delA R_cntlatch_delA R_srdy_delA_ R_ctr0_ce R_ctr0_cin R_ctr1_ce R_ctr1_cin
       R_ctr2_ce R_ctr2_cin R_ctr3_ce R_ctr3_cin R_icr_loadA R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst
       R_int0_dis R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del
       R_srdy_del_ R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden R_ctr1_mux_sel R_ctr1_irden
       R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden R_ctr3_mux_sel
       R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden
       R_sr_rden :bool)
     (I_ad_in I_be_ Cpu_fail Reset_cpu S_state Id ChannelID C_ss :wordn)
     (ClkA ClkB Rst I_rale_ I_last_ I_mrdy_ Disable_int Disable_writes Piu_fail Pmm_fail
       CB_parity MB_parity :bool) .
   PH_A_inst rep
            (R_fsm_stateA, R_fsm_cntlatch, R_fsm_srdy_, R_int0_en, R_int0_disA, R_int3_en, R_int3_disA,
             R_c01_cout, R_c01_cout_delA, R_c23_cout, R_c23_cout_delA, R_cntlatch_delA, R_srdy_delA_,
             R_reg_selA, R_ctr0, R_ctr0_ce, R_ctr0_cin, R_ctr0_outA, R_ctr1, R_ctr1_ce, R_ctr1_cin,
             R_ctr1_outA, R_ctr2, R_ctr2_ce, R_ctr2_cin, R_ctr2_outA, R_ctr3, R_ctr3_ce, R_ctr3_cin,
             R_ctr3_outA, R_icr_loadA, R_icr_oldA, R_icrA, R_busA_latch, R_fsm_state, R_fsm_ale_,
             R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_int0_dis, R_int3_dis, R_c01_cout_del, R_int1_en,
             R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_, R_reg_sel, R_ctr0_in,
             R_ctr0_mux_sel, R_ctr0_irden, R_ctr0_cry, R_ctr0_new, R_ctr0_out, R_ctr0_orden, R_ctr1_in,
             R_ctr1_mux_sel, R_ctr1_irden, R_ctr1_cry, R_ctr1_new, R_ctr1_out, R_ctr1_orden, R_ctr2_in,
             R_ctr2_mux_sel, R_ctr2_irden, R_ctr2_cry, R_ctr2_new, R_ctr2_out, R_ctr2_orden, R_ctr3_in,
             R_ctr3_mux_sel, R_ctr3_irden, R_ctr3_cry, R_ctr3_new, R_ctr3_out, R_ctr3_orden, R_icr_load,
             R_icr_old, R_icr_mask, R_icr, R_icr_rden, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr,
             R_sr_rden)
            (ClkA, ClkB, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
             Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss) =

   let new_R_fsm_stateA =
        ((R_fsm_rst) => RI |
        ((R_fsm_state = RI) => ((~R_fsm_ale_) => RA | RI) |
        ((R_fsm_state = RA) => ((~R_fsm_mrdy_) => RD | RA) |
        ((~R_fsm_last_) => RI | RA)))) in
   let new_R_fsm_cntlatch = ((R_fsm_state = RI) /\ ~R_fsm_ale_) in
let new_R_fsm_srdy_ = ~((R_fsm_state = RA) ∧ ~R_fsm_mrdy_) in
let new_R_cntlatch_delA = R_cntlatch_del in
let new_R_srdy_delA_ = R_srdy_del_ in
let new_R_reg_selA = R_reg_sel in
let r_reg_sel = ((~new_R_srdy_delA_) => (INCN 3 new_R_reg_selA) | new_R_reg_selA) in
let r_write = (~Disable_writes ∧ R_wr ∧ (new_R_fsm_stateA = RD)) in
let r_read = (~R_wr ∧ (new_R_fsm_stateA = RA)) in
let r_cir_wr01 = (r_write ∧ ((r_reg_sel = (WORDN 8)) ∨ (r_reg_sel = (WORDN 9)))) in
let r_cir_wr23 = (r_write ∧ ((r_reg_sel = (WORDN 10)) ∨ (r_reg_sel = (WORDN 11)))) in
let new_R_ctr0 = ((R_ctr0_mux_sel) => R_ctr0_in | R_ctr0_new) in
let new_R_ctr0_ce = (ELEMENT R_gcr (19)) in
let new_R_ctr0_cin = T in
let new_R_ctr0_outA = R_ctr0_new in
let new_R_ctr1 = ((R_ctr1_mux_sel) => R_ctr1_in | R_ctr1_new) in
let new_R_ctr1_ce = T in
let new_R_ctr1_cin = R_ctr0_cry in
let new_R_ctr1_outA = R_ctr1_new in
let new_R_ctr2 = ((R_ctr2_mux_sel) => R_ctr2_in | R_ctr2_new) in
let new_R_ctr2_ce = (ELEMENT R_gcr (23)) in
let new_R_ctr2_cin = T in
let new_R_ctr2_outA = R_ctr2_new in
let new_R_ctr3 = ((R_ctr3_mux_sel) => R_ctr3_in | R_ctr3_new) in
let new_R_ctr3_ce = T in
let new_R_ctr3_cin = R_ctr2_cry in
let new_R_ctr3_outA = R_ctr3_new in
let new_R_icr_loadA = R_icr_load in
let new_R_icr_oldA =
      (((new_R_fsm_stateA = RA) ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => R_icr | R_icr_oldA) in
let new_R_icrA =
      ((~(r_reg_sel = (WORDN 1))) => Andn rep (R_icr_old, R_icr_mask) | Orn rep (R_icr_old, R_icr_mask)) in
let new_R_int0_en = (((ELEMENT R_icr (0)) ∧ (ELEMENT R_icr (8))) ∨
                     ((ELEMENT R_icr (1)) ∧ (ELEMENT R_icr (9))) ∨
                     ((ELEMENT R_icr (2)) ∧ (ELEMENT R_icr (10))) ∨
                     ((ELEMENT R_icr (3)) ∧ (ELEMENT R_icr (11))) ∨
                     ((ELEMENT R_icr (4)) ∧ (ELEMENT R_icr (12))) ∨
                     ((ELEMENT R_icr (5)) ∧ (ELEMENT R_icr (13))) ∨
                     ((ELEMENT R_icr (6)) ∧ (ELEMENT R_icr (14))) ∨
                     ((ELEMENT R_icr (7)) ∧ (ELEMENT R_icr (15)))) in
let new_R_int0_disA = R_int0_dis in
let new_R_int3_en = (((ELEMENT R_icr (16)) ∧ (ELEMENT R_icr (24))) ∨
                     ((ELEMENT R_icr (17)) ∧ (ELEMENT R_icr (25))) ∨
                     ((ELEMENT R_icr (18)) ∧ (ELEMENT R_icr (26))) ∨
                     ((ELEMENT R_icr (19)) ∧ (ELEMENT R_icr (27))) ∨
                     ((ELEMENT R_icr (20)) ∧ (ELEMENT R_icr (28))) ∨
                     ((ELEMENT R_icr (21)) ∧ (ELEMENT R_icr (29))) ∨
                     ((ELEMENT R_icr (22)) ∧ (ELEMENT R_icr (30))) ∨
                     ((ELEMENT R_icr (23)) ∧ (ELEMENT R_icr (31)))) in
let new_R_int3_disA = R_int3_dis in
let new_R_c01_cout = R_ctr1_cry in
let new_R_c01_cout_delA = R_c01_cout_del in
let new_R_c23_cout = R_ctr3_cry in
let new_R_c23_cout_delA = R_c23_cout_del in
let new_R_busA_latch =
      (((R_ctr0_irden) => R_ctr0_in |
        ((R_ctr0_orden) => R_ctr0_out |
        ((R_ctr1_irden) => R_ctr1_in |
        ((R_ctr1_orden) => R_ctr1_out |
        ((R_ctr2_irden) => R_ctr2_in |
        ((R_ctr2_orden) => R_ctr2_out |
        ((R_ctr3_irden) => R_ctr3_in |
        ((R_ctr3_orden) => R_ctr3_out |
        ((R_icr_rden) => R_icr |
        ((R_ccr_rden) => R_ccr |
        ((R_gcr_rden) => R_gcr |
        ((R_sr_rden) => R_sr | ARBN)))))))))))) in
let new_R_fsm_state = R_fsm_state in
let new_R_fsm_ale_ = R_fsm_ale_ in
let new_R_fsm_mrdy_ = R_fsm_mrdy_ in
let new_R_fsm_last_ = R_fsm_last_ in
let new_R_fsm_rst = R_fsm_rst in
let new_R_int0_dis = R_int0_dis in
let new_R_int3_dis = R_int3_dis in
let new_R_c01_cout_del = R_c01_cout_del in
let new_R_int1_en = R_int1_en in
let new_R_c23_cout_del = R_c23_cout_del in
let new_R_int2_en = R_int2_en in
let new_R_wr = R_wr in
let new_R_cntlatch_del = R_cntlatch_del in
let new_R_srdy_del_ = R_srdy_del_ in
let new_R_reg_sel = R_reg_sel in
let new_R_ctr0_in = R_ctr0_in in
let new_R_ctr0_mux_sel = R_ctr0_mux_sel in
let new_R_ctr0_irden = R_ctr0_irden in
let new_R_ctr0_cry = R_ctr0_cry in
let new_R_ctr0_new = R_ctr0_new in
let new_R_ctr0_out = R_ctr0_out in
let new_R_ctr0_orden = R_ctr0_orden in
let new_R_ctr1_in = R_ctr1_in in
let new_R_ctr1_mux_sel = R_ctr1_mux_sel in
let new_R_ctr1_irden = R_ctr1_irden in
let new_R_ctr1_cry = R_ctr1_cry in
let new_R_ctr1_new = R_ctr1_new in
let new_R_ctr1_out = R_ctr1_out in
let new_R_ctr1_orden = R_ctr1_orden in
let new_R_ctr2_in = R_ctr2_in in
let new_R_ctr2_mux_sel = R_ctr2_mux_sel in
let new_R_ctr2_irden = R_ctr2_irden in
let new_R_ctr2_cry = R_ctr2_cry in
let new_R_ctr2_new = R_ctr2_new in
let new_R_ctr2_out = R_ctr2_out in
let new_R_ctr2_orden = R_ctr2_orden in
let new_R_ctr3_in = R_ctr3_in in
let new_R_ctr3_mux_sel = R_ctr3_mux_sel in
let new_R_ctr3_irden = R_ctr3_irden in
let new_R_ctr3_cry = R_ctr3_cry in
let new_R_ctr3_new = R_ctr3_new in
let new_R_ctr3_out = R_ctr3_out in
let new_R_ctr3_orden = R_ctr3_orden in
```

```
    let new_R_icr_load = R_icr_load in
    let new_R_icr_old = R_icr_old in
    let new_R_icr_mask = R_icr_mask in
    let new_R_icr = R_icr in
    let new_R_icr_rden = R_icr_rden in
    let new_R_ccr = R_ccr in
    let new_R_ccr_rden = R_ccr_rden in
    let new_R_gcr = R_gcr in
    let new_R_gcr_rden = R_gcr_rden in
    let new_R_sr = R_sr in
    let new_R_sr_rden = R_sr_rden in

    (new_R_fsm_stateA, new_R_fsm_cntlatch, new_R_fsm_srdy_, new_R_int0_en, new_R_int0_disA, new_R_int3_en,
     new_R_int3_disA, new_R_c01_cout, new_R_c01_cout_delA, new_R_c23_cout, new_R_c23_cout_delA,
     new_R_cntlatch_delA,
     new_R_srdy_delA_, new_R_reg_selA, new_R_ctr0, new_R_ctr0_ce, new_R_ctr0_cin, new_R_ctr0_outA, new_R_ctr1,
     new_R_ctr1_ce, new_R_ctr1_cin, new_R_ctr1_outA, new_R_ctr2, new_R_ctr2_ce, new_R_ctr2_cin, new_R_ctr2_outA,
     new_R_ctr3, new_R_ctr3_ce, new_R_ctr3_cin, new_R_ctr3_outA, new_R_icr_loadA, new_R_icr_oldA, new_R_icrA,
     new_R_busA_latch, new_R_fsm_state, new_R_fsm_ale_, new_R_fsm_mrdy_, new_R_fsm_last_, new_R_fsm_rst,
     new_R_int0_dis, new_R_int3_dis, new_R_c01_cout_del, new_R_int1_en, new_R_c23_cout_del, new_R_int2_en,
     new_R_wr,
     new_R_cntlatch_del, new_R_srdy_del_, new_R_reg_sel, new_R_ctr0_in, new_R_ctr0_mux_sel, new_R_ctr0_irden,
     new_R_ctr0_cry, new_R_ctr0_new, new_R_ctr0_out, new_R_ctr0_orden, new_R_ctr1_in, new_R_ctr1_mux_sel,
     new_R_ctr1_irden, new_R_ctr1_cry, new_R_ctr1_new, new_R_ctr1_out, new_R_ctr1_orden, new_R_ctr2_in,
     new_R_ctr2_mux_sel, new_R_ctr2_irden, new_R_ctr2_cry, new_R_ctr2_new, new_R_ctr2_out, new_R_ctr2_orden,
     new_R_ctr3_in, new_R_ctr3_mux_sel, new_R_ctr3_irden, new_R_ctr3_cry, new_R_ctr3_new, new_R_ctr3_out,
     new_R_ctr3_orden, new_R_icr_load, new_R_icr_old, new_R_icr_mask, new_R_icr, new_R_icr_rden, new_R_ccr,
     new_R_ccr_rden, new_R_gcr, new_R_gcr_rden, new_R_sr, new_R_sr_rden)"
    );;

%-----------------------------------------------------------------------------------------------------------------
    Output definition for Phase-A instruction.
-----------------------------------------------------------------------------------------------------------------%

let PH_A_out_def = new_definition
 ('PH_A_out',
    "! (rep:^rep_ty)
       (R_fsm_stateA R_fsm_state :rfsm_ty)
       (R_reg_selA R_ctr0 R_ctr0_outA R_ctr1 R_ctr1_outA R_ctr2 R_ctr2_outA R_ctr3 R_ctr3_outA R_icr_oldA
        R_icrA R_busA_latch R_reg_sel R_ctr0_in R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1_new R_ctr1_out
        R_ctr2_in R_ctr2_new R_ctr2_out R_ctr3_in R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr
        R_ccr R_gcr R_sr :wordn)
       (R_fsm_cntlatch R_fsm_srdy_ R_int0_en R_int0_disA R_int3_en R_int3_disA R_c01_cout R_c01_cout_delA
        R_c23_cout R_c23_cout_delA R_cntlatch_delA R_srdy_delA_ R_ctr0_ce R_ctr0_cin R_ctr1_ce R_ctr1_cin
        R_ctr2_ce R_ctr2_cin R_ctr3_ce R_ctr3_cin R_icr_loadA R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst
        R_int0_dis R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del
        R_srdy_del_ R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden R_ctr1_mux_sel R_ctr1_irden
        R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden R_ctr3_mux_sel
        R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden
        R_sr_rden :bool)
       (I_ad_in I_be_ Cpu_fail Reset_cpu S_state Id ChannelID C_ss :wordn)
       (ClkA ClkB Rst I_rale_ I_last_ I_mrdy_ Disable_int Disable_writes Piu_fail Pmm_fail
        CB_parity MB_parity :bool) .
    PH_A_out rep
```

```
            (R_fsm_stateA, R_fsm_cntlatch, R_fsm_srdy_, R_int0_en, R_int0_disA, R_int3_en, R_int3_disA,
             R_c01_cout, R_c01_cout_delA, R_c23_cout, R_c23_cout_delA, R_cntlatch_delA, R_srdy_delA_,
             R_reg_selA, R_ctr0, R_ctr0_ce, R_ctr0_cin, R_ctr0_outA, R_ctr1, R_ctr1_ce, R_ctr1_cin,
             R_ctr1_outA, R_ctr2, R_ctr2_ce, R_ctr2_cin, R_ctr2_outA, R_ctr3, R_ctr3_ce, R_ctr3_cin,
             R_ctr3_outA, R_icr_loadA, R_icr_oldA, R_icrA, R_busA_latch, R_fsm_state, R_fsm_ale_,
             R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_int0_dis, R_int3_dis, R_c01_cout_del, R_int1_en,
             R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_, R_reg_sel, R_ctr0_in,
             R_ctr0_mux_sel, R_ctr0_irden, R_ctr0_cry, R_ctr0_new, R_ctr0_out, R_ctr0_orden, R_ctr1_in,
             R_ctr1_mux_sel, R_ctr1_irden, R_ctr1_cry, R_ctr1_new, R_ctr1_out, R_ctr1_orden, R_ctr2_in,
             R_ctr2_mux_sel, R_ctr2_irden, R_ctr2_cry, R_ctr2_new, R_ctr2_out, R_ctr2_orden, R_ctr3_in,
             R_ctr3_mux_sel, R_ctr3_irden, R_ctr3_cry, R_ctr3_new, R_ctr3_out, R_ctr3_orden, R_icr_load,
             R_icr_old, R_icr_mask, R_icr, R_icr_rden, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr,
             R_sr_rden)
            (ClkA, ClkB, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
             Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss) =

let new_R_fsm_stateA =
      ((R_fsm_rst) => RI |
       ((R_fsm_state = RI) => ((~R_fsm_ale_) => RA | RI) |
       ((R_fsm_state = RA) => ((~R_fsm_mrdy_) => RD | RA) |
       ((~R_fsm_last_) => RI | RA)))) in
let new_R_fsm_cntlatch = ((R_fsm_state = RI) /\ ~R_fsm_ale_) in
let new_R_fsm_srdy_ = ~((R_fsm_state = RA) /\ ~R_fsm_mrdy_) in
let new_R_cntlatch_delA = R_cntlatch_del in
let new_R_srdy_delA_ = R_srdy_del_ in
let new_R_reg_selA = R_reg_sel in
let r_reg_sel = ((~new_R_srdy_delA_) => (INCN 3 new_R_reg_selA) | new_R_reg_selA) in
let r_write = (~Disable_writes /\ R_wr /\ (new_R_fsm_stateA = RD)) in
let r_read = (~R_wr /\ (new_R_fsm_stateA = RA)) in
let r_cir_wr01 = (r_write /\ ((r_reg_sel = (WORDN 8)) \/ (r_reg_sel = (WORDN 9)))) in
let r_cir_wr23 = (r_write /\ ((r_reg_sel = (WORDN 10)) \/ (r_reg_sel = (WORDN 11)))) in
let new_R_ctr0 = ((R_ctr0_mux_sel) => R_ctr0_in | R_ctr0_new) in
let new_R_ctr0_ce = (ELEMENT R_gcr (19)) in
let new_R_ctr0_cin = T in
let new_R_ctr0_outA = R_ctr0_new in
let new_R_ctr1 = ((R_ctr1_mux_sel) => R_ctr1_in | R_ctr1_new) in
let new_R_ctr1_ce = T in
let new_R_ctr1_cin = R_ctr0_cry in
let new_R_ctr1_outA = R_ctr1_new in
let new_R_ctr2 = ((R_ctr2_mux_sel) => R_ctr2_in | R_ctr2_new) in
let new_R_ctr2_ce = (ELEMENT R_gcr (23)) in
let new_R_ctr2_cin = T in
let new_R_ctr2_outA = R_ctr2_new in
let new_R_ctr3 = ((R_ctr3_mux_sel) => R_ctr3_in | R_ctr3_new) in
let new_R_ctr3_ce = T in
let new_R_ctr3_cin = R_ctr2_cry in
let new_R_ctr3_outA = R_ctr3_new in
let new_R_icr_loadA = R_icr_load in
let new_R_icr_oldA =
      (((new_R_fsm_stateA = RA) /\ ((r_reg_sel = (WORDN 0)) \/ (r_reg_sel = (WORDN 1)))) => R_icr | R_icr_oldA) in
let new_R_icrA =
      ((~(r_reg_sel = (WORDN 1))) => Andn rep (R_icr_old, R_icr_mask) | Orn rep (R_icr_old, R_icr_mask)) in
let new_R_int0_en = (((ELEMENT R_icr (0)) /\ (ELEMENT R_icr (8))) \/
                     ((ELEMENT R_icr (1)) /\ (ELEMENT R_icr (9))) \/
```

```
                    ((ELEMENT R_icr (2)) ∧ (ELEMENT R_icr (10))) ∨
                    ((ELEMENT R_icr (3)) ∧ (ELEMENT R_icr (11))) ∨
                    ((ELEMENT R_icr (4)) ∧ (ELEMENT R_icr (12))) ∨
                    ((ELEMENT R_icr (5)) ∧ (ELEMENT R_icr (13))) ∨
                    ((ELEMENT R_icr (6)) ∧ (ELEMENT R_icr (14))) ∨
                    ((ELEMENT R_icr (7)) ∧ (ELEMENT R_icr (15)))) in
let new_R_int0_disA = R_int0_dis in
let new_R_int3_en = (((ELEMENT R_icr (16)) ∧ (ELEMENT R_icr (24))) ∨
                    ((ELEMENT R_icr (17)) ∧ (ELEMENT R_icr (25))) ∨
                    ((ELEMENT R_icr (18)) ∧ (ELEMENT R_icr (26))) ∨
                    ((ELEMENT R_icr (19)) ∧ (ELEMENT R_icr (27))) ∨
                    ((ELEMENT R_icr (20)) ∧ (ELEMENT R_icr (28))) ∨
                    ((ELEMENT R_icr (21)) ∧ (ELEMENT R_icr (29))) ∨
                    ((ELEMENT R_icr (22)) ∧ (ELEMENT R_icr (30))) ∨
                    ((ELEMENT R_icr (23)) ∧ (ELEMENT R_icr (31)))) in
let new_R_int3_disA = R_int3_dis in
let new_R_c01_cout = R_ctr1_cry in
let new_R_c01_cout_delA = R_c01_cout_del in
let new_R_c23_cout = R_ctr3_cry in
let new_R_c23_cout_delA = R_c23_cout_del in
let new_R_busA_latch =
      (((R_ctr0_irden) => R_ctr0_in |
       ((R_ctr0_orden) => R_ctr0_out |
       ((R_ctr1_irden) => R_ctr1_in |
       ((R_ctr1_orden) => R_ctr1_out |
       ((R_ctr2_irden) => R_ctr2_in |
       ((R_ctr2_orden) => R_ctr2_out |
       ((R_ctr3_irden) => R_ctr3_in |
       ((R_ctr3_orden) => R_ctr3_out |
       ((R_icr_rden) => R_icr |
       ((R_ccr_rden) => R_ccr |
       ((R_gcr_rden) => R_gcr |
       ((R_sr_rden) => R_sr | ARBN)))))))))))) in
let new_R_fsm_state = R_fsm_state in
let new_R_fsm_ale_ = R_fsm_ale_ in
let new_R_fsm_mrdy_ = R_fsm_mrdy_ in
let new_R_fsm_last_ = R_fsm_last_ in
let new_R_fsm_rst = R_fsm_rst in
let new_R_int0_dis = R_int0_dis in
let new_R_int3_dis = R_int3_dis in
let new_R_c01_cout_del = R_c01_cout_del in
let new_R_int1_en = R_int1_en in
let new_R_c23_cout_del = R_c23_cout_del in
let new_R_int2_en = R_int2_en in
let new_R_wr = R_wr in
let new_R_cntlatch_del = R_cntlatch_del in
let new_R_srdy_del_ = R_srdy_del_ in
let new_R_reg_sel = R_reg_sel in
let new_R_ctr0_in = R_ctr0_in in
let new_R_ctr0_mux_sel = R_ctr0_mux_sel in
let new_R_ctr0_irden = R_ctr0_irden in
let new_R_ctr0_cry = R_ctr0_cry in
let new_R_ctr0_new = R_ctr0_new in
let new_R_ctr0_out = R_ctr0_out in
```

```
    let new_R_ctr0_orden = R_ctr0_orden in
    let new_R_ctr1_in = R_ctr1_in in
    let new_R_ctr1_mux_sel = R_ctr1_mux_sel in
    let new_R_ctr1_irden = R_ctr1_irden in
    let new_R_ctr1_cry = R_ctr1_cry in
    let new_R_ctr1_new = R_ctr1_new in
    let new_R_ctr1_out = R_ctr1_out in
    let new_R_ctr1_orden = R_ctr1_orden in
    let new_R_ctr2_in = R_ctr2_in in
    let new_R_ctr2_mux_sel = R_ctr2_mux_sel in
    let new_R_ctr2_irden = R_ctr2_irden in
    let new_R_ctr2_cry = R_ctr2_cry in
    let new_R_ctr2_new = R_ctr2_new in
    let new_R_ctr2_out = R_ctr2_out in
    let new_R_ctr2_orden = R_ctr2_orden in
    let new_R_ctr3_in = R_ctr3_in in
    let new_R_ctr3_mux_sel = R_ctr3_mux_sel in
    let new_R_ctr3_irden = R_ctr3_irden in
    let new_R_ctr3_cry = R_ctr3_cry in
    let new_R_ctr3_new = R_ctr3_new in
    let new_R_ctr3_out = R_ctr3_out in
    let new_R_ctr3_orden = R_ctr3_orden in
    let new_R_icr_load = R_icr_load in
    let new_R_icr_old = R_icr_old in
    let new_R_icr_mask = R_icr_mask in
    let new_R_icr = R_icr in
    let new_R_icr_rden = R_icr_rden in
    let new_R_ccr = R_ccr in
    let new_R_ccr_rden = R_ccr_rden in
    let new_R_gcr = R_gcr in
    let new_R_gcr_rden = R_gcr_rden in
    let new_R_sr = R_sr in
    let new_R_sr_rden = R_sr_rden in

    let I_ad_out = ((~new_R_wr ∧ ((new_R_fsm_stateA = RA) ∨ (new_R_fsm_stateA = RD))) => new_R_busA_latch | ARBN) in
    let I_srdy_ = (((new_R_fsm_stateA = RD) ∨ ((new_R_fsm_stateA = RA))) => new_R_fsm_srdy_ | ARB) in
    let Int0_ = ~(new_R_int0_en ∧ ~new_R_int0_disA ∧ ~Disable_int) in
    let Int1 = (new_R_c01_cout ∧ new_R_int1_en ∧ ~Disable_int) in
    let Int2 = (new_R_c23_cout ∧ new_R_int2_en ∧ ~Disable_int) in
    let Int3_ = ~(new_R_int3_en ∧ ~new_R_int3_disA ∧ ~Disable_int) in
    let Ccr = new_R_ccr in
    let Led = (SUBARRAY new_R_gcr (3,0)) in
    let Reset_error = (ELEMENT new_R_gcr (24)) in
    let Pmm_invalid = (ELEMENT new_R_gcr (28)) in

    (I_ad_out, I_srdy_, Int0_, Int1, Int2, Int3_, Ccr, Led, Reset_error, Pmm_invalid)"
    );;

%------------------------------------------------------------------------------------------------------------
    Next-state definition for Phase-B instruction.
------------------------------------------------------------------------------------------------------------%

let PH_B_inst_def = new_definition
('PH_B_inst',
"! (rep:^rep_ty)
   (R_fsm_stateA R_fsm_state :rfsm_ty)
   (R_reg_selA R_ctr0 R_ctr0_outA R_ctr1 R_ctr1_outA R_ctr2 R_ctr2_outA R_ctr3 R_ctr3_outA R_icr_oldA
    R_icrA R_busA_latch R_reg_sel R_ctr0_in R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1_new R_ctr1_out
    R_ctr2_in R_ctr2_new R_ctr2_out R_ctr3_in R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr
    R_ccr R_gcr R_sr :wordn)
   (R_fsm_cntlatch R_fsm_srdy_ R_int0_en R_int0_disA R_int3_en R_int3_disA R_c01_cout R_c01_cout_delA
    R_c23_cout R_c23_cout_delA R_cntlatch_delA R_srdy_delA_ R_ctr0_ce R_ctr0_cin R_ctr1_ce R_ctr1_cin
    R_ctr2_ce R_ctr2_cin R_ctr3_ce R_ctr3_cin R_icr_loadA R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst
    R_int0_dis R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del
    R_srdy_del_ R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden R_ctr1_mux_sel R_ctr1_irden
    R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden R_ctr3_mux_sel
    R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden
    R_sr_rden :bool)
   (I_ad_in I_be_ Cpu_fail Reset_cpu S_state Id ChannelID C_ss :wordn)
   (ClkA ClkB Rst I_rale_ I_last_ I_mrdy_ Disable_int Disable_writes Piu_fail Pmm_fail
    CB_parity MB_parity :bool) .
PH_B_inst rep
         (R_fsm_stateA, R_fsm_cntlatch, R_fsm_srdy_, R_int0_en, R_int0_disA, R_int3_en, R_int3_disA,
          R_c01_cout, R_c01_cout_delA, R_c23_cout, R_c23_cout_delA, R_cntlatch_delA, R_srdy_delA_,
          R_reg_selA, R_ctr0, R_ctr0_ce, R_ctr0_cin, R_ctr0_outA, R_ctr1, R_ctr1_ce, R_ctr1_cin,
          R_ctr1_outA, R_ctr2, R_ctr2_ce, R_ctr2_cin, R_ctr2_outA, R_ctr3, R_ctr3_ce, R_ctr3_cin,
          R_ctr3_outA, R_icr_loadA, R_icr_oldA, R_icrA, R_busA_latch, R_fsm_state, R_fsm_ale_,
          R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_int0_dis, R_int3_dis, R_c01_cout_del, R_int1_en,
          R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_, R_reg_sel, R_ctr0_in,
          R_ctr0_mux_sel, R_ctr0_irden, R_ctr0_cry, R_ctr0_new, R_ctr0_out, R_ctr0_orden, R_ctr1_in,
          R_ctr1_mux_sel, R_ctr1_irden, R_ctr1_cry, R_ctr1_new, R_ctr1_out, R_ctr1_orden, R_ctr2_in,
          R_ctr2_mux_sel, R_ctr2_irden, R_ctr2_cry, R_ctr2_new, R_ctr2_out, R_ctr2_orden, R_ctr3_in,
          R_ctr3_mux_sel, R_ctr3_irden, R_ctr3_cry, R_ctr3_new, R_ctr3_out, R_ctr3_orden, R_icr_load,
          R_icr_old, R_icr_mask, R_icr, R_icr_rden, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr,
          R_sr_rden)
         (ClkA, ClkB, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
          Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss) =

let new_R_wr = ((~I_rale_) => (ELEMENT I_ad_in (27)) | R_wr) in
let new_R_srdy_del_ = R_fsm_srdy_ in
let new_R_reg_sel =
      ((~I_rale_) => (SUBARRAY I_ad_in (3,0)) |
       ((~R_srdy_delA_) => (INCN 3 R_reg_selA) | R_reg_selA)) in
let new_R_cntlatch_del = R_fsm_cntlatch in
let r_reg_sel = ((~R_srdy_delA_) => (INCN 3 R_reg_selA) | R_reg_selA) in
let r_write = (~Disable_writes ∧ new_R_wr ∧ (R_fsm_stateA = RD)) in
let r_read = (~new_R_wr ∧ (R_fsm_stateA = RA)) in
let r_cir_wr01 = (r_write ∧ ((r_reg_sel = (WORDN 8)) ∨ (r_reg_sel = (WORDN 9)))) in
let r_cir_wr23 = (r_write ∧ ((r_reg_sel = (WORDN 10)) ∨ (r_reg_sel = (WORDN 11)))) in
let new_R_ccr = ((r_write ∧ (r_reg_sel = (WORDN 3))) => I_ad_in | R_ccr) in
let new_R_ccr_rden = (r_read ∧ (r_reg_sel = (WORDN 3))) in
let new_R_gcr = ((r_write ∧ (r_reg_sel = (WORDN 2))) => I_ad_in | R_gcr) in
let new_R_gcr_rden = (r_read ∧ (r_reg_sel = (WORDN 2))) in
let new_R_ctr0_in = ((r_write ∧ (r_reg_sel = (WORDN 8))) => I_ad_in | R_ctr0_in) in
let new_R_ctr0_mux_sel = (r_cir_wr01 ∨ ((ELEMENT new_R_gcr (16)) ∧ R_c01_cout)) in
let new_R_ctr0_irden = (r_read ∧ (r_reg_sel = (WORDN 8))) in
let new_R_ctr0_new = ((R_ctr0_ce ∧ R_ctr0_cin) => (INCN 31 R_ctr0) | R_ctr0) in
let new_R_ctr0_cry = (R_ctr0_ce ∧ R_ctr0_cin ∧ (ONES 31 R_ctr0)) in
let new_R_ctr0_out = ((R_fsm_cntlatch) => R_ctr0_outA | R_ctr0_out) in
let new_R_ctr0_orden = (r_read ∧ (r_reg_sel = (WORDN 12))) in
let new_R_ctr1_in = ((r_write ∧ (r_reg_sel = (WORDN 9))) => I_ad_in | R_ctr1_in) in
let new_R_ctr1_mux_sel = (r_cir_wr01 ∨ ((ELEMENT new_R_gcr (16)) ∧ R_c01_cout)) in
let new_R_ctr1_irden = (r_read ∧ (r_reg_sel = (WORDN 9))) in
let new_R_ctr1_new = ((R_ctr1_ce ∧ R_ctr1_cin) => (INCN 31 R_ctr1) | R_ctr1) in
let new_R_ctr1_cry = (R_ctr1_ce ∧ R_ctr1_cin ∧ (ONES 31 R_ctr1)) in
let new_R_ctr1_out = ((R_cntlatch_delA) => R_ctr1_outA | R_ctr1_out) in
let new_R_ctr1_orden = (r_read ∧ (r_reg_sel = (WORDN 13))) in
let new_R_ctr2_in = ((r_write ∧ (r_reg_sel = (WORDN 10))) => I_ad_in | R_ctr2_in) in
let new_R_ctr2_mux_sel = (r_cir_wr23 ∨ ((ELEMENT new_R_gcr (20)) ∧ R_c23_cout)) in
let new_R_ctr2_irden = (r_read ∧ (r_reg_sel = (WORDN 10))) in
let new_R_ctr2_new = ((R_ctr2_ce ∧ R_ctr2_cin) => (INCN 31 R_ctr2) | R_ctr2) in
let new_R_ctr2_cry = (R_ctr2_ce ∧ R_ctr2_cin ∧ (ONES 31 R_ctr2)) in
let new_R_ctr2_out = ((R_fsm_cntlatch) => R_ctr2_outA | R_ctr2_out) in
let new_R_ctr2_orden = (r_read ∧ (r_reg_sel = (WORDN 14))) in
let new_R_ctr3_in = ((r_write ∧ (r_reg_sel = (WORDN 11))) => I_ad_in | R_ctr3_in) in
let new_R_ctr3_mux_sel = (r_cir_wr23 ∨ ((ELEMENT new_R_gcr (20)) ∧ R_c23_cout)) in
let new_R_ctr3_irden = (r_read ∧ (r_reg_sel = (WORDN 11))) in
let new_R_ctr3_new = ((R_ctr3_ce ∧ R_ctr3_cin) => (INCN 31 R_ctr3) | R_ctr3) in
let new_R_ctr3_cry = (R_ctr3_ce ∧ R_ctr3_cin ∧ (ONES 31 R_ctr3)) in
let new_R_ctr3_out = ((R_cntlatch_delA) => R_ctr3_outA | R_ctr3_out) in
let new_R_ctr3_orden = (r_read ∧ (r_reg_sel = (WORDN 15))) in
let new_R_icr_load = (r_write ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let new_R_icr_old =
        ((r_write ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => R_icr_oldA | R_icr_old) in
let new_R_icr_mask =
        ((r_write ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => I_ad_in | R_icr_mask) in
let new_R_icr = ((R_icr_loadA) => R_icrA | R_icr) in
let new_R_icr_rden = ((R_fsm_stateA = RA) ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let sr28 = (ALTER ARBN (28) MB_parity) in
let sr28_25 = (MALTER sr28 (27,25) C_ss) in
let sr28_24 = (ALTER sr28_25 (24) CB_parity) in
let sr28_22 = (MALTER sr28_24 (23,22) ChannelID) in
let sr28_16 = (MALTER sr28_22 (21,16) Id) in
let sr28_12 = (MALTER sr28_16 (15,12) S_state) in
let sr28_9 = (ALTER sr28_12 (9) Pmm_fail) in
let sr28_8 = (ALTER sr28_9 (8) Piu_fail) in
let sr28_2 = (MALTER sr28_8 (3,2) Reset_cpu) in
let sr28_0 = (MALTER sr28_2 (1,0) Cpu_fail) in
let new_R_sr = ((R_fsm_cntlatch) => sr28_0 | R_sr) in
let new_R_sr_rden = (r_read ∧ (r_reg_sel = (WORDN 4))) in
let new_R_int0_dis = R_int0_en in
let new_R_int3_dis = R_int3_en in
let new_R_c01_cout_del = R_c01_cout in
let new_R_c23_cout_del = R_c23_cout in
let new_R_int1_en =
        ((((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01 ∨ (R_c01_cout ∧ (ELEMENT new_R_gcr (16)))))
         ∧ ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => T |
        ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01 ∨ (R_c01_cout ∧ (ELEMENT new_R_gcr (16)))))
         ∧ (~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => F |
        ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01 ∨ (R_c01_cout ∧ (ELEMENT new_R_gcr (16)))))
         ∧ ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => R_int1_en | ARB))) in
let new_R_int2_en =
        ((((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23 ∨ (R_c23_cout ∧ (ELEMENT new_R_gcr (20)))))
         ∧ ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => T |
        ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23 ∨ (R_c23_cout ∧ (ELEMENT new_R_gcr (20)))))
         ∧ (~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => F |
        ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23 ∨ (R_c23_cout ∧ (ELEMENT new_R_gcr (20)))))
         ∧ ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => R_int2_en | ARB))) in
let new_R_fsm_state = R_fsm_stateA in
let new_R_fsm_ale_ = I_rale_ in
let new_R_fsm_mrdy_ = I_mrdy_ in
let new_R_fsm_last_ = I_last_ in
let new_R_fsm_rst = Rst in
let new_R_fsm_stateA = R_fsm_stateA in
let new_R_fsm_cntlatch = R_fsm_cntlatch in
let new_R_fsm_srdy_ = R_fsm_srdy_ in
let new_R_int0_en = R_int0_en in
let new_R_int0_disA = R_int0_disA in
let new_R_int3_en = R_int3_en in
let new_R_int3_disA = R_int3_disA in
let new_R_c01_cout = R_c01_cout in
let new_R_c01_cout_delA = R_c01_cout_delA in
let new_R_c23_cout = R_c23_cout in
let new_R_c23_cout_delA = R_c23_cout_delA in
let new_R_cntlatch_delA = R_cntlatch_delA in
let new_R_srdy_delA_ = R_srdy_delA_ in
let new_R_reg_selA = R_reg_selA in
let new_R_ctr0 = R_ctr0 in
let new_R_ctr0_ce = R_ctr0_ce in
let new_R_ctr0_cin = R_ctr0_cin in
let new_R_ctr0_outA = R_ctr0_outA in
let new_R_ctr1 = R_ctr1 in
let new_R_ctr1_ce = R_ctr1_ce in
let new_R_ctr1_cin = R_ctr1_cin in
let new_R_ctr1_outA = R_ctr1_outA in
let new_R_ctr2 = R_ctr2 in
let new_R_ctr2_ce = R_ctr2_ce in
let new_R_ctr2_cin = R_ctr2_cin in
let new_R_ctr2_outA = R_ctr2_outA in
let new_R_ctr3 = R_ctr3 in
let new_R_ctr3_ce = R_ctr3_ce in
let new_R_ctr3_cin = R_ctr3_cin in
let new_R_ctr3_outA = R_ctr3_outA in
let new_R_icr_loadA = R_icr_loadA in
let new_R_icr_oldA = R_icr_oldA in
let new_R_icrA = R_icrA in
let new_R_busA_latch = R_busA_latch in

(new_R_fsm_stateA, new_R_fsm_cntlatch, new_R_fsm_srdy_, new_R_int0_en, new_R_int0_disA, new_R_int3_en,
 new_R_int3_disA, new_R_c01_cout, new_R_c01_cout_delA, new_R_c23_cout, new_R_c23_cout_delA,
 new_R_cntlatch_delA,
 new_R_srdy_delA_, new_R_reg_selA, new_R_ctr0, new_R_ctr0_ce, new_R_ctr0_cin, new_R_ctr0_outA, new_R_ctr1,
 new_R_ctr1_ce, new_R_ctr1_cin, new_R_ctr1_outA, new_R_ctr2, new_R_ctr2_ce, new_R_ctr2_cin, new_R_ctr2_outA,
 new_R_ctr3, new_R_ctr3_ce, new_R_ctr3_cin, new_R_ctr3_outA, new_R_icr_loadA, new_R_icr_oldA, new_R_icrA,
 new_R_busA_latch, new_R_fsm_state, new_R_fsm_ale_, new_R_fsm_mrdy_, new_R_fsm_last_, new_R_fsm_rst,
 new_R_int0_dis, new_R_int3_dis, new_R_c01_cout_del, new_R_int1_en, new_R_c23_cout_del, new_R_int2_en,
    new_R_wr,
    new_R_cntlatch_del, new_R_srdy_del_, new_R_reg_sel, new_R_ctr0_in, new_R_ctr0_mux_sel, new_R_ctr0_irden,
    new_R_ctr0_cry, new_R_ctr0_new, new_R_ctr0_out, new_R_ctr0_orden, new_R_ctr1_in, new_R_ctr1_mux_sel,
    new_R_ctr1_irden, new_R_ctr1_cry, new_R_ctr1_new, new_R_ctr1_out, new_R_ctr1_orden, new_R_ctr2_in,
    new_R_ctr2_mux_sel, new_R_ctr2_irden, new_R_ctr2_cry, new_R_ctr2_new, new_R_ctr2_out, new_R_ctr2_orden,
    new_R_ctr3_in, new_R_ctr3_mux_sel, new_R_ctr3_irden, new_R_ctr3_cry, new_R_ctr3_new, new_R_ctr3_out,
    new_R_ctr3_orden, new_R_icr_load, new_R_icr_old, new_R_icr_mask, new_R_icr, new_R_icr_rden, new_R_ccr,
    new_R_ccr_rden, new_R_gcr, new_R_gcr_rden, new_R_sr, new_R_sr_rden)"
  );;

%-------------------------------------------------------------------------------------------------------
  Output definition for Phase-B instruction.
-------------------------------------------------------------------------------------------------------%

let PH_B_out_def = new_definition
 ('PH_B_out',
    "! (rep:^rep_ty)
      (R_fsm_stateA R_fsm_state :rfsm_ty)
      (R_reg_selA R_ctr0 R_ctr0_outA R_ctr1 R_ctr1_outA R_ctr2 R_ctr2_outA R_ctr3 R_ctr3_outA R_icr_oldA
       R_icrA R_busA_latch R_reg_sel R_ctr0_in R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1_new R_ctr1_out
       R_ctr2_in R_ctr2_new R_ctr2_out R_ctr3_in R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr
       R_ccr R_gcr R_sr :wordn)
      (R_fsm_cntlatch R_fsm_srdy_ R_int0_en R_int0_disA R_int3_en R_int3_disA R_c01_cout R_c01_cout_delA
       R_c23_cout R_c23_cout_delA R_cntlatch_delA R_srdy_delA_ R_ctr0_ce R_ctr0_cin R_ctr1_ce R_ctr1_cin
       R_ctr2_ce R_ctr2_cin R_ctr3_ce R_ctr3_cin R_icr_loadA R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst
       R_int0_dis R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del
       R_srdy_del_ R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden R_ctr1_mux_sel R_ctr1_irden
       R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden R_ctr3_mux_sel
       R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden
       R_sr_rden :bool)
      (I_ad_in I_be_ Cpu_fail Reset_cpu S_state Id ChannelID C_ss :wordn)
      (ClkA ClkB Rst I_rale_ I_last_ I_mrdy_ Disable_int Disable_writes Piu_fail Pmm_fail
       CB_parity MB_parity :bool) .
   PH_B_out rep
           (R_fsm_stateA, R_fsm_cntlatch, R_fsm_srdy_, R_int0_en, R_int0_disA, R_int3_en, R_int3_disA,
            R_c01_cout, R_c01_cout_delA, R_c23_cout, R_c23_cout_delA, R_cntlatch_delA, R_srdy_delA_,
            R_reg_selA, R_ctr0, R_ctr0_ce, R_ctr0_cin, R_ctr0_outA, R_ctr1, R_ctr1_ce, R_ctr1_cin,
            R_ctr1_outA, R_ctr2, R_ctr2_ce, R_ctr2_cin, R_ctr2_outA, R_ctr3, R_ctr3_ce, R_ctr3_cin,
            R_ctr3_outA, R_icr_loadA, R_icr_oldA, R_icrA, R_busA_latch, R_fsm_state, R_fsm_ale_,
            R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_int0_dis, R_int3_dis, R_c01_cout_del, R_int1_en,
            R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_, R_reg_sel, R_ctr0_in,
            R_ctr0_mux_sel, R_ctr0_irden, R_ctr0_cry, R_ctr0_new, R_ctr0_out, R_ctr0_orden, R_ctr1_in,
            R_ctr1_mux_sel, R_ctr1_irden, R_ctr1_cry, R_ctr1_new, R_ctr1_out, R_ctr1_orden, R_ctr2_in,
            R_ctr2_mux_sel, R_ctr2_irden, R_ctr2_cry, R_ctr2_new, R_ctr2_out, R_ctr2_orden, R_ctr3_in,
            R_ctr3_mux_sel, R_ctr3_irden, R_ctr3_cry, R_ctr3_new, R_ctr3_out, R_ctr3_orden, R_icr_load,
            R_icr_old, R_icr_mask, R_icr, R_icr_rden, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr,
            R_sr_rden)
           (ClkA, ClkB, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
            Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss) =

    let new_R_wr = ((~I_rale_) => (ELEMENT I_ad_in (27)) | R_wr) in
    let new_R_srdy_del_ = R_fsm_srdy_ in
    let new_R_reg_sel =
          ((~I_rale_) => (SUBARRAY I_ad_in (3,0)) |
        ((~R_srdy_delA_) => (INCN 3 R_reg_selA) | R_reg_selA)) in
let new_R_cntlatch_del = R_fsm_cntlatch in
let r_reg_sel = ((~R_srdy_delA_) => (INCN 3 R_reg_selA) | R_reg_selA) in
let r_write = (~Disable_writes ∧ new_R_wr ∧ (R_fsm_stateA = RD)) in
let r_read = (~new_R_wr ∧ (R_fsm_stateA = RA)) in
let r_cir_wr01 = (r_write ∧ ((r_reg_sel = (WORDN 8)) ∨ (r_reg_sel = (WORDN 9)))) in
let r_cir_wr23 = (r_write ∧ ((r_reg_sel = (WORDN 10)) ∨ (r_reg_sel = (WORDN 11)))) in
let new_R_ccr = ((r_write ∧ (r_reg_sel = (WORDN 3))) => I_ad_in | R_ccr) in
let new_R_ccr_rden = (r_read ∧ (r_reg_sel = (WORDN 3))) in
let new_R_gcr = ((r_write ∧ (r_reg_sel = (WORDN 2))) => I_ad_in | R_gcr) in
let new_R_gcr_rden = (r_read ∧ (r_reg_sel = (WORDN 2))) in
let new_R_ctr0_in = ((r_write ∧ (r_reg_sel = (WORDN 8))) => I_ad_in | R_ctr0_in) in
let new_R_ctr0_mux_sel = (r_cir_wr01 ∨ ((ELEMENT new_R_gcr (16)) ∧ R_c01_cout)) in
let new_R_ctr0_irden = (r_read ∧ (r_reg_sel = (WORDN 8))) in
let new_R_ctr0_new = ((R_ctr0_ce ∧ R_ctr0_cin) => (INCN 31 R_ctr0) | R_ctr0) in
let new_R_ctr0_cry = (R_ctr0_ce ∧ R_ctr0_cin ∧ (ONES 31 R_ctr0)) in
let new_R_ctr0_out = ((R_fsm_cntlatch) => R_ctr0_outA | R_ctr0_out) in
let new_R_ctr0_orden = (r_read ∧ (r_reg_sel = (WORDN 12))) in
let new_R_ctr1_in = ((r_write ∧ (r_reg_sel = (WORDN 9))) => I_ad_in | R_ctr1_in) in
let new_R_ctr1_mux_sel = (r_cir_wr01 ∨ ((ELEMENT new_R_gcr (16)) ∧ R_c01_cout)) in
let new_R_ctr1_irden = (r_read ∧ (r_reg_sel = (WORDN 9))) in
let new_R_ctr1_new = ((R_ctr1_ce ∧ R_ctr1_cin) => (INCN 31 R_ctr1) | R_ctr1) in
let new_R_ctr1_cry = (R_ctr1_ce ∧ R_ctr1_cin ∧ (ONES 31 R_ctr1)) in
let new_R_ctr1_out = ((R_cntlatch_delA) => R_ctr1_outA | R_ctr1_out) in
let new_R_ctr1_orden = (r_read ∧ (r_reg_sel = (WORDN 13))) in
let new_R_ctr2_in = ((r_write ∧ (r_reg_sel = (WORDN 10))) => I_ad_in | R_ctr2_in) in
let new_R_ctr2_mux_sel = (r_cir_wr23 ∨ ((ELEMENT new_R_gcr (20)) ∧ R_c23_cout)) in
let new_R_ctr2_irden = (r_read ∧ (r_reg_sel = (WORDN 10))) in
let new_R_ctr2_new = ((R_ctr2_ce ∧ R_ctr2_cin) => (INCN 31 R_ctr2) | R_ctr2) in
let new_R_ctr2_cry = (R_ctr2_ce ∧ R_ctr2_cin ∧ (ONES 31 R_ctr2)) in
let new_R_ctr2_out = ((R_fsm_cntlatch) => R_ctr2_outA | R_ctr2_out) in
let new_R_ctr2_orden = (r_read ∧ (r_reg_sel = (WORDN 14))) in
let new_R_ctr3_in = ((r_write ∧ (r_reg_sel = (WORDN 11))) => I_ad_in | R_ctr3_in) in
let new_R_ctr3_mux_sel = (r_cir_wr23 ∨ ((ELEMENT new_R_gcr (20)) ∧ R_c23_cout)) in
let new_R_ctr3_irden = (r_read ∧ (r_reg_sel = (WORDN 11))) in
let new_R_ctr3_new = ((R_ctr3_ce ∧ R_ctr3_cin) => (INCN 31 R_ctr3) | R_ctr3) in
let new_R_ctr3_cry = (R_ctr3_ce ∧ R_ctr3_cin ∧ (ONES 31 R_ctr3)) in
let new_R_ctr3_out = ((R_cntlatch_delA) => R_ctr3_outA | R_ctr3_out) in
let new_R_ctr3_orden = (r_read ∧ (r_reg_sel = (WORDN 15))) in
let new_R_icr_load = (r_write ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let new_R_icr_old =
      ((r_write ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => R_icr_oldA | R_icr_old) in
let new_R_icr_mask =
      ((r_write ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => I_ad_in | R_icr_mask) in
let new_R_icr = ((R_icr_loadA) => R_icrA | R_icr) in
let new_R_icr_rden = ((R_fsm_stateA = RA) ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let sr28 = (ALTER ARBN (28) MB_parity) in
let sr28_25 = (MALTER sr28 (27,25) C_ss) in
let sr28_24 = (ALTER sr28_25 (24) CB_parity) in
let sr28_22 = (MALTER sr28_24 (23,22) ChannelID) in
let sr28_16 = (MALTER sr28_22 (21,16) Id) in
let sr28_12 = (MALTER sr28_16 (15,12) S_state) in
let sr28_9 = (ALTER sr28_12 (9) Pmm_fail) in
let sr28_8 = (ALTER sr28_9 (8) Piu_fail) in
let sr28_2 = (MALTER sr28_8 (3,2) Reset_cpu) in
let sr28_0 = (MALTER sr28_2 (1,0) Cpu_fail) in
let new_R_sr = ((R_fsm_cntlatch) => sr28_0 | R_sr) in
let new_R_sr_rden = (r_read ∧ (r_reg_sel = (WORDN 4))) in
let new_R_int0_dis = R_int0_en in
let new_R_int3_dis = R_int3_en in
let new_R_c01_cout_del = R_c01_cout in
let new_R_c23_cout_del = R_c23_cout in
let new_R_int1_en =
      ((((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01 ∨ (R_c01_cout ∧ (ELEMENT new_R_gcr (16)))))
        ∧ ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => T |
       ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01 ∨ (R_c01_cout ∧ (ELEMENT new_R_gcr (16)))))
        ∧ (~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => F |
       ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01 ∨ (R_c01_cout ∧ (ELEMENT new_R_gcr (16)))))
        ∧ ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => R_int1_en | ARB))) in
let new_R_int2_en =
      ((((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23 ∨ (R_c23_cout ∧ (ELEMENT new_R_gcr (20)))))
        ∧ ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => T |
       ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23 ∨ (R_c23_cout ∧ (ELEMENT new_R_gcr (20)))))
        ∧ (~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => F |
       ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23 ∨ (R_c23_cout ∧ (ELEMENT new_R_gcr (20)))))
        ∧ ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => R_int2_en | ARB))) in
let new_R_fsm_state = R_fsm_stateA in
let new_R_fsm_ale_ = I_rale_ in
let new_R_fsm_mrdy_ = I_mrdy_ in
let new_R_fsm_last_ = I_last_ in
let new_R_fsm_rst = Rst in
let new_R_fsm_stateA = R_fsm_stateA in
let new_R_fsm_cntlatch = R_fsm_cntlatch in
let new_R_fsm_srdy_ = R_fsm_srdy_ in
let new_R_int0_en = R_int0_en in
let new_R_int0_disA = R_int0_disA in
let new_R_int3_en = R_int3_en in
let new_R_int3_disA = R_int3_disA in
let new_R_c01_cout = R_c01_cout in
let new_R_c01_cout_delA = R_c01_cout_delA in
let new_R_c23_cout = R_c23_cout in
let new_R_c23_cout_delA = R_c23_cout_delA in
let new_R_cntlatch_delA = R_cntlatch_delA in
let new_R_srdy_delA_ = R_srdy_delA_ in
let new_R_reg_selA = R_reg_selA in
let new_R_ctr0 = R_ctr0 in
let new_R_ctr0_ce = R_ctr0_ce in
let new_R_ctr0_cin = R_ctr0_cin in
let new_R_ctr0_outA = R_ctr0_outA in
let new_R_ctr1 = R_ctr1 in
let new_R_ctr1_ce = R_ctr1_ce in
let new_R_ctr1_cin = R_ctr1_cin in
let new_R_ctr1_outA = R_ctr1_outA in
let new_R_ctr2 = R_ctr2 in
let new_R_ctr2_ce = R_ctr2_ce in
let new_R_ctr2_cin = R_ctr2_cin in
let new_R_ctr2_outA = R_ctr2_outA in
let new_R_ctr3 = R_ctr3 in
```

```
    let new_R_ctr3_ce = R_ctr3_ce in
    let new_R_ctr3_cin = R_ctr3_cin in
    let new_R_ctr3_outA = R_ctr3_outA in
    let new_R_icr_loadA = R_icr_loadA in
    let new_R_icr_oldA = R_icr_oldA in
    let new_R_icrA = R_icrA in
    let new_R_busA_latch = R_busA_latch in

    let I_ad_out = ((~new_R_wr ∧ ((new_R_fsm_stateA = RA) ∨ (new_R_fsm_stateA = RD))) => new_R_busA_latch | ARBN) in
    let I_srdy_ = (((new_R_fsm_stateA = RD) ∨ ((new_R_fsm_stateA = RA))) => new_R_fsm_srdy_ | ARB) in
    let Int0_ = ~(new_R_int0_en ∧ ~new_R_int0_disA ∧ ~Disable_int) in
    let Int1 = (new_R_c01_cout ∧ new_R_int1_en ∧ ~Disable_int) in
    let Int2 = (new_R_c23_cout ∧ new_R_int2_en ∧ ~Disable_int) in
    let Int3_ = ~(new_R_int3_en ∧ ~new_R_int3_disA ∧ ~Disable_int) in
    let Ccr = new_R_ccr in
    let Led = (SUBARRAY new_R_gcr (3,0)) in
    let Reset_error = (ELEMENT new_R_gcr (24)) in
    let Pmm_invalid = (ELEMENT new_R_gcr (28)) in

    (I_ad_out, I_srdy_, Int0_, Int1, Int2, Int3_, Ccr, Led, Reset_error, Pmm_invalid)"
    );;

close_theory();;
```

C.4 C Port Specification

```
%-----------------------------------------------------------------------------------------------

    File:          c_phase.ml

    Author:        (c) D.A. Fura 1992

    Date:          31 March 1992

    This file contains the ml source for the phase-level specification of the C-Port of the FTEP PIU,
    an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
    The bulk of this code was translated from an M-language simulation program using a translator
    written by P.J. Windley at the University of Idaho.

-----------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm c_phase.th';;

new_theory 'c_phase';;

loadf 'abstract';;

map new_parent ['caux_def';'aux_def';'array_def';'wordn_def'];;

let MSTART = "WORDN 4";;
let MEND = "WORDN 5";;
let MRDY = "WORDN 6";;
let MWAIT = "WORDN 7";;
let MABORT = "WORDN 0";;

let SACK = "WORDN 5";;
let SRDY = "WORDN 6";;
let SWAIT = "WORDN 7";;
let SABORT = "WORDN 0";;

let c_state_ty = ":(cmfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                  wordn#bool#bool#bool#bool#bool#
                  csfsm_ty#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                  cefsm_ty#bool#
                  bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn#wordn#
                  cmfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#bool#
                  csfsm_ty#bool#bool#bool#bool#bool#bool#wordn#
                  cefsm_ty#bool#bool#bool#bool#bool#bool#
                  bool#wordn#bool#bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool#bool#bool#
                  bool#bool#wordn#wordn#wordn)";;
let c_state = "((C_mfsm_stateA, C_mfsm_mabort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
               C_mfsm_ma1,
               C_mfsm_ma0,C_mfsm_md1,C_mfsm_md0,C_mfsm_iad_en_m,C_mfsm_m_cout_sel1,C_mfsm_m_cout_sel0,
               C_mfsm_ms,C_mfsm_rqt_,C_mfsm_cgnt_,C_mfsm_cm_en,C_mfsm_abort_le_en_,C_mfsm_mparity,
               C_sfsm_stateA,C_sfsm_ss,C_sfsm_iad_en_s,C_sfsm_sidle,C_sfsm_slock,C_sfsm_sa1,C_sfsm_sa0,
                C_sfsm_sale,C_sfsm_sd1,C_sfsm_sd0,C_sfsm_sack,C_sfsm_sabort,C_sfsm_s_cout_sel0,C_sfsm_sparity,
                C_efsm_stateA,C_efsm_srdy_en,
                C_clkAA,C_sidle_delA,C_mrqt_delA,C_last_inA_,C_ssA,C_holdA_,C_cout_0_le_delA,
                C_cin_2_leA,C_mrdy_delA_,C_iad_en_s_delA,C_wrdyA,C_rrdyA,C_iad_out,C_a1a0,C_a3a2,
                C_mfsm_state,C_mfsm_srdy_en,C_mfsm_D,C_mfsm_grant,C_mfsm_rst,C_mfsm_busy,C_mfsm_write,
                C_mfsm_crqt_,C_mfsm_hold_,C_mfsm_last_,C_mfsm_lock_,C_mfsm_ss,C_mfsm_invalid,
                C_sfsm_state,C_sfsm_D,C_sfsm_grant,C_sfsm_rst,C_sfsm_write,C_sfsm_addressed,C_sfsm_hlda_,C_sfsm_ms,
                C_efsm_state,C_efsm_cale_,C_efsm_last_,C_efsm_male_,C_efsm_rale_,C_efsm_srdy_,C_efsm_rst,
                C_wr,C_sizewrbe,C_clkA,C_sidle_del,C_mrqt_del,C_last_in_,C_lock_in_,C_ss,C_last_out_,
                C_hold_,C_cout_0_le_del,C_cin_2_le,C_mrdy_del_,C_iad_en_s_del,C_wrdy,
                C_rrdy,C_parity,C_source,C_data_in,C_iad_in)
               ^c_state_ty)";;

let c_env_ty = ":(wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                 wordn#wordn#wordn#wordn#bool#bool#bool#bool#wordn#wordn#bool#bool#wordn#bool)";;
let c_env = "((I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_,
               I_lock_, I_cale_, I_hlda_, I_crqt_,
               CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in,
               Rst, ClkA, ClkB, ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr,
               Reset_error)
              :^c_env_ty)";;

let c_out_ty = ":(bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
                 bool#wordn#wordn#wordn#wordn#bool#bool)";;
let c_out = "((I_cgnt_, I_mrdy_out_, I_hold_, I_rale_out_, I_male_out_, I_last_out_, I_srdy_out_,
              I_ad_out, I_be_out_,
              CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out, C_ss_out, Disable_writes, CB_parity)
             :^c_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%------------------------------------------------------------------------------------------------
   Next-state definition for Phase-A instruction.
------------------------------------------------------------------------------------------------%

let PH_A_inst_def = new_definition
 ('PH_A_inst',
    "! (rep:^rep_ty)
        (C_mfsm_stateA C_mfsm_state :cmfsm_ty)
        (C_sfsm_stateA C_sfsm_state :csfsm_ty)
        (C_efsm_stateA C_efsm_state :cefsm_ty)
        (C_mfsm_ms C_sfsm_ss C_ssA C_iad_out C_a1a0 C_a3a2 C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss
         C_source C_data_in C_iad_in :wordn)
        (C_mfsm_mabort C_mfsm_midle C_mfsm_mrequest C_mfsm_ma3 C_mfsm_ma2 C_mfsm_ma1
         C_mfsm_ma0 C_mfsm_md1 C_mfsm_md0 C_mfsm_iad_en_m C_mfsm_m_cout_sel1 C_mfsm_m_cout_sel0
         C_mfsm_rqt_ C_mfsm_cgnt_ C_mfsm_cm_en C_mfsm_abort_le_en_ C_mfsm_mparity
         C_sfsm_iad_en_s C_sfsm_sidle C_sfsm_slock C_sfsm_sa1 C_sfsm_sa0
         C_sfsm_sale C_sfsm_sd1 C_sfsm_sd0 C_sfsm_sack C_sfsm_sabort C_sfsm_s_cout_sel0 C_sfsm_sparity
         C_efsm_srdy_en
         C_clkAA C_sidle_delA C_mrqt_delA C_last_inA_ C_holdA_ C_cout_0_le_delA
         C_cin_2_leA C_mrdy_delA_ C_iad_en_s_delA C_wrdyA C_rrdyA
         C_mfsm_srdy_en C_mfsm_D C_mfsm_grant C_mfsm_rst C_mfsm_busy C_mfsm_write
         C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_last_ C_mfsm_lock_ C_mfsm_invalid
         C_sfsm_D C_sfsm_grant C_sfsm_rst C_sfsm_write C_sfsm_addressed C_sfsm_hlda_
        C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
        C_wr C_clkA C_sidle_del C_mrqt_del C_last_in_ C_lock_in_ C_last_out_
        C_hold_ C_cout_0_le_del C_cin_2_le C_mrdy_del_ C_iad_en_s_del C_wrdy
        C_rrdy C_parity :bool)
       (I_mrdy_in_ I_rale_in_ I_male_in_ I_last_in_ I_srdy_in_ I_lock_ I_cale_ I_hlda_ I_crqt_
        Rst ClkA ClkB ClkD Pmm_failure Piu_invalid Reset_error :bool)
       (I_ad_in I_be_in_ CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID Ccr :wordn)
       (I_cgnt_ I_mrdy_out_ I_hold_ I_rale_out_ I_male_out_ I_last_out_ I_srdy_out_ CB_rqt_out_
        Disable_writes CB_parity :bool) .
PH_A_inst rep
            (C_mfsm_stateA, C_mfsm_mabort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
             C_mfsm_ma1, C_mfsm_ma0, C_mfsm_md1, C_mfsm_md0, C_mfsm_iad_en_m, C_mfsm_m_cout_sel1,
             C_mfsm_m_cout_sel0, C_mfsm_ms, C_mfsm_rqt_, C_mfsm_cgnt_, C_mfsm_cm_en, C_mfsm_abort_le_en_,
             C_mfsm_mparity, C_sfsm_stateA, C_sfsm_ss, C_sfsm_iad_en_s, C_sfsm_sidle, C_sfsm_slock,
             C_sfsm_sa1, C_sfsm_sa0, C_sfsm_sale, C_sfsm_sd1, C_sfsm_sd0, C_sfsm_sack, C_sfsm_sabort,
             C_sfsm_s_cout_sel0, C_sfsm_sparity, C_efsm_stateA, C_efsm_srdy_en, C_clkAA, C_sidle_delA,
             C_mrqt_delA, C_last_inA_, C_ssA, C_holdA_, C_cout_0_le_delA, C_cin_2_leA,
             C_mrdy_delA_, C_iad_en_s_delA, C_wrdyA, C_rrdyA, C_iad_out, C_a1a0, C_a3a2, C_mfsm_state,
             C_mfsm_srdy_en, C_mfsm_D, C_mfsm_grant, C_mfsm_rst, C_mfsm_busy, C_mfsm_write, C_mfsm_crqt_,
             C_mfsm_hold_, C_mfsm_last_, C_mfsm_lock_, C_mfsm_ss, C_mfsm_invalid, C_sfsm_state, C_sfsm_D,
             C_sfsm_grant, C_sfsm_rst, C_sfsm_write, C_sfsm_addressed, C_sfsm_hlda_, C_sfsm_ms,
             C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_,
             C_efsm_rst, C_wr, C_sizewrbe, C_clkA, C_sidle_del, C_mrqt_del, C_last_in_, C_lock_in_,
             C_ss, C_last_out_, C_hold_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_wrdy,
             C_rrdy, C_parity, C_source, C_data_in, C_iad_in)
            (I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_, I_lock_,
             I_cale_, I_hlda_, I_crqt_, CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in, Rst, ClkA, ClkB,
             ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr, Reset_error) =

 let new_C_mfsm_stateA =
      ((C_mfsm_rst) => CMI |
      ((C_mfsm_state = CMI) => (C_mfsm_D ∧ ~C_mfsm_crqt_ ∧ ~C_mfsm_busy ∧ ~C_mfsm_invalid) => CMR | CMI |
      ((C_mfsm_state = CMR) => (C_mfsm_D ∧ C_mfsm_grant ∧ C_mfsm_hold_) => CMA3 | CMR |
      ((C_mfsm_state = CMA3) => ((C_mfsm_D) => CMA1 | CMA3) |
      ((C_mfsm_state = CMA1) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA0 |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA1 |
      ((C_mfsm_state = CMA0) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA2 |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA0 |
      ((C_mfsm_state = CMA2) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD1 |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA2 |
      ((C_mfsm_state = CMD1) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD0 |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD1 |
      ((C_mfsm_state = CMD0) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ C_mfsm_last_) => CMD1 |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_mfsm_last_) => CMW |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD0 |
      ((C_mfsm_state = CMW) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SACK) ∧ C_mfsm_lock_) => CMI |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_mfsm_lock_ ∧ ~C_mfsm_crqt_) => CMA3 | CMW |
        ((~C_mfsm_last_) => CMI | CMABT)))))))))) in

let new_C_mfsm_mabort = (new_C_mfsm_stateA = CMABT) in
let new_C_mfsm_midle = (new_C_mfsm_stateA = CMI) in
let new_C_mfsm_mrequest = (new_C_mfsm_stateA = CMR) in
let new_C_mfsm_ma3 = (new_C_mfsm_stateA = CMA3) in
let new_C_mfsm_ma2 = (new_C_mfsm_stateA = CMA2) in
let new_C_mfsm_ma1 = (new_C_mfsm_stateA = CMA1) in
let new_C_mfsm_ma0 = (new_C_mfsm_stateA = CMA0) in
let new_C_mfsm_md1 = (new_C_mfsm_stateA = CMD1) in
let new_C_mfsm_md0 = (new_C_mfsm_stateA = CMD0) in
let new_C_mfsm_iad_en_m = (((new_C_mfsm_stateA = CMD1) ∧ ~C_mfsm_write ∧ C_mfsm_srdy_en)
                          ∨ ((new_C_mfsm_stateA = CMD0) ∧ ~C_mfsm_write ∧ C_mfsm_srdy_en)
                          ∨ ((new_C_mfsm_stateA = CMW) ∧ (C_mfsm_state = CMD0) ∧ ~C_mfsm_write
                                                                        ∧ C_mfsm_srdy_en)) in
let new_C_mfsm_m_cout_sel1 = ((new_C_mfsm_stateA = CMA3) ∨ (new_C_mfsm_stateA = CMA2)) in
let new_C_mfsm_m_cout_sel0 = ((new_C_mfsm_stateA = CMA3) ∨ (new_C_mfsm_stateA = CMA1)
                                                  ∨ (new_C_mfsm_stateA = CMD1)) in
let ms2 = (ALTER ARBN (2) ((new_C_mfsm_stateA = CMA3) ∨ (new_C_mfsm_stateA = CMA1) ∨
                           (new_C_mfsm_stateA = CMA0) ∨ (new_C_mfsm_stateA = CMA2) ∨
                           (new_C_mfsm_stateA = CMD1) ∨ (new_C_mfsm_stateA = CMD0) ∨
                           (new_C_mfsm_stateA = CMW) ∨ (new_C_mfsm_stateA = CMABT))) in
let ms1 = (ALTER ms2 (1) ((new_C_mfsm_stateA = CMA1) ∨ (new_C_mfsm_stateA = CMA0) ∨
                          (new_C_mfsm_stateA = CMA2) ∨ (new_C_mfsm_stateA = CMD1) ∨
                          ((new_C_mfsm_stateA = CMD0) ∧ C_mfsm_last_) ∨ (new_C_mfsm_stateA = CMW) ∨
                          (new_C_mfsm_stateA = CMABT))) in
let ms0 = (ALTER ms1 (0) (((new_C_mfsm_stateA = CMD0) ∧ ~C_mfsm_last_) ∨
                          ((new_C_mfsm_stateA = CMW) ∧ C_mfsm_lock_) ∨ (new_C_mfsm_stateA = CMABT))) in
let new_C_mfsm_ms = ms0 in
let new_C_mfsm_rqt_ = ~(~(new_C_mfsm_stateA = CMI)) in
let new_C_mfsm_cgnt_ = ~(new_C_mfsm_stateA = CMA3) in
let new_C_mfsm_cm_en = ((~(new_C_mfsm_stateA = CMI)) ∧ (~(new_C_mfsm_stateA = CMR))) in
let new_C_mfsm_abort_le_en_ = ~((new_C_mfsm_stateA = CMABT) ∨ (new_C_mfsm_stateA = CMI)) in
let new_C_mfsm_mparity = ((new_C_mfsm_stateA = CMA3) ∨ (new_C_mfsm_stateA = CMA1)
                         ∨ (new_C_mfsm_stateA = CMA0) ∨ (new_C_mfsm_stateA = CMA2)
                         ∨ (new_C_mfsm_stateA = CMD1) ∨ (new_C_mfsm_stateA = CMD0)
                         ∨ (C_mfsm_state = CMA1) ∨ (C_mfsm_state = CMA0)
                         ∨ (C_mfsm_state = CMA2) ∨ (C_mfsm_state = CMD1)) in

let new_C_sfsm_stateA =
        ((C_sfsm_rst) => CSI |
         (C_sfsm_state = CSI) => ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART)
                                      ∧ ~C_sfsm_grant ∧ C_sfsm_addressed) => CSA1 | CSI) |
         (C_sfsm_state = CSL) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~C_sfsm_grant ∧ C_sfsm_addressed) => CSA1 |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~C_sfsm_grant ∧ ~C_sfsm_addressed) => CSI |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSL) |
         (C_sfsm_state = CSA1) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSA0 |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA1) |
         (C_sfsm_state = CSA0) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ C_sfsm_hlda_) => CSA0W |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0) |
        (C_sfsm_state = CSA0W) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0W) |
        (C_sfsm_state = CSALE) =>
              ((C_sfsm_D ∧ C_sfsm_write ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
               (C_sfsm_D ∧ ~C_sfsm_write ∧ (C_sfsm_ms = ^MRDY)) => CSRR |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSALE) |
        (C_sfsm_state = CSRR) =>
              ((C_sfsm_D ∧ ~(C_sfsm_ms = ^MABORT)) => CSD1 |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSRR) |
        (C_sfsm_state = CSD1) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD0 |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD1) |
        (C_sfsm_state = CSD0) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MEND)) => CSACK |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD0) |
        (C_sfsm_state = CSACK) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSL |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MWAIT)) => CSI |
               (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSACK) |
               (C_sfsm_D) => CSI | CSABT) in

let ss2 = (ALTER ARBN (2) ((~(new_C_sfsm_stateA = CSI)) ∧ (~(new_C_sfsm_stateA = CSABT)))) in
let ss1 = (ALTER ss2 (1) ((~(new_C_sfsm_stateA = CSI)) ∧ (~(new_C_sfsm_stateA = CSACK))
                                                   ∧ (~(new_C_sfsm_stateA = CSABT)))) in
let ss0 = (ALTER ss1 (0) ((new_C_sfsm_stateA = CSA0W) V
                         ((new_C_sfsm_stateA = CSALE) ∧ ~C_sfsm_write) V
                         (new_C_sfsm_stateA = CSACK))) in
let new_C_sfsm_ss = ss0 in
let new_C_sfsm_iad_en_s = (((new_C_sfsm_stateA = CSALE) ∧ (~(C_sfsm_state = CSALE)))
                          V ((new_C_sfsm_stateA = CSALE) ∧ C_sfsm_write)
                          V ((new_C_sfsm_stateA = CSD1) ∧ C_sfsm_write ∧ (~(C_sfsm_state = CSRR)))
                          V ((new_C_sfsm_stateA = CSD0) ∧ C_sfsm_write)
                          V ((new_C_sfsm_stateA = CSACK) ∧ C_sfsm_write)) in
let new_C_sfsm_sidle = (new_C_sfsm_stateA = CSI) in
let new_C_sfsm_slock = (new_C_sfsm_stateA = CSL) in
let new_C_sfsm_sa1 = (new_C_sfsm_stateA = CSA1) in
let new_C_sfsm_sa0 = (new_C_sfsm_stateA = CSA0) in
let new_C_sfsm_sale = (new_C_sfsm_stateA = CSALE) in
let new_C_sfsm_sd1 = (new_C_sfsm_stateA = CSD1) in
let new_C_sfsm_sd0 = (new_C_sfsm_stateA = CSD0) in
let new_C_sfsm_sack = (new_C_sfsm_stateA = CSACK) in
let new_C_sfsm_sabort = (new_C_sfsm_stateA = CSABT) in
let new_C_sfsm_s_cout_sel0 = (new_C_sfsm_stateA = CSD1) in
let new_C_sfsm_sparity = ((~(new_C_sfsm_stateA = CSI)) ∧ (~(new_C_sfsm_stateA = CSACK))
                                                   ∧ (~(new_C_sfsm_stateA = CSABT))) in
let new_C_efsm_stateA =
        ((C_efsm_rst) => CEI |
         (C_efsm_state = CEI) => ((~C_efsm_cale_) => CEE | CEI) |
         ((~C_efsm_last_ ∧ ~C_efsm_srdy_) V ~C_efsm_male_ V ~C_efsm_rale_) => CEI | CEE) in
let new_C_efsm_srdy_en = ((new_C_efsm_stateA = CEE) V (C_efsm_state = CEE)) in
let cout_sel0 = (ALTER ARBN (0) ((new_C_sfsm_sd1 V new_C_sfsm_sd0) =>
                                    new_C_sfsm_s_cout_sel0 | new_C_mfsm_m_cout_sel0)) in
```

```
let cout_sel1 = (ALTER cout_sel0 (1) ((new_C_sfsm_sd1 V new_C_sfsm_sd0) => F | new_C_mfsm_m_cout_sel1)) in
let c_cout_sel = cout_sel1 in
let c_busy = (~((SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
let c_grant = ((((SUBARRAY Id (1,0)) = (WORDN 0)) Λ ~(ELEMENT CB_rqt_in_ (0)))
           V (((SUBARRAY Id (1,0)) = (WORDN 1)) Λ ~(ELEMENT CB_rqt_in_ (0)) Λ (ELEMENT CB_rqt_in_ (1)))
           V (((SUBARRAY Id (1,0)) = (WORDN 2)) Λ ~(ELEMENT CB_rqt_in_ (0)) Λ (ELEMENT CB_rqt_in_ (1))
                                                                     Λ (ELEMENT CB_rqt_in_ (2)))
           V (((SUBARRAY Id (1,0)) = (WORDN 3)) Λ ~(ELEMENT CB_rqt_in_ (0)) Λ (ELEMENT CB_rqt_in_ (1))
                                          Λ (ELEMENT CB_rqt_in_ (2)) Λ (ELEMENT CB_rqt_in_ (3)))) in
let c_write = ((new_C_mfsm_cm_en) => C_wr | (ELEMENT C_sizewrbe (5))) in
let new_C_clkAA = C_clkA in
let new_C_sidle_delA = C_sidle_del in
let new_C_mrqt_delA = C_mrqt_del in
let c_dfsm_srdy = (CB_ss_in = ^SRDY) in
let c_dfsm_master = (new_C_mfsm_ma3 V new_C_mfsm_ma2 V new_C_mfsm_ma1 V
                 new_C_mfsm_ma0 V new_C_mfsm_md1 V new_C_mfsm_md0) in
let c_dfsm_slave = (~new_C_sfsm_sidle Λ ~new_C_sfsm_slock) in
let c_dfsm_cin_0_le = (ClkD Λ ((new_C_mfsm_md0 Λ c_dfsm_srdy Λ ~c_write) V
                        (new_C_sfsm_sa0) V (new_C_sfsm_sd0 Λ c_write))) in
let c_dfsm_cin_1_le = (ClkD Λ ((new_C_mfsm_md1 Λ c_dfsm_srdy Λ ~c_write) V
                        (new_C_sfsm_sa1) V (new_C_sfsm_sd1 Λ c_write))) in
let c_dfsm_cin_3_le = (ClkD Λ (new_C_sfsm_sidle V new_C_sfsm_slock)) in
let c_dfsm_cin_4_le = (new_C_clkAA Λ new_C_sfsm_sa0) in
let c_dfsm_cout_0_le = ((I_cale_) V (I_srdy_in_ Λ ~c_write)
                         V (new_C_mfsm_ma0 Λ c_dfsm_srdy Λ c_write Λ ClkD)
                         V (new_C_mfsm_md0 Λ c_write Λ c_dfsm_srdy Λ ClkD)) in
let c_dfsm_cout_1_le = (new_C_clkAA Λ new_C_sfsm_sd1) in
let c_dfsm_cad_en = ~((new_C_mfsm_ma3) V (new_C_mfsm_ma1) V (new_C_mfsm_ma0)
                 V (new_C_mfsm_ma2) V (c_write Λ (new_C_mfsm_md1 V new_C_mfsm_md0))
                 V (~c_write Λ (new_C_sfsm_sd1 V new_C_sfsm_sd0))) in
let c_dfsm_i_male_ = ~(new_C_sfsm_sale Λ (~((SUBARRAY C_sizewrbe (1,0)) = (WORDN 3))) Λ new_C_clkAA) in
let c_dfsm_i_rale_ = ~(new_C_sfsm_sale Λ ((SUBARRAY C_sizewrbe (1,0)) = (WORDN 3)) Λ new_C_clkAA) in
let c_dfsm_i_mrdy_ = ~((~c_write Λ ClkD Λ (new_C_sfsm_sale V new_C_sfsm_sd1))
                  V (~c_write Λ new_C_clkAA Λ new_C_sfsm_sack)
                  V (c_write Λ ClkD Λ new_C_sfsm_sd0)) in
let new_C_last_inA_ = I_last_in_ in
let new_C_ssA = CB_ss_in in
let new_C_holdA_ = ((ClkD) => C_hold_ | C_holdA_) in
let new_C_cout_0_le_delA = C_cout_0_le_del in
let new_C_cin_2_leA = C_cin_2_le in
let new_C_mrdy_delA_ = C_mrdy_del_ in
let new_C_iad_en_s_delA = ((ClkD) => C_iad_en_s_del | C_iad_en_s_delA) in
let new_C_wrdyA = C_wrdy in
let new_C_rrdyA = C_rrdy in
let new_C_iad_out = ((new_C_cin_2_leA) => C_data_in | C_iad_out) in
let new_C_a1a0 =
      (((c_dfsm_master Λ new_C_cout_0_le_delA) V (~c_dfsm_master Λ c_dfsm_cout_1_le)) => C_iad_in | C_a1a0) in
let new_C_a3a2 = ((new_C_mfsm_mrequest) => Ccr | C_a3a2) in
let new_C_mfsm_state = C_mfsm_state in
let new_C_mfsm_srdy_en = C_mfsm_srdy_en in
let new_C_mfsm_D = C_mfsm_D in
let new_C_mfsm_grant = C_mfsm_grant in
let new_C_mfsm_rst = C_mfsm_rst in
let new_C_mfsm_busy = C_mfsm_busy in
let new_C_mfsm_write = C_mfsm_write in
let new_C_mfsm_crqt_ = C_mfsm_crqt_ in
let new_C_mfsm_hold_ = C_mfsm_hold_ in
let new_C_mfsm_last_ = C_mfsm_last_ in
let new_C_mfsm_lock_ = C_mfsm_lock_ in
let new_C_mfsm_ss = C_mfsm_ss in
let new_C_mfsm_invalid = C_mfsm_invalid in
let new_C_sfsm_state = C_sfsm_state in
let new_C_sfsm_D = C_sfsm_D in
let new_C_sfsm_grant = C_sfsm_grant in
let new_C_sfsm_rst = C_sfsm_rst in
let new_C_sfsm_write = C_sfsm_write in
let new_C_sfsm_addressed = C_sfsm_addressed in
let new_C_sfsm_hlda_ = C_sfsm_hlda_ in
let new_C_sfsm_ms = C_sfsm_ms in
let new_C_efsm_state = C_efsm_state in
let new_C_efsm_cale_ = C_efsm_cale_ in
let new_C_efsm_last_ = C_efsm_last_ in
let new_C_efsm_male_ = C_efsm_male_ in
let new_C_efsm_rale_ = C_efsm_rale_ in
let new_C_efsm_srdy_ = C_efsm_srdy_ in
let new_C_efsm_rst = C_efsm_rst in
let new_C_wr = C_wr in
let new_C_sizewrbe = C_sizewrbe in
let new_C_clkA = C_clkA in
let new_C_sidle_del = C_sidle_del in
let new_C_mrqt_del = C_mrqt_del in
let new_C_last_in_ = C_last_in_ in
let new_C_lock_in_ = C_lock_in_ in
let new_C_ss = C_ss in
let new_C_last_out_ = C_last_out_ in
let new_C_hold_ = C_hold_ in
let new_C_cout_0_le_del = C_cout_0_le_del in
let new_C_cin_2_le = C_cin_2_le in
let new_C_mrdy_del_ = C_mrdy_del_ in
let new_C_iad_en_s_del = C_iad_en_s_del in
let new_C_wrdy = C_wrdy in
let new_C_rrdy = C_rrdy in
let new_C_parity = C_parity in
let new_C_source = C_source in
let new_C_data_in = C_data_in in
let new_C_iad_in = C_iad_in in

(new_C_mfsm_stateA, new_C_mfsm_mabort, new_C_mfsm_midle, new_C_mfsm_mrequest, new_C_mfsm_ma3,
 new_C_mfsm_ma2, new_C_mfsm_ma1, new_C_mfsm_ma0, new_C_mfsm_md1, new_C_mfsm_md0,
 new_C_mfsm_iad_en_m,
 new_C_mfsm_m_cout_sel1, new_C_mfsm_m_cout_sel0, new_C_mfsm_ms, new_C_mfsm_rqt_, new_C_mfsm_cgnt_,
 new_C_mfsm_cm_en, new_C_mfsm_abort_le_en_, new_C_mfsm_mparity, new_C_sfsm_stateA, new_C_sfsm_ss,
 new_C_sfsm_iad_en_s, new_C_sfsm_sidle, new_C_sfsm_slock, new_C_sfsm_sa1, new_C_sfsm_sa0,
 new_C_sfsm_sale, new_C_sfsm_sd1, new_C_sfsm_sd0, new_C_sfsm_sack, new_C_sfsm_sabort,
 new_C_sfsm_s_cout_sel0, new_C_sfsm_sparity, new_C_efsm_stateA, new_C_efsm_srdy_en, new_C_clkAA,
 new_C_sidle_delA, new_C_mrqt_delA, new_C_last_inA_, new_C_ssA, new_C_holdA_,
 new_C_cout_0_le_delA, new_C_cin_2_leA, new_C_mrdy_delA_, new_C_iad_en_s_delA, new_C_wrdyA, new_C_rrdyA,
 new_C_iad_out, new_C_a1a0, new_C_a3a2, new_C_mfsm_state, new_C_mfsm_srdy_en, new_C_mfsm_D,
    new_C_mfsm_grant, new_C_mfsm_rst, new_C_mfsm_busy, new_C_mfsm_write, new_C_mfsm_crqt_,
    new_C_mfsm_hold_, new_C_mfsm_last_, new_C_mfsm_lock_, new_C_mfsm_ss, new_C_mfsm_invalid,
    new_C_sfsm_state, new_C_sfsm_D, new_C_sfsm_grant, new_C_sfsm_rst, new_C_sfsm_write,
    new_C_sfsm_addressed, new_C_sfsm_hlda_, new_C_sfsm_ms, new_C_efsm_state, new_C_efsm_cale_,
    new_C_efsm_last_, new_C_efsm_male_, new_C_efsm_rale_, new_C_efsm_srdy_, new_C_efsm_rst, new_C_wr,
    new_C_sizewrbe, new_C_clkA, new_C_sidle_del, new_C_mrqt_del, new_C_last_in_, new_C_lock_in_,
    new_C_ss, new_C_last_out_, new_C_hold_, new_C_cout_0_le_del, new_C_cin_2_le, new_C_mrdy_del_,
    new_C_iad_en_s_del, new_C_wrdy, new_C_rrdy, new_C_parity, new_C_source, new_C_data_in, new_C_iad_in)"
  );;

%------------------------------------------------------------------------------------------------------
  Output definition for Phase-A instruction.
------------------------------------------------------------------------------------------------------%

let PH_A_out_def = new_definition
 ('PH_A_out',
    "! (rep:^rep_ty)
       (C_mfsm_stateA C_mfsm_state :cmfsm_ty)
       (C_sfsm_stateA C_sfsm_state :csfsm_ty)
       (C_efsm_stateA C_efsm_state :cefsm_ty)
       (C_mfsm_ms C_sfsm_ss C_ssA C_iad_out C_a1a0 C_a3a2 C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss
        C_source C_data_in C_iad_in :wordn)
       (C_mfsm_mabort C_mfsm_midle C_mfsm_mrequest C_mfsm_ma3 C_mfsm_ma2 C_mfsm_ma1
        C_mfsm_ma0 C_mfsm_md1 C_mfsm_md0 C_mfsm_iad_en_m C_mfsm_m_cout_sel1 C_mfsm_m_cout_sel0
        C_mfsm_rqt_ C_mfsm_cgnt_ C_mfsm_cm_en C_mfsm_abort_le_en_ C_mfsm_mparity
        C_sfsm_iad_en_s C_sfsm_sidle C_sfsm_slock C_sfsm_sa1 C_sfsm_sa0
        C_sfsm_sale C_sfsm_sd1 C_sfsm_sd0 C_sfsm_sack C_sfsm_sabort C_sfsm_s_cout_sel0 C_sfsm_sparity
        C_efsm_srdy_en
        C_clkAA C_sidle_delA C_mrqt_delA C_last_inA_ C_holdA_ C_cout_0_le_delA
        C_cin_2_leA C_mrdy_delA_ C_iad_en_s_delA C_wrdyA C_rrdyA
        C_mfsm_srdy_en C_mfsm_D C_mfsm_grant C_mfsm_rst C_mfsm_busy C_mfsm_write
        C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_last_ C_mfsm_lock_ C_mfsm_invalid
        C_sfsm_D C_sfsm_grant C_sfsm_rst C_sfsm_write C_sfsm_addressed C_sfsm_hlda_
        C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
        C_wr C_clkA C_sidle_del C_mrqt_del C_last_in_ C_lock_in_ C_last_out_
        C_hold_ C_cout_0_le_del C_cin_2_le C_mrdy_del_ C_iad_en_s_del C_wrdy
        C_rrdy C_parity :bool)
       (I_mrdy_in_ I_rale_in_ I_male_in_ I_last_in_ I_srdy_in_ I_lock_ I_cale_ I_hlda_ I_crqt_
        Rst ClkA ClkB ClkD Pmm_failure Piu_invalid Reset_error :bool)
       (I_ad_in I_be_in_ CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID Ccr :wordn)
       (I_cgnt_ I_mrdy_out_ I_hold_ I_rale_out_ I_male_out_ I_last_out_ I_srdy_out_ CB_rqt_out_
        Disable_writes CB_parity :bool) .
  PH_A_out rep
           (C_mfsm_stateA, C_mfsm_mabort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
            C_mfsm_ma1, C_mfsm_ma0, C_mfsm_md1, C_mfsm_md0, C_mfsm_iad_en_m, C_mfsm_m_cout_sel1,
            C_mfsm_m_cout_sel0, C_mfsm_ms, C_mfsm_rqt_, C_mfsm_cgnt_, C_mfsm_cm_en, C_mfsm_abort_le_en_,
            C_mfsm_mparity, C_sfsm_stateA, C_sfsm_ss, C_sfsm_iad_en_s, C_sfsm_sidle, C_sfsm_slock,
            C_sfsm_sa1, C_sfsm_sa0, C_sfsm_sale, C_sfsm_sd1, C_sfsm_sd0, C_sfsm_sack, C_sfsm_sabort,
            C_sfsm_s_cout_sel0, C_sfsm_sparity, C_efsm_stateA, C_efsm_srdy_en, C_clkAA, C_sidle_delA,
            C_mrqt_delA, C_last_inA_, C_ssA, C_holdA_, C_cout_0_le_delA, C_cin_2_leA,
            C_mrdy_delA_, C_iad_en_s_delA, C_wrdyA, C_rrdyA, C_iad_out, C_a1a0, C_a3a2, C_mfsm_state,
            C_mfsm_srdy_en, C_mfsm_D, C_mfsm_grant, C_mfsm_rst, C_mfsm_busy, C_mfsm_write, C_mfsm_crqt_,
            C_mfsm_hold_, C_mfsm_last_, C_mfsm_lock_, C_mfsm_ss, C_mfsm_invalid, C_sfsm_state, C_sfsm_D,
            C_sfsm_grant, C_sfsm_rst, C_sfsm_write, C_sfsm_addressed, C_sfsm_hlda_, C_sfsm_ms,
                C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_,
                C_efsm_rst, C_wr, C_sizewrbe, C_clkA, C_sidle_del, C_mrqt_del, C_last_in_, C_lock_in_,
                C_ss, C_last_out_, C_hold_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_wrdy,
                C_rrdy, C_parity, C_source, C_data_in, C_iad_in)
               (I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_, I_lock_,
                I_cale_, I_hlda_, I_crqt_, CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in, Rst, ClkA, ClkB,
                ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr, Reset_error) =

   let new_C_mfsm_stateA =
         ((C_mfsm_rst) => CMI |
          ((C_mfsm_state = CMI) => (C_mfsm_D ∧ ~C_mfsm_crqt_ ∧ ~C_mfsm_busy ∧ ~C_mfsm_invalid) => CMR | CMI |
          ((C_mfsm_state = CMR) => (C_mfsm_D ∧ C_mfsm_grant ∧ C_mfsm_hold_) => CMA3 | CMR |
          ((C_mfsm_state = CMA3) => ((C_mfsm_D) => CMA1 | CMA3) |
          ((C_mfsm_state = CMA1) =>
                (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA0 |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA1 |
          ((C_mfsm_state = CMA0) =>
                (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA2 |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA0 |
          ((C_mfsm_state = CMA2) =>
                (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD1 |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA2 |
          ((C_mfsm_state = CMD1) =>
                (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD0 |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD1 |
          ((C_mfsm_state = CMD0) =>
                (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ C_mfsm_last_) => CMD1 |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_mfsm_last_) => CMW |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD0 |
          ((C_mfsm_state = CMW) =>
                (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SACK) ∧ C_mfsm_lock_) => CMI |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_mfsm_lock_ ∧ ~C_mfsm_crqt_) => CMA3 | CMW |
          ((~C_mfsm_last_) => CMI | CMABT)))))))))) in

    let new_C_mfsm_mabort = (new_C_mfsm_stateA = CMABT) in
    let new_C_mfsm_midle = (new_C_mfsm_stateA = CMI) in
    let new_C_mfsm_mrequest = (new_C_mfsm_stateA = CMR) in
    let new_C_mfsm_ma3 = (new_C_mfsm_stateA = CMA3) in
    let new_C_mfsm_ma2 = (new_C_mfsm_stateA = CMA2) in
    let new_C_mfsm_ma1 = (new_C_mfsm_stateA = CMA1) in
    let new_C_mfsm_ma0 = (new_C_mfsm_stateA = CMA0) in
    let new_C_mfsm_md1 = (new_C_mfsm_stateA = CMD1) in
    let new_C_mfsm_md0 = (new_C_mfsm_stateA = CMD0) in
    let new_C_mfsm_iad_en_m = (((new_C_mfsm_stateA = CMD1) ∧ ~C_mfsm_write ∧ C_mfsm_srdy_en)
                               ∨ ((new_C_mfsm_stateA = CMD0) ∧ ~C_mfsm_write ∧ C_mfsm_srdy_en)
                               ∨ ((new_C_mfsm_stateA = CMW) ∧ (C_mfsm_state = CMD0) ∧ ~C_mfsm_write ∧ C_mfsm_-
srdy_en)) in
     let new_C_mfsm_m_cout_sel1 = ((new_C_mfsm_stateA = CMA3) ∨ (new_C_mfsm_stateA = CMA2)) in
     let new_C_mfsm_m_cout_sel0 = ((new_C_mfsm_stateA = CMA3) ∨ (new_C_mfsm_stateA = CMA1) ∨ (new_C_mfsm_-
stateA = CMD1)) in
     let ms2 = (ALTER ARBN (2) ((new_C_mfsm_stateA = CMA3) ∨ (new_C_mfsm_stateA = CMA1) ∨
                                (new_C_mfsm_stateA = CMA0) ∨ (new_C_mfsm_stateA = CMA2) ∨
                                (new_C_mfsm_stateA = CMD1) ∨ (new_C_mfsm_stateA = CMD0) ∨
```

```
                              (new_C_mfsm_stateA = CMW) ∨ (new_C_mfsm_stateA = CMABT))) in
let ms1 = (ALTER ms2 (1) ((new_C_mfsm_stateA = CMA1) ∨ (new_C_mfsm_stateA = CMA0) ∨
                          (new_C_mfsm_stateA = CMA2) ∨ (new_C_mfsm_stateA = CMD1) ∨
                          ((new_C_mfsm_stateA = CMD0) ∧ C_mfsm_last_) ∨ (new_C_mfsm_stateA = CMW) ∨
                          (new_C_mfsm_stateA = CMABT))) in
let ms0 = (ALTER ms1 (0) (((new_C_mfsm_stateA = CMD0) ∧ ~C_mfsm_last_) ∨
                          ((new_C_mfsm_stateA = CMW) ∧ C_mfsm_lock_) ∨ (new_C_mfsm_stateA = CMABT))) in
let new_C_mfsm_ms = ms0 in
let new_C_mfsm_rqt_ = ~(~(new_C_mfsm_stateA = CMI)) in
let new_C_mfsm_cgnt_ = ~(new_C_mfsm_stateA = CMA3) in
let new_C_mfsm_cm_en = ((~(new_C_mfsm_stateA = CMI)) ∧ (~(new_C_mfsm_stateA = CMR))) in
let new_C_mfsm_abort_le_en_ = ~((new_C_mfsm_stateA = CMABT) ∨ (new_C_mfsm_stateA = CMI)) in
let new_C_mfsm_mparity = ((new_C_mfsm_stateA = CMA3) ∨ (new_C_mfsm_stateA = CMA1)
                         ∨ (new_C_mfsm_stateA = CMA0) ∨ (new_C_mfsm_stateA = CMA2)
                         ∨ (new_C_mfsm_stateA = CMD1) ∨ (new_C_mfsm_stateA = CMD0)
                         ∨ (C_mfsm_state = CMA1) ∨ (C_mfsm_state = CMA0)
                         ∨ (C_mfsm_state = CMA2) ∨ (C_mfsm_state = CMD1)) in

let new_C_sfsm_stateA =
    ((C_sfsm_rst) => CSI |
     (C_sfsm_state = CSI) => ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~C_sfsm_grant
                                                  ∧ C_sfsm_addressed) => CSA1 | CSI) |
     (C_sfsm_state = CSL) =>
         ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~C_sfsm_grant ∧ C_sfsm_addressed) => CSA1 |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~C_sfsm_grant ∧ ~C_sfsm_addressed) => CSI |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSL) |
     (C_sfsm_state = CSA1) =>
         ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSA0 |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA1) |
     (C_sfsm_state = CSA0) =>
         ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ C_sfsm_hlda_) => CSA0W |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0) |
     (C_sfsm_state = CSA0W) =>
         ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0W) |
     (C_sfsm_state = CSALE) =>
         ((C_sfsm_D ∧ C_sfsm_write ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
          (C_sfsm_D ∧ ~C_sfsm_write ∧ (C_sfsm_ms = ^MRDY)) => CSRR |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSALE) |
     (C_sfsm_state = CSRR) =>
         ((C_sfsm_D ∧ ~(C_sfsm_ms = ^MABORT)) => CSD1 |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSRR) |
     (C_sfsm_state = CSD1) =>
         ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD0 |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD1) |
     (C_sfsm_state = CSD0) =>
         ((C_sfsm_D ∧ (C_sfsm_ms = ^MEND)) => CSACK |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD0) |
     (C_sfsm_state = CSACK) =>
         ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSL |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MWAIT)) => CSI |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSACK) |
                  (C_sfsm_D) => CSI | CSABT) in

let ss2 = (ALTER ARBN (2) ((~(new_C_sfsm_stateA = CSI)) ∧ (~(new_C_sfsm_stateA = CSABT)))) in
let ss1 = (ALTER ss2 (1) ((~(new_C_sfsm_stateA = CSI)) ∧ (~(new_C_sfsm_stateA = CSACK))
                                                    ∧ (~(new_C_sfsm_stateA = CSABT)))) in
let ss0 = (ALTER ss1 (0) ((new_C_sfsm_stateA = CSA0W) ∨
                              ((new_C_sfsm_stateA = CSALE) ∧ ~C_sfsm_write) ∨
                              (new_C_sfsm_stateA = CSACK))) in
let new_C_sfsm_ss = ss0 in
let new_C_sfsm_iad_en_s = (((new_C_sfsm_stateA = CSALE) ∧ (~(C_sfsm_state = CSALE)))
                              ∨ ((new_C_sfsm_stateA = CSALE) ∧ C_sfsm_write)
                              ∨ ((new_C_sfsm_stateA = CSD1) ∧ C_sfsm_write ∧ (~(C_sfsm_state = CSRR)))
                              ∨ ((new_C_sfsm_stateA = CSD0) ∧ C_sfsm_write)
                              ∨ ((new_C_sfsm_stateA = CSACK) ∧ C_sfsm_write)) in
let new_C_sfsm_sidle = (new_C_sfsm_stateA = CSI) in
let new_C_sfsm_slock = (new_C_sfsm_stateA = CSL) in
let new_C_sfsm_sa1 = (new_C_sfsm_stateA = CSA1) in
let new_C_sfsm_sa0 = (new_C_sfsm_stateA = CSA0) in
let new_C_sfsm_sale = (new_C_sfsm_stateA = CSALE) in
let new_C_sfsm_sd1 = (new_C_sfsm_stateA = CSD1) in
let new_C_sfsm_sd0 = (new_C_sfsm_stateA = CSD0) in
let new_C_sfsm_sack = (new_C_sfsm_stateA = CSACK) in
let new_C_sfsm_sabort = (new_C_sfsm_stateA = CSABT) in
let new_C_sfsm_s_cout_sel0 = (new_C_sfsm_stateA = CSD1) in
let new_C_sfsm_sparity = ((~(new_C_sfsm_stateA = CSI)) ∧ (~(new_C_sfsm_stateA = CSACK))
                                                    ∧ (~(new_C_sfsm_stateA = CSABT))) in
let new_C_efsm_stateA =
      ((C_efsm_rst) => CEI |
       (C_efsm_state = CEI) => ((~C_efsm_cale_) => CEE | CEI) |
       ((~C_efsm_last_ ∧ ~C_efsm_srdy_) ∨ ~C_efsm_male_ ∨ ~C_efsm_rale_) => CEI | CEE) in
let new_C_efsm_srdy_en = ((new_C_efsm_stateA = CEE) ∨ (C_efsm_state = CEE)) in
let cout_sel0 = (ALTER ARBN (0) ((new_C_sfsm_sd1 ∨ new_C_sfsm_sd0) =>
                                     new_C_sfsm_s_cout_sel0 | new_C_mfsm_m_cout_sel0)) in
let cout_sel10 = (ALTER cout_sel0 (1) ((new_C_sfsm_sd1 ∨ new_C_sfsm_sd0) => F | new_C_mfsm_m_cout_sel1)) in
let c_cout_sel = cout_sel10 in
let c_busy = (~((SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
let c_grant = ((((SUBARRAY Id (1,0)) = (WORDN 0)) ∧ ~(ELEMENT CB_rqt_in_ (0)))
            ∨ (((SUBARRAY Id (1,0)) = (WORDN 1)) ∧ ~(ELEMENT CB_rqt_in_ (0)) ∧ (ELEMENT CB_rqt_in_ (1)))
            ∨ (((SUBARRAY Id (1,0)) = (WORDN 2)) ∧ ~(ELEMENT CB_rqt_in_ (0)) ∧ (ELEMENT CB_rqt_in_ (1))
                                                                ∧ (ELEMENT CB_rqt_in_ (2)))
             ∨ (((SUBARRAY Id (1,0)) = (WORDN 3)) ∧ ~(ELEMENT CB_rqt_in_ (0)) ∧ (ELEMENT CB_rqt_in_ (1))
                                        ∧ (ELEMENT CB_rqt_in_ (2)) ∧ (ELEMENT CB_rqt_in_ (3)))) in
let c_write = ((new_C_mfsm_cm_en) => C_wr | (ELEMENT C_sizewrbe (5))) in
let new_C_clkAA = C_clkA in
let new_C_sidle_delA = C_sidle_del in
let new_C_mrqt_delA = C_mrqt_del in
let c_dfsm_srdy = (CB_ss_in = ^SRDY) in
let c_dfsm_master = (new_C_mfsm_ma3 ∨ new_C_mfsm_ma2 ∨ new_C_mfsm_ma1 ∨
                        new_C_mfsm_ma0 ∨ new_C_mfsm_md1 ∨ new_C_mfsm_md0) in
let c_dfsm_slave = (~new_C_sfsm_sidle ∧ ~new_C_sfsm_slock) in
let c_dfsm_cin_0_le = (ClkD ∧ ((new_C_mfsm_md0 ∧ c_dfsm_srdy ∧ ~c_write) ∨
                                 (new_C_sfsm_sa0) ∨ (new_C_sfsm_sd0 ∧ c_write))) in
let c_dfsm_cin_1_le = (ClkD ∧ ((new_C_mfsm_md1 ∧ c_dfsm_srdy ∧ ~c_write) ∨
                            (new_C_sfsm_sa1) ∨ (new_C_sfsm_sd1 ∧ c_write))) in
let c_dfsm_cin_3_le = (ClkD ∧ (new_C_sfsm_sidle ∨ new_C_sfsm_slock)) in
let c_dfsm_cin_4_le = (new_C_clkAA ∧ new_C_sfsm_sa0) in
let c_dfsm_cout_0_le = ((I_cale_) ∨ (I_srdy_in_ ∧ ~c_write)
                       ∨ (new_C_mfsm_ma0 ∧ c_dfsm_srdy ∧ c_write ∧ ClkD)
                       ∨ (new_C_mfsm_md0 ∧ c_write ∧ c_dfsm_srdy ∧ ClkD)) in
let c_dfsm_cout_1_le = (new_C_clkAA ∧ new_C_sfsm_sd1) in
let c_dfsm_cad_en = ~((new_C_mfsm_ma3) ∨ (new_C_mfsm_ma1) ∨ (new_C_mfsm_ma0)
                     ∨ (new_C_mfsm_ma2) ∨ (c_write ∧ (new_C_mfsm_md1 ∨ new_C_mfsm_md0))
                     ∨ (~c_write ∧ (new_C_sfsm_sd1 ∨ new_C_sfsm_sd0))) in
let c_dfsm_i_male_ = ~(new_C_sfsm_sale ∧ (~((SUBARRAY C_sizewrbe (1,0)) = (WORDN 3))) ∧ new_C_clkAA) in
let c_dfsm_i_rale_ = ~(new_C_sfsm_sale ∧ ((SUBARRAY C_sizewrbe (1,0)) = (WORDN 3)) ∧ new_C_clkAA) in
let c_dfsm_i_mrdy_ = ~((~c_write ∧ ClkD ∧ (new_C_sfsm_sale ∨ new_C_sfsm_sd1))
                      ∨ (~c_write ∧ new_C_clkAA ∧ new_C_sfsm_sack)
                      ∨ (c_write ∧ ClkD ∧ new_C_sfsm_sd0)) in
let new_C_last_inA_ = I_last_in_ in
let new_C_ssA = CB_ss_in in
let new_C_holdA_ = ((ClkD) => C_hold_ | C_holdA_) in
let new_C_cout_0_le_delA = C_cout_0_le_del in
let new_C_cin_2_leA = C_cin_2_le in
let new_C_mrdy_delA_ = C_mrdy_del_ in
let new_C_iad_en_s_delA = ((ClkD) => C_iad_en_s_del | C_iad_en_s_delA) in
let new_C_wrdyA = C_wrdy in
let new_C_rrdyA = C_rrdy in
let new_C_iad_out = ((new_C_cin_2_leA) => C_data_in | C_iad_out) in
let new_C_a1a0 =
        (((c_dfsm_master ∧ new_C_cout_0_le_delA) ∨ (~c_dfsm_master ∧ c_dfsm_cout_1_le)) => C_iad_in | C_a1a0) in
let new_C_a3a2 = ((new_C_mfsm_mrequest) => Ccr | C_a3a2) in
let new_C_mfsm_state = C_mfsm_state in
let new_C_mfsm_srdy_en = C_mfsm_srdy_en in
let new_C_mfsm_D = C_mfsm_D in
let new_C_mfsm_grant = C_mfsm_grant in
let new_C_mfsm_rst = C_mfsm_rst in
let new_C_mfsm_busy = C_mfsm_busy in
let new_C_mfsm_write = C_mfsm_write in
let new_C_mfsm_crqt_ = C_mfsm_crqt_ in
let new_C_mfsm_hold_ = C_mfsm_hold_ in
let new_C_mfsm_last_ = C_mfsm_last_ in
let new_C_mfsm_lock_ = C_mfsm_lock_ in
let new_C_mfsm_ss = C_mfsm_ss in
let new_C_mfsm_invalid = C_mfsm_invalid in
let new_C_sfsm_state = C_sfsm_state in
let new_C_sfsm_D = C_sfsm_D in
let new_C_sfsm_grant = C_sfsm_grant in
let new_C_sfsm_rst = C_sfsm_rst in
let new_C_sfsm_write = C_sfsm_write in
let new_C_sfsm_addressed = C_sfsm_addressed in
let new_C_sfsm_hlda_ = C_sfsm_hlda_ in
let new_C_sfsm_ms = C_sfsm_ms in
let new_C_efsm_state = C_efsm_state in
let new_C_efsm_cale_ = C_efsm_cale_ in
let new_C_efsm_last_ = C_efsm_last_ in
let new_C_efsm_male_ = C_efsm_male_ in
let new_C_efsm_rale_ = C_efsm_rale_ in
let new_C_efsm_srdy_ = C_efsm_srdy_ in
let new_C_efsm_rst = C_efsm_rst in
let new_C_wr = C_wr in
let new_C_sizewrbe = C_sizewrbe in
let new_C_clkA = C_clkA in
let new_C_sidle_del = C_sidle_del in
let new_C_mrqt_del = C_mrqt_del in
let new_C_last_in_ = C_last_in_ in
let new_C_lock_in_ = C_lock_in_ in
let new_C_ss = C_ss in
let new_C_last_out_ = C_last_out_ in
let new_C_hold_ = C_hold_ in
let new_C_cout_0_le_del = C_cout_0_le_del in
let new_C_cin_2_le = C_cin_2_le in
let new_C_mrdy_del_ = C_mrdy_del_ in
let new_C_iad_en_s_del = C_iad_en_s_del in
let new_C_wrdy = C_wrdy in
let new_C_rrdy = C_rrdy in
let new_C_parity = C_parity in
let new_C_source = C_source in
let new_C_data_in = C_data_in in
let new_C_iad_in = C_iad_in in

let I_cgnt_ = new_C_mfsm_cgnt_ in
let I_mrdy_out_ = ((~I_hlda_) => new_C_mrdy_delA_ | ARB) in
let I_hold_ = new_C_holdA_ in
let I_rale_out_ = ((~I_hlda_) => c_dfsm_i_rale_ | ARB) in
let I_male_out_ = ((~I_hlda_) => c_dfsm_i_male_ | ARB) in
let I_last_out_ = ((~I_hlda_) => new_C_last_out_ | ARB) in
let I_srdy_out_ =
        ((~I_cale_ V new_C_efsm_srdy_en) => ~(new_C_wrdyA V new_C_rrdyA V new_C_mfsm_mabort) | ARB) in
let I_be_out_ = ((~I_hlda_) => (SUBARRAY new_C_sizewrbe (9,6)) | ARBN) in
let I_ad_out =
        ((new_C_iad_en_s_delA V new_C_mfsm_iad_en_m V new_C_sfsm_iad_en_s) => new_C_iad_out | ARBN) in
let CB_rqt_out_ = new_C_mfsm_rqt_ in
let cbms10 = (MALTER ARBN (1,0) (SUBARRAY new_C_mfsm_ms (1,0))) in
let cbms210 = (ALTER cbms10 (2) ((ELEMENT new_C_mfsm_ms (2)) Λ ~Pmm_failure Λ ~Piu_invalid)) in
let CB_ms_out = ((~new_C_mfsm_cm_en) => cbms210 | ARBN) in
let cbss10 = (MALTER ARBN (1,0) (SUBARRAY new_C_sfsm_ss (1,0))) in
let cbss210 = (ALTER cbms10 (2) ((ELEMENT new_C_sfsm_ss (2)) Λ ~Pmm_failure Λ ~Piu_invalid)) in
let CB_ss_out = ((~new_C_sfsm_sidle Λ ~new_C_sfsm_sabort) => cbss210 | ARBN) in
let CB_ad_out = ((c_dfsm_cad_en) =>
                        ((c_cout_sel = (WORDN 0)) => Par_Enc rep ((SUBARRAY new_C_a1a0 (15,0))) |
                        ((c_cout_sel = (WORDN 1)) => Par_Enc rep ((SUBARRAY new_C_a1a0 (31,16))) |
                        ((c_cout_sel = (WORDN 2)) => Par_Enc rep ((SUBARRAY new_C_a3a2 (15,0))) |
                        Par_Enc rep ((SUBARRAY new_C_a3a2 (31,16)))))) |
                        ARBN) in
let C_ss_out = new_C_ss in
let Disable_writes = (c_dfsm_slave Λ ~((ChannelID = (WORDN 0)) Λ (ELEMENT new_C_source (6)))
                                        Λ ~((ChannelID = (WORDN 1)) Λ (ELEMENT new_C_source (7)))
                                        Λ ~((ChannelID = (WORDN 2)) Λ (ELEMENT new_C_source (8)))
                                        Λ ~((ChannelID = (WORDN 3)) Λ (ELEMENT new_C_source (9)))) in
let CB_parity = new_C_parity in
```

```
   (I_cgnt_, I_mrdy_out_, I_hold_, I_rale_out_, I_male_out_, I_last_out_, I_srdy_out_, I_ad_out, I_be_out_,
    CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out, C_ss_out, Disable_writes, CB_parity)"
   );;

%------------------------------------------------------------------------------------------------
   Next-state definition for Phase-B instruction.
------------------------------------------------------------------------------------------------%

let PH_B_inst_def = new_definition
 ('PH_B_inst',
    "! (rep:^rep_ty)
        (C_mfsm_stateA C_mfsm_state :cmfsm_ty)
        (C_sfsm_stateA C_sfsm_state :csfsm_ty)
        (C_efsm_stateA C_efsm_state :cefsm_ty)
        (C_mfsm_ms C_sfsm_ss C_ssA C_iad_out C_a1a0 C_a3a2 C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss
         C_source C_data_in C_iad_in :wordn)
        (C_mfsm_mabort C_mfsm_midle C_mfsm_mrequest C_mfsm_ma3 C_mfsm_ma2 C_mfsm_ma1
         C_mfsm_ma0 C_mfsm_md1 C_mfsm_md0 C_mfsm_iad_en_m C_mfsm_m_cout_sel1 C_mfsm_m_cout_sel0
         C_mfsm_rqt_ C_mfsm_cgnt_ C_mfsm_cm_en C_mfsm_abort_le_en_ C_mfsm_mparity
         C_sfsm_iad_en_s C_sfsm_sidle C_sfsm_slock C_sfsm_sa1 C_sfsm_sa0
         C_sfsm_sale C_sfsm_sd1 C_sfsm_sd0 C_sfsm_sack C_sfsm_sabort C_sfsm_s_cout_sel0 C_sfsm_sparity
         C_efsm_srdy_en
         C_clkAA C_sidle_delA C_mrqt_delA C_last_inA_ C_holdA_ C_cout_0_le_delA
         C_cin_2_leA C_mrdy_delA_ C_iad_en_s_delA C_wrdyA C_rrdyA
         C_mfsm_srdy_en C_mfsm_D C_mfsm_grant C_mfsm_rst C_mfsm_busy C_mfsm_write
         C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_last_ C_mfsm_lock_ C_mfsm_invalid
         C_sfsm_D C_sfsm_grant C_sfsm_rst C_sfsm_write C_sfsm_addressed C_sfsm_hlda_
         C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
         C_wr C_clkA C_sidle_del C_mrqt_del C_last_in_ C_lock_in_ C_last_out_
         C_hold_ C_cout_0_le_del C_cin_2_le C_mrdy_del_ C_iad_en_s_del C_wrdy
         C_rrdy C_parity :bool)
        (I_mrdy_in_ I_rale_in_ I_male_in_ I_last_in_ I_srdy_in_ I_lock_ I_cale_ I_hlda_ I_crqt_
         Rst ClkA ClkB ClkD Pmm_failure Piu_invalid Reset_error :bool)
        (I_ad_in I_be_in_ CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID Ccr :wordn)
        (I_cgnt_ I_mrdy_out_ I_hold_ I_rale_out_ I_male_out_ I_last_out_ I_srdy_out_ CB_rqt_out_
         Disable_writes CB_parity :bool) .
    PH_B_inst rep
             (C_mfsm_stateA, C_mfsm_mabort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
              C_mfsm_ma1, C_mfsm_ma0, C_mfsm_md1, C_mfsm_md0, C_mfsm_iad_en_m, C_mfsm_m_cout_sel1,
              C_mfsm_m_cout_sel0, C_mfsm_ms, C_mfsm_rqt_, C_mfsm_cgnt_, C_mfsm_cm_en, C_mfsm_abort_le_en_,
              C_mfsm_mparity, C_sfsm_stateA, C_sfsm_ss, C_sfsm_iad_en_s, C_sfsm_sidle, C_sfsm_slock,
              C_sfsm_sa1, C_sfsm_sa0, C_sfsm_sale, C_sfsm_sd1, C_sfsm_sd0, C_sfsm_sack, C_sfsm_sabort,
              C_sfsm_s_cout_sel0, C_sfsm_sparity, C_efsm_stateA, C_efsm_srdy_en, C_clkAA, C_sidle_delA,
              C_mrqt_delA, C_last_inA_, C_ssA, C_holdA_, C_cout_0_le_delA, C_cin_2_leA,
              C_mrdy_delA_, C_iad_en_s_delA, C_wrdyA, C_rrdyA, C_iad_out, C_a1a0, C_a3a2, C_mfsm_state,
              C_mfsm_srdy_en, C_mfsm_D, C_mfsm_grant, C_mfsm_rst, C_mfsm_busy, C_mfsm_write, C_mfsm_crqt_,
              C_mfsm_hold_, C_mfsm_last_, C_mfsm_lock_, C_mfsm_ss, C_mfsm_invalid, C_sfsm_state, C_sfsm_D,
              C_sfsm_grant, C_sfsm_rst, C_sfsm_write, C_sfsm_addressed, C_sfsm_hlda_, C_sfsm_ms,
              C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_,
              C_efsm_rst, C_wr, C_sizewrbe, C_clkA, C_sidle_del, C_mrqt_del, C_last_in_, C_lock_in_,
              C_ss, C_last_out_, C_hold_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_wrdy,
              C_rrdy, C_parity, C_source, C_data_in, C_iad_in)
             (I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_, I_lock_,
              I_cale_, I_hlda_, I_crqt_, CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in, Rst, ClkA, ClkB,
            ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr, Reset_error) =

let new_C_wr = ((~I_cale_) => (ELEMENT I_ad_in (27)) | C_wr) in
let new_C_sizewrbe = ((Rst) => ARBN |
                     ((C_sfsm_sa0 ∧ C_clkAA) => (SUBARRAY C_data_in (31,22)) | C_sizewrbe)) in
let c_write = ((C_mfsm_cm_en) => new_C_wr | (ELEMENT new_C_sizewrbe (5))) in
let cout_sel0 = (ALTER ARBN (0) ((C_sfsm_sd1 V C_sfsm_sd0) =>
                                   C_sfsm_s_cout_sel0 | C_mfsm_m_cout_sel0)) in
let cout_sel10 = (ALTER cout_sel0 (1) ((C_sfsm_sd1 V C_sfsm_sd0) => F | C_mfsm_m_cout_sel1)) in
let c_cout_sel = cout_sel10 in
let c_busy = (~((SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
let c_grant = ((((SUBARRAY Id (1,0)) = (WORDN 0)) ∧ ~(ELEMENT CB_rqt_in_ (0)))
        V (((SUBARRAY Id (1,0)) = (WORDN 1)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                              ∧ (ELEMENT CB_rqt_in_ (1)))
        V (((SUBARRAY Id (1,0)) = (WORDN 2)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                              ∧ (ELEMENT CB_rqt_in_ (1))
                                              ∧ (ELEMENT CB_rqt_in_ (2)))
        V (((SUBARRAY Id (1,0)) = (WORDN 3)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                              ∧ (ELEMENT CB_rqt_in_ (1))
                                              ∧ (ELEMENT CB_rqt_in_ (2))
                                              ∧ (ELEMENT CB_rqt_in_ (3)))) in
let c_dfsm_srdy = (CB_ss_in = ^SRDY) in
let c_dfsm_master = (C_mfsm_ma3 V C_mfsm_ma2 V C_mfsm_ma1 V C_mfsm_ma0 V C_mfsm_md1 V C_mfsm_md0) in
let c_dfsm_slave = (~C_sfsm_sidle ∧ ~C_sfsm_slock) in
let c_dfsm_cin_0_le = (ClkD ∧ ((C_mfsm_md0 ∧ c_dfsm_srdy ∧ ~c_write) V (C_sfsm_sa0)
                                                       V (C_sfsm_sd0 ∧ c_write))) in
let c_dfsm_cin_1_le = (ClkD ∧ ((C_mfsm_md1 ∧ c_dfsm_srdy ∧ ~c_write) V (C_sfsm_sa1)
                                                       V (C_sfsm_sd1 ∧ c_write))) in
let c_dfsm_cin_3_le = (ClkD ∧ (C_sfsm_sidle V C_sfsm_slock)) in
let c_dfsm_cin_4_le = (C_clkAA ∧ C_sfsm_sa0) in
let c_dfsm_cout_0_le = ((I_cale_) V (I_srdy_in_ ∧ ~c_write)
                        V (C_mfsm_ma0 ∧ c_dfsm_srdy ∧ c_write ∧ ClkD)
                        V (C_mfsm_md0 ∧ c_write ∧ c_dfsm_srdy ∧ ClkD)) in
let c_dfsm_cout_1_le = (C_clkAA ∧ C_sfsm_sd1) in
let c_dfsm_cad_en = ~((C_mfsm_ma3) V (C_mfsm_ma1) V (C_mfsm_ma0) V (C_mfsm_ma2) V
                   (c_write ∧ (C_mfsm_md1 V C_mfsm_md0)) V (~c_write ∧ (C_sfsm_sd1 V C_sfsm_sd0))) in
let c_dfsm_i_male_ = ~(C_sfsm_sale ∧ (~((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3))) ∧ C_clkAA) in
let c_dfsm_i_rale_ = ~(C_sfsm_sale ∧ ((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3)) ∧ C_clkAA) in
let c_dfsm_i_mrdy_ = ~((~c_write ∧ ClkD ∧ (C_sfsm_sale V C_sfsm_sd1)) V
                   (~c_write ∧ C_clkAA ∧ C_sfsm_sack) V (c_write ∧ ClkD ∧ C_sfsm_sd0)) in
let new_C_clkA = ClkD in
let new_C_sidle_del = C_sfsm_sidle in
let new_C_mrqt_del = C_mfsm_mrequest in
let new_C_last_in_ = ((Rst) => F |
                     ((C_mfsm_mabort V C_mfsm_md1 ∧ ClkD) => C_last_inA_ | C_last_in_)) in
let new_C_lock_in_ = ((Rst) => F | ((C_mfsm_ma1) => I_lock_ | C_lock_in_)) in
let new_C_ss = ((C_mfsm_abort_le_en_) => C_ssA | C_ss) in
let mend = (CB_ms_in = ^MEND) in
let mabort = (CB_ms_in = ^MABORT) in
let new_C_last_out_ =
      ((C_sfsm_sa1 ∧ ~(ClkD ∧ (mend V mabort))) => T |
       ((~C_sfsm_sa1 ∧ (ClkD ∧ (mend V mabort))) => F |
       ((~C_sfsm_sa1 ∧ ~(ClkD ∧ (mend V mabort))) => C_last_out_ | ARB))) in
let new_C_hold_ = C_sfsm_sidle in
let new_C_cout_0_le_del = c_dfsm_cout_0_le in
let new_C_cin_2_le = c_dfsm_cin_0_le in
let new_C_mrdy_del_ = c_dfsm_i_mrdy_ in
let new_C_iad_en_s_del = C_sfsm_iad_en_s in
let new_C_wrdy = (c_dfsm_srdy ∧ c_write ∧ C_mfsm_md1 ∧ ClkD) in
let new_C_rrdy = (c_dfsm_srdy ∧ ~c_write ∧ C_mfsm_md0 ∧ ClkD) in
let c_pe = (Par_Det rep CB_ad_in) in
let c_pe_cnt = (ClkD ∧ ((~(C_mfsm_mparity = C_sfsm_sparity)) ∨ ((SUBARRAY CB_ss_in (1,0)) = (WORDN 0)))) in
let new_C_parity =
        (((ClkD ∧ c_pe ∧ c_pe_cnt) ∧ I_cale_) => T |
         ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ ~I_cale_) => F |
         ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ I_cale_) => C_parity | ARB))) in
let new_C_source = ((Rst) => (WORDN 0) |
                            ((c_dfsm_cin_3_le) => Par_Dec rep (CB_ad_in) | C_source)) in
let data_in31_16 = (MALTER ARBN (31,16) ((Rst) => (WORDN 0) |
                                                   ((c_dfsm_cin_1_le) => Par_Dec rep (CB_ad_in) |
                                                   (SUBARRAY C_data_in (31,16))))) in
let data_in31_0 = (MALTER data_in31_16 (15,0) ((Rst) => (WORDN 0) |
                                                     ((c_dfsm_cin_0_le) => Par_Dec rep (CB_ad_in) |
                                                     (SUBARRAY C_data_in (15,0))))) in
let new_C_data_in = data_in31_0 in
let new_C_iad_in = ((c_dfsm_cout_0_le) => I_ad_in | C_iad_in) in
let new_C_mfsm_state = C_mfsm_stateA in
let new_C_mfsm_srdy_en = C_efsm_srdy_en in
let new_C_mfsm_D = ClkD in
let new_C_mfsm_grant = c_grant in
let new_C_mfsm_rst = Rst in
let new_C_mfsm_busy = c_busy in
let new_C_mfsm_write = c_write in
let new_C_mfsm_crqt_ = I_crqt_ in
let new_C_mfsm_hold_ = C_holdA_ in
let new_C_mfsm_last_ = new_C_last_in_ in
let new_C_mfsm_lock_ = new_C_lock_in_ in
let new_C_mfsm_ss = CB_ss_in in
let new_C_mfsm_invalid = Piu_invalid in
let new_C_sfsm_state = C_sfsm_state in
let new_C_sfsm_D = ClkD in
let new_C_sfsm_grant = c_grant in
let new_C_sfsm_rst = Rst in
let new_C_sfsm_write = c_write in
let new_C_sfsm_addressed = (Id = (SUBARRAY new_C_source (15,10))) in
let new_C_sfsm_hlda_ = I_hlda_ in
let new_C_sfsm_ms = CB_ms_in in
let new_C_efsm_state = C_efsm_state in
let new_C_efsm_cale_ = I_cale_ in
let new_C_efsm_last_ = I_last_in_ in
let new_C_efsm_male_ = I_male_in_ in
let new_C_efsm_rale_ = I_rale_in_ in
let new_C_efsm_srdy_ = I_srdy_in_ in
let new_C_efsm_rst = Rst in
let new_C_mfsm_stateA = C_mfsm_stateA in
let new_C_mfsm_mabort = C_mfsm_mabort in
let new_C_mfsm_midle = C_mfsm_midle in
let new_C_mfsm_mrequest = C_mfsm_mrequest in
let new_C_mfsm_ma3 = C_mfsm_ma3 in
let new_C_mfsm_ma2 = C_mfsm_ma2 in
let new_C_mfsm_ma1 = C_mfsm_ma1 in
let new_C_mfsm_ma0 = C_mfsm_ma0 in
let new_C_mfsm_md1 = C_mfsm_md1 in
let new_C_mfsm_md0 = C_mfsm_md0 in
let new_C_mfsm_iad_en_m = C_mfsm_iad_en_m in
let new_C_mfsm_m_cout_sel1 = C_mfsm_m_cout_sel1 in
let new_C_mfsm_m_cout_sel0 = C_mfsm_m_cout_sel0 in
let new_C_mfsm_ms = C_mfsm_ms in
let new_C_mfsm_rqt_ = C_mfsm_rqt_ in
let new_C_mfsm_cgnt_ = C_mfsm_cgnt_ in
let new_C_mfsm_cm_en = C_mfsm_cm_en in
let new_C_mfsm_abort_le_en_ = C_mfsm_abort_le_en_ in
let new_C_mfsm_mparity = C_mfsm_mparity in
let new_C_sfsm_stateA = C_sfsm_stateA in
let new_C_sfsm_ss = C_sfsm_ss in
let new_C_sfsm_iad_en_s = C_sfsm_iad_en_s in
let new_C_sfsm_sidle = C_sfsm_sidle in
let new_C_sfsm_slock = C_sfsm_slock in
let new_C_sfsm_sa1 = C_sfsm_sa1 in
let new_C_sfsm_sa0 = C_sfsm_sa0 in
let new_C_sfsm_sale = C_sfsm_sale in
let new_C_sfsm_sd1 = C_sfsm_sd1 in
let new_C_sfsm_sd0 = C_sfsm_sd0 in
let new_C_sfsm_sack = C_sfsm_sack in
let new_C_sfsm_sabort = C_sfsm_sabort in
let new_C_sfsm_s_cout_sel0 = C_sfsm_s_cout_sel0 in
let new_C_sfsm_sparity = C_sfsm_sparity in
let new_C_efsm_stateA = C_efsm_stateA in
let new_C_efsm_srdy_en = C_efsm_srdy_en in
let new_C_clkAA = C_clkAA in
let new_C_sidle_delA = C_sidle_delA in
let new_C_mrqt_delA = C_mrqt_delA in
let new_C_last_inA_ = C_last_inA_ in
let new_C_ssA = C_ssA in
let new_C_holdA_ = C_holdA_ in
let new_C_cout_0_le_delA = C_cout_0_le_delA in
let new_C_cin_2_leA = C_cin_2_leA in
let new_C_mrdy_delA_ = C_mrdy_delA_ in
let new_C_iad_en_s_delA = C_iad_en_s_delA in
let new_C_wrdyA = C_wrdyA in
let new_C_rrdyA = C_rrdyA in
let new_C_iad_out = C_iad_out in
let new_C_a1a0 = C_a1a0 in
let new_C_a3a2 = C_a3a2 in

(new_C_mfsm_stateA, new_C_mfsm_mabort, new_C_mfsm_midle, new_C_mfsm_mrequest, new_C_mfsm_ma3,
 new_C_mfsm_ma2, new_C_mfsm_ma1, new_C_mfsm_ma0, new_C_mfsm_md1, new_C_mfsm_md0,
 new_C_mfsm_iad_en_m,
 new_C_mfsm_m_cout_sel1, new_C_mfsm_m_cout_sel0, new_C_mfsm_ms, new_C_mfsm_rqt_, new_C_mfsm_cgnt_,
 new_C_mfsm_cm_en, new_C_mfsm_abort_le_en_, new_C_mfsm_mparity, new_C_sfsm_stateA, new_C_sfsm_ss,
 new_C_sfsm_iad_en_s, new_C_sfsm_sidle, new_C_sfsm_slock, new_C_sfsm_sa1, new_C_sfsm_sa0,
 new_C_sfsm_sale, new_C_sfsm_sd1, new_C_sfsm_sd0, new_C_sfsm_sack, new_C_sfsm_sabort,
    new_C_sfsm_s_cout_sel0, new_C_sfsm_sparity, new_C_efsm_stateA, new_C_efsm_srdy_en, new_C_clkAA,
    new_C_sidle_delA, new_C_mrqt_delA, new_C_last_inA_, new_C_ssA, new_C_holdA_,
    new_C_cout_0_le_delA, new_C_cin_2_leA, new_C_mrdy_delA_, new_C_iad_en_s_delA, new_C_wrdyA, new_C_rrdyA,
    new_C_iad_out, new_C_a1a0, new_C_a3a2, new_C_mfsm_state, new_C_mfsm_srdy_en, new_C_mfsm_D,
    new_C_mfsm_grant, new_C_mfsm_rst, new_C_mfsm_busy, new_C_mfsm_write, new_C_mfsm_crqt_,
    new_C_mfsm_hold_, new_C_mfsm_last_, new_C_mfsm_lock_, new_C_mfsm_ss, new_C_mfsm_invalid,
    new_C_sfsm_state, new_C_sfsm_D, new_C_sfsm_grant, new_C_sfsm_rst, new_C_sfsm_write,
    new_C_sfsm_addressed, new_C_sfsm_hlda_, new_C_sfsm_ms, new_C_efsm_state, new_C_efsm_cale_,
    new_C_efsm_last_, new_C_efsm_male_, new_C_efsm_rale_, new_C_efsm_srdy_, new_C_efsm_rst, new_C_wr,
    new_C_sizewrbe, new_C_clkA, new_C_sidle_del, new_C_mrqt_del, new_C_last_in_, new_C_lock_in_,
    new_C_ss, new_C_last_out_, new_C_hold_, new_C_cout_0_le_del, new_C_cin_2_le, new_C_mrdy_del_,
    new_C_iad_en_s_del, new_C_wrdy, new_C_rrdy, new_C_parity, new_C_source, new_C_data_in, new_C_iad_in)"
  );;

%-------------------------------------------------------------------------------------------------
   Output definition for Phase-B instruction.
-------------------------------------------------------------------------------------------------%

let PH_B_out_def = new_definition
 ('PH_B_out',
   "! (rep:^rep_ty)
      (C_mfsm_stateA C_mfsm_state :cmfsm_ty)
      (C_sfsm_stateA C_sfsm_state :csfsm_ty)
      (C_efsm_stateA C_efsm_state :cefsm_ty)
      (C_mfsm_ms C_sfsm_ss C_ssA C_iad_out C_a1a0 C_a3a2 C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss
       C_source C_data_in C_iad_in :wordn)
      (C_mfsm_mabort C_mfsm_midle C_mfsm_mrequest C_mfsm_ma3 C_mfsm_ma2 C_mfsm_ma1
       C_mfsm_ma0 C_mfsm_md1 C_mfsm_md0 C_mfsm_iad_en_m C_mfsm_m_cout_sel1 C_mfsm_m_cout_sel0
       C_mfsm_rqt_ C_mfsm_cgnt_ C_mfsm_cm_en C_mfsm_abort_le_en_ C_mfsm_mparity
       C_sfsm_iad_en_s C_sfsm_sidle C_sfsm_slock C_sfsm_sa1 C_sfsm_sa0
       C_sfsm_sale C_sfsm_sd1 C_sfsm_sd0 C_sfsm_sack C_sfsm_sabort C_sfsm_s_cout_sel0 C_sfsm_sparity
       C_efsm_srdy_en
       C_clkAA C_sidle_delA C_mrqt_delA C_last_inA_ C_holdA_ C_cout_0_le_delA
       C_cin_2_leA C_mrdy_delA_ C_iad_en_s_delA C_wrdyA C_rrdyA
       C_mfsm_srdy_en C_mfsm_D C_mfsm_grant C_mfsm_rst C_mfsm_busy C_mfsm_write
       C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_last_ C_mfsm_lock_ C_mfsm_invalid
       C_sfsm_D C_sfsm_grant C_sfsm_rst C_sfsm_write C_sfsm_addressed C_sfsm_hlda_
       C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
       C_wr C_clkA C_sidle_del C_mrqt_del C_last_in_ C_lock_in_ C_last_out_
       C_hold_ C_cout_0_le_del C_cin_2_le C_mrdy_del_ C_iad_en_s_del C_wrdy
       C_rrdy C_parity :bool)
      (I_mrdy_in_ I_rale_in_ I_male_in_ I_last_in_ I_srdy_in_ I_lock_ I_cale_ I_hlda_ I_crqt_
       Rst ClkA ClkB ClkD Pmm_failure Piu_invalid Reset_error :bool)
      (I_ad_in I_be_in_ CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID Ccr :wordn)
      (I_cgnt_ I_mrdy_out_ I_hold_ I_rale_out_ I_male_out_ I_last_out_ I_srdy_out_ CB_rqt_out_
       Disable_writes CB_parity :bool) .
   PH_B_out rep
           (C_mfsm_stateA, C_mfsm_mabort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
            C_mfsm_ma1, C_mfsm_ma0, C_mfsm_md1, C_mfsm_md0, C_mfsm_iad_en_m, C_mfsm_m_cout_sel1,
            C_mfsm_m_cout_sel0, C_mfsm_ms, C_mfsm_rqt_, C_mfsm_cgnt_, C_mfsm_cm_en, C_mfsm_abort_le_en_,
            C_mfsm_mparity, C_sfsm_stateA, C_sfsm_ss, C_sfsm_iad_en_s, C_sfsm_sidle, C_sfsm_slock,
            C_sfsm_sa1, C_sfsm_sa0, C_sfsm_sale, C_sfsm_sd1, C_sfsm_sd0, C_sfsm_sack, C_sfsm_sabort,
            C_sfsm_s_cout_sel0, C_sfsm_sparity, C_efsm_stateA, C_efsm_srdy_en, C_clkAA, C_sidle_delA,
            C_mrqt_delA, C_last_inA_, C_ssA, C_holdA_, C_cout_0_le_delA, C_cin_2_leA,
            C_mrdy_delA_, C_iad_en_s_delA, C_wrdyA, C_rrdyA, C_iad_out, C_a1a0, C_a3a2, C_mfsm_state,
            C_mfsm_srdy_en, C_mfsm_D, C_mfsm_grant, C_mfsm_rst, C_mfsm_busy, C_mfsm_write, C_mfsm_crqt_,
            C_mfsm_hold_, C_mfsm_last_, C_mfsm_lock_, C_mfsm_ss, C_mfsm_invalid, C_sfsm_state, C_sfsm_D,
            C_sfsm_grant, C_sfsm_rst, C_sfsm_write, C_sfsm_addressed, C_sfsm_hlda_, C_sfsm_ms,
            C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_,
            C_efsm_rst, C_wr, C_sizewrbe, C_clkA, C_sidle_del, C_mrqt_del, C_last_in_, C_lock_in_,
            C_ss, C_last_out_, C_hold_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_wrdy,
            C_rrdy, C_parity, C_source, C_data_in, C_iad_in)
           (I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_, I_lock_,
            I_cale_, I_hlda_, I_crqt_, CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in, Rst, ClkA, ClkB,
            ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr, Reset_error) =

let new_C_wr = ((~I_cale_) => (ELEMENT I_ad_in (27)) | C_wr) in
let new_C_sizewrbe = ((Rst) => ARBN |
                          ((C_sfsm_sa0 ∧ C_clkAA) => (SUBARRAY C_data_in (31,22)) | C_sizewrbe)) in
let c_write = ((C_mfsm_cm_en) => new_C_wr | (ELEMENT new_C_sizewrbe (5))) in
let cout_sel0 = (ALTER ARBN (0) ((C_sfsm_sd1 ∨ C_sfsm_sd0) =>
                                          C_sfsm_s_cout_sel0 | C_mfsm_m_cout_sel0)) in
let cout_sel10 = (ALTER cout_sel0 (1) ((C_sfsm_sd1 ∨ C_sfsm_sd0) => F | C_mfsm_m_cout_sel1)) in
let c_cout_sel = cout_sel10 in
let c_busy = (~((SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
let c_grant = ((((SUBARRAY Id (1,0)) = (WORDN 0)) ∧ ~(ELEMENT CB_rqt_in_ (0)))
           ∨ (((SUBARRAY Id (1,0)) = (WORDN 1)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                  ∧ (ELEMENT CB_rqt_in_ (1)))
           ∨ (((SUBARRAY Id (1,0)) = (WORDN 2)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                  ∧ (ELEMENT CB_rqt_in_ (1))
                                                  ∧ (ELEMENT CB_rqt_in_ (2)))
           ∨ (((SUBARRAY Id (1,0)) = (WORDN 3)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                  ∧ (ELEMENT CB_rqt_in_ (1))
                                                  ∧ (ELEMENT CB_rqt_in_ (2))
                                                  ∧ (ELEMENT CB_rqt_in_ (3)))) in
let c_dfsm_srdy = (CB_ss_in = ^SRDY) in
let c_dfsm_master = (C_mfsm_ma3 ∨ C_mfsm_ma2 ∨ C_mfsm_ma1 ∨ C_mfsm_ma0 ∨ C_mfsm_md1 ∨ C_mfsm_md0) in
let c_dfsm_slave = (~C_sfsm_sidle ∧ ~C_sfsm_slock) in
let c_dfsm_cin_0_le = (ClkD ∧ ((C_mfsm_md0 ∧ c_dfsm_srdy ∧ ~c_write) ∨ (C_sfsm_sa0)
                                                             ∨ (C_sfsm_sd0 ∧ c_write))) in
let c_dfsm_cin_1_le = (ClkD ∧ ((C_mfsm_md1 ∧ c_dfsm_srdy ∧ ~c_write) ∨ (C_sfsm_sa1)
                                                             ∨ (C_sfsm_sd1 ∧ c_write))) in
let c_dfsm_cin_3_le = (ClkD ∧ (C_sfsm_sidle ∨ C_sfsm_slock)) in
let c_dfsm_cin_4_le = (C_clkAA ∧ C_sfsm_sa0) in
let c_dfsm_cout_0_le = ((I_cale_) ∨ (I_srdy_in_ ∧ ~c_write)
                         ∨ (C_mfsm_ma0 ∧ c_dfsm_srdy ∧ c_write ∧ ClkD)

                         ∨ (C_mfsm_md0 ∧ c_write ∧ c_dfsm_srdy ∧ ClkD)) in
let c_dfsm_cout_1_le = (C_clkAA ∧ C_sfsm_sd1) in
let c_dfsm_cad_en = ~((C_mfsm_ma3) ∨ (C_mfsm_ma1) ∨ (C_mfsm_ma0) ∨ (C_mfsm_ma2) ∨
                       (c_write ∧ (C_mfsm_md1 ∨ C_mfsm_md0)) ∨ (~c_write ∧ (C_sfsm_sd1 ∨ C_sfsm_sd0))) in
let c_dfsm_i_male_ = ~(C_sfsm_sale ∧ (~((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3))) ∧ C_clkAA) in
let c_dfsm_i_rale_ = ~(C_sfsm_sale ∧ ((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3)) ∧ C_clkAA) in
let c_dfsm_i_mrdy_ = ~((~c_write ∧ ClkD ∧ (C_sfsm_sale ∨ C_sfsm_sd1)) ∨
                       (~c_write ∧ C_clkAA ∧ C_sfsm_sack) ∨ (c_write ∧ ClkD ∧ C_sfsm_sd0)) in
let new_C_clkA = ClkD in
let new_C_sidle_del = C_sfsm_sidle in
let new_C_mrqt_del = C_mfsm_mrequest in
let new_C_last_in_ = ((Rst) => F |
                              ((C_mfsm_mabort V C_mfsm_md1 Λ ClkD) => C_last_inA_ | C_last_in_)) in
let new_C_lock_in_ = ((Rst) => F | ((C_mfsm_ma1) => I_lock_ | C_lock_in_)) in
let new_C_ss = ((C_mfsm_abort_le_en_) => C_ssA | C_ss) in
let mend = (CB_ms_in = ^MEND) in
let mabort = (CB_ms_in = ^MABORT) in
let new_C_last_out_ =
        ((C_sfsm_sa1 Λ ~(ClkD Λ (mend V mabort))) => T |
         ((~C_sfsm_sa1 Λ (ClkD Λ (mend V mabort))) => F |
         ((~C_sfsm_sa1 Λ ~(ClkD Λ (mend V mabort))) => C_last_out_ | ARB))) in
let new_C_hold_ = C_sfsm_sidle in
let new_C_cout_0_le_del = c_dfsm_cout_0_le in
let new_C_cin_2_le = c_dfsm_cin_0_le in
let new_C_mrdy_del_ = c_dfsm_i_mrdy_ in
let new_C_iad_en_s_del = C_sfsm_iad_en_s in
let new_C_wrdy = (c_dfsm_srdy Λ c_write Λ C_mfsm_md1 Λ ClkD) in
let new_C_rrdy = (c_dfsm_srdy Λ ~c_write Λ C_mfsm_md0 Λ ClkD) in
let c_pe = (Par_Det rep CB_ad_in) in
let c_pe_cnt = (ClkD Λ ((~(C_mfsm_mparity = C_sfsm_sparity)) V ((SUBARRAY CB_ss_in (1,0)) = (WORDN 0)))) in
let new_C_parity =
        (((ClkD Λ c_pe Λ c_pe_cnt) Λ I_cale_) => T |
         ((~(ClkD Λ c_pe Λ c_pe_cnt) Λ ~I_cale_) => F |
         ((~(ClkD Λ c_pe Λ c_pe_cnt) Λ I_cale_) => C_parity | ARB))) in
let new_C_source = ((Rst) => (WORDN 0) |
                             ((c_dfsm_cin_3_le) => Par_Dec rep (CB_ad_in) | C_source)) in
let data_in31_16 = (MALTER ARBN (31,16) ((Rst) => (WORDN 0) |
                                                   ((c_dfsm_cin_1_le) => Par_Dec rep (CB_ad_in) |
                                                   (SUBARRAY C_data_in (31,16))))) in
let data_in31_0 = (MALTER data_in31_16 (15,0) ((Rst) => (WORDN 0) |
                                                        ((c_dfsm_cin_0_le) => Par_Dec rep (CB_ad_in) |
                                                        (SUBARRAY C_data_in (15,0))))) in
let new_C_data_in = data_in31_0 in
let new_C_iad_in = ((c_dfsm_cout_0_le) => I_ad_in | C_iad_in) in
let new_C_mfsm_state = C_mfsm_stateA in
let new_C_mfsm_srdy_en = C_efsm_srdy_en in
let new_C_mfsm_D = ClkD in
let new_C_mfsm_grant = c_grant in
let new_C_mfsm_rst = Rst in
let new_C_mfsm_busy = c_busy in
let new_C_mfsm_write = c_write in
let new_C_mfsm_crqt_ = I_crqt_ in
let new_C_mfsm_hold_ = C_holdA_ in
let new_C_mfsm_last_ = new_C_last_in_ in
let new_C_mfsm_lock_ = new_C_lock_in_ in
let new_C_mfsm_ss = CB_ss_in in
let new_C_mfsm_invalid = Piu_invalid in
let new_C_sfsm_state = C_sfsm_state in
let new_C_sfsm_D = ClkD in
let new_C_sfsm_grant = c_grant in
let new_C_sfsm_rst = Rst in
let new_C_sfsm_write = c_write in
let new_C_sfsm_addressed = (Id = (SUBARRAY new_C_source (15,10))) in
let new_C_sfsm_hlda_ = I_hlda_ in
let new_C_sfsm_ms = CB_ms_in in
let new_C_efsm_state = C_efsm_state in
let new_C_efsm_cale_ = I_cale_ in
let new_C_efsm_last_ = I_last_in_ in
let new_C_efsm_male_ = I_male_in_ in
let new_C_efsm_rale_ = I_rale_in_ in
let new_C_efsm_srdy_ = I_srdy_in_ in
let new_C_efsm_rst = Rst in
let new_C_mfsm_stateA = C_mfsm_stateA in
let new_C_mfsm_mabort = C_mfsm_mabort in
let new_C_mfsm_midle = C_mfsm_midle in
let new_C_mfsm_mrequest = C_mfsm_mrequest in
let new_C_mfsm_ma3 = C_mfsm_ma3 in
let new_C_mfsm_ma2 = C_mfsm_ma2 in
let new_C_mfsm_ma1 = C_mfsm_ma1 in
let new_C_mfsm_ma0 = C_mfsm_ma0 in
let new_C_mfsm_md1 = C_mfsm_md1 in
let new_C_mfsm_md0 = C_mfsm_md0 in
let new_C_mfsm_iad_en_m = C_mfsm_iad_en_m in
let new_C_mfsm_m_cout_sel1 = C_mfsm_m_cout_sel1 in
let new_C_mfsm_m_cout_sel0 = C_mfsm_m_cout_sel0 in
let new_C_mfsm_ms = C_mfsm_ms in
let new_C_mfsm_rqt_ = C_mfsm_rqt_ in
let new_C_mfsm_cgnt_ = C_mfsm_cgnt_ in
let new_C_mfsm_cm_en = C_mfsm_cm_en in
let new_C_mfsm_abort_le_en_ = C_mfsm_abort_le_en_ in
let new_C_mfsm_mparity = C_mfsm_mparity in
let new_C_sfsm_stateA = C_sfsm_stateA in
let new_C_sfsm_ss = C_sfsm_ss in
let new_C_sfsm_iad_en_s = C_sfsm_iad_en_s in
let new_C_sfsm_sidle = C_sfsm_sidle in
let new_C_sfsm_slock = C_sfsm_slock in
let new_C_sfsm_sa1 = C_sfsm_sa1 in
let new_C_sfsm_sa0 = C_sfsm_sa0 in
let new_C_sfsm_sale = C_sfsm_sale in
let new_C_sfsm_sd1 = C_sfsm_sd1 in
let new_C_sfsm_sd0 = C_sfsm_sd0 in
let new_C_sfsm_sack = C_sfsm_sack in
let new_C_sfsm_sabort = C_sfsm_sabort in
let new_C_sfsm_s_cout_sel0 = C_sfsm_s_cout_sel0 in
let new_C_sfsm_sparity = C_sfsm_sparity in
let new_C_efsm_stateA = C_efsm_stateA in
let new_C_efsm_srdy_en = C_efsm_srdy_en in
let new_C_clkAA = C_clkAA in
let new_C_sidle_delA = C_sidle_delA in
let new_C_mrqt_delA = C_mrqt_delA in
let new_C_last_inA_ = C_last_inA_ in
let new_C_ssA = C_ssA in
let new_C_holdA_ = C_holdA_ in
let new_C_cout_0_le_delA = C_cout_0_le_delA in
let new_C_cin_2_leA = C_cin_2_leA in
let new_C_mrdy_delA_ = C_mrdy_delA_ in
let new_C_iad_en_s_delA = C_iad_en_s_delA in
let new_C_wrdyA = C_wrdyA in
let new_C_rrdyA = C_rrdyA in
```

```
    let new_C_iad_out = C_iad_out in
    let new_C_a1a0 = C_a1a0 in
    let new_C_a3a2 = C_a3a2 in

    let I_cgnt_ = new_C_mfsm_cgnt_ in
    let I_mrdy_out_ = ((~I_hlda_) => new_C_mrdy_delA_ | ARB) in
    let I_hold_ = new_C_holdA_ in
    let I_rale_out_ = ((~I_hlda_) => c_dfsm_i_rale_ | ARB) in
    let I_male_out_ = ((~I_hlda_) => c_dfsm_i_male_ | ARB) in
    let I_last_out_ = ((~I_hlda_) => new_C_last_out_ | ARB) in
    let I_srdy_out_ =
          ((~I_cale_ V new_C_efsm_srdy_en) => ~(new_C_wrdyA V new_C_rrdyA V new_C_mfsm_mabort) | ARB) in
    let I_be_out_ = ((~I_hlda_) => (SUBARRAY new_C_sizewrbe (9,6)) | ARBN) in
    let I_ad_out =
          ((new_C_iad_en_s_delA V new_C_mfsm_iad_en_m V new_C_sfsm_iad_en_s) => new_C_iad_out | ARBN) in
    let CB_rqt_out_ = new_C_mfsm_rqt_ in
    let cbms10 = (MALTER ARBN (1,0) (SUBARRAY new_C_mfsm_ms (1,0))) in
    let cbms210 = (ALTER cbms10 (2) ((ELEMENT new_C_mfsm_ms (2)) Λ ~Pmm_failure Λ ~Piu_invalid)) in
    let CB_ms_out = ((~new_C_mfsm_cm_en) => cbms210 | ARBN) in
    let cbss10 = (MALTER ARBN (1,0) (SUBARRAY new_C_sfsm_ss (1,0))) in
    let cbss210 = (ALTER cbms10 (2) ((ELEMENT new_C_sfsm_ss (2)) Λ ~Pmm_failure Λ ~Piu_invalid)) in
    let CB_ss_out = ((~new_C_sfsm_sidle Λ ~new_C_sfsm_sabort) => cbss210 | ARBN) in
    let CB_ad_out = ((c_dfsm_cad_en) =>
                    ((c_cout_sel = (WORDN 0)) => Par_Enc rep ((SUBARRAY new_C_a1a0 (15,0))) |
                    ((c_cout_sel = (WORDN 1)) => Par_Enc rep ((SUBARRAY new_C_a1a0 (31,16))) |
                    ((c_cout_sel = (WORDN 2)) => Par_Enc rep ((SUBARRAY new_C_a3a2 (15,0))) |
                    Par_Enc rep ((SUBARRAY new_C_a3a2 (31,16)))))) |
                   ARBN) in
    let C_ss_out = new_C_ss in
    let Disable_writes = (c_dfsm_slave Λ ~((ChannelID = (WORDN 0)) Λ (ELEMENT new_C_source (6)))
                                       Λ ~((ChannelID = (WORDN 1)) Λ (ELEMENT new_C_source (7)))
                                       Λ ~((ChannelID = (WORDN 2)) Λ (ELEMENT new_C_source (8)))
                                       Λ ~((ChannelID = (WORDN 3)) Λ (ELEMENT new_C_source (9)))) in
    let CB_parity = new_C_parity in

    (I_cgnt_, I_mrdy_out_, I_hold_, I_rale_out_, I_male_out_, I_last_out_, I_srdy_out_, I_ad_out, I_be_out_,
    CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out, C_ss_out, Disable_writes, CB_parity)"
   );;

close_theory();;
```

C.5 SU_Cont Specification

```
%------------------------------------------------------------------------------------------------------

    File:       s_phase.ml

    Author:     (c) D.A. Fura 1992

    Date:       31 March 1992

    This file contains the ml source for the phase-level specification of the P-Port of the FTEP PIU,
    an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
    The bulk this code was translated from an M-language simulation program using a translator
    written by P.J. Windley at the University of Idaho.

------------------------------------------------------------------------------------------------------%


set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm s_block.th';;

new_theory 's_block';;

map new_parent ['saux_def';'aux_def';'array_def';'wordn_def'];;

let s_state_ty = ":(sfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                    bool#bool#wordn#wordn#bool#bool#
                    sfsm_ty#bool#bool#bool#bool#bool#
                    bool#wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let s_state = "((S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_srcp, S_fsm_sdi, S_fsm_srp, S_fsm_src0, S_fsm_src1,
                S_fsm_spf, S_fsm_sc0f, S_fsm_sc1f, S_fsm_spmf, S_fsm_sb, S_fsm_src, S_fsm_sec, S_fsm_srs,
                S_fsm_scs, S_soft_shot, S_soft_shot_delA, S_soft_cntA, S_delayA, S_instart, S_cpu_histA,
                S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
                S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
                S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_cpu_hist, S_piu_fail)
               :^s_state_ty)";;

let s_env_ty = ":(bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let s_env = "((ClkA, ClkB, Rst, Bypass, Test, Gcrb, Gcrl, Failure0_, Failure1_)
              :^s_env_ty)";;

let s_out_ty = ":(wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let s_out = "((S_state, Reset_cport, Disable_int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist,
              Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail)
              :^s_out_ty)";;

%------------------------------------------------------------------------------------------------------
    Next-state definition for Phase-A instruction.
------------------------------------------------------------------------------------------------------%

let PH_A_inst_def = new_definition
 ('PH_A_inst',
"! (S_fsm_stateA S_fsm_state :sfsm_ty)
   (S_soft_cntA S_delayA S_soft_cnt S_delay :wordn)
   (S_fsm_sn S_fsm_so S_fsm_srcp S_fsm_sdi S_fsm_srp S_fsm_src0 S_fsm_src1 S_fsm_spf S_fsm_sc0f
    S_fsm_sc1f S_fsm_spmf S_fsm_sb S_fsm_src S_fsm_sec S_fsm_srs S_fsm_scs S_soft_shot S_soft_shot_delA
    S_instart S_cpu_histA S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass
    S_soft_shot_del S_bad_cpu0 S_bad_cpu1 S_reset_cpu0 S_reset_cpu1 S_pmm_fail S_cpu0_fail S_cpu1_fail
    S_cpu_hist S_piu_fail :bool)
   (ClkA ClkB Rst Bypass Test Gcrh Gcrl Failure0_ Failure1_ :bool) .
PH_A_inst (S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_srcp, S_fsm_sdi, S_fsm_srp, S_fsm_src0, S_fsm_src1,
           S_fsm_spf, S_fsm_sc0f, S_fsm_sc1f, S_fsm_spmf, S_fsm_sb, S_fsm_src, S_fsm_sec, S_fsm_srs,
           S_fsm_scs, S_soft_shot, S_soft_shot_delA, S_soft_cntA, S_delayA, S_instart, S_cpu_histA,
           S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
           S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
           S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_cpu_hist, S_piu_fail)
          (ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_) =

let new_S_fsm_stateA =
      ((S_fsm_rst) => SSTART |
       ((S_fsm_state = SSTART) => SRA |
       ((S_fsm_state = SRA) => ((S_fsm_delay6) => ((S_fsm_bypass) => SO | SPF) | SRA) |
       ((S_fsm_state = SPF) => SC0I |
       ((S_fsm_state = SC0I) => ((S_fsm_delay17) => SC0F | SC0I) |
       ((S_fsm_state = SC0F) => ST |
       ((S_fsm_state = ST) => SC1I |
       ((S_fsm_state = SC1I) => ((S_fsm_delay17) => SC1F | SC1I) |
       ((S_fsm_state = SC1F) => SS |
       ((S_fsm_state = SS) => ((S_fsm_bothbad) => SSTOP | SCS) |
       ((S_fsm_state = SSTOP) => SSTOP |
       ((S_fsm_state = SCS) => ((S_fsm_delay6) => SN | SCS) |
       ((S_fsm_state = SN) => ((S_fsm_delay17) => SO | SN) |
       ((S_fsm_state = SO) => SO | S_ILL)))))))))))))) in
let new_S_fsm_sn = (new_S_fsm_stateA = SN) in
let new_S_fsm_so = (new_S_fsm_stateA = SO) in
let new_S_fsm_srcp = (((~(new_S_fsm_stateA = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let new_S_fsm_sdi = (((~(new_S_fsm_stateA = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let new_S_fsm_srp = ((new_S_fsm_stateA = SSTART) ∨ (new_S_fsm_stateA = SRA)
                     ∨ (new_S_fsm_stateA = SC0F) ∨ (new_S_fsm_stateA = ST)
                     ∨ (new_S_fsm_stateA = SC1F) ∨ (new_S_fsm_stateA = SS)
                     ∨ (new_S_fsm_stateA = SCS)) in
let new_S_fsm_src0 = ((~(new_S_fsm_stateA = SPF)) ∧ (~(new_S_fsm_stateA = SC0I))) in
let new_S_fsm_src1 = ((~(new_S_fsm_stateA = ST)) ∧ (~(new_S_fsm_stateA = SC1I))) in
let new_S_fsm_spf = ((S_fsm_state = SRA) ∧ S_fsm_delay6 ∧ ~S_fsm_rst) in
let new_S_fsm_sc0f = (new_S_fsm_stateA = SC0F) in
let new_S_fsm_sc1f = (new_S_fsm_stateA = SC1F) in
let new_S_fsm_spmf = (new_S_fsm_stateA = SO) in
let new_S_fsm_sb = (new_S_fsm_stateA = SSTART) in
let new_S_fsm_src = ((new_S_fsm_stateA = SSTART) ∨ ((S_fsm_state = SRA) ∧ S_fsm_delay6)
                     ∨ (new_S_fsm_stateA = SC0F) ∨ (new_S_fsm_stateA = ST)
                     ∨ (new_S_fsm_stateA = SC1F) ∨ (new_S_fsm_stateA = SS)
                     ∨ ((S_fsm_state = SCS) ∧ S_fsm_delay6)) in
let new_S_fsm_sec = (((~(new_S_fsm_stateA = SSTOP)) ∧ (~(new_S_fsm_stateA = SO))) ∨ (S_fsm_state = SN)) in
let new_S_fsm_srs = (((S_fsm_state = SPF) ∧ ~S_fsm_rst) ∨ ((S_fsm_state = ST) ∧ ~S_fsm_rst)) in
let new_S_fsm_scs = (new_S_fsm_stateA = SCS) in
let new_S_soft_shot = (~Gcrh ∧ Gcrl) in
let new_S_soft_shot_delA = S_soft_shot_del in
let new_S_soft_cntA = ((new_S_fsm_srs) => (WORDN 0) | S_soft_cnt) in
let s_delay_out = ((S_fsm_sec) => (INCN 17 S_delayA) | S_delayA) in
let new_S_delayA = ((new_S_fsm_src V (new_S_fsm_scs Λ (ELEMENT s_delay_out (6)))) => (WORDN 0) | S_delay) in
let s_delay_out = ((new_S_fsm_sec) => (INCN 17 new_S_delayA) | new_S_delayA) in
let new_S_instart = ((Test) => (ELEMENT s_delay_out (5)) | (ELEMENT s_delay_out (16))) in
let s_soft_cnt_out = ((new_S_soft_shot Λ ~new_S_soft_shot_delA) =>
        (INCN 2 new_S_soft_cntA) | new_S_soft_cntA) in
let s_cpu0_ok = (new_S_fsm_sc0f Λ Failure0_ Λ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu1_ok = (new_S_fsm_sc1f Λ Failure1_ Λ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu0_select = ((new_S_fsm_sn V new_S_fsm_so) Λ ~S_cpu0_fail) in
let s_cpu1_select = ((new_S_fsm_sn V new_S_fsm_so) Λ S_cpu0_fail Λ ~S_cpu1_fail) in
let new_S_cpu_histA = (S_reset_cpu0 Λ S_reset_cpu1 Λ Bypass) in
let new_S_fsm_state = S_fsm_state in
let new_S_fsm_rst = S_fsm_rst in
let new_S_fsm_delay6 = S_fsm_delay6 in
let new_S_fsm_delay17 = S_fsm_delay17 in
let new_S_fsm_bothbad = S_fsm_bothbad in
let new_S_fsm_bypass = S_fsm_bypass in
let new_S_soft_shot_del = S_soft_shot_del in
let new_S_soft_cnt = S_soft_cnt in
let new_S_delay = S_delay in
let new_S_bad_cpu0 = S_bad_cpu0 in
let new_S_bad_cpu1 = S_bad_cpu1 in
let new_S_reset_cpu0 = S_reset_cpu0 in
let new_S_reset_cpu1 = S_reset_cpu1 in
let new_S_pmm_fail = S_pmm_fail in
let new_S_cpu0_fail = S_cpu0_fail in
let new_S_cpu1_fail = S_cpu1_fail in
let new_S_cpu_hist = S_cpu_hist in
let new_S_piu_fail = S_piu_fail in

(new_S_fsm_stateA, new_S_fsm_sn, new_S_fsm_so, new_S_fsm_srcp, new_S_fsm_sdi, new_S_fsm_srp,
 new_S_fsm_src0, new_S_fsm_src1, new_S_fsm_spf, new_S_fsm_sc0f, new_S_fsm_sc1f, new_S_fsm_spmf,
 new_S_fsm_sb, new_S_fsm_src, new_S_fsm_sec, new_S_fsm_srs, new_S_fsm_scs, new_S_soft_shot,
 new_S_soft_shot_delA, new_S_soft_cntA, new_S_delayA, new_S_instart, new_S_cpu_histA, new_S_fsm_state,
 new_S_fsm_rst, new_S_fsm_delay6, new_S_fsm_delay17, new_S_fsm_bothbad, new_S_fsm_bypass,
 new_S_soft_shot_del, new_S_soft_cnt, new_S_delay, new_S_bad_cpu0, new_S_bad_cpu1, new_S_reset_cpu0,
 new_S_reset_cpu1, new_S_pmm_fail, new_S_cpu0_fail, new_S_cpu1_fail, new_S_cpu_hist, new_S_piu_fail)"
);;

%--------------------------------------------------------------------------------------------------
   Output definition for Phase-A instruction.
--------------------------------------------------------------------------------------------------%

let PH_A_out_def = new_definition
 ('PH_A_out',
    "! (S_fsm_stateA S_fsm_state :sfsm_ty)
      (S_soft_cntA S_delayA S_soft_cnt S_delay :wordn)
      (S_fsm_sn S_fsm_so S_fsm_srcp S_fsm_sdi S_fsm_srp S_fsm_src0 S_fsm_src1 S_fsm_spf S_fsm_sc0f
       S_fsm_sc1f S_fsm_spmf S_fsm_sb S_fsm_src S_fsm_sec S_fsm_srs S_fsm_scs S_soft_shot S_soft_shot_delA
       S_instart S_cpu_histA S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass
       S_soft_shot_del S_bad_cpu0 S_bad_cpu1 S_reset_cpu0 S_reset_cpu1 S_pmm_fail S_cpu0_fail S_cpu1_fail
       S_cpu_hist S_piu_fail :bool)
    (ClkA ClkB Rst Bypass Test Gcrh Gcrl Failure0_ Failure1_ :bool) .
PH_A_out (S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_srcp, S_fsm_sdi, S_fsm_srp, S_fsm_src0, S_fsm_src1,
            S_fsm_spf, S_fsm_sc0f, S_fsm_sc1f, S_fsm_spmf, S_fsm_sb, S_fsm_src, S_fsm_sec, S_fsm_srs,
            S_fsm_scs, S_soft_shot, S_soft_shot_delA, S_soft_cntA, S_delayA, S_instart, S_cpu_histA,
            S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
            S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
            S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_cpu_hist, S_piu_fail)
           (ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_) =

 let new_S_fsm_stateA =
        ((S_fsm_rst) => SSTART |
         ((S_fsm_state = SSTART) => SRA |
         ((S_fsm_state = SRA) => ((S_fsm_delay6) => ((S_fsm_bypass) => SO | SPF) | SRA) |
         ((S_fsm_state = SPF) => SC0I |
         ((S_fsm_state = SC0I) => ((S_fsm_delay17) => SC0F | SC0I) |
         ((S_fsm_state = SC0F) => ST |
         ((S_fsm_state = ST) => SC1I |
         ((S_fsm_state = SC1I) => ((S_fsm_delay17) => SC1F | SC1I) |
         ((S_fsm_state = SC1F) => SS |
         ((S_fsm_state = SS) => ((S_fsm_bothbad) => SSTOP | SCS) |
         ((S_fsm_state = SSTOP) => SSTOP |
         ((S_fsm_state = SCS) => ((S_fsm_delay6) => SN | SCS) |
         ((S_fsm_state = SN) => ((S_fsm_delay17) => SO | SN) |
         ((S_fsm_state = SO) => SO | S_ILL)))))))))))))) in
let new_S_fsm_sn = (new_S_fsm_stateA = SN) in
let new_S_fsm_so = (new_S_fsm_stateA = SO) in
let new_S_fsm_srcp = (((~(new_S_fsm_stateA = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let new_S_fsm_sdi = (((~(new_S_fsm_stateA = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let new_S_fsm_srp = ((new_S_fsm_stateA = SSTART) ∨ (new_S_fsm_stateA = SRA)
                        ∨ (new_S_fsm_stateA = SC0F) ∨ (new_S_fsm_stateA = ST)
                        ∨ (new_S_fsm_stateA = SC1F) ∨ (new_S_fsm_stateA = SS)
                        ∨ (new_S_fsm_stateA = SCS)) in
let new_S_fsm_src0 = ((~(new_S_fsm_stateA = SPF)) ∧ (~(new_S_fsm_stateA = SC0I))) in
let new_S_fsm_src1 = ((~(new_S_fsm_stateA = ST)) ∧ (~(new_S_fsm_stateA = SC1I))) in
let new_S_fsm_spf = ((S_fsm_state = SRA) ∧ S_fsm_delay6 ∧ ~S_fsm_rst) in
let new_S_fsm_sc0f = (new_S_fsm_stateA = SC0F) in
let new_S_fsm_sc1f = (new_S_fsm_stateA = SC1F) in
let new_S_fsm_spmf = (new_S_fsm_stateA = SO) in
let new_S_fsm_sb = (new_S_fsm_stateA = SSTART) in
let new_S_fsm_src = ((new_S_fsm_stateA = SSTART) ∨ ((S_fsm_state = SRA) ∧ S_fsm_delay6)
                        ∨ (new_S_fsm_stateA = SC0F) ∨ (new_S_fsm_stateA = ST)
                        ∨ (new_S_fsm_stateA = SC1F) ∨ (new_S_fsm_stateA = SS)
                        ∨ ((S_fsm_state = SCS) ∧ S_fsm_delay6)) in
let new_S_fsm_sec = (((~(new_S_fsm_stateA = SSTOP)) ∧ (~(new_S_fsm_stateA = SO))) ∨ (S_fsm_state = SN)) in
let new_S_fsm_srs = (((S_fsm_state = SPF) ∧ ~S_fsm_rst) ∨ ((S_fsm_state = ST) ∧ ~S_fsm_rst)) in
let new_S_fsm_scs = (new_S_fsm_stateA = SCS) in
let new_S_soft_shot = (~Gcrh ∧ Gcrl) in
let new_S_soft_shot_delA = S_soft_shot_del in
let new_S_soft_cntA = ((new_S_fsm_srs) => (WORDN 0) | S_soft_cnt) in
let s_delay_out = ((S_fsm_sec) => (INCN 17 S_delayA) | S_delayA) in
let new_S_delayA = ((new_S_fsm_src ∨ (new_S_fsm_scs ∧ (ELEMENT s_delay_out (6)))) => (WORDN 0) | S_delay) in
let s_delay_out = ((new_S_fsm_sec) => (INCN 17 new_S_delayA) | new_S_delayA) in
let new_S_instart = ((Test) => (ELEMENT s_delay_out (5)) | (ELEMENT s_delay_out (16))) in
let s_soft_cnt_out = ((new_S_soft_shot ∧ ~new_S_soft_shot_delA) =>
        (INCN 2 new_S_soft_cntA) | new_S_soft_cntA) in
    let s_cpu0_ok = (new_S_fsm_sc0f ∧ Failure0_ ∧ (s_soft_cnt_out = (WORDN 5))) in
    let s_cpu1_ok = (new_S_fsm_sc1f ∧ Failure1_ ∧ (s_soft_cnt_out = (WORDN 5))) in
    let s_cpu0_select = ((new_S_fsm_sn ∨ new_S_fsm_so) ∧ ~S_cpu0_fail) in
    let s_cpu1_select = ((new_S_fsm_sn ∨ new_S_fsm_so) ∧ S_cpu0_fail ∧ ~S_cpu1_fail) in
    let new_S_cpu_histA = (S_reset_cpu0 ∧ S_reset_cpu1 ∧ Bypass) in
    let new_S_fsm_state = S_fsm_state in
    let new_S_fsm_rst = S_fsm_rst in
    let new_S_fsm_delay6 = S_fsm_delay6 in
    let new_S_fsm_delay17 = S_fsm_delay17 in
    let new_S_fsm_bothbad = S_fsm_bothbad in
    let new_S_fsm_bypass = S_fsm_bypass in
    let new_S_soft_shot_del = S_soft_shot_del in
    let new_S_soft_cnt = S_soft_cnt in
    let new_S_delay = S_delay in
    let new_S_bad_cpu0 = S_bad_cpu0 in
    let new_S_bad_cpu1 = S_bad_cpu1 in
    let new_S_reset_cpu0 = S_reset_cpu0 in
    let new_S_reset_cpu1 = S_reset_cpu1 in
    let new_S_pmm_fail = S_pmm_fail in
    let new_S_cpu0_fail = S_cpu0_fail in
    let new_S_cpu1_fail = S_cpu1_fail in
    let new_S_cpu_hist = S_cpu_hist in
    let new_S_piu_fail = S_piu_fail in
    let ss0 = (ALTER ARBN (0) ((new_S_fsm_stateA = SS) ∨ (new_S_fsm_stateA = SSTOP)
                              ∨ (new_S_fsm_stateA = SCS) ∨ (new_S_fsm_stateA = SN)
                              ∨ (new_S_fsm_stateA = SO))) in
    let ss1 = (ALTER ss0 (1) ((new_S_fsm_stateA = SC0F) ∨ (new_S_fsm_stateA = ST)
                             ∨ (new_S_fsm_stateA = SC1I) ∨ (new_S_fsm_stateA = SC1F)
                             ∨ (new_S_fsm_stateA = SS) ∨ (new_S_fsm_stateA = SSTOP)
                             ∨ (new_S_fsm_stateA = SCS))) in
    let ss2 = (ALTER ss1 (2) ((new_S_fsm_stateA = SPF) ∨ (new_S_fsm_stateA = SC0I)
                             ∨ (new_S_fsm_stateA = SC0F) ∨ (new_S_fsm_stateA = ST)
                             ∨ (new_S_fsm_stateA = SSTOP) ∨ (new_S_fsm_stateA = SO))) in
    let ss3 = (ALTER ss2 (3) ((new_S_fsm_stateA = SRA) ∨ (new_S_fsm_stateA = SPF)
                             ∨ (new_S_fsm_stateA = ST) ∨ (new_S_fsm_stateA = SC1I)
                             ∨ (new_S_fsm_stateA = SCS) ∨ (new_S_fsm_stateA = SN)
                             ∨ (new_S_fsm_stateA = SO))) in
    let S_state = ss3 in
    let Reset_cport = new_S_fsm_srcp in
    let Disable_int = (~new_S_instart ∧ ~(new_S_fsm_sn ∧ (ELEMENT s_delay_out (6))) ∧ new_S_fsm_sdi) in
    let Reset_piu = new_S_fsm_srp in
    let Reset_cpu0 = new_S_reset_cpu0 in
    let Reset_cpu1 = new_S_reset_cpu1 in
    let Cpu_hist = new_S_cpu_hist in
    let Piu_fail = new_S_piu_fail in
    let Cpu0_fail = new_S_cpu0_fail in
    let Cpu1_fail = new_S_cpu1_fail in
    let Pmm_fail = new_S_pmm_fail in

    (S_state, Reset_cport, Disable_int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist, Piu_fail, Cpu0_fail,  Cpu1_fail, Pmm_fail)"
    );;

%-----------------------------------------------------------------------------------------------
```

```
   Next-state definition for Phase-B instruction.
-------------------------------------------------------------------------------------------------%

let PH_B_inst_def = new_definition
 (`PH_B_inst`,
   "! (S_fsm_stateA S_fsm_state :sfsm_ty)
      (S_soft_cntA S_delayA S_soft_cnt S_delay :wordn)
      (S_fsm_sn S_fsm_so S_fsm_srcp S_fsm_sdi S_fsm_srp S_fsm_src0 S_fsm_src1 S_fsm_spf S_fsm_sc0f
       S_fsm_sc1f S_fsm_spmf S_fsm_sb S_fsm_src S_fsm_sec S_fsm_srs S_fsm_scs S_soft_shot S_soft_shot_delA
       S_instart S_cpu_histA S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass
       S_soft_shot_del S_bad_cpu0 S_bad_cpu1 S_reset_cpu0 S_reset_cpu1 S_pmm_fail S_cpu0_fail S_cpu1_fail
       S_cpu_hist S_piu_fail :bool)
      (ClkA ClkB Rst Bypass Test Gcrh Gcrl Failure0_ Failure1_ :bool) .
   PH_B_inst (S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_srcp, S_fsm_sdi, S_fsm_srp, S_fsm_src0, S_fsm_src1,
              S_fsm_spf, S_fsm_sc0f, S_fsm_sc1f, S_fsm_spmf, S_fsm_sb, S_fsm_src, S_fsm_sec, S_fsm_srs,
              S_fsm_scs, S_soft_shot, S_soft_shot_delA, S_soft_cntA, S_delayA, S_instart, S_cpu_histA,
              S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
              S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
              S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_cpu_hist, S_piu_fail)
             (ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_) =

   let s_soft_cnt_out = ((S_soft_shot ∧ ~S_soft_shot_delA) => (INCN 2 S_soft_cntA) | S_soft_cntA) in
   let s_delay_out = ((S_fsm_sec) => (INCN 17 S_delayA) | S_delayA) in
   let s_cpu0_ok = (S_fsm_sc0f ∧ Failure0_ ∧ (s_soft_cnt_out = (WORDN 5))) in
   let s_cpu1_ok = (S_fsm_sc1f ∧ Failure1_ ∧ (s_soft_cnt_out = (WORDN 5))) in
   let new_S_soft_shot_del = S_soft_shot in
   let new_S_soft_cnt = ((~Gcrh ∧ ~Gcrl) => (WORDN 0) | s_soft_cnt_out) in
   let new_S_delay = s_delay_out in
   let new_S_pmm_fail =
         ((S_fsm_sb ∧ ~S_fsm_spmf) => T |
          ((~S_fsm_sb ∧ S_fsm_spmf) => F |
          ((~S_fsm_sb ∧ ~S_fsm_spmf) => S_pmm_fail | ARB))) in
   let new_S_cpu0_fail =
         ((S_fsm_sb ∧ ~(s_cpu0_ok ∨ Bypass)) => T |
          ((~S_fsm_sb ∧ (s_cpu0_ok ∨ Bypass)) => F |
          ((~S_fsm_sb ∧ ~(s_cpu0_ok ∨ Bypass)) => S_cpu0_fail | ARB))) in
   let new_S_cpu1_fail =
         ((S_fsm_sb ∧ ~(s_cpu1_ok ∨ Bypass)) => T |
          ((~S_fsm_sb ∧ (s_cpu1_ok ∨ Bypass)) => F |
          ((~S_fsm_sb ∧ ~(s_cpu1_ok ∨ Bypass)) => S_cpu1_fail | ARB))) in
   let new_S_piu_fail =
         ((S_fsm_sb ∧ ~(S_fsm_spf ∨ Bypass)) => T |
          ((~S_fsm_sb ∧ (S_fsm_spf ∨ Bypass)) => F |
          ((~S_fsm_sb ∧ ~(S_fsm_spf ∨ Bypass)) => S_piu_fail | ARB))) in
   let s_cpu0_select = ((S_fsm_sn ∨ S_fsm_so) ∧ ~new_S_cpu0_fail) in
   let s_cpu1_select = ((S_fsm_sn ∨ S_fsm_so) ∧ new_S_cpu0_fail ∧ ~new_S_cpu1_fail) in
   let new_S_bad_cpu0 =
         ((S_fsm_sb ∧ ~s_cpu0_select) => T |
          ((~S_fsm_sb ∧ s_cpu0_select) => F |
          ((~S_fsm_sb ∧ ~s_cpu0_select) => S_bad_cpu0 | ARB))) in
   let new_S_bad_cpu1 =
         ((S_fsm_sb ∧ ~s_cpu1_select) => T |
          ((~S_fsm_sb ∧ s_cpu1_select) => F |
          ((~S_fsm_sb ∧ ~s_cpu1_select) => S_bad_cpu1 | ARB))) in
```

```
    let new_S_reset_cpu0 = (new_S_bad_cpu0 ∧ S_fsm_src0) in
    let new_S_reset_cpu1 = (new_S_bad_cpu1 ∧ S_fsm_src1) in
    let new_S_cpu_hist = S_cpu_histA in
    let new_S_fsm_state = S_fsm_stateA in
    let new_S_fsm_rst = Rst in
    let new_S_fsm_delay6 = (ELEMENT s_delay_out (6)) in
    let new_S_fsm_delay17 = ((Test) => (ELEMENT s_delay_out (6)) | (ELEMENT s_delay_out (17))) in
    let new_S_fsm_bothbad = (new_S_cpu0_fail ∧ new_S_cpu1_fail) in
    let new_S_fsm_bypass = Bypass in
    let new_S_fsm_stateA = S_fsm_stateA in
    let new_S_fsm_sn = S_fsm_sn in
    let new_S_fsm_so = S_fsm_so in
    let new_S_fsm_srcp = S_fsm_srcp in
    let new_S_fsm_sdi = S_fsm_sdi in
    let new_S_fsm_srp = S_fsm_srp in
    let new_S_fsm_src0 = S_fsm_src0 in
    let new_S_fsm_src1 = S_fsm_src1 in
    let new_S_fsm_spf = S_fsm_spf in
    let new_S_fsm_sc0f = S_fsm_sc0f in
    let new_S_fsm_sc1f = S_fsm_sc1f in
    let new_S_fsm_spmf = S_fsm_spmf in
    let new_S_fsm_sb = S_fsm_sb in
    let new_S_fsm_src = S_fsm_src in
    let new_S_fsm_sec = S_fsm_sec in
    let new_S_fsm_srs = S_fsm_srs in
    let new_S_fsm_scs = S_fsm_scs in
    let new_S_soft_shot = S_soft_shot in
    let new_S_soft_shot_delA = S_soft_shot_delA in
    let new_S_soft_cntA = S_soft_cntA in
    let new_S_delayA = S_delayA in
    let new_S_instart = S_instart in
    let new_S_cpu_histA = S_cpu_histA in

    (new_S_fsm_stateA, new_S_fsm_sn, new_S_fsm_so, new_S_fsm_srcp, new_S_fsm_sdi, new_S_fsm_srp,
     new_S_fsm_src0, new_S_fsm_src1, new_S_fsm_spf, new_S_fsm_sc0f, new_S_fsm_sc1f, new_S_fsm_spmf,
     new_S_fsm_sb, new_S_fsm_src, new_S_fsm_sec, new_S_fsm_srs, new_S_fsm_scs, new_S_soft_shot,
     new_S_soft_shot_delA, new_S_soft_cntA, new_S_delayA, new_S_instart, new_S_cpu_histA, new_S_fsm_state,
     new_S_fsm_rst, new_S_fsm_delay6, new_S_fsm_delay17, new_S_fsm_bothbad, new_S_fsm_bypass,
     new_S_soft_shot_del, new_S_soft_cnt, new_S_delay, new_S_bad_cpu0, new_S_bad_cpu1, new_S_reset_cpu0,
     new_S_reset_cpu1, new_S_pmm_fail, new_S_cpu0_fail, new_S_cpu1_fail, new_S_cpu_hist, new_S_piu_fail)"
    );;

%-------------------------------------------------------------------------------------------
    Output definition for Phase-B instruction.
-------------------------------------------------------------------------------------------%

let PH_B_out_def = new_definition
 ('PH_B_out',
    "! (S_fsm_stateA S_fsm_state :sfsm_ty)
       (S_soft_cntA S_delayA S_soft_cnt S_delay :wordn)
       (S_fsm_sn S_fsm_so S_fsm_srcp S_fsm_sdi S_fsm_srp S_fsm_src0 S_fsm_src1 S_fsm_spf S_fsm_sc0f
        S_fsm_sc1f S_fsm_spmf S_fsm_sb S_fsm_src S_fsm_sec S_fsm_srs S_fsm_scs S_soft_shot S_soft_shot_delA
        S_instart S_cpu_histA S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass
        S_soft_shot_del S_bad_cpu0 S_bad_cpu1 S_reset_cpu0 S_reset_cpu1 S_pmm_fail S_cpu0_fail S_cpu1_fail
    S_cpu_hist S_piu_fail :bool)
    (ClkA ClkB Rst Bypass Test Gcrh Gcrl Failure0_ Failure1_ :bool) .
PH_B_out (S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_srcp, S_fsm_sdi, S_fsm_srp, S_fsm_src0, S_fsm_src1,
          S_fsm_spf, S_fsm_sc0f, S_fsm_sc1f, S_fsm_spmf, S_fsm_sb, S_fsm_src, S_fsm_sec, S_fsm_srs,
          S_fsm_scs, S_soft_shot, S_soft_shot_delA, S_soft_cntA, S_delayA, S_instart, S_cpu_histA,
          S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
          S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
          S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_cpu_hist, S_piu_fail)
         (ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_) =

let s_soft_cnt_out = ((S_soft_shot ∧ ~S_soft_shot_delA) => (INCN 2 S_soft_cntA) | S_soft_cntA) in
let s_delay_out = ((S_fsm_sec) => (INCN 17 S_delayA) | S_delayA) in
let s_cpu0_ok = (S_fsm_sc0f ∧ Failure0_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu1_ok = (S_fsm_sc1f ∧ Failure1_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let new_S_soft_shot_del = S_soft_shot in
let new_S_soft_cnt = ((~Gcrh ∧ ~Gcrl) => (WORDN 0) | s_soft_cnt_out) in
let new_S_delay = s_delay_out in
let new_S_pmm_fail =
      ((S_fsm_sb ∧ ~S_fsm_spmf) => T |
       ((~S_fsm_sb ∧ S_fsm_spmf) => F |
       ((~S_fsm_sb ∧ ~S_fsm_spmf) => S_pmm_fail | ARB))) in
let new_S_cpu0_fail =
      ((S_fsm_sb ∧ ~(s_cpu0_ok ∨ Bypass)) => T |
       ((~S_fsm_sb ∧ (s_cpu0_ok ∨ Bypass)) => F |
       ((~S_fsm_sb ∧ ~(s_cpu0_ok ∨ Bypass)) => S_cpu0_fail | ARB))) in
let new_S_cpu1_fail =
      ((S_fsm_sb ∧ ~(s_cpu1_ok ∨ Bypass)) => T |
       ((~S_fsm_sb ∧ (s_cpu1_ok ∨ Bypass)) => F |
       ((~S_fsm_sb ∧ ~(s_cpu1_ok ∨ Bypass)) => S_cpu1_fail | ARB))) in
let new_S_piu_fail =
      ((S_fsm_sb ∧ ~(S_fsm_spf ∨ Bypass)) => T |
       ((~S_fsm_sb ∧ (S_fsm_spf ∨ Bypass)) => F |
       ((~S_fsm_sb ∧ ~(S_fsm_spf ∨ Bypass)) => S_piu_fail | ARB))) in
let s_cpu0_select = ((S_fsm_sn ∨ S_fsm_so) ∧ ~new_S_cpu0_fail) in
let s_cpu1_select = ((S_fsm_sn ∨ S_fsm_so) ∧ new_S_cpu0_fail ∧ ~new_S_cpu1_fail) in
let new_S_bad_cpu0 =
      ((S_fsm_sb ∧ ~s_cpu0_select) => T |
       ((~S_fsm_sb ∧ s_cpu0_select) => F |
       ((~S_fsm_sb ∧ ~s_cpu0_select) => S_bad_cpu0 | ARB))) in
let new_S_bad_cpu1 =
      ((S_fsm_sb ∧ ~s_cpu1_select) => T |
       ((~S_fsm_sb ∧ s_cpu1_select) => F |
       ((~S_fsm_sb ∧ ~s_cpu1_select) => S_bad_cpu1 | ARB))) in
let new_S_reset_cpu0 = (new_S_bad_cpu0 ∧ S_fsm_src0) in
let new_S_reset_cpu1 = (new_S_bad_cpu1 ∧ S_fsm_src1) in
let new_S_cpu_hist = S_cpu_histA in
let new_S_fsm_state = S_fsm_stateA in
let new_S_fsm_rst = Rst in
let new_S_fsm_delay6 = (ELEMENT s_delay_out (6)) in
let new_S_fsm_delay17 = ((Test) => (ELEMENT s_delay_out (6)) | (ELEMENT s_delay_out (17))) in
let new_S_fsm_bothbad = (new_S_cpu0_fail ∧ new_S_cpu1_fail) in
let new_S_fsm_bypass = Bypass in
let new_S_fsm_stateA = S_fsm_stateA in
let new_S_fsm_sn = S_fsm_sn in
```

```
    let new_S_fsm_so = S_fsm_so in
    let new_S_fsm_srcp = S_fsm_srcp in
    let new_S_fsm_sdi = S_fsm_sdi in
    let new_S_fsm_srp = S_fsm_srp in
    let new_S_fsm_src0 = S_fsm_src0 in
    let new_S_fsm_src1 = S_fsm_src1 in
    let new_S_fsm_spf = S_fsm_spf in
    let new_S_fsm_sc0f = S_fsm_sc0f in
    let new_S_fsm_sc1f = S_fsm_sc1f in
    let new_S_fsm_spmf = S_fsm_spmf in
    let new_S_fsm_sb = S_fsm_sb in
    let new_S_fsm_src = S_fsm_src in
    let new_S_fsm_sec = S_fsm_sec in
    let new_S_fsm_srs = S_fsm_srs in
    let new_S_fsm_scs = S_fsm_scs in
    let new_S_soft_shot = S_soft_shot in
    let new_S_soft_shot_delA = S_soft_shot_delA in
    let new_S_soft_cntA = S_soft_cntA in
    let new_S_delayA = S_delayA in
    let new_S_instart = S_instart in
    let new_S_cpu_histA = S_cpu_histA in
    let ss0 = (ALTER ARBN (0) ((new_S_fsm_stateA = SS) V (new_S_fsm_stateA = SSTOP)
                                V (new_S_fsm_stateA = SCS) V (new_S_fsm_stateA = SN)
                                V (new_S_fsm_stateA = SO))) in
    let ss1 = (ALTER ss0 (1) ((new_S_fsm_stateA = SC0F) V (new_S_fsm_stateA = ST)
                              V (new_S_fsm_stateA = SC1I) V (new_S_fsm_stateA = SC1F)
                              V (new_S_fsm_stateA = SS) V (new_S_fsm_stateA = SSTOP)
                              V (new_S_fsm_stateA = SCS))) in
    let ss2 = (ALTER ss1 (2) ((new_S_fsm_stateA = SPF) V (new_S_fsm_stateA = SC0I)
                              V (new_S_fsm_stateA = SC0F) V (new_S_fsm_stateA = ST)
                              V (new_S_fsm_stateA = SSTOP) V (new_S_fsm_stateA = SO))) in
    let ss3 = (ALTER ss2 (3) ((new_S_fsm_stateA = SRA) V (new_S_fsm_stateA = SPF)
                              V (new_S_fsm_stateA = ST) V (new_S_fsm_stateA = SC1I)
                              V (new_S_fsm_stateA = SCS) V (new_S_fsm_stateA = SN)
                              V (new_S_fsm_stateA = SO))) in
    let S_state = ss3 in
    let Reset_cport = new_S_fsm_srcp in
    let Disable_int = (~new_S_instart Λ ~(new_S_fsm_sn Λ (ELEMENT s_delay_out (6))) Λ new_S_fsm_sdi) in
    let Reset_piu = new_S_fsm_srp in
    let Reset_cpu0 = new_S_reset_cpu0 in
    let Reset_cpu1 = new_S_reset_cpu1 in
    let Cpu_hist = new_S_cpu_hist in
    let Piu_fail = new_S_piu_fail in
    let Cpu0_fail = new_S_cpu0_fail in
    let Cpu1_fail = new_S_cpu1_fail in
    let Pmm_fail = new_S_pmm_fail in

    (S_state, Reset_cport, Disable_int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist, Piu_fail, Cpu0_fail,
    Cpu1_fail, Pmm_fail)"
    );;

close_theory();;
```

Appendix D ML Source for the Clock-Level Specification of the PIU Ports.

This appendix contains the HOL models for the clock-level specification for the PIU ports. The ports are listed in the order: P_Port, M_Port, R_Port, C_Port, and SU_Cont.

D.1 P Port Specification

```
%----------------------------------------------------------------------------------------------------

    File:          p_clock1.ml

    Author:        (c) D.A. Fura 1992

    Date:          31 March 1992

    This file contains the ml source for the clock-level specification of the P-Port of the FTEP PIU,
    an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
    The bulk of this code was translated from an M-language simulation program using a translator
    written by P.J. Windley at the University of Idaho.

----------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm p_clock1.th';;

new_theory 'p_clock1';;

map new_parent ['paux_def';'aux_def';'array_def';'wordn_def'];;

let pc_state_ty = ":(wordn#bool#wordn#bool#pfsm_ty#bool#bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool)";;
let pc_state = "((P_addr, P_dest1, P_be_, P_wr, P_fsm_state, P_fsm_rst, P_fsm_sack, P_fsm_cgnt_, P_fsm_hold_,
                  P_rqt, P_size, P_down, P_lock_, P_lock_inh_, P_male_, P_rale_)
                 :^pc_state_ty)";;

let pc_env_ty = ":(bool#bool#bool#wordn#bool#bool#wordn#bool#bool#wordn#bool#bool#bool)";;
let pc_env = "((ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_, I_srdy_)
               :^pc_env_ty)";;

let pc_out_ty = ":(wordn#bool#wordn#wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool)";;
let pc_out = "((L_ad_out, L_ready_, I_ad_data_out, I_ad_addr_out, I_be_, I_rale_, I_male_, I_crqt_, I_cale_,
                I_mrdy_, I_last_, I_hlda_, I_lock_)
               :^pc_out_ty)";;

%----------------------------------------------------------------------------------------------------
    Next-state definition for EXEC instruction.
----------------------------------------------------------------------------------------------------%
let pEXEC_inst_def = new_definition
 ('pEXEC_inst',
    "! (P_fsm_state :pfsm_ty)
       (P_addr P_be_ P_size :wordn)
       (P_dest1 P_wr P_fsm_rst P_fsm_sack P_fsm_cgnt_ P_fsm_hold_ P_rqt P_down P_lock_
    P_lock_inh_ P_male_ P_rale_ :bool)
    (L_ad_in L_be_ I_ad_in:wordn)
    (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ I_cgnt_ I_hold_ I_srdy_ :bool) .

pEXEC_inst (P_addr, P_dest1, P_be_, P_wr, P_fsm_state, P_fsm_rst, P_fsm_sack, P_fsm_cgnt_, P_fsm_hold_,
              P_rqt, P_size, P_down, P_lock_, P_lock_inh_, P_male_, P_rale_)
             (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_,
              I_srdy_) =

 let new_P_fsm_state =
       ((P_fsm_rst) => PA |
        ((P_fsm_state = PH) => ((~P_fsm_hold_) => PH | PA) |
        ((P_fsm_state = PA) =>
           (((P_rqt ∧ ~P_dest1) ∨ (P_rqt ∧ P_dest1 ∧ ~P_fsm_cgnt_)) => PD |
           ((~P_fsm_hold_ ∧ P_lock_) => PH | PA)) |
        ((P_fsm_state = PD) =>
           (((P_fsm_sack ∧ P_fsm_hold_) ∨ (P_fsm_sack ∧ ~P_fsm_hold_ ∧ ~P_lock_)) => PA |
           ((P_fsm_sack ∧ ~P_fsm_hold_ ∧ P_lock_) => PH | PD)) | P_ILL)))) in
 let new_P_addr = ((~P_rqt) => (SUBARRAY L_ad_in (25,0)) | P_addr) in
 let new_P_dest1 = ((~P_rqt) => (ELEMENT L_ad_in (31)) | P_dest1) in
 let new_P_be_ = ((~P_rqt) => L_be_ | P_be_) in
 let new_P_wr = ((~P_rqt) => L_wr | P_wr) in
 let new_P_size =
     ((~P_rqt) => (SUBARRAY L_ad_in (1,0)) |
 ((P_down) => (DECN 1 P_size) | P_size)) in
 let p_ale = (~L_ads_ ∧ L_den_) in
 let p_sack = ((P_size = ((P_down) => (WORDN 1) | (WORDN 0))) ∧ ~I_srdy_ ∧ (new_P_fsm_state = PD)) in
 let new_P_rqt =
     ((p_ale ∧ ~(p_sack ∨ Rst)) => T |
 ((~p_ale ∧ (p_sack ∨ Rst)) => F |
     ((~p_ale ∧ ~(p_sack ∨ Rst)) => P_rqt | ARB))) in
 let new_P_down = (~I_srdy_ ∧ (new_P_fsm_state = PD)) in
 let new_P_male_ = ((new_P_fsm_state = PA) =>
     ~(~new_P_dest1 ∧ (~((SUBARRAY new_P_addr (25,24)) = (WORDN 3))) ∧ new_P_rqt) | P_male_) in
 let new_P_rale_ = ((new_P_fsm_state = PA) =>
     ~(~new_P_dest1 ∧ ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) ∧ new_P_rqt) | P_rale_) in
 let new_P_lock_ =
     ((Rst) => T |
     ((new_P_fsm_state = PD) => L_lock_ | P_lock_)) in
 let new_P_lock_inh_ =
     ((Rst) => T |
     ((~new_P_male_ ∨ ~new_P_rale_) => L_lock_ | P_lock_inh_)) in
 let new_P_fsm_rst = Rst in
 let new_P_fsm_sack = p_sack in
 let new_P_fsm_cgnt_ = I_cgnt_ in
 let new_P_fsm_hold_ = I_hold_ in

 (new_P_addr, new_P_dest1, new_P_be_, new_P_wr, new_P_fsm_state, new_P_fsm_rst, new_P_fsm_sack,
  new_P_fsm_cgnt_, new_P_fsm_hold_, new_P_rqt, new_P_size, new_P_down, new_P_lock_, new_P_lock_inh_,
  new_P_male_, new_P_rale_)"
 );;

%-----------------------------------------------------------------------------------------------------------
  Output definition for EXEC instruction.
```

```
--------------------------------------------------------------------------------------------------------%

let pEXEC_out_def = new_definition
('pEXEC_out',
    "! (P_fsm_state :pfsm_ty)
       (P_addr P_be_ P_size :wordn)
       (P_dest1 P_wr P_fsm_rst P_fsm_sack P_fsm_cgnt_ P_fsm_hold_ P_rqt P_down P_lock_
        P_lock_inh_ P_male_ P_rale_ :bool)
       (L_ad_in L_be_ I_ad_in:wordn)
       (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ I_cgnt_ I_hold_ I_srdy_ :bool) .
   pEXEC_out (P_addr, P_dest1, P_be_, P_wr, P_fsm_state, P_fsm_rst, P_fsm_sack, P_fsm_cgnt_, P_fsm_hold_,
              P_rqt, P_size, P_down, P_lock_, P_lock_inh_, P_male_, P_rale_)
             (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, I_ad_in, I_cgnt_, I_hold_,
              I_srdy_) =

    let new_P_fsm_state =
          ((P_fsm_rst) => PA |
           ((P_fsm_state = PH) => ((~P_fsm_hold_) => PH | PA) |
           ((P_fsm_state = PA) =>
              (((P_rqt ∧ ~P_dest1) ∨ (P_rqt ∧ P_dest1 ∧ ~P_fsm_cgnt_)) => PD |
              ((~P_fsm_hold_ ∧ P_lock_) => PH | PA)) |
           ((P_fsm_state = PD) =>
              (((P_fsm_sack ∧ P_fsm_hold_) ∨ (P_fsm_sack ∧ ~P_fsm_hold_ ∧ ~P_lock_)) => PA |
              ((P_fsm_sack ∧ ~P_fsm_hold_ ∧ P_lock_) => PH | PD)) | P_ILL)))) in
    let new_P_addr = ((~P_rqt) => (SUBARRAY L_ad_in (25,0)) | P_addr) in
    let new_P_dest1 = ((~P_rqt) => (ELEMENT L_ad_in (31)) | P_dest1) in
    let new_P_be_ = ((~P_rqt) => L_be_ | P_be_) in
    let new_P_wr = ((~P_rqt) => L_wr | P_wr) in
    let new_P_size =
        ((~P_rqt) => (SUBARRAY L_ad_in (1,0)) |
    ((P_down) => (DECN 1 P_size) | P_size)) in
    let p_ale = (~L_ads_ ∧ L_den_) in
    let p_sack = ((new_P_size = ((P_down) => (WORDN 1) | (WORDN 0))) ∧ ~I_srdy_ ∧ (new_P_fsm_state = PD)) in
    let new_P_rqt =
        ((p_ale ∧ ~(p_sack ∨ Rst)) => T |
    ((~p_ale ∧ (p_sack ∨ Rst)) => F |
        ((~p_ale ∧ ~(p_sack ∨ Rst)) => P_rqt | ARB))) in
    let new_P_down = (~I_srdy_ ∧ (new_P_fsm_state = PD)) in
    let new_P_male_ = ((new_P_fsm_state = PA) =>
        ~(~new_P_dest1 ∧ (~((SUBARRAY new_P_addr (25,24)) = (WORDN 3))) ∧ new_P_rqt) | P_male_) in
    let new_P_rale_ = ((new_P_fsm_state = PA) =>
        ~(~new_P_dest1 ∧ ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) ∧ new_P_rqt) | P_rale_) in
    let new_P_lock_ =
        ((Rst) => T |
        ((new_P_fsm_state = PD) => L_lock_ | P_lock_)) in
    let new_P_lock_inh_ =
        ((Rst) => T |
        ((~new_P_male_ ∨ ~new_P_rale_) => L_lock_ | P_lock_inh_)) in
    let new_P_fsm_rst = Rst in
    let new_P_fsm_sack = p_sack in
    let new_P_fsm_cgnt_ = I_cgnt_ in
    let new_P_fsm_hold_ = I_hold_ in
    let L_ad_out = (((~(new_P_fsm_state = PA))
                    ∧ (~(new_P_fsm_state = PH))
```

```
                ∧ ~((new_P_fsm_state = PD) ∧ new_P_wr)) => I_ad_in | ARBN) in
    let L_ready_ = ~(~I_srdy_ ∧ (new_P_fsm_state = PD)) in
    let od0 = ARBN in
    let od1 = (MALTER od0 (31,27) new_P_be_) in
    let od2 = (ALTER od1 (26) F) in
    let od3 = (MALTER od2 (25,24) (SUBARRAY new_P_addr (1,0))) in
    let od4 = (MALTER od3 (23,0) (SUBARRAY new_P_addr (25,2))) in
    let I_ad_addr_out = ((new_P_fsm_state = PA) => od4 | ARBN) in
    let I_ad_data_out = (((new_P_fsm_state = PD) ∧ new_P_wr) => L_ad_in | ARBN) in
    let I_be_ = ((~(new_P_fsm_state = PH)) => ((new_P_fsm_state = PA) => new_P_be_ | L_be_) | ARBN) in
    let I_rale_ = ((~(new_P_fsm_state = PH)) =>
        ~(~new_P_dest1 ∧ ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) ∧ (new_P_fsm_state = PA)
                                                                    ∧ new_P_rqt) | ARB) in
    let I_male_ = ((~(new_P_fsm_state = PH)) =>
        ~(~new_P_dest1 ∧ (~((SUBARRAY new_P_addr (25,24)) = (WORDN 3))) ∧ (new_P_fsm_state = PA)
                                                                    ∧ new_P_rqt) | ARB) in
    let I_crqt_ = ~(new_P_dest1 ∧ new_P_rqt) in
    let I_cale_ = ~(~I_cgnt_ ∧ (new_P_fsm_state = PA) ∧ I_hold_) in
    let I_mrdy_ = ((~(new_P_fsm_state = PH)) => F | ARB) in
    let I_last_ = ((~(new_P_fsm_state = PH)) => (P_size = ((P_down) => (WORDN 1) | (WORDN 0))) | ARB) in
    let I_hlda_ = ~(new_P_fsm_state = PH) in
    let I_lock_ = ~(~new_P_lock_ ∧ new_P_lock_inh_) in

    (L_ad_out, L_ready_, I_ad_data_out, I_ad_addr_out, I_be_, I_rale_, I_male_, I_crqt_, I_cale_, I_mrdy_,
     I_last_, I_hlda_, I_lock_)"
);;

close_theory();;
```

D.2 M Port Specification

```
%------------------------------------------------------------------------------------------

    File:        m_clock1.ml

    Author:      (c) D.A. Fura 1992

    Date:        31 March 1992

    This file contains the ml source for the clock-level specification of the M-Port of the FTEP PIU,
    an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
    The bulk of this code was translated from an M-language simulation program using a translator
    written by P.J. Windley at the University of Idaho.

------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm m_clock1.th';;

new_theory 'm_clock1';;

loadf 'abstract';;

map new_parent ['maux_def';'aux_def';'array_def';'wordn_def'];;

let mc_state_ty = ":(mfsm_ty#bool#bool#bool#bool#wordn#bool#bool#wordn#wordn#bool#bool#bool#wordn#wordn)";;
let mc_state = "((M_fsm_state, M_fsm_male_, M_fsm_last_, M_fsm_mrdy_, M_fsm_rst, M_count, M_se, M_wr, M_addr,
                  M_be, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
                 :^mc_state_ty)";;

let mc_env_ty = ":(bool#bool#bool#bool#bool#wordn#bool#bool#wordn#bool#wordn#bool#bool)";;
let mc_env = "((ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
                 I_mrdy_, MB_data_in, Edac_en_, Reset_parity)
                :^mc_env_ty)";;

let mc_out_ty = ":(wordn#bool#wordn#wordn#bool#bool#bool#bool#bool)";;
let mc_out = "((I_ad_out, I_srdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_, MB_parity)
              :^mc_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%------------------------------------------------------------------------------------------
    Next-state definition for EXEC instruction.
------------------------------------------------------------------------------------------%

let mEXEC_inst_def = new_definition
 ('mEXEC_inst',
   "! (M_fsm_state :mfsm_ty)
      (M_count M_addr M_be M_rd_data M_detect :wordn)
      (M_fsm_male_ M_fsm_last_ M_fsm_mrdy_ M_fsm_rst M_se M_wr M_rdy M_wwdel M_parity :bool)
      (I_ad_in I_be_ MB_data_in :wordn)
    (ClkA ClkB Rst Disable_eeprom Disable_writes I_male_ I_last_ I_mrdy_ Edac_en_ Reset_parity :bool)
    (rep:^rep_ty) .
mEXEC_inst (M_fsm_state, M_fsm_male_, M_fsm_last_, M_fsm_mrdy_, M_fsm_rst, M_count, M_se, M_wr, M_addr,
              M_be, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
             (ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
              I_mrdy_, MB_data_in, Edac_en_, Reset_parity)
             rep =

let m_bw = ((~(M_be = (WORDN 15))) ∧ M_wr ∧ (~(M_fsm_state = MI))) in
let m_ww = ((M_be = (WORDN 15)) ∧ M_wr ∧ (~(M_fsm_state = MI))) in
let new_M_fsm_state =
        ((M_fsm_rst) => MI |
         ((M_fsm_state = MI) => ((~M_fsm_male_) => MA | MI) |
         ((M_fsm_state = MA) =>
                ((~M_fsm_mrdy_ ∧ m_ww) => MW |
                ((~M_fsm_mrdy_ ∧ ((~M_wr ∧ (~(M_fsm_state = MI))) ∨ m_bw)) => MR | MA)) |
         ((M_fsm_state = MR) =>
                ((m_bw ∧ (M_count = (WORDN 0))) => MBW |
                ((M_fsm_last_ ∧ ~M_wr ∧ (~(M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) => MA |
                ((~M_fsm_last_ ∧ ~M_wr ∧ (~(M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) => MRR | MR))) |
         ((M_fsm_state = MRR) => MI |
         ((M_fsm_state = MW) =>
                ((~M_fsm_last_ ∧ (M_count = (WORDN 0))) => MI |
                ((M_fsm_last_ ∧ (M_count = (WORDN 0))) => MA | MW)) |
         ((M_fsm_state = MBW) => MW | M_ILL))))))) in
let new_M_se = ((~I_male_) => (ELEMENT I_ad_in (23)) | M_se) in
let new_M_wr = ((~I_male_) => (ELEMENT I_ad_in (27)) | M_wr) in
let new_M_addr =
        ((~I_male_) => (SUBARRAY I_ad_in (18,0)) |
         ((M_rdy) => (INCN 18 M_addr) | M_addr)) in
let new_M_count =
        (((new_M_fsm_state = MA) ∨ (new_M_fsm_state = MBW)) => ((new_M_se) => (WORDN 1) | (WORDN 2)) |
         (((new_M_fsm_state = MW) ∨ (new_M_fsm_state = MR)) => (DECN 2 M_count) | M_count)) in
let m_rdy = (((new_M_fsm_state = MW) ∧ (new_M_count = (WORDN 0)))
            ∨ ((new_M_fsm_state = MR) ∧ (new_M_count = (WORDN 0)) ∧ ~new_M_wr)) in
let m_srdy_ = ~((M_rdy ∧ ~new_M_wr) ∨ (m_rdy ∧ new_M_wr)) in
let new_M_be = ((~I_male_ ∨ ~m_srdy_) => (NOTN 3 I_be_) | M_be) in
let new_M_rdy = m_rdy in
let new_M_wwdel = ((new_M_fsm_state = MA) ∧ new_M_wr ∧ (new_M_be = (WORDN 15))) in
let new_M_rd_data = (((new_M_fsm_state = MR)) => (Ham_Dec rep MB_data_in) | M_rd_data) in
let new_M_detect =
        ((((new_M_fsm_state = MR) ∧ ~new_M_wr) ∨ new_M_wr ∨ (new_M_fsm_state = MI)) =>
                ((~Edac_en_) => (Ham_Det1 rep MB_data_in) | WORDN 0) | M_detect) in
let m_error = (~m_srdy_ ∧ (~(new_M_fsm_state = MI)) ∧ Ham_Det2 rep (new_M_detect, ~Edac_en_)) in
let new_M_parity =
        ((m_error ∧ ~(Rst ∨ Reset_parity)) => T |
         ((~m_error ∧ (Rst ∨ Reset_parity)) => F |
         ((~m_error ∧ ~(Rst ∨ Reset_parity)) => M_parity | ARB))) in
let new_M_fsm_male_ = I_male_ in
let new_M_fsm_last_ = I_last_ in
let new_M_fsm_mrdy_ = I_mrdy_ in
let new_M_fsm_rst = Rst in

(new_M_fsm_state, new_M_fsm_male_, new_M_fsm_last_, new_M_fsm_mrdy_, new_M_fsm_rst, new_M_count,
```

a-3

```
    new_M_se, new_M_wr, new_M_addr, new_M_be, new_M_rdy, new_M_wwdel, new_M_parity, new_M_rd_data,
    new_M_detect)"
   );;

%-----------------------------------------------------------------------------------------------
   Output definition for EXEC instruction.
-----------------------------------------------------------------------------------------------%

let mEXEC_out_def = new_definition
 ('mEXEC_out',
   "! (M_fsm_state :mfsm_ty)
      (M_count M_addr M_be M_rd_data M_detect :wordn)
      (M_fsm_male_ M_fsm_last_ M_fsm_mrdy_ M_fsm_rst M_se M_wr M_rdy M_wwdel M_parity :bool)
      (I_ad_in I_be_ MB_data_in :wordn)
      (ClkA ClkB Rst Disable_eeprom Disable_writes I_male_ I_last_ I_mrdy_ Edac_en_ Reset_parity :bool)
      (rep:^rep_ty) .
   mEXEC_out (M_fsm_state, M_fsm_male_, M_fsm_last_, M_fsm_mrdy_, M_fsm_rst, M_count, M_se, M_wr, M_addr,
                  M_be, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
                 (ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I_last_, I_be_,
                  I_mrdy_, MB_data_in, Edac_en_, Reset_parity)
                  rep =

   let m_bw = ((~(M_be = (WORDN 15))) ∧ M_wr ∧ (~(M_fsm_state = MI))) in
   let m_ww = ((M_be = (WORDN 15)) ∧ M_wr ∧ (~(M_fsm_state = MI))) in
   let new_M_fsm_state =
          ((M_fsm_rst) => MI |
           ((M_fsm_state = MI) => ((~M_fsm_male_) => MA | MI) |
           ((M_fsm_state = MA) =>
                  ((~M_fsm_mrdy_ ∧ m_ww) => MW |
                  ((~M_fsm_mrdy_ ∧ ((~M_wr ∧ (~(M_fsm_state = MI))) ∨ m_bw)) => MR | MA)) |
           ((M_fsm_state = MR) =>
                  ((m_bw ∧ (M_count = (WORDN 0))) => MBW |
                  ((M_fsm_last_ ∧ ~M_wr ∧ (~(M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) => MA |
                  ((~M_fsm_last_ ∧ ~M_wr ∧ (~(M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) => MRR | MR))) |
           ((M_fsm_state = MRR) => MI |
           ((M_fsm_state = MW) =>
                  ((~M_fsm_last_ ∧ (M_count = (WORDN 0))) => MI |
                  ((M_fsm_last_ ∧ (M_count = (WORDN 0))) => MA | MW)) |
           ((M_fsm_state = MBW) => MW | M_ILL))))))) in
   let new_M_se = ((~I_male_) => (ELEMENT I_ad_in (23)) | M_se) in
   let new_M_wr = ((~I_male_) => (ELEMENT I_ad_in (27)) | M_wr) in
   let new_M_addr =
          ((~I_male_) => (SUBARRAY I_ad_in (18,0)) |
           ((M_rdy) => (INCN 18 M_addr) | M_addr)) in
   let new_M_count =
          (((new_M_fsm_state = MA) ∨ (new_M_fsm_state = MBW)) => ((new_M_se) => (WORDN 1) | (WORDN 2)) |
           (((new_M_fsm_state = MW) ∨ (new_M_fsm_state = MR)) => (DECN 2 M_count) | M_count)) in
   let m_rdy = (((new_M_fsm_state = MW) ∧ (new_M_count = (WORDN 0)))
                ∨ ((new_M_fsm_state = MR) ∧ (new_M_count = (WORDN 0)) ∧ ~new_M_wr)) in
   let m_srdy_ = ~((M_rdy ∧ ~new_M_wr) ∨ (m_rdy ∧ new_M_wr)) in
   let new_M_be = ((~I_male_ ∨ ~m_srdy_) => (NOTN 3 I_be_) | M_be) in
   let new_M_rdy = m_rdy in
   let new_M_wwdel = ((new_M_fsm_state = MA) ∧ new_M_wr ∧ (new_M_be = (WORDN 15))) in
   let new_M_rd_data = (((new_M_fsm_state = MR)) => (Ham_Dec rep MB_data_in) | M_rd_data) in
```

```
    let new_M_detect =
          ((((new_M_fsm_state = MR) ∧ ~new_M_wr) ∨ new_M_wr ∨ (new_M_fsm_state = MI)) =>
                  ((~Edac_en_) => (Ham_Det1 rep MB_data_in) | WORDN 0) | M_detect) in
    let m_error = (~m_srdy_ ∧ (~(new_M_fsm_state = MI)) ∧ Ham_Det2 rep (new_M_detect, ~Edac_en_)) in
    let new_M_parity =
          ((m_error ∧ ~(Rst ∨ Reset_parity)) => T |
           ((~m_error ∧ (Rst ∨ Reset_parity)) => F |
           ((~m_error ∧ ~(Rst ∨ Reset_parity)) => M_parity | ARB))) in
    let new_M_fsm_male_ = I_male_ in
    let new_M_fsm_last_ = I_last_ in
    let new_M_fsm_mrdy_ = I_mrdy_ in
    let new_M_fsm_rst = Rst in
    let I_ad_out = ((~new_M_wr ∧ (~(new_M_fsm_state = MI))) => M_rd_data | ARBN) in
    let I_srdy_ = (((~(new_M_fsm_state = MI))) => m_srdy_ | ARB) in
    let MB_addr = ((M_rdy) => (INCN 18 M_addr) | M_addr) in
    let mb_data_7_0 = (((ELEMENT M_be (0))) => (SUBARRAY I_ad_in (7,0)) | (SUBARRAY M_rd_data (7,0))) in
    let mb_data_15_8 = (((ELEMENT M_be (1))) => (SUBARRAY I_ad_in (15,8)) | (SUBARRAY M_rd_data (15,8))) in
    let mb_data_23_16 = (((ELEMENT M_be (2))) => (SUBARRAY I_ad_in (23,16)) | (SUBARRAY M_rd_data (23,16))) in
    let mb_data_31_24 = (((ELEMENT M_be (3))) => (SUBARRAY I_ad_in (31,24)) | (SUBARRAY M_rd_data (31,24))) in
    let mb_data = ((MALTER (MALTER (MALTER (MALTER ARBN (7,0) mb_data_7_0)
                                                              (15,8) mb_data_15_8)
                                                              (23,16) mb_data_23_16)
                                                              (31,24) mb_data_31_24)) in
    let MB_data_out = ((new_M_fsm_state = MW) => (Ham_Enc rep mb_data) | ARBN) in
    let MB_cs_eeprom_ = ~((~(new_M_fsm_state = MI)) ∧ ~new_M_se) in
    let MB_cs_sram_ = ~((~(new_M_fsm_state = MI)) ∧ new_M_se) in
    let MB_we_ = ~((new_M_se ∨ ~(~(new_M_fsm_state = MI)) ∨ ~Disable_eeprom)
                  ∧ ~Disable_writes
                  ∧ ((new_M_fsm_state = MBW) ∨ (new_M_fsm_state = MW) ∨ new_M_wwdel)) in
    let MB_oe_ = ~((~new_M_wr ∧ (new_M_fsm_state = MA)) ∨ (new_M_fsm_state = MR)) in
    let MB_parity = new_M_parity in

    (I_ad_out, I_srdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_, MB_parity)"
    );;

close_theory();;
```

D.3 R Port Specification

```
%------------------------------------------------------------------------------------------------------------

    File:        r_clock1.ml

    Author:      (c) D.A. Fura 1992

    Date:        31 March 1992

    This file contains the ml source for the clock-level specification of the R-Port of the FTEP PIU,
    an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
    The bulk of this code was translated from an M-language simulation program using a translator
    written by P.J. Windley at the University of Idaho.

------------------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm r_clock1.th';;

new_theory 'r_clock1';;

loadf 'abstract';;

map new_parent ['raux_def';'aux_def';'array_def';'wordn_def'];;

let rc_state_ty = ":(rfsm_ty#bool#bool#bool#bool#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#
                    wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#
                    wordn#bool#wordn#bool#wordn#bool#bool#wordn#wordn#bool#wordn#wordn#bool#wordn#bool#wordn#
                    bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn)";;
let rc_state = "((R_fsm_state, R_fsm_ale_, R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_ctr0_in, R_ctr0_mux_sel, R_ctr0,
                 R_ctr0_irden, R_ctr0_new, R_ctr0_cry, R_ctr0_out, R_ctr0_orden, R_ctr1_in, R_ctr1_mux_sel,
                 R_ctr1, R_ctr1_irden, R_ctr1_new, R_ctr1_cry, R_ctr1_out, R_ctr1_orden, R_ctr2_in, R_ctr2_mux_sel,
                 R_ctr2, R_ctr2_irden, R_ctr2_new, R_ctr2_cry, R_ctr2_out, R_ctr2_orden, R_ctr3_in, R_ctr3_mux_sel,
                 R_ctr3, R_ctr3_irden, R_ctr3_new, R_ctr3_cry, R_ctr3_out, R_ctr3_orden, R_icr_load, R_icr_old,
                 R_icr_mask, R_icr_rden, R_icr, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr, R_sr_rden, R_int0_dis,
                 R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_,
                 R_reg_sel, R_busA_latch)
                :^rc_state_ty)";;

let rc_env_ty = ":(bool#bool#wordn#bool#bool#wordn#bool#bool#bool#wordn#wordn#bool#bool#
                  wordn#wordn#wordn#bool#bool#wordn)";;
let rc_env = "((ClkA, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
               Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss)
              :^rc_env_ty)";;

let r_out_ty = ":(wordn#bool#bool#bool#bool#bool#wordn#wordn#bool#bool)";;
let r_out = "((I_ad_out, I_srdy_, Int0_, Int1, Int2, Int3_, Ccr, Led, Reset_error, Pmm_invalid)
             :^r_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;
```

```
%------------------------------------------------------------------------------------------------------------
   Next-state definition for EXEC instruction.
------------------------------------------------------------------------------------------------------------%

let rEXEC_inst_def = new_definition
 ('rEXEC_inst',
   "! (rep :^rep_ty)
      (R_fsm_state :rfsm_ty)
      (R_ctr0_in R_ctr0 R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1 R_ctr1_new R_ctr1_out R_ctr2_in R_ctr2 R_ctr2_new
       R_ctr2_out R_ctr3_in R_ctr3 R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr R_ccr R_gcr R_sr R_reg_sel
       R_busA_latch :wordn)
      (R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden
       R_ctr1_mux_sel
       R_ctr1_irden R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden R_ctr3_mux_sel
       R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden R_sr_rden R_int0_dis
       R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del R_srdy_del_ :bool)
      (I_ad_in I_be_ Cpu_fail Reset_cpu S_state Id ChannelID C_ss :wordn)
      (ClkA Rst I_rale_ I_last_ I_mrdy_ Disable_int Disable_writes Piu_fail Pmm_fail CB_parity MB_parity :bool) .
   rEXEC_inst rep
            (R_fsm_state, R_fsm_ale_, R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_ctr0_in, R_ctr0_mux_sel, R_ctr0,
             R_ctr0_irden, R_ctr0_new, R_ctr0_cry, R_ctr0_out, R_ctr0_orden, R_ctr1_in, R_ctr1_mux_sel,
             R_ctr1, R_ctr1_irden, R_ctr1_new, R_ctr1_cry, R_ctr1_out, R_ctr1_orden, R_ctr2_in, R_ctr2_mux_sel,
             R_ctr2, R_ctr2_irden, R_ctr2_new, R_ctr2_cry, R_ctr2_out, R_ctr2_orden, R_ctr3_in, R_ctr3_mux_sel,
             R_ctr3, R_ctr3_irden, R_ctr3_new, R_ctr3_cry, R_ctr3_out, R_ctr3_orden, R_icr_load, R_icr_old,
             R_icr_mask, R_icr_rden, R_icr, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr, R_sr_rden, R_int0_dis,
             R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_,
             R_reg_sel, R_busA_latch)
            (ClkA, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
             Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss) =

    let new_R_fsm_state =
          ((R_fsm_rst) => RI |
           ((R_fsm_state = RI) => ((~R_fsm_ale_) => RA | RI) |
           ((R_fsm_state = RA) => ((~R_fsm_mrdy_) => RD | RA) |
           ((~R_fsm_last_) => RI | RA)))) in
    let r_fsm_cntlatch = ((R_fsm_state = RI) /\ ~R_fsm_ale_) in
    let r_fsm_srdy_ = ~((R_fsm_state = RA) /\ ~R_fsm_mrdy_) in
    let new_R_wr = ((~I_rale_) => (ELEMENT I_ad_in (27)) | R_wr) in
    let new_R_cntlatch_del = r_fsm_cntlatch in
    let new_R_srdy_del_ = r_fsm_srdy_ in
    let new_R_reg_sel =
          ((~I_rale_) => (SUBARRAY I_ad_in (3,0)) |
           ((~R_srdy_del_) => (INCN 3 R_reg_sel) | R_reg_sel)) in
    let r_reg_sel = ((~R_srdy_del_) => (INCN 3 R_reg_sel) | R_reg_sel) in
    let r_writeA = (~Disable_writes /\ R_wr /\ (new_R_fsm_state = RD)) in
    let r_writeB = (~Disable_writes /\ new_R_wr /\ (new_R_fsm_state = RD)) in
    let r_readA = (~R_wr /\ (new_R_fsm_state = RA)) in
    let r_readB = (~new_R_wr /\ (new_R_fsm_state = RA)) in
    let r_cir_wr01A = ((r_writeA /\ ((r_reg_sel = (WORDN 8)) \/ (r_reg_sel = (WORDN 9))))) in
    let r_cir_wr01B = ((r_writeB /\ ((r_reg_sel = (WORDN 8)) \/ (r_reg_sel = (WORDN 9))))) in
    let r_cir_wr23A = ((r_writeA /\ ((r_reg_sel = (WORDN 10)) \/ (r_reg_sel = (WORDN 11))))) in
    let r_cir_wr23B = ((r_writeB /\ ((r_reg_sel = (WORDN 10)) \/ (r_reg_sel = (WORDN 11))))) in
    let new_R_ccr = ((r_writeB /\ (r_reg_sel = (WORDN 3))) => I_ad_in | R_ccr) in
    let new_R_ccr_rden = (r_readB /\ (r_reg_sel = (WORDN 3))) in
let new_R_gcr = ((r_writeB ∧ (r_reg_sel = (WORDN 2))) => I_ad_in | R_gcr) in
let new_R_gcr_rden = (r_readB ∧ (r_reg_sel = (WORDN 2))) in
let new_R_c01_cout_del = R_ctr1_cry in
let new_R_int1_en =
        ((((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
         ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => T |
         ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
         (~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => F |
         ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
         ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => R_int1_en | ARB))) in
let new_R_c23_cout_del = R_ctr3_cry in
let new_R_int2_en =
        ((((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
         ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => T |
         ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
         (~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => F |
         ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
         ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => R_int2_en | ARB))) in
let new_R_ctr0_in = ((r_writeB ∧ (r_reg_sel = (WORDN 8))) => I_ad_in | R_ctr0_in) in
let new_R_ctr0_mux_sel = (r_cir_wr01B ∨ ((ELEMENT new_R_gcr (16)) ∧ R_ctr1_cry)) in
let new_R_ctr0_irden = (r_readB ∧ (r_reg_sel = (WORDN 8))) in
let new_R_ctr0 = ((R_ctr0_mux_sel) => R_ctr0_in | R_ctr0_new) in
let new_R_ctr0_new = (((ELEMENT new_R_gcr (19))) => (INCN 31 R_ctr0) | R_ctr0) in
let new_R_ctr0_cry = ((ONES 31 R_ctr0) ∧ (ELEMENT new_R_gcr (19))) in
let new_R_ctr0_out = ((r_fsm_cntlatch) => R_ctr0_new | R_ctr0_out) in
let new_R_ctr0_orden = (r_readB ∧ (r_reg_sel = (WORDN 12))) in
let new_R_ctr1_in = ((r_writeB ∧ (r_reg_sel = (WORDN 9))) => I_ad_in | R_ctr1_in) in
let new_R_ctr1_mux_sel = (r_cir_wr01B ∨ ((ELEMENT new_R_gcr (16)) ∧ R_ctr1_cry)) in
let new_R_ctr1_irden = (r_readB ∧ (r_reg_sel = (WORDN 9))) in
let new_R_ctr1 = ((R_ctr1_mux_sel) => R_ctr1_in | R_ctr1_new) in
let new_R_ctr1_new = ((R_ctr0_cry) => (INCN 31 R_ctr1) | R_ctr1) in
let new_R_ctr1_cry = ((ONES 31 R_ctr1) ∧ R_ctr0_cry) in
let new_R_ctr1_out = ((R_cntlatch_del) => R_ctr1_new | R_ctr1_out) in
let new_R_ctr1_orden = (r_readB ∧ (r_reg_sel = (WORDN 13))) in
let new_R_ctr2_in = ((r_writeB ∧ (r_reg_sel = (WORDN 10))) => I_ad_in | R_ctr2_in) in
let new_R_ctr2_mux_sel = ((r_cir_wr23B ∨ ((ELEMENT new_R_gcr (20)) ∧ R_ctr3_cry))) in
let new_R_ctr2_irden = (r_readB ∧ (r_reg_sel = (WORDN 10))) in
let new_R_ctr2 = ((R_ctr2_mux_sel) => R_ctr2_in | R_ctr2_new) in
let new_R_ctr2_new = (((ELEMENT new_R_gcr (23))) => (INCN 31 R_ctr2) | R_ctr2) in
let new_R_ctr2_cry = ((ONES 31 R_ctr2) ∧ (ELEMENT new_R_gcr (23))) in
let new_R_ctr2_out = ((r_fsm_cntlatch) => R_ctr2_new | R_ctr2_out) in
let new_R_ctr2_orden = (r_readB ∧ (r_reg_sel = (WORDN 14))) in
let new_R_ctr3_in = ((r_writeB ∧ (r_reg_sel = (WORDN 11))) => I_ad_in | R_ctr3_in) in
let new_R_ctr3_mux_sel = ((r_cir_wr23B ∨ ((ELEMENT new_R_gcr (20)) ∧ R_ctr3_cry))) in
let new_R_ctr3_irden = (r_readB ∧ (r_reg_sel = (WORDN 11))) in
let new_R_ctr3 = ((R_ctr3_mux_sel) => R_ctr3_in | R_ctr3_new) in
let new_R_ctr3_new = ((R_ctr2_cry) => (INCN 31 R_ctr3) | R_ctr3) in
let new_R_ctr3_cry = ((ONES 31 R_ctr3) ∧ R_ctr3_cry) in
let new_R_ctr3_out = ((R_cntlatch_del) => R_ctr3_new | R_ctr3_out) in
let new_R_ctr3_orden = (r_readB ∧ (r_reg_sel = (WORDN 15))) in
let new_R_icr_load = (r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let new_R_icr_old =
       ((r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => R_icr | R_icr_old) in
let new_R_icr_mask =
        ((r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => I_ad_in | R_icr_mask) in
let new_R_icr =
        ((R_icr_load) =>
           ((~(r_reg_sel = (WORDN 1))) => (Andn rep (R_icr_old, R_icr_mask)) | (Orn rep (R_icr_old, R_icr_mask))) |
         R_icr) in
let new_R_icr_rden = ((new_R_fsm_state = RA) ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let sr28 = (ALTER ARBN (28) MB_parity) in
let sr28_25 = (MALTER sr28 (27,25) C_ss) in
let sr28_24 = (ALTER sr28_25 (24) CB_parity) in
let sr28_22 = (MALTER sr28_24 (23,22) ChannelID) in
let sr28_16 = (MALTER sr28_22 (21,16) Id) in
let sr28_12 = (MALTER sr28_16 (15,12) S_state) in
let sr28_9 = (ALTER sr28_12 (9) Pmm_fail) in
let sr28_8 = (ALTER sr28_9 (8) Piu_fail) in
let sr28_2 = (MALTER sr28_8 (3,2) Reset_cpu) in
let sr28_0 = (MALTER sr28_2 (1,0) Cpu_fail) in
let new_R_sr = ((r_fsm_cntlatch) => sr28_0 | R_sr) in
let new_R_sr_rden = (r_readB ∧ (r_reg_sel = (WORDN 4))) in
let r_int0_en = (((ELEMENT R_icr (0)) ∧ (ELEMENT R_icr (8))) ∨
                 ((ELEMENT R_icr (1)) ∧ (ELEMENT R_icr (9))) ∨
                 ((ELEMENT R_icr (2)) ∧ (ELEMENT R_icr (10))) ∨
                 ((ELEMENT R_icr (3)) ∧ (ELEMENT R_icr (11))) ∨
                 ((ELEMENT R_icr (4)) ∧ (ELEMENT R_icr (12))) ∨
                 ((ELEMENT R_icr (5)) ∧ (ELEMENT R_icr (13))) ∨
                 ((ELEMENT R_icr (6)) ∧ (ELEMENT R_icr (14))) ∨
                 ((ELEMENT R_icr (7)) ∧ (ELEMENT R_icr (15)))) in
let new_R_int0_dis = r_int0_en in
let r_int3_en = (((ELEMENT R_icr (16)) ∧ (ELEMENT R_icr (24))) ∨
                 ((ELEMENT R_icr (17)) ∧ (ELEMENT R_icr (25))) ∨
                 ((ELEMENT R_icr (18)) ∧ (ELEMENT R_icr (26))) ∨
                 ((ELEMENT R_icr (19)) ∧ (ELEMENT R_icr (27))) ∨
                 ((ELEMENT R_icr (20)) ∧ (ELEMENT R_icr (28))) ∨
                 ((ELEMENT R_icr (21)) ∧ (ELEMENT R_icr (29))) ∨
                 ((ELEMENT R_icr (22)) ∧ (ELEMENT R_icr (30))) ∨
                 ((ELEMENT R_icr (23)) ∧ (ELEMENT R_icr (31)))) in
let new_R_int3_dis = r_int3_en in
let new_R_busA_latch =
        ((R_ctr0_irden) => R_ctr0_in |
         ((R_ctr0_orden) => R_ctr0_out |
         ((R_ctr1_irden) => R_ctr1_in |
         ((R_ctr1_orden) => R_ctr1_out |
         ((R_ctr2_irden) => R_ctr2_in |
         ((R_ctr2_orden) => R_ctr2_out |
         ((R_ctr3_irden) => R_ctr3_in |
         ((R_ctr3_orden) => R_ctr3_out |
         ((R_icr_rden) => new_R_icr |
         ((R_ccr_rden) => R_ccr |
         ((R_gcr_rden) => R_gcr |
         ((R_sr_rden) => R_sr | ARB)))))))))))) in
let new_R_fsm_ale_ = I_rale_ in
let new_R_fsm_mrdy_ = I_mrdy_ in
let new_R_fsm_last_ = I_last_ in
let new_R_fsm_rst = Rst in
```

```
    (new_R_fsm_state, new_R_fsm_ale_, new_R_fsm_mrdy_, new_R_fsm_last_, new_R_fsm_rst, new_R_ctr0_in,
     new_R_ctr0_mux_sel, new_R_ctr0, new_R_ctr0_irden, new_R_ctr0_new, new_R_ctr0_cry, new_R_ctr0_out,
     new_R_ctr0_orden, new_R_ctr1_in, new_R_ctr1_mux_sel, new_R_ctr1, new_R_ctr1_irden, new_R_ctr1_new,
     new_R_ctr1_cry,
     new_R_ctr1_out, new_R_ctr1_orden, new_R_ctr2_in, new_R_ctr2_mux_sel, new_R_ctr2, new_R_ctr2_irden,
     new_R_ctr2_new,
     new_R_ctr2_cry, new_R_ctr2_out, new_R_ctr2_orden, new_R_ctr3_in, new_R_ctr3_mux_sel, new_R_ctr3,
     new_R_ctr3_irden,
     new_R_ctr3_new, new_R_ctr3_cry, new_R_ctr3_out, new_R_ctr3_orden, new_R_icr_load, new_R_icr_old,
     new_R_icr_mask,
     new_R_icr_rden, new_R_icr, new_R_ccr, new_R_ccr_rden, new_R_gcr, new_R_gcr_rden, new_R_sr, new_R_sr_rden,
     new_R_int0_dis, new_R_int3_dis, new_R_c01_cout_del, new_R_int1_en, new_R_c23_cout_del, new_R_int2_en,
     new_R_wr,
     new_R_cntlatch_del, new_R_srdy_del_, new_R_reg_sel, new_R_busA_latch)"
   );;

%----------------------------------------------------------------------------------------------------------
   Output definition for EXEC instruction.
----------------------------------------------------------------------------------------------------------%

let rEXEC_out_def = new_definition
('rEXEC_out',
    "! (rep :^rep_ty)
       (R_fsm_state :rfsm_ty)
       (R_ctr0_in R_ctr0 R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1 R_ctr1_new R_ctr1_out R_ctr2_in R_ctr2 R_ctr2_new
        R_ctr2_out R_ctr3_in R_ctr3 R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr R_ccr R_gcr R_sr R_reg_sel
        R_busA_latch :wordn)
       (R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden
        R_ctr1_mux_sel
        R_ctr1_irden R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden R_ctr3_mux_sel
        R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden R_sr_rden R_int0_dis
        R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del R_srdy_del_ :bool)
       (I_ad_in I_be_ Cpu_fail Reset_cpu S_state Id ChannelID C_ss :wordn)
       (ClkA Rst I_rale_ I_last_ I_mrdy_ Disable_int Disable_writes Piu_fail Pmm_fail CB_parity MB_parity :bool) .
   rEXEC_out rep
              (R_fsm_state, R_fsm_ale_, R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_ctr0_in, R_ctr0_mux_sel, R_ctr0,
               R_ctr0_irden, R_ctr0_new, R_ctr0_cry, R_ctr0_out, R_ctr0_orden, R_ctr1_in, R_ctr1_mux_sel,
               R_ctr1, R_ctr1_irden, R_ctr1_new, R_ctr1_cry, R_ctr1_out, R_ctr1_orden, R_ctr2_in, R_ctr2_mux_sel,
               R_ctr2, R_ctr2_irden, R_ctr2_new, R_ctr2_cry, R_ctr2_out, R_ctr2_orden, R_ctr3_in, R_ctr3_mux_sel,
               R_ctr3, R_ctr3_irden, R_ctr3_new, R_ctr3_cry, R_ctr3_out, R_ctr3_orden, R_icr_load, R_icr_old,
               R_icr_mask, R_icr_rden, R_icr, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr, R_sr_rden, R_int0_dis,
               R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_,
               R_reg_sel, R_busA_latch)
              (ClkA, Rst, I_ad_in, I_rale_, I_last_, I_be_, I_mrdy_, Disable_int, Disable_writes,
               Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss) =

    let new_R_fsm_state =
          ((R_fsm_rst) => RI |
           ((R_fsm_state = RI) => ((~R_fsm_ale_) => RA | RI) |
           ((R_fsm_state = RA) => ((~R_fsm_mrdy_) => RD | RA) |
           ((~R_fsm_last_) => RI | RA)))) in
    let r_fsm_cntlatch = ((R_fsm_state = RI) /\ ~R_fsm_ale_) in
    let r_fsm_srdy_ = ~((R_fsm_state = RA) /\ ~R_fsm_mrdy_) in
    let new_R_wr = ((~I_rale_) => (ELEMENT I_ad_in (27)) | R_wr) in
let new_R_cntlatch_del = r_fsm_cntlatch in
let new_R_srdy_del_ = r_fsm_srdy_ in
let new_R_reg_sel =
        ((~I_rale_) => (SUBARRAY I_ad_in (3,0)) |
         ((~R_srdy_del_) => (INCN 3 R_reg_sel) | R_reg_sel)) in
let r_reg_sel = ((~R_srdy_del_) => (INCN 3 R_reg_sel) | R_reg_sel) in
let r_writeA = (~Disable_writes ∧ R_wr ∧ (new_R_fsm_state = RD)) in
let r_writeB = (~Disable_writes ∧ new_R_wr ∧ (new_R_fsm_state = RD)) in
let r_readA = (~R_wr ∧ (new_R_fsm_state = RA)) in
let r_readB = (~new_R_wr ∧ (new_R_fsm_state = RA)) in
let r_cir_wr01A = ((r_writeA ∧ ((r_reg_sel = (WORDN 8)) ∨ (r_reg_sel = (WORDN 9))))) in
let r_cir_wr01B = ((r_writeB ∧ ((r_reg_sel = (WORDN 8)) ∨ (r_reg_sel = (WORDN 9))))) in
let r_cir_wr23A = ((r_writeA ∧ ((r_reg_sel = (WORDN 10)) ∨ (r_reg_sel = (WORDN 11))))) in
let r_cir_wr23B = ((r_writeB ∧ ((r_reg_sel = (WORDN 10)) ∨ (r_reg_sel = (WORDN 11))))) in
let new_R_ccr = ((r_writeB ∧ (r_reg_sel = (WORDN 3))) => I_ad_in | R_ccr) in
let new_R_ccr_rden = (r_readB ∧ (r_reg_sel = (WORDN 3))) in
let new_R_gcr = ((r_writeB ∧ (r_reg_sel = (WORDN 2))) => I_ad_in | R_gcr) in
let new_R_gcr_rden = (r_readB ∧ (r_reg_sel = (WORDN 2))) in
let new_R_c01_cout_del = R_ctr1_cry in
let new_R_int1_en =
        ((((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
          ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => T |
          ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
          (~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => F |
          ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
          ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => R_int1_en | ARB))) in
let new_R_c23_cout_del = R_ctr3_cry in
let new_R_int2_en =
        ((((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
          ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => T |
          ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
          (~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => F |
          ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
          ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => R_int2_en | ARB))) in
let new_R_ctr0_in = ((r_writeB ∧ (r_reg_sel = (WORDN 8))) => I_ad_in | R_ctr0_in) in
let new_R_ctr0_mux_sel = (r_cir_wr01B ∨ ((ELEMENT new_R_gcr (16)) ∧ R_ctr1_cry)) in
let new_R_ctr0_irden = (r_readB ∧ (r_reg_sel = (WORDN 8))) in
let new_R_ctr0 = ((R_ctr0_mux_sel) => R_ctr0_in | R_ctr0_new) in
let new_R_ctr0_new = (((ELEMENT new_R_gcr (19))) => (INCN 31 R_ctr0) | R_ctr0) in
let new_R_ctr0_cry = ((ONES 31 R_ctr0) ∧ (ELEMENT new_R_gcr (19))) in
let new_R_ctr0_out = ((r_fsm_cntlatch) => R_ctr0_new | R_ctr0_out) in
let new_R_ctr0_orden = (r_readB ∧ (r_reg_sel = (WORDN 12))) in
let new_R_ctr1_in = ((r_writeB ∧ (r_reg_sel = (WORDN 9))) => I_ad_in | R_ctr1_in) in
let new_R_ctr1_mux_sel = (r_cir_wr01B ∨ ((ELEMENT new_R_gcr (16)) ∧ R_ctr1_cry)) in
let new_R_ctr1_irden = (r_readB ∧ (r_reg_sel = (WORDN 9))) in
let new_R_ctr1 = ((R_ctr1_mux_sel) => R_ctr1_in | R_ctr1_new) in
let new_R_ctr1_new = ((R_ctr0_cry) => (INCN 31 R_ctr1) | R_ctr1) in
let new_R_ctr1_cry = ((ONES 31 R_ctr1) ∧ R_ctr0_cry) in
let new_R_ctr1_out = ((R_cntlatch_del) => R_ctr1_new | R_ctr1_out) in
let new_R_ctr1_orden = (r_readB ∧ (r_reg_sel = (WORDN 13))) in
let new_R_ctr2_in = ((r_writeB ∧ (r_reg_sel = (WORDN 10))) => I_ad_in | R_ctr2_in) in
let new_R_ctr2_mux_sel = ((r_cir_wr23B ∨ ((ELEMENT new_R_gcr (20)) ∧ R_ctr3_cry))) in
let new_R_ctr2_irden = (r_readB ∧ (r_reg_sel = (WORDN 10))) in
let new_R_ctr2 = ((R_ctr2_mux_sel) => R_ctr2_in | R_ctr2_new) in
let new_R_ctr2_new = (((ELEMENT new_R_gcr (23))) => (INCN 31 R_ctr2) | R_ctr2) in
let new_R_ctr2_cry = ((ONES 31 R_ctr2) ∧ (ELEMENT new_R_gcr (23))) in
let new_R_ctr2_out = ((r_fsm_cntlatch) => R_ctr2_new | R_ctr2_out) in
let new_R_ctr2_orden = (r_readB ∧ (r_reg_sel = (WORDN 14))) in
let new_R_ctr3_in = ((r_writeB ∧ (r_reg_sel = (WORDN 11))) => I_ad_in | R_ctr3_in) in
let new_R_ctr3_mux_sel = ((r_cir_wr23B ∨ ((ELEMENT new_R_gcr (20)) ∧ R_ctr3_cry))) in
let new_R_ctr3_irden = (r_readB ∧ (r_reg_sel = (WORDN 11))) in
let new_R_ctr3 = ((R_ctr3_mux_sel) => R_ctr3_in | R_ctr3_new) in
let new_R_ctr3_new = ((R_ctr2_cry) => (INCN 31 R_ctr3) | R_ctr3) in
let new_R_ctr3_cry = ((ONES 31 R_ctr3) ∧ R_ctr3_cry) in
let new_R_ctr3_out = ((R_cntlatch_del) => R_ctr3_new | R_ctr3_out) in
let new_R_ctr3_orden = (r_readB ∧ (r_reg_sel = (WORDN 15))) in
let new_R_icr_load = (r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let new_R_icr_old =
        ((r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => R_icr | R_icr_old) in
let new_R_icr_mask =
        ((r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => I_ad_in | R_icr_mask) in
let new_R_icr =
        ((R_icr_load) =>
            ((~(r_reg_sel = (WORDN 1))) => (Andn rep (R_icr_old, R_icr_mask)) | (Orn rep (R_icr_old, R_icr_mask))) |
         R_icr) in
let new_R_icr_rden = ((new_R_fsm_state = RA) ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let sr28 = (ALTER ARBN (28) MB_parity) in
let sr28_25 = (MALTER sr28 (27,25) C_ss) in
let sr28_24 = (ALTER sr28_25 (24) CB_parity) in
let sr28_22 = (MALTER sr28_24 (23,22) ChannelID) in
let sr28_16 = (MALTER sr28_22 (21,16) Id) in
let sr28_12 = (MALTER sr28_16 (15,12) S_state) in
let sr28_9 = (ALTER sr28_12 (9) Pmm_fail) in
let sr28_8 = (ALTER sr28_9 (8) Piu_fail) in
let sr28_2 = (MALTER sr28_8 (3,2) Reset_cpu) in
let sr28_0 = (MALTER sr28_2 (1,0) Cpu_fail) in
let new_R_sr = ((r_fsm_cntlatch) => sr28_0 | R_sr) in
let new_R_sr_rden = (r_readB ∧ (r_reg_sel = (WORDN 4))) in
let r_int0_en = (((ELEMENT R_icr (0)) ∧ (ELEMENT R_icr (8))) ∨
                ((ELEMENT R_icr (1)) ∧ (ELEMENT R_icr (9))) ∨
                ((ELEMENT R_icr (2)) ∧ (ELEMENT R_icr (10))) ∨
                ((ELEMENT R_icr (3)) ∧ (ELEMENT R_icr (11))) ∨
                ((ELEMENT R_icr (4)) ∧ (ELEMENT R_icr (12))) ∨
                ((ELEMENT R_icr (5)) ∧ (ELEMENT R_icr (13))) ∨
                ((ELEMENT R_icr (6)) ∧ (ELEMENT R_icr (14))) ∨
                ((ELEMENT R_icr (7)) ∧ (ELEMENT R_icr (15)))) in
let new_R_int0_dis = r_int0_en in
let r_int3_en = (((ELEMENT R_icr (16)) ∧ (ELEMENT R_icr (24))) ∨
                ((ELEMENT R_icr (17)) ∧ (ELEMENT R_icr (25))) ∨
                ((ELEMENT R_icr (18)) ∧ (ELEMENT R_icr (26))) ∨
                ((ELEMENT R_icr (19)) ∧ (ELEMENT R_icr (27))) ∨
                ((ELEMENT R_icr (20)) ∧ (ELEMENT R_icr (28))) ∨
                ((ELEMENT R_icr (21)) ∧ (ELEMENT R_icr (29))) ∨
                ((ELEMENT R_icr (22)) ∧ (ELEMENT R_icr (30))) ∨
                ((ELEMENT R_icr (23)) ∧ (ELEMENT R_icr (31)))) in
let new_R_int3_dis = r_int3_en in
let new_R_busA_latch =
        ((R_ctr0_irden) => R_ctr0_in |
        ((R_ctr0_orden) => R_ctr0_out |
        ((R_ctr1_irden) => R_ctr1_in |
        ((R_ctr1_orden) => R_ctr1_out |
        ((R_ctr2_irden) => R_ctr2_in |
        ((R_ctr2_orden) => R_ctr2_out |
        ((R_ctr3_irden) => R_ctr3_in |
        ((R_ctr3_orden) => R_ctr3_out |
        ((R_icr_rden) => new_R_icr |
        ((R_ccr_rden) => R_ccr |
        ((R_gcr_rden) => R_gcr |
        ((R_sr_rden) => R_sr | ARB))))))))))) in
let new_R_fsm_ale_ = I_rale_ in
let new_R_fsm_mrdy_ = I_mrdy_ in
let new_R_fsm_last_ = I_last_ in
let new_R_fsm_rst = Rst in

let I_ad_out = ((~R_wr ∧ ((new_R_fsm_state = RA) ∨ (new_R_fsm_state = RD))) => new_R_busA_latch | ARBN) in
let I_srdy_ =
        (((new_R_fsm_state = RA) ∨ (new_R_fsm_state = RD)) => ~((R_fsm_state = RA) ∧ (new_R_fsm_state = RD)) |
                                                                        ARB) in
let Int0_ = ~(r_int0_en ∧ ~R_int0_dis ∧ ~Disable_int) in
let Int1 = (R_ctr1_cry ∧ new_R_int1_en ∧ ~Disable_int) in
let Int2 = (R_ctr3_cry ∧ new_R_int2_en ∧ ~Disable_int) in
let Int3_ = ~(r_int3_en ∧ ~R_int3_dis ∧ ~Disable_int) in
let Ccr = R_ccr in
let Led = (SUBARRAY new_R_gcr (3,0)) in
let Reset_error = (ELEMENT new_R_gcr (24)) in
let Pmm_invalid = (ELEMENT new_R_gcr (28)) in

(I_ad_out, I_srdy_, Int0_, Int1, Int2, Int3_, Ccr, Led, Reset_error, Pmm_invalid)"
);;
```

D.4 C Port Specification

```
%----------------------------------------------------------------------------------------------------

    File:          c_clock1.ml

    Author:        (c) D.A. Fura 1992

    Date:          31 March 1992

    This file contains the ml source for the clock-level specification of the C-Port of the FTEP PIU,
    an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
    The bulk of this code was translated from an M-language simulation program using a translator
    written by P.J. Windley at the University of Idaho.

----------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm c_clock1.th';;

new_theory 'c_clock1';;

loadf 'abstract';;

map new_parent ['caux_def';'aux_def';'array_def';'wordn_def'];;

let MSTART = "WORDN 4";;
let MEND = "WORDN 5";;
let MRDY = "WORDN 6";;
let MWAIT = "WORDN 7";;
let MABORT = "WORDN 0";;

let SACK = "WORDN 5";;
let SRDY = "WORDN 6";;
let SWAIT = "WORDN 7";;
let SABORT = "WORDN 0";;

let cc_state_ty = ":(cmfsm_ty#bool#bool#bool#bool#wordn#bool#
                    csfsm_ty#bool#bool#bool#wordn#
                    cefsm_ty#bool#bool#bool#bool#bool#bool#
                    bool#wordn#bool#bool#bool#wordn#bool#
                    bool#bool#bool#bool#bool#bool#bool#
                    bool#bool#bool#wordn#wordn#wordn#wordn#wordn#wordn)";;
let cc_state = "((C_mfsm_state,C_mfsm_D,C_mfsm_rst,C_mfsm_crqt_,C_mfsm_hold_,C_mfsm_ss,C_mfsm_invalid,
                 C_sfsm_state,C_sfsm_D,C_sfsm_rst,C_sfsm_hlda_,C_sfsm_ms,
                 C_efsm_state,C_efsm_cale_,C_efsm_last_,C_efsm_male_,C_efsm_rale_,C_efsm_srdy_,C_efsm_rst,
                 C_wr,C_sizewrbe,C_clkA,C_last_in_,C_lock_in_,C_ss,C_last_out_,
                 C_hold_,C_holdA_,C_cout_0_le_del,C_cin_2_le,C_mrdy_del_,C_iad_en_s_del,C_iad_en_s_delA,
                 C_wrdy,C_rrdy,C_parity,C_source,C_data_in,C_iad_out,C_iad_in,C_a1a0,C_a3a2)
                :^cc_state_ty)";;

let cc_env_ty = ":(wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#
                  wordn#wordn#wordn#wordn#bool#bool#bool#bool#wordn#wordn#bool#bool#wordn#bool)";;
let cc_env = "((I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_,
               I_lock_, I_cale_, I_hlda_, I_crqt_,
               CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in,
               Rst, ClkA, ClkB, ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr,
               Reset_error)
               :^cc_env_ty)";;

let cc_out_ty = ":(bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
                  bool#wordn#wordn#wordn#wordn#bool#bool)";;
let cc_out = "((I_cgnt_, I_mrdy_out_, I_hold_, I_rale_out_, I_male_out_, I_last_out_, I_srdy_out_,
               I_ad_out, I_be_out_,
               CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out, C_ss_out, Disable_writes, CB_parity)
               :^cc_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%--------------------------------------------------------------------------------------------------
   Next-state definition for EXEC instruction.
--------------------------------------------------------------------------------------------------%

let cEXEC_inst_def = new_definition
   ('cEXEC_inst',
   "! (rep:^rep_ty)
      (C_mfsm_state:cmfsm_ty) (C_sfsm_state:csfsm_ty) (C_efsm_state:cefsm_ty)
      (C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss C_source C_data_in C_iad_out C_iad_in C_a1a0 C_a3a2 :wordn)
      (C_mfsm_D C_mfsm_rst C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_invalid C_sfsm_D C_sfsm_rst C_sfsm_hlda_
      C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
      C_wr C_clkA C_last_in_ C_lock_in_ C_last_out_ C_hold_ C_holdA_ C_cout_0_le_del C_cin_2_le
      C_mrdy_del_ C_iad_en_s_del C_iad_en_s_delA C_wrdy C_rrdy C_parity :bool)
      (I_ad_in I_be_in_ CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID Ccr :wordn)
      (I_mrdy_in_ I_rale_in_ I_male_in_ I_last_in_ I_srdy_in_ I_lock_ I_cale_ I_hlda_ I_crqt_
      Rst ClkA ClkB ClkD Pmm_failure Piu_invalid Reset_error :bool) .
   cEXEC_inst rep
            (C_mfsm_state, C_mfsm_D, C_mfsm_rst, C_mfsm_crqt_, C_mfsm_hold_, C_mfsm_ss, C_mfsm_invalid,
             C_sfsm_state, C_sfsm_D, C_sfsm_rst, C_sfsm_hlda_, C_sfsm_ms,
             C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_, C_efsm_rst,
             C_wr, C_sizewrbe, C_clkA, C_last_in_, C_lock_in_, C_ss, C_last_out_,
             C_hold_, C_holdA_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_iad_en_s_delA,
             C_wrdy, C_rrdy, C_parity, C_source, C_data_in, C_iad_out, C_iad_in, C_a1a0,C_a3a2)
            (I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_,
             I_lock_, I_cale_, I_hlda_, I_crqt_, CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in,
             Rst, ClkA, ClkB, ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr, Reset_error) =

   let c_write = (((~(C_mfsm_state = CMI)) /\ (~(C_mfsm_state = CMR))) => C_wr | (ELEMENT C_sizewrbe (5))) in
   let c_busy = (~((SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
   let c_grant = ((((SUBARRAY Id (1,0)) = (WORDN 0)) /\ ~(ELEMENT CB_rqt_in_ (0)))
               \/ (((SUBARRAY Id (1,0)) = (WORDN 1)) /\ ~(ELEMENT CB_rqt_in_ (0))
                                                    /\ (ELEMENT CB_rqt_in_ (1)))
               \/ (((SUBARRAY Id (1,0)) = (WORDN 2)) /\ ~(ELEMENT CB_rqt_in_ (0))
                                                    /\ (ELEMENT CB_rqt_in_ (1))
                                                    /\ (ELEMENT CB_rqt_in_ (2)))
               \/ (((SUBARRAY Id (1,0)) = (WORDN 3)) /\ ~(ELEMENT CB_rqt_in_ (0))
                                                    /\ (ELEMENT CB_rqt_in_ (1))
```

```
                                                   ∧ (ELEMENT CB_rqt_in_ (2))
                                                   ∧ (ELEMENT CB_rqt_in_ (3)))) in
let c_addressed = (Id = (SUBARRAY C_source (15,10))) in
let c_mfsm_stateA =
    ((C_mfsm_rst) => CMI |
    ((C_mfsm_state = CMI) =>
        (C_mfsm_D ∧ ~C_mfsm_crqt_ ∧ ~c_busy ∧ ~C_mfsm_invalid) => CMR | CMI |
    ((C_mfsm_state = CMR) => (C_mfsm_D ∧ c_grant ∧ C_mfsm_hold_) => CMA3 | CMR |
    ((C_mfsm_state = CMA3) => ((C_mfsm_D) => CMA1 | CMA3) |
    ((C_mfsm_state = CMA1) =>
        (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA0 |
        (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA1 |
    ((C_mfsm_state = CMA0) =>
        (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA2 |
        (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA0 |
    ((C_mfsm_state = CMA2) =>
        (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD1 |
        (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA2 |
    ((C_mfsm_state = CMD1) =>
        (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD0 |
        (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD1 |
    ((C_mfsm_state = CMD0) =>
        (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ C_last_in_) => CMD1 |
        (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_last_in_) => CMW |
        (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD0 |
    ((C_mfsm_state = CMW) =>
        (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT |
        (C_mfsm_D ∧ (C_mfsm_ss = ^SACK) ∧ C_lock_in_) => CMI |
        (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_lock_in_ ∧ ~C_mfsm_crqt_) => CMA3 | CMW |
    ((~C_last_in_) => CMI | CMABT)))))))))) in
let c_sfsm_stateA =
    ((C_sfsm_rst) => CSI |
        (C_sfsm_state = CSI) =>
           ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ c_addressed) => CSA1 | CSI) |
    (C_sfsm_state = CSL) =>
        ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ c_addressed) => CSA1 |
        (C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ ~c_addressed) => CSI |
        (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSL) |
    (C_sfsm_state = CSA1) =>
        ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSA0 |
        (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA1) |
    (C_sfsm_state = CSA0) =>
        ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
        (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ C_sfsm_hlda_) => CSA0W |
        (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0) |
    (C_sfsm_state = CSA0W) =>
        ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
        (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0W) |
    (C_sfsm_state = CSALE) =>
        ((C_sfsm_D ∧ c_write ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
        (C_sfsm_D ∧ ~c_write ∧ (C_sfsm_ms = ^MRDY)) => CSRR |
        (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSALE) |
    (C_sfsm_state = CSRR) =>
        ((C_sfsm_D ∧ ~(C_sfsm_ms = ^MABORT)) => CSD1 |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSRR) |
        (C_sfsm_state = CSD1) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD0 |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD1) |
        (C_sfsm_state = CSD0) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MEND)) => CSACK |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD0) |
        (C_sfsm_state = CSACK) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSL |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MWAIT)) => CSI |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSACK) |
        (C_sfsm_D) => CSI | CSABT) in

let c_efsm_stateA =
        ((C_efsm_rst) => CEI |
         (C_efsm_state = CEI) => ((~C_efsm_cale_) => CEE | CEI) |
         ((~C_efsm_last_ ∧ ~C_efsm_srdy_) ∨ ~C_efsm_male_ ∨ ~C_efsm_rale_) => CEI | CEE) in
let c_srdy_en = ((c_efsm_stateA = CEE) ∨ (C_efsm_state = CEE)) in
let cout_sel0 = (ALTER ARBN (0) (((c_sfsm_stateA = CSD1) ∨ (c_sfsm_stateA = CSD0)) =>
                                        (c_sfsm_stateA = CSD1) |
                                        (c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA1)
                                                                ∨ (c_mfsm_stateA = CMD1))) in
let cout_sel10 = (ALTER cout_sel0 (1) (((c_sfsm_stateA = CSD1) ∨ (c_sfsm_stateA = CSD0)) =>
                                        F |
                                        (c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA2))) in
let c_cout_sel = cout_sel10 in
let new_C_wr = ((~I_cale_) => (ELEMENT I_ad_in (27)) | C_wr) in
let new_C_sizewrbe = ((Rst) => (WORDN 0) |
                                (((c_sfsm_stateA = CSA0) ∧ C_clkA) => (SUBARRAY C_data_in (31,22)) | C_sizewrbe)) in
let c_new_write = (((~(c_mfsm_stateA = CMI)) ∧ (~(c_mfsm_stateA = CMR))) =>
                        new_C_wr | (ELEMENT new_C_sizewrbe (5))) in
let new_C_clkA = ClkD in
let new_C_last_in_ = ((Rst) => F |
                        (((c_mfsm_stateA = CMABT) ∨ (c_mfsm_stateA = CMD1) ∧ ClkD) => I_last_in_ |
                        C_last_in_)) in
let new_C_lock_in_ = ((Rst) => F |
                        ((c_mfsm_stateA = CMA1) => I_lock_ |
                        C_lock_in_)) in
let new_C_ss = (((~(c_mfsm_stateA = CMABT)) ∧ (~(c_mfsm_stateA = CMI))) => CB_ss_in | C_ss) in
let c_mend = (CB_ms_in = ^MEND) in
let c_mabort = (CB_ms_in = ^MABORT) in
let new_C_last_out_ =
        (((c_sfsm_stateA = CSA1) ∧ ~(ClkD ∧ (c_mend ∨ c_mabort))) => T |
         ((~(c_sfsm_stateA = CSA1) ∧ (ClkD ∧ (c_mend ∨ c_mabort))) => F |
         ((~(c_sfsm_stateA = CSA1) ∧ ~(ClkD ∧ (c_mend ∨ c_mabort))) => C_last_out_ | ARB))) in
let c_srdy = (CB_ss_in = ^SRDY) in
let c_dfsm_master = ((c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA2) ∨ (c_mfsm_stateA = CMA1)
                    ∨ (c_mfsm_stateA = CMA0) ∨ (c_mfsm_stateA = CMD1) ∨ (c_mfsm_stateA = CMD0)) in
let c_dfsm_cad_en = ~((c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA1) ∨ (c_mfsm_stateA = CMA0)
                        ∨ (c_mfsm_stateA = CMA2)
                        ∨ (c_new_write ∧ ((c_mfsm_stateA = CMD1) ∨ (c_mfsm_stateA = CMD0)))
                        ∨ (~c_new_write ∧ ((c_sfsm_stateA = CSD1) ∨ (c_sfsm_stateA = CSD0)))) in
let new_C_hold_ = (c_sfsm_stateA = CSI) in
let new_C_holdA_ = ((ClkD) => C_hold_ | C_holdA_) in
let new_C_cout_0_le_del = ((I_cale_) ∨ (I_srdy_in_ ∧ ~c_new_write)
                                ∨ ((c_mfsm_stateA = CMA0) ∧ c_srdy ∧ c_new_write ∧ ClkD)
                                ∨ ((c_mfsm_stateA = CMD0) ∧ c_new_write ∧ c_srdy ∧ ClkD)) in
let new_C_cin_2_le = (ClkD ∧ (((c_mfsm_stateA = CMD0) ∧ c_srdy ∧ ~c_new_write) ∨
                                    ((c_sfsm_stateA = CSA0)) ∨
                                    ((c_sfsm_stateA = CSD0) ∧ c_new_write))) in
let new_C_mrdy_del_ = ~((~c_new_write ∧ ClkD ∧ ((c_sfsm_stateA = CSALE) ∨ (c_sfsm_stateA = CSD1))) ∨
                                (~c_new_write ∧ C_clkA ∧ (c_sfsm_stateA = CSACK)) ∨
                                (c_new_write ∧ ClkD ∧ (c_sfsm_stateA = CSD0))) in
let new_C_iad_en_s_del = (((c_sfsm_stateA = CSALE) ∧ (~(C_sfsm_state = CSALE)))
                                ∨ ((c_sfsm_stateA = CSALE) ∧ c_new_write)
                                ∨ ((c_sfsm_stateA = CSD1) ∧ c_new_write ∧ (~(C_sfsm_state = CSRR)))
                                ∨ ((c_sfsm_stateA = CSD0) ∧ c_new_write) ∨
                                  ((c_sfsm_stateA = CSACK) ∧ c_new_write)) in
let new_C_iad_en_s_delA = ((ClkD) => C_iad_en_s_del | C_iad_en_s_delA) in
let new_C_wrdy = (c_srdy ∧ c_new_write ∧ (c_mfsm_stateA = CMD1) ∧ ClkD) in
let new_C_rrdy = (c_srdy ∧ ~c_new_write ∧ (c_mfsm_stateA = CMD0) ∧ ClkD) in
let c_pe = (Par_Det rep (CB_ad_in)) in
let c_mparity = ((c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA1) ∨ (c_mfsm_stateA = CMA0)
                ∨ (c_mfsm_stateA = CMA2) ∨ (c_mfsm_stateA = CMD1) ∨ (c_mfsm_stateA = CMD0)
                ∨ (C_mfsm_state = CMA1) ∨ (C_mfsm_state = CMA0) ∨ (C_mfsm_state = CMA2)
                ∨ (C_mfsm_state = CMD1)) in
let c_sparity = ((~(c_sfsm_stateA = CSI)) ∧ (~(c_sfsm_stateA = CSACK)) ∧ (~(c_sfsm_stateA = CSABT))) in
let c_pe_cnt = (ClkD ∧ ((~(c_mparity = c_sparity)) ∨ ((SUBARRAY CB_ss_in (1,0)) = (WORDN 0)))) in
let new_C_parity =
        (((ClkD ∧ c_pe ∧ c_pe_cnt) ∧ ~Reset_error) => T |
         ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ Reset_error) => F |
         ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ ~Reset_error) => C_parity | ARB))) in
let new_C_source =
        ((Rst) => (WORDN 0) |
         ((ClkD ∧ ((c_sfsm_stateA = CSI) ∨ (c_sfsm_stateA = CSL))) => Par_Dec rep (CB_ad_in) | C_source)) in
let data_in31_16 =
        (MALTER ARBN (31,16) ((Rst) => (WORDN 0) |
                                    ((ClkD ∧ (((c_mfsm_stateA = CMD1) ∧ c_srdy ∧ ~c_new_write) ∨
                                              ((c_sfsm_stateA = CSA1)) ∨
                                              ((c_sfsm_stateA = CSD1) ∧ c_new_write))) => Par_Dec rep (CB_ad_in) |
                                    (SUBARRAY C_data_in (31,16))))) in
let data_in31_0 =
        (MALTER data_in31_16 (15,0) ((Rst) => (WORDN 0) |
                                        ((new_C_cin_2_le) => Par_Dec rep (CB_ad_in) |
                                        (SUBARRAY C_data_in (15,0))))) in
let new_C_data_in = data_in31_0 in
let new_C_iad_out = ((C_cin_2_le) => C_data_in | C_iad_out) in
let new_C_iad_in = ((new_C_cout_0_le_del) => I_ad_in | C_iad_in) in
let new_C_a1a0 =
        (((c_dfsm_master ∧ C_cout_0_le_del) ∨
         (~c_dfsm_master ∧ C_clkA ∧ (c_sfsm_stateA = CSD1))) => C_iad_in | C_a1a0) in
let new_C_a3a2 = ((c_mfsm_stateA = CMR) => Ccr | C_a3a2) in
let new_C_mfsm_state = c_mfsm_stateA in
let new_C_mfsm_D = ClkD in
let new_C_mfsm_rst = Rst in
let new_C_mfsm_crqt_ = I_crqt_ in
let new_C_mfsm_hold_ = new_C_holdA_ in
```

```
    let new_C_mfsm_ss = CB_ss_in in
    let new_C_mfsm_invalid = Piu_invalid in
    let new_C_sfsm_state = c_sfsm_stateA in
    let new_C_sfsm_D = ClkD in
    let new_C_sfsm_rst = Rst in
    let new_C_sfsm_hlda_ = I_hlda_ in
    let new_C_sfsm_ms = CB_ms_in in
    let new_C_efsm_cale_ = I_cale_ in
    let new_C_efsm_last_ = I_last_in_ in
    let new_C_efsm_male_ = I_male_in_ in
    let new_C_efsm_rale_ = I_rale_in_ in
    let new_C_efsm_srdy_ = I_srdy_in_ in
    let new_C_efsm_rst = Rst in

    (C_mfsm_state, C_mfsm_D, C_mfsm_rst, C_mfsm_crqt_, C_mfsm_hold_, C_mfsm_ss, C_mfsm_invalid,
     C_sfsm_state, C_sfsm_D, C_sfsm_rst, C_sfsm_hlda_, C_sfsm_ms, C_efsm_state, C_efsm_cale_, C_efsm_last_,
     C_efsm_male_, C_efsm_rale_, C_efsm_srdy_, C_efsm_rst, C_wr, C_sizewrbe, C_clkA, C_last_in_, C_lock_in_,
     C_ss, C_last_out_, C_hold_, C_holdA_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del,
     C_iad_en_s_delA, C_wrdy, C_rrdy, C_parity, C_source, C_data_in, C_iad_out, C_iad_in, C_a1a0, C_a3a2)"
    );;

%------------------------------------------------------------------------------------------
    Output definition for EXEC instruction.
------------------------------------------------------------------------------------------%

let cEXEC_out_def = new_definition
    ('cEXEC_out',
    "! (rep:^rep_ty)
      (C_mfsm_state:cmfsm_ty) (C_sfsm_state:csfsm_ty) (C_efsm_state:cefsm_ty)
      (C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss C_source C_data_in C_iad_out C_iad_in C_a1a0 C_a3a2 :wordn)
      (C_mfsm_D C_mfsm_rst C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_invalid C_sfsm_D C_sfsm_rst C_sfsm_hlda_
       C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
       C_wr C_clkA C_last_in_ C_lock_in_ C_last_out_ C_hold_ C_holdA_ C_cout_0_le_del C_cin_2_le
       C_mrdy_del_ C_iad_en_s_del C_iad_en_s_delA C_wrdy C_rrdy C_parity :bool)
      (I_ad_in I_be_in_ CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID Ccr :wordn)
      (I_mrdy_in_ I_rale_in_ I_male_in_ I_last_in_ I_srdy_in_ I_lock_ I_cale_ I_hlda_ I_crqt_
       Rst ClkA ClkB ClkD Pmm_failure Piu_invalid Reset_error :bool) .
    cEXEC_out rep
              (C_mfsm_state, C_mfsm_D, C_mfsm_rst, C_mfsm_crqt_, C_mfsm_hold_, C_mfsm_ss, C_mfsm_invalid,
               C_sfsm_state, C_sfsm_D, C_sfsm_rst, C_sfsm_hlda_, C_sfsm_ms,
               C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_, C_efsm_rst,
               C_wr, C_sizewrbe, C_clkA, C_last_in_, C_lock_in_, C_ss, C_last_out_,
               C_hold_, C_holdA_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_iad_en_s_delA,
               C_wrdy, C_rrdy, C_parity, C_source, C_data_in, C_iad_out, C_iad_in, C_a1a0,C_a3a2)
              (I_ad_in, I_be_in_, I_mrdy_in_, I_rale_in_, I_male_in_, I_last_in_, I_srdy_in_,
               I_lock_, I_cale_, I_hlda_, I_crqt_, CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in,
               Rst, ClkA, ClkB, ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr, Reset_error) =

    let c_write = (((~(C_mfsm_state = CMI)) ∧ (~(C_mfsm_state = CMR))) => C_wr | (ELEMENT C_sizewrbe (5))) in
    let c_busy = (~((SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
    let c_grant = ((((SUBARRAY Id (1,0)) = (WORDN 0)) ∧ ~(ELEMENT CB_rqt_in_ (0)))
                 ∨ (((SUBARRAY Id (1,0)) = (WORDN 1)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                       ∧ (ELEMENT CB_rqt_in_ (1)))
                 ∨ (((SUBARRAY Id (1,0)) = (WORDN 2)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                  ∧ (ELEMENT CB_rqt_in_ (1))
                                                  ∧ (ELEMENT CB_rqt_in_ (2)))
              ∨ (((SUBARRAY Id (1,0)) = (WORDN 3)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                  ∧ (ELEMENT CB_rqt_in_ (1))
                                                  ∧ (ELEMENT CB_rqt_in_ (2))
                                                  ∧ (ELEMENT CB_rqt_in_ (3)))) in
let c_addressed = (Id = (SUBARRAY C_source (15,10))) in
let c_mfsm_stateA =
     ((C_mfsm_rst) => CMI |
     ((C_mfsm_state = CMI) =>
          (C_mfsm_D ∧ ~C_mfsm_crqt_ ∧ ~c_busy ∧ ~C_mfsm_invalid) => CMR | CMI |
     ((C_mfsm_state = CMR) => (C_mfsm_D ∧ c_grant ∧ C_mfsm_hold_) => CMA3 | CMR |
     ((C_mfsm_state = CMA3) => ((C_mfsm_D) => CMA1 | CMA3) |
     ((C_mfsm_state = CMA1) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA0 |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA1 |
     ((C_mfsm_state = CMA0) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA2 |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA0 |
     ((C_mfsm_state = CMA2) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD1 |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA2 |
     ((C_mfsm_state = CMD1) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD0 |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD1 |
     ((C_mfsm_state = CMD0) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ C_last_in_) => CMD1 |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_last_in_) => CMW |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD0 |
     ((C_mfsm_state = CMW) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SACK) ∧ C_lock_in_) => CMI |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_lock_in_ ∧ ~C_mfsm_crqt_) => CMA3 | CMW |
     ((~C_last_in_) => CMI | CMABT)))))))))) in
let c_sfsm_stateA =
     ((C_sfsm_rst) => CSI |
          (C_sfsm_state = CSI) =>
             ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ c_addressed) => CSA1 | CSI) |
     (C_sfsm_state = CSL) =>
          ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ c_addressed) => CSA1 |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ ~c_addressed) => CSI |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSL) |
     (C_sfsm_state = CSA1) =>
          ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSA0 |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA1) |
     (C_sfsm_state = CSA0) =>
          ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ C_sfsm_hlda_) => CSA0W |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0) |
     (C_sfsm_state = CSA0W) =>
          ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
          (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0W) |
     (C_sfsm_state = CSALE) =>
          ((C_sfsm_D ∧ c_write ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
            (C_sfsm_D ∧ ~c_write ∧ (C_sfsm_ms = ^MRDY)) => CSRR |
            (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSALE) |
        (C_sfsm_state = CSRR) =>
            ((C_sfsm_D ∧ ~(C_sfsm_ms = ^MABORT)) => CSD1 |
            (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSRR) |
        (C_sfsm_state = CSD1) =>
            ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD0 |
            (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD1) |
        (C_sfsm_state = CSD0) =>
            ((C_sfsm_D ∧ (C_sfsm_ms = ^MEND)) => CSACK |
            (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
            (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD0) |
        (C_sfsm_state = CSACK) =>
            ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSL |
            (C_sfsm_D ∧ (C_sfsm_ms = ^MWAIT)) => CSI |
            (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSACK) |
        (C_sfsm_D) => CSI | CSABT) in
let c_efsm_stateA =
        ((C_efsm_rst) => CEI |
        (C_efsm_state = CEI) => ((~C_efsm_cale_) => CEE | CEI) |
        ((~C_efsm_last_ ∧ ~C_efsm_srdy_) ∨ ~C_efsm_male_ ∨ ~C_efsm_rale_) => CEI | CEE) in
let c_srdy_en = ((c_efsm_stateA = CEE) ∨ (C_efsm_state = CEE)) in
let cout_sel0 = (ALTER ARBN (0) (((c_sfsm_stateA = CSD1) ∨ (c_sfsm_stateA = CSD0)) =>
                                    (c_sfsm_stateA = CSD1) |
                                    (c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA1)
                                                        ∨ (c_mfsm_stateA = CMD1))) in
let cout_sel10 = (ALTER cout_sel0 (1) (((c_sfsm_stateA = CSD1) ∨ (c_sfsm_stateA = CSD0)) =>
                                        F |
                                        (c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA2))) in
let c_cout_sel = cout_sel10 in
let new_C_wr = ((~I_cale_) => (ELEMENT I_ad_in (27)) | C_wr) in
let new_C_sizewrbe = ((Rst) => (WORDN 0) |
                        (((c_sfsm_stateA = CSA0) ∧ C_clkA) => (SUBARRAY C_data_in (31,22)) | C_sizewrbe)) in
let c_new_write = (((~(c_mfsm_stateA = CMI)) ∧ (~(c_mfsm_stateA = CMR))) =>
                        new_C_wr | (ELEMENT new_C_sizewrbe (5))) in
let new_C_clkA = ClkD in
let new_C_last_in_ = ((Rst) => F |
                        (((c_mfsm_stateA = CMABT) ∨ (c_mfsm_stateA = CMD1) ∧ ClkD) => I_last_in_ |
                        C_last_in_)) in
let new_C_lock_in_ = ((Rst) => F |
                        ((c_mfsm_stateA = CMA1) => I_lock_ |
                        C_lock_in_)) in
let new_C_ss = (((~(c_mfsm_stateA = CMABT)) ∧ (~(c_mfsm_stateA = CMI))) => CB_ss_in | C_ss) in
let c_mend = (CB_ms_in = ^MEND) in
let c_mabort = (CB_ms_in = ^MABORT) in
let new_C_last_out_ =
        (((c_sfsm_stateA = CSA1) ∧ ~(ClkD ∧ (c_mend ∨ c_mabort))) => T |
        ((~(c_sfsm_stateA = CSA1) ∧ (ClkD ∧ (c_mend ∨ c_mabort))) => F |
        ((~(c_sfsm_stateA = CSA1) ∧ ~(ClkD ∧ (c_mend ∨ c_mabort))) => C_last_out_ | ARB))) in
let c_srdy = (CB_ss_in = ^SRDY) in
let c_dfsm_master = ((c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA2) ∨ (c_mfsm_stateA = CMA1)
                    ∨ (c_mfsm_stateA = CMA0) ∨ (c_mfsm_stateA = CMD1) ∨ (c_mfsm_stateA = CMD0)) in
let c_dfsm_cad_en = ~((c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA1) ∨ (c_mfsm_stateA = CMA0)
                    ∨ (c_mfsm_stateA = CMA2)
```

```
                        ∨ (c_new_write ∧ ((c_mfsm_stateA = CMD1) ∨ (c_mfsm_stateA = CMD0)))
                        ∨ (~c_new_write ∧ ((c_sfsm_stateA = CSD1) ∨ (c_sfsm_stateA = CSD0)))) in
let new_C_hold_ = (c_sfsm_stateA = CSI) in
let new_C_holdA_ = ((ClkD) => C_hold_ | C_holdA_) in
let new_C_cout_0_le_del = ((I_cale_) ∨ (I_srdy_in_ ∧ ~c_new_write)
                           ∨ ((c_mfsm_stateA = CMA0) ∧ c_srdy ∧ c_new_write ∧ ClkD)
                           ∨ ((c_mfsm_stateA = CMD0) ∧ c_new_write ∧ c_srdy ∧ ClkD)) in
let new_C_cin_2_le = (ClkD ∧ (((c_mfsm_stateA = CMD0) ∧ c_srdy ∧ ~c_new_write) ∨
                              ((c_sfsm_stateA = CSA0)) ∨
                              ((c_sfsm_stateA = CSD0) ∧ c_new_write))) in
let new_C_mrdy_del_ = ~((~c_new_write ∧ ClkD ∧ ((c_sfsm_stateA = CSALE) ∨ (c_sfsm_stateA = CSD1))) ∨
                        (~c_new_write ∧ C_clkA ∧ (c_sfsm_stateA = CSACK)) ∨
                        (c_new_write ∧ ClkD ∧ (c_sfsm_stateA = CSD0))) in
let new_C_iad_en_s_del = (((c_sfsm_stateA = CSALE) ∧ (~(C_sfsm_state = CSALE)))
                         ∨ ((c_sfsm_stateA = CSALE) ∧ c_new_write)
                         ∨ ((c_sfsm_stateA = CSD1) ∧ c_new_write ∧ (~(C_sfsm_state = CSRR)))
                         ∨ ((c_sfsm_stateA = CSD0) ∧ c_new_write) ∨
                          ((c_sfsm_stateA = CSACK) ∧ c_new_write)) in
let new_C_iad_en_s_delA = ((ClkD) => C_iad_en_s_del | C_iad_en_s_delA) in
let new_C_wrdy = (c_srdy ∧ c_new_write ∧ (c_mfsm_stateA = CMD1) ∧ ClkD) in
let new_C_rrdy = (c_srdy ∧ ~c_new_write ∧ (c_mfsm_stateA = CMD0) ∧ ClkD) in
let c_pe = (Par_Det rep (CB_ad_in)) in
let c_mparity = ((c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA1) ∨ (c_mfsm_stateA = CMA0)
               ∨ (c_mfsm_stateA = CMA2) ∨ (c_mfsm_stateA = CMD1) ∨ (c_mfsm_stateA = CMD0)
               ∨ (C_mfsm_state = CMA1) ∨ (C_mfsm_state = CMA0) ∨ (C_mfsm_state = CMA2)
               ∨ (C_mfsm_state = CMD1)) in
let c_sparity = ((~(c_sfsm_stateA = CSI)) ∧ (~(c_sfsm_stateA = CSACK)) ∧ (~(c_sfsm_stateA = CSABT))) in
let c_pe_cnt = (ClkD ∧ ((~(c_mparity = c_sparity)) ∨ ((SUBARRAY CB_ss_in (1,0)) = (WORDN 0)))) in
let new_C_parity =
    (((ClkD ∧ c_pe ∧ c_pe_cnt) ∧ ~Reset_error) => T |
     ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ Reset_error) => F |
     ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ ~Reset_error) => C_parity | ARB))) in
let new_C_source =
    ((Rst) => (WORDN 0) |
     ((ClkD ∧ ((c_sfsm_stateA = CSI) ∨ (c_sfsm_stateA = CSL))) => Par_Dec rep (CB_ad_in) | C_source)) in
let data_in31_16 =
    (MALTER ARBN (31,16) ((Rst) => (WORDN 0) |
                              ((ClkD ∧ (((c_mfsm_stateA = CMD1) ∧ c_srdy ∧ ~c_new_write) ∨
                                        ((c_sfsm_stateA = CSA1)) ∨
                                        ((c_sfsm_stateA = CSD1) ∧ c_new_write))) => Par_Dec rep (CB_ad_in) |
                              (SUBARRAY C_data_in (31,16))))) in
let data_in31_0 =
    (MALTER data_in31_16 (15,0) ((Rst) => (WORDN 0) |
                                     ((new_C_cin_2_le) => Par_Dec rep (CB_ad_in) |
                                     (SUBARRAY C_data_in (15,0))))) in
let new_C_data_in = data_in31_0 in
let new_C_iad_out = ((C_cin_2_le) => C_data_in | C_iad_out) in
let new_C_iad_in = ((new_C_cout_0_le_del) => I_ad_in | C_iad_in) in
let new_C_a1a0 =
    (((c_dfsm_master ∧ C_cout_0_le_del) ∨
     (~c_dfsm_master ∧ C_clkA ∧ (c_sfsm_stateA = CSD1))) => C_iad_in | C_a1a0) in
let new_C_a3a2 = ((c_mfsm_stateA = CMR) => Ccr | C_a3a2) in
let new_C_mfsm_state = c_mfsm_stateA in
let new_C_mfsm_D = ClkD in
let new_C_mfsm_rst = Rst in
let new_C_mfsm_crqt_ = I_crqt_ in
let new_C_mfsm_hold_ = new_C_holdA_ in
let new_C_mfsm_ss = CB_ss_in in
let new_C_mfsm_invalid = Piu_invalid in
let new_C_sfsm_state = c_sfsm_stateA in
let new_C_sfsm_D = ClkD in
let new_C_sfsm_rst = Rst in
let new_C_sfsm_hlda_ = I_hlda_ in
let new_C_sfsm_ms = CB_ms_in in
let new_C_efsm_cale_ = I_cale_ in
let new_C_efsm_last_ = I_last_in_ in
let new_C_efsm_male_ = I_male_in_ in
let new_C_efsm_rale_ = I_rale_in_ in
let new_C_efsm_srdy_ = I_srdy_in_ in
let new_C_efsm_rst = Rst in

let I_cgnt_ = ~(c_mfsm_stateA = CMA3) in
let I_mrdy_out_ = ((~I_hlda_) => C_mrdy_del_ | ARB) in
let I_hold_ = new_C_holdA_ in
let I_rale_out_ =
      ((~I_hlda_) =>
         ~((c_sfsm_stateA = CSALE) ∧ ((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3)) ∧ C_clkA) | ARB) in
let I_male_out_ =
      ((~I_hlda_) =>
         ~((c_sfsm_stateA = CSALE) ∧ (~((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3))) ∧ C_clkA) | ARB) in
let I_last_out_ = ((~I_hlda_) => C_last_out_ | ARB) in
let I_srdy_out_ = ((~I_cale_ ∨ c_srdy_en) => ~(C_wrdy ∨ C_rrdy ∨ (c_mfsm_stateA = CMABT)) | ARB) in
let I_be_out_ = ((~I_hlda_) => (SUBARRAY new_C_sizewrbe (9,6)) | ARBN) in
let I_ad_out = ((new_C_iad_en_s_delA
               ∨ ((c_mfsm_stateA = CMD1) ∧ ~c_new_write ∧ c_srdy_en)
               ∨ ((c_mfsm_stateA = CMD0) ∧ ~c_new_write ∧ c_srdy_en)
               ∨ ((c_mfsm_stateA = CMW) ∧ (C_mfsm_state = CMD0) ∧ ~c_new_write ∧ c_srdy_en)
               ∨ ((c_sfsm_stateA = CSALE) ∧ (~(C_sfsm_state = CSALE)))
               ∨ ((c_sfsm_stateA = CSALE) ∧ c_new_write)
               ∨ ((c_sfsm_stateA = CSD1) ∧ c_new_write ∧ (~(C_sfsm_state = CSRR)))
               ∨ ((c_sfsm_stateA = CSD0) ∧ c_new_write)
               ∨ ((c_sfsm_stateA = CSACK) ∧ c_new_write)) => new_C_iad_out | ARBN) in
let CB_rqt_out_ = ~(~(c_mfsm_stateA = CMI)) in
let ms0 = (ALTER ARBN (0) (((c_mfsm_stateA = CMD0) ∧ ~C_last_in_) ∨
                           ((c_mfsm_stateA = CMW) ∧ C_lock_in_) ∨
                           (c_mfsm_stateA = CMABT))) in
let ms10 = (ALTER ms0 (1) (((c_mfsm_stateA = CMA1) ∨ (c_mfsm_stateA = CMA0) ∨
                          (c_mfsm_stateA = CMA2) ∨ (c_mfsm_stateA = CMD1) ∨
                          ((c_mfsm_stateA = CMD0) ∧ C_last_in_) ∨ (c_mfsm_stateA = CMW) ∨
                          (c_mfsm_stateA = CMABT)))) in
let ms210 = (ALTER ms10 (2) (((c_mfsm_stateA = CMA3) ∨ (c_mfsm_stateA = CMA1) ∨
                             (c_mfsm_stateA = CMA0) ∨ (c_mfsm_stateA = CMA2) ∨
                             (c_mfsm_stateA = CMD1) ∨ (c_mfsm_stateA = CMD0) ∨
                             (c_mfsm_stateA = CMW) ∨ (c_mfsm_stateA = CMABT)) ∧ ~Pmm_failure ∧ ~Piu_invalid))
in
let CB_ms_out = (((~(c_mfsm_stateA = CMI)) ∧ (~(c_mfsm_stateA = CMR))) => ms210 | ARBN) in
let ss0 = (ALTER ARBN (0) ((c_sfsm_stateA = CSA0W) ∨
                          ((c_sfsm_stateA = CSALE) ∧ ~c_new_write) ∨
```

```
                                    (c_sfsm_stateA = CSACK))) in
   let ss10 = (ALTER ss0 (1) ~(c_sfsm_stateA = CSACK)) in
   let ss210 = (ALTER ss10 (2) (~Pmm_failure ∧ ~Piu_invalid)) in
   let CB_ss_out = (((~(c_sfsm_stateA = CSI)) ∧ (~(c_sfsm_stateA = CSABT))) => ss210 | ARBN) in
   let CB_ad_out = ((c_dfsm_cad_en) =>
                                ((c_cout_sel = (WORDN 0)) => Par_Enc rep (SUBARRAY new_C_a1a0 (15,0)) |
                                ((c_cout_sel = (WORDN 1)) => Par_Enc rep (SUBARRAY new_C_a1a0 (31,16)) |
                                ((c_cout_sel = (WORDN 2)) => Par_Enc rep (SUBARRAY new_C_a3a2 (15,0)) |
                                Par_Enc rep (SUBARRAY new_C_a3a2 (31,16))))) | ARBN) in
   let C_ss_out = new_C_ss in
   let Disable_writes = ((~(c_sfsm_stateA = CSI)) ∧ (~(c_sfsm_stateA = CSL)) ∧
                              ~((ChannelID = (WORDN 0)) ∧ (ELEMENT C_source (6))) ∧
                              ~((ChannelID = (WORDN 1)) ∧ (ELEMENT C_source (7))) ∧
                              ~((ChannelID = (WORDN 2)) ∧ (ELEMENT C_source (8))) ∧
                              ~((ChannelID = (WORDN 3)) ∧ (ELEMENT C_source (9)))) in
   let CB_parity = new_C_parity in

   (I_cgnt_, I_mrdy_out_, I_hold_, I_rale_out_, I_male_out_, I_last_out_, I_srdy_out_, I_ad_out, I_be_out_,
    CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out, C_ss_out, Disable_writes, CB_parity)"
  );;

close_theory();;
```

D.5 SU_Cont Specification

```
%----------------------------------------------------------------------------------------------

    File:        s_clock1.ml

    Author:      (c) D.A. Fura 1992

    Date:        31 March 1992

    This file contains the ml source for the clock-level specification of the startup controller of the
    FTEP PIU, an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.
    The bulk of this code was translated from an M-language simulation program using a translator written
    by P.J. Windley at the University of Idaho.

----------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/']);;

system 'rm s_clock1.th';;

new_theory 's_clock1';;

map new_parent ['saux_def';'aux_def';'array_def';'wordn_def'];;

let sc_state_ty = ":(sfsm_ty#bool#bool#bool#bool#bool#bool#wordn#wordn#
                    bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let sc_state = "((S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
                 S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
                 S_cpu_hist, S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_piu_fail)
                :^sc_state_ty)";;

let sc_env_ty = ":(bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let sc_env = "((ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_)
              :^sc_env_ty)";;

let sc_out_ty = ":(wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let sc_out = "((S_state, Reset_cport, Disable_int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist,
              Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail)
              :^sc_out_ty)";;

%----------------------------------------------------------------------------------------------
    Next-state definition for EXEC instruction.
----------------------------------------------------------------------------------------------%

let sEXEC_inst_def = new_definition
 ('sEXEC_inst',
    "! (S_fsm_state :sfsm_ty)
       (S_soft_cnt S_delay :wordn)
       (S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass S_soft_shot_del S_bad_cpu0
        S_bad_cpu1 S_reset_cpu0 S_reset_cpu1 S_cpu_hist S_pmm_fail S_cpu0_fail S_cpu1_fail
        S_piu_fail :bool)
       (ClkA ClkB Rst Bypass Test Gcrh Gcrl Failure0_ Failure1_ :bool) .
```

```
sEXEC_inst (S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
            S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
            S_cpu_hist, S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_piu_fail)
           (ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_) =

let new_S_fsm_state =
      ((S_fsm_rst) => SSTART |
       ((S_fsm_state = SSTART) => SRA |
       ((S_fsm_state = SRA) => ((S_fsm_delay6) => ((S_fsm_bypass) => SO | SPF) | SRA) |
       ((S_fsm_state = SPF) => SC0I |
       ((S_fsm_state = SC0I) => ((S_fsm_delay17) => SC0F | SC0I) |
       ((S_fsm_state = SC0F) => ST |
       ((S_fsm_state = ST) => SC1I |
       ((S_fsm_state = SC1I) => ((S_fsm_delay17) => SC1F | SC1I) |
       ((S_fsm_state = SC1F) => SS |
       ((S_fsm_state = SS) => ((S_fsm_bothbad) => SSTOP | SCS) |
       ((S_fsm_state = SSTOP) => SSTOP |
       ((S_fsm_state = SCS) => ((S_fsm_delay6) => SN | SCS) |
       ((S_fsm_state = SN) => ((S_fsm_delay17) => SO | SN) |
       ((S_fsm_state = SO) => SO | S_ILL)))))))))))))) in
let s_fsm_sn = (new_S_fsm_state = SN) in
let s_fsm_so = (new_S_fsm_state = SO) in
let s_fsm_srcp = (((~(new_S_fsm_state = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let s_fsm_sdi = (((~(new_S_fsm_state = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let s_fsm_srp = ((new_S_fsm_state = SSTART) ∨ (new_S_fsm_state = SRA)
            ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
            ∨ (new_S_fsm_state = SC1F) ∨ (new_S_fsm_state = SS) ∨ (new_S_fsm_state = SCS)) in
let s_fsm_src0 = ((~(new_S_fsm_state = SPF)) ∧ (~(new_S_fsm_state = SC0I))) in
let s_fsm_src1 = ((~(new_S_fsm_state = ST)) ∧ (~(new_S_fsm_state = SC1I))) in
let s_fsm_spf = ((S_fsm_state = SRA) ∧ S_fsm_delay6 ∧ ~S_fsm_rst) in
let s_fsm_sc0f = (new_S_fsm_state = SC0F) in
let s_fsm_sc1f = (new_S_fsm_state = SC1F) in
let s_fsm_spmf = (new_S_fsm_state = SO) in
let s_fsm_sb = (new_S_fsm_state = SSTART) in
let s_fsm_src = ((new_S_fsm_state = SSTART) ∨ ((S_fsm_state = SRA) ∧ S_fsm_delay6)
            ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST) ∨ (new_S_fsm_state = SC1F)
            ∨ (new_S_fsm_state = SS) ∨ ((S_fsm_state = SCS) ∧ S_fsm_delay6)) in
let s_fsm_sec = (((~(new_S_fsm_state = SSTOP)) ∧ (~(new_S_fsm_state = SO))) ∨ (S_fsm_state = SN)) in
let s_fsm_srs = (((S_fsm_state = SPF) ∧ ~S_fsm_rst) ∨ ((S_fsm_state = ST) ∧ ~S_fsm_rst)) in
let s_fsm_scs = (new_S_fsm_state = SCS) in
let new_S_soft_shot_del = (~Gcrh ∧ Gcrl) in
let s_soft_cnt_out =
      ((s_fsm_srs) =>
          ((Gcrl ∧ ~Gcrh ∧ ~S_soft_shot_del) => (WORDN 1) | (WORDN 0)) |
          ((Gcrl ∧ ~Gcrh ∧ ~S_soft_shot_del) => (INCN 2 S_soft_cnt) | S_soft_cnt)) in
let new_S_soft_cnt = ((~Gcrh ∧ ~Gcrl) => (WORDN 0) | s_soft_cnt_out) in
let s_delay_out =
      ((s_fsm_src ∨ (s_fsm_scs ∧ (ELEMENT S_delay (6)))) =>
          ((s_fsm_sec) => (WORDN 1) | (WORDN 0)) |
          ((s_fsm_sec) => (INCN 17 S_delay) | S_delay)) in
let new_S_delay = s_delay_out in
let s_cpu0_ok = (s_fsm_sc0f ∧ Failure0_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu1_ok = (s_fsm_sc1f ∧ Failure1_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let new_S_pmm_fail =
        ((s_fsm_sb ∧ ~s_fsm_spmf) => T |
         ((~s_fsm_sb ∧ s_fsm_spmf) => F |
         ((~s_fsm_sb ∧ ~s_fsm_spmf) => S_pmm_fail | ARB))) in
  let new_S_cpu0_fail =
        ((s_fsm_sb ∧ ~(s_cpu0_ok V Bypass)) => T |
         ((~s_fsm_sb ∧ (s_cpu0_ok V Bypass)) => F |
         ((~s_fsm_sb ∧ ~(s_cpu0_ok V Bypass)) => S_cpu0_fail | ARB))) in
  let new_S_cpu1_fail =
        ((s_fsm_sb ∧ ~(s_cpu1_ok V Bypass)) => T |
         ((~s_fsm_sb ∧ (s_cpu1_ok V Bypass)) => F |
         ((~s_fsm_sb ∧ ~(s_cpu1_ok V Bypass)) => S_cpu1_fail | ARB))) in
  let new_S_piu_fail =
        ((s_fsm_sb ∧ ~(s_fsm_spf V Bypass)) => T |
         ((~s_fsm_sb ∧ (s_fsm_spf V Bypass)) => F |
         ((~s_fsm_sb ∧ ~(s_fsm_spf V Bypass)) => S_piu_fail | ARB))) in
  let s_cpu0_select = ((s_fsm_sn V s_fsm_so) ∧ ~S_cpu0_fail) in
  let s_cpu1_select = ((s_fsm_sn V s_fsm_so) ∧ S_cpu0_fail ∧ ~S_cpu1_fail) in
  let new_S_bad_cpu0 =
        ((s_fsm_sb ∧ ~s_cpu0_select) => T |
         ((~s_fsm_sb ∧ s_cpu0_select) => F |
         ((~s_fsm_sb ∧ ~s_cpu0_select) => S_bad_cpu0 | ARB))) in
  let new_S_bad_cpu1 =
        ((s_fsm_sb ∧ ~s_cpu1_select) => T |
         ((~s_fsm_sb ∧ s_cpu1_select) => F |
         ((~s_fsm_sb ∧ ~s_cpu1_select) => S_bad_cpu1 | ARB))) in
  let new_S_reset_cpu0 = (new_S_bad_cpu0 ∧ s_fsm_src0) in
  let new_S_reset_cpu1 = (new_S_bad_cpu1 ∧ s_fsm_src1) in
  let new_S_cpu_hist = (S_reset_cpu0 ∧ S_reset_cpu1 ∧ Bypass) in
  let new_S_fsm_rst = Rst in
  let new_S_fsm_delay6 = (ELEMENT s_delay_out (6)) in
  let new_S_fsm_delay17 = ((Test) => (ELEMENT s_delay_out (6)) | (ELEMENT s_delay_out (17))) in
  let new_S_fsm_bothbad = (new_S_cpu0_fail ∧ new_S_cpu1_fail) in
  let new_S_fsm_bypass = Bypass in

  (new_S_fsm_state, new_S_fsm_rst, new_S_fsm_delay6, new_S_fsm_delay17, new_S_fsm_bothbad,
   new_S_fsm_bypass, new_S_soft_shot_del, new_S_soft_cnt, new_S_delay, new_S_bad_cpu0, new_S_bad_cpu1,
   new_S_reset_cpu0, new_S_reset_cpu1, new_S_cpu_hist, new_S_pmm_fail, new_S_cpu0_fail, new_S_cpu1_fail,
   new_S_piu_fail)"
  );;

%-----------------------------------------------------------------------------------------------------
  Output definition for EXEC instruction.
-----------------------------------------------------------------------------------------------------%

let sEXEC_out_def = new_definition
 ('sEXEC_out',
    "! (S_fsm_state :sfsm_ty)
       (S_soft_cnt S_delay :wordn)
       (S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass S_soft_shot_del S_bad_cpu0
        S_bad_cpu1 S_reset_cpu0 S_reset_cpu1 S_cpu_hist S_pmm_fail S_cpu0_fail S_cpu1_fail
        S_piu_fail :bool)
       (ClkA ClkB Rst Bypass Test Gcrh Gcrl Failure0_ Failure1_ :bool) .
   sEXEC_out (S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
                  S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
            S_cpu_hist, S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_piu_fail)
           (ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_) =

let new_S_fsm_state =
      ((S_fsm_rst) => SSTART |
       ((S_fsm_state = SSTART) => SRA |
       ((S_fsm_state = SRA) => ((S_fsm_delay6) => ((S_fsm_bypass) => SO | SPF) | SRA) |
       ((S_fsm_state = SPF) => SC0I |
       ((S_fsm_state = SC0I) => ((S_fsm_delay17) => SC0F | SC0I) |
       ((S_fsm_state = SC0F) => ST |
       ((S_fsm_state = ST) => SC1I |
       ((S_fsm_state = SC1I) => ((S_fsm_delay17) => SC1F | SC1I) |
       ((S_fsm_state = SC1F) => SS |
       ((S_fsm_state = SS) => ((S_fsm_bothbad) => SSTOP | SCS) |
       ((S_fsm_state = SSTOP) => SSTOP |
       ((S_fsm_state = SCS) => ((S_fsm_delay6) => SN | SCS) |
       ((S_fsm_state = SN) => ((S_fsm_delay17) => SO | SN) |
       ((S_fsm_state = SO) => SO | S_ILL)))))))))))))) in
let s_fsm_sn = (new_S_fsm_state = SN) in
let s_fsm_so = (new_S_fsm_state = SO) in
let s_fsm_srcp = (((~(new_S_fsm_state = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let s_fsm_sdi = (((~(new_S_fsm_state = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let s_fsm_srp = ((new_S_fsm_state = SSTART) ∨ (new_S_fsm_state = SRA)
                 ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
                 ∨ (new_S_fsm_state = SC1F) ∨ (new_S_fsm_state = SS) ∨ (new_S_fsm_state = SCS)) in
let s_fsm_src0 = ((~(new_S_fsm_state = SPF)) ∧ (~(new_S_fsm_state = SC0I))) in
let s_fsm_src1 = ((~(new_S_fsm_state = ST)) ∧ (~(new_S_fsm_state = SC1I))) in
let s_fsm_spf = ((S_fsm_state = SRA) ∧ S_fsm_delay6 ∧ ~S_fsm_rst) in
let s_fsm_sc0f = (new_S_fsm_state = SC0F) in
let s_fsm_sc1f = (new_S_fsm_state = SC1F) in
let s_fsm_spmf = (new_S_fsm_state = SO) in
let s_fsm_sb = (new_S_fsm_state = SSTART) in
let s_fsm_src = ((new_S_fsm_state = SSTART) ∨ ((S_fsm_state = SRA) ∧ S_fsm_delay6)
                 ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST) ∨ (new_S_fsm_state = SC1F)
                 ∨ (new_S_fsm_state = SS) ∨ ((S_fsm_state = SCS) ∧ S_fsm_delay6)) in
let s_fsm_sec = (((~(new_S_fsm_state = SSTOP)) ∧ (~(new_S_fsm_state = SO))) ∨ (S_fsm_state = SN)) in
let s_fsm_srs = (((S_fsm_state = SPF) ∧ ~S_fsm_rst) ∨ ((S_fsm_state = ST) ∧ ~S_fsm_rst)) in
let s_fsm_scs = (new_S_fsm_state = SCS) in
let new_S_soft_shot_del = (~Gcrh ∧ Gcrl) in
let s_soft_cnt_out =
      ((s_fsm_srs) =>
         ((Gcrl ∧ ~Gcrh ∧ ~S_soft_shot_del) => (WORDN 1) | (WORDN 0)) |
         ((Gcrl ∧ ~Gcrh ∧ ~S_soft_shot_del) => (INCN 2 S_soft_cnt) | S_soft_cnt)) in
let new_S_soft_cnt = ((~Gcrh ∧ ~Gcrl) => (WORDN 0) | s_soft_cnt_out) in
let s_delay_out =
      ((s_fsm_src ∨ (s_fsm_scs ∧ (ELEMENT S_delay (6)))) =>
         ((s_fsm_sec) => (WORDN 1) | (WORDN 0)) |
         ((s_fsm_sec) => (INCN 17 S_delay) | S_delay)) in
let new_S_delay = s_delay_out in
let s_cpu0_ok = (s_fsm_sc0f ∧ Failure0_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu1_ok = (s_fsm_sc1f ∧ Failure1_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let new_S_pmm_fail =
      ((s_fsm_sb ∧ ~s_fsm_spmf) => T |
       ((~s_fsm_sb ∧ s_fsm_spmf) => F |
        ((~s_fsm_sb ∧ ~s_fsm_spmf) => S_pmm_fail | ARB))) in
let new_S_cpu0_fail =
       ((s_fsm_sb ∧ ~(s_cpu0_ok ∨ Bypass)) => T |
        ((~s_fsm_sb ∧ (s_cpu0_ok ∨ Bypass)) => F |
        ((~s_fsm_sb ∧ ~(s_cpu0_ok ∨ Bypass)) => S_cpu0_fail | ARB))) in
let new_S_cpu1_fail =
       ((s_fsm_sb ∧ ~(s_cpu1_ok ∨ Bypass)) => T |
        ((~s_fsm_sb ∧ (s_cpu1_ok ∨ Bypass)) => F |
        ((~s_fsm_sb ∧ ~(s_cpu1_ok ∨ Bypass)) => S_cpu1_fail | ARB))) in
let new_S_piu_fail =
       ((s_fsm_sb ∧ ~(s_fsm_spf ∨ Bypass)) => T |
        ((~s_fsm_sb ∧ (s_fsm_spf ∨ Bypass)) => F |
        ((~s_fsm_sb ∧ ~(s_fsm_spf ∨ Bypass)) => S_piu_fail | ARB))) in
let s_cpu0_select = ((s_fsm_sn ∨ s_fsm_so) ∧ ~S_cpu0_fail) in
let s_cpu1_select = ((s_fsm_sn ∨ s_fsm_so) ∧ S_cpu0_fail ∧ ~S_cpu1_fail) in
let new_S_bad_cpu0 =
       ((s_fsm_sb ∧ ~s_cpu0_select) => T |
        ((~s_fsm_sb ∧ s_cpu0_select) => F |
        ((~s_fsm_sb ∧ ~s_cpu0_select) => S_bad_cpu0 | ARB))) in
let new_S_bad_cpu1 =
       ((s_fsm_sb ∧ ~s_cpu1_select) => T |
        ((~s_fsm_sb ∧ s_cpu1_select) => F |
        ((~s_fsm_sb ∧ ~s_cpu1_select) => S_bad_cpu1 | ARB))) in
let new_S_reset_cpu0 = (new_S_bad_cpu0 ∧ s_fsm_src0) in
let new_S_reset_cpu1 = (new_S_bad_cpu1 ∧ s_fsm_src1) in
let new_S_cpu_hist = (S_reset_cpu0 ∧ S_reset_cpu1 ∧ Bypass) in
let new_S_fsm_rst = Rst in
let new_S_fsm_delay6 = (ELEMENT s_delay_out (6)) in
let new_S_fsm_delay17 = ((Test) => (ELEMENT s_delay_out (6)) | (ELEMENT s_delay_out (17))) in
let new_S_fsm_bothbad = (new_S_cpu0_fail ∧ new_S_cpu1_fail) in
let new_S_fsm_bypass = Bypass in
let ss0 = (ALTER ARBN (0) ((new_S_fsm_state = SS) ∨ (new_S_fsm_state = SSTOP)
                            ∨ (new_S_fsm_state = SCS) ∨ (new_S_fsm_state = SN)
                            ∨ (new_S_fsm_state = SO))) in
let ss1 = (ALTER ss0 (1) ((new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
                         ∨ (new_S_fsm_state = SC1I) ∨ (new_S_fsm_state = SC1F)
                         ∨ (new_S_fsm_state = SS) ∨ (new_S_fsm_state = SSTOP)
                         ∨ (new_S_fsm_state = SCS))) in
let ss2 = (ALTER ss1 (2) ((new_S_fsm_state = SPF) ∨ (new_S_fsm_state = SC0I)
                         ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
                         ∨ (new_S_fsm_state = SSTOP) ∨ (new_S_fsm_state = SO))) in
let ss3 = (ALTER ss2 (3) ((new_S_fsm_state = SRA) ∨ (new_S_fsm_state = SPF)
                         ∨ (new_S_fsm_state = ST) ∨ (new_S_fsm_state = SC1I)
                         ∨ (new_S_fsm_state = SCS) ∨ (new_S_fsm_state = SN)
                         ∨ (new_S_fsm_state = SO))) in
let S_state = ss3 in
let Reset_cport = s_fsm_srcp in
let Disable_int = (~(s_fsm_sn ∧ (ELEMENT s_delay_out (6))) ∧ s_fsm_sdi
                 ∧ ((Test) => ~(ELEMENT s_delay_out (5)) | ~(ELEMENT s_delay_out (16)))) in
let Reset_piu = s_fsm_srp in
let Reset_cpu0 = new_S_reset_cpu0 in
let Reset_cpu1 = new_S_reset_cpu1 in
let Cpu_hist = new_S_cpu_hist in
let Piu_fail = new_S_piu_fail in
```

```
    let Cpu0_fail = new_S_cpu0_fail in
    let Cpu1_fail = new_S_cpu1_fail in
    let Pmm_fail = new_S_pmm_fail in

    (S_state, Reset_cport, Disable_int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist, Piu_fail, Cpu0_fail,
     Cpu1_fail, Pmm_fail)"
   );;

close_theory();;
```

Appendix E ML Source for the PIU Block-Level Specification.

This appendix contains the HOL model for the PIU block-level structural specification.

```
%-------------------------------------------------------------------------------------------------------

    File:        piu_block.ml

    Author:      (c) D.A. Fura 1992

    Date:        31 March 1992

    This file contains the ml source for the block-level specification of the FTEP PIU, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center. At this level
    the blocks correspond to the four PIU ports and the startup controller.

-------------------------------------------------------------------------------------------------------%

set_search_path (search_path() @ [`/home/titan3/dfura/ftep/piu/hol/lib/`;
                                  `/home/titan3/dfura/ftep/piu/hol/pport/`;
                                  `/home/titan3/dfura/ftep/piu/hol/cport/`;
                                  `/home/titan3/dfura/ftep/piu/hol/mport/`;
                                  `/home/titan3/dfura/ftep/piu/hol/cport/`;
                                  `/home/titan3/dfura/ftep/piu/hol/sucont/`]);;

system `rm piu_block.th`;;

new_theory `piu_block`;;

loadf `abstract`;;

map new_parent [`aux_def`;`p_clock1`;`c_clock1`;`m_clock1`;`c_clock1`;`s_clock1`];;

let rep_ty = abstract_type `aux_def` `Andn`;;

let PIU_Block_SPEC = new_definition
   (`PIU_Block_SPEC`,
    "! (rep:^rep_ty)
       (P_fsm_state :pfsm_ty)
       (P_addr P_be_ P_size :wordn)
       (P_dest1 P_wr P_fsm_rst P_fsm_sack P_fsm_cgnt_ P_fsm_hold_ P_rqt P_down P_lock_
        P_lock_inh_ P_male_ P_rale_ :bool)
       (C_mfsm_state :cmfsm_ty) (C_sfsm_state :csfsm_ty) (C_efsm_state :cefsm_ty)
       (C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss C_source C_data_in C_iad_out C_iad_in C_a1a0 C_a3a2 :wordn)
       (C_mfsm_D C_mfsm_rst C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_invalid C_sfsm_D C_sfsm_rst C_sfsm_hlda_
        C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
        C_wr C_clkA C_last_in_ C_lock_in_ C_last_out_ C_hold_ C_holdA_ C_cout_0_le_del C_cin_2_le
        C_mrdy_del_ C_iad_en_s_del C_iad_en_s_delA C_wrdy C_rrdy C_parity :bool)
       (M_fsm_state :mfsm_ty)
       (M_count M_addr M_be M_rd_data M_detect :wordn)
       (M_fsm_male_ M_fsm_last_ M_fsm_mrdy_ M_fsm_rst M_se M_wr M_rdy M_wwdel M_parity :bool)
       (R_fsm_state :rfsm_ty)
       (R_ctr0_in R_ctr0 R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1 R_ctr1_new R_ctr1_out R_ctr2_in R_ctr2 R_ctr2_new
        R_ctr2_out R_ctr3_in R_ctr3 R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr R_ccr R_gcr R_sr
```

```
  R_reg_sel R_busA_latch :wordn)
 (R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden
  R_ctr1_mux_sel R_ctr1_irden R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden
  R_ctr3_mux_sel R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden R_sr_rden
  R_int0_dis R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del R_srdy_del_ :bool)
 (S_fsm_state :sfsm_ty)
 (S_soft_cnt S_delay :wordn)
 (S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass S_soft_shot_del S_bad_cpu0 S_bad_cpu1
  S_reset_cpu0 S_reset_cpu1 S_cpu_hist S_pmm_fail S_cpu0_fail S_cpu1_fail S_piu_fail :bool)

 (L_ad_in L_be_ :wordn)
 (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ :bool)
 (CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID :wordn)
 (ClkD :bool)
 (MB_data_in :wordn)
 (Edac_en_ :bool)
 (Bypass Test Failure0_ Failure1_ :bool)

 (L_ad_out :wordn)
 (L_ready_ :bool)
 (CB_ad_out CB_ms_out CB_ss_out :wordn)
 (CB_rqt_out_ :bool)
 (MB_addr MB_data_out :wordn)
 (MB_cs_eeprom_ MB_cs_sram_ MB_we_ MB_oe_ :bool)
 (Led :wordn)
 (Int0_ Int1 Int2 Int3_ Cpu_hist :bool) .

PIU_Block_SPEC rep
            (P_addr, P_dest1, P_be_, P_wr, P_fsm_state, P_fsm_rst, P_fsm_sack, P_fsm_cgnt_, P_fsm_hold_,
             P_rqt, P_size, P_down, P_lock_, P_lock_inh_, P_male_, P_rale_,
             C_mfsm_state, C_mfsm_D, C_mfsm_rst, C_mfsm_crqt_, C_mfsm_hold_, C_mfsm_ss, C_mfsm_invalid,
             C_sfsm_state, C_sfsm_D, C_sfsm_rst, C_sfsm_hlda_, C_sfsm_ms,
             C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_, C_efsm_rst,
             C_wr, C_sizewrbe, C_clkA, C_last_in_, C_lock_in_, C_ss, C_last_out_,
             C_hold_, C_holdA_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_iad_en_s_delA,
             C_wrdy, C_rrdy, C_parity, C_source, C_data_in, C_iad_out, C_iad_in, C_a1a0,C_a3a2,
             M_fsm_state, M_fsm_male_, M_fsm_last_, M_fsm_mrdy_, M_fsm_rst, M_count, M_se, M_wr, M_addr,
             M_be, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect,
             R_fsm_state, R_fsm_ale_, R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_ctr0_in, R_ctr0_mux_sel, R_ctr0,
             R_ctr0_irden, R_ctr0_new, R_ctr0_cry, R_ctr0_out, R_ctr0_orden, R_ctr1_in, R_ctr1_mux_sel,
             R_ctr1, R_ctr1_irden, R_ctr1_new, R_ctr1_cry, R_ctr1_out, R_ctr1_orden, R_ctr2_in, R_ctr2_mux_sel,
             R_ctr2, R_ctr2_irden, R_ctr2_new, R_ctr2_cry, R_ctr2_out, R_ctr2_orden, R_ctr3_in, R_ctr3_mux_sel,
             R_ctr3, R_ctr3_irden, R_ctr3_new, R_ctr3_cry, R_ctr3_out, R_ctr3_orden, R_icr_load, R_icr_old,
             R_icr_mask, R_icr_rden, R_icr, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr, R_sr_rden, R_int0_dis,
             R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_,
             R_reg_sel, R_busA_latch,
             S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass, S_soft_shot_del,
             S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1, S_cpu_hist, S_pmm_fail,
             S_cpu0_fail, S_cpu1_fail, S_piu_fail)
            (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_,
             CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in, ClkD, Id, ChannelID,
             MB_data_in, Edac_en_,
             Bypass, Test, Failure0_, Failure1_)
            (L_ad_out, L_ready_,
                    CB_ad_out, CB_ms_out, CB_ss_out, CB_rqt_out_,
                    MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_,
                    Int0_, Int1, Int2, Int3_, Led, Cpu_hist) =

? (i_ad i_be_ :wordn)
  (i_male_ i_rale_ i_crqt_ i_cgnt_ i_cale_ i_mrdy_ i_srdy_ i_last_ i_hold_ i_hlda_ i_lock_ :bool)

  (c_ss :wordn)
  (disable_writes cb_parity :bool)

  (ccr :wordn)
  (reset_error piu_invalid :bool)

  (mb_parity :bool)

  (s_state :wordn)
  (reset_cport disable_int reset_piu reset_cpu0 reset_cpu1 piu_fail pmm_fail cpu0_fail cpu1_fail :bool) .

(p_interp rep ((P_addr, P_dest1, P_be_, P_wr, P_fsm_state, P_fsm_rst, P_fsm_sack, P_fsm_cgnt_, P_fsm_hold_,
               P_rqt, P_size, P_down, P_lock_, P_lock_inh_, P_male_, P_rale_),
              (ClkA, ClkB, reset_piu, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, i_ad, i_cgnt_, i_hold_, i_srdy_),
              (L_ad_out, L_ready_, i_ad, i_ad, i_be_, i_rale_, i_male_, i_crqt_, i_cale_, i_mrdy_, i_last_, i_hlda_, i_lock_))) ∧
(c_interp rep ((C_mfsm_state,C_mfsm_D,C_mfsm_rst,C_mfsm_crqt_,C_mfsm_hold_,C_mfsm_ss,C_mfsm_invalid,
               C_sfsm_state,C_sfsm_D,C_sfsm_rst,C_sfsm_hlda_,C_sfsm_ms,
               C_efsm_state,C_efsm_cale_,C_efsm_last_,C_efsm_male_,C_efsm_rale_,C_efsm_srdy_,C_efsm_rst,
               C_wr,C_sizewrbe,C_clkA,C_last_in_,C_lock_in_,C_ss,C_last_out_,
               C_hold_,C_holdA_,C_cout_0_le_del,C_cin_2_le,C_mrdy_del_,C_iad_en_s_del,C_iad_en_s_delA,
               C_wrdy,C_rrdy,C_parity,C_source,C_data_in,C_iad_out,C_iad_in,C_a1a0,C_a3a2),
              (i_ad, i_be_, i_mrdy_, i_rale_, i_male_, i_last_, i_srdy_, i_lock_, i_cale_, i_hlda_, i_crqt_,
               CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in,
               reset_cport, ClkA, ClkB, ClkD, Id, ChannelID, pmm_fail, piu_invalid, ccr, reset_error),
              (i_cgnt_, i_mrdy_, i_hold_, i_rale_, i_male_, i_last_, i_srdy_, i_ad, i_be_,
               CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out, c_ss, disable_writes, cb_parity))) ∧
(m_interp rep ((M_fsm_state, M_fsm_male_, M_fsm_last_, M_fsm_mrdy_, M_fsm_rst, M_count, M_se,
               M_wr, M_addr, M_be, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect),
              (ClkA, ClkB, reset_piu, reset_cport, disable_writes, i_ad, i_male_, i_last_, i_be_,
               i_mrdy_, MB_data_in, Edac_en_, reset_error),
              (i_ad, i_srdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_, mb_parity))) ∧
(r_interp rep ((R_fsm_state, R_fsm_ale_, R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_ctr0_in, R_ctr0_mux_sel, R_ctr0,
               R_ctr0_irden, R_ctr0_new, R_ctr0_cry, R_ctr0_out, R_ctr0_orden, R_ctr1_in, R_ctr1_mux_sel,
               R_ctr1, R_ctr1_irden, R_ctr1_new, R_ctr1_cry, R_ctr1_out, R_ctr1_orden, R_ctr2_in, R_ctr2_mux_sel,
               R_ctr2, R_ctr2_irden, R_ctr2_new, R_ctr2_cry, R_ctr2_out, R_ctr2_orden, R_ctr3_in, R_ctr3_mux_sel,
               R_ctr3, R_ctr3_irden, R_ctr3_new, R_ctr3_cry, R_ctr3_out, R_ctr3_orden, R_icr_load, R_icr_old,
               R_icr_mask, R_icr_rden, R_icr, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr, R_sr_rden, R_int0_dis,
               R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_,
               R_reg_sel, R_busA_latch),
              (ClkA, reset_piu, i_ad, i_rale_, i_last_, i_be_, i_mrdy_, disable_int, disable_writes,
               cpu0_fail, cpu1_fail, reset_cpu0, reset_cpu1, piu_fail, pmm_fail, s_state, Id,
               ChannelID, cb_parity, mb_parity, c_ss),
              (i_ad, i_srdy_, Int0_, Int1, Int2, Int3_, ccr, Led, reset_error, piu_invalid))) ∧
(s_interp rep ((S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
               S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
               S_cpu_hist, S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_piu_fail),
              (ClkA, ClkB, Rst, Bypass, Test, Led, Failure0_, Failure1_),
                    (s_state, reset_cport, disable_int, reset_piu, reset_cpu0, reset_cpu1, Cpu_hist,
                     piu_fail, cpu0_fail, cpu1_fail, pmm_fail)))"
    );;
close_theory();;
```

Appendix F ML Source for the PIU Clock-Level Specification.

This appendix contains the HOL model for the clock-level specification of the PIU.

```
%----------------------------------------------------------------------------------------------

    File:       piu_clock1.ml

    Author:     (c) D.A. Fura 1992

    Date:       31 March 1992

    This file contains the ml source for the clock-level specification of the FTEP PIU, an ASIC
    developed by the Embedded Processing Laboratory, Boeing High Technology Center.

----------------------------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/titan3/dfura/ftep/piu/hol/lib/';
                                   '/home/titan3/dfura/ftep/piu/hol/pport/';
                                   '/home/titan3/dfura/ftep/piu/hol/cport/';
                                   '/home/titan3/dfura/ftep/piu/hol/mport/';
                                   '/home/titan3/dfura/ftep/piu/hol/rport/';
                                   '/home/titan3/dfura/ftep/piu/hol/sucont/']);;

system 'rm piu_clock1.th';;

new_theory 'piu_clock1';;

map new_parent ['paux_def';'caux_def';'maux_def';'raux_def';'saux_def';'aux_def';'array_def';'wordn_def'];;

loadf 'abstract';;

let MSTART = "WORDN 4";;
let MEND = "WORDN 5";;
let MRDY = "WORDN 6";;
let MWAIT = "WORDN 7";;
let MABORT = "WORDN 0";;

let SACK = "WORDN 5";;
let SRDY = "WORDN 6";;
let SWAIT = "WORDN 7";;
let SABORT = "WORDN 0";;

let piu_state_ty = ":(wordn#bool#wordn#bool#pfsm_ty#bool#bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool#
                    cmfsm_ty#bool#bool#bool#bool#wordn#bool#
                    csfsm_ty#bool#bool#bool#wordn#
                    cefsm_ty#bool#bool#bool#bool#bool#bool#
                    bool#wordn#bool#bool#bool#wordn#bool#
                    bool#bool#bool#bool#bool#bool#bool#
                    bool#bool#bool#wordn#wordn#wordn#wordn#wordn#wordn#
                    mfsm_ty#bool#bool#bool#bool#wordn#bool#bool#wordn#wordn#bool#bool#bool#wordn#wordn#
                    rfsm_ty#bool#bool#bool#bool#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#
                    wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#wordn#bool#
                    wordn#bool#wordn#bool#wordn#bool#bool#wordn#wordn#bool#wordn#wordn#bool#wordn#bool#wordn#
                    bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
                    sfsm_ty#bool#bool#bool#bool#bool#bool#wordn#wordn#
                    bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let piu_state = "((P_addr, P_dest1, P_be_, P_wr, P_fsm_state, P_fsm_rst, P_fsm_sack, P_fsm_cgnt_, P_fsm_hold_,
                  P_rqt, P_size, P_down, P_lock_, P_lock_inh_, P_male_, P_rale_,
                  C_mfsm_state,C_mfsm_D,C_mfsm_rst,C_mfsm_crqt_,C_mfsm_hold_,C_mfsm_ss,C_mfsm_invalid,
                  C_sfsm_state,C_sfsm_D,C_sfsm_rst,C_sfsm_hlda_,C_sfsm_ms,
                  C_efsm_state,C_efsm_cale_,C_efsm_last_,C_efsm_male_,C_efsm_rale_,C_efsm_srdy_,C_efsm_rst,
                  C_wr,C_sizewrbe,C_clkA,C_last_in_,C_lock_in_,C_ss,C_last_out_,
                  C_hold_,C_holdA_,C_cout_0_le_del,C_cin_2_le,C_mrdy_del_,C_iad_en_s_del,C_iad_en_s_delA,
                  C_wrdy,C_rrdy,C_parity,C_source,C_data_in,C_iad_out,C_iad_in,C_a1a0,C_a3a2,
                  M_fsm_state, M_fsm_male_, M_fsm_last_, M_fsm_mrdy_, M_fsm_rst, M_count, M_se, M_wr, M_addr,
                  M_be, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect,
                  R_fsm_state, R_fsm_ale_, R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_ctr0_in, R_ctr0_mux_sel, R_ctr0,
                  R_ctr0_irden, R_ctr0_new, R_ctr0_cry, R_ctr0_out, R_ctr0_orden, R_ctr1_in, R_ctr1_mux_sel,
                  R_ctr1, R_ctr1_irden, R_ctr1_new, R_ctr1_cry, R_ctr1_out, R_ctr1_orden, R_ctr2_in, R_ctr2_mux_sel,
                  R_ctr2, R_ctr2_irden, R_ctr2_new, R_ctr2_cry, R_ctr2_out, R_ctr2_orden, R_ctr3_in, R_ctr3_mux_sel,
                  R_ctr3, R_ctr3_irden, R_ctr3_new, R_ctr3_cry, R_ctr3_out, R_ctr3_orden, R_icr_load, R_icr_old,
                  R_icr_mask, R_icr_rden, R_icr, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr, R_sr_rden, R_int0_dis,
                  R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_,
                  R_reg_sel, R_busA_latch,
                  S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass,
                  S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
                  S_cpu_hist, S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_piu_fail)
                 :^piu_state_ty)";;

let piu_env_ty = ":(bool#bool#bool#wordn#bool#bool#wordn#bool#bool#
                   wordn#wordn#wordn#wordn#bool#wordn#wordn#
                   wordn#bool#
                   bool#bool#bool#bool)";;
let piu_env = "((ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_,
                CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in, ClkD, Id, ChannelID,
                MB_data_in, Edac_en_,
                Bypass, Test, Failure0_, Failure1_)
               :^piu_env_ty)";;

let piu_out_ty = ":(wordn#bool#
                   bool#wordn#wordn#wordn#
                   wordn#wordn#bool#bool#bool#bool#
                   bool#bool#bool#bool#wordn#
                   bool#bool#bool#bool#bool#bool#bool)";;
let piu_out = "((L_ad_out, L_ready_,
               CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out,
               MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_,
               Int0_, Int1, Int2, Int3_, Led,
               Reset_cpu0, Reset_cpu1, Cpu_hist, Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail)
              :^piu_out_ty)";;

let rep_ty = abstract_type 'aux_def' 'Andn';;

%------------------------------------------------------------------------------------------------
  Next-state definition for EXEC instruction.
------------------------------------------------------------------------------------------------%
```

```
let piuEXEC_inst_def = new_definition
 ('piuEXEC_inst',

   "! (rep:^rep_ty)

      (P_fsm_state :pfsm_ty)
      (P_addr P_be_ P_size :wordn)
      (P_dest1 P_wr P_fsm_rst P_fsm_sack P_fsm_cgnt_ P_fsm_hold_ P_rqt P_down P_lock_
       P_lock_inh_ P_male_ P_rale_ :bool)
      (C_mfsm_state :cmfsm_ty) (C_sfsm_state :csfsm_ty) (C_efsm_state :cefsm_ty)
      (C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss C_source C_data_in C_iad_out C_iad_in C_a1a0 C_a3a2 :wordn)
      (C_mfsm_D C_mfsm_rst C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_invalid C_sfsm_D C_sfsm_rst C_sfsm_hlda_
       C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
       C_wr C_clkA C_last_in_ C_lock_in_ C_last_out_ C_hold_ C_holdA_ C_cout_0_le_del C_cin_2_le
       C_mrdy_del_ C_iad_en_s_del C_iad_en_s_delA C_wrdy C_rrdy C_parity :bool)
      (M_fsm_state :mfsm_ty)
      (M_count M_addr M_be M_rd_data M_detect :wordn)
      (M_fsm_male_ M_fsm_last_ M_fsm_mrdy_ M_fsm_rst M_se M_wr M_rdy M_wwdel M_parity :bool)
      (R_fsm_state :rfsm_ty)
      (R_ctr0_in R_ctr0 R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1 R_ctr1_new R_ctr1_out R_ctr2_in R_ctr2 R_ctr2_new
       R_ctr2_out R_ctr3_in R_ctr3 R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr R_ccr R_gcr R_sr
       R_reg_sel R_busA_latch :wordn)
      (R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden
       R_ctr1_mux_sel R_ctr1_irden R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden
       R_ctr3_mux_sel R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden R_sr_rden
       R_int0_dis R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del R_srdy_del_ :bool)
      (S_fsm_state :sfsm_ty)
      (S_soft_cnt S_delay :wordn)
      (S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass S_soft_shot_del S_bad_cpu0 S_bad_cpu1
       S_reset_cpu0 S_reset_cpu1 S_cpu_hist S_pmm_fail S_cpu0_fail S_cpu1_fail S_piu_fail :bool)

      (L_ad_in L_be_ :wordn)
      (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ :bool)
      (CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID :wordn)
      (ClkD :bool)
      (MB_data_in :wordn)
      (Edac_en_ :bool)
      (Bypass Test Failure0_ Failure1_ :bool) .

  piuEXEC_inst rep
            (P_addr, P_dest1, P_be_, P_wr, P_fsm_state, P_fsm_rst, P_fsm_sack, P_fsm_cgnt_, P_fsm_hold_,
             P_rqt, P_size, P_down, P_lock_, P_lock_inh_, P_male_, P_rale_,
             C_mfsm_state, C_mfsm_D, C_mfsm_rst, C_mfsm_crqt_, C_mfsm_hold_, C_mfsm_ss, C_mfsm_invalid,
             C_sfsm_state, C_sfsm_D, C_sfsm_rst, C_sfsm_hlda_, C_sfsm_ms,
             C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_, C_efsm_rst,
             C_wr, C_sizewrbe, C_clkA, C_last_in_, C_lock_in_, C_ss, C_last_out_,
             C_hold_, C_holdA_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_iad_en_s_delA,
             C_wrdy, C_rrdy, C_parity, C_source, C_data_in, C_iad_out, C_iad_in, C_a1a0,C_a3a2,
             M_fsm_state, M_fsm_male_, M_fsm_last_, M_fsm_mrdy_, M_fsm_rst, M_count, M_se, M_wr, M_addr,
             M_be, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect,
             R_fsm_state, R_fsm_ale_, R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_ctr0_in, R_ctr0_mux_sel, R_ctr0,
             R_ctr0_irden, R_ctr0_new, R_ctr0_cry, R_ctr0_out, R_ctr0_orden, R_ctr1_in, R_ctr1_mux_sel,
             R_ctr1, R_ctr1_irden, R_ctr1_new, R_ctr1_cry, R_ctr1_out, R_ctr1_orden, R_ctr2_in, R_ctr2_mux_sel,
             R_ctr2, R_ctr2_irden, R_ctr2_new, R_ctr2_cry, R_ctr2_out, R_ctr2_orden, R_ctr3_in, R_ctr3_mux_sel,
          R_ctr3, R_ctr3_irden, R_ctr3_new, R_ctr3_cry, R_ctr3_out, R_ctr3_orden, R_icr_load, R_icr_old,
          R_icr_mask, R_icr_rden, R_icr, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr, R_sr_rden, R_int0_dis,
          R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_,
          R_reg_sel, R_busA_latch,
          S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass, S_soft_shot_del,
          S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1, S_cpu_hist, S_pmm_fail,
          S_cpu0_fail, S_cpu1_fail, S_piu_fail)

         (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_,
          CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in, ClkD, Id, ChannelID,
          MB_data_in, Edac_en_,
          Bypass, Test, Failure0_, Failure1_) =

let new_P_fsm_state =
      ((P_fsm_rst) => PA |
       ((P_fsm_state = PH) => ((~P_fsm_hold_) => PH | PA) |
       ((P_fsm_state = PA) =>
          (((P_rqt ∧ ~P_dest1) ∨ (P_rqt ∧ P_dest1 ∧ ~P_fsm_cgnt_)) => PD |
          ((~P_fsm_hold_ ∧ P_lock_) => PH | PA)) |
       ((P_fsm_state = PD) =>
          (((P_fsm_sack ∧ P_fsm_hold_) ∨ (P_fsm_sack ∧ ~P_fsm_hold_ ∧ ~P_lock_)) => PA |
          ((P_fsm_sack ∧ ~P_fsm_hold_ ∧ P_lock_) => PH | PD)) | P_ILL)))) in

let c_write = (((~(C_mfsm_state = CMI)) ∧ (~(C_mfsm_state = CMR))) => C_wr | (ELEMENT C_sizewrbe (5))) in
let c_busy = (~((SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
let c_grant = ((((SUBARRAY Id (1,0)) = (WORDN 0)) ∧ ~(ELEMENT CB_rqt_in_ (0)))
          ∨ (((SUBARRAY Id (1,0)) = (WORDN 1)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                   ∧ (ELEMENT CB_rqt_in_ (1)))
          ∨ (((SUBARRAY Id (1,0)) = (WORDN 2)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                   ∧ (ELEMENT CB_rqt_in_ (1))
                                                   ∧ (ELEMENT CB_rqt_in_ (2)))
          ∨ (((SUBARRAY Id (1,0)) = (WORDN 3)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                   ∧ (ELEMENT CB_rqt_in_ (1))
                                                   ∧ (ELEMENT CB_rqt_in_ (2))
                                                   ∧ (ELEMENT CB_rqt_in_ (3)))) in
let c_addressed = (Id = (SUBARRAY C_source (15,10))) in
let new_C_mfsm_state =
      ((C_mfsm_rst) => CMI |
      ((C_mfsm_state = CMI) =>
          (C_mfsm_D ∧ ~C_mfsm_crqt_ ∧ ~c_busy ∧ ~C_mfsm_invalid) => CMR | CMI |
      ((C_mfsm_state = CMR) => (C_mfsm_D ∧ c_grant ∧ C_mfsm_hold_) => CMA3 | CMR |
      ((C_mfsm_state = CMA3) => ((C_mfsm_D) => CMA1 | CMA3) |
      ((C_mfsm_state = CMA1) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA0 |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA1 |
      ((C_mfsm_state = CMA0) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA2 |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA0 |
      ((C_mfsm_state = CMA2) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD1 |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA2 |
      ((C_mfsm_state = CMD1) =>
          (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD0 |
          (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD1 |
        ((C_mfsm_state = CMD0) =>
              (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ C_last_in_) => CMD1 |
              (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_last_in_) => CMW |
              (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD0 |
        ((C_mfsm_state = CMW) =>
              (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT |
              (C_mfsm_D ∧ (C_mfsm_ss = ^SACK) ∧ C_lock_in_) => CMI |
              (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_lock_in_ ∧ ~C_mfsm_crqt_) => CMA3 | CMW |
        ((~C_last_in_) => CMI | CMABT)))))))))) in

let new_C_sfsm_state =
       ((C_sfsm_rst) => CSI |
              (C_sfsm_state = CSI) =>
                 ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ c_addressed) => CSA1 | CSI) |
        (C_sfsm_state = CSL) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ c_addressed) => CSA1 |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ ~c_addressed) => CSI |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSL) |
        (C_sfsm_state = CSA1) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSA0 |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA1) |
        (C_sfsm_state = CSA0) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ C_sfsm_hlda_) => CSA0W |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0) |
        (C_sfsm_state = CSA0W) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0W) |
        (C_sfsm_state = CSALE) =>
              ((C_sfsm_D ∧ c_write ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
              (C_sfsm_D ∧ ~c_write ∧ (C_sfsm_ms = ^MRDY)) => CSRR |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSALE) |
        (C_sfsm_state = CSRR) =>
              ((C_sfsm_D ∧ ~(C_sfsm_ms = ^MABORT)) => CSD1 |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSRR) |
        (C_sfsm_state = CSD1) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD0 |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD1) |
        (C_sfsm_state = CSD0) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MEND)) => CSACK |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD0) |
        (C_sfsm_state = CSACK) =>
              ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSL |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MWAIT)) => CSI |
              (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSACK) |
        (C_sfsm_D) => CSI | CSABT) in

let new_C_efsm_state =
       ((C_efsm_rst) => CEI |
        (C_efsm_state = CEI) => ((~C_efsm_cale_) => CEE | CEI) |
        ((~C_efsm_last_ ∧ ~C_efsm_srdy_) ∨ ~C_efsm_male_ ∨ ~C_efsm_rale_) => CEI | CEE) in

let m_bw = ((~(M_be = (WORDN 15))) ∧ M_wr ∧ (~(M_fsm_state = MI))) in
let m_ww = ((M_be = (WORDN 15)) ∧ M_wr ∧ (~(M_fsm_state = MI))) in
let new_M_fsm_state =
        ((M_fsm_rst) => MI |
         ((M_fsm_state = MI) => ((~M_fsm_male_) => MA | MI) |
         ((M_fsm_state = MA) =>
                ((~M_fsm_mrdy_ ∧ m_ww) => MW |
                ((~M_fsm_mrdy_ ∧ ((~M_wr ∧ (~(M_fsm_state = MI))) ∨ m_bw)) => MR | MA)) |
         ((M_fsm_state = MR) =>
                ((m_bw ∧ (M_count = (WORDN 0))) => MBW |
                ((M_fsm_last_ ∧ ~M_wr ∧ (~(M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) => MA |
                ((~M_fsm_last_ ∧ ~M_wr ∧ (~(M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) => MRR | MR))) |
         ((M_fsm_state = MRR) => MI |
         ((M_fsm_state = MW) =>
                ((~M_fsm_last_ ∧ (M_count = (WORDN 0))) => MI |
                ((M_fsm_last_ ∧ (M_count = (WORDN 0))) => MA | MW)) |
         ((M_fsm_state = MBW) => MW | M_ILL))))))) in

let new_R_fsm_state =
        ((R_fsm_rst) => RI |
         ((R_fsm_state = RI) => ((~R_fsm_ale_) => RA | RI) |
         ((R_fsm_state = RA) => ((~R_fsm_mrdy_) => RD | RA) |
         ((~R_fsm_last_) => RI | RA)))) in
let r_fsm_cntlatch = ((R_fsm_state = RI) ∧ ~R_fsm_ale_) in
let r_fsm_srdy_ = ~((R_fsm_state = RA) ∧ ~R_fsm_mrdy_) in

let new_S_fsm_state =
        ((S_fsm_rst) => SSTART |
        ((S_fsm_state = SSTART) => SRA |
        ((S_fsm_state = SRA) => ((S_fsm_delay6) => ((S_fsm_bypass) => SO | SPF) | SRA) |
        ((S_fsm_state = SPF) => SC0I |
        ((S_fsm_state = SC0I) => ((S_fsm_delay17) => SC0F | SC0I) |
        ((S_fsm_state = SC0F) => ST |
        ((S_fsm_state = ST) => SC1I |
        ((S_fsm_state = SC1I) => ((S_fsm_delay17) => SC1F | SC1I) |
        ((S_fsm_state = SC1F) => SS |
        ((S_fsm_state = SS) => ((S_fsm_bothbad) => SSTOP | SCS) |
        ((S_fsm_state = SSTOP) => SSTOP |
        ((S_fsm_state = SCS) => ((S_fsm_delay6) => SN | SCS) |
        ((S_fsm_state = SN) => ((S_fsm_delay17) => SO | SN) |
        ((S_fsm_state = SO) => SO | S_ILL)))))))))))))) in
let s_fsm_sn = (new_S_fsm_state = SN) in
let s_fsm_so = (new_S_fsm_state = SO) in
let reset_cport = (((~(new_S_fsm_state = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let s_fsm_sdi = (((~(new_S_fsm_state = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let reset_piu = ((new_S_fsm_state = SSTART) ∨ (new_S_fsm_state = SRA)
                ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
                ∨ (new_S_fsm_state = SC1F) ∨ (new_S_fsm_state = SS) ∨ (new_S_fsm_state = SCS)) in
let s_fsm_src0 = ((~(new_S_fsm_state = SPF)) ∧ (~(new_S_fsm_state = SC0I))) in
let s_fsm_src1 = ((~(new_S_fsm_state = ST)) ∧ (~(new_S_fsm_state = SC1I))) in
let s_fsm_spf = ((S_fsm_state = SRA) ∧ S_fsm_delay6 ∧ ~S_fsm_rst) in
let s_fsm_sc0f = (new_S_fsm_state = SC0F) in
let s_fsm_sc1f = (new_S_fsm_state = SC1F) in
let s_fsm_spmf = (new_S_fsm_state = SO) in
let s_fsm_sb = (new_S_fsm_state = SSTART) in
let s_fsm_src = ((new_S_fsm_state = SSTART) ∨ ((S_fsm_state = SRA) ∧ S_fsm_delay6)
                ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST) ∨ (new_S_fsm_state = SC1F)
                ∨ (new_S_fsm_state = SS) ∨ ((S_fsm_state = SCS) ∧ S_fsm_delay6)) in
let s_fsm_sec = (((~(new_S_fsm_state = SSTOP)) ∧ (~(new_S_fsm_state = SO))) ∨ (S_fsm_state = SN)) in
let s_fsm_srs = (((S_fsm_state = SPF) ∧ ~S_fsm_rst) ∨ ((S_fsm_state = ST) ∧ ~S_fsm_rst)) in
let s_fsm_scs = (new_S_fsm_state = SCS) in

let new_P_addr = ((~P_rqt) => (SUBARRAY L_ad_in (25,0)) | P_addr) in
let new_P_destl = ((~P_rqt) => (ELEMENT L_ad_in (31)) | P_destl) in
let new_P_be_ = ((~P_rqt) => L_be_ | P_be_) in
let new_P_wr = ((~P_rqt) => L_wr | P_wr) in
let new_P_size =
   ((~P_rqt) => (SUBARRAY L_ad_in (1,0)) |
((P_down) => (DECN 1 P_size) | P_size)) in
let new_C_holdA_ = ((ClkD) => C_hold_ | C_holdA_) in
let i_cale_ = ~((new_C_mfsm_state = CMA3) ∧ (new_P_fsm_state = PA) ∧ new_C_holdA_) in
let c_srdy_en = ((new_C_efsm_state = CEE) ∨ (C_efsm_state = CEE)) in
let new_M_count =
     (((new_M_fsm_state = MA) ∨ (new_M_fsm_state = MBW)) => ((M_se) => (WORDN 1) | (WORDN 2)) |
      (((new_M_fsm_state = MW) ∨ (new_M_fsm_state = MR)) => (DECN 2 M_count) | M_count)) in
let m_rdy = (((new_M_fsm_state = MW) ∧ (new_M_count = (WORDN 0)))
            ∨ ((new_M_fsm_state = MR) ∧ (new_M_count = (WORDN 0)) ∧ ~M_wr)) in
let m_srdy_ = ~((M_rdy ∧ ~M_wr) ∨ (m_rdy ∧ M_wr)) in
let i_srdy_ = ((~i_cale_ ∨ c_srdy_en) => ~(C_wrdy ∨ C_rrdy ∨ (new_C_mfsm_state = CMABT)) |
                ~(new_M_fsm_state = MI) => m_srdy_ |
                ((new_R_fsm_state = RA) ∨ (new_R_fsm_state = RD)) => ~((R_fsm_state = RA) ∧
                                                             (new_R_fsm_state = RD)) | ARB) in
let p_ale = (~L_ads_ ∧ L_den_) in
let p_sack = ((P_size = ((P_down) => (WORDN 1) | (WORDN 0))) ∧ ~i_srdy_ ∧ (new_P_fsm_state = PD)) in
let new_P_rqt =
   ((p_ale ∧ ~(p_sack ∨ reset_piu)) => T |
((~p_ale ∧ (p_sack ∨ reset_piu)) => F |
   ((~p_ale ∧ ~(p_sack ∨ reset_piu)) => P_rqt | ARB))) in
let new_P_down = (~i_srdy_ ∧ (new_P_fsm_state = PD)) in
let new_P_male_ = ((new_P_fsm_state = PA) =>
   ~(~new_P_destl ∧ (~((SUBARRAY new_P_addr (25,24)) = (WORDN 3))) ∧ new_P_rqt) | P_male_) in
let new_P_rale_ = ((new_P_fsm_state = PA) =>
   ~(~new_P_destl ∧ ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) ∧ new_P_rqt) | P_rale_) in
let new_P_lock_ =
   ((reset_piu) => T |
   ((new_P_fsm_state = PD) => L_lock_ | P_lock_)) in
let new_P_lock_inh_ =
   ((reset_piu) => T |
   ((~new_P_male_ ∨ ~new_P_rale_) => L_lock_ | P_lock_inh_)) in
let pod31_27 = (MALTER ARBN (31,27) new_P_be_) in
let pod31_26 = (ALTER pod31_27 (26) F) in
let pod31_24 = (MALTER pod31_26 (25,24) (SUBARRAY new_P_addr (1,0))) in
let new_C_iad_en_s_delA = ((ClkD) => C_iad_en_s_del | C_iad_en_s_delA) in
let new_C_sizewrbe = ((reset_cport) => (WORDN 0) |
                              (((new_C_sfsm_state = CSA0) ∧ C_clkA) => (SUBARRAY C_data_in (31,22)) | C_sizewrbe)) in
let c_new_write = (((~(new_C_mfsm_state = CMI)) ∧ (~(new_C_mfsm_state = CMR))) =>
                         C_wr | (ELEMENT new_C_sizewrbe (5))) in
let new_C_iad_out = ((C_cin_2_le) => C_data_in | C_iad_out) in
let r_reg_sel = ((~R_srdy_del_) => (INCN 3 R_reg_sel) | R_reg_sel) in
let new_R_icr =
        ((R_icr_load) =>
            ((~(r_reg_sel = (WORDN 1))) => (Andn rep (R_icr_old, R_icr_mask)) | (Orn rep (R_icr_old, R_icr_mask))) |
         R_icr) in
let new_R_busA_latch =
        ((R_ctr0_irden) => R_ctr0_in |
         ((R_ctr0_orden) => R_ctr0_out |
         ((R_ctr1_irden) => R_ctr1_in |
         ((R_ctr1_orden) => R_ctr1_out |
         ((R_ctr2_irden) => R_ctr2_in |
         ((R_ctr2_orden) => R_ctr2_out |
         ((R_ctr3_irden) => R_ctr3_in |
         ((R_ctr3_orden) => R_ctr3_out |
         ((R_icr_rden) => new_R_icr |
         ((R_ccr_rden) => R_ccr |
         ((R_gcr_rden) => R_gcr |
         ((R_sr_rden) => R_sr | ARB))))))))))) in
let i_ad = ((new_P_fsm_state = PA) => pod31_24 |
             ((new_P_fsm_state = PD) ∧ new_P_wr) => L_ad_in |
             (new_C_iad_en_s_delA ∨
              ((new_C_mfsm_state = CMD1) ∧ ~c_new_write ∧ c_srdy_en) ∨
              ((new_C_mfsm_state = CMD0) ∧ ~c_new_write ∧ c_srdy_en) ∨
              ((new_C_mfsm_state = CMW) ∧ (C_mfsm_state = CMD0) ∧ ~c_new_write ∧ c_srdy_en) ∨
              ((new_C_sfsm_state = CSALE) ∧ (~(C_sfsm_state = CSALE))) ∨
              ((new_C_sfsm_state = CSALE) ∧ c_new_write) ∨
              ((new_C_sfsm_state = CSD1) ∧ c_new_write ∧ (~(C_sfsm_state = CSRR))) ∨
              ((new_C_sfsm_state = CSD0) ∧ c_new_write) ∨
              ((new_C_sfsm_state = CSACK) ∧ c_new_write)) => new_C_iad_out |
             (M_wr ∧ ~(new_M_fsm_state = MI)) => M_rd_data |
             (~R_wr ∧ ((new_R_fsm_state = RA) ∨ (new_R_fsm_state = RD))) => new_R_busA_latch | ARB) in
let disable_writes = ((~(new_C_sfsm_state = CSI)) ∧ (~(new_C_sfsm_state = CSL)) ∧
                      ~((ChannelID = (WORDN 0)) ∧ (ELEMENT C_source (6))) ∧
                      ~((ChannelID = (WORDN 1)) ∧ (ELEMENT C_source (7))) ∧
                      ~((ChannelID = (WORDN 2)) ∧ (ELEMENT C_source (8))) ∧
                      ~((ChannelID = (WORDN 3)) ∧ (ELEMENT C_source (9)))) in
let i_rale_ =
        (~(new_P_fsm_state = PH) =>
         ~(~new_P_dest1 ∧ ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) ∧ (new_P_fsm_state = PA) ∧ new_P_rqt) |
         ~((new_C_sfsm_state = CSALE) ∧ ((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3)) ∧ C_clkA)) in
let new_R_wr = ((~i_rale_) => (ELEMENT i_ad (27)) | R_wr) in
let r_writeB = (~disable_writes ∧ new_R_wr ∧ (new_R_fsm_state = RD)) in
let r_readB = (~new_R_wr ∧ (new_R_fsm_state = RA)) in
let new_R_gcr = ((r_writeB ∧ (r_reg_sel = (WORDN 2))) => i_ad | R_gcr) in
let new_R_gcr_rden = (r_readB ∧ (r_reg_sel = (WORDN 2))) in
let gcrl = (ELEMENT new_R_gcr (0)) in
let gcrh = (ELEMENT new_R_gcr (1)) in
let reset_error = (ELEMENT new_R_gcr (24)) in
let piu_invalid = (ELEMENT new_R_gcr (28)) in
let cout_sel0 = (ALTER ARBN (0) (((new_C_sfsm_state = CSD1) ∨ (new_C_sfsm_state = CSD0)) =>
                                  (new_C_sfsm_state = CSD1) |
                                  (new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA1)
                                                     ∨ (new_C_mfsm_state = CMD1))) in
let c_cout_sel = (ALTER cout_sel0 (1) (((new_C_sfsm_state = CSD1) ∨ (new_C_sfsm_state = CSD0)) =>
                                   F |
                              (new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA2))) in
let new_C_hold_ = (new_C_sfsm_state = CSI) in
let new_C_wr = ((~i_cale_) => (ELEMENT i_ad (27)) | C_wr) in
let new_C_clkA = ClkD in
let i_last_ =
      (~(new_P_fsm_state = PH) =>
       (P_size = ((P_down) => (WORDN 1) | (WORDN 0))) |
       C_last_out_) in
let new_C_last_in_ = ((reset_cport) => F |
                      (((new_C_mfsm_state = CMABT) ∨ (new_C_mfsm_state = CMD1) ∧ ClkD) => i_last_ |
                      C_last_in_)) in
let new_C_lock_in_ = ((reset_cport) => F |
                       ((new_C_mfsm_state = CMA1) => ~(~new_P_lock_ ∧ new_P_lock_inh_) |
                       C_lock_in_)) in
let new_C_ss = (((~(new_C_mfsm_state = CMABT)) ∧ (~(new_C_mfsm_state = CMI))) => CB_ss_in | C_ss) in
let new_C_last_out_ =
      (((new_C_sfsm_state = CSA1) ∧ ~(ClkD ∧ ((CB_ms_in = ^MEND) ∨ (CB_ms_in = ^MABORT)))) => T |
       ((~(new_C_sfsm_state = CSA1) ∧ (ClkD ∧ ((CB_ms_in = ^MEND) ∨ (CB_ms_in = ^MABORT)))) => F |
       ((~(new_C_sfsm_state = CSA1) ∧ ~(ClkD ∧ ((CB_ms_in = ^MEND) ∨ (CB_ms_in = ^MABORT)))) => C_last_out_ |
       ARB))) in
let c_srdy = (CB_ss_in = ^SRDY) in
let c_dfsm_master = ((new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA2) ∨ (new_C_mfsm_state = CMA1)
                    ∨ (new_C_mfsm_state = CMA0) ∨ (new_C_mfsm_state = CMD1) ∨ (new_C_mfsm_state = CMD0)) in
let c_dfsm_cad_en = ~((new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA1) ∨ (new_C_mfsm_state = CMA0)
                     ∨ (new_C_mfsm_state = CMA2)
                     ∨ (c_new_write ∧ ((new_C_mfsm_state = CMD1) ∨ (new_C_mfsm_state = CMD0)))
                     ∨ (~c_new_write ∧ ((new_C_sfsm_state = CSD1) ∨ (new_C_sfsm_state = CSD0)))) in
let new_C_cout_0_le_del = ((i_cale_) ∨ (i_srdy_ ∧ ~c_new_write)
                          ∨ ((new_C_mfsm_state = CMA0) ∧ c_srdy ∧ c_new_write ∧ ClkD)
                          ∨ ((new_C_mfsm_state = CMD0) ∧ c_new_write ∧ c_srdy ∧ ClkD)) in
let new_C_cin_2_le = (ClkD ∧ (((new_C_mfsm_state = CMD0) ∧ c_srdy ∧ ~c_new_write) ∨
                             ((new_C_sfsm_state = CSA0)) ∨
                             ((new_C_sfsm_state = CSD0) ∧ c_new_write))) in
let new_C_mrdy_del_ = ~((~c_new_write ∧ ClkD ∧ ((new_C_sfsm_state = CSALE) ∨ (new_C_sfsm_state = CSD1))) ∨
                        (~c_new_write ∧ C_clkA ∧ (new_C_sfsm_state = CSACK)) ∨
                        (c_new_write ∧ ClkD ∧ (new_C_sfsm_state = CSD0))) in
let new_C_iad_en_s_del = (((new_C_sfsm_state = CSALE) ∧ (~(C_sfsm_state = CSALE)))
                         ∨ ((new_C_sfsm_state = CSALE) ∧ c_new_write)
                         ∨ ((new_C_sfsm_state = CSD1) ∧ c_new_write ∧ (~(C_sfsm_state = CSRR)))
                         ∨ ((new_C_sfsm_state = CSD0) ∧ c_new_write) ∨
                           ((new_C_sfsm_state = CSACK) ∧ c_new_write)) in
let new_C_wrdy = (c_srdy ∧ c_new_write ∧ (new_C_mfsm_state = CMD1) ∧ ClkD) in
let new_C_rrdy = (c_srdy ∧ ~c_new_write ∧ (new_C_mfsm_state = CMD0) ∧ ClkD) in
let c_pe = (Par_Det rep (CB_ad_in)) in
let c_mparity = ((new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA1) ∨ (new_C_mfsm_state = CMA0)
                ∨ (new_C_mfsm_state = CMA2) ∨ (new_C_mfsm_state = CMD1) ∨ (new_C_mfsm_state = CMD0)
                ∨ (C_mfsm_state = CMA1) ∨ (C_mfsm_state = CMA0) ∨ (C_mfsm_state = CMA2)
                ∨ (C_mfsm_state = CMD1)) in
let c_sparity = ((~(new_C_sfsm_state = CSI)) ∧ (~(new_C_sfsm_state = CSACK)) ∧ (~(new_C_sfsm_state = CSABT))) in
let c_pe_cnt = (ClkD ∧ ((~(c_mparity = c_sparity)) ∨ ((SUBARRAY CB_ss_in (1,0)) = (WORDN 0)))) in
let new_C_parity =
      (((ClkD ∧ c_pe ∧ c_pe_cnt) ∧ ~reset_error) => T |
       ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ reset_error) => F |
       ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ ~reset_error) => C_parity | ARB))) in
```

```
let new_C_source =
        ((reset_cport) => (WORDN 0) |
         ((ClkD ∧ ((new_C_sfsm_state = CSI) ∨ (new_C_sfsm_state = CSL))) => Par_Dec rep (CB_ad_in) | C_source)) in
let data_in31_16 =
        (MALTER ARBN (31,16) ((reset_cport) => (WORDN 0) |
                                        ((ClkD ∧ (((new_C_mfsm_state = CMD1) ∧ c_srdy ∧ ~c_new_write) ∨
                                                  ((new_C_sfsm_state = CSA1)) ∨
                                                  ((new_C_sfsm_state = CSD1) ∧ c_new_write))) => Par_Dec rep (CB_ad_in) |
                                        (SUBARRAY C_data_in (31,16))))) in
let new_C_data_in =
        (MALTER data_in31_16 (15,0) ((reset_cport) => (WORDN 0) |
                                            ((new_C_cin_2_le) => Par_Dec rep (CB_ad_in) |
                                            (SUBARRAY C_data_in (15,0))))) in
let new_C_iad_in = ((new_C_cout_0_le_del) => i_ad | C_iad_in) in
let new_C_a1a0 =
        (((c_dfsm_master ∧ C_cout_0_le_del) ∨
         (~c_dfsm_master ∧ C_clkA ∧ (new_C_sfsm_state = CSD1))) => C_iad_in | C_a1a0) in
let new_C_a3a2 = ((new_C_mfsm_state = CMR) => R_ccr | C_a3a2) in
let i_be_ = ((new_P_fsm_state = PA) => new_P_be_ |
             (new_P_fsm_state = PD) => L_be_ | SUBARRAY new_C_sizewrbe (9,6)) in
let i_male_ =
        (~(new_P_fsm_state = PH) =>
         ~(~new_P_dest1 ∧ (~((SUBARRAY new_P_addr (25,24)) = (WORDN 3))) ∧ (new_P_fsm_state = PA) ∧ new_P_rqt) |
         ~((new_C_sfsm_state = CSALE) ∧ (~((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3))) ∧ C_clkA)) in
let new_M_se = ((~i_male_) => (ELEMENT i_ad (23)) | M_se) in
let new_M_wr = ((~i_male_) => (ELEMENT i_ad (27)) | M_wr) in
let new_M_addr =
        ((~i_male_) => (SUBARRAY i_ad (18,0)) |
         ((M_rdy) => (INCN 18 M_addr) | M_addr)) in
let new_M_be = ((~i_male_ ∨ ~m_srdy_) => (NOTN 3 i_be_) | M_be) in
let new_M_rdy = m_rdy in
let new_M_wwdel = ((new_M_fsm_state = MA) ∧ new_M_wr ∧ (new_M_be = (WORDN 15))) in
let new_M_rd_data = (((new_M_fsm_state = MR)) => (Ham_Dec rep MB_data_in) | M_rd_data) in
let new_M_detect =
        ((((new_M_fsm_state = MR) ∧ ~new_M_wr) ∨ new_M_wr ∨ (new_M_fsm_state = MI)) =>
         ((~Edac_en_) => (Ham_Det1 rep MB_data_in) | WORDN 0) | M_detect) in
let m_error = (~m_srdy_ ∧ (~(new_M_fsm_state = MI)) ∧ Ham_Det2 rep (new_M_detect, ~Edac_en_)) in
let new_M_parity =
        ((m_error ∧ ~(reset_piu ∨ reset_error)) => T |
         ((~m_error ∧ (reset_piu ∨ reset_error)) => F |
         ((~m_error ∧ ~(reset_piu ∨ reset_error)) => M_parity | ARB))) in
let new_R_cntlatch_del = r_fsm_cntlatch in
let new_R_srdy_del_ = r_fsm_srdy_ in
let new_R_reg_sel =
        ((~i_rale_) => (SUBARRAY i_ad (3,0)) |
         ((~R_srdy_del_) => (INCN 3 R_reg_sel) | R_reg_sel)) in
let r_writeA = (~disable_writes ∧ R_wr ∧ (new_R_fsm_state = RD)) in
let r_readA = (~R_wr ∧ (new_R_fsm_state = RA)) in
let r_cir_wr01A = ((r_writeA ∧ ((r_reg_sel = (WORDN 8)) ∨ (r_reg_sel = (WORDN 9))))) in
let r_cir_wr01B = ((r_writeB ∧ ((r_reg_sel = (WORDN 8)) ∨ (r_reg_sel = (WORDN 9))))) in
let r_cir_wr23A = ((r_writeA ∧ ((r_reg_sel = (WORDN 10)) ∨ (r_reg_sel = (WORDN 11))))) in
let r_cir_wr23B = ((r_writeB ∧ ((r_reg_sel = (WORDN 10)) ∨ (r_reg_sel = (WORDN 11))))) in
let new_R_ccr = ((r_writeB ∧ (r_reg_sel = (WORDN 3))) => i_ad | R_ccr) in
let new_R_ccr_rden = (r_readB ∧ (r_reg_sel = (WORDN 3))) in
let new_R_c01_cout_del = R_ctr1_cry in
let new_R_int1_en =
        ((((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
         ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => T |
         ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
         (~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => F |
         ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
         ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => R_int1_en | ARB))) in
let new_R_c23_cout_del = R_ctr3_cry in
let new_R_int2_en =
        ((((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
         ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => T |
         ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
         (~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => F |
         ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
         ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => R_int2_en | ARB))) in
let new_R_ctr0_in = ((r_writeB ∧ (r_reg_sel = (WORDN 8))) => i_ad | R_ctr0_in) in
let new_R_ctr0_mux_sel = (r_cir_wr01B ∨ ((ELEMENT new_R_gcr (16)) ∧ R_ctr1_cry)) in
let new_R_ctr0_irden = (r_readB ∧ (r_reg_sel = (WORDN 8))) in
let new_R_ctr0 = ((R_ctr0_mux_sel) => R_ctr0_in | R_ctr0_new) in
let new_R_ctr0_new = (((ELEMENT new_R_gcr (19))) => (INCN 31 R_ctr0) | R_ctr0) in
let new_R_ctr0_cry = ((ONES 31 R_ctr0) ∧ (ELEMENT new_R_gcr (19))) in
let new_R_ctr0_out = ((r_fsm_cntlatch) => R_ctr0_new | R_ctr0_out) in
let new_R_ctr0_orden = (r_readB ∧ (r_reg_sel = (WORDN 12))) in
let new_R_ctr1_in = ((r_writeB ∧ (r_reg_sel = (WORDN 9))) => i_ad | R_ctr1_in) in
let new_R_ctr1_mux_sel = (r_cir_wr01B ∨ ((ELEMENT new_R_gcr (16)) ∧ R_ctr1_cry)) in
let new_R_ctr1_irden = (r_readB ∧ (r_reg_sel = (WORDN 9))) in
let new_R_ctr1 = ((R_ctr1_mux_sel) => R_ctr1_in | R_ctr1_new) in
let new_R_ctr1_new = ((R_ctr0_cry) => (INCN 31 R_ctr1) | R_ctr1) in
let new_R_ctr1_cry = ((ONES 31 R_ctr1) ∧ R_ctr0_cry) in
let new_R_ctr1_out = ((R_cntlatch_del) => R_ctr1_new | R_ctr1_out) in
let new_R_ctr1_orden = (r_readB ∧ (r_reg_sel = (WORDN 13))) in
let new_R_ctr2_in = ((r_writeB ∧ (r_reg_sel = (WORDN 10))) => i_ad | R_ctr2_in) in
let new_R_ctr2_mux_sel = ((r_cir_wr23B ∨ ((ELEMENT new_R_gcr (20)) ∧ R_ctr3_cry))) in
let new_R_ctr2_irden = (r_readB ∧ (r_reg_sel = (WORDN 10))) in
let new_R_ctr2 = ((R_ctr2_mux_sel) => R_ctr2_in | R_ctr2_new) in
let new_R_ctr2_new = (((ELEMENT new_R_gcr (23))) => (INCN 31 R_ctr2) | R_ctr2) in
let new_R_ctr2_cry = ((ONES 31 R_ctr2) ∧ (ELEMENT new_R_gcr (23))) in
let new_R_ctr2_out = ((r_fsm_cntlatch) => R_ctr2_new | R_ctr2_out) in
let new_R_ctr2_orden = (r_readB ∧ (r_reg_sel = (WORDN 14))) in
let new_R_ctr3_in = ((r_writeB ∧ (r_reg_sel = (WORDN 11))) => i_ad | R_ctr3_in) in
let new_R_ctr3_mux_sel = ((r_cir_wr23B ∨ ((ELEMENT new_R_gcr (20)) ∧ R_ctr3_cry))) in
let new_R_ctr3_irden = (r_readB ∧ (r_reg_sel = (WORDN 11))) in
let new_R_ctr3 = ((R_ctr3_mux_sel) => R_ctr3_in | R_ctr3_new) in
let new_R_ctr3_new = ((R_ctr2_cry) => (INCN 31 R_ctr3) | R_ctr3) in
let new_R_ctr3_cry = ((ONES 31 R_ctr3) ∧ R_ctr3_cry) in
let new_R_ctr3_out = ((R_cntlatch_del) => R_ctr3_new | R_ctr3_out) in
let new_R_ctr3_orden = (r_readB ∧ (r_reg_sel = (WORDN 15))) in
let new_R_icr_load = (r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let new_R_icr_old =
        ((r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => R_icr | R_icr_old) in
let new_R_icr_mask =
        ((r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => i_ad | R_icr_mask) in
let new_R_icr_rden = ((new_R_fsm_state = RA) ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let r_int0_en = (((ELEMENT R_icr (0)) ∧ (ELEMENT R_icr (8))) ∨
                 ((ELEMENT R_icr (1)) ∧ (ELEMENT R_icr (9))) ∨
                 ((ELEMENT R_icr (2)) ∧ (ELEMENT R_icr (10))) ∨
                 ((ELEMENT R_icr (3)) ∧ (ELEMENT R_icr (11))) ∨
                 ((ELEMENT R_icr (4)) ∧ (ELEMENT R_icr (12))) ∨
                 ((ELEMENT R_icr (5)) ∧ (ELEMENT R_icr (13))) ∨
                 ((ELEMENT R_icr (6)) ∧ (ELEMENT R_icr (14))) ∨
                 ((ELEMENT R_icr (7)) ∧ (ELEMENT R_icr (15)))) in
let new_R_int0_dis = r_int0_en in
let r_int3_en = (((ELEMENT R_icr (16)) ∧ (ELEMENT R_icr (24))) ∨
                 ((ELEMENT R_icr (17)) ∧ (ELEMENT R_icr (25))) ∨
                 ((ELEMENT R_icr (18)) ∧ (ELEMENT R_icr (26))) ∨
                 ((ELEMENT R_icr (19)) ∧ (ELEMENT R_icr (27))) ∨
                 ((ELEMENT R_icr (20)) ∧ (ELEMENT R_icr (28))) ∨
                 ((ELEMENT R_icr (21)) ∧ (ELEMENT R_icr (29))) ∨
                 ((ELEMENT R_icr (22)) ∧ (ELEMENT R_icr (30))) ∨
                 ((ELEMENT R_icr (23)) ∧ (ELEMENT R_icr (31)))) in
let new_R_int3_dis = r_int3_en in

let new_S_soft_shot_del = (~gcrh ∧ gcrl) in
let s_soft_cnt_out =
      ((s_fsm_srs) =>
         ((gcrl ∧ ~gcrh ∧ ~S_soft_shot_del) => (WORDN 1) | (WORDN 0)) |
         ((gcrl ∧ ~gcrh ∧ ~S_soft_shot_del) => (INCN 2 S_soft_cnt) | S_soft_cnt)) in
let new_S_soft_cnt = ((~gcrh ∧ ~gcrl) => (WORDN 0) | s_soft_cnt_out) in
let s_delay_out =
      ((s_fsm_src ∨ (s_fsm_scs ∧ (ELEMENT S_delay (6)))) =>
      ((s_fsm_sec) => (WORDN 1) | (WORDN 0)) |
      ((s_fsm_sec) => (INCN 17 S_delay) | S_delay)) in
let new_S_delay = s_delay_out in
let s_cpu0_ok = (s_fsm_sc0f ∧ Failure0_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu1_ok = (s_fsm_sc1f ∧ Failure1_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let new_S_pmm_fail =
      ((s_fsm_sb ∧ ~s_fsm_spmf) => T |
      ((~s_fsm_sb ∧ s_fsm_spmf) => F |
      ((~s_fsm_sb ∧ ~s_fsm_spmf) => S_pmm_fail | ARB))) in
let new_S_cpu0_fail =
      ((s_fsm_sb ∧ ~(s_cpu0_ok ∨ Bypass)) => T |
      ((~s_fsm_sb ∧ (s_cpu0_ok ∨ Bypass)) => F |
      ((~s_fsm_sb ∧ ~(s_cpu0_ok ∨ Bypass)) => S_cpu0_fail | ARB))) in
let new_S_cpu1_fail =
      ((s_fsm_sb ∧ ~(s_cpu1_ok ∨ Bypass)) => T |
       ((~s_fsm_sb ∧ (s_cpu1_ok ∨ Bypass)) => F |
       ((~s_fsm_sb ∧ ~(s_cpu1_ok ∨ Bypass)) => S_cpu1_fail | ARB))) in
let new_S_piu_fail =
      ((s_fsm_sb ∧ ~(s_fsm_spf ∨ Bypass)) => T |
       ((~s_fsm_sb ∧ (s_fsm_spf ∨ Bypass)) => F |
       ((~s_fsm_sb ∧ ~(s_fsm_spf ∨ Bypass)) => S_piu_fail | ARB))) in
let s_cpu0_select = ((s_fsm_sn ∨ s_fsm_so) ∧ ~S_cpu0_fail) in
let s_cpu1_select = ((s_fsm_sn ∨ s_fsm_so) ∧ S_cpu0_fail ∧ ~S_cpu1_fail) in
let new_S_bad_cpu0 =
      ((s_fsm_sb ∧ ~s_cpu0_select) => T |
       ((~s_fsm_sb ∧ s_cpu0_select) => F |
       ((~s_fsm_sb ∧ ~s_cpu0_select) => S_bad_cpu0 | ARB))) in
```

```
  let new_S_bad_cpu1 =
        ((s_fsm_sb ∧ ~s_cpu1_select) => T |
         ((~s_fsm_sb ∧ s_cpu1_select) => F |
         ((~s_fsm_sb ∧ ~s_cpu1_select) => S_bad_cpu1 | ARB))) in
  let new_S_reset_cpu0 = (new_S_bad_cpu0 ∧ s_fsm_src0) in
  let new_S_reset_cpu1 = (new_S_bad_cpu1 ∧ s_fsm_src1) in
  let new_S_cpu_hist = (S_reset_cpu0 ∧ S_reset_cpu1 ∧ Bypass) in
  let ss0 = (ALTER ARBN (0) ((new_S_fsm_state = SS) ∨ (new_S_fsm_state = SSTOP)
                              ∨ (new_S_fsm_state = SCS) ∨ (new_S_fsm_state = SN)
                              ∨ (new_S_fsm_state = SO))) in
  let ss1 = (ALTER ss0 (1) ((new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
                            ∨ (new_S_fsm_state = SC1I) ∨ (new_S_fsm_state = SC1F)
                            ∨ (new_S_fsm_state = SS) ∨ (new_S_fsm_state = SSTOP)
                            ∨ (new_S_fsm_state = SCS))) in
  let ss2 = (ALTER ss1 (2) ((new_S_fsm_state = SPF) ∨ (new_S_fsm_state = SC0I)
                            ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
                            ∨ (new_S_fsm_state = SSTOP) ∨ (new_S_fsm_state = SO))) in
  let ss3 = (ALTER ss2 (3) ((new_S_fsm_state = SRA) ∨ (new_S_fsm_state = SPF)
                            ∨ (new_S_fsm_state = ST) ∨ (new_S_fsm_state = SC1I)
                            ∨ (new_S_fsm_state = SCS) ∨ (new_S_fsm_state = SN)
                            ∨ (new_S_fsm_state = SO))) in
let s_state = ss3 in
  let sr28 = (ALTER ARBN (28) new_M_parity) in
  let sr28_25 = (MALTER sr28 (27,25) new_C_ss) in
  let sr28_24 = (ALTER sr28_25 (24) new_C_parity) in
  let sr28_22 = (MALTER sr28_24 (23,22) ChannelID) in
  let sr28_16 = (MALTER sr28_22 (21,16) Id) in
  let sr28_12 = (MALTER sr28_16 (15,12) s_state) in
  let sr28_9 = (ALTER sr28_12 (9) new_S_pmm_fail) in
  let sr28_8 = (ALTER sr28_9 (8) new_S_piu_fail) in
  let sr28_3 = (ALTER sr28_8 (3) new_S_reset_cpu1) in
  let sr28_2 = (ALTER sr28_3 (2) new_S_reset_cpu0) in
  let sr28_1 = (ALTER sr28_2 (1) new_S_cpu1_fail) in
  let sr28_0 = (ALTER sr28_1 (0) new_S_cpu0_fail) in
  let new_R_sr = ((r_fsm_cntlatch) => sr28_0 | R_sr) in
  let new_R_sr_rden = (r_readB ∧ (r_reg_sel = (WORDN 4))) in

  let new_P_fsm_rst = reset_piu in
  let new_P_fsm_sack = p_sack in
  let new_P_fsm_cgnt_ = ~(new_C_mfsm_state = CMA3) in
  let new_P_fsm_hold_ = new_C_holdA_ in
  let new_C_mfsm_D = ClkD in
  let new_C_mfsm_rst = reset_cport in
  let new_C_mfsm_crqt_ = ~(new_P_dest1 ∧ new_P_rqt) in
  let new_C_mfsm_hold_ = new_C_holdA_ in
  let new_C_mfsm_ss = CB_ss_in in
  let new_C_mfsm_invalid = piu_invalid in
  let new_C_sfsm_D = ClkD in
  let new_C_sfsm_rst = reset_cport in
  let new_C_sfsm_hlda_ = ~(new_P_fsm_state = PH) in
  let new_C_sfsm_ms = CB_ms_in in
  let new_C_efsm_cale_ = i_cale_ in
  let new_C_efsm_last_ = i_last_ in
  let new_C_efsm_male_ = i_male_ in
```

```
    let new_C_efsm_rale_ = i_rale_ in
    let new_C_efsm_srdy_ = i_srdy_ in
    let new_C_efsm_rst = reset_cport in
    let new_M_fsm_male_ = i_male_ in
    let new_M_fsm_last_ = i_last_ in
    let new_M_fsm_mrdy_ = ((~(P_fsm_state = PH)) => F | C_mrdy_del_) in
    let new_M_fsm_rst = reset_piu in
    let new_R_fsm_ale_ = i_rale_ in
    let new_R_fsm_mrdy_ = ((~(P_fsm_state = PH)) => F | C_mrdy_del_) in
    let new_R_fsm_last_ = i_last_ in
    let new_R_fsm_rst = reset_piu in
    let new_S_fsm_rst = Rst in
    let new_S_fsm_delay6 = (ELEMENT s_delay_out (6)) in
    let new_S_fsm_delay17 = ((Test) => (ELEMENT s_delay_out (6)) | (ELEMENT s_delay_out (17))) in
    let new_S_fsm_bothbad = (new_S_cpu0_fail /\ new_S_cpu1_fail) in
    let new_S_fsm_bypass = Bypass in

    (new_P_addr, new_P_dest1, new_P_be_, new_P_wr, new_P_fsm_state, new_P_fsm_rst, new_P_fsm_sack,
     new_P_fsm_cgnt_, new_P_fsm_hold_, new_P_rqt, new_P_size, new_P_down, new_P_lock_, new_P_lock_inh_,
     new_P_male_, new_P_rale_,
     new_C_mfsm_state, new_C_mfsm_D, new_C_mfsm_rst, new_C_mfsm_crqt_, new_C_mfsm_hold_, new_C_mfsm_ss,
     new_C_mfsm_invalid, new_C_sfsm_state, new_C_sfsm_D, new_C_sfsm_rst, new_C_sfsm_hlda_, new_C_sfsm_ms,
     new_C_efsm_state, new_C_efsm_cale_, new_C_efsm_last_, new_C_efsm_male_, new_C_efsm_rale_, new_C_efsm_srdy_,
     new_C_efsm_rst, new_C_wr, new_C_sizewrbe, new_C_clkA, new_C_last_in_, new_C_lock_in_, new_C_ss,
     new_C_last_out_, new_C_hold_, new_C_holdA_, new_C_cout_0_le_del, new_C_cin_2_le, new_C_mrdy_del_,
     new_C_iad_en_s_del, new_C_iad_en_s_delA, new_C_wrdy, new_C_rrdy, new_C_parity, new_C_source, new_C_data_in,
     new_C_iad_out, new_C_iad_in, new_C_a1a0, new_C_a3a2,
     new_M_fsm_state, new_M_fsm_male_, new_M_fsm_last_, new_M_fsm_mrdy_, new_M_fsm_rst, new_M_count,
     new_M_se, new_M_wr, new_M_addr, new_M_be, new_M_rdy, new_M_wwdel, new_M_parity, new_M_rd_data,
     new_M_detect,
     new_R_fsm_state, new_R_fsm_ale_, new_R_fsm_mrdy_, new_R_fsm_last_, new_R_fsm_rst, new_R_ctr0_in,
     new_R_ctr0_mux_sel, new_R_ctr0, new_R_ctr0_irden, new_R_ctr0_new, new_R_ctr0_cry, new_R_ctr0_out,
     new_R_ctr0_orden, new_R_ctr1_in, new_R_ctr1_mux_sel, new_R_ctr1, new_R_ctr1_irden, new_R_ctr1_new,
     new_R_ctr1_cry,
     new_R_ctr1_out, new_R_ctr1_orden, new_R_ctr2_in, new_R_ctr2_mux_sel, new_R_ctr2, new_R_ctr2_irden,
     new_R_ctr2_new,
     new_R_ctr2_cry, new_R_ctr2_out, new_R_ctr2_orden, new_R_ctr3_in, new_R_ctr3_mux_sel, new_R_ctr3,
     new_R_ctr3_irden,
     new_R_ctr3_new, new_R_ctr3_cry, new_R_ctr3_out, new_R_ctr3_orden, new_R_icr_load, new_R_icr_old,
     new_R_icr_mask,
     new_R_icr_rden, new_R_icr, new_R_ccr, new_R_ccr_rden, new_R_gcr, new_R_gcr_rden, new_R_sr, new_R_sr_rden,
     new_R_int0_dis, new_R_int3_dis, new_R_c01_cout_del, new_R_int1_en, new_R_c23_cout_del, new_R_int2_en,
     new_R_wr,
     new_R_cntlatch_del, new_R_srdy_del_, new_R_reg_sel, new_R_busA_latch,
     new_S_fsm_state, new_S_fsm_rst, new_S_fsm_delay6, new_S_fsm_delay17, new_S_fsm_bothbad,
     new_S_fsm_bypass, new_S_soft_shot_del, new_S_soft_cnt, new_S_delay, new_S_bad_cpu0, new_S_bad_cpu1,
     new_S_reset_cpu0, new_S_reset_cpu1, new_S_cpu_hist, new_S_pmm_fail, new_S_cpu0_fail, new_S_cpu1_fail,
     new_S_piu_fail)"
   );;

%------------------------------------------------------------------------------------------------
  Output definition for EXEC instruction.
------------------------------------------------------------------------------------------------%
```

```
let piuEXEC_out_def = new_definition
 ('piuEXEC_out',

    "! (rep:^rep_ty)
      (P_fsm_state :pfsm_ty)
      (P_addr P_be_ P_size :wordn)
      (P_dest1 P_wr P_fsm_rst P_fsm_sack P_fsm_cgnt_ P_fsm_hold_ P_rqt P_down P_lock_
       P_lock_inh_ P_male_ P_rale_ :bool)
      (C_mfsm_state :cmfsm_ty) (C_sfsm_state :csfsm_ty) (C_efsm_state :cefsm_ty)
      (C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss C_source C_data_in C_iad_out C_iad_in C_a1a0 C_a3a2 :wordn)
      (C_mfsm_D C_mfsm_rst C_mfsm_crqt_ C_mfsm_hold_ C_mfsm_invalid C_sfsm_D C_sfsm_rst C_sfsm_hlda_
       C_efsm_cale_ C_efsm_last_ C_efsm_male_ C_efsm_rale_ C_efsm_srdy_ C_efsm_rst
       C_wr C_clkA C_last_in_ C_lock_in_ C_last_out_ C_hold_ C_holdA_ C_cout_0_le_del C_cin_2_le
       C_mrdy_del_ C_iad_en_s_del C_iad_en_s_delA C_wrdy C_rrdy C_parity :bool)
      (M_fsm_state :mfsm_ty)
      (M_count M_addr M_be M_rd_data M_detect :wordn)
      (M_fsm_male_ M_fsm_last_ M_fsm_mrdy_ M_fsm_rst M_se M_wr M_rdy M_wwdel M_parity :bool)
      (R_fsm_state :rfsm_ty)
      (R_ctr0_in R_ctr0 R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1 R_ctr1_new R_ctr1_out R_ctr2_in R_ctr2 R_ctr2_new
       R_ctr2_out R_ctr3_in R_ctr3 R_ctr3_new R_ctr3_out R_icr_old R_icr_mask R_icr R_ccr R_gcr R_sr
       R_reg_sel R_busA_latch :wordn)
      (R_fsm_ale_ R_fsm_mrdy_ R_fsm_last_ R_fsm_rst R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_orden
       R_ctr1_mux_sel R_ctr1_irden R_ctr1_cry R_ctr1_orden R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_orden
       R_ctr3_mux_sel R_ctr3_irden R_ctr3_cry R_ctr3_orden R_icr_load R_icr_rden R_ccr_rden R_gcr_rden R_sr_rden
       R_int0_dis R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatch_del R_srdy_del_ :bool)
      (S_fsm_state :sfsm_ty)
      (S_soft_cnt S_delay :wordn)
      (S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsm_bothbad S_fsm_bypass S_soft_shot_del S_bad_cpu0 S_bad_cpu1
       S_reset_cpu0 S_reset_cpu1 S_cpu_hist S_pmm_fail S_cpu0_fail S_cpu1_fail S_piu_fail :bool)

      (L_ad_in L_be_ :wordn)
      (ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ :bool)
      (CB_rqt_in_ CB_ad_in CB_ms_in CB_ss_in Id ChannelID :wordn)
      (ClkD :bool)
      (MB_data_in :wordn)
      (Edac_en_ :bool)
      (Bypass Test Failure0_ Failure1_ :bool) .
  piuEXEC_out rep
              (P_addr, P_dest1, P_be_, P_wr, P_fsm_state, P_fsm_rst, P_fsm_sack, P_fsm_cgnt_, P_fsm_hold_,
               P_rqt, P_size, P_down, P_lock_, P_lock_inh_, P_male_, P_rale_,
               C_mfsm_state, C_mfsm_D, C_mfsm_rst, C_mfsm_crqt_, C_mfsm_hold_, C_mfsm_ss, C_mfsm_invalid,
               C_sfsm_state, C_sfsm_D, C_sfsm_rst, C_sfsm_hlda_, C_sfsm_ms,
               C_efsm_state, C_efsm_cale_, C_efsm_last_, C_efsm_male_, C_efsm_rale_, C_efsm_srdy_, C_efsm_rst,
               C_wr, C_sizewrbe, C_clkA, C_last_in_, C_lock_in_, C_ss, C_last_out_,
               C_hold_, C_holdA_, C_cout_0_le_del, C_cin_2_le, C_mrdy_del_, C_iad_en_s_del, C_iad_en_s_delA,
               C_wrdy, C_rrdy, C_parity, C_source, C_data_in, C_iad_out, C_iad_in, C_a1a0,C_a3a2,
               M_fsm_state, M_fsm_male_, M_fsm_last_, M_fsm_mrdy_, M_fsm_rst, M_count, M_se, M_wr, M_addr,
               M_be, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect,
               R_fsm_state, R_fsm_ale_, R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_ctr0_in, R_ctr0_mux_sel, R_ctr0,
               R_ctr0_irden, R_ctr0_new, R_ctr0_cry, R_ctr0_out, R_ctr0_orden, R_ctr1_in, R_ctr1_mux_sel,
               R_ctr1, R_ctr1_irden, R_ctr1_new, R_ctr1_cry, R_ctr1_out, R_ctr1_orden, R_ctr2_in, R_ctr2_mux_sel,
               R_ctr2, R_ctr2_irden, R_ctr2_new, R_ctr2_cry, R_ctr2_out, R_ctr2_orden, R_ctr3_in, R_ctr3_mux_sel,
               R_ctr3, R_ctr3_irden, R_ctr3_new, R_ctr3_cry, R_ctr3_out, R_ctr3_orden, R_icr_load, R_icr_old,
               R_icr_mask, R_icr_rden, R_icr, R_ccr, R_ccr_rden, R_gcr, R_gcr_rden, R_sr, R_sr_rden, R_int0_dis,
            R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_,
            R_reg_sel, R_busA_latch,
            S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass, S_soft_shot_del,
            S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1, S_cpu_hist, S_pmm_fail,
            S_cpu0_fail, S_cpu1_fail, S_piu_fail)

           (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_,
            CB_rqt_in_, CB_ad_in, CB_ms_in, CB_ss_in, ClkD, Id, ChannelID,
            MB_data_in, Edac_en_,
            Bypass, Test, Failure0_, Failure1_) =

let new_P_fsm_state =
      ((P_fsm_rst) => PA |
       ((P_fsm_state = PH) => ((~P_fsm_hold_) => PH | PA) |
       ((P_fsm_state = PA) =>
          (((P_rqt ∧ ~P_dest1) ∨ (P_rqt ∧ P_dest1 ∧ ~P_fsm_cgnt_)) => PD |
          ((~P_fsm_hold_ ∧ P_lock_) => PH | PA)) |
       ((P_fsm_state = PD) =>
          (((P_fsm_sack ∧ P_fsm_hold_) ∨ (P_fsm_sack ∧ ~P_fsm_hold_ ∧ ~P_lock_)) => PA |
          ((P_fsm_sack ∧ ~P_fsm_hold_ ∧ P_lock_) => PH | PD)) | P_ILL)))) in

let c_write = (((~(C_mfsm_state = CMI)) ∧ (~(C_mfsm_state = CMR))) => C_wr | (ELEMENT C_sizewrbe (5))) in
let c_busy = (~((SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
let c_grant = ((((SUBARRAY Id (1,0)) = (WORDN 0)) ∧ ~(ELEMENT CB_rqt_in_ (0)))
            ∨ (((SUBARRAY Id (1,0)) = (WORDN 1)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                  ∧ (ELEMENT CB_rqt_in_ (1)))
            ∨ (((SUBARRAY Id (1,0)) = (WORDN 2)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                  ∧ (ELEMENT CB_rqt_in_ (1))
                                                  ∧ (ELEMENT CB_rqt_in_ (2)))
            ∨ (((SUBARRAY Id (1,0)) = (WORDN 3)) ∧ ~(ELEMENT CB_rqt_in_ (0))
                                                  ∧ (ELEMENT CB_rqt_in_ (1))
                                                  ∧ (ELEMENT CB_rqt_in_ (2))
                                                  ∧ (ELEMENT CB_rqt_in_ (3)))) in
let c_addressed = (Id = (SUBARRAY C_source (15,10))) in
let new_C_mfsm_state =
      ((C_mfsm_rst) => CMI |
      ((C_mfsm_state = CMI) =>
            (C_mfsm_D ∧ ~C_mfsm_crqt_ ∧ ~c_busy ∧ ~C_mfsm_invalid) => CMR | CMI |
      ((C_mfsm_state = CMR) => (C_mfsm_D ∧ c_grant ∧ C_mfsm_hold_) => CMA3 | CMR |
      ((C_mfsm_state = CMA3) => ((C_mfsm_D) => CMA1 | CMA3) |
      ((C_mfsm_state = CMA1) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA0 |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA1 |
      ((C_mfsm_state = CMA0) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMA2 |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA0 |
      ((C_mfsm_state = CMA2) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD1 |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMA2 |
      ((C_mfsm_state = CMD1) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY)) => CMD0 |
            (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD1 |
      ((C_mfsm_state = CMD0) =>
            (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ C_last_in_) => CMD1 |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_last_in_) => CMW |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT | CMD0 |
        ((C_mfsm_state = CMW) =>
                (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SACK) ∧ C_lock_in_) => CMI |
                (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ~C_lock_in_ ∧ ~C_mfsm_crqt_) => CMA3 | CMW |
        ((~C_last_in_) => CMI | CMABT)))))))))) in

let new_C_sfsm_state =
        ((C_sfsm_rst) => CSI |
                (C_sfsm_state = CSI) =>
                   ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ c_addressed) => CSA1 | CSI) |
         (C_sfsm_state = CSL) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ c_addressed) => CSA1 |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ~c_grant ∧ ~c_addressed) => CSI |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSL) |
         (C_sfsm_state = CSA1) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSA0 |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA1) |
         (C_sfsm_state = CSA0) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ C_sfsm_hlda_) => CSA0W |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0) |
         (C_sfsm_state = CSA0W) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ~C_sfsm_hlda_) => CSALE |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSA0W) |
         (C_sfsm_state = CSALE) =>
                ((C_sfsm_D ∧ c_write ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
                (C_sfsm_D ∧ ~c_write ∧ (C_sfsm_ms = ^MRDY)) => CSRR |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSALE) |
         (C_sfsm_state = CSRR) =>
                ((C_sfsm_D ∧ ~(C_sfsm_ms = ^MABORT)) => CSD1 |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSRR) |
         (C_sfsm_state = CSD1) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD0 |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD1) |
         (C_sfsm_state = CSD0) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MEND)) => CSACK |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD1 |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSD0) |
         (C_sfsm_state = CSACK) =>
                ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSL |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MWAIT)) => CSI |
                (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT | CSACK) |
         (C_sfsm_D) => CSI | CSABT) in

let new_C_efsm_state =
        ((C_efsm_rst) => CEI |
         (C_efsm_state = CEI) => ((~C_efsm_cale_) => CEE | CEI) |
         ((~C_efsm_last_ ∧ ~C_efsm_srdy_) ∨ ~C_efsm_male_ ∨ ~C_efsm_rale_) => CEI | CEE) in

let m_bw = ((~(M_be = (WORDN 15))) ∧ M_wr ∧ (~(M_fsm_state = MI))) in
let m_ww = ((M_be = (WORDN 15)) ∧ M_wr ∧ (~(M_fsm_state = MI))) in
let new_M_fsm_state =
        ((M_fsm_rst) => MI |
         ((M_fsm_state = MI) => ((~M_fsm_male_) => MA | MI) |
         ((M_fsm_state = MA) =>
                ((~M_fsm_mrdy_ ∧ m_ww) => MW |
                ((~M_fsm_mrdy_ ∧ ((~M_wr ∧ (~(M_fsm_state = MI))) ∨ m_bw)) => MR | MA)) |
         ((M_fsm_state = MR) =>
                ((m_bw ∧ (M_count = (WORDN 0))) => MBW |
                ((M_fsm_last_ ∧ ~M_wr ∧ (~(M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) => MA |
                ((~M_fsm_last_ ∧ ~M_wr ∧ (~(M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) => MRR | MR))) |
         ((M_fsm_state = MRR) => MI |
         ((M_fsm_state = MW) =>
                ((~M_fsm_last_ ∧ (M_count = (WORDN 0))) => MI |
                ((M_fsm_last_ ∧ (M_count = (WORDN 0))) => MA | MW)) |
         ((M_fsm_state = MBW) => MW | M_ILL))))))) in

let new_R_fsm_state =
        ((R_fsm_rst) => RI |
         ((R_fsm_state = RI) => ((~R_fsm_ale_) => RA | RI) |
         ((R_fsm_state = RA) => ((~R_fsm_mrdy_) => RD | RA) |
         ((~R_fsm_last_) => RI | RA)))) in
let r_fsm_cntlatch = ((R_fsm_state = RI) ∧ ~R_fsm_ale_) in
let r_fsm_srdy_ = ~((R_fsm_state = RA) ∧ ~R_fsm_mrdy_) in

let new_S_fsm_state =
        ((S_fsm_rst) => SSTART |
        ((S_fsm_state = SSTART) => SRA |
        ((S_fsm_state = SRA) => ((S_fsm_delay6) => ((S_fsm_bypass) => SO | SPF) | SRA) |
        ((S_fsm_state = SPF) => SC0I |
        ((S_fsm_state = SC0I) => ((S_fsm_delay17) => SC0F | SC0I) |
        ((S_fsm_state = SC0F) => ST |
        ((S_fsm_state = ST) => SC1I |
        ((S_fsm_state = SC1I) => ((S_fsm_delay17) => SC1F | SC1I) |
        ((S_fsm_state = SC1F) => SS |
        ((S_fsm_state = SS) => ((S_fsm_bothbad) => SSTOP | SCS) |
        ((S_fsm_state = SSTOP) => SSTOP |
        ((S_fsm_state = SCS) => ((S_fsm_delay6) => SN | SCS) |
        ((S_fsm_state = SN) => ((S_fsm_delay17) => SO | SN) |
        ((S_fsm_state = SO) => SO | S_ILL)))))))))))))) in
let s_fsm_sn = (new_S_fsm_state = SN) in
let s_fsm_so = (new_S_fsm_state = SO) in
let reset_cport = (((~(new_S_fsm_state = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let s_fsm_sdi = (((~(new_S_fsm_state = SO)) ∧ (~(S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let reset_piu = ((new_S_fsm_state = SSTART) ∨ (new_S_fsm_state = SRA)
                ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
                ∨ (new_S_fsm_state = SC1F) ∨ (new_S_fsm_state = SS) ∨ (new_S_fsm_state = SCS)) in
let s_fsm_src0 = ((~(new_S_fsm_state = SPF)) ∧ (~(new_S_fsm_state = SC0I))) in
let s_fsm_src1 = ((~(new_S_fsm_state = ST)) ∧ (~(new_S_fsm_state = SC1I))) in
let s_fsm_spf = ((S_fsm_state = SRA) ∧ S_fsm_delay6 ∧ ~S_fsm_rst) in
let s_fsm_sc0f = (new_S_fsm_state = SC0F) in
let s_fsm_sc1f = (new_S_fsm_state = SC1F) in
let s_fsm_spmf = (new_S_fsm_state = SO) in
let s_fsm_sb = (new_S_fsm_state = SSTART) in
let s_fsm_src = ((new_S_fsm_state = SSTART) ∨ ((S_fsm_state = SRA) ∧ S_fsm_delay6)
                 ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST) ∨ (new_S_fsm_state = SC1F)
                ∨ (new_S_fsm_state = SS) ∨ ((S_fsm_state = SCS) ∧ S_fsm_delay6)) in
let s_fsm_sec = (((~(new_S_fsm_state = SSTOP)) ∧ (~(new_S_fsm_state = SO))) ∨ (S_fsm_state = SN)) in
let s_fsm_srs = (((S_fsm_state = SPF) ∧ ~S_fsm_rst) ∨ ((S_fsm_state = ST) ∧ ~S_fsm_rst)) in
let s_fsm_scs = (new_S_fsm_state = SCS) in

let new_P_addr = ((~P_rqt) => (SUBARRAY L_ad_in (25,0)) | P_addr) in
let new_P_dest1 = ((~P_rqt) => (ELEMENT L_ad_in (31)) | P_dest1) in
let new_P_be_ = ((~P_rqt) => L_be_ | P_be_) in
let new_P_wr = ((~P_rqt) => L_wr | P_wr) in
let new_P_size =
   ((~P_rqt) => (SUBARRAY L_ad_in (1,0)) |
((P_down) => (DECN 1 P_size) | P_size)) in
let new_C_holdA_ = ((ClkD) => C_hold_ | C_holdA_) in
let i_cale_ = ~((new_C_mfsm_state = CMA3) ∧ (new_P_fsm_state = PA) ∧ new_C_holdA_) in
let c_srdy_en = ((new_C_efsm_state = CEE) ∨ (C_efsm_state = CEE)) in
let new_M_count =
      (((new_M_fsm_state = MA) ∨ (new_M_fsm_state = MBW)) => ((M_se) => (WORDN 1) | (WORDN 2)) |
       (((new_M_fsm_state = MW) ∨ (new_M_fsm_state = MR)) => (DECN 2 M_count) | M_count)) in
let m_rdy = (((new_M_fsm_state = MW) ∧ (new_M_count = (WORDN 0)))
             ∨ ((new_M_fsm_state = MR) ∧ (new_M_count = (WORDN 0)) ∧ ~M_wr)) in
let m_srdy_ = ~((M_rdy ∧ ~M_wr) ∨ (m_rdy ∧ M_wr)) in
let i_srdy_ = ((~i_cale_ ∨ c_srdy_en) => ~(C_wrdy ∨ C_rrdy ∨ (new_C_mfsm_state = CMABT)) |
               ~(new_M_fsm_state = MI) => m_srdy_ |
               ((new_R_fsm_state = RA) ∨ (new_R_fsm_state = RD)) => ~((R_fsm_state = RA) ∧ (new_R_fsm_state = RD)) |
               ARB) in
let p_ale = (~L_ads_ ∧ L_den_) in
let p_sack = ((P_size = ((P_down) => (WORDN 1) | (WORDN 0))) ∧ ~i_srdy_ ∧ (new_P_fsm_state = PD)) in
let new_P_rqt =
   ((p_ale ∧ ~(p_sack ∨ reset_piu)) => T |
   ((~p_ale ∧ (p_sack ∨ reset_piu)) => F |
   ((~p_ale ∧ ~(p_sack ∨ reset_piu)) => P_rqt | ARB))) in
let new_P_down = (~i_srdy_ ∧ (new_P_fsm_state = PD)) in
let new_P_male_ = ((new_P_fsm_state = PA) =>
   ~(~new_P_dest1 ∧ (~((SUBARRAY new_P_addr (25,24)) = (WORDN 3))) ∧ new_P_rqt) | P_male_) in
let new_P_rale_ = ((new_P_fsm_state = PA) =>
   ~(~new_P_dest1 ∧ ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) ∧ new_P_rqt) | P_rale_) in
let new_P_lock_ =
   ((reset_piu) => T |
   ((new_P_fsm_state = PD) => L_lock_ | P_lock_)) in
let new_P_lock_inh_ =
   ((reset_piu) => T |
   ((~new_P_male_ ∨ ~new_P_rale_) => L_lock_ | P_lock_inh_)) in
let pod31_27 = (MALTER ARBN (31,27) new_P_be_) in
let pod31_26 = (ALTER pod31_27 (26) F) in
let pod31_24 = (MALTER pod31_26 (25,24) (SUBARRAY new_P_addr (1,0))) in
let new_C_iad_en_s_delA = ((ClkD) => C_iad_en_s_del | C_iad_en_s_delA) in
let new_C_sizewrbe = ((reset_cport) => (WORDN 0) |
                        (((new_C_sfsm_state = CSA0) ∧ C_clkA) => (SUBARRAY C_data_in (31,22)) | C_sizewrbe)) in
let c_new_write = (((~(new_C_mfsm_state = CMI)) ∧ (~(new_C_mfsm_state = CMR))) =>
                   C_wr | (ELEMENT new_C_sizewrbe (5))) in
let new_C_iad_out = ((C_cin_2_le) => C_data_in | C_iad_out) in
let r_reg_sel = ((~R_srdy_del_) => (INCN 3 R_reg_sel) | R_reg_sel) in
let new_R_icr =
      ((R_icr_load) =>
            ((~(r_reg_sel = (WORDN 1))) => (Andn rep (R_icr_old, R_icr_mask)) | (Orn rep (R_icr_old, R_icr_mask))) |
          R_icr) in
let new_R_busA_latch =
        ((R_ctr0_irden) => R_ctr0_in |
         ((R_ctr0_orden) => R_ctr0_out |
         ((R_ctr1_irden) => R_ctr1_in |
         ((R_ctr1_orden) => R_ctr1_out |
         ((R_ctr2_irden) => R_ctr2_in |
         ((R_ctr2_orden) => R_ctr2_out |
         ((R_ctr3_irden) => R_ctr3_in |
         ((R_ctr3_orden) => R_ctr3_out |
         ((R_icr_rden) => new_R_icr |
         ((R_ccr_rden) => R_ccr |
         ((R_gcr_rden) => R_gcr |
         ((R_sr_rden) => R_sr | ARB)))))))))))) in
let i_ad = ((new_P_fsm_state = PA) => pod31_24 |
            ((new_P_fsm_state = PD) ∧ new_P_wr) => L_ad_in |
            (new_C_iad_en_s_delA ∨
             ((new_C_mfsm_state = CMD1) ∧ ~c_new_write ∧ c_srdy_en) ∨
             ((new_C_mfsm_state = CMD0) ∧ ~c_new_write ∧ c_srdy_en) ∨
             ((new_C_mfsm_state = CMW) ∧ (C_mfsm_state = CMD0) ∧ ~c_new_write ∧ c_srdy_en) ∨
             ((new_C_sfsm_state = CSALE) ∧ (~(C_sfsm_state = CSALE))) ∨
             ((new_C_sfsm_state = CSALE) ∧ c_new_write) ∨
             ((new_C_sfsm_state = CSD1) ∧ c_new_write ∧ (~(C_sfsm_state = CSRR))) ∨
             ((new_C_sfsm_state = CSD0) ∧ c_new_write) ∨
             ((new_C_sfsm_state = CSACK) ∧ c_new_write)) => new_C_iad_out |
            (M_wr ∧ ~(new_M_fsm_state = MI)) => M_rd_data |
            (~R_wr ∧ ((new_R_fsm_state = RA) ∨ (new_R_fsm_state = RD))) => new_R_busA_latch | ARB) in
let disable_writes = ((~(new_C_sfsm_state = CSI)) ∧ (~(new_C_sfsm_state = CSL)) ∧
                      ~((ChannelID = (WORDN 0)) ∧ (ELEMENT C_source (6))) ∧
                      ~((ChannelID = (WORDN 1)) ∧ (ELEMENT C_source (7))) ∧
                      ~((ChannelID = (WORDN 2)) ∧ (ELEMENT C_source (8))) ∧
                      ~((ChannelID = (WORDN 3)) ∧ (ELEMENT C_source (9)))) in
let i_rale_ =
        (~(new_P_fsm_state = PH) =>
         ~(~new_P_destl ∧ ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) ∧ (new_P_fsm_state = PA) ∧ new_P_rqt) |
         ~((new_C_sfsm_state = CSALE) ∧ ((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3)) ∧ C_clkA)) in
let new_R_wr = ((~i_rale_) => (ELEMENT i_ad (27)) | R_wr) in
let r_writeB = (~disable_writes ∧ new_R_wr ∧ (new_R_fsm_state = RD)) in
let r_readB = (~new_R_wr ∧ (new_R_fsm_state = RA)) in
let new_R_gcr = ((r_writeB ∧ (r_reg_sel = (WORDN 2))) => i_ad | R_gcr) in
let new_R_gcr_rden = (r_readB ∧ (r_reg_sel = (WORDN 2))) in
let gcrl = (ELEMENT new_R_gcr (0)) in
let gcrh = (ELEMENT new_R_gcr (1)) in
let reset_error = (ELEMENT new_R_gcr (24)) in
let piu_invalid = (ELEMENT new_R_gcr (28)) in
let cout_sel0 = (ALTER ARBN (0) (((new_C_sfsm_state = CSD1) ∨ (new_C_sfsm_state = CSD0)) =>
                                  (new_C_sfsm_state = CSD1) |
                                  (new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA1)
                                                            ∨ (new_C_mfsm_state = CMD1))) in
let c_cout_sel = (ALTER cout_sel0 (1) (((new_C_sfsm_state = CSD1) ∨ (new_C_sfsm_state = CSD0)) =>
                                       F |
                                       (new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA2))) in
let new_C_hold_ = (new_C_sfsm_state = CSI) in
let new_C_wr = ((~i_cale_) => (ELEMENT i_ad (27)) | C_wr) in
let new_C_clkA = ClkD in
let i_last_ =
        (~(new_P_fsm_state = PH) =>
         (P_size = ((P_down) => (WORDN 1) | (WORDN 0))) |
         C_last_out_) in
let new_C_last_in_ = ((reset_cport) => F |
                        (((new_C_mfsm_state = CMABT) ∨ (new_C_mfsm_state = CMD1) ∧ ClkD) => i_last_ |
                        C_last_in_)) in
let new_C_lock_in_ = ((reset_cport) => F |
                        ((new_C_mfsm_state = CMA1) => ~(~new_P_lock_ ∧ new_P_lock_inh_) |
                        C_lock_in_)) in
let new_C_ss = (((~(new_C_mfsm_state = CMABT)) ∧ (~(new_C_mfsm_state = CMI))) => CB_ss_in | C_ss) in
let new_C_last_out_ =
        (((new_C_sfsm_state = CSA1) ∧ ~(ClkD ∧ ((CB_ms_in = ^MEND) ∨ (CB_ms_in = ^MABORT)))) => T |
         ((~(new_C_sfsm_state = CSA1) ∧ (ClkD ∧ ((CB_ms_in = ^MEND) ∨ (CB_ms_in = ^MABORT)))) => F |
         ((~(new_C_sfsm_state = CSA1) ∧ ~(ClkD ∧ ((CB_ms_in = ^MEND) ∨ (CB_ms_in = ^MABORT)))) => C_last_out_ |
         ARB))) in
let c_srdy = (CB_ss_in = ^SRDY) in
let c_dfsm_master = ((new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA2) ∨ (new_C_mfsm_state = CMA1)
                    ∨ (new_C_mfsm_state = CMA0) ∨ (new_C_mfsm_state = CMD1) ∨ (new_C_mfsm_state = CMD0)) in
let c_dfsm_cad_en = ~((new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA1) ∨ (new_C_mfsm_state = CMA0)
                      ∨ (new_C_mfsm_state = CMA2)
                      ∨ (c_new_write ∧ ((new_C_mfsm_state = CMD1) ∨ (new_C_mfsm_state = CMD0)))
                      ∨ (~c_new_write ∧ ((new_C_sfsm_state = CSD1) ∨ (new_C_sfsm_state = CSD0)))) in
let new_C_cout_0_le_del = ((i_cale_) ∨ (i_srdy_ ∧ ~c_new_write)
                           ∨ ((new_C_mfsm_state = CMA0) ∧ c_srdy ∧ c_new_write ∧ ClkD)
                           ∨ ((new_C_mfsm_state = CMD0) ∧ c_new_write ∧ c_srdy ∧ ClkD)) in
let new_C_cin_2_le = (ClkD ∧ (((new_C_mfsm_state = CMD0) ∧ c_srdy ∧ ~c_new_write) ∨
                               ((new_C_sfsm_state = CSA0)) ∨
                               ((new_C_sfsm_state = CSD0) ∧ c_new_write))) in
let new_C_mrdy_del_ = ~((~c_new_write ∧ ClkD ∧ ((new_C_sfsm_state = CSALE) ∨ (new_C_sfsm_state = CSD1))) ∨
                         (~c_new_write ∧ C_clkA ∧ (new_C_sfsm_state = CSACK)) ∨
                         (c_new_write ∧ ClkD ∧ (new_C_sfsm_state = CSD0))) in
let new_C_iad_en_s_del = (((new_C_sfsm_state = CSALE) ∧ (~(C_sfsm_state = CSALE)))
                          ∨ ((new_C_sfsm_state = CSALE) ∧ c_new_write)
                          ∨ ((new_C_sfsm_state = CSD1) ∧ c_new_write ∧ (~(C_sfsm_state = CSRR)))
                          ∨ ((new_C_sfsm_state = CSD0) ∧ c_new_write) ∨
                           ((new_C_sfsm_state = CSACK) ∧ c_new_write)) in
let new_C_wrdy = (c_srdy ∧ c_new_write ∧ (new_C_mfsm_state = CMD1) ∧ ClkD) in
let new_C_rrdy = (c_srdy ∧ ~c_new_write ∧ (new_C_mfsm_state = CMD0) ∧ ClkD) in
let c_pe = (Par_Det rep (CB_ad_in)) in
let c_mparity = ((new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA1) ∨ (new_C_mfsm_state = CMA0)
                ∨ (new_C_mfsm_state = CMA2) ∨ (new_C_mfsm_state = CMD1) ∨ (new_C_mfsm_state = CMD0)
                ∨ (C_mfsm_state = CMA1) ∨ (C_mfsm_state = CMA0) ∨ (C_mfsm_state = CMA2)
                ∨ (C_mfsm_state = CMD1)) in
let c_sparity = ((~(new_C_sfsm_state = CSI)) ∧ (~(new_C_sfsm_state = CSACK)) ∧ (~(new_C_sfsm_state = CSABT))) in
let c_pe_cnt = (ClkD ∧ ((~(c_mparity = c_sparity)) ∨ ((SUBARRAY CB_ss_in (1,0)) = (WORDN 0)))) in
let new_C_parity =
        (((ClkD ∧ c_pe ∧ c_pe_cnt) ∧ ~reset_error) => T |
         ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ reset_error) => F |
         ((~(ClkD ∧ c_pe ∧ c_pe_cnt) ∧ ~reset_error) => C_parity | ARB))) in
let new_C_source =
        ((reset_cport) => (WORDN 0) |
     ((ClkD ∧ ((new_C_sfsm_state = CSI) ∨ (new_C_sfsm_state = CSL))) => Par_Dec rep (CB_ad_in) | C_source)) in
let data_in31_16 =
    (MALTER ARBN (31,16) ((reset_cport) => (WORDN 0) |
                         ((ClkD ∧ (((new_C_mfsm_state = CMD1) ∧ c_srdy ∧ ~c_new_write) ∨
                                   ((new_C_sfsm_state = CSA1)) ∨
                                   ((new_C_sfsm_state = CSD1) ∧ c_new_write))) => Par_Dec rep (CB_ad_in) |
                         (SUBARRAY C_data_in (31,16))))) in
let new_C_data_in =
    (MALTER data_in31_16 (15,0) ((reset_cport) => (WORDN 0) |
                               ((new_C_cin_2_le) => Par_Dec rep (CB_ad_in) |
                               (SUBARRAY C_data_in (15,0))))) in
let new_C_iad_in = ((new_C_cout_0_le_del) => i_ad | C_iad_in) in
let new_C_a1a0 =
    (((c_dfsm_master ∧ C_cout_0_le_del) ∨
     (~c_dfsm_master ∧ C_clkA ∧ (new_C_sfsm_state = CSD1))) => C_iad_in | C_a1a0) in
let new_C_a3a2 = ((new_C_mfsm_state = CMR) => R_ccr | C_a3a2) in
let i_be_ = ((new_P_fsm_state = PA) => new_P_be_ |
           (new_P_fsm_state = PD) => L_be_ | SUBARRAY new_C_sizewrbe (9,6)) in
let i_male_ =
    (~(new_P_fsm_state = PH) =>
     ~(~new_P_dest1 ∧ (~((SUBARRAY new_P_addr (25,24)) = (WORDN 3))) ∧ (new_P_fsm_state = PA) ∧ new_P_rqt) |
     ~((new_C_sfsm_state = CSALE) ∧ (~((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3))) ∧ C_clkA)) in
let new_M_se = ((~i_male_) => (ELEMENT i_ad (23)) | M_se) in
let new_M_wr = ((~i_male_) => (ELEMENT i_ad (27)) | M_wr) in
let new_M_addr =
    ((~i_male_) => (SUBARRAY i_ad (18,0)) |
     ((M_rdy) => (INCN 18 M_addr) | M_addr)) in
let new_M_be = ((~i_male_ ∨ ~m_srdy_) => (NOTN 3 i_be_) | M_be) in
let new_M_rdy = m_rdy in
let new_M_wwdel = ((new_M_fsm_state = MA) ∧ new_M_wr ∧ (new_M_be = (WORDN 15))) in
let new_M_rd_data = (((new_M_fsm_state = MR)) => (Ham_Dec rep MB_data_in) | M_rd_data) in
let new_M_detect =
    ((((new_M_fsm_state = MR) ∧ ~new_M_wr) ∨ new_M_wr ∨ (new_M_fsm_state = MI)) =>
          ((~Edac_en_) => (Ham_Det1 rep MB_data_in) | WORDN 0) | M_detect) in
let m_error = (~m_srdy_ ∧ (~(new_M_fsm_state = MI)) ∧ Ham_Det2 rep (new_M_detect, ~Edac_en_)) in
let new_M_parity =
    ((m_error ∧ ~(reset_piu ∨ reset_error)) => T |
     ((~m_error ∧ (reset_piu ∨ reset_error)) => F |
     ((~m_error ∧ ~(reset_piu ∨ reset_error)) => M_parity | ARB))) in
let new_R_cntlatch_del = r_fsm_cntlatch in
let new_R_srdy_del_ = r_fsm_srdy_ in
let new_R_reg_sel =
    ((~i_rale_) => (SUBARRAY i_ad (3,0)) |
     ((~R_srdy_del_) => (INCN 3 R_reg_sel) | R_reg_sel)) in
let r_writeA = (~disable_writes ∧ R_wr ∧ (new_R_fsm_state = RD)) in
let r_readA = (~R_wr ∧ (new_R_fsm_state = RA)) in
let r_cir_wr01A = ((r_writeA ∧ ((r_reg_sel = (WORDN 8)) ∨ (r_reg_sel = (WORDN 9))))) in
let r_cir_wr01B = ((r_writeB ∧ ((r_reg_sel = (WORDN 8)) ∨ (r_reg_sel = (WORDN 9))))) in
let r_cir_wr23A = ((r_writeA ∧ ((r_reg_sel = (WORDN 10)) ∨ (r_reg_sel = (WORDN 11))))) in
let r_cir_wr23B = ((r_writeB ∧ ((r_reg_sel = (WORDN 10)) ∨ (r_reg_sel = (WORDN 11))))) in
let new_R_ccr = ((r_writeB ∧ (r_reg_sel = (WORDN 3))) => i_ad | R_ccr) in
let new_R_ccr_rden = (r_readB ∧ (r_reg_sel = (WORDN 3))) in
let new_R_c01_cout_del = R_ctr1_cry in
let new_R_int1_en =
        ((((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
         ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => T |
         ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
         (~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => F |
         ((~((ELEMENT new_R_gcr (18)) ∧ (r_cir_wr01B ∨ (R_ctr1_cry ∧ (ELEMENT new_R_gcr (16))))) ∧
         ~(~(ELEMENT new_R_gcr (18)) ∨ ((ELEMENT new_R_gcr (17)) ∧ R_c01_cout_del))) => R_int1_en | ARB))) in
let new_R_c23_cout_del = R_ctr3_cry in
let new_R_int2_en =
        ((((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
         ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => T |
         ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
         (~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => F |
         ((~((ELEMENT new_R_gcr (22)) ∧ (r_cir_wr23B ∨ (R_ctr3_cry ∧ (ELEMENT new_R_gcr (20))))) ∧
         ~(~(ELEMENT new_R_gcr (22)) ∨ ((ELEMENT new_R_gcr (21)) ∧ R_c23_cout_del))) => R_int2_en | ARB))) in
let new_R_ctr0_in = ((r_writeB ∧ (r_reg_sel = (WORDN 8))) => i_ad | R_ctr0_in) in
let new_R_ctr0_mux_sel = (r_cir_wr01B ∨ ((ELEMENT new_R_gcr (16)) ∧ R_ctr1_cry)) in
let new_R_ctr0_irden = (r_readB ∧ (r_reg_sel = (WORDN 8))) in
let new_R_ctr0 = ((R_ctr0_mux_sel) => R_ctr0_in | R_ctr0_new) in
let new_R_ctr0_new = (((ELEMENT new_R_gcr (19))) => (INCN 31 R_ctr0) | R_ctr0) in
let new_R_ctr0_cry = ((ONES 31 R_ctr0) ∧ (ELEMENT new_R_gcr (19))) in
let new_R_ctr0_out = ((r_fsm_cntlatch) => R_ctr0_new | R_ctr0_out) in
let new_R_ctr0_orden = (r_readB ∧ (r_reg_sel = (WORDN 12))) in
let new_R_ctr1_in = ((r_writeB ∧ (r_reg_sel = (WORDN 9))) => i_ad | R_ctr1_in) in
let new_R_ctr1_mux_sel = (r_cir_wr01B ∨ ((ELEMENT new_R_gcr (16)) ∧ R_ctr1_cry)) in
let new_R_ctr1_irden = (r_readB ∧ (r_reg_sel = (WORDN 9))) in
let new_R_ctr1 = ((R_ctr1_mux_sel) => R_ctr1_in | R_ctr1_new) in
let new_R_ctr1_new = ((R_ctr0_cry) => (INCN 31 R_ctr1) | R_ctr1) in
let new_R_ctr1_cry = ((ONES 31 R_ctr1) ∧ R_ctr0_cry) in
let new_R_ctr1_out = ((R_cntlatch_del) => R_ctr1_new | R_ctr1_out) in
let new_R_ctr1_orden = (r_readB ∧ (r_reg_sel = (WORDN 13))) in
let new_R_ctr2_in = ((r_writeB ∧ (r_reg_sel = (WORDN 10))) => i_ad | R_ctr2_in) in
let new_R_ctr2_mux_sel = ((r_cir_wr23B ∨ ((ELEMENT new_R_gcr (20)) ∧ R_ctr3_cry))) in
let new_R_ctr2_irden = (r_readB ∧ (r_reg_sel = (WORDN 10))) in
let new_R_ctr2 = ((R_ctr2_mux_sel) => R_ctr2_in | R_ctr2_new) in
let new_R_ctr2_new = (((ELEMENT new_R_gcr (23))) => (INCN 31 R_ctr2) | R_ctr2) in
let new_R_ctr2_cry = ((ONES 31 R_ctr2) ∧ (ELEMENT new_R_gcr (23))) in
let new_R_ctr2_out = ((r_fsm_cntlatch) => R_ctr2_new | R_ctr2_out) in
let new_R_ctr2_orden = (r_readB ∧ (r_reg_sel = (WORDN 14))) in
let new_R_ctr3_in = ((r_writeB ∧ (r_reg_sel = (WORDN 11))) => i_ad | R_ctr3_in) in
let new_R_ctr3_mux_sel = ((r_cir_wr23B ∨ ((ELEMENT new_R_gcr (20)) ∧ R_ctr3_cry))) in
let new_R_ctr3_irden = (r_readB ∧ (r_reg_sel = (WORDN 11))) in
let new_R_ctr3 = ((R_ctr3_mux_sel) => R_ctr3_in | R_ctr3_new) in
let new_R_ctr3_new = ((R_ctr2_cry) => (INCN 31 R_ctr3) | R_ctr3) in
let new_R_ctr3_cry = ((ONES 31 R_ctr3) ∧ R_ctr3_cry) in
let new_R_ctr3_out = ((R_cntlatch_del) => R_ctr3_new | R_ctr3_out) in
let new_R_ctr3_orden = (r_readB ∧ (r_reg_sel = (WORDN 15))) in
let new_R_icr_load = (r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let new_R_icr_old =
        ((r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => R_icr | R_icr_old) in
let new_R_icr_mask =
        ((r_writeB ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) => i_ad | R_icr_mask) in
let new_R_icr_rden = ((new_R_fsm_state = RA) ∧ ((r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)))) in
let r_int0_en = (((ELEMENT R_icr (0)) ∧ (ELEMENT R_icr (8))) ∨
                ((ELEMENT R_icr (1)) ∧ (ELEMENT R_icr (9))) ∨
```

```
                  ((ELEMENT R_icr (2)) ∧ (ELEMENT R_icr (10))) V
                  ((ELEMENT R_icr (3)) ∧ (ELEMENT R_icr (11))) V
                  ((ELEMENT R_icr (4)) ∧ (ELEMENT R_icr (12))) V
                  ((ELEMENT R_icr (5)) ∧ (ELEMENT R_icr (13))) V
                  ((ELEMENT R_icr (6)) ∧ (ELEMENT R_icr (14))) V
                  ((ELEMENT R_icr (7)) ∧ (ELEMENT R_icr (15)))) in
let new_R_int0_dis = r_int0_en in
let r_int3_en = (((ELEMENT R_icr (16)) ∧ (ELEMENT R_icr (24))) V
                  ((ELEMENT R_icr (17)) ∧ (ELEMENT R_icr (25))) V
                  ((ELEMENT R_icr (18)) ∧ (ELEMENT R_icr (26))) V
                  ((ELEMENT R_icr (19)) ∧ (ELEMENT R_icr (27))) V
                  ((ELEMENT R_icr (20)) ∧ (ELEMENT R_icr (28))) V
                  ((ELEMENT R_icr (21)) ∧ (ELEMENT R_icr (29))) V
                  ((ELEMENT R_icr (22)) ∧ (ELEMENT R_icr (30))) V
                  ((ELEMENT R_icr (23)) ∧ (ELEMENT R_icr (31)))) in
let new_R_int3_dis = r_int3_en in

let new_S_soft_shot_del = (~gcrh ∧ gcrl) in
let s_soft_cnt_out =
        ((s_fsm_srs) =>
            ((gcrl ∧ ~gcrh ∧ ~S_soft_shot_del) => (WORDN 1) | (WORDN 0)) |
            ((gcrl ∧ ~gcrh ∧ ~S_soft_shot_del) => (INCN 2 S_soft_cnt) | S_soft_cnt)) in
let new_S_soft_cnt = ((~gcrh ∧ ~gcrl) => (WORDN 0) | s_soft_cnt_out) in
let s_delay_out =
        ((s_fsm_src V (s_fsm_scs ∧ (ELEMENT S_delay (6)))) =>
        ((s_fsm_sec) => (WORDN 1) | (WORDN 0)) |
        ((s_fsm_sec) => (INCN 17 S_delay) | S_delay)) in
let new_S_delay = s_delay_out in
let s_cpu0_ok = (s_fsm_sc0f ∧ Failure0_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu1_ok = (s_fsm_sc1f ∧ Failure1_ ∧ (s_soft_cnt_out = (WORDN 5))) in
let new_S_pmm_fail =
        ((s_fsm_sb ∧ ~s_fsm_spmf) => T |
        ((~s_fsm_sb ∧ s_fsm_spmf) => F |
        ((~s_fsm_sb ∧ ~s_fsm_spmf) => S_pmm_fail | ARB))) in
let new_S_cpu0_fail =
        ((s_fsm_sb ∧ ~(s_cpu0_ok V Bypass)) => T |
        ((~s_fsm_sb ∧ (s_cpu0_ok V Bypass)) => F |
        ((~s_fsm_sb ∧ ~(s_cpu0_ok V Bypass)) => S_cpu0_fail | ARB))) in
let new_S_cpu1_fail =
        ((s_fsm_sb ∧ ~(s_cpu1_ok V Bypass)) => T |
         ((~s_fsm_sb ∧ (s_cpu1_ok V Bypass)) => F |
         ((~s_fsm_sb ∧ ~(s_cpu1_ok V Bypass)) => S_cpu1_fail | ARB))) in
let new_S_piu_fail =
        ((s_fsm_sb ∧ ~(s_fsm_spf V Bypass)) => T |
         ((~s_fsm_sb ∧ (s_fsm_spf V Bypass)) => F |
         ((~s_fsm_sb ∧ ~(s_fsm_spf V Bypass)) => S_piu_fail | ARB))) in
let s_cpu0_select = ((s_fsm_sn V s_fsm_so) ∧ ~S_cpu0_fail) in
let s_cpu1_select = ((s_fsm_sn V s_fsm_so) ∧ S_cpu0_fail ∧ ~S_cpu1_fail) in
let new_S_bad_cpu0 =
        ((s_fsm_sb ∧ ~s_cpu0_select) => T |
         ((~s_fsm_sb ∧ s_cpu0_select) => F |
         ((~s_fsm_sb ∧ ~s_cpu0_select) => S_bad_cpu0 | ARB))) in
let new_S_bad_cpu1 =
        ((s_fsm_sb ∧ ~s_cpu1_select) => T |
            ((~s_fsm_sb ∧ s_cpu1_select) => F |
            ((~s_fsm_sb ∧ ~s_cpu1_select) => S_bad_cpu1 | ARB))) in
    let new_S_reset_cpu0 = (new_S_bad_cpu0 ∧ s_fsm_src0) in
    let new_S_reset_cpu1 = (new_S_bad_cpu1 ∧ s_fsm_src1) in
    let new_S_cpu_hist = (S_reset_cpu0 ∧ S_reset_cpu1 ∧ Bypass) in
    let ss0 = (ALTER ARBN (0) ((new_S_fsm_state = SS) ∨ (new_S_fsm_state = SSTOP)
                                  ∨ (new_S_fsm_state = SCS) ∨ (new_S_fsm_state = SN)
                                  ∨ (new_S_fsm_state = SO))) in
    let ss1 = (ALTER ss0 (1) ((new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
                               ∨ (new_S_fsm_state = SC1I) ∨ (new_S_fsm_state = SC1F)
                               ∨ (new_S_fsm_state = SS) ∨ (new_S_fsm_state = SSTOP)
                               ∨ (new_S_fsm_state = SCS))) in
    let ss2 = (ALTER ss1 (2) ((new_S_fsm_state = SPF) ∨ (new_S_fsm_state = SC0I)
                               ∨ (new_S_fsm_state = SC0F) ∨ (new_S_fsm_state = ST)
                               ∨ (new_S_fsm_state = SSTOP) ∨ (new_S_fsm_state = SO))) in
    let ss3 = (ALTER ss2 (3) ((new_S_fsm_state = SRA) ∨ (new_S_fsm_state = SPF)
                               ∨ (new_S_fsm_state = ST) ∨ (new_S_fsm_state = SC1I)
                               ∨ (new_S_fsm_state = SCS) ∨ (new_S_fsm_state = SN)
                               ∨ (new_S_fsm_state = SO))) in
let s_state = ss3 in
    let sr28 = (ALTER ARBN (28) new_M_parity) in
    let sr28_25 = (MALTER sr28 (27,25) new_C_ss) in
    let sr28_24 = (ALTER sr28_25 (24) new_C_parity) in
    let sr28_22 = (MALTER sr28_24 (23,22) ChannelID) in
    let sr28_16 = (MALTER sr28_22 (21,16) Id) in
    let sr28_12 = (MALTER sr28_16 (15,12) s_state) in
    let sr28_9 = (ALTER sr28_12 (9) new_S_pmm_fail) in
    let sr28_8 = (ALTER sr28_9 (8) new_S_piu_fail) in
    let sr28_3 = (ALTER sr28_8 (3) new_S_reset_cpu1) in
    let sr28_2 = (ALTER sr28_3 (2) new_S_reset_cpu0) in
    let sr28_1 = (ALTER sr28_2 (1) new_S_cpu1_fail) in
    let sr28_0 = (ALTER sr28_1 (0) new_S_cpu0_fail) in
    let new_R_sr = ((r_fsm_cntlatch) => sr28_0 | R_sr) in
    let new_R_sr_rden = (r_readB ∧ (r_reg_sel = (WORDN 4))) in

    let new_P_fsm_rst = reset_piu in
    let new_P_fsm_sack = p_sack in
    let new_P_fsm_cgnt_ = ~(new_C_mfsm_state = CMA3) in
    let new_P_fsm_hold_ = new_C_holdA_ in
    let new_C_mfsm_D = ClkD in
    let new_C_mfsm_rst = reset_cport in
    let new_C_mfsm_crqt_ = ~(new_P_dest1 ∧ new_P_rqt) in
    let new_C_mfsm_hold_ = new_C_holdA_ in
    let new_C_mfsm_ss = CB_ss_in in
    let new_C_mfsm_invalid = piu_invalid in
    let new_C_sfsm_D = ClkD in
    let new_C_sfsm_rst = reset_cport in
    let new_C_sfsm_hlda_ = ~(new_P_fsm_state = PH) in
    let new_C_sfsm_ms = CB_ms_in in
    let new_C_efsm_cale_ = i_cale_ in
    let new_C_efsm_last_ = i_last_ in
    let new_C_efsm_male_ = i_male_ in
    let new_C_efsm_rale_ = i_rale_ in
    let new_C_efsm_srdy_ = i_srdy_ in
let new_C_efsm_rst = reset_cport in
let new_M_fsm_male_ = i_male_ in
let new_M_fsm_last_ = i_last_ in
let new_M_fsm_mrdy_ = ((~(P_fsm_state = PH)) => F | C_mrdy_del_) in
let new_M_fsm_rst = reset_piu in
let new_R_fsm_ale_ = i_rale_ in
let new_R_fsm_mrdy_ = ((~(P_fsm_state = PH)) => F | C_mrdy_del_) in
let new_R_fsm_last_ = i_last_ in
let new_R_fsm_rst = reset_piu in
let new_S_fsm_rst = Rst in
let new_S_fsm_delay6 = (ELEMENT s_delay_out (6)) in
let new_S_fsm_delay17 = ((Test) => (ELEMENT s_delay_out (6)) | (ELEMENT s_delay_out (17))) in
let new_S_fsm_bothbad = (new_S_cpu0_fail ∧ new_S_cpu1_fail) in
let new_S_fsm_bypass = Bypass in

let L_ad_out = (((~(new_P_fsm_state = PA))
                ∧ (~(new_P_fsm_state = PH))
                ∧ ~((new_P_fsm_state = PD) ∧ new_P_wr)) => i_ad | ARBN) in
let L_ready_ = ~(~i_srdy_ ∧ (new_P_fsm_state = PD)) in
let CB_rqt_out_ = ~(~(new_C_mfsm_state = CMI)) in
let ms0 = (ALTER ARBN (0) (((new_C_mfsm_state = CMD0) ∧ ~C_last_in_) ∨
                           ((new_C_mfsm_state = CMW) ∧ C_lock_in_) ∨
                           (new_C_mfsm_state = CMABT))) in
let ms10 = (ALTER ms0 (1) (((new_C_mfsm_state = CMA1) ∨ (new_C_mfsm_state = CMA0) ∨
                           (new_C_mfsm_state = CMA2) ∨ (new_C_mfsm_state = CMD1) ∨
                           ((new_C_mfsm_state = CMD0) ∧ C_last_in_) ∨ (new_C_mfsm_state = CMW) ∨
                           (new_C_mfsm_state = CMABT)))) in
let ms210 = (ALTER ms10 (2) (((new_C_mfsm_state = CMA3) ∨ (new_C_mfsm_state = CMA1) ∨
                             (new_C_mfsm_state = CMA0) ∨ (new_C_mfsm_state = CMA2) ∨
                             (new_C_mfsm_state = CMD1) ∨ (new_C_mfsm_state = CMD0) ∨
                             (new_C_mfsm_state = CMW) ∨ (new_C_mfsm_state = CMABT)) ∧
                                                       ~new_S_pmm_fail ∧ ~(ELEMENT new_R_gcr (28)))) in
let CB_ms_out = (((~(new_C_mfsm_state = CMI)) ∧ (~(new_C_mfsm_state = CMR))) => ms210 | ARBN) in
let ss0 = (ALTER ARBN (0) ((new_C_sfsm_state = CSA0W) ∨
                           ((new_C_sfsm_state = CSALE) ∧ ~c_new_write) ∨
                           (new_C_sfsm_state = CSACK))) in
let ss10 = (ALTER ss0 (1) ~(new_C_sfsm_state = CSACK)) in
let ss210 = (ALTER ss10 (2) (~new_S_pmm_fail ∧ ~(ELEMENT new_R_gcr (28)))) in
let CB_ss_out = (((~(new_C_sfsm_state = CSI)) ∧ (~(new_C_sfsm_state = CSABT))) => ss210 | ARBN) in
let CB_ad_out = ((c_dfsm_cad_en) =>
                ((c_cout_sel = (WORDN 0)) => Par_Enc rep (SUBARRAY new_C_a1a0 (15,0)) |
                ((c_cout_sel = (WORDN 1)) => Par_Enc rep (SUBARRAY new_C_a1a0 (31,16)) |
                ((c_cout_sel = (WORDN 2)) => Par_Enc rep (SUBARRAY new_C_a3a2 (15,0)) |
                Par_Enc rep (SUBARRAY new_C_a3a2 (31,16))))) | ARBN) in
let MB_addr = ((M_rdy) => (INCN 18 M_addr) | M_addr) in
let mb_data_7_0 = (((ELEMENT M_be (0))) => (SUBARRAY i_ad (7,0)) | (SUBARRAY M_rd_data (7,0))) in
let mb_data_15_8 = (((ELEMENT M_be (1))) => (SUBARRAY i_ad (15,8)) | (SUBARRAY M_rd_data (15,8))) in
let mb_data_23_16 = (((ELEMENT M_be (2))) => (SUBARRAY i_ad (23,16)) | (SUBARRAY M_rd_data (23,16))) in
let mb_data_31_24 = (((ELEMENT M_be (3))) => (SUBARRAY i_ad (31,24)) | (SUBARRAY M_rd_data (31,24))) in
let mb_data = ((MALTER (MALTER (MALTER (MALTER ARBN (7,0) mb_data_7_0)
                                                    (15,8) mb_data_15_8)
                                                    (23,16) mb_data_23_16)
                                                    (31,24) mb_data_31_24)) in
let MB_data_out = ((new_M_fsm_state = MW) => (Ham_Enc rep mb_data) | ARBN) in
```

```
    let MB_cs_eeprom_ = ~((~(new_M_fsm_state = MI)) ∧ ~new_M_se) in
    let MB_cs_sram_ = ~((~(new_M_fsm_state = MI)) ∧ new_M_se) in
    let MB_we_ = ~((new_M_se ∨ ~(~(new_M_fsm_state = MI)) ∨ ~reset_cport)
                  ∧ ~disable_writes
                  ∧ ((new_M_fsm_state = MBW) ∨ (new_M_fsm_state = MW) ∨ new_M_wwdel)) in
    let MB_oe_ = ~((~new_M_wr ∧ (new_M_fsm_state = MA)) ∨ (new_M_fsm_state = MR)) in
    let disable_int = (~(s_fsm_sn ∧ (ELEMENT s_delay_out (6))) ∧ s_fsm_sdi
                  ∧ ((Test) => ~(ELEMENT s_delay_out (5)) | ~(ELEMENT s_delay_out (16)))) in
    let Int0_ = ~(r_int0_en ∧ ~R_int0_dis ∧ ~disable_int) in
    let Int1 = (R_ctr1_cry ∧ new_R_int1_en ∧ ~disable_int) in
    let Int2 = (R_ctr3_cry ∧ new_R_int2_en ∧ ~disable_int) in
    let Int3_ = ~(r_int3_en ∧ ~R_int3_dis ∧ ~disable_int) in
    let Led = (SUBARRAY new_R_gcr (3,0)) in
    let Reset_cpu0 = new_S_reset_cpu0 in
    let Reset_cpu1 = new_S_reset_cpu1 in
    let Cpu_hist = new_S_cpu_hist in
    let Piu_fail = new_S_piu_fail in
    let Cpu0_fail = new_S_cpu0_fail in
    let Cpu1_fail = new_S_cpu1_fail in
    let Pmm_fail = new_S_pmm_fail in

    (L_ad_out, L_ready_,
     CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out,
     MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_,
     Int0_, Int1, Int2, Int3_, Led,
     Reset_cpu0, Reset_cpu1, Cpu_hist, Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail)"
    );;

close_theory();;
```

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE November 1, 1992	3. REPORT TYPE AND DATES COVERED Contractor Report

4. TITLE AND SUBTITLE Formal Design Specification of a Processor Interface Unit	5. FUNDING NUMBERS C NAS1-18586 WU 505-64-10-07
6. AUTHOR(S) David A. Fura Phillip J. Windley Gerald C. Cohen	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Boeing Military Airplanes P.O. Box 3707 M/S 4C-70 Seattle, WA 98124-2207	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA CR-189698

11. SUPPLEMENTARY NOTES

Langley Technical Monitor: Sally C. Johnson
Task 9 Report

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified-Unlimited Subject Category 60	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes work to formally specify the requirements and design of a processor interface unit (PIU), a single-chip subsystem providing memory-interface bus-interface, and additional support services for a commercial microprocessor within a fault-tolerant computer system. This system, the Fault-Tolerant Embedded Processor (FTEP), is targeted towards applications in avionics and space requiring extremely high levels of mission reliability, extended maintenance-free operation, or both. The need for high-quality design assurance in such applications is an undisputed fact, given the disastrous consequences that even a single design flaw can produce. Thus, the further development and application of formal methods to fault-tolerant systems is of critical importance as these systems see increasing use in modern society.

14. SUBJECT TERMS Fault Tolerant HOL Bus Interface	Specification Generic Interpreter Theory Memory Interface Fault Tolerant Embedded Processor (FTEP)	15. NUMBER OF PAGES 256 16. PRICE CODE A12

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102