Circuits and Systems, 2014, 5, 170-180
Published Online July 2014 in SciRes. http://www.scirp.org/journal/cs
http://dx.doi.org/10.4236/cs.2014.57019

Scientific Research

An IEEE 1149.x Embedded Test Coprocessor

Ukbagiorgis Iyasu Gebremeskel, José Manuel Martins Ferreira

Faculty of Technology and Maritime Sciences, Buskerud and Vestfold University College, Kongsberg, Norway
Email: ukbieso@gmail.com, josemmf@hbv.no

Received 27 May 2014; revised 30 June 2014; accepted 11 July 2014

Abstract

This paper describes a microprogrammed architecture for an embedded coprocessor that is able to control IEEE 1149.1 to IEEE 1149.7 test infrastructures, and explains how to expand the supported test command set. The coprocessor uses a fast simplex link (FSL) channel to interface a 32-bit MicroBlaze CPU, but it can work with any microprocessor core that accepts this simple FIFO-based interface method. The implementation cost (logic resource usage for a Xilinx Spartan-6 FPGA) and the performance data (operating frequency) are presented for a test command set comprising two parts: 1) the full IEEE 1149.1 structural test operations; 2) a subset of IEEE 1149.7 operations selected to illustrate the implementation of advanced scan formats.

Keywords

Built-In Test, Boundary-Scan, Embedded Coprocessors, MicroBlaze, IEEE 1149.1, IEEE 1149.7

1. Introduction

After nearly 25 years of industry acceptance, starting immediately after its approval as a standard in 1990 [1], the IEEE 1149.1 Boundary-Scan (BS) Architecture and Test Access Port is now being used in a range of application areas that largely exceeds its original scope, which is restricted to structural testing of digital printed circuit boards. Newer application domains where BS is being used, such as in-circuit debugging, require more efficient and powerful data transfer mechanisms, and lead many designers to implement workarounds that compromised full 1149.1-compliance. In recognition of the need for a more powerful test standard, compatible with all the investment made in 1149.1, the IEEE 1149.7 Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture was approved in 2009 [2], and offered a far more powerful solution to test and applications debug. The development of solutions that demonstrate its ability to coexist with 1149.1 applications, while at the same time supporting the more powerful scan transfer formats required for debugging multi-core architectures, represents a fundamental step to unlock the power of the new standard and to promote its wider acceptance [3]-[5].

How to cite this paper: Gebremeskel, U.I. and Ferreira, J.M.M. (2014) An IEEE 1149.x Embedded Test Coprocessor. *Circuits and Systems*, **5**, 170-180. http://dx.doi.org/10.4236/cs.2014.57019

This paper presents a microprogrammed architecture for an embedded coprocessor dedicated to IEEE 1149.1/1149.7 test operations. Its current instruction set supports all IEEE 1149.1 test operations (state transition, shift operations, shift and compare), and an additional subset of IEEE1149.7 test operations designed as proof of concept for the proposed architecture (optimized scan format Oscan1, zero-bit DR scan and escape sequences). The following section briefly describes the evolution from IEEE 1149.1 to 1149.7. Section 3 describes the micro-programmed architecture of the proposed embedded test coprocessor, and Section 4 explains the test command design process. Section 5 presents implementation cost and performance data. The main conclusions of this work are presented in the last section.

2. IEEE 1149.x Standards and Test Command Set

Several IEEE 1149.x standards and proposed standards were developed by the test community since the mid-1980s, when the Joint Test Action Group (JTAG) initiated the development of the test technology that became known as the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. The approval of IEEE 1149.1 in 1990 marked the beginning of a series of IEEE 1149.x test standards that include successful and unsuccessful attempts to provide industry-accepted production test technologies [6]. The IEEE 1149.5 Standard for Module Test and Maintenance Bus (MTM-Bus) Protocol [7], and the IEEE 1149.4 Standard for a Mixed-Signal Test Bus [8], are two examples of proposals that met little or no industry acceptance, in spite of having become recognized IEEE standards. The IEEE 1149.6 Standard for Boundary-Scan Testing of Advanced Digital Networks [9] deserved better attention, mostly because it overcomes a specific limitation of boundary-scan—the testing of AC-coupled differential interconnections. Far more successful that all its 1149.x successors, the IEEE 1149.1 boundary-scan test technology is now being used to address application areas that go far beyond its original development scope. The use of the standard 4-pin test access port (TAP) to control in-circuit debugging is particularly worth mentioning, both because it gained wide acceptance since the early years of boundary-scan, and also because it has led to *the creation of TAPs whose operation includes behavior that deviates from the behavior specified by IEEE Std* 1149.1-2001 [2].

The IEEE 1149.7 standard, approved in 2009, offers a framework that ensures interoperability between 1149.1 and 1149.7 components, while enabling enhanced test and debug features, and a reduced 2-pin TAP (where mode select, data in and data out information are multiplexed in a single TMSC pin). Besides supporting several types of advanced scan formats (MScan, OScan, SScan), which enable a variety of tradeoffs between capability and performance, the IEEE 1149.7 test architecture also supports a power-down mode to reduce consumption when the test and debug logic is not in use. The original BS architecture, illustrated in **Figure 1**, remains at the core of the new IEEE 1149.7 standard, and helps us to understand why most IEEE 1149.x test operations can be implemented with a reduced set of basic test commands, that are able to control state transition and to carry out shift, and shift and compare operations.

Table 1 presents test command set that is supported by our embedded test coprocessor. Besides covering all the basic 1149.1 requirements, it includes four additional commands to illustrate 1149.7 test operations. This set offers close equivalents to the main Serial Vector Format (SVF) [10] commands, which are accepted by most test equipment manufacturers. The following section discusses the alternatives available to implement its control path, and offers a detailed description of the micro-programmed architecture that was chosen for this purpose.

3. A Micro-Programmed Architecture for an IEEE 1449.7 Coprocessor

The first step for implementing a controller architecture supporting the set of commands referred in the previous section consists of building a formal representation of their functionality, bringing into evidence the corresponding control and data flow operations. The data flow operations will determine the blocks required in the coprocessor data path, which will be implemented with regular sequential circuits. There is a higher degree of freedom in what concerns the implementation of the control path, where a hardwired or a microprogrammed architecture can be used to implement the corresponding control flow operations. Notice also that the formal representation of each test command will be influenced by the decision of implementing it in Moore or Mealy form. A Moore machine will only update its outputs (the control signals to the data path) upon the rising edge of the system clock, while Mealy machines can update their outputs at any moment. To illustrate the influence of this choice, **Figure 2(a)** shows an excerpt of an ASMD chart that illustrates a Moore specification for the SHF1 test command.

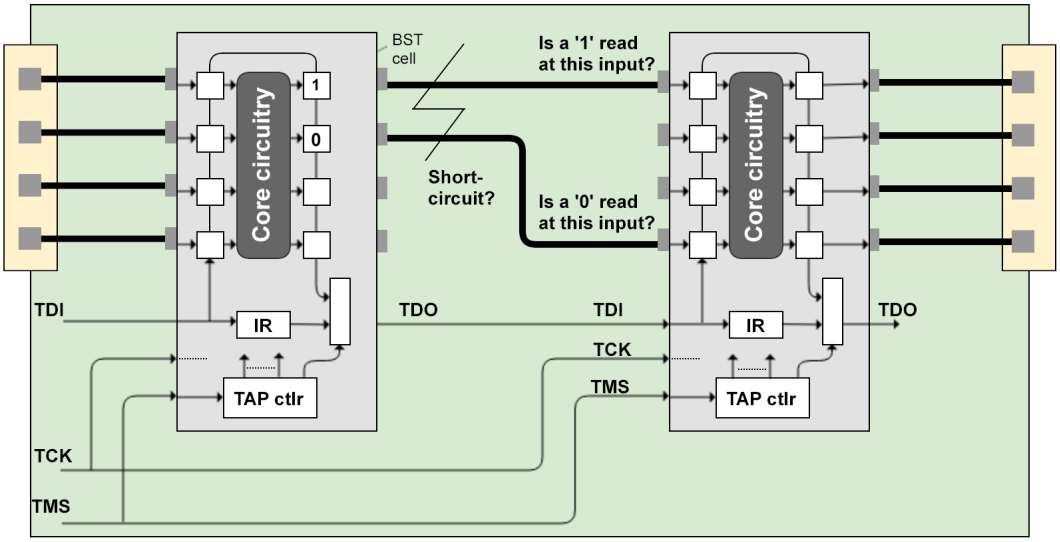


Figure 1. Original IEEE 1149.1 boundary-scan test infrastructure for structural fault detection.

Table 1. IEEE 1149.1/1149.7 test command set supported by the proposed embedded test coprocessor.

Test command	Description
RESET	Takes the boundary-scan logic to Test-Logic-Reset (equivalent to SVF "STATE RESET")
TMS0, TMS1	Set TMS to 0/1 and generate one TCK clock pulse (enabling the implementation of any SVF "STATE" command)
MTCK N	Sets TMS to 0 and generates N TCK pulses (enables the multiple TCK tests carried out by SVF "RUNTEST")
SHF1 N X	Shifts an N-bit bitstream (X) into the [instruction | selected data] register(s) (data bits shifted out are ignored)
SHFCP1 N X, Y, Z	Shifts an N-bit bitstream (X) into the [instruction | selected data] register(s), and compares the output bitstream with its expected response (Y), in all positions indicated by the mask bitstream (Z) (enables the scan & compare "SDR" and "SIR" SVF commands)
ZBS N	Issues N zero bit scan sequences by counting N TMS edge counts (used to generate IEEE 1149.7 commands and to set the control level)
ESC N	Issues N escape sequences (used to select, deselect, reset, and set up other IEEE 1149.7 specific operations)
SHF7O1 N X	Equivalent to SHF1 above, but using the 1149.7 advanced scan format OScan1
SHFCP7O1 N X, Y, Z	Equivalent to SHFCP1 above, but using the 1149.7 advanced scan format OScan1

States 2 and 3 in **Figure 2(a)** include conditional output boxes used to define Mealy outputs that depend on the condition indicated in the preceding decision box. States 2, 4 and 5 comprise non-empty state boxes used to define Moore outputs, which will be asserted while the controller remains in the corresponding state. The implementation of the same functional behavior in the form of a Moore machine will increase the number of states, since each conditional output box will have to be converted into an equivalent state box, generating a corresponding extra state. The formal representation for the same ASMD chart excerpt, now in the form of a Moore machine, is illustrated in **Figure 2(b)**.

The conversion into a Moore machine increased the number of states by a factor of 2.25 (from 4 to 9). However, this higher number of states does not imply a proportional degradation of cost or performance, be it in the number of logic gates or microprogram memory positions (spatial resources), or in the number of clock cycles (speed). A simple benchmarking of the execution speed can take place by considering the time required to move from state 2 to state 4, assuming that: 1) conditions B and C hold true, and 2) condition D holds false for five times.

The corresponding state transition for the Mealy representation of **Figure 2(a)** will be 2-3-5-3-5-3-5-3-5-3-5-3-4, requiring a total of 12 clock cycles. Considering the same initial and final states, and the same assumptions, the equivalent transition for the Moore representation will in this case go through states 2-3-8-5-9-5-9-5-9-5-9-

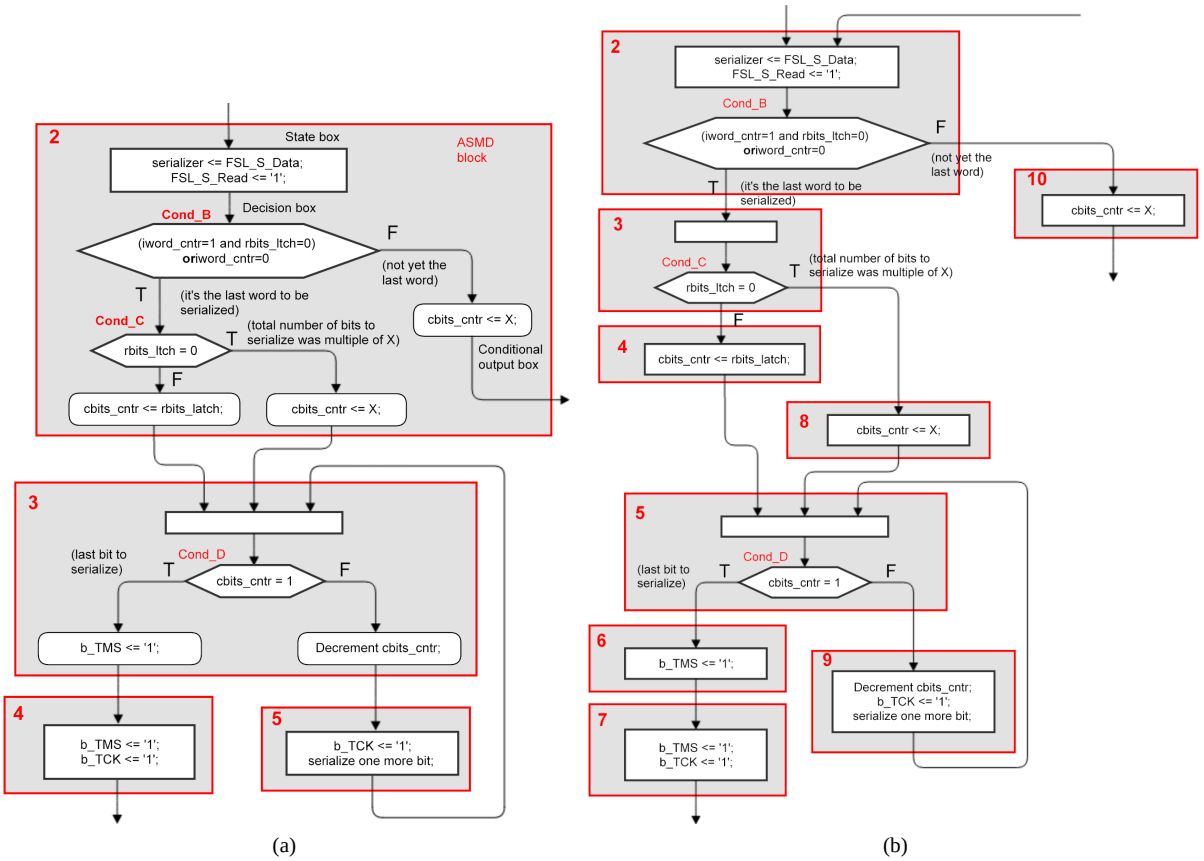


Figure 2. Partial representation of the ASMD chart for SHF1. (a) Mealy representation. (b) Moore representation.

5-9-5-6-7, and requires a total of 15 clock cycles (that is to say, 1.25 times more than the corresponding Mealy representation).

In order to compare the corresponding implementations in terms of logic resources/FPGA floorspace, a choice will have to be made between a hardwired or a microprogrammed control path architecture. The following reasons explain our choice of a microprogrammed coprocessor architecture:

- A hardwired controller needs to be completely redesigned when a new test command is required, and the designer will have to code each new ASMD chart in the chosen hardware description language (e.g. VHDL, Verilog).
- Even if major differences are not to be expected, a hardwired architecture will correspond to a new sum-of-products (or a similar canonical form) structure for each set of test commands, meaning that there will be variations in the critical path and maximum propagation delay (on the contrary, there will be no variation in the control elements of a microprogrammed architecture, since any additional commands correspond solely to further positions added to the microprogram memory).
- Assuming that the "micro-operation set" is able to cover the primitive constructs required to implement new test commands, the designer does not have to write VHDL/Verilog code whenever a new command is added, but simply to translate the respective ASMD chart into the corresponding microprogram memory bank positions.

The basic control path architecture for a microprogrammed implementation is shown in **Figure 3** (adapted from [11]). In this case, the most significant address bits of the microprogram memory are determined by the test command opcode (which is loaded into the Bank_reg latch), and the ASMD state encoding defines the least significant address bits.

The architecture illustrated in **Figure 3** is best in terms of simplicity, and enables a very straightforward implementation of any test command—each state in the ASMD chart will correspond to one word in the microprogram memory, and the operation flow can simply be specified as a sequence of CONTINUE, JUMP or

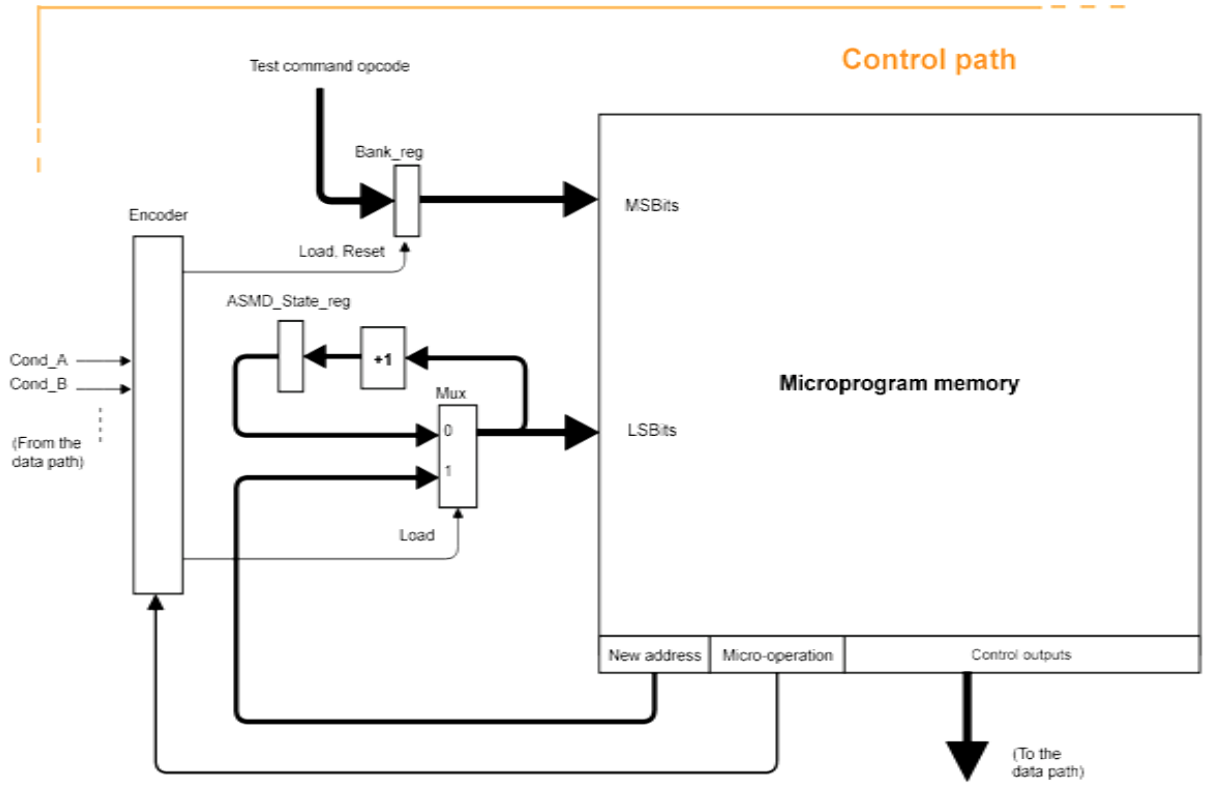


Figure 3. Basic microprogrammed architecture for a Moore control path.

BRANCH IF microinstructions, directing the state transition from beginning to end. However, it can only implement Moore machines, since each ASMD state selects a single microprogram memory position. Since the ASMD blocks frequently contain conditional output boxes, or more than one decision box, this means that the simplicity of the architecture shown in **Figure 3** comes at the price of preprocessing the ASMD chart, in order to ensure a pure Moore behavior. The total number of states increases, and so does the number of microprogram memory positions, as well as the number of system clock cycles needed to complete the execution of the corresponding test command (one clock cycle per ASMD chart state).

The general rule for preprocessing the ASMD charts consists of eliminating all conditional output boxes (which will become unconditional outputs specified in a state of their own), and splitting the state when more than one decision box is present. In addition, and since the most significant bits come directly from the test command opcode, the number of microprogram memory positions used to implement each command is fixed. This represents a waste of FPGA floorspace, since the most complex command, with the longest ASMD chart representation, will dictate the number of positions that will be used for all other commands. The control path architecture illustrated in **Figure 3** is able to implement any Moore ASMD chart, and would take 9 memory positions to implement the excerpt represented in **Figure 2(b)**. If this was the highest number of states in any command, we would need 4 least significant bits, meaning that the total storage requirements would be given by $2^4 * O = 16 * O$, where O is the number of opcodes comprised in the test command set.

If we want to enable a Mealy implementation, the least significant address bits will have to be driven from data path conditions. This solution enables any number of decision boxes per state, provided that the corresponding conditions are used to drive the least significant address bits of the microprogram memory. The main drawback is that the number of microprogram memory positions will be equal to $S * 2^D$, where S is the number of states, and D is the maximum number of decision boxes existing in a single state (16 memory positions for the example presented earlier). For a test command set with O opcodes, we would have a total microprogram memory storage requirement given by $S * 2^D * O$. While this modification means that we will now have two (or a power of two) microprogram memory positions for each ASMD chart state, the end result is not necessarily an explosion of the microprogram memory space, since state decomposition is restricted to the need of preventing

multiple decision boxes per ASMD state (a situation that is rather seldom).

The non-limited simple Mealy architecture enables the fastest implementation. On the other hand, it is the most expensive in terms of microprogram memory storage. The intermediate simple Mealy solution limited to one decision box per state is likewise an intermediate solution in terms of speed vs. microprogram memory storage, while the Moore representation is cheapest in terms of microprogram memory storage. It is also the slowest, although the number of clock cycles is not proportional to the number of states (instead it is dictated by the path through the ASMD chart). **Table 2** summarizes the pros and cons of three alternatives: Moore (no conditional outputs allowed, maximum state decomposition), Mealy 1 (one-level Mealy, enabling one decision box and its corresponding conditional output boxes per state), and Mealy 2 (two-level Mealy, enabling up to two decision boxes and four conditional output boxes). Since preprocessing takes place only once, and the difference in speed is small (cf. the simple benchmarking of execution speed that was presented in **Figure 2(a)** and **Figure 2(b)**), we adopted the Moore architecture that is represented in **Figure 4**. Its main advantage over the basic Mealy and basic Moore (**Figure 3**) architectures consists of eliminating the waste of memory positions that takes place in those two solutions, since the microcode storage requirements are in this case limited to the number of states in the ASMD chart (instead of the being fixed, and dictated by the ASMD chart with the highest number of states).

The specific nature of scan test infrastructures dictates that the number of required primitive constructs (micro-operations) is very small. The data path architecture will therefore have a small number of elements, consisting only of counters, latches and serializers. The conditions associated with the operation of these data path elements are easy to typify, and are limited to detecting if the latches and counters reached one or zero. The proposed test coprocessor architecture may be represented as shown in **Figure 5**.

Table 2. Pros and cons of Moore, Mealy-1, and Mealy-2 microprogrammed control path implementations.

Topology	Pros	Cons	Comments
Moore	Maximum simplicity	Lowest speed	Preprocessing required
Mealy 1	Good speed upgrade	Higher number of memory positions	Minimum preprocessing
Mealy 2	Marginal speed upgrade over Mealy 1	Highest number of memory positions, most of which will not be used	No preprocessing required (if no. of decision boxes per state is ≤2)

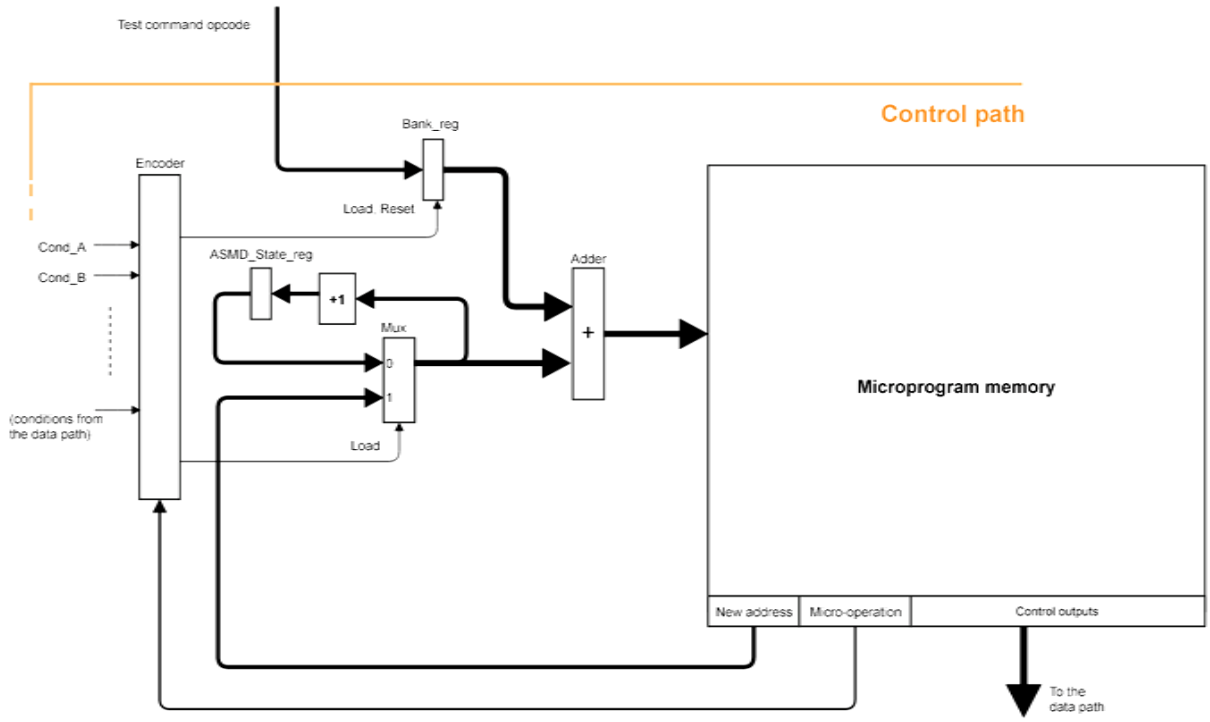


Figure 4. Microprogrammed architecture adopted for the proposed embedded test coprocessor.

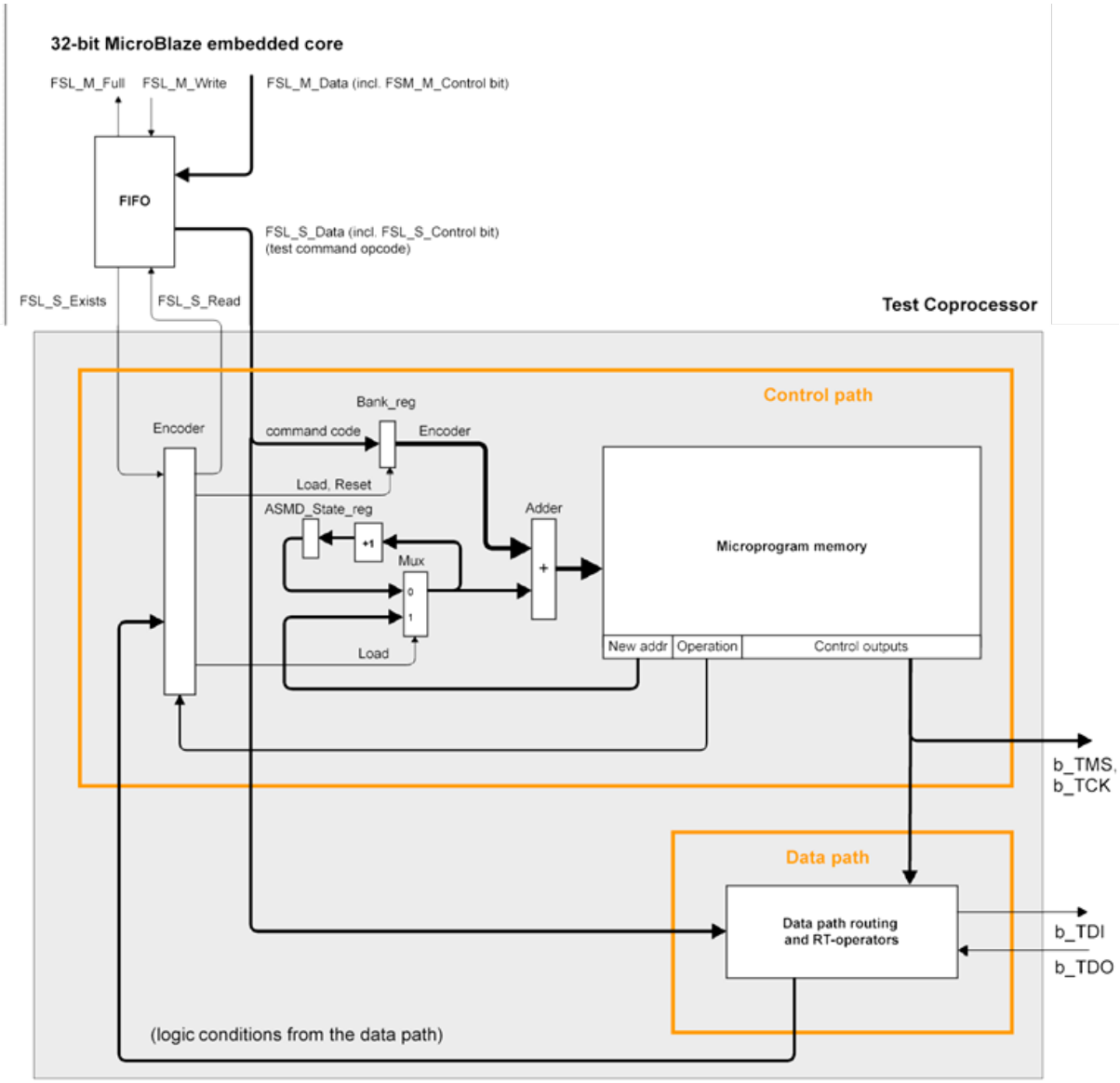


Figure 5. The proposed test coprocessor architecture and interface method.

4. Test Command Design

This section presents the micro-operation set that is supported by the microprogrammed control path, and explains the sequence of steps that will enable any designer to add further commands, in order to expand the application domain of the proposed test coprocessor (e.g. further IEEE 1149.7 operations). The microprogrammed control path architecture illustrated in **Figure 4** is able to implement the control flow associated to any ASMD chart that is specified in the form of a Moore machine. Each state in the ASMD chart corresponds to one position in the microprogram memory, which comprises the three fields shown in **Figure 4** and **Figure 5**:

- The leftmost bits represent the new address, to be used whenever the next state encoding is different from the current state encoding plus 1 (in which case we'll jump into a new address, instead of incrementing the current address). The number of bits in this field is dictated by the maximum number of states in the ASMD chart of a single test command.
- The middle field contains the micro-operation that represents the required control flow. The implementation of any ASMD chart can be carried out using three basic types of micro-operations: 1) CONTINUE (*i.e.* increment the current microprogram memory address); 2) JUMP TO ADDRESS (*i.e.* load the "new address"

that is represented in the leftmost bits of the current microprogram memory position); and 3) BRANCH IF CONDITION TO ADDRESS (jumps to the indicated address if the condition is true, continues to the next address otherwise). This third type actually generates a variety of different micro-operations, *i.e.* BRANCH IF CONDITION_A TO ADDRESS is formally different from BRANCH IF/CONDITION_A TO ADDRESS (branch if not_condition). Each new test command to be supported will most likely require additional BRANCH IF micro-operations, to cope with the specific conditions associated with its execution. The number of bits in this field is determined by 2 (CONTINUE, JUMP TO ADDRESS) plus the total number of BRANCH IF micro-operations required.

- The rightmost field comprises all the control bits that determine the data flow operations indicated in the ASMD chart. In a horizontal microprogrammed architecture, such as the one that was adopted in this work, the number of bits in this field is equal to the total number of control bits required by the data path elements, plus all additional bits that are directly connected to test access port pins (e.g. board TMS and board TCK).

In order to expand the test command set supported by the coprocessor, the designer shall proceed as follows:

1) Draw the ASMD chart specifying the operation of the required test command.

2) If the ASMD chart contains conditional output boxes (Mealy machine), split those states so as to convert all conditional output boxes into state boxes (convert from Mealy to Moore).

3) Once the Moore ASMD chart is ready, fill in the microprogram memory template with the micro-operations and control bit patterns corresponding to each ASMD state.

4) Update the content of the microprogram memory.

As an illustrative example, **Figure 6(a)** shows an initial ASMD chart that describes the operation of a MTCK N test command. This command is needed to carry out various types of built-in self-test functions (e.g. all those that rely on pseudo-random pattern generation and parallel signature analysis modes), and generates N test clock (b_TCK) pulses, while keeping b_TMS at "0". MTCK starts by loading the number of required b_TCK pulses (present in the FSL_S_Data bus) into one of the data path counters (cbits_cntr in this case), which will be decremented for each b_TCK pulse. As represented in **Figure 6(a)**, state 1 requires two microprogram memory positions, since the decrement control signal for cbits_cntr may, or may not, be active in this state, according to the condition shown. This example also shows that splitting a state, in order to ensure that a conditional output box is converted into a corresponding state box, does not necessarily increase the number of states in the ASMD chart. **Figure 6(b)** represents the equivalent Moore ASMD chart, which contains the same number of states. Each state in the chart represented in **Figure 6(b)** corresponds to a single microprogram memory position. We are now ready to move into the third test command design step, where the microprogram memory template is filled to represent all control and data flow operations associated to the execution of MTCK. **Table 3** shows the content of the four microprogram memory positions that are needed to specify the execution of this test command set.

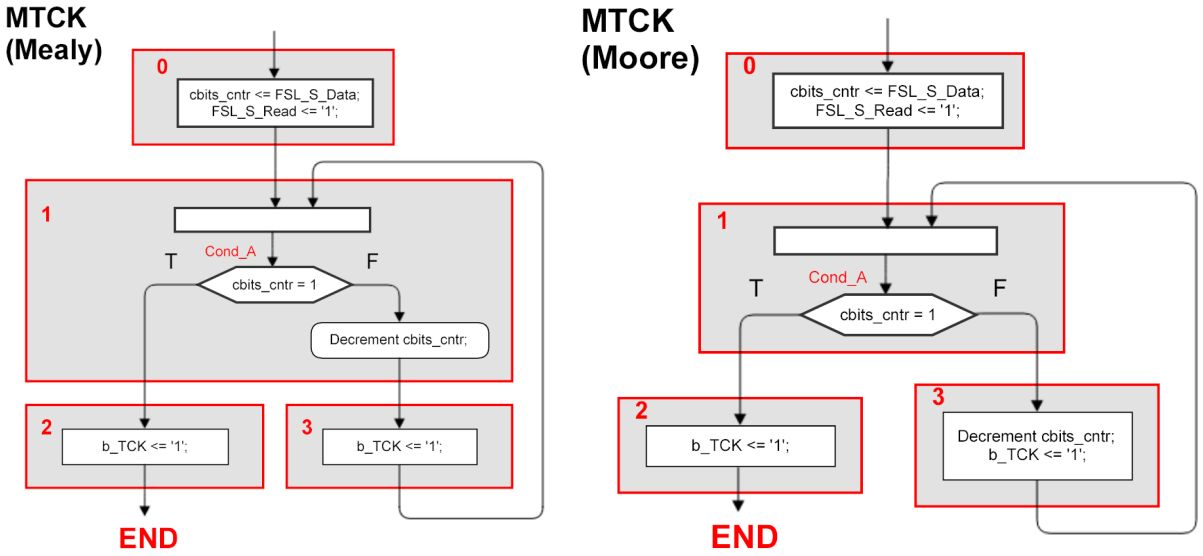
As this example shows, expanding the test command set simply consists of updating the content of the microprogram memory, releasing the designer from the need to understand and modify the VHDL code that describes the control path architecture.

5. Implementation Cost and Performance

To evaluate the overall performance of the proposed embedded test coprocessor, **Table 4** and **Table 5** show the data that were collected for each test command separately:

- The columns showing data for each test command (the four rightmost columns) correspond to the implementation of a single test command, without the FSL interface.
- Since the FSL interface is predesigned and independent of the proposed microprogrammed architecture, all the tables include two columns showing the implementation data for all test commands when no FSL interface is present, and when a single 1-word 32-bit FIFO interface is added from the MicroBlaze to the test coprocessor.

Table 4 shows the comparison of the logic resources needed by various IEEE 1149.1 test commands with a 4-pin TAP test infrastructure. TMS1 and TMS0 belong to the same group of results, and the same happens with MTCK and RESET, so each second command was omitted in all the tables to improve readability. It is important to notice that the usage of logic resources imposed by the most complex test command (SHFCP1) dictates the cost of the full implementation—the sole implementation of SHFCP1 requires practically the same resources as the full implementation of all the IEEE 1149.1 test commands that are presented in **Table 1**. The number of



(a) (b)

Figure 6. ASMD chart for MTCK. (a) Mealy representation. (b) Moore representation.

Table 3. Content of the microprogram memory for the MTCK test command.

ASMD state	µprogram ROM		µprogram ROM: control outputs									
	New_Addr	µoperation	iL	iD	rL	cL	cD	cM	sL	sS	bS	bK
0	X	CONT	0	0	0	1	0	0	0	0	0	0
1	Offset to pos. 3	BIF/A	0	0	0	0	0	0	0	0	0	0
2	Offset to END pos.	JUMP	0	0	0	0	0	0	0	0	0	1
3	Offset to pos. 1	JUMP	0	0	0	0	1	0	0	0	0	1

(iL: iwordL; iD: iwordD; rL: bitsL; cL: cbitsL; cD: cbitsD; cM: cbitsM; sL: serL; sS: serS; bS: b_TMS; bK: b_TCK).

Table 4. Logic resources usage by the IEEE 1149.1 test commands.

Parameter	All (no FSL)	All (one FSL)	Only TMS1	Only MTCK	Only SHF1	Only SHFCP1
No. of slice registers	388 (1%)	374 (2%)	88 -	120 -	274 -	338 -
No. of slice LUTs	383 (4%)	424 (4%)	125 -	207 -	307 -	380 -
No. of memory positions	52	52	4	7	16	23

Table 5. Logic resources usage by the IEEE 1149.7 test commands.

Parameter	All (no FSL)	All (one FSL)	Only ESC	Only ZBS	Only SHF7O1	Only SHFCP7O1
No. of slice registers	160 (1%)	377 (2%)	75 -	75 -	144 -	160 -
No. of slice LUTs	232 (4%)	461 (5%)	179 -	146 -	178 -	224 -
No. of memory positions	98	98	9	23	32	40

microprogram memory positions when the full set of commands is implemented, can be calculated by adding the equivalent value for each command individually, but taking into account that the first two positions are common to all the test commands.

The same comparison has been done for the IEEE 1149.7 test commands, and the summary of the collected data is shown in **Table 5**. This table shows that the usage of logic resources imposed by the most complex test command (SHFCP7O1) again dictates the cost of the full implementation—the sole implementation of SHFCP7O1 requires almost the same resources as the full implementation of all the IEEE 1149.7 test commands that are presented in **Table 1**. The number of microprogram memory positions required when implementing the full set of test commands can be calculated as indicated previously for the 1149.1 commands.

With respect to timing performance, we again notice that the most complex command is the main contributor to determine the minimum period/maximum frequency of the embedded test coprocessor operation. **Table 6** shows the timing performance of the implemented IEEE 1149.1 test commands with a 4-pin TAP. The full implementation, in the case of "All (no FSL)", is circa 30% slower than what would correspond to SHFCP1 alone. The minimum period is very close to the maximum combinational path delay, in the case of the sole implementation of the test commands TMS1, TMS0, MTCK, or RESET, but it is less so as we move into the more complex test commands (SHF1 and SHFCP1).

Table 7 shows the timing performance of the implemented IEEE 1149.7 test commands with a 2-pin TAP. One can notice that, as we move to the right, the minimum period/maximum frequency is increasing. This shows that the most complex test command (SHFTCP7O1) is the major contributor to determine the minimum period/maximum frequency of the operation of the proposed embedded test coprocessor. The data presented in **Table 7** shows that the proposed embedded test coprocessor is able to run at TCK frequencies slightly above 70% of the FPGA system clock (which in the case of the Nexys 3™ board runs at 100 MHz).

6. Conclusions

Several IEEE 1149.1 test controller solutions have been developed over the years [12]-[17], but little attention has been given to controller architectures proposed as embedded coprocessors. The work described in this paper offers two main contributions with this respect, by proposing:

1) An embedded test coprocessor that has the capability to test any IEEE 1149.x-compatible system. This coprocessor supports a wide range of testing scenarios for embedded systems based on single or multi-core 32-bit MicroBlaze CPUs, from built-in self-test to online fault detection and diagnosis.

2) A microprogrammed control path architecture for the test coprocessor, enabling a straightforward expansion of the test command set to cope with additional application domains, e.g. those made possible by IEEE 1149.7. The simplicity of the selected architecture enables a relatively fast execution of the test commands, reaching above 70% of the Nexys 3™ board system clock (100 MHz).

Table 6. Timing performance for the IEEE 1149.1 test commands.

Parameter	All (no FSL)	All (one FSL)	Only TMS1	Only MTCK	Only SHF1	Only SHFCP1
Min. period - Max. freq.	12.276 ns - 81.457 MHz	12.315 ns - 81.199 MHz	6.053 ns - 165.211 MHz	6.249 ns - 160.037 MHz	7.722 ns - 128.662 MHz	11.246 ns - 88.917 MHz
Max. comb. path delay	5.157 ns	8.102 ns	5.157 ns	5.385 ns	5.385 ns	5.519 ns

Table 7. Timing performance for the IEEE 1149.7 test commands.

Parameter	All (no FSL)	All (one FSL)	Only ESC	Only ZBS	Only SHF7O1	Only SHFCP7O1
Min. period - Max. freq.	12.657 ns - 79.005 MHz	13.655 ns - 73.235 MHz	7.756 ns - 128.929 MHz	8.302 ns - 120.449 MHz	11.469 ns - 87.192 MHz	12.249 ns - 81.639 MHz
Max. comb. path delay	-	5.94 ns	-	-	-	-

An FSL interface is used to interact with the MicroBlaze for exchanging the command opcode, the arguments of the various functions, and the test result. The FSL is a very popular coprocessor interface method that uses a simple hardware protocol. It is supported by a flexible, yet dedicated instruction set to write to the output and read from the input port. The 32-bit wide FSL bus interface used offers a dedicated point-to-point data streaming interface. Its point-to-point nature and its minimum hardware requirement are the greatest advantages of this interface method. Because the FSL channels are dedicated, no arbitration or bus mastering is required, ensuring an extremely fast interface with very low data latency.

The proposed architecture has the capability to cope with emerging requirements of test and debug standards based on scan test infrastructures. The microprogrammed control path architecture made the embedded test coprocessor scalable, and the FSL interface enabled an optimized solution for IEEE 1149.x test infrastructures, where the most time-consuming functions were executed in hardware, and the remaining functions were implemented by software at a higher abstraction level.

References

[1] (2001) IEEE Standard Test Access Port and Boundary Scan Architecture. IEEE Std 1149.1-2001, 212 p. http://dx.doi.org/10.1109/IEEESTD.2001.92950

[2] (2009) IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture. IEEE Std 1149.7-2009, 985. http://dx.doi.org/10.1109/IEEESTD.2010.5412866

[3] Ley, A.W. (2009) Doing More with Less—An IEEE 1149.7 Embedded Tutorial: Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture. *International Test Conference*, Austin, 1-6 November 2009, 1-10. http://dx.doi.org/10.1109/TEST.2009.5355572

[4] Williams, M. (2009) Low Pin-Count Debug Interfaces for Multi-Device Systems. ARM White Paper. http://goo.gl/KyyVvP

[5] Fernandes, F.R., Machado, R.J.S., Ferreira, J.M.M. and Gericota, M.G.O. (2012) IEEE Std 1149.7: What, Why, Where? *Proceedings of the 27th Conference on Design of Circuits and Integrated Systems*, Avignon, 28-30 November 2012, 118-123.

[6] Ungar, L.Y., Bleeker, H., McDermid, J.E. and Hulvershorn, H. (2001) IEEE-1149.x Standards: Achievements vs. Expectations. *IEEE Systems Readiness Technology Conference*, Valley Forge, 20-23 August 2001, 188-205. http://dx.doi.org/10.1109/AUTEST.2001.948964

[7] IEEE Standard for Module Test and Maintenance Bus (MTM-Bus) Protocol (1996) IEEE Std 1149.5-1995. http://dx.doi.org/10.1109/IEEESTD.1996.80835

[8] IEEE Standard for a Mixed-Signal Test Bus (1999) IEEE Std 1149.4-2010 (Revision of IEEE Std 1149.4-1999). http://dx.doi.org/10.1109/IEEESTD.2011.5738198

[9] IEEE Standard for Boundary-Scan Testing of Advanced Digital Networks (2003) IEEE Std 1149.6-2003. http://dx.doi.org/10.1109/IEEESTD.2003.94249

[10] Serial Vector Format Specification (1994) Revision E. http://goo.gl/bQQkGI

[11] Lynch, M.A. (1993) Microprogrammed State Machine Design. CRC Press, Boca Raton.

[12] Yau, C.W. and Jarwala, N. (1990) The Boundary-Scan Master: Target Applications and Functional Requirements. *Proceedings of the IEEE International Test Conference*, Washington DC, 10-14 September 1990, 311-315.

[13] Ferreira, J.M., Matos, J.S. and Pinto, F.S. (1992) Automatic Generation of a Single-Chip Solution for Board-Level BIST of Boundary Scan Boards. *Proceedings of the European Design Automation Conference*, Brussels, 16-19 March 1992, 154-158. http://dx.doi.org/10.1109/EDAC.1992.205913

[14] SN54ACT8990, SN74ACT8990, IEEE STD 1149.1 (JTAG) TAP Masters with 16-Bit Generic Host Interfaces (1997) Texas Instruments Datasheet.

[15] Chen, S.J., Zhou, Y., Zhu, D.X. and Guo, S.L. (2011) Design of High-Speed Boundary-Scan Master Controller Base on SOPC. *Proceedings of the 2nd International Conference on Mechanic Automation and Control Engineering*, Hohhot, 15-17 July 2011, 1195-1198. http://dx.doi.org/10.1109/MACE.2011.5987152.

[16] Kaur, M. and Singh, B. (2012) VHDL Implementation of Test Access Port Controller. *International Journal of Engineering Science and Technology* (*IJEST*), **4**.

[17] Cabral, C.J. (2012) Design and Implementation of an IEEE 1149.7-Compliant cJTAG Controller for Debug and Trace Probe. Master of Science in Engineering Report, University of Texas at Austin, Austin, 55 p.

Scientific Research Publishing (SCIRP) is one of the largest Open Access journal publishers. It is currently publishing more than 200 open access, online, peer-reviewed journals covering a wide range of academic disciplines. SCIRP serves the worldwide academic communities and contributes to the progress and application of science with its publication.

Other selected journals from SCIRP are listed as below. Submit your manuscript to us via either submit@scirp.org or Online Submission Portal.