THE UNIVERSITY
OF QUEENSLAND
AUSTRALIA

Common Criteria Compliance for the Trusted Filter at EAL7 - Formal Arguments

Geoff Watson and Luke Wildman

January 2008

Technical Report SSE-2008-01

Division of Systems and Software Engineering Research
School of Information Technology and Electrical Engineering
The University of Queensland
QLD, 4072, Australia

http://www.itee.uq.edu.au/~sse

Common Criteria Compliance for the Trusted Filter at EAL7 – Formal Arguments

G Watson and L Wildman, ITEE, UQ

January 25, 2008

This research is funded by an Australian Research Council Discovery grant, DP0558408: *Analysing and Generating Fault-tolerant Real-time Systems.*

Contents

Acronyms and Initialisms

This section collects the main acronyms and initialisms used in this document.
Glossary The most important of these terms are also given a definition from the Common Criteria online documentation. Except where otherwise indicated, these definitions are from CC 3.1 Part 1 [4], Section 4 *Terms and definitions*.

CC Common Criteria

CEM Common Evaluation Methodology

DSD Defence Signals Directorate

EAL Evaluation Assurance Level

HLD High-Level Design

OSP Organizational Security Policy
– a set of security rules, procedures, or guidelines imposed (or presumed to be imposed) now and/or in the future by an actual or hypothetical organisation in the operational environment.

PIC Peripheral Interface Controller

PP Protection Profile

SAR Security Assurance Requirement

SFP Security Function Policy
–a set of rules describing specific security behaviour enforced by the TSF and expressible as a set of SFRs.

SFR Security Functional Requirement
– The SFRs define the rules by which the TOE governs access to and use of its resources, and thus information and services controlled by the TOE. (CC 3.1 Part 2 [5], Section 6 *Functional requirements paradigm*)

SPM Security Policy Model

ST Security Target
– an implementation-dependent statement of security needs for a specific identified TOE.

TDS TSF Design Documentation

TOE Target of Evaluation
– target of evaluation (TOE) a set of software, firmware and/or hardware possibly accompanied by guidance.

TSF TOE Security Functionality
– a set consisting of all hardware, software, and firmware of the TOE that must be relied upon for the correct enforcement of the SFRs.

TSFI TSF interface
– a means by which external entities (or subjects in the TOE but outside of the TSF) supply data to the TSF, receive data from the TSF and invoke services from the TSF.

TSP TOE Security Policy

B symbols

Logical:	
A or B	$A \lor B$
C and D	$C \land D$
P implies Q	$P \Rightarrow Q$
For all x such that S	$\forall x \bullet S$
There exists a y such that T	$\exists y \bullet T$
a equal b	$a = b$
c not equal d	$c \neq d$
Sets:	
a is a member of X	$a \in X$
b is not a member of Y	$b \notin Y$
The empty set	$\varnothing$
S intersect T	$S \cap T$
S subset of T	$S \subseteq T$
S proper subset of T	$S \subset T$
The set of integers from m to n	$m..n$
The natural numbers	$\mathbb{N}$
A finite set of natural numbers	$\mathbb{F}$
The power sets of set S	$\mathbb{P}(S)$
The non-zero natural numbers (etc.)	$\mathbb{N}_1$ etc.
Functions:	
f is a total function from X to Y	$f : X \rightarrow Y$
f is a partial function from X to Y	$f : X \nrightarrow Y$
The relation R restricted to domain D	$D \lhd R$
Assignments:	
Assign X the value V	$X := V$
Assign X some value in the set S	$X :\in S$
Perform assignments I and J in parallel	$I \parallel J$
Sequences:	
x is of type: finite sequence of t	$x \in seq(t)$
The prefix of sequence s upto index i	$s \downarrow i$
The suffix of sequence t from index j	$t \uparrow j$
Sequence s with element p prepended	$p \rightarrow s$
Sequence t with element q appended	$t \leftarrow q$
The reverse of a sequence s	$rev(s)$

Chapter 1

Introduction

1.1 Summary

The *Common Criteria (CC)* is an international standard for the evaluation of security products. It provides a framework within which the security aspects of products can be described, and within which such claims can be evaluated in a rigorous manner.

As is common with schemes for evaluating and certifying information technology (IT) systems for safety or security purposes, the Common Criteria uses a system of *assurance levels*. CC assurance levels are called *EALs* (Evaluation Assurance Levels). The EAL describes the degree of confidence that can be placed in a positive evaluation of the CC claims for a product. EALs range from 1 to 7, where EAL7 is the highest level.

The Australian Defence Signals Directorate (DSD) administers the Australian Information Security Evaluation Program. This involves oversight of CC evaluation. However, evaluation at the highest levels (EAL6 and EAL7) is rarely attempted. This report is part of an investigation into the practical issues of applying the Common Criteria at these high levels.

This report deals with EAL7 (CC version 3.1) Rev 1. 2006), which requires formal descriptions and proofs for some components. Specifically the report investigates the process of developing the *formal* support documents that are required by the evaluator. The report describes the experience of applying one applicable formal method to an actual device, which was developed in prototype by the DSD as a subject for exercises of this kind.

In summary, this report describes the formal development of a Trusted Filter design from formal specification to high-level design. This was done using the *Event-B Method*. The formal security policy of the trusted filter is also supplied. Formal proofs were completed that show that the formal specification and the high-level design satisfy the security policy and that the high-level design satisfies the formal specification. This document also describes how the formal parts above correspond to documents required by the CC.

The trusted filter design is based on a prototype design supplied by DSD. Several

design faults were discovered in the prototype design and so some aspects of the design were adjusted in order to allow a formal verification to be completed.

1.1.1 Document Overview

The structure of this document is as follows.

- The rest of this chapter contains the following.
  - A description of the prototype device – the Trusted Filter.
  - A more detailed description of the Common Criteria evaluation. process, and its requirements for formality at EAL7.
  - A brief overview of the B method.
- The following three chapters discuss in detail the application of the CC to the Trusted Filter for those parts that require formal methods to be applied.
  - Chapter 2 describes the development of a formal security policy.
  - Chapter 3 describes the development of formal security functional requirements.
  - Chapter 4 describes the development of a formal high level design.
- Finally, Chapter 5 discusses the lessons learned from the exercise.

1.2 The Trusted Filter

A *trusted filter* is a device which checks command data being sent from a secure to a non-secure environment and only allows a specific set of commands to pass. It is a *filter* because it is designed to block non-designated commands, and it is a *trusted* filter because the filtering is done for security purposes.

The Trusted Filter that is the object of this report is a test device built by DSD especially for the investigation of high-level evaluation procedures. The initial description of the device was as follows. [12]

> 1 INTRODUCTION
>
> The trusted filter demonstration device was developed to provide a model for research into fault analysis automation. The trusted filter has been designed to interface between a high security classification host and a device on a low security classification network. Unlike more complex commercial trusted filters, micro controllers or processors do not control this device. All core functionality is provided via hardware logic and clocking. It is expected that the trusted filter will be more secure and have a higher fault tolerance because of its hardware implementation.
>
> 2 FUNCTIONALITY

> A trusted filter is used for unidirectional data transfer between two systems of varying classification levels. The filter in such a scenario acts as a control point between the two systems determining the traffic flow between the two levels. The most common implementation of a trusted filter is between a higher classified system and a lower classified system in which traffic flow from low to high is blocked whilst traffic from high to low is allowed to flow. The trusted filter has been designed to connect to both systems via a serial RS-232 interface. This interface is handled by PIC micro-controllers whose function is to control the input and output of data into the trusted filter core in an appropriate format.

Further details can be found in the DSD design document [12]. Note that:

- The Trusted Filter has a single channel.
- The device is unidirectional. Responses from the insecure side do not pass through the filter.
- The command set is loaded at the time of manufacture, and there is no functionality for its replacement or update.

For our investigation we were supplied with a number of design documents, plus the VHDL description of the device (from the Protel tool). However, the implementation of the prototype evolved during the project and we were given successive VHDL models as changes were made. In consequence the design documentation did not match the final product. The discussion in this report is based on the final implementation. So, for instance, the detailed High-level Design in Chapter 4 is based on the 'reverse engineering' the implementation rather than the supplied high-level design (which was inconsistent with the final design).

1.3 The Common Criteria

The *International Common Criteria for Information Technology Security Evaluation*, usually known as the *Common Criteria (CC)*, is an international standard (ISO/IEC 15408) for the evaluation of security products. It provides a framework within which the security aspects of products can be described, and within which such claims can be evaluated in a rigorous manner. Authorities which adopt the CC framework establish a mechanism for evaluating claims made under the CC and publishing the results.

The CC was developed by an international collaboration between Canada, France, Germany, the Netherlands, the United Kingdom and the United States. Each country has its own evaluation and certification scheme to support the CC standard. The current version of the Common Criteria is version 3.1 Rev. 1, 2006 [2]. It is this version that is used throughout this report.

1.3.1 The Common Criteria Process

A basic concept of the CC framework is that of a *Target of Evaluation (TOE)*. A TOE is "a set of software, firmware and/or hardware possibly accompanied by guidance"

[4], which is the artifact that is the subject of the evaluation process. In the evaluation process the evaluating authority is provided with an example of the TOE, a statement of the security claims that are being made about the TOE, and supporting documentation as specified by the CC criteria for the desired evaluation level.

The primary focus of the evaluation of a TOE is described in the following extract from CC Part 2 sect 6, *Functional requirements paradigm* [5].

> TOE evaluation is concerned primarily with ensuring that a defined set of security functional requirements (SFRs) is enforced over the TOE resources. The SFRs define the rules by which the TOE governs access to and use of its resources, and thus information and services controlled by the TOE.
>
> The SFRs may, in turn, include multiple Security Function Policies (SFPs). Each SFP has a scope of control, that defines the subjects, objects, resources or information, and operations controlled under the SFP. All SFPs are implemented by the TSF (see below), whose mechanisms enforce the rules defined in the SFRs and provide necessary capabilities.
>
> Those portions of a TOE that must be relied on for the correct enforcement of the SFRs are collectively referred to as the TOE Security Functionality (TSF). The TSF consists of all hardware, software, and firmware of a TOE that is either directly or indirectly relied upon for security enforcement.

In some cases there is a generic *Protection Profile (PP)* covering the TOE. A PP a document that has itself been certified under the Common Criteria by an approved certification body. It describes a class of devices. There is currently no PP that covers Trusted Filters.

Since there is no applicable PP, an implementation-specific *Security Target (ST)* must be defined for the filter. The aim of a ST is to define what claims are to be evaluated. One function of a ST is to define the security problem that the TOE addresses. The security problem is defined as consisting of *threats*, *Organisational Security Policies (OSPs)* and *assumptions*. The ST also describes the solution provided by the TOE. This is done in terms of *security objectives*. Each security objective has one or more *Security Functional Requirements (SFRs)* identified for it. The SFRs must be shown to achieve their associated objectives.

A security objectives rationale is also provided that shows that if all security objectives are achieved, the security problem is solved: all threats are countered, all OSPs are enforced, and all assumptions are upheld. The relationship between the components of a ST is shown in Figure 1.1.

1.3.2 The CC Documents

The Common Criteria are described in four documents. These documents are as follows ([4, section 6.2.5]).

Part 1 – Introduction and general model [4] This is the introduction to the CC. It defines the general concepts and principles of IT security evaluation and presents a general model of evaluation.

Part 2 – Security functional components [5] Part 2 establishes a set of functional components that serve as standard templates upon which to base functional requirements for TOEs. CC Part 2 catalogues the set of functional components and organises them in families and classes.

Part 3 – Security assurance components [6] Establishes a set of assurance components that serve as standard templates upon which to base assurance requirements for TOEs. CC Part 3 catalogues the set of assurance components and organises them into families and classes. CC Part 3 also defines evaluation criteria for PPs and STs and presents seven pre-defined assurance packages which are called the Evaluation Assurance Levels (EALs).

Evaluation methodology [3] Describes the evaluation process for an evaluator.

1.3.3 Component Naming

Parts 2 and 3 define functional and assurance components respectively. Functional components are used in the Security Target to describe the functionality claimed for the TOE. The assurance components defined in CC Part 3 are used to assist the evaluator make a decision on these claims.

The documentation of CC components uses a four level hierarchical system.

- Classes, consisting of
- Families, consisting of

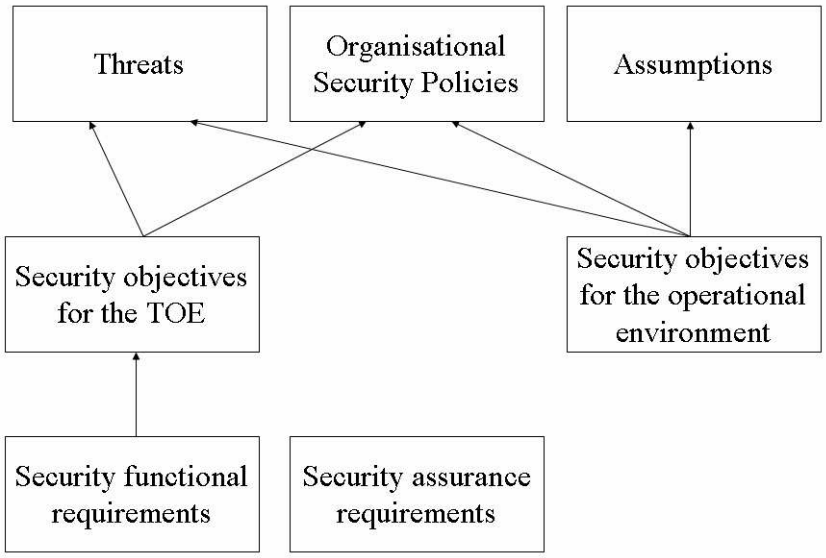

Figure 1.1: Parts of a Security Target

- Components, consisting of
- Elements.

For instance, Part 3 of the CC describes 10 *Assurance Classes* (see Figure 1.2). The development documentation, with which most of this report is concerned, is part of the *Development* class ADV. A typical element of this class is ADV_TDS.6.8 which is identified as follows.

— Class ADV – *Development*

— Family TDS – *TOE Development*

— Component 6 – *Complete semi-formal modular design with formal high-level design presentation*.

— Element 8 – *The formal specification of the TOE Security Function (TSF) subsystems shall describe the TSF using a formal style, supported by informal, explanatory text where appropriate*.

1.3.4 EAL7

When submitting a device such as the *Trusted Filter* for Common criteria evaluation, supporting documents must be tendered to demonstrate that the Security Target has been met. The documents required are determined by the assurance level being sought. This level also determines the level of formality required of the documents.

The assurance sought level is determined by the degree of confidence in the evaluation that is required for the device being evaluated. Assurance level EAL7 is the highest level, and is the only one that requires extensive formal descriptions and proofs (although a formal security policy is required for EAL5 and EAL6 as well). The CC describes the applicability of EAL7 as follows.

> EAL7 is applicable to the development of security TOEs for application in extremely high risk situations and/or where the high value of the assets justifies the higher costs. Practical application of EAL7 is currently limited to TOEs with tightly focused security functionality that is amenable to extensive formal analysis.

EAL7 requires the use of formal methods for the definition and verification of some parts of the design and implementation, and this document focuses on those tasks, since they are mostly novel to EAL7. The aim of the document is to demonstrate one way in which the formal requirements of a CC Security Target at EAL7 for the Trusted Filter could be met.

Note that besides formal analysis, EAL7 also requires a higher general standard of assurance than other levels, for instance a more rigorous standard of testing (which must also be independent). However, this report is only concerned with the application of formal methods in the CC process.

The formal requirements at level EAL7 are highlighted in Figure 1.2, and are as follows.

Assurance Class	Assurance components
ADV: Development	ADV_ARC.1 Security architecture description
	ADV_FSP.6 Complete semi-formal functional specification with additional formal specification
	ADV_IMP.2 Implementation of the TSF
	ADV_INT.3 Minimally complex internals
	ADV_SPM.1 Formal TOE security policy model
	ADV_TDS.6 Complete semi formal modular design with formal high-level design presentation
AGD: Guidance documents	. . .
ALC: Life-cycle support	. . .
ASE: Security Target evaluation	. . .
ATE: Tests	. . .
. . .	
AVA: Vulnerability assessment	. . .

Figure 1.2: Requirements for EAL7 Evaluation (formal requirements highlighted).
CC part 3 [6] - Section 8.9, Table 8

- *TOE Security Policy Model* (ADV_SPM.1)
- *Formal Functional Specification* (ADV_FSP.6)
- *Formal High Level Design* (ADV_TDS.6)

These three formal requirements are discussed in more detail in Chapters 2, 3 and 4.

1.4 The B Method

The B method [7, 1] is a well established formal program development method that was originally devised by J-R Abrial. It is one of the formal methods mentioned in the Common Criteria itself. The choice of the B method is discussed further in Section 5.2.

B has excellent tool support. There are a number of tool sets available. For this report we used the *B4free* system [10]. This is a toolset for academic use which is derived from *Atelier-B* [9] system that has been highly successful in industrial applications. *B4free* has an interactive proof management system called *Click-n-Prove* [10].

B has a well founded mathematical basis, but from a practical point of view it is based on the concept of a *B machine*. A machine looks rather like a module in an imperative program language, and its syntax is deliberately cast in the program language mould. (The mathematical theory has a different syntax.) The format of a B machine can be seen by examining the examples in Appendix A. Machines can declare variables, and define operations which have assignments, conditionals, case statements etc.

For the Trusted Filter we in fact used the *Event B* [8] formalism. This is an extension of the original B method, and was also developed by J-R Abrial. Event B allows one to model distributed systems and to verify the correctness of functionality distributed between different components. *Event B* is supported by the standard B toolset.

An Event-B model is made up of a *context* containing necessary definitions and constants, a list of *variables* that may be used in the model, an *invariant* which describes the types of the variables and any other properties of the filter that must be maintained, some *theorems* that may be derived from the invariant, an *initialization* that describes the initial values of the model variables, and a set of events.

The B method enables a developer to specify a system at a high abstract level and develop this down to actual code, all within the same framework, using the same tools, and broadly within the same language. B machines are arranged in a hierarchy at three different abstraction levels. The development proceeds from abstract specification to final implementation by a process of formal refinement. In this report we are only concerned with formalizing specifications and designs, so we do not use the implementation levels of the B method.

At each level the B tools can check the machines for consistency and type correctness. Consistency is checked by generating and proving a number of *proof obligations*. Many of these are trivial, and most can be proven automatically by the B proof tool. In addition the B tool can check the validity of the refinement of one B machine by another.

Since formal proof is such a key part of the B method there are extensive facilities, and a rich language, for expressing properties of machines in a formal way. B expresses such properties as logical predicates over mathematical theories. As with all formal methods the symbols used for this purpose can be intimidating to the non-expert. Some of the symbols most commonly used in the machines comprising the Trusted Filter development are listed on page 6.

Chapter 2

Security Policy Model

The *Security Policy Model* is defined in component **ADV_SPM.1** of the *Development* assurance class **ADV**. The component items are itemized below, they refer back to the *Security Functional Requirements* (SFRs) of the Security Target, and to the *TOE Security Functions* (TSFs). The TSFs are the hardware and software mechanisms that enforce the security policy – in the case of a simple device like the Trusted Filter, the TOE and TSF are the same.

ADV_SPM.1.1D The developer shall provide a formal security policy model for the policies that are formally modelled. They should identify the relevant portions of the statement of SFRs that comprise each of the modelled policies.

ADV_SPM.1.2D The developer shall provide a formal proof of correspondence between the model and any formal functional specification.

ADV_SPM.1.3D The developer shall provide a demonstration of correspondence between the model and the functional specification.

ADV_SPM.1.1C The model shall be in a formal style, supported by explanatory text as required, and identify the security policies of the TSF that are modelled.

ADV_SPM.1.2C For all policies that are modelled, the model shall define security for the TOE and provide a formal proof that the TOE cannot reach a state that is not secure.

ADV_SPM.1.3C The correspondence between the model and the functional specification shall be at the correct level of formality.

ADV_SPM.1.4C The correspondence shall show that the functional specification is consistent and complete with respect to the model.

ADV_SPM.1.5C The demonstration of correspondence shall show that the interfaces in the functional specification are consistent and complete with respect to the policies in the ADV_SPM.1.1D assignment.

In CC Part 3 [6] a CC Security Policy Model is described in the following terms. "[It equates] to whatever SFRs are being claimed. Therefore, the formal security policy model is merely a formal representation of the set of SFRs being claimed."

2.1 Security Function Policy

The Security Function Policy requirement for the *Trusted Filter* device in the Security Target is the following.

> *Only commands in the authorized dictionary are transmitted through the link passing to the insecure side.*

This informal presentation of the policy is adequate for most uses of the filter. However, it does not address some important exceptional cases. These cases are due to the fact that the trusted filter may be interrupted at any time due to power interruption and may also be reset at an arbitrary point in its operation by the reset switch. This interruption or reset may even occur while a command is being transmitted. In these cases we cannot guarantee that a complete command will be transmitted. Therefore we must re-express the security policy as follows.

> *Only commands in the authorized dictionary, or prefixes of commands in the authorized dictionary, are transmitted through the link passing to the insecure side.*

2.2 Formal Security Policy Model

The Security Target defines the Security Policy informally in a way that is is applicable at any EAL level. Evaluation at EAL7 requires that we provide a formal Security Policy for the implementation. To formalize this we need to make a number of decisions.

First we need to decide on the formal language to use. Since we are using the *Event B* language for formal modelling of the device design, we also specify the Formal Security Policy Model in B notation. The B-Method [7, 1] is mentioned in part 3 of the Common Criteria, as an example of applicable methods [6, § 603].

Next we need to decide at what level we should define the policy, this may be called the *granularity* of the policy. The policy talks about the "transmission of commands". However, because of the fact that the Trusted Filter actually sends and receives characters, and because of the fact that the trusted filter may be subject to arbitrary use of the reset switch or power interruption *in between character events*, a security policy which only makes reference to commands is impossible to implement because it is possible that only part of a command is output. Therefore we formalise the policy in terms of the sequence of characters that it may output.

In an effort to make the final formal security policy easier to understand we will define the policy in two steps. In the first step we will give an approximation to the policy that only considers complete commands and which therefore ignores the possibility of arbitrary resets or interruption. In the second step we will consider the possibility of partial commands and rephrase the policy as required.

2.2.1 Complete words

We now ignore the possibility of partial commands for the moment.

For any formal description we need to make definitions for the names and symbols that we will use. Let the set W (word) represent all possible words that may be input into the filter. The set of *proper words* PW is a non-empty subset of W.

$$PW \in \mathbb{P}_1(W)$$

PW represents all words that satisfy the input restrictions of the filter. I.e. that they must be of a restricted size and that they must only contain certain characters. The dictionary D is a predefined *finite* set of proper words.

$$D \in \mathbb{F}_1(PW)$$

This models the set of proper words that may be output by the filter.

In addition, we model the sequence of commands that are received by the device as win (or words in),

$$win \in seq(PW)$$

Note that this sequence is represented mathematically as a set of pairs of natural numbers and proper words. E.g., {(1,w1), (2,w2), (3,w3)} represents the sequence of length 3 with w1 in the first position and w2 in the second position and w3 in the third position.

The commands that are output by the device are modelled by wout.

$$wout \in \mathbb{N}_1 \pfun PW$$

The type of wout is a partial function from Natural numbers (but starting from 1) to proper words. It is like a sequence in that the words are ordered, but unlike a sequence, it allows gaps to appear in the output. E.g., {(1,w1), (3,w3)}. This models the fact that some input words will be filtered out but also preserves the indexes of the input words.

Using these definitions we can state the first approximation to the formal security policy as follows.

$$wout \in \mathbb{N}_1 \pfun D \land$$
$$wout \subseteq win$$

I.e., that the output only contains dictionary words and the output is made up of input words in the order in which they are input. Notice that we interpret the informal text *transmit through the device* in the second conjunct by saying that the output is a subset of the input. This means that all ordered pairs appearing in the output must have appeared in the input.

2.2.2 Dealing with partial words

The possibility of interruption and resets leads to the possibility of partial words being output from the filter and it also leads to the possibility of partial words being input into

the filter. A partial word may be output if a reset occurs during character-by-character output. A partial word may be input if a reset occurs during character-by-character input and the filter (re-) starts reading characters part-way through a word. If the part word input also matches a dictionary command then this leads to the possibility that an unintended command is output by the filter. This is a design problem with the original filter and ultimately this makes it impossible to prove an adequate security property. Therefore we have adapted the design to make an adequate security property possible.

The word-level model of the filter is not adequate because its level of granularity is too coarse. In order to address the possibility of partial words we must model the input and output at the character level. Characters are defined in the model `Char` in Appendix A as a set `C`. Rather than using words `W` and proper words `PW` we will use sequences of characters `seq(C)` (or strings) and proper strings `PS`.

In order to correctly distinguish complete words from partial words some structure needs to be added to strings. That is, a string is bracketed in begin and end word characters (bwc, ewc). These characters are defined in the model `Char` presented in Appendix A.

$$\begin{array}{l} \texttt{bwc} \in \texttt{C} \land \\ \texttt{ewc} \in \texttt{C} \land \\ \texttt{bwc} \neq \texttt{ewc} \end{array}$$

By bracketing strings with these characters, partial words can be recognised on the input by looking for properly bracketed words. Similarly, partial words may be recognised by any receiver of words downstream from the filter.

The model $PString$ of Appendix A defines the structure of a *proper* string `PS` as sequences bracketed by begin and end characters:

$$\begin{array}{l} \texttt{PS} = \{\texttt{s} \mid \texttt{s} \in \texttt{seq(C)} \land \texttt{size(s)} \leq \texttt{maxsize} \land \\ \qquad \exists\ \texttt{t.(t} \in \texttt{seq}_1\texttt{(callow)} \land \texttt{s} = \texttt{bwc} \rightarrow \texttt{t} \leftarrow \texttt{ewc)}\} \end{array}$$

where `callow` is some set of pre-defined *allowed* characters disjoint from the begin and end word characters,

$$\begin{array}{l} \texttt{callow} \in \mathbb{P}_1\texttt{(C)} \land \\ \texttt{\{bwc,ewc\}} \cap \texttt{callow} = \varnothing \end{array}$$

and `maxsize` is some pre-defined maximum size that must be greater than `3` in order for `PS` to contains some non-empty content.

$$\begin{array}{l} \texttt{maxsize} \in \mathbb{N}_1 \land \\ \texttt{3} \leq \texttt{maxsize} \end{array}$$

The model $CDict$ describes the dictionary as a set of proper strings.

$$\texttt{CD} \in \mathbb{F}_1\texttt{(PS)}$$

Let us now consider the input to the filter to be the infinite stream of characters present and future. We choose an infinite stream of inputs because we now model the filter being started, stopped, reset, restarted etc, on and on for its complete lifetime.

$$\texttt{cin} \in \mathbb{N}_1 \rightarrow \texttt{C}$$

As the stream is infinite we model by a total function from the Natural numbers (but starting from 1), which by the way of being total (defined for every natural number greater or equal to 1) must be contiguous.

The output of the filter will be some resultant stream of characters.

$$\texttt{cout} \in \mathbb{N}_1 \nrightarrow \texttt{C}$$

Given this model of the filter we may now express the security property as follows. First, as before, any complete word that appears in the output must be a dictionary word.

$$\forall\ \texttt{(i,j).(j} \in \mathbb{N}_1 \wedge\ \texttt{i} \in \texttt{1 .. j} \wedge \\ \texttt{squash(i .. j} \lhd \texttt{cout)} \in \texttt{PS} \Rightarrow\ \texttt{squash(i .. j} \lhd \texttt{cout)} \in \texttt{CD)}$$

Where `squash(i .. j◁cout)` is the subsequence from `i` upto `j` and `squash` compacts a substring with possible gaps into a sequence starting from `1`.

This predicate allows any downstream user of the output to trust any complete commands that it finds, however, it does not say anything about incomplete commands (a possible covert channel). Therefore we require that the output is formed only from nonempty prefixes or complete commands of the dictionary.

$$\forall\ \texttt{j.(j} \in 1 \mathinner{..} \mathit{size}(\mathit{cout}) \Rightarrow\ \exists\ \texttt{(i,cd).(i} \in \texttt{0..\ j-1} \wedge \\ \texttt{cd} \in \texttt{CD} \wedge\ \texttt{squash(i .. j} \lhd \texttt{cout)} \subseteq \texttt{cd))}$$

Where by the fact that both `squash(i .. j◁cout)` and `cd` are sequences, $(\_ \subseteq \_)$ implies that the first is a ***prefix*** of the second. This is because they both must start at `1`. The previous property, that complete words in the output must be dictionary words, is a consequence of considering just those j such that $\mathit{cout}(j) = \mathit{ewc}$ in this more general property (the last character of a proper word is always ewc).

Finally, the output is a subset of the input.

$$\texttt{cout} \subseteq \texttt{cin}$$

2.2.3 Discussion

The Formal Security Policy given above is deliberately a general one. For instance, it does not say that, where an input command is valid, then it *has* to be transmitted. Since it is general, it can be implemented in a number of ways. For instance, the TOE under consideration implements a restrictive policy, whereby any invalid command causes the device to shutdown and await a reset. However we consider this an implementation issue rather than a part of the security policy.

In addition, in order to address the problem of partial words correctly we have had to make a minor change to the TOE. That is, the original TOE only separated input commands using the RTN character. However, this protocol is not sufficient for users of the protocol to tell the difference between part-commands and complete commands. An implementable security policy required that partial commands be distinguishable from complete ones, and to effect this the protocol was extended to include both start and end word markers.

Chapter 3

Functional Specification

The *Functional Specification (FSP)* defines the functional behaviour required of the device in order to fulfil the Security Policy. The Functional Specification at EAL7 is defined using component **ADV_FSP.6** of assurance class ADV (see Figure 1.2). The elements that require formal specification are as follows.

ADV_FSP.6.2D The developer shall provide a formal presentation of the functional specification of the TSF.

ADV_FSP.6.2C The functional specification shall describe the TSF interface (TSFI) using a formal style.
(The TSFI is the means by which external entities supply data to the TSF, receive data from the TSF and invoke services from the TSF.)

ADV_FSP.6.9C The formal presentation of the functional specification of the TSF shall describe the TSFI using a formal style, supported by informal, explanatory text where appropriate.

The Security Target must define the functional requirements for the FSP using the components set out in CC Part 2 [5]. The primary component invoked is FDP_IFC.2 - complete information flow. (FDP is the class *User Data Protection*.)
The elements of FDP_IFC.2 are as follows. [5, page 66]

1. The TSF shall enforce the information flow control SFP on the subjects and information items and all operations that cause that information to flow to and from subjects covered by the SFP.

2. The TSF shall ensure that all operations that cause any information in the TOE to flow to and from any subject in the TOE are covered by an information flow control SFP.

We apply component FDP_IFC.2 to the Trusted Filter to ensure that the security policy defined in Chapter 2 is met. To satisfy the assurance component ADV_FSP.6, which is required at EAL7, we must provide a formal presentation of this functional specification. We define this in the *Event B* language [7, 8].

3.1 Formal Functional Specification

As discussed in the previous chapter, the functional specification is described at the character-processing level. A listing of this specification in the form of the Event-B model *CFilterR* is given on the following pages. An Event-B model is made up of a *context* containing necessary definitions and constants, a list of *variables* that may be used in the model, an *invariant* which describes the types of the variables and any other properties of the filter that must be maintained, some *theorems* that may be derived from the invariant, an *initialization* that describes the initial values of the model variables, and a set of events.

CFilterR has character variables ic into which characters are which read (from the secure side) and oc from which characters are written (to the insecure side). This is done by the operations $sendc$ and $recvc$ respectively. These events are commented as being *external* since they belong to the environment. CFilterR has the property that the transmission of a valid command by *sendc* can occur in parallel with reading the next word.

CFilterR also has the two operations $readc$ and $writec$ which are the internal correlates of $sendc$ and $recvc$. The internal and external read and write operations are synchronised by the pairs of counters ci/cr (characters input by $sendc$ / read internally by $readc$) and co/cw (characters output/written by $recvd/writec$).

CFilterR has four other operations which describe processing at the word level. These describe the acceptance of a new command (`input`), the checking of this command against the next in the dictionary (`chkc`), the successful location of the command in the dictionary (`release`), and the successful transmission of the command (`output`). These operations are controlled by the four variables p, r, t and s. which represent the number of commands being parsed, received, tested and sent (transmitted) respectively.
The invariant constrains p, r, s and t by the following condition.

$$p \in r \ldots r+1 \land p \in t \ldots t+1 \land$$
$$r \in t \ldots t+1 \land r \in s \ldots s+1 \land$$
$$t \in s \ldots s+1$$

This allows three distinct states:

- $p = r = t > s$
- $p > r = t = s$
- $p = r > t = s$

The first condition is the condition for new input to be parsed, and output to be sent, which may occur concurrently. The second is the conditions for the **input** event. The third is the condition for the **release** event.

There are many invariant conjuncts in the model. Most are *simple* in that just describe the type of a variable. Others are more complex, for instance,

$$\texttt{rev(ib)} \subseteq \texttt{rev(cin}\uparrow\texttt{cr)}$$

Expresses the property that the input buffer is always either empty or contains the last few characters read in from the buffer. (`cr` is the position of the last character read in from the buffer.) This is expressed using the function reverse (`rev`) which makes the reverse of a sequence. I.e. the reverse of the buffer is a *prefix* of the reverse of the input characters `cin` upto `cr`. All the invariant properties must be proved to be maintained by the events.

The events of the specification are now described in more detail.

readc $Readc$ parses the input character stream until the buffer contains a proper string. This event must occur before input (`p=r`). This model includes a scanning mode that allows input characters to be scanned without parsing up until a bwc is found. This is needed after initialization or after a reset because the bwc character may not be the next character in these states. For instance, after a reset, a part word may be left in the input.

input When $readc$ has finished parsing and the input buffer contains a proper string and the output event has finished sending **input** sets the abstract variable iw to the input buffer and increments r (thus changing the state).

chkc This event serves as an *anticipating event* for the electronic part of the filter, which is required to check that the input buffer is in the dictionary and also set the output buffer to the input buffer if it is in the dictionary.

release This event has the condition $t \neq r \wedge ib \in CD \wedge ob = ib$. Thus when the device is in the state $t \neq r$ (i.e. after an input), release is only enabled if the input command ib is in the dictionary and the output buffer ob also contains the contents of the input buffer. If a **release** does occur, it sets the abstract output word ow to iw, and changes state by incrementing t.

However, if $ib \notin CD$ then release is not enabled. This is a deadlocked situation (in that no other events are enabled except reset) in which the device shuts down, a state that can only be escaped by a reset event.

writec This event writes out the output buffer (using the index o) if it has been released ($t > s$).

output After the output buffer has been written it changes state by incrementing s.

Finally there is the operation *reset*. This is always enabled, and can thus occur after any other event. For instance, it can occur mid-way through either receiving a command (by a sequence of *send*cs) or transmitting a word (by a sequence of *recv*cs).

REFINEMENT
`CFilterR`
REFINES
`CFilter1`
SEES
`CDict, Char, PString, SeqSimp, PStringSimp1, PStringSimp2, PStringSimp3`
VARIABLES
`ib,ob,r,t,s,o,p,scan,  /* internal */`

```
   ic,oc,ci,cr,cw,co    /* external */
INVARIANT
   o ∈ 0 .. maxsize ∧
   (s/=t ⇒ cout\|/size(cout)-o = ob/|\o) ∧
   o  ≤ cw ∧
   (t/=r ⇒ o=0) ∧
   (s=t ⇒ o=0) ∧
   (ib ≠ <> ⇒ scan = F) ∧
   (ib = <> ⇒ scan = T)
THEOREMS
   (∃ ps.(ps:PS ∧ ib<-ic  ⊆ ps) ∧ ib = <> ⇒ ic = bwc) ∧
   (∃ ps.(ps:PS ∧ ib<-ic  ⊆ ps) ∧ ic ≠ bwc ⇒ ib ≠ <>) ∧
   cin/|\cr+1 = cin/|\cr<-cin(cr+1) ∧
   rev([bwc])  ⊆ rev(cin/|\cr  ← bwc)
INITIALISATION
  p,r,t,s,o := 0,0,0,0,0 ‖ ib,ob := <>,<> ‖ scan := T ‖
  ci,cr,co,cw := 0,0,0,0 ‖ ic :∈ C ‖ oc :∈ C
EVENTS
/* internal */
  readc = WHEN r = t ∧ cr ≠ ci ∧
    ((p=r ∧ ic = bwc) ∨ ∃ ps.(ps:PS ∧ ib<-ic  ⊆ ps) ∨ scan = T )
    THEN
      IF ∃ ps.(ps:PS ∧ ib<-ic  ⊆ ps) THEN
        ib,scan := ib  ← ic, F
      ELSIF ic = bwc THEN
        ib,scan := [bwc], F
      ELSIF scan = T THEN
        ib := <>
      END ‖
      cr := cr+1 ‖
      IF p=r THEN p:= p+1 END
    END;
  input = WHEN p ≠ r ∧ s = r ∧ ib ∈ PS THEN
        r := r+1
    END;
  chkc = WHEN t ≠ r THEN
        ob :| (ob ∈ seq(C) ∧ ∃ d.(d:CD ∧ ob  ⊆ d))
    END;
  release = WHEN t ≠ r ∧ ib ∈ CD ∧ ob = ib THEN
        t := t+1
    END;
  writec = WHEN s ≠ t ∧ cw = co ∧ o < size(ob)THEN
        oc,o,cw := ob(o+1),o+1,cw+1
    END;
  output = WHEN s ≠ t ∧ o = size(ob) THEN
        s := s+1 ‖ o := 0
```

```
    END;
  reset = BEGIN
        o,ib,ob,p,s,t,cr,co,scan := 0,<>,<>,r,r,r,ci,cw, T
    END;
/*external*/
  sendc = WHEN cr = ci THEN
         ic := cin(ci+1) ∥ ci := ci+1
    END;
  recvc = WHEN cw ≠ co THEN
         co := co +1
    END
END
```

3.2 Proof of correspondence with security policy

The functional specification has been proved to satisfy the following security policies using the Click-n-Prove interface to the Altelier-B prover.

```
∀ (i,j).(i ∈ 0 .. cw ∧ j ∈ 0 .. cw ∧ i ≤ j ∧ cout↑j↓i ∈ PS ⇒
      cout↑j↓i ∈ CD) ∧
∀ j.(j ∈ 0 .. cw ⇒
  ∃ (i,cd).(i∈0 .. j ∧ cd ∈ CD ∧ cout↑j↓i ⊆ cd))
∀ i.(i ∈ 0 .. cw ∧ cout↓i ∈ PS ⇒
       ∃ j.(j∈0 .. ci ∧ rev(cout↓i) ⊆ rev(cin↑j))) ∧
```

These are not expressed exactly the same as the formal policy expressed earlier. This is largely due to lack of time. However, the policy actually proved is very close to the one prescribed and is almost as strong. The main difference being that `cout` is a sequence rather than a partial function. This enables us to remove the necessity for the function squash. However it complicates the last security property which was expressed more simply earlier as `cout`$\subseteq$`cin`. Instead we prove that the end of `cout` is a prefix of `cin` if it is a proper string.

Chapter 4

High Level Design

Assurance level EAL7 requires a formal *High-Level Design* (see Figure 1.2). Although we were supplied with a number of design documents, as noted in Chapter 1, these did not match the final implementation. Also, as identified in Chapter 2, the actual implementation allows interruption at arbitrary points, resulting in the transmission of partial words. Thus the formal specification and design is based on the actual implementation and not the design documents supplied.

The implementation was supplied as a VHDL file generated by the Protel tool from the implementation design. Protel allows a high-level block design to be derived from this low-level VHDL. This block design is a useful starting point for understanding the structure at the high level, provided that the physical layout is constructed to correspond to the functional components. This was the case with the Trusted Filter.

The informal Protel high level design of the Trusted Filter is shown in Figure 4.1. It can be compared to the visualisation of the formal High-level Design shown later in 4.2.

This design shows three Protel "sheets", labelled: PICS and CLK, DICTIONARY and COMPARISON. The first of these contains both the input and output processing components, which we separate for the formal analysis. The formal high-level design therefore has four major components:

1. An *Input* handler.
2. A *Dictionary* component, which handles the retrieval of authorized words from the dictionary.
3. The *Comparator*, which manages the comparison of the input word with the authorized words retrieved from the dictionary.
4. An *Output* handler.

These components are described by the B machines *InputCOMP*, *DictCOMP*, *CompCOMP* and *OutputCOMP* respectively. These machines are given on pages 28-34 below.

One advantage of specifying the high-level design in the B language is that there is good tool support. In particular, the B tools can check the specification for consistency and type correctness. Consistency is checked by generating and proving a number of *proof obligations*. Most of these are trivial, and many of the rest can be proven automatically by the B support tools.

In addition B tools can check the *refinement* of one B machine by another which is closer to the implementation. This process is the basis of the B method. The four components which make up the HLD have been derived from the functional specification

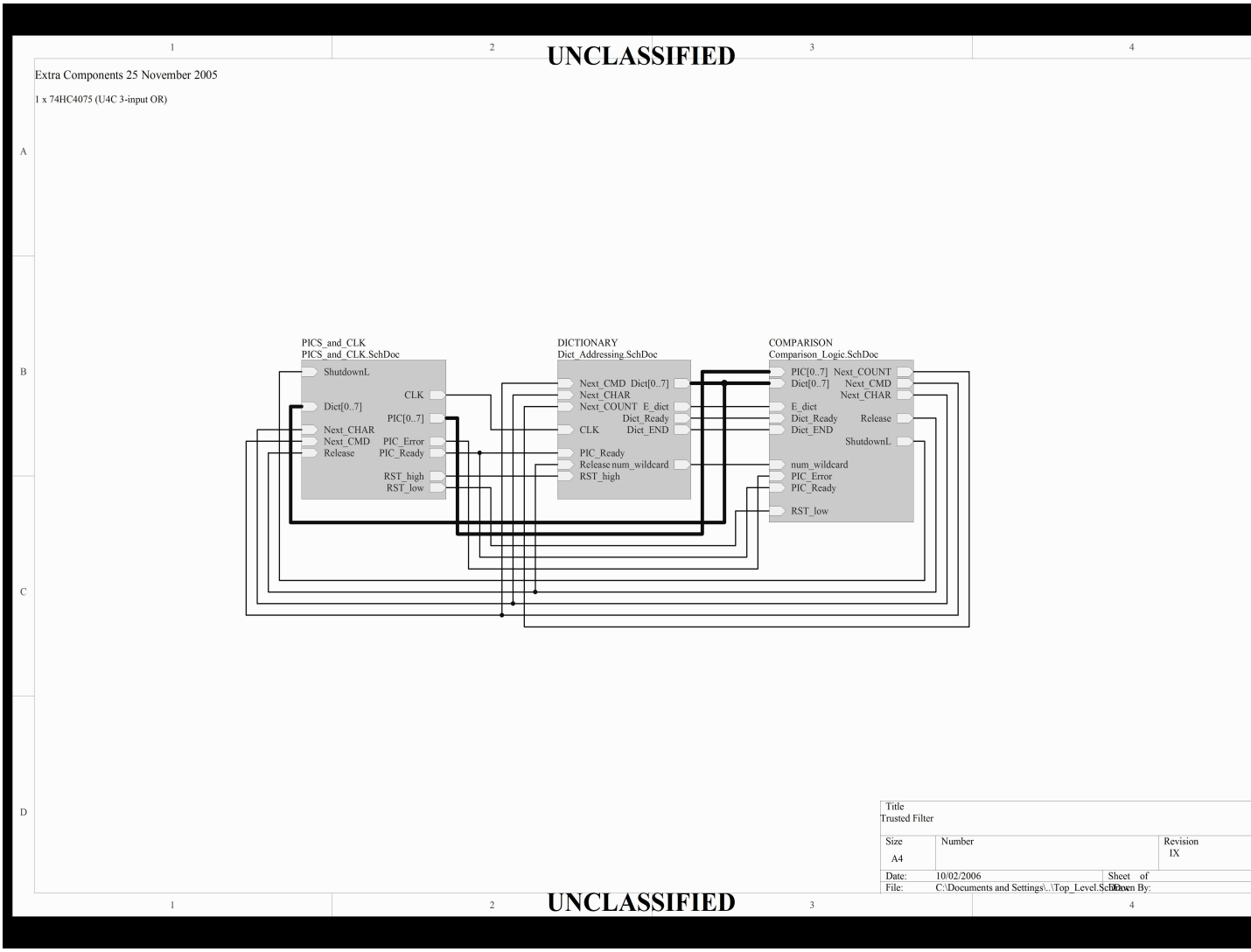

Figure 4.1: Informal High-level block design used in the Protel development

3.1 by a process of refinement and decomposition. This is done in a formal framework and can be checked by the B tools. This process allows an evaluator to check both the correctness of the high-level design, and its decomposition and traceability to the SFRs.

MODEL
InputCOMP
SEES
Char, PString, SeqSimp
VARIABLES
ib,iwi,scan,p, /* internal */
r,s,t,u,ic,ci,cr, /* external */
it,dt,itc,dtc,iwc
DEFINITIONS
boo(E) $\triangleq$ (E $=$ T);
InputReady $\triangleq$ (t $\neq$ r);
OPICReady $\triangleq$ (s $=$ r)
INVARIANT
ib $\in$ seq(C) $\wedge$
$\exists$ ps.(ps $\in$ PS $\wedge$ ib $\subseteq$ ps) $\wedge$
r $\in$ $\mathbb{N}$ $\wedge$
t $\in$ $\mathbb{N}$ $\wedge$
p $\in$ $\mathbb{N}$ $\wedge$
u $\in$ $\mathbb{N}$ $\wedge$
s $\in$ $\mathbb{N}$ $\wedge$
r $\in$ t .. t+1 $\wedge$
r $\in$ s .. s+1 $\wedge$
t $\in$ s .. s+1 $\wedge$
p $\in$ r .. r+1 $\wedge$
p $\in$ t .. t+1 $\wedge$
u $\in$ t .. t+1 $\wedge$
u $\in$ r .. r+1 $\wedge$
(r $\neq$ t $\Rightarrow$ ib $\in$ PS) $\wedge$
scan $\in$ BOOL $\wedge$
ic $\in$ C $\wedge$
ci $\in$ $\mathbb{N}$ $\wedge$
cr $\in$ ci-1 .. ci $\wedge$
it $\in$ $\mathbb{N}$ $\wedge$
iwc $\in$ C $\wedge$
iwi $\in$ $\mathbb{N}$ $\wedge$
(r $\neq$ t $\Rightarrow$ iwi $\in$ 0 .. size(ib)) $\wedge$
itc $\in$ $\mathbb{N}$ $\wedge$
dt $\in$ $\mathbb{N}$ $\wedge$
dt $\in$ it .. it+1 $\wedge$
dtc $\in$ $\mathbb{N}$ $\wedge$
dtc $\in$ itc .. itc+1 $\wedge$
(itc $\neq$ dtc $\Rightarrow$ r $\neq$ t)
THEOREMS
(s=r $\Rightarrow$ t=r) $\wedge$ (s\net$\Rightarrow$r=t)
INITIALISATION
p,r,t,s,u := 0,0,0,0,0 $\|$ ib := $\langle\rangle$ $\|$ scan := T $\|$

```
  ci,cr := 0,0 ‖ ic :∈ C ‖ it,dt:=0,0 ‖
  iwi,itc := 0,0 ‖ iwc :∈ C ‖
  dtc := 0
EVENTS
/* internal */
  readc = WHEN r = t ∧ cr ≠ ci ∧
      ((p=r ∧
      ic = bwc) ∨ ∃ ps.(ps ∈ PS ∧ ib←ic  ⊆ ps) ∨ scan = T )
    THEN
      IF  ∃ ps.(ps ∈ PS ∧ ib←ic  ⊆ ps) THEN
        ib,scan := ib←ic,F
      ELSIF ic = bwc THEN
        ib,scan := [bwc], F
      ELSIF scan = T THEN
        ib := ⟨⟩
      END ‖
      cr := cr+1 ‖
      IF p=r THEN p:= p+1 END
    END;
  input = WHEN p≠ r ∧ u ≠ r ∧ s = r ∧ ib ∈ PS THEN
        r := r+1 ‖ iwi:= 0
    END;
  nextiw = WHEN it ≠ dt THEN
       it,iwi := it+1,0
    END;
  nextiwc = WHEN it = dt ∧ itc ≠ dtc ∧ iwi<size(ib) THEN
       iwc := ib(iwi+1)‖ iwi,itc := iwi+1,itc+1
    END;
/*external*/
/*dictionary*/
  initd = WHEN u = t ∧ u = r THEN
        u:= u+1
    END;
  nextd = WHEN r\net ∧ it = dt THEN
        dt:= dt+1
    END;
  nextdc = WHEN r\net ∧ it = dt ∧ dtc = itc THEN
        dtc :=  dtc+1
    END;
/*comparitor*/
  release = WHEN t ≠ r ∧ itc = dtc THEN
        t := t+1
    END;
/*output PIC*/
  output = WHEN s ≠ t THEN
        s := s+1
```

```
    END;
/*reset*/
  reset = BEGIN
        ib,p,u,s,t,cr,scan := ⟨⟩,r,r,r,r,ci,T ||
        it := dt ||
        itc := dtc ||
        iwi:= 0
    END;
/*environment*/
  sendc = WHEN cr = ci THEN
         ic :∈ C || ci := ci+1
    END
END
```

MODEL
DictCOMP
SEES
CDict, Char, PString, SeqSimp
VARIABLES
tw,dc,dt,V,DE,dec,twc, /*internal*/

ED,it,itc,twi,dtc,dcc,ot,otc, /*external*/
r,t,s,u
DEFINITIONS
boo(E) $\triangleq$ (E $=$ T);
InputReady $\triangleq$ (t $\neq$ r);
NEXTCMD1 $\triangleq$ (dt $=$ dc);
NEXTCMD2 $\triangleq$ (ot $=$ dc);
NEXTCMD3 $\triangleq$ (it $=$ dc);
NEXTCHAR $\triangleq$ (dtc $=$ dcc);
OPICReady $\triangleq$ (s $=$ r);
DICTReady $\triangleq$ (dec $\neq$ dcc);
DICTAvail $\triangleq$ (otc $\neq$ dtc);
RELEASE $\triangleq$ (DICTReady $\wedge$ boo(ED))
INVARIANT
r $\in$ $\mathbb{N}$ $\wedge$
s $\in$ $\mathbb{N}$ $\wedge$
t $\in$ $\mathbb{N}$ $\wedge$
r $\in$ t .. t+1 $\wedge$
r $\in$ s .. s+1 $\wedge$
t $\in$ s .. s+1 $\wedge$
u $\in$ r .. r+1 $\wedge$
u $\in$ t .. t+1 $\wedge$
tw $\in$ CD $\wedge$
dc $\in$ $\mathbb{N}$ $\wedge$
dt $\in$ $\mathbb{N}$ $\wedge$
dt $\in$ dc .. dc+1 $\wedge$
V $\subseteq$ CD $\wedge$
twc $\in$ C $\wedge$
twi $\in$ 0 .. size(tw) $\wedge$
dtc $\in$ $\mathbb{N}$ $\wedge$
dcc $\in$ $\mathbb{N}$ $\wedge$
dtc $\in$ dcc .. dcc+1 $\wedge$
ot $\in$ $\mathbb{N}$ $\wedge$
ot $\in$ dt .. dt+1 $\wedge$
ot $\in$ dc .. dc+1 $\wedge$
otc $\in$ $\mathbb{N}$ $\wedge$
dtc $\in$ otc .. otc+1 $\wedge$
otc $\in$ dcc .. dcc+1 $\wedge$
it $\in$ $\mathbb{N}$ $\wedge$

$$
\begin{array}{l}
\quad \texttt{itc} \in \mathbb{N} \land \\
\quad \texttt{dt} \in \texttt{it} \ .. \ \texttt{it+1} \land \\
\quad \texttt{it} \in \texttt{dc} \ .. \ \texttt{dc+1} \land \\
\quad \texttt{itc} \in \texttt{dcc} \ .. \ \texttt{dcc+1} \land \\
\quad \texttt{dtc} \in \texttt{itc} \ .. \ \texttt{itc+1} \land \\
\quad (\texttt{t} \neq \texttt{r} \land \texttt{dtc} \neq \texttt{dcc} \Rightarrow 1 \leq \texttt{twi} \land \texttt{twc} = \texttt{tw(twi)}) \land \\
\quad (\texttt{t} = \texttt{r} \Rightarrow \texttt{dtc} = \texttt{dcc}) \land \\
\quad (\texttt{dt} = \texttt{dc} \Rightarrow \texttt{dtc} = \texttt{dcc}) \land \\
\quad \texttt{DE} \in \texttt{BOOL} \land \\
\quad (\texttt{boo(DE)} \Rightarrow \texttt{V} = \texttt{CD}) \land \\
\quad (\neg\ \texttt{boo(DE)} \land \texttt{dt} = \texttt{dc} \ \Rightarrow \texttt{V} \neq \texttt{CD}) \land \\
\quad \texttt{ED} \in \texttt{BOOL} \land \\
\quad \texttt{dec} \in \texttt{dcc} \ .. \ \texttt{dcc+1} \land \\
\quad \texttt{itc} \in \texttt{dec} \ .. \ \texttt{dec+1} \land \\
\quad \texttt{dtc} \in \texttt{dec} \ .. \ \texttt{dec+1} \land \\
\quad (\neg\ \texttt{boo(DE)} \land \texttt{dec} \neq \texttt{dcc} \ \Rightarrow \texttt{V} \neq \texttt{CD}) \land \\
\quad (\texttt{DICTReady} \Rightarrow \texttt{itc} \neq \texttt{dcc} \land \texttt{dtc} \neq \texttt{dcc} \land \texttt{otc} \neq \texttt{dcc}) \land \\
\quad (\texttt{DICTReady} \Rightarrow \texttt{it} \neq \texttt{dc} \land \texttt{ot} \neq \texttt{dc}) \land \\
\quad (\texttt{DICTReady} \Rightarrow \texttt{InputReady}) \land \\
\quad (\texttt{dt} \neq \texttt{ot} \Rightarrow \texttt{InputReady}) \land \\
\quad (\texttt{dt} \neq \texttt{dc} \Rightarrow \texttt{InputReady}) \land \\
\quad (\texttt{DICTAvail} \Rightarrow \texttt{InputReady}) \land \\
\quad (\texttt{u} \neq \texttt{r} \Rightarrow \texttt{ED} = F) \land \\
\quad (\texttt{it} \neq \texttt{dt} \Rightarrow \texttt{ED} = F) \land \\
\quad (\texttt{itc} \neq \texttt{dtc} \Rightarrow \texttt{ED} = F) \land \\
\quad (\texttt{dt} \neq \texttt{dc} \land \texttt{dec} = \texttt{dcc} \Rightarrow \texttt{ED} = F) \land \\
\quad (\texttt{itc} \neq \texttt{dec} \Rightarrow \texttt{ED} = F) \land \\
\quad (\texttt{dec} \neq \texttt{dcc} \land \texttt{ED} = F \Rightarrow \texttt{twc} \neq \texttt{ewc}) \land \\
\quad (\texttt{dec} \neq \texttt{dcc} \land \texttt{ED} = F \Rightarrow \texttt{twi} < \texttt{size(tw)}) \land \\
\quad (\texttt{r} \neq \texttt{t} \land \texttt{dec} = \texttt{dtc} \land \texttt{boo(ED)} \Rightarrow \texttt{twi} = \texttt{size(tw)} \land \texttt{twc} = \texttt{ewc}) \land \\
\quad (\texttt{it} \neq \texttt{dt} \Rightarrow \texttt{itc} = \texttt{dcc}) \land \\
\quad (\texttt{dt} = \texttt{dc} \Rightarrow \texttt{itc} = \texttt{dtc}) \land \\
\quad (\texttt{ot} = \texttt{dc} \Rightarrow \texttt{otc} = \texttt{dtc}) \land \\
\quad (\texttt{otc} \neq \texttt{dtc} \Rightarrow \texttt{ot} \neq \texttt{dc}) \land \\
\quad (\texttt{dt} \neq \texttt{dc} \Rightarrow \texttt{ot} \neq \texttt{dc}) \land \\
\quad (\texttt{dt} \neq \texttt{dc} \Rightarrow \texttt{u} \neq \texttt{t}) \land \\
\quad (\texttt{RELEASE} \Rightarrow \ \neg(\texttt{NEXTCMD1}))
\end{array}
$$

THEOREMS

$$
\begin{array}{l}
\quad (\texttt{s=r} \Rightarrow \texttt{t=r}) \land (\texttt{s\textbackslash net} \Rightarrow \texttt{r=t}) \land \\
\quad (\texttt{ot} = \texttt{dc} \Rightarrow \texttt{ot} = \texttt{dt}) \land \\
\quad (\texttt{dtc} = \texttt{dcc} \Rightarrow \texttt{otc} = \texttt{dcc}) \land \\
\quad (\texttt{otc} \neq \texttt{dcc} \Rightarrow \texttt{otc} = \texttt{dtc}) \land \\
\quad (\texttt{otc} \neq \texttt{dtc} \Rightarrow \texttt{otc} = \texttt{dcc}) \land \\
\quad (\texttt{dt} = \texttt{dc} \Rightarrow \texttt{it} = \texttt{dc}) \land (\texttt{it} \neq \texttt{dt} \Rightarrow \texttt{dt} \neq \texttt{dc}) \land \\
\quad (\texttt{dtc} = \texttt{dcc} \Rightarrow \texttt{itc} = \texttt{dcc}) \land (\texttt{itc} \neq \texttt{dtc} \Rightarrow \texttt{dtc} \neq \texttt{dcc})
\end{array}
$$

INITIALISATION

```
  r,t,s,u := 0,0,0,0 ‖ tw:: CD ‖
  V := ∅ ‖ it,dc,dt,dec,ot:=0,0,0,0,0 ‖ DE,ED :=
  F,F ‖
  itc := 0 ‖
  twi,dtc := 0,0 ‖ twc :∈ C ‖
  dcc,otc := 0,0
EVENTS
/* internal */
  initd = WHEN u = t ∧ u = r THEN
        DE, ED, V := F, F, ∅ ‖
        u:= u+1
    END;
  nextd = WHEN  u ≠ t ∧ NEXTCMD1 ∧ ¬ boo(DE) ∧ dt≠ot THEN
        tw:: CD-V ‖ dt:= dt+1 ‖ V:=V∪{tw} ‖ ED := F ‖
        twi:= 0
    END;
  nextdc = WHEN dt\nedc ∧ it = dt ∧ NEXTCHAR ∧ dtc=dcc ∧
                twi<size(tw) ∧ dtc =otc ∧ dtc = dec
    THEN
        twc := tw(twi+1) ‖ twi,dtc := twi+1, dtc+1
    END;
  decode = WHEN itc ≠ dec ∧ otc ≠ dec  THEN
        DE :| ((boo(DE))  ⇔ (V=CD)) ‖
        ED :| ((boo(ED))  ⇔ (twc=ewc))‖
        dec:= dec+1
    END;
/*external components*/
/*input PIC*/
  input = WHEN  OPICReady ∧ u ≠ r  THEN
        r := r+1
    END;
  nextiw = WHEN it ≠ dt THEN
       it := it+1
    END;
  nextiwc = WHEN it = dt ∧ itc ≠ dtc THEN
       itc := itc+1
    END;
/*comparator*/
  chkc = WHEN DICTReady THEN
      IF ¬ boo(ED) THEN
         dcc:: dcc  ..  dcc+1
      ELSE CHOICE
         dc,dcc := dc,dcc OR
         dc,dcc := dc+1,dcc+1
      END END
    END;
```

```
  release = WHEN RELEASE THEN
        t := t+1 ‖  dc := dc+1 ‖ dcc := dcc+1
    END;
/*output PIC*/
  nextow = WHEN  InputReady ∧ NEXTCMD2 THEN
        ot := ot+1
    END;
  nextowc = WHEN DICTAvail THEN
        otc := otc+1
  END;
  output = WHEN s\net THEN
        s := s+1
    END;
/*reset*/
  reset = BEGIN
        s,t,u,DE,ED := r,r,r,F,F ‖
        dc,it,ot := dt,dt,dt ‖
        dec,itc,dcc,otc := dtc,dtc,dtc,dtc ‖
        twi:= 0 ‖ V:= ∅
    END
END
```

MODEL
CompCOMP
SEES
CDict, Char, PString, SeqSimp
VARIABLES
dc,dt,dec,twc,iwc,ED, /* all external*/
it,itc,dtc,dcc,ot,otc,
r,t,u
DEFINITIONS
boo(E) $\triangleq$ (E $=$ T);
InputReady $\triangleq$ (t $\neq$ r);
NEXTCMD1 $\triangleq$ (dt $=$ dc);
NEXTCMD2 $\triangleq$ (ot $=$ dc);
NEXTCMD3 $\triangleq$ (it $=$ dc);
NEXTCHAR $\triangleq$ (dtc $=$ dcc);
MATCH $\triangleq$ (twc $=$ iwc);
DICTReady $\triangleq$ (dec $\neq$ dcc);
DICTAvail $\triangleq$ (otc $\neq$ dtc);
RELEASE $\triangleq$ (MATCH $\wedge$ DICTReady $\wedge$ boo(ED))
INVARIANT
r $\in$ $\mathbb{N}$ $\wedge$
t $\in$ $\mathbb{N}$ $\wedge$
r $\in$ t .. t+1 $\wedge$
u $\in$ r .. r+1 $\wedge$
u $\in$ t .. t+1 $\wedge$
iwc $\in$ C $\wedge$
twc $\in$ C $\wedge$
dc $\in$ $\mathbb{N}$ $\wedge$
dt $\in$ $\mathbb{N}$ $\wedge$
dt $\in$ dc .. dc+1 $\wedge$
dtc $\in$ $\mathbb{N}$ $\wedge$
dcc $\in$ $\mathbb{N}$ $\wedge$
dtc $\in$ dcc .. dcc+1 $\wedge$
ot $\in$ $\mathbb{N}$ $\wedge$
ot $\in$ dt .. dt+1 $\wedge$
ot $\in$ dc .. dc+1 $\wedge$
otc $\in$ $\mathbb{N}$ $\wedge$
dtc $\in$ otc .. otc+1 $\wedge$
otc $\in$ dcc .. dcc+1 $\wedge$
it $\in$ $\mathbb{N}$ $\wedge$
itc $\in$ $\mathbb{N}$ $\wedge$
dt $\in$ it .. it+1 $\wedge$
it $\in$ dc .. dc+1 $\wedge$
itc $\in$ dcc .. dcc+1 $\wedge$
dtc $\in$ itc .. itc+1 $\wedge$
(t $=$ r $\Rightarrow$ dtc $=$ dcc) $\wedge$

```
  (dt = dc ⇒ dtc = dcc) ∧
  ED ∈ BOOL ∧
  dec ∈ dcc  ..  dcc+1 ∧
  itc ∈ dec  ..  dec+1 ∧
  dtc ∈ dec  ..  dec+1 ∧
  (r\net ∧ dec = dtc ∧ boo(ED) ⇒ twc = ewc) ∧
  (u ≠ r ⇒ ED = F) ∧
  (itc ≠ dtc ⇒ ED = F) ∧
  (itc ≠ dec ⇒ ED = F) ∧
  (it ≠ dt ⇒ ED = F) ∧
  (it ≠ dt ⇒ itc = dcc) ∧
  (dt = dc ⇒ itc = dtc) ∧
  (ot = dc ⇒ otc = dtc) ∧
  (otc ≠ dtc ⇒ ot ≠ dc) ∧
  (dt ≠ dc ⇒ ot ≠ dc) ∧
  (DICTReady ⇒ itc ≠ dcc ∧ dtc ≠ dcc ∧ otc ≠ dcc) ∧
  (DICTReady ⇒ it ≠ dc ∧ ot ≠ dc) ∧
  (DICTReady ⇒ InputReady) ∧
  (dt ≠ ot ⇒ InputReady) ∧
  (dt ≠ dc ⇒ InputReady) ∧
  (DICTAvail ⇒ InputReady) ∧
  (u ≠ r ⇒ ED = F) ∧
  (it ≠ dt ⇒ ED = F) ∧
  (dt ≠ dc ∧ dec = dcc ⇒ ED = F) ∧
  (itc ≠ dtc ⇒ ED = F) ∧
  (itc ≠ dec ⇒ ED = F) ∧
  (dec ≠ dcc ∧ ED = F ⇒ twc ≠ ewc)
THEOREMS
  (ot = dc ⇒ ot = dt) ∧
  (dtc = dcc ⇒ otc = dcc) ∧
  (otc ≠ dcc ⇒ otc = dtc) ∧
  (otc ≠ dtc ⇒ otc = dcc) ∧
  (dt = dc ⇒ it = dc) ∧ (it ≠ dt ⇒ dt ≠ dc) ∧
  (dtc = dcc ⇒ itc = dcc) ∧ (itc ≠ dtc ⇒ dtc ≠ dcc)
INITIALISATION
  r,t,u := 0,0,0 ||
  it,dc,dt,dec,ot:=0,0,0,0,0 || ED := F ||
  itc := 0 || iwc :∈ C ||
  dtc := 0 || twc :∈ C ||
  dcc,otc := 0,0
EVENTS
/* internal */
  chkc = WHEN DICTReady THEN
      IF MATCH ∧ ¬ boo(ED) THEN
        dcc:= dcc+1
      ELSIF ¬ (MATCH) THEN
```

```
        dc,dcc := dc+1, dcc+1
      END
    END;
  release = WHEN RELEASE THEN
        t := t+1 ‖ dc,dcc := dc+1, dcc+1
    END;
/*external*/
  input = WHEN t = r ∧ u ≠ r THEN
        r := r+1
    END;
  nextiw = WHEN it ≠ dt THEN
       it := it+1
    END;
  nextiwc = WHEN it = dt ∧ itc ≠ dtc THEN
       iwc :∈ C ‖ itc := itc+1
    END;
  initd = WHEN  u = t ∧ u = r THEN
        ED,u := F,u+1
    END;
  nextd = WHEN  NEXTCMD1 ∧ ¬ boo(ED) ∧ dt\neot THEN
        dt:= dt+1 ‖ ED := F
    END;
  nextdc = WHEN  dt\nedc ∧ it = dt ∧ NEXTCHAR ∧ ¬ boo(ED) ∧
                dtc =otc ∧ dtc = dec
    THEN
        twc :∈ C ‖ dtc := dtc+1
    END;
  decode = WHEN itc ≠ dec ∧ otc ≠ dec  THEN
        ED :| ((boo(ED))  ⇔ (twc=ewc))‖
        dec:= dec+1
    END;
  nextow = WHEN InputReady ∧ NEXTCMD2 THEN
        ot := ot+1
    END;
  nextowc = WHEN DICTAvail THEN
        otc := otc+1
  END;
  reset = BEGIN
        t,ED := r,F ‖
        dc,it,ot := dt,dt,dt ‖
        dec,itc,dcc,otc := dtc,dtc,dtc,dtc
    END
END
```

MODEL
OutputCOMP
SEES
Char, PString, SeqSimp
VARIABLES
ob,o, /* internal */
r,t,s,oc,cw,co,ot,dc,otc,dtc,twc /* external */
INVARIANT
ob ∈ seq(C) ∧
∃ ps.(ps ∈ PS ∧ ob ⊆ ps) ∧
r ∈ ℕ ∧
t ∈ r-1 .. r ∧
s ∈ r-1 .. r ∧
s ∈ t-1 .. t ∧
(s ≠ t ⇒ ob ∈ PS) ∧
(s = r ⇒ ob = ⟨⟩) ∧
o ∈ 0 .. maxsize ∧
oc ∈ C ∧
cw ∈ ℕ ∧
o ≤ cw ∧
co ∈ cw-1 .. cw ∧
(t\ner ⇒ o=0) ∧
(s=t ⇒ o=0) ∧
dc ∈ ℕ ∧
twc ∈ C ∧
dtc ∈ ℕ ∧
ot ∈ dc .. dc+1 ∧
ot ∈ ℕ ∧
otc ∈ ℕ ∧
dtc ∈ otc .. otc+1 ∧
(ot = dc ⇒ otc = dtc) ∧
(otc ≠ dtc ⇒ ot ≠ dc) ∧
(r≠ t ∧ otc ≠ dtc ⇒ ∃ ps.(ps ∈ PS ∧ ob←twc ⊆ ps)) ∧
(r=t ⇒ otc = dtc)
THEOREMS
(s=r ⇒ t=r) ∧ (s\net⇒r=t)
INITIALISATION
r,t,s,o := 0,0,0,0 ‖ ob := ⟨⟩ ‖
co,cw := 0,0 ‖ oc :∈ C ‖ dc,ot:=0,0 ‖
dtc := 0 ‖ twc :∈ C ‖ otc := 0
EVENTS
/* internal */
nextow = **WHEN** r\net ∧ ot = dc **THEN**
ot := ot+1 ‖ ob := ⟨⟩
END;
nextowc = **WHEN** r≠t ∧ otc ≠ dtc **THEN**

```
        ob := ob←twc ‖ otc := otc+1
  END;
  writec = WHEN s ≠ t ∧ cw = co ∧ o < size(ob)THEN
        oc,o,cw := ob(o+1),o+1,cw+1
    END;
  output = WHEN s ≠ t ∧ o = size(ob) THEN
        s := s+1 ‖ o := 0 ‖ ob := ⟨⟩
    END;
/*external*/
/* input PIC */
  input = WHEN s = r  THEN
        r := r+1
    END;
/* dictionary */
  nextdc = WHEN r≠t ∧ dtc = otc ∧ ot ≠ dc THEN
        twc :| (twc ∈ C ∧
        ∃ ps.(ps ∈ PS ∧ ob←twc  ⊆ ps)) ‖ dtc := dtc+1
    END;
/* comparator */
  chkc = WHEN r≠t ∧ otc = dtc ∧ ot ≠ dc THEN
        dc :∈ dc .. dc+1
    END;
  release = WHEN t ≠ r ∧ ob ∈ PS ∧ ot ≠ dc ∧ otc = dtc THEN
        t := t+1
    END;
/*reset*/
  reset = BEGIN
        o,ob,s,t,co := 0,⟨⟩,r,r,cw ‖
        otc := dtc ‖ ot:= dc
    END;
/*environment*/
  recvc = WHEN cw ≠ co THEN
         co := co +1
    END
END
```

4.1 High-level Design Components

4.1.1 Input handler

The input handler *InputCOMP* has ten events. Event *sendc* models the sending of characters to the filter from the environment. It models an **external** event and is not refined. Event *reset* is also an environment event, and models resetting the device when the reset button is pressed.

Four **internal** events process data that is directly handled by *InputCOMP*. Corresponding events in the other three machines synchronise with these where necessary. The events *readc* and *input* handle the input and output of data. I/O is at character level and characters are assembled into words (which is handled by the input and output PICs in the implementation). The events *nextiw* and *nextiwc* handle the passing of an input word, character by character, from the input PIC when needed by the comparator.

The other four events synchronise with actions in the other three components. *nextd* and *nextdc* synchronise with the dictionary, *release* with the comparator and *output* with *OutputCOMP*. When these events occur data local to *InputCOMP* is updated to keep track of events in other components. These are also **external** events relative to the Input handler.

4.1.2 Dictionary

The dictionary handler *DictCOMP* has 12 events. Again there is a *reset* event. The other external synchronisation events are *input*, *nextw*, *nextwc* (synchronising with Input), *chkc*, *release* (Comparator) and *output*, *nextow*, *nextowc* (Output).

Three internal events process data that is directly handled by *DictCOMP*. The events *nextd* and *nextdc* keep track of the next dictionary word and next dictionary character respectively, while *decode* checks for end of word and end of dictionary.

4.1.3 Comparator

The comparator *CompCOMP* has 11 events. The external events are *input*, *nextw*, *nextwc* (Input), *nextd*, *nextdc*, *decode* (Dictionary) and *output*, *nextow*, *nextowc* (Output), plus the em reset event.

The internal events are *chkc* and *release*. *chkc* compares the current character in the input word with the next character of the dictionary word. If there is match then the dictionary character counter is incremented. If there is not a match then the dictionary word counter is incremented. The *release* event occurs when there is a match on the last character of the input word. This increments t - the number of words matched, which in turn causes the system to commence transmitting the current output word (which at this point matches the input word).

4.1.4 Output handler

The output handler *OutputCOMP* also has 11 events. Again we have a reset *event* and the external events, *input*, *nextw*, *nextwc* (Input), *nextd*, *nextdc*, *decode* (Dictionary) and *chkc*, *release* (Comparator).

There are four internal events are *writec*, *output*, *nextow* and *nextowc*. These handle output words and characters in a parallel way to the events in *InputCOMP*.

Similarly there is event *recvc*, which is an external event that models an output character being received by the environment from the filter's output.

4.2 High-level Block Design

The block structure of the formal high-level design is illustrated in Figure 4.2. This is an abstract block design, constructed in the *SIFA* analysis tool [11] from the formal B design described by the machines *InputCOMP*, *DictCOMP*, *CompCOMP* and *OutputCOMP*. Each major component (i.e. each top-level B-machine) is shown as a block, and shared variables are shown as *ports* connected to these blocks.

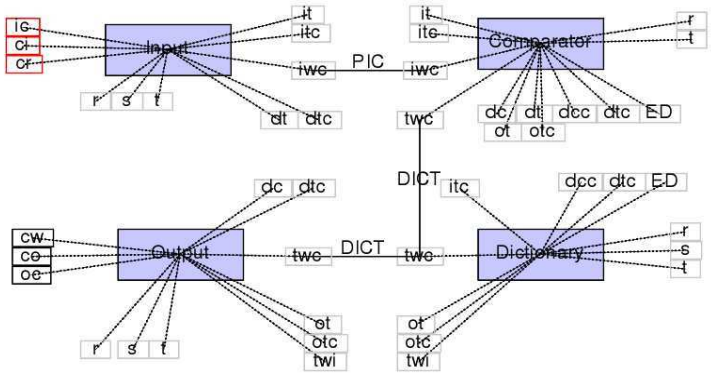

Figure 4.2: High-level block design derived from the formal specification

The formal requirements of CC EAL7 stop at the high-level design level. The tracing of this design to the implementation is only required to be semi-formal. Key abstractions in the HLD with respect to the actual implementation include the following.

- The simple dictionary in the HLD actually represents a sub-system which implements the handling of numeric wild-cards in the dictionary entries. These wild-cards match any of the digits 0-9.
- The comparison of characters is atomic in the HLD (i.e. at the byte level) whereas this is implemented on a bit-wise cascading comparison circuit.
- The implementation includes error checking of the input and output PICs (which effects shut-down on failure).
- Some synchronization is implemented by timing assumptions.

The internal variables (ports) in Figure 4.2 can be matched by name to generate connections which correspond to the connections between the components in the implementation. The correspondence is not a simple one-to-one at this level, since in the formal high-level design the connections represent shared variables and not logical signals. For example, as described on page 21, the variables r, s, t and p together define three states or stages of processing a command. However, these states are defined by

relationships between the variables in the design and will be implemented by sequencing between signals. Nevertheless the implementation using signals can be derived from the design in terms of shared variables.

Chapter 5

Comments on the Process

5.1 Ease of use of the Common Criteria

Neither of the investigators had used the Common Criteria directly before. Eventually it proved to be reasonably usable, however the learning curve was quite steep especially in respect to the quantity of documentation and the number of acronyms to be mastered.

During the course of the investigation CC version 3.1 was officially adopted over the earlier version 2. This caused some reworking, but version 3 was felt to be a noticeable improvement over the previous version, it was sensibly simplified and the organisation of the material was clearer and more logical.

These remarks are qualified by our focus on the formal requirements of EAL7. Thus we only worked in detail with the small portion of the CC that was relevant to our investigation. For instance, we cannot comment on how easy it is to achieve completeness in the required documentation, since we only had to deal with a single aspect.

5.2 The Use of B as the Formal Method

Although the Common Criteria is just a framework that can be used in a variety of circumstance, its documentation does provide some guidance on the use of formal methods. This is in Annex 5 of CC Part 3 [6] *Supplementary material on formal methods*.

Although it does not recommend any particular formal methods it does give four 'examples'.

- The Z specification language.
- ACL2.
- Isabelle.
- The B method.

These are four of the most well known and popular formal methods. Annex 5 also states that "if the developer uses a formal system which is already accepted by the certification body the evaluator can rely on the level of formality and strength of the system and focus on the instantiation of the formal system to the TOE specifications and correspondence proofs". Thus using a standard formal system such as B assists the evaluator as well as the developer.

The CC supplement notes that there are "two aspects of formal methods: the specification language that is used for formal expression, and the theorem prover that mathematically proves the completeness and correctness of the formal specification". In respect to the latter, one advantage of the B method [7, 1] is that there is excellent tool support. In particular, the B tools can check the specification for consistency and type correctness. Consistency is checked by generating and proving a number of *proof obligations*. Most of these are trivial, and many of the rest can be proven automatically by the B support tools. In addition B tools can check the *refinement* of one B machine by another one which is closer to the implementation.

For the Trusted Filter we in fact used the *Event B* [8] formalism, which is an extension of the original B method. Event B allows one to model distributed systems and to verify the correctness of functionality distributed between different components. Since *Event B* is just an extension of 'standard' B, it is supported by the standard toolset.

In one way the efficiency of the B Toolset causes a difficulty for evaluators. Most B proof obligations are discharged automatically by the B proof tool, but it is not easy to record and document this process. One way to handle this is for evaluators to satisfy themselves that the B proof tool performs automatic proof correctly, and to verify a level of confidence in the B toolset itself – as a published EAL level for use of the toolkit. If this were done, then the evaluation of any particular *use* of the B tool as part of a CC certification case (up to the accepted EAL) would simply require an audit that the B tool was used correctly.

5.3 Formalising the Trusted Filter

5.3.1 Structure of the Formal Model

As noted in Section 1.2 we were supplied with a number of development documents for the Trusted Filter. However, none of these were consistent with each other, since the filter had evolved during its development. Since the context of the exercise was a CC evaluation, which is focused on the final product, the correctness of our formal model was judged against the final Protel circuit diagram.

Nevertheless, our understanding of the practical functionality of the filter was derived, at first, from the high-level design documents. This resulted in an iterative process of model construction. A model was constructed according to our high-level understanding, a lower level model was derived from this, discrepancies with the actual construction of the device were found and the high-level model was re-visited and corrected to accommodate the new insight.

There were several major iterations of this kind.

- Initially we started bottom-up from an early version of the Protel block model

Figure 4.1. But since this model matched neither the original requirements, nor the final circuits, we started again top-down.

- The second model formalized the processing of a single command, i.e. a single cycle: receive command, check command, transmit or shutdown. However, this did not handle shutdown properly, since this is essentially a state that exists across a sequence of commands.
- The next formalisation was in terms of sequences of commands, to handle shutdown. This too was found to be insufficient, since it could not handle the synchronization between the various components, especially when it was realised that input and transmission were concurrent processes, and that either could be interrupted at any time by a reset.
- The difficulty with synchronization required that we change the *granularity* of the formal model. A word-level model was inappropriate and we had to change the granularity to a *character-based* model. This could not be achieved by refinement from the word-based model, because reset was enabled more often in the character-base one. Instead it was necessary to define operations at a character level from the start, so, for example, the input and output operations could be interrupted between any pair of characters.
- The final major change to the formalisation was the introduction of *Proper Strings* (see Section 2.2.2). This was necessary in order for an insecure-side receiver to resynchronise after an interruption such as a reset. To achieve this it is necessary to have both start word *and* end word markers in the transmitted stream of characters.

During this iterative construction of the formal model, several bugs found were found in the design and implementation.

- The command representation was inadequate to define a policy that was strong enough to achieve its basic objective. Words needed to have both start and end control characters.
- There is a race condition between input and output which requires changes to the PIC code to fix.
- There is also a race condition between the release and next-char signals. This results in a delay to the release signal which may allow the data on the dictionary output lines to be changed before it is transmitted by the output PIC. Incorrect data may then be transmitted. This requires a new signal to correct.

5.3.2 The Functional Specification

Overall the aim of the modelling process was to define a B model that served as a formal functional specification of the filter. The B method then allows a high-level design to be derived from this specification. The possibility of continuing the B refinement to derive a model at the VHDL level was considered. We think that this is feasible, but it

is outside of the scope of the project (and beyond what is envisaged by the Common Criteria).

During the iterations described above two major specification decisions had to be made.

- Should the specification be simple e.g. commands, or complex (chars with resets)
- Should the specification describe shutdown as a variable or a (deadlock) state.

5.3.3 The Security Policy

Decisions were also made about the appropriate policy, but this was done less directly than for the security functionality. As mention previously (page. 16), the CC describes a Security Policy Model as "merely a formal representation of the set of SFRs being claimed". Thus, in one sense, the SPM is simply derived from the SFRs. However, from a modelling perspective, the SPM describes the most important features of the SFR model, and thus much of the discussion about how the models should be constructed was done at level of the SPM.

Key questions that arose during development of the SPM and which determined basic characteristics of the final model include the following.

- What was appropriate for partial domain separation.
- Should the policy refer to a single command or sequences of commands.
- Should interruption/reset be part of the formalised policy or part of the assumptions.
- Should covert channels/noninterference be addressed.
- What was the appropriate granularity-level for modelling.

5.3.4 The High-level Design

The Common Criteria does not define explicitly what a *High-level Design* should consist of, although it does describe the relationship between the HLD and other development artefacts. In the context of this report, one of the most important features of the HLD is that it is the'lower limit' of the CC's requirements for formality at EAL7. The design must be formalized, but its relationship to the implementation need only be demonstrated semi-formally.

We chose to place the high-level design at a similar level to the draft documents that we had received from DSD (see Figures 4.1 and 4.2). This was an exploratory investigation, and much more evidence and experience is necessary before a fully informed choice on the optimal level the HLD could be made.

For the Trusted Filter a good choice for HLD would seem to be one that minimizes formal development but which has enough detail that it can be proven to possess the required fault-tolerant and security properties. In this case, to have separated Input and

Output and no data line between them, with all Output coming from dictionary. Thus there is no need for the HLD to model implementation details like wild cards or specific components such as multi-plexors etc.

One characteristic that seemed important for the plausibility of the design is that the wiring between the various components is finalised in the HLD. No new inter HLD component wires should be added during the further (semi-formal) development. Although they can be removed by suitable timing assumptions or other cleverness. This ensures that the formal properties proven for the HLD carry through to the implementation.

5.3.5 Proofs

The B method generates logical theorems (obligations) requiring proof, both when B machines are defined (in order to prove that the machines are well formed and consistent), and also when B machines are elaborated (refined) (in order to show that the refinement preserves the requisite properties). In the case of the filter, the security properties were also safety properties which are preserved by refinement. This is the simplest case. In other developments this is may not always be the case, and the properties have to be proved again for final HLD.

The B development environment has a proof tool that, in many case, will prove such obligations automatically, but which also allows the developer to prove difficult cases 'by hand' (with machine assistance to manage the proof). In this project 90% of proofs were completed automatically by the tool. 10% had to be done by hand, but proofs were often similar to each other. A typical way of proceeding with such proofs is to prove subsidiary *lemmas* that encapsulate useful properties of the system. These can then be used in the proofs of similar obligations. For the filter, the proofs of lemmas involving properties of proper strings were hard, but these were the key to proving the security properties.

In general the proofs were easier for the better specification, while the failure to prove obligations for some earlier versions of the specification indicated faults in specification or basic filter design problems.

5.3.6 Final Recommendations

1. Get the specification right
    - The hardest thing was getting the specification right.
    - A bad specification was very difficult to develop further.
    - Once the specification was right, the development was much easier.
    - The development required intimate understanding of refinement.
2. Do development top-down from specification.
    - The best design evolves from a good specification and an understanding of the issues.

- For example the problem of partial words was not addressed in the implementation at all and it is a fundamental design aspect.

Bibliography

[1] The B-Method. http://vl.fmnet.info/b/. Visited 28th Nov. '06. 13, 16, 44

[2] *CC v3.1/CEM v3.1*. September 2006. http://niap.bahialab.com/cc-scheme/cc_docs/index.cfm. 9

[3] *CCITSE v3.1 - Common Methodology for Information Technology Security Evaluation*. September 2006. http://www.commoncriteriaportal.org/public/files/CEMV3.1R1.pdf. 11

[4] *CCITSE v3.1 - Part 1: Introduction and General Model*. September 2006. http://www.commoncriteriaportal.org/public/files/CCPART1V3.1R1.pdf. 5, 10, 91

[5] *CCITSE v3.1 - Part 2 Security Functional Components*. September 2006. http://www.commoncriteriaportal.org/public/files/CCPART2V3.1R1.pdf. 5, 10, 11, 20

[6] *CCITSE v3.1 - Part 3 Security Assurance Components*. September 2006. http://www.commoncriteriaportal.org/public/files/CCPART3V3.1R1.pdf. 11, 13, 16, 43

[7] J.-R. Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, New York, NY, USA, 1996. 13, 16, 20, 44

[8] Jean-Raymond Abrial and Louis Mussat. Introducing dynamic constraints in b. In *B '98: Proceedings of the Second International B Conference on Recent Advances in the Development and Use of the B Method*, pages 83–128, London, UK, 1998. Springer-Verlag. 14, 20, 44

[9] ClearSy. Atelier B web site. http://www.atelierb.societe.com/index_uk.htm. Visited 10th Dec. '06. 13

[10] ClearSy. B4free and Click'n'Prove web site. http://www.b4free.com/. Visited 10th Dec. '06. 13

[11] T. McComb and L. P. Wildman. SIFA: A tool for evaluation of high-grade security devices. In *ACISP 2005, 10th Australasian Conference on Information Security and Privacy*, Lecture Notes in Computer Science, pages 230 – 241. Springer-Verlag, 2005. 41

[12] Vicky Moyle and Luke Sledziona. Trusted filter design overview, 2005. 8, 9, 94

Appendix

B Models

Appendix A

Input and Output

Char

Defines the set of allowed characters $callow$ (ASCII in the implementation), and the beginning and end of word markers bwc and ewc.

MODEL
Char
SETS
C
CONSTANTS
bwc, ewc, callow
PROPERTIES
bwc ∈ C ∧ /*start word char*/
ewc ∈ C ∧ /*end word char*/
callow ∈ POW(C) ∧ /*allowed characters */
bwc ≠ ewc ∧
{bwc,ewc} ∩ callow = ∅
END

PString

Defines the type PS of permitted strings (which are the character-based equivalent of words).

- Permitted strings start with bwc and end with ewc.
- Permitted strings have a maximum size $maxsize$.

MODEL
PString
SEES
Char, SeqSimp
CONSTANTS
PS, maxsize, ps
PROPERTIES
maxsize $\in \mathbb{N}_1$ $\wedge$
3 $\leq$ maxsize $\wedge$
PS $\subseteq$ seq(C) $\wedge$
ps $\in$ PS $\wedge$
PS = {s | s $\in$ seq(C) $\wedge$ size(s) $\leq$ maxsize $\wedge$
$\exists$ t.(t $\in$ seq_1(callow) $\wedge$ s = bwc->t<-ewc)}
THEOREMS
PS $\subseteq$ seq(C) $\wedge$

$\forall$ ps.(ps:PS $\Rightarrow$ [bwc] $\subseteq$ ps) $\wedge$

$\forall$ (ps,c).(ps:PS $\wedge$ c:C $\wedge$ [c] $\subseteq$ ps $\Rightarrow$ c = bwc) $\wedge$

$\forall$ ps.(ps:PS $\Rightarrow$ ps|>{ewc} = {size(ps)|->ewc}) $\wedge$

$\forall$ ps.(ps:PS $\Rightarrow$ ps|>{bwc} = {1|->bwc})
END

CDict

This (re)defines the dictionary as a finite set of permitted strings.

MODEL
CDict
SEES
Char, PString
CONSTANTS
CD
PROPERTIES
CD $\in$ $\mathbb{F}_1$(PS)
THEOREMS
CD $\subseteq$ PS
END

CIO

This model captures the pure character I/O interface aspects of the filter. It was used to derive the specification of the filter.

MODEL
CIO
SEES
Char
CONSTANTS
cin
PROPERTIES
cin ∈ ℕ → C
VARIABLES
ic,oc,ci,cr,co,cw,cout /* external */
INVARIANT
ic ∈ C ∧
oc ∈ C ∧
ci ∈ ℕ ∧
cr ∈ ci-1 .. ci&
cw ∈ ℕ ∧
co ∈ cw-1 .. cw ∧
cout ∈ seq(C) ∧
cw = size(cout)
INITIALISATION
ci,cr,co,cw:= 0,0,0,0 ‖ ic :∈ C ‖ oc :∈ C ‖ cout := <>
EVENTS
/*internal*/
readc = **WHEN** cr ≠ ci **THEN**
cr := cr+1
END;
writec = **WHEN** cw = co **THEN**
ANY c **WHERE** c ∈ C **THEN**
oc := c ‖ cw := cw+1 ‖ cout := cout ← c
END
END;
reset = **BEGIN**
cr,co := ci,cw
END;
/*external*/
sendc = **WHEN** cr = ci **THEN**
ic := cin(ci+1) ‖ ci := ci+1
END;
recvc = **WHEN** cw ≠ co **THEN**
co := co+1

END
END

CFilter0

This model captures the high level safety properties of the filter that are sufficient to prove the formal security policy (in both the approximate and final forms). However, this model is not sufficient to form the formal specification of the filter because the specification of `readc` event is too weak: it is not guarenteed to parse the input into the input buffer properly. For instance, this machine may repeatedly output the same word without parsing a new word into the input buffer `ib`, i.e. `input` does not wait for `readc`. Also, it may continously parse the input characters without ever loading the characters into the input buffer `ib`.

MODEL
`CFilter0`
SEES
`CDict, Char, PString, SeqSimp, PStringSimp1, PStringSimp2, PStringSimp3`
CONSTANTS
`cin`
PROPERTIES
$\texttt{cin} \in \mathbb{N}_1 \rightarrow \texttt{C}$
VARIABLES

```
cout,win,wout,        /* proof */
iw, ow,               /* abstract */
ib,ob,r,t,s,          /* internal */
ic,oc,ci,cr,cw,co     /* external */
```

INVARIANT

$$
\begin{array}{l}
\texttt{cout} \in \texttt{seq(C)} \land \\
\texttt{win} \in \texttt{seq(PS)} \land \\
\texttt{wout} \in \mathbb{N}_1 \nrightarrow \texttt{CD} \land \\
\texttt{iw} \in \texttt{seq(C)} \land \\
\texttt{ow} \in \texttt{seq(C)} \land \\
\texttt{ib} \in \texttt{seq(C)} \land \\
\texttt{ob} \in \texttt{seq(C)} \land \\
\texttt{r} \in \mathbb{N} \land \\
\texttt{s} \in \mathbb{N} \land \\
\texttt{t} \in \mathbb{N} \land \\
\texttt{r = size(win)} \land \\
\texttt{r} \in \texttt{t .. t+1} \land \\
\texttt{r} \in \texttt{s .. s+1} \land \\
\texttt{t} \in \texttt{s .. s+1} \land \\
(\texttt{r} \neq \texttt{t} \Rightarrow \texttt{ib} \in \texttt{PS} \land \texttt{iw} \in \texttt{PS} \land \texttt{iw = ib} \land \texttt{iw = win(r)}) \land \\
(\texttt{s} \neq \texttt{t} \Rightarrow \texttt{ob} \in \texttt{CD} \land \texttt{ow} \in \texttt{CD} \land \texttt{ob = iw} \land \texttt{ow = ob} \land \texttt{ow = win(r)}) \land
\end{array}
$$

```
   (s = r ⇒ rev(ob)  ⊆ rev(cout)) ∧
   ic ∈ C ∧
   oc ∈ C ∧
   ci ∈ ℕ ∧
   cr ∈ ℕ ∧
   ci ∈ cr .. cr+1 ∧
   cw ∈ ℕ ∧
   cw = size(cout) ∧
   co ∈ ℕ ∧
   cw ∈ co .. co+1 ∧
   size(ib)  ≤ ci ∧
   (ci ≠ 0 ⇒ ic = cin(ci)) ∧
/*   ∃ ps.(ps:PS ∧ ib  ⊆ ps) ∧ */
   rev(ib)  ⊆ rev(cin/|\cr) ∧
   ∃ j.(j:0 .. ci ∧ rev(iw)  ⊆ rev(cin/|\j)) ∧
/*   ∃ ps.(ps:PS ∧ ob  ⊆ ps) ∧ */
/*   ∃ cd.(cd:CD ∧ ob  ⊆ cd) ∧ */
   ∃ i.(i:0 .. cw ∧ cout\|/i  ⊆ ob) ∧
   wout  ⊆ win ↑ s ∧
   ∀ i.(i ∈ 0 .. cw ∧ cout\|/i ∈ PS ⇒ cout\|/i ∈ CD) ∧
   ∀ (i,j).(i ∈ 0 .. cw ∧ j ∈ 0 .. cw ∧ i  ≤ j ∧ cout/|\j\|/i ∈ PS ⇒
         cout/|\j\|/i ∈ CD) ∧
   ∀ i.(i ∈ 0 .. cw ∧ cout\|/i ∈ PS ⇒
         ∃ j.(j:0 .. ci ∧ rev(cout\|/i)  ⊆ rev(cin/|\j))) ∧
   ∀ j.(j ∈ 1 .. cw ⇒
     ∃ (i,cd).(i∈0 .. j-1 ∧ cd : CD ∧ cout/|\j\|/i  ⊆ cd))
THEOREMS
  (s=r ⇒ t=r) ∧ (s ≠ t ⇒ r=t) ∧
  ∀ i.(i ∈ ℕ ⇒ cin/|\i ∈ seq(C))
INITIALISATION
  r,t,s := 0,0,0 ‖ ib,iw,ob,ow := <>,<>,<>,<> ‖
  ci,cr,co,cw := 0,0,0,0 ‖ ic :∈ C ‖ oc :∈ C ‖
  cout,win,wout := <>, <>, <>
EVENTS
/* internal */
  readc = WHEN r = t ∧ cr ≠ ci
    THEN
      ib :| (ib ∈ seq(C) ∧ rev(ib)  ⊆ rev(cin/|\cr+1) ) ‖
      cr := cr+1
    END;
  input = WHEN s = r ∧ ib ∈ PS THEN
        iw,r := ib,r+1 ‖ win := win  ← ib
    END;
  chkc = WHEN t ≠ r THEN
        ob :|(ob ∈ seq(C) )
    END;
```

```
  release = WHEN t ≠ r ∧ ib ∈ CD ∧ ob = ib THEN
        t := t+1 ‖ ow := ob
    END;
  writec = WHEN s ≠ t ∧ cw = co THEN
    ANY c, cout0 WHERE c∈C ∧ cout0 : seq(C) ∧ cout0 = cout THEN
        oc := c ‖
        cout :|(cout = cout0<-c ∧ ∃ i.(i:0 .. size(cout0) ∧ cout\|/i  ⊆ ob))
    END ‖
    cw:= cw+1
    END;
  output = WHEN s ≠ t ∧ rev(ob)  ⊆ rev(cout) THEN
        s := s+1 ‖ wout(s+1) := ow
    END;
  reset = BEGIN
        ib,ob,s,t,cr,co := <>,<>,r,r,ci,cw
    END;
/*external*/
  sendc = WHEN cr = ci THEN
         ic := cin(ci+1) ‖ ci := ci+1
    END;
  recvc = WHEN cw ≠ co THEN
         co := co + 1
    END
END
```

CFilter1

This model strengthens the guard so that it incorporates parsing and scaning. The variable `p` (for parsing) is used to make sure that parsing occurs before an input event may occur. The variable `scan` is used to allow the filter to read ahead after a reset event or initialisation. This is because the next character in the input stream may not be a begin-word character (`bwc`) at the start or after reset event (for instance if a reset occurs part way through a word). Without the scan variable, the filter would continuously shutdown unnecessarily after a reset. Scanning mode is only entered initially or after reset. After a normal input event we require that the next character is a bwc. Hence the guard (`p = r ∧ ic = bwc`). Note that if `scan` is `FALSE` and `p≠r` and the next character does not form a proper extension of the buffer then the `readc` event will not be enabled so no more input will occur until after a reset. However, output may continue concurrently even though input is disabled.

While the guard is now correct, the body of the `readc` event is still too weak as it still does not require any characters to be put in the input buffer.

```
REFINEMENT
   CFilter1
```

```
REFINES
   CFilter0
SEES
   CDict, Char, PString, SeqSimp, PStringSimp1, PStringSimp2, PStringSimp3
VARIABLES
   cout,                /* proof */
   ib,ob,r,t,s,p,scan,  /* internal */
   ic,oc,ci,cr,cw,co    /* external */
INVARIANT
   p ∈ ℕ ∧
   p ∈ r  ..  r+1 ∧
   p ∈ t  ..  t+1 ∧
   scan ∈ BOOL ∧
   ∃ d.(d:CD ∧ ob  ⊆ d) ∧
   ∃ ps.(ps:PS ∧ ib  ⊆ ps)
INITIALISATION
  p,r,t,s := 0,0,0,0 ‖ ib,ob := <>,<> ‖  scan := T ‖
  ci,cr,co,cw := 0,0,0,0 ‖ ic :∈ C ‖ oc :∈ C ‖
  cout := <>
EVENTS
/* internal */
  readc = WHEN r = t ∧ cr ≠ ci ∧
     ((p=r ∧ ic = bwc) ∨ ∃ ps.(ps:PS ∧ ib<-ic  ⊆ ps) ∨ scan = T)
    THEN
      ib :| (ib ∈ seq(C) ∧ rev(ib)  ⊆ rev(cin/|\cr+1) ∧
            ∃ ps.(ps:PS ∧ ib  ⊆ ps)) ‖
      cr := cr+1 ‖
      IF p = r THEN p := p+1 END ‖
      IF ic = bwc THEN scan := F END
    END;
  input = WHEN p ≠ r ∧ s = r ∧ ib ∈ PS THEN
        r := r+1
    END;
  chkc = WHEN t ≠ r THEN
        ob :|(ob ∈ seq(C) ∧ ∃ d.(d:CD ∧ ob  ⊆ d) )
    END;
  release = WHEN t ≠ r ∧ ib ∈ CD ∧ ob = ib THEN
        t := t+1
    END;
  writec = WHEN s ≠ t ∧ cw = co THEN
    ANY c, cout0 WHERE c∈C ∧ cout0 : seq(C) ∧ cout0 = cout THEN
        oc := c ‖
        cout :|(cout = cout0<-c ∧ ∃ i.(i:0 .. size(cout0) ∧ cout\|/i  ⊆ ob))
    END ‖
    cw:= cw+1
    END;
```

```
  output = WHEN s ≠ t ∧ rev(ob)  ⊆ rev(cout) THEN
        s := s+1
    END;
  reset = BEGIN
        ib,ob,s,t,p,cr,co,scan := <>,<>,r,r,r,ci,cw,T
    END;
/*external*/
  sendc = WHEN cr = ci THEN
         ic := cin(ci+1) ‖ ci := ci+1
    END;
  recvc = WHEN cw ≠ co THEN
         co := co + 1
    END
END
```

CFilterR

This final version of CFilter is sufficient to serve as the specification of the filter as it correctly adds characters to the input buffer when they form a proper extension of the input buffer. In addition, the `writec` and `output` events have been refined in such a way that they may now be implemented in code.

```
REFINEMENT
   CFilterR
REFINES
   CFilter1
SEES
   CDict, Char, PString, SeqSimp, PStringSimp1, PStringSimp2, PStringSimp3
VARIABLES
   ib,ob,r,t,s,o,p,scan,  /* internal */
   ic,oc,ci,cr,cw,co    /* external */
INVARIANT
   o ∈ 0 .. maxsize ∧
   (s/=t ⇒ cout\|/size(cout)-o = ob/|\o) ∧
   o  ≤ cw ∧
   (t/=r ⇒ o=0) ∧
   (s=t ⇒ o=0) ∧
   (ib ≠ <> ⇒ scan = F) ∧
   (ib = <> ⇒ scan = T)
THEOREMS
   (∃ ps.(ps:PS ∧ ib<-ic  ⊆ ps) ∧ ib = <> ⇒ ic = bwc) ∧
   (∃ ps.(ps:PS ∧ ib<-ic  ⊆ ps) ∧ ic ≠ bwc ⇒ ib ≠ <>) ∧
   cin/|\cr+1 = cin/|\cr<-cin(cr+1) ∧
   rev([bwc])  ⊆ rev(cin/|\cr  ← bwc)
```

```
INITIALISATION
  p,r,t,s,o := 0,0,0,0,0 ‖ ib,ob := <>,<> ‖ scan := T ‖
  ci,cr,co,cw := 0,0,0,0 ‖ ic :∈ C ‖ oc :∈ C
EVENTS
/* internal */
  readc = WHEN r = t ∧ cr ≠ ci ∧
    ((p=r ∧ ic = bwc) ∨ ∃ ps.(ps:PS ∧ ib<-ic  ⊆ ps) ∨ scan = T )
    THEN
      IF ∃ ps.(ps:PS ∧ ib<-ic  ⊆ ps) THEN
        ib,scan := ib  ← ic, F
      ELSIF ic = bwc THEN
        ib,scan := [bwc], F
      ELSIF scan = T THEN
        ib := <>
      END ‖
      cr := cr+1 ‖
      IF p=r THEN p:= p+1 END
    END;
  input = WHEN p ≠ r ∧ s = r ∧ ib ∈ PS THEN
        r := r+1
    END;
  chkc = WHEN t ≠ r THEN
        ob :| (ob ∈ seq(C) ∧ ∃ d.(d:CD ∧ ob  ⊆ d))
    END;
  release = WHEN t ≠ r ∧ ib ∈ CD ∧ ob = ib THEN
        t := t+1
    END;
  writec = WHEN s ≠ t ∧ cw = co ∧ o < size(ob)THEN
        oc,o,cw := ob(o+1),o+1,cw+1
    END;
  output = WHEN s ≠ t ∧ o = size(ob) THEN
        s := s+1 ‖ o := 0
    END;
  reset = BEGIN
        o,ib,ob,p,s,t,cr,co,scan := 0,<>,<>,r,r,r,ci,cw,T
    END;
/*external*/
  sendc = WHEN cr = ci THEN
         ic := cin(ci+1) ‖ ci := ci+1
    END;
  recvc = WHEN cw ≠ co THEN
         co := co +1
    END
END
```

Auxiliary Machines Stating Properties and Theorems

The following models contain theorems about sequences and proper strings that were usefull in proving the Filter development.

SeqSimp

MODEL
SeqSimp
SEES
Char
CONSTANTS
squash
PROPERTIES

```
squash ∈ (ℕ ⇸ C) ⇸ seq(C) ∧
dom(squash) = {f | f ∈ ℕ ⇸ C ∧ dom(f) ∈ 𝔽(ℕ)} ∧
squash(∅) = [] ∧
∀ f.(f ∈ ℕ ⇸ C ∧ dom(f) ∈ 𝔽₁(ℕ) ⇒
    squash(f) = squash({max(dom(f))} <⩤ f)  ← f(max(dom(f))))
```

THEOREMS

```
∀ (x).(x:C ⇒ {1|->x} = [x]) ∧

∀ (s,c,i).(s:seq(C) ∧ c:C ∧ i:dom(s)⇒
     (s|>{c}={i|->c} ⇒ s(i)=c)) ∧

∀ (s,c,i).(s∈seq(C) ∧ c:C ∧ i : 0 .. size(s) ⇒
      s<-c\|/i = s\|/i<-c) ∧

∀ (s,i).(s∈seq(C) ∧ i : 0 .. size(s)-1 ⇒
      s/|\i<-s(i+1) = s/|\i+1) ∧

∀ (s,t).(s:seq(C)∧t:seq(C)∧s<:t ⇒ s=t/|\size(s)) ∧

∀ (s,i,j).(s∈seq(C) ∧ i ∈ 0 .. size(s) ∧ j : 0 .. size(s) ∧ i  ≤ j ⇒
          (s\|/i)\|/j-i = s\|/j
         ) ∧

∀ (sx,i,j).(sx:seq(C) ∧ i:0 .. size(sx) ∧ j:0 .. size(sx) ∧ i  ≤ j ⇒
          rev(sx\|/j)  ⊆ rev(sx\|/i)) ∧

∀ (sx,sy,c).(sx∈seq(C) ∧ sy:seq(C) ∧ c : C ∧ rev(sx)<:rev(sy) ⇒
       rev(sx<-c)  ⊆ rev(sy<-c)) ∧

∀ (sx,sy).(sx:seq(C) ∧ sy:seq(C) ∧ sx  ⊆ sy ⇒ size(sx)  ≤ size(sy))
```

END

PStringSimp1

MODEL
PStringSimp1
SEES PString, Char, SeqSimp

THEOREMS
∀ (sx,sy).(sx:PS ∧ sy:PS ∧ sx ⊆ sy ⇒ sx = sy) ∧
∀ (sx,sy).(sx:PS ∧ sy:PS ∧ rev(sx) ⊆ rev(sy) ⇒ sx = sy) ∧
∀ (ps,cs).(ps∈PS ∧ cs : seq_1(C) ∧ cs ⊆ ps ⇒ cs(1) = bwc)
END

PStringSimp2

MODEL
PStringSimp2
SEES PStringSimp1, PString, SeqSimp, Char
THEOREMS

∀ (sx,i,j).(sx:seq(C) ∧ i:0 .. size(sx) ∧ j:0 .. size(sx) ⇒
(∃ (ps,pt).(ps:PS ∧ pt:PS ∧ sx\|/i ⊆ ps ∧ sx\|/j ⊆ pt
∧ sx\|/i ≠ <> ∧ sx\|/j ≠ <>)
⇒ i = j)) ∧
∀ (ps,cs).(ps∈PS ∧ cs : seq(C) ∧ cs<-bwc ⊆ ps ⇒ cs = <>)

END

PStringSimp3

MODEL
PStringSimp3
SEES PStringSimp2, PStringSimp1, PString, Char
THEOREMS
∀ (sx,i,j).(sx:seq(C) ∧ i:0 .. size(sx) ∧ j:0 .. size(sx) ∧
sx\|/i ∈ PS ∧ sx\|/j ∈ PS ⇒ i = j)

END

Appendix B

Comparator

This chapter contains the refinement of the electronic part of the filter. It begins with a simplifed version of the specification where the events to do with character I/O are abstracted, and anticipating events for the electronics are put in place. This refinement is to do with implementing the comparison of the dictionary words with the input word and the loading of the resultant output word.

Compare

MODEL
Compare
SEES
CDict, Char, PString, SeqSimp
VARIABLES
/* internal */
ib,ob,r,t,s /* external */
INVARIANT
ib ∈ seq(C) ∧
ob ∈ seq(C) ∧
∃ cd.(cd ∈ CD ∧ ob ⊆ cd) ∧
r ∈ ℕ ∧
t ∈ r-1 .. r ∧
s ∈ r-1 .. r ∧
s ∈ t-1 .. t ∧
(r ≠ t ⇒ ib ∈ PS) ∧
(s ≠ t ⇒ ob ∈ CD)
THEOREMS
(s=r ⇒ t=r) ∧ (s\net⇒r=t)
INITIALISATION
r,t,s := 0,0,0 ‖ ib,ob := ⟨⟩,⟨⟩

EVENTS
```
/* internal */
  nextiw = skip;
  nextiwc = skip;
  initd = skip;
  nextdc = skip;
  decode = skip;
  nextd = WHEN t ≠ r THEN
        ob := ⟨⟩
    END;
  lastd = WHEN t ≠ r THEN
        ob := ⟨⟩
    END;
  chkc = WHEN t ≠ r THEN
        ob :| (ob ∈ seq(C) ∧ ∃ d.(d ∈ CD ∧ ob  ⊆ d))
    END;
  release = WHEN t ≠ r ∧ ib ∈ CD ∧ ob = ib THEN
        t := t+1
    END;
  nextow = WHEN t\ne r THEN
        ob:= ⟨⟩
    END;
  nextowc= WHEN t\ne r THEN
        ob∈|(ob ∈ seq(C) ∧ ∃ cd.(cd : CD ∧ ob  ⊆ cd))
    END;
/*external*/
  input = WHEN s = r ∧ ib ∈ PS THEN
        r := r+1
    END;
  output = WHEN s ≠ t  THEN
        s := s+1
    END;
  reset = BEGIN
        ib,ob,s,t := ⟨⟩,⟨⟩,r,r
    END
END
```

CompareR

In the first refinement we introduce a trusted word which is a word chosen from the dicitonary. In addition there are some signals introduced to indicate that a dictionary word has been chosen.

REFINEMENT

```
   CompareR
REFINES
   Compare
SEES
   CDict, Char, PString, SeqSimp
VARIABLES
   tw,dc,dt,            /* internal */
   ib,ob,r,t,s          /* external */
INVARIANT
   tw ∈ CD ∧
   dc ∈ ℕ ∧
   dt ∈ ℕ ∧
   dt ∈ dc  ..  dc+1
THEOREMS
  (s=r ⇒ t=r) ∧ (s\net⇒r=t)
INITIALISATION
  r,t,s := 0,0,0 ‖ ib,ob := ⟨⟩,⟨⟩ ‖ tw:: CD ‖ dc,dt:=0,0
EVENTS
/* internal */
  nextiw = skip;
  nextiwc = skip;
  initd = skip;
  nextdc = skip;
  decode = skip;
  nextd = WHEN t ≠ r ∧ dt = dc THEN
        tw:: CD ‖ dt:= dt+1 ‖ ob := ⟨⟩
    END;
  lastd = WHEN t ≠ r ∧ dt = dc  THEN
        tw := [edc] ‖ dt:= dt+1 ‖ ob:= ⟨⟩
    END;
  chkc = WHEN t ≠ r ∧ dt ≠ dc THEN
        ob :| (ob ∈ seq(C) ∧ ob  ⊆ tw)‖
        IF ib ≠ tw THEN dc :∈ dc  ..  dc+1 END
    END;
  release = WHEN t ≠ r ∧ ib = tw ∧ ob = ib ∧ dt ≠ dc THEN
        t := t+1 ‖ dc:= dc+1
    END;
  nextow = WHEN t\ne r THEN
        ob:= ⟨⟩
    END;
  nextowc= WHEN t\ne r THEN
        ob∈|(ob ∈ seq(C) ∧ ∃ cd.(cd : CD ∧ ob  ⊆ cd))
    END;
/*external*/
  input = WHEN s = r ∧ ib ∈ PS THEN
        r := r+1
```

```
    END;
  output = WHEN s ≠ t  THEN
        s := s+1
    END;
  reset = BEGIN
        ib,ob,s,t,dc := ⟨⟩,⟨⟩,r,r,dt
    END
END
```

CompareRR

This refinment is about showing that the strategy of choosing trusted words one-by-one from the dictionary will terminate when there are no more dictionary words to chose. If it reaches the end of the dictionary then the `nextd` event will be disabled and the filter will shutdown.

```
REFINEMENT
   CompareRR
REFINES
   CompareR
SEES
   CDict, Char, PString, SeqSimp
VARIABLES
   tw,dc,dt,V,              /* internal */
   ib,ob,r,t,s,u               /* external */
INVARIANT
   V  ⊆ CD ∧
   u ∈ ℕ ∧
   u ∈ t  ..  t+1
THEOREMS
  (s=r ⇒ t=r) ∧ (s\net⇒r=t)
INITIALISATION
  r,t,s,u := 0,0,0,0 ‖ ib,ob := ⟨⟩,⟨⟩ ‖ tw:: CD ‖
  V := ∅ ‖ dc,dt:=0,0
EVENTS
/* internal */
  nextiw = skip;
  nextiwc = skip;
  nextdc = skip;
  decode = skip;
  initd = WHEN u = t THEN
        V,u:= ∅,u+1
    END;
  nextd = WHEN u ≠ t ∧ t ≠ r ∧ dt = dc ∧ V ≠ CD THEN
```

```
        tw:: CD-V ‖ dt:= dt+1 ‖ V:=V∪{tw} ‖ ob:= ⟨⟩
    END;
  lastd = WHEN u ≠ t ∧ t ≠ r ∧ dt = dc ∧ V = CD  THEN
        tw := [edc] ‖ dt:= dt+1 ‖ ob := ⟨⟩
    END;
  chkc = WHEN t ≠ r ∧ dt ≠ dc THEN
        ob :| (ob ∈ seq(C) ∧ ob  ⊆ tw)‖
        IF ib ≠ tw THEN dc :∈ dc  ..  dc+1 END
    END;
  release = WHEN u ≠ t ∧ t
  ≠ r ∧ ib = tw ∧ ob = ib ∧ dt ≠ dc THEN
        t := t+1 ‖ dc := dc+1
    END;
  nextow = WHEN t ≠ r THEN
        ob:= ⟨⟩
    END;
  nextowc= WHEN t ≠ r THEN
        ob∈|(ob : seq(C) ∧ ob  ⊆ tw)
    END;
/*external*/
  input = WHEN s = r ∧ ib ∈ PS THEN
        r := r+1
    END;
  output = WHEN s ≠ t  THEN
        s := s+1
    END;
  reset = BEGIN
        ib,ob,s,t,u,dc,V := ⟨⟩,⟨⟩,r,r,r,dt,∅
    END
END
```

CompareRRR

Now we refine the events for checking the trusted word character by character. This involves choosing the next trusted character, loading it into the output buffer and comparing the output buffer to the input buffer.

REFINEMENT
   CompareRRR
REFINES
   CompareRR
SEES
   CDict, Char, PString, SeqSimp, PStringSimp1
VARIABLES

```
   tw,dc,dt,V,twc,u,                    /* internal */
   twi,dtc,dcc,ot,otc,
   ib,ob,r,t,s                          /* external */
DEFINITIONS
   boo(E) ≙ (E = T)
INVARIANT
   twc ∈ C ∧
   twi ∈ 0 .. size(tw) ∧
   dtc ∈ ℕ ∧
   dcc ∈ ℕ ∧
   dtc ∈ dcc  ..  dcc+1 ∧
   ot ∈ ℕ ∧
   ot ∈ dt  ..  dt+1 ∧
   ot ∈ dc  ..  dc+1 ∧
   otc ∈ ℕ ∧
   dtc ∈ otc  ..  otc+1 ∧
   otc ∈ dcc  ..  dcc+1 ∧
   ob  ⊆ tw ∧
   (dt ≠ ot ⇒ ob = ⟨⟩) ∧
   (otc = dtc ∧ ot = dt ⇒ ob = tw↑twi) ∧
   (otc ≠ dtc ∧ ot ≠ dc ⇒ ob = tw↑twi-1) ∧

   (t ≠ r ∧ dtc ≠ dcc ⇒ 1 ≤ twi ∧ twc = tw(twi)) ∧
   (t = r ⇒ dtc = dcc) ∧
   (dt = dc ⇒ dtc = dcc) ∧
   (dt ≠ dc ⇒ u ≠ t) ∧
   (dcc = dtc ∧ dt ≠ dc ⇒ twi < size(tw))
THEOREMS
  (ot = dc ⇒ ot = dt) ∧
  (dtc = dcc ⇒ otc = dcc) ∧
  (otc ≠ dcc ⇒ otc = dtc) ∧
  (otc ≠ dtc ⇒ otc = dcc) ∧
  tw ∈ seq(C)
INITIALISATION
  r,t,s,u := 0,0,0,0 ‖ ib,ob := ⟨⟩,⟨⟩ ‖ tw:: CD ‖
  V := ∅ ‖ dc,dt,ot:=0,0,0 ‖
  twi,dtc := 0,0 ‖ twc :∈ C ‖
  dcc,otc := 0,0
EVENTS
/* internal */
  nextiw = skip;
  nextiwc = skip;
  decode = skip;
  initd = WHEN u = t THEN
        V,u := ∅, u+1
    END;
```

```
  nextd = WHEN u ≠ t ∧ t
  ≠ r ∧ dt = dc ∧ V ≠ CD ∧ dt ≠ ot THEN
        tw:: CD-V ‖ dt:= dt+1 ‖ V:=V∪{tw} ‖ twi:= 0
    END;
  lastd = WHEN u ≠ t ∧ t
  ≠ r ∧ dt = dc ∧ V = CD ∧ dt ≠ ot THEN
        tw := [edc] ‖ dt:= dt+1 ‖ twi:= 0
    END;
  nextdc = WHEN t\ner ∧ dt\nedc ∧ dtc = dcc ∧ dtc =otc
    THEN
        twc := tw(twi+1) ‖ twi,dtc := twi+1, dtc+1
    END;
  chkc = WHEN t ≠ r ∧ dt ≠ dc ∧ dtc ≠ dcc ∧
              otc ≠ dcc ∧ ot ≠ dc
    THEN
      IF ob  ⊆ ib ∧ twi < size(tw) THEN
        dcc:= dcc+1
      ELSIF ob / ⊆ ib THEN
        dc,dcc := dc+1,dcc+1
      END
    END;
  release = WHEN t ≠ r ∧ ib = tw ∧ ob = ib ∧ twi = size(tw) ∧
                dtc ≠ dcc ∧ otc ≠ dcc ∧ dt ≠ dc THEN
        t := t+1 ‖ dc := dc+1 ‖ dcc := dcc+1
    END;
  nextow = WHEN t\ne r ∧ ot = dc THEN
        ob:= ⟨⟩ ‖ ot := ot+1
    END;
  nextowc= WHEN t ≠ r ∧ otc ≠ dtc THEN
        ob:= tw↑twi ‖ otc := otc+1
    END;
/*external*/
  input = WHEN s = r ∧ ib ∈ PS THEN
        r := r+1
    END;
  output = WHEN s ≠ t  THEN
        s := s+1
    END;
  reset = BEGIN
        ib,ob,s,t,u := ⟨⟩,⟨⟩,r,r,r ‖
        dc,ot := dt,dt ‖
        dcc,otc := dtc,dtc ‖ twi := 0 ‖ V:= ∅
    END
END
```

CompareRRRR

We now consider extracting the characters fmom the input buffer one-by one. The character from the input is then compared to the character from the dictonary to decide if the substrings are a match so far.

REFINEMENT
CompareRRRR
REFINES
CompareRRR
SEES
CDict, Char, PString, SeqSimp, PStringSimp1
VARIABLES
tw,dc,dt,V,u,twc,iwc, /* internal */
it,iwi,itc,twi,dtc,dcc,ot,otc,
ib,ob,r,t,s /* external */
INVARIANT
$it \in \mathbb{N} \wedge$
$iwc \in C \wedge$
$iwi \in 0 .. size(ib) \wedge$
$(it = dt \Rightarrow twi \in iwi .. iwi+1) \wedge$
$itc \in \mathbb{N} \wedge$
$dt \in it .. it+1 \wedge$
$it \in dc .. dc+1 \wedge$
$itc \in dcc .. dcc+1 \wedge$
$dtc \in itc .. itc+1 \wedge$

$(t \neq r \wedge itc \neq dcc \Rightarrow 1 \leq iwi \wedge iwc = ib(iwi)) \wedge$

$(it \neq dc \wedge itc \neq dtc \Rightarrow iwi = twi-1) \wedge$
$(it \neq dc \wedge itc = dtc \Rightarrow iwi = twi) \wedge$

$(t \neq r \wedge otc \neq dcc \Rightarrow twc = ob(twi)) \wedge$

$(itc \neq dcc \wedge it \neq dc \Rightarrow ob{\uparrow}iwi-1 = ib{\uparrow}iwi-1) \wedge$
$(itc = dcc \wedge it \neq dc \Rightarrow ob{\uparrow}iwi = ib{\uparrow}iwi) \wedge$

$(it \neq dt \Rightarrow ob = \langle\rangle \wedge twi = 0) \wedge$
$(it \neq dt \Rightarrow itc = dcc) \wedge$
$(dt = dc \Rightarrow itc = dtc) \wedge$
$(ot = dc \Rightarrow otc = dtc) \wedge$
$(otc \neq dtc \Rightarrow ot \neq dc) \wedge$
$(dt \neq dc \Rightarrow ot \neq dc)$
THEOREMS
$(dt = dc \Rightarrow it = dc) \wedge (it \neq dt \Rightarrow dt \neq dc) \wedge$

```
   (dtc = dcc ⇒ itc = dcc) ∧ (itc ≠ dtc ⇒ dtc ≠ dcc)
INITIALISATION
  r,t,s,u := 0,0,0,0 ‖ ib,ob := ⟨⟩,⟨⟩ ‖ tw:: CD ‖
  V := ∅ ‖ it,dc,dt,ot:=0,0,0,0 ‖
  iwi,itc := 0,0 ‖ iwc :∈ C ‖
  twi,dtc := 0,0 ‖ twc :∈ C ‖
  dcc,otc := 0,0
EVENTS
/* internal */
  decode = skip;
  nextiw = WHEN it ≠ dt ∧ itc = dtc THEN
       it := it+1 ‖
       iwi := 0
    END;
  nextiwc = WHEN it = dt ∧ itc ≠ dtc ∧ iwi<size(ib) THEN
       iwc := ib(iwi+1)‖ iwi,itc := iwi+1,itc+1
    END;
  initd = WHEN u = t THEN
        V,u:= ∅,u+1
    END;
  nextd = WHEN u ≠ t ∧ t
  ≠ r ∧ dt = dc ∧ V ≠ CD ∧ dt ≠ ot THEN
        tw:: CD-V ‖ dt:= dt+1 ‖ V:=V∪{tw} ‖ twi:= 0
    END;
  lastd = WHEN u ≠ t ∧ t
  ≠ r ∧ dt = dc ∧ V = CD ∧ dt ≠ ot THEN
        tw := [edc] ‖ dt:= dt+1 ‖ twi:= 0
    END;
  nextdc = WHEN t\ner ∧ dt\nedc ∧ it = dt ∧ dtc = dcc ∧
                dtc = otc ∧ dtc = itc
    THEN
        twc := tw(twi+1) ‖ twi,dtc := twi+1, dtc+1
    END;
  chkc = WHEN t ≠ r ∧ dt ≠ dc ∧ dtc ≠ dcc ∧ it≠dc ∧
              itc ≠ dcc ∧ otc ≠ dcc ∧ ot ≠ dc
    THEN
      IF twc = iwc ∧ twi < size(tw) THEN
        dcc:= dcc+1
      ELSIF twc ≠ iwc THEN
        dc,dcc := dc+1, dcc+1
      END
    END;
  release = WHEN t ≠ r ∧ ib = tw ∧ ob = ib ∧ twi = size(tw) ∧
                itc ≠ dcc ∧ otc ≠ dcc ∧ dtc ≠ dcc ∧
                dt ≠ dc ∧ ot ≠ dc ∧ it ≠ dc THEN
        t := t+1 ‖ dc := dc+1 ‖ dcc := dcc+1
```

```
    END;
  nextow = WHEN t\ne r ∧ ot = dc THEN
        ob:= ⟨⟩ ‖ ot := ot+1
    END;
  nextowc= WHEN t ≠ r ∧ otc ≠ dtc THEN
        ob:= ob←twc ‖ otc := otc+1
    END;
/*external*/
  input = WHEN s = r ∧ ib ∈ PS THEN
        r := r+1
    END;
  output = WHEN s ≠ t  THEN
        s := s+1
    END;
  reset = BEGIN
        ib,ob,s,t,u := ⟨⟩,⟨⟩,r,r,r ‖
        dc,ot,it := dt,dt,dt ‖
        itc, dcc,otc := dtc,dtc,dtc ‖
        twi,iwi:= 0,0 ‖ V:= ∅
    END
END
```

CompareRRRRR

Finally we introduce logical functions to implement the comparison logic in the comparator. This concludes the refinement of the HLD.

```
REFINEMENT
   CompareRRRRR
REFINES
   CompareRRRR
SEES
   CDict, Char, PString, SeqSimp, PStringSimp1
VARIABLES
   tw,dc,dt,V,u,dec,twc,iwc,ED,
   it,iwi,itc,twi,dtc,dcc,ot,otc,
   ib,ob,r,t,s
DEFINITIONS
   boo(E) ≙ (E = T);
   InputReady ≙ (t ≠ r);
   NEXTCMD1 ≙ (dt = dc);
   NEXTCMD2 ≙ (ot = dc);
   NEXTCMD3 ≙ (it = dc);
   NEXTCHAR ≙ (dtc = dcc);
```

```
   MATCH ≙ (twc = iwc);
   OPICReady ≙ (s = r);
   DICTReady ≙ (dec ≠ dcc);
   DICTAvail ≙ (otc ≠ dtc);
   RELEASE ≙ (MATCH ∧ DICTReady ∧ boo(ED))
INVARIANT
  u ∈ r  ..  r+1 ∧
  ED ∈ BOOL ∧
  dec ∈ dcc  ..  dcc+1 ∧
  itc ∈ dec  ..  dec+1 ∧
  dtc ∈ dec  ..  dec+1 ∧
  (dt = dc ⇒ dec = dcc) ∧
  (DICTReady ⇒ itc ≠ dcc ∧ dtc ≠ dcc ∧ otc ≠ dcc) ∧
  (DICTReady ⇒ it ≠ dc ∧ ot ≠ dc) ∧
  (DICTReady ⇒ InputReady) ∧
  (dt ≠ ot ⇒ InputReady) ∧
  (dt ≠ dc ⇒ InputReady) ∧
  (DICTAvail ⇒ InputReady) ∧
  (u ≠ r ⇒ ED = F) ∧
  (it ≠ dt ⇒ ED = F) ∧
  (dt ≠ dc ∧ dec = dcc ⇒ ED = F) ∧
  (itc ≠ dtc ⇒ ED = F) ∧
  (itc ≠ dec ⇒ ED = F) ∧
  (dec ≠ dcc ∧ ED = F ⇒ twc ≠ ewc) ∧
  (dec ≠ dcc ∧ ED = F ∧ twc ≠ edc ⇒ twi < size(tw)) ∧
  (r ≠ t ∧ dec = dtc ∧ boo(ED) ⇒ twi = size(tw) ∧ twc = ewc) ∧
  (r ≠ t ∧ dec = itc ∧ boo(ED) ∧ MATCH ⇒ iwi=size(ib)) ∧
  (RELEASE ⇒  ¬(NEXTCMD1))
INITIALISATION
  r,t,s,u := 0,0,0,0 ‖ ib,ob := ⟨⟩,⟨⟩ ‖ tw:: CD ‖
  V := ∅ ‖ it,dc,dt,dec,ot:=0,0,0,0,0 ‖ ED := F ‖
  iwi,itc := 0,0 ‖ iwc :∈ C ‖
  twi,dtc := 0,0 ‖ twc :∈ C ‖
  dcc,otc := 0,0
EVENTS
/* internal */
  nextiw = WHEN it ≠ dt THEN
       it,iwi := it+1,0
    END;
  nextiwc = WHEN it = dt ∧ itc ≠ dtc ∧ iwi<size(ib) THEN
       iwc := ib(iwi+1)‖ iwi,itc := iwi+1,itc+1
    END;
  initd = WHEN u = t ∧ u = r THEN
        ED, V := F, ∅ ‖
        u:= u+1
    END;
```

```
  nextd = WHEN u ≠ t ∧ NEXTCMD1 ∧ V ≠ CD ∧ dt≠ot THEN
        tw:: CD-V ‖ dt:= dt+1 ‖ V:=V∪{tw} ‖ ED := F ‖
        twi:= 0
    END;
  lastd = WHEN u ≠ t ∧ NEXTCMD1 ∧ V = CD ∧ dt≠ot THEN
        tw := [edc] ‖ dt:= dt+1 ‖ ED := F ‖ twi:= 0
    END;
  nextdc = WHEN dt\nedc ∧ it = dt ∧ NEXTCHAR ∧
                ¬ boo(ED) ∧ dtc = otc ∧ dtc = dec
    THEN
        twc := tw(twi+1) ‖ twi,dtc := twi+1, dtc+1
    END;
  decode = WHEN itc ≠ dec ∧ otc ≠ dec THEN
        ED :| ((boo(ED))  ⇔ (twc=ewc))‖
        dec:= dec+1
    END;
  chkc = WHEN DICTReady ∧ twc ≠ edc THEN
      IF MATCH ∧ ¬ boo(ED) THEN
        dcc:= dcc+1
      ELSIF ¬(MATCH) THEN
        dc,dcc := dc+1, dcc+1
      END
    END;
  release = WHEN RELEASE THEN
        t := t+1 ‖ dc := dc+1 ‖ dcc := dcc+1
    END;

  nextow = WHEN InputReady ∧ NEXTCMD2 THEN
        ot := ot+1 ‖ ob := ⟨⟩
    END;
  nextowc = WHEN DICTAvail THEN
        ob := ob←twc ‖ otc := otc+1
  END;

/*external*/
  input = WHEN OPICReady ∧ u ≠ r ∧ ib ∈ PS THEN
        r := r+1
    END;
  output = WHEN s ≠ t THEN
        s := s+1
    END;
  reset = BEGIN
        ib,ob,s,t,u,ED := ⟨⟩,⟨⟩,r,r,r,F ‖
        dc,it,ot := dt,dt,dt ‖
        dec,itc,dcc,otc := dtc,dtc,dtc,dtc ‖
        twi,iwi:= 0,0 ‖ V:= ∅
```

```
        END
END
```

Appendix C

High-level Design

In this chapter we reorganise the models into components corresponding to the high-level design.

- CompCOMP corresponds to the comparison logic.
- DictComp corresponds to the dictionary logic.
- InputComp corresponds to the Input PIC.
- OutputComp corresponds to the Output PIC.

Each component has internal events correponding to the sub-machine that it implements and external events corresponding to the events that it expects the rest of the components to implement.

CompCOMP

MODEL
CompCOMP
SEES
CDict, Char, PString, SeqSimp
VARIABLES
dc,dt,dec,twc,iwc,ED, /* all external*/
it,itc,dtc,dcc,ot,otc,
r,t,u
DEFINITIONS
boo(E) $\triangleq$ (E $=$ T);
InputReady $\triangleq$ (t $\neq$ r);
NEXTCMD1 $\triangleq$ (dt $=$ dc);
NEXTCMD2 $\triangleq$ (ot $=$ dc);
NEXTCMD3 $\triangleq$ (it $=$ dc);

```
   NEXTCHAR ≜ (dtc = dcc);
   MATCH ≜ (twc = iwc);
   DICTReady ≜ (dec ≠ dcc);
   DICTAvail ≜ (otc ≠ dtc);
   RELEASE ≜ (MATCH ∧ DICTReady ∧ boo(ED))
INVARIANT
  r ∈ ℕ ∧
  t ∈ ℕ ∧
  r ∈ t  ..  t+1 ∧
  u ∈ r  ..  r+1 ∧
  u ∈ t  ..  t+1 ∧
  iwc ∈ C ∧
  twc ∈ C ∧
  dc ∈ ℕ ∧
  dt ∈ ℕ ∧
  dt ∈ dc  ..  dc+1 ∧
  dtc ∈ ℕ ∧
  dcc ∈ ℕ ∧
  dtc ∈ dcc  ..  dcc+1 ∧
  ot ∈ ℕ ∧
  ot ∈ dt  ..  dt+1 ∧
  ot ∈ dc  ..  dc+1 ∧
  otc ∈ ℕ ∧
  dtc ∈ otc  ..  otc+1 ∧
  otc ∈ dcc  ..  dcc+1 ∧
  it ∈ ℕ ∧
  itc ∈ ℕ ∧
  dt ∈ it  ..  it+1 ∧
  it ∈ dc  ..  dc+1 ∧
  itc ∈ dcc  ..  dcc+1 ∧
  dtc ∈ itc  ..  itc+1 ∧
  (t = r ⇒ dtc = dcc) ∧
  (dt = dc ⇒ dtc = dcc) ∧
  ED ∈ BOOL ∧
  dec ∈ dcc  ..  dcc+1 ∧
  itc ∈ dec  ..  dec+1 ∧
  dtc ∈ dec  ..  dec+1 ∧
  (r\net ∧ dec = dtc ∧ boo(ED) ⇒ twc = ewc) ∧
  (u ≠ r ⇒ ED = F) ∧
  (itc ≠ dtc ⇒ ED = F) ∧
  (itc ≠ dec ⇒ ED = F) ∧
  (it ≠ dt ⇒ ED = F) ∧
  (it ≠ dt ⇒ itc = dcc) ∧
  (dt = dc ⇒ itc = dtc) ∧
  (ot = dc ⇒ otc = dtc) ∧
  (otc ≠ dtc ⇒ ot ≠ dc) ∧
```

```
  (dt ≠ dc ⇒ ot ≠ dc) ∧
  (DICTReady ⇒ itc ≠ dcc ∧ dtc ≠ dcc ∧ otc ≠ dcc) ∧
  (DICTReady ⇒ it ≠ dc ∧ ot ≠ dc) ∧
  (DICTReady ⇒ InputReady) ∧
  (dt ≠ ot ⇒ InputReady) ∧
  (dt ≠ dc ⇒ InputReady) ∧
  (DICTAvail ⇒ InputReady) ∧
  (u ≠ r ⇒ ED = F) ∧
  (it ≠ dt ⇒ ED = F) ∧
  (dt ≠ dc ∧ dec = dcc ⇒ ED = F) ∧
  (itc ≠ dtc ⇒ ED = F) ∧
  (itc ≠ dec ⇒ ED = F) ∧
  (dec ≠ dcc ∧ ED = F ⇒ twc ≠ ewc)
THEOREMS
  (ot = dc ⇒ ot = dt) ∧
  (dtc = dcc ⇒ otc = dcc) ∧
  (otc ≠ dcc ⇒ otc = dtc) ∧
  (otc ≠ dtc ⇒ otc = dcc) ∧
  (dt = dc ⇒ it = dc) ∧ (it ≠ dt ⇒ dt ≠ dc) ∧
  (dtc = dcc ⇒ itc = dcc) ∧ (itc ≠ dtc ⇒ dtc ≠ dcc)
INITIALISATION
  r,t,u := 0,0,0 ‖
  it,dc,dt,dec,ot:=0,0,0,0,0 ‖ ED := F ‖
  itc := 0 ‖ iwc :∈ C ‖
  dtc := 0 ‖ twc :∈ C ‖
  dcc,otc := 0,0
EVENTS
/* internal */
  chkc = WHEN DICTReady THEN
      IF MATCH ∧ ¬ boo(ED) THEN
        dcc:= dcc+1
      ELSIF ¬ (MATCH) THEN
        dc,dcc := dc+1, dcc+1
      END
    END;
  release = WHEN RELEASE THEN
        t := t+1 ‖ dc,dcc := dc+1, dcc+1
    END;
/*external*/
  input = WHEN t = r ∧ u ≠ r THEN
        r := r+1
    END;
  nextiw = WHEN it ≠ dt THEN
       it := it+1
    END;
  nextiwc = WHEN it = dt ∧ itc ≠ dtc THEN
```

```
      iwc :∈ C ‖ itc := itc+1
    END;
  initd = WHEN  u = t ∧ u = r THEN
      ED,u := F,u+1
    END;
  nextd = WHEN  NEXTCMD1 ∧ ¬ boo(ED) ∧ dt\neot THEN
      dt:= dt+1 ‖ ED := F
    END;
  nextdc = WHEN  dt\nedc ∧ it = dt ∧ NEXTCHAR ∧ ¬ boo(ED) ∧
              dtc =otc ∧ dtc = dec
    THEN
      twc :∈ C ‖ dtc := dtc+1
    END;
  decode = WHEN itc ≠ dec ∧ otc ≠ dec  THEN
      ED :| ((boo(ED))  ⇔ (twc=ewc))‖
      dec:= dec+1
    END;
  nextow = WHEN InputReady ∧ NEXTCMD2 THEN
      ot := ot+1
    END;
  nextowc = WHEN DICTAvail THEN
      otc := otc+1
  END;
  reset = BEGIN
      t,ED := r,F ‖
      dc,it,ot := dt,dt,dt ‖
      dec,itc,dcc,otc := dtc,dtc,dtc,dtc
    END
END
```

DictCOMP

```
MODEL
   DictCOMP
SEES
   CDict, Char, PString, SeqSimp
VARIABLES
   tw,dc,dt,V,DE,dec,twc,  /*internal*/

   ED,it,itc,twi,dtc,dcc,ot,otc, /*external*/
   r,t,s,u
DEFINITIONS
   boo(E) ≙ (E = T);
```

```
   InputReady ≜ (t ≠ r);
   NEXTCMD1 ≜ (dt = dc);
   NEXTCMD2 ≜ (ot = dc);
   NEXTCMD3 ≜ (it = dc);
   NEXTCHAR ≜ (dtc = dcc);
   OPICReady ≜ (s = r);
   DICTReady ≜ (dec ≠ dcc);
   DICTAvail ≜ (otc ≠ dtc);
   RELEASE ≜ (DICTReady ∧ boo(ED))
```

INVARIANT

```
   r ∈ ℕ ∧
   s ∈ ℕ ∧
   t ∈ ℕ ∧
   r ∈ t  ..  t+1 ∧
   r ∈ s  ..  s+1 ∧
   t ∈ s  ..  s+1 ∧
   u ∈ r  ..  r+1 ∧
   u ∈ t  ..  t+1 ∧
   tw ∈ CD ∧
   dc ∈ ℕ ∧
   dt ∈ ℕ ∧
   dt ∈ dc  ..  dc+1 ∧
   V  ⊆ CD ∧
   twc ∈ C ∧
   twi ∈ 0 .. size(tw) ∧
   dtc ∈ ℕ ∧
   dcc ∈ ℕ ∧
   dtc ∈ dcc  ..  dcc+1 ∧
   ot ∈ ℕ ∧
   ot ∈ dt  ..  dt+1 ∧
   ot ∈ dc  ..  dc+1 ∧
   otc ∈ ℕ ∧
   dtc ∈ otc  ..  otc+1 ∧
   otc ∈ dcc  ..  dcc+1 ∧
   it ∈ ℕ ∧
   itc ∈ ℕ ∧
   dt ∈ it  ..  it+1 ∧
   it ∈ dc  ..  dc+1 ∧
   itc ∈ dcc  ..  dcc+1 ∧
   dtc ∈ itc  ..  itc+1 ∧
   (t ≠ r ∧ dtc ≠ dcc ⇒ 1 ≤ twi ∧ twc = tw(twi)) ∧
   (t = r ⇒ dtc = dcc) ∧
   (dt = dc ⇒ dtc = dcc) ∧
   DE ∈ BOOL ∧
  (boo(DE) ⇒ V = CD) ∧
  (¬ boo(DE) ∧ dt = dc  ⇒ V ≠ CD) ∧
```

```
   ED ∈ BOOL ∧
  dec ∈ dcc  ..  dcc+1 ∧
  itc ∈ dec  ..  dec+1 ∧
  dtc ∈ dec  ..  dec+1 ∧
  (¬ boo(DE) ∧ dec ≠ dcc  ⇒ V ≠ CD) ∧
  (DICTReady ⇒ itc ≠ dcc ∧ dtc ≠ dcc ∧ otc ≠ dcc) ∧
  (DICTReady ⇒ it ≠ dc ∧ ot ≠ dc) ∧
  (DICTReady ⇒ InputReady) ∧
  (dt ≠ ot ⇒ InputReady) ∧
  (dt ≠ dc ⇒ InputReady) ∧
  (DICTAvail ⇒ InputReady) ∧
  (u ≠ r ⇒ ED = F) ∧
  (it ≠ dt ⇒ ED = F) ∧
  (itc ≠ dtc ⇒ ED = F) ∧
  (dt ≠ dc ∧ dec = dcc ⇒ ED = F) ∧
  (itc ≠ dec ⇒ ED = F) ∧
  (dec ≠ dcc ∧ ED = F ⇒ twc ≠ ewc) ∧
  (dec ≠ dcc ∧ ED = F ⇒ twi < size(tw)) ∧
  (r ≠ t ∧ dec = dtc ∧ boo(ED) ⇒ twi = size(tw) ∧ twc = ewc) ∧
   (it ≠ dt ⇒ itc = dcc) ∧
   (dt = dc ⇒ itc = dtc) ∧
   (ot = dc ⇒ otc = dtc) ∧
   (otc ≠ dtc ⇒ ot ≠ dc) ∧
   (dt ≠ dc ⇒ ot ≠ dc) ∧
   (dt ≠ dc ⇒ u ≠ t) ∧
  (RELEASE ⇒  ¬(NEXTCMD1))
```

THEOREMS

```
  (s=r ⇒ t=r) ∧ (s\net⇒r=t) ∧
  (ot = dc ⇒ ot = dt) ∧
  (dtc = dcc ⇒ otc = dcc) ∧
  (otc ≠ dcc ⇒ otc = dtc) ∧
  (otc ≠ dtc ⇒ otc = dcc) ∧
   (dt = dc ⇒ it = dc) ∧ (it ≠ dt ⇒ dt ≠ dc) ∧
   (dtc = dcc ⇒ itc = dcc) ∧ (itc ≠ dtc ⇒ dtc ≠ dcc)
```

INITIALISATION

```
  r,t,s,u := 0,0,0,0 ‖ tw:: CD ‖
  V := ∅ ‖ it,dc,dt,dec,ot:=0,0,0,0,0 ‖ DE,ED :=
  F,F ‖
  itc := 0 ‖
  twi,dtc := 0,0 ‖ twc :∈ C ‖
  dcc,otc := 0,0
```

EVENTS

```
/* internal */
  initd = WHEN u = t ∧ u = r THEN
        DE, ED, V := F, F, ∅ ‖
        u:= u+1
```

```
    END;
  nextd = WHEN  u ≠ t ∧ NEXTCMD1 ∧ ¬ boo(DE) ∧ dt≠ot THEN
        tw:: CD-V ‖ dt:= dt+1 ‖ V:=V∪{tw} ‖ ED := F ‖
        twi:= 0
    END;
  nextdc = WHEN dt\nedc ∧ it = dt ∧ NEXTCHAR ∧ dtc=dcc ∧
                twi<size(tw) ∧ dtc =otc ∧ dtc = dec
    THEN
        twc := tw(twi+1) ‖ twi,dtc := twi+1, dtc+1
    END;
  decode = WHEN itc ≠ dec ∧ otc ≠ dec  THEN
        DE :| ((boo(DE))  ⇔ (V=CD)) ‖
        ED :| ((boo(ED))  ⇔ (twc=ewc))‖
        dec:= dec+1
    END;
/*external components*/
/*input PIC*/
  input = WHEN  OPICReady ∧ u ≠ r  THEN
        r := r+1
    END;
  nextiw = WHEN it ≠ dt THEN
       it := it+1
    END;
  nextiwc = WHEN it = dt ∧ itc ≠ dtc THEN
       itc := itc+1
    END;
/*comparator*/
  chkc = WHEN DICTReady THEN
      IF ¬ boo(ED) THEN
         dcc:: dcc  ..  dcc+1
      ELSE CHOICE
         dc,dcc := dc,dcc OR
         dc,dcc := dc+1,dcc+1
      END END
    END;
  release = WHEN RELEASE THEN
        t := t+1 ‖  dc := dc+1 ‖ dcc := dcc+1
    END;
/*output PIC*/
  nextow = WHEN  InputReady ∧ NEXTCMD2 THEN
        ot := ot+1
    END;
  nextowc = WHEN DICTAvail THEN
        otc := otc+1
  END;
  output = WHEN s\net THEN
```

```
        s := s+1
    END;
/*reset*/
  reset = BEGIN
        s,t,u,DE,ED := r,r,r,F,F ||
        dc,it,ot := dt,dt,dt ||
        dec,itc,dcc,otc := dtc,dtc,dtc,dtc ||
        twi:= 0 || V:= ∅
    END
END
```

InputCOMP

```
MODEL
   InputCOMP
SEES
   Char, PString, SeqSimp
VARIABLES
   ib,iwi,scan,p,                /* internal */
   r,s,t,u,ic,ci,cr,             /* external */
   it,dt,itc,dtc,iwc
DEFINITIONS
   boo(E) ≙ (E = T);
   InputReady ≙ (t ≠ r);
   OPICReady ≙ (s = r)
INVARIANT
   ib ∈ seq(C) ∧
   ∃ ps.(ps ∈ PS ∧ ib  ⊆ ps) ∧
   r ∈ ℕ ∧
   t ∈ ℕ ∧
   p ∈ ℕ ∧
   u ∈ ℕ ∧
   s ∈ ℕ ∧
   r ∈ t  ..  t+1 ∧
   r ∈ s  ..  s+1 ∧
   t ∈ s  ..  s+1 ∧
   p ∈ r  ..  r+1 ∧
   p ∈ t  ..  t+1 ∧
   u ∈ t  ..  t+1 ∧
   u ∈ r  ..  r+1 ∧
   (r ≠ t ⇒ ib ∈ PS) ∧
   scan ∈ BOOL ∧
   ic ∈ C ∧
```

```
   ci ∈ ℕ ∧
   cr ∈ ci-1 .. ci ∧
   it ∈ ℕ ∧
   iwc ∈ C ∧
   iwi ∈ ℕ ∧
   (r ≠ t ⇒ iwi ∈ 0 .. size(ib)) ∧
   itc ∈ ℕ ∧
   dt ∈ ℕ ∧
   dt ∈ it  ..  it+1 ∧
   dtc ∈ ℕ ∧
   dtc ∈ itc  ..  itc+1 ∧
   (itc ≠ dtc ⇒ r ≠ t)
THEOREMS
  (s=r ⇒ t=r) ∧ (s\net⇒r=t)
INITIALISATION
  p,r,t,s,u := 0,0,0,0,0 ‖ ib := ⟨⟩ ‖ scan := T ‖
  ci,cr := 0,0 ‖ ic :∈ C ‖ it,dt:=0,0 ‖
  iwi,itc := 0,0 ‖ iwc :∈ C ‖
  dtc := 0
EVENTS
/* internal */
  readc = WHEN r = t ∧ cr ≠ ci ∧
      ((p=r ∧
      ic = bwc) ∨ ∃ ps.(ps ∈ PS ∧ ib←ic  ⊆ ps) ∨ scan = T )
    THEN
      IF  ∃ ps.(ps ∈ PS ∧ ib←ic  ⊆ ps) THEN
        ib,scan := ib←ic,F
      ELSIF ic = bwc THEN
        ib,scan := [bwc], F
      ELSIF scan = T THEN
        ib := ⟨⟩
      END ‖
      cr := cr+1 ‖
      IF p=r THEN p:= p+1 END
    END;
  input = WHEN p≠ r ∧ u ≠ r ∧ s = r ∧ ib ∈ PS THEN
        r := r+1 ‖ iwi:= 0
    END;
  nextiw = WHEN it ≠ dt THEN
       it,iwi := it+1,0
    END;
  nextiwc = WHEN it = dt ∧ itc ≠ dtc ∧ iwi<size(ib) THEN
       iwc := ib(iwi+1)‖ iwi,itc := iwi+1,itc+1
    END;
/*external*/
/*dictionary*/
```

```
  initd = WHEN u = t ∧ u = r THEN
        u:= u+1
    END;
  nextd = WHEN r\net ∧ it = dt THEN
        dt:= dt+1
    END;
  nextdc = WHEN r\net ∧ it = dt ∧ dtc = itc THEN
        dtc :=  dtc+1
    END;
/*comparitor*/
  release = WHEN t ≠ r ∧ itc = dtc THEN
        t := t+1
    END;
/*output PIC*/
  output = WHEN s ≠ t THEN
        s := s+1
    END;
/*reset*/
  reset = BEGIN
        ib,p,u,s,t,cr,scan := ⟨⟩,r,r,r,r,ci,T ||
        it := dt ||
        itc := dtc ||
        iwi:= 0
    END;
/*environment*/
  sendc = WHEN cr = ci THEN
         ic :∈ C || ci := ci+1
    END
END
```

OutputCOMP

```
MODEL
   OutputCOMP
SEES
   Char, PString, SeqSimp
VARIABLES
   ob,o,                                  /* internal */
   r,t,s,oc,cw,co,ot,dc,otc,dtc,twc       /* external */
INVARIANT
   ob ∈ seq(C) ∧
   ∃ ps.(ps ∈ PS ∧ ob  ⊆ ps) ∧
   r ∈ ℕ ∧
```

```
   t ∈ r-1  ..  r ∧
   s ∈ r-1  ..  r ∧
   s ∈ t-1  ..  t ∧
   (s ≠ t ⇒ ob ∈ PS) ∧
   (s = r ⇒ ob = ⟨⟩) ∧
   o ∈ 0 .. maxsize ∧
   oc ∈ C ∧
   cw ∈ ℕ ∧
   o ≤ cw ∧
   co ∈ cw-1 .. cw ∧
   (t\ner ⇒ o=0) ∧
   (s=t ⇒ o=0) ∧
   dc ∈ ℕ ∧
   twc ∈ C ∧
   dtc ∈ ℕ ∧
   ot ∈ dc  ..  dc+1 ∧
   ot ∈ ℕ ∧
   otc ∈ ℕ ∧
   dtc ∈ otc  ..  otc+1 ∧
   (ot = dc ⇒ otc = dtc) ∧
   (otc ≠ dtc ⇒ ot ≠ dc) ∧
   (r≠ t ∧ otc ≠ dtc ⇒ ∃ ps.(ps ∈ PS ∧ ob←twc  ⊆ ps)) ∧
   (r=t ⇒ otc = dtc)
THEOREMS
  (s=r ⇒ t=r) ∧ (s\net⇒r=t)
INITIALISATION
  r,t,s,o := 0,0,0,0 ‖ ob := ⟨⟩ ‖
  co,cw := 0,0 ‖ oc :∈ C ‖ dc,ot:=0,0 ‖
  dtc := 0 ‖ twc :∈ C ‖ otc := 0
EVENTS
/* internal */
  nextow = WHEN r\net ∧ ot = dc THEN
        ot := ot+1 ‖ ob := ⟨⟩
    END;
  nextowc = WHEN r≠t ∧ otc ≠ dtc  THEN
        ob := ob←twc ‖ otc := otc+1
  END;
  writec = WHEN s ≠ t ∧ cw = co ∧ o < size(ob)THEN
        oc,o,cw := ob(o+1),o+1,cw+1
    END;
  output = WHEN s ≠ t ∧ o = size(ob) THEN
        s := s+1 ‖ o := 0 ‖ ob := ⟨⟩
    END;
/*external*/
/* input PIC */
  input = WHEN s = r  THEN
```

```
        r := r+1
    END;
/* dictionary */
  nextdc = WHEN r≠t ∧ dtc = otc ∧ ot ≠ dc THEN
        twc :| (twc ∈ C ∧
        ∃ ps.(ps ∈ PS ∧ ob←twc  ⊆ ps)) ‖ dtc := dtc+1
    END;
/* comparator */
  chkc = WHEN r≠t ∧ otc = dtc ∧ ot ≠ dc THEN
        dc :∈ dc .. dc+1
    END;
  release = WHEN t ≠ r ∧ ob ∈ PS ∧ ot ≠ dc ∧ otc = dtc THEN
        t := t+1
    END;
/*reset*/
  reset = BEGIN
        o,ob,s,t,co := 0,⟨⟩,r,r,cw ‖
        otc := dtc ‖ ot:= dc
    END;
/*environment*/
  recvc = WHEN cw ≠ co THEN
         co := co +1
    END
END
```

Appendix

Security Target

A Security Target for the Trusted Filter

Appendix 1

Components of a Security Target

1.1 Security Target Contents

Figure 1.1 (from the Common Criteria part 1 [4], Fig. 5) lists the components of a Security Target. It contains the following sections.

1. an ST introduction containing three narrative descriptions of the TOE on different levels of abstraction;
2. a conformance claim, showing whether the ST claims conformance to any PPs and/or packages, and if so, to which PPs and/or packages;
3. a security problem definition, showing the threats, OSPs and assumptions that must be countered, enforced and upheld by the TOE and its operational environment;
4. security objectives, showing how the solution to the security problem is divided between security objectives for the TOE and security objectives for the operational environment of the TOE;
5. security requirements, where a translation of the security objectives for the TOE into a standardised language is provided. This standardised language is in the form of SFRs. Additionally this section defines the SARs;
6. a TOE summary specification, showing how the SFRs are implemented in the TOE.

An ST may also contain an *extended components definition*, where new components (i.e. not included in CC Part 2 or CC Part 3) may be defined. But this is not required for the *Trusted Filter*.

These components are described briefly in the following sections.

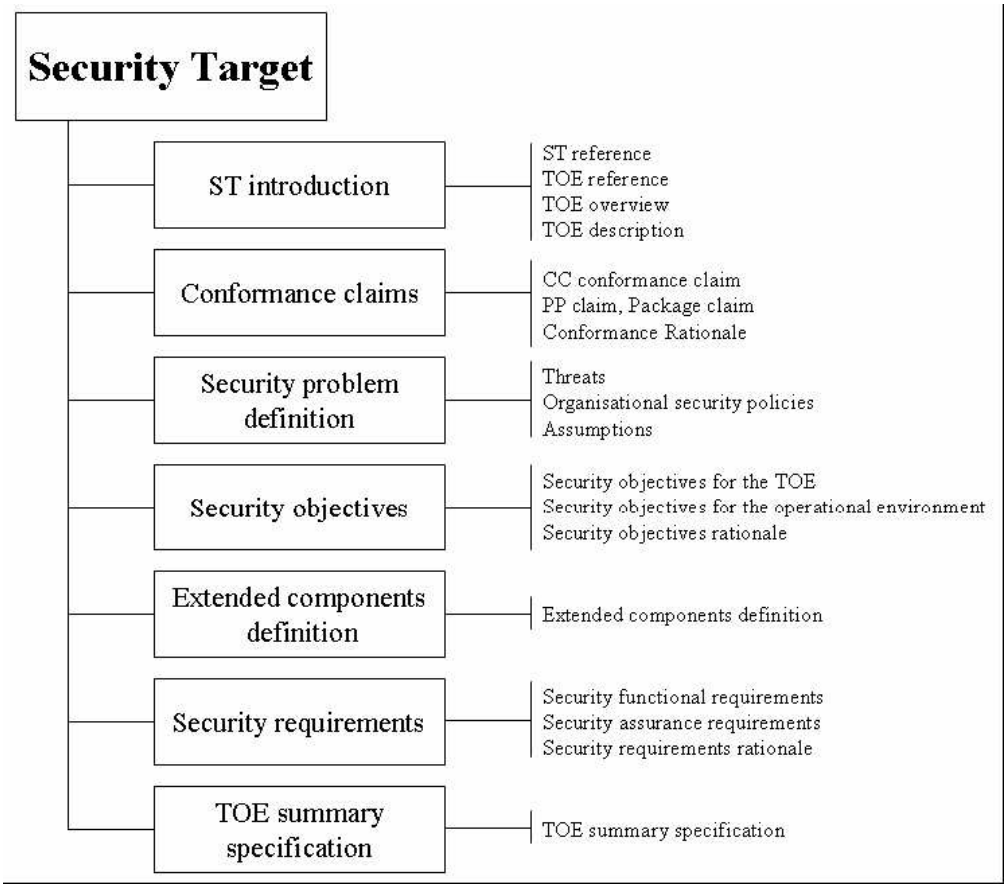

Figure 1.1: Components of a Security Target

Appendix 2

Security Target Introduction

2.1 ST and TOE Reference

- Security Target: *ST for the Trusted Filter* – version 1.0.
- Target of Evaluation: *Trusted Filter Prototype* – version 1.1 (12/12/1005).

2.2 TOE Overview

A *Trusted Filter* is used to control unidirectional data transfer between two systems with different security classification levels. The filter acts as a control point between the two systems, determining the traffic flow between them.

The particular Trusted Filter considered here controls traffic between a a high-security and a low-security system. It only allows traffic to be transmitted from high to low. This traffic is a set of commands issued by high security user, typically to control some process or device on the low side. Authorization is determined word by word, by checking against a fixed list of permitted commands. If a word is not in the list of authorized commands then it is not transmitted.

The prime security requirement is that no unauthorized words that might be input to the high side are transmitted to the low side.

In addition, the implementation is required to shut down if an attempt is made to transmit an unauthorized word. This requirement has no impact on security, since a device that does not transmit anything trivially satisfies the security requirements.

2.3 TOE Description

Note: this section is a summary of the of information in the document Trusted Filter – Design Overview, *Vicky Moyle, Luke Sledziona, 2005*. [12]

The trusted filter has been designed to interface between a high security classification host and a device on a low security classification network. Unlike more complex commercial trusted filters, micro controllers or processors do not control this device. All core functionality is provided via hardware logic and clocking. It is expected that the trusted filter will be more secure and have a higher fault tolerance because of its hardware implementation.

A trusted filter is used for unidirectional data transfer between two systems of varying classification levels. The filter in such a scenario acts as a control point between the two systems determining the traffic flow between the two levels. The most common implementation of a trusted filter is between a higher classified system and a lower classified system in which traffic flow from low to high is blocked whilst traffic from high to low is allowed to flow. The trusted filter has been designed to connect to both systems via a serial RS-232 interface.

Appendix 3

TOE Conformance Claims

3.1 CC conformance claim

This Security Target conforms to CC 3.1 Rev 1 and contains no extended security requirements.

Appendix 4

Security Problem Definition

4.1 Security Problem Definition

The general setting of the problem is the interface between areas of high and low security and the passing of data from high to low in a controlled manner. The particular situation in which the *Trusted Filter* is used is where a command stream of limited vocabulary is transmitted from an area of high security to one of low security. An example of this is a link sending device-control commands from a workstation in a secure area to an external device in an unsecured area.

The security problem is to ensure that the link cannot be used to pass unauthorized information to the insecure area.

4.2 Threats

A threat consists of of a *threat agent* an *asset* and and an *adverse action* of that threat agent on that asset. In the case of the *Trusted Filter* the asset is access to classified information.

Table of Threats

#	Agent	Action.
T1	Unauthorized user	Undetected compromise of asset due to unauthorized use
T2	Authorized user	An authorized user may intentionally or accidentally transmit sensitive information
T3	User	An authorized or unauthorized person may cause a security breach by an unauthorized restart
T4	–	Integrity of information may be compromised due to hardware errors
T5	–	Weaknesses in the design or implementation may transmit unchecked data

4.3 Organisational Security Policies

This section itemizes any actual organizational security policies that the TOE must meet.

P1 Only data not classified as *secure* or specifically declassified may be transmitted from secure to insecure areas.

4.4 Assumptions

A1 The TOE is operated within a physically secure environment, protected from unauthorized access.

A2 The contents of the reference dictionary is:

(a) Checked and authorized by someone at an appropriate level of security clearance and authority.

(b) Installed correctly in the device.

A3 Operation procedures are defined which cover both normal use and the action to be taken in the event of the machine shutting down due to invalid data or other circumstances.

A4 the device is only operated by personnel who are trained in the authorized procedures.

Appendix 5

Security Objectives

5.1 High-level solution

The *Trusted Filter* controls all data passing over the subject link. It checks every command passed from the high-security area and confirms that it is one of the authorized commands listed in its dictionary. All authorized commands are transmitted, but if an attempt is made to send a non-authorized command the filter shuts down and awaits intervention by an operator (i.e. a manual reset). The filter also monitors its own correct functioning and similarly closes down if an internal error is detected.

5.2 Part-wise Solution

5.2.1 Security objectives for the TOE

TOE1 The TOE shall control all data passing over the subject link.

TOE2 The TOE shall check every command it receives against the commands listed in its dictionary.

TOE3 The TOE shall only forward commands received if they match a dictionary entry.

TOE4 The TOE shall shut-down if an attempt is made to send a non-authorized command or if an internal error is detected.

5.2.2 Security objectives for the environment

OE1 The operational environment shall ensure that the dictionary in the TOE is set up by an authorized person and only contains authorized data.

OE2 The operational environment shall set up the TOE in a secure area.

OE3 The operational procedures shall ensure that only authorized persons have access to the TOE.

5.3 Relation between the objectives and the problem definition

#	Objective.	Threat	Policy	Assump
TOE1	TOE controls all data passing over the subject link.	T2	P1	
TOE2	TOE checks every command received	T2	P1	
TOE3	TOE only forwards commands in the dictionary	T2	P1	
TOE4	TOE shuts down on error	T4, T5		
OE1	OE ensures dictionary set up correctly	T1, T5	P1	A2
OE2	OE sets up the TOE in a secure area.	T1, T3	P1	A1
OE3	OE ensures that only authorized persons have access to the TOE.	T1, T3		A1

5.4 Security objectives rationale

Threat T1 Undetected compromise of asset due to unauthorized use. OE2 and OE3 ensure that the TOE is in a secure environment with no unauthorized access.

Threat T2 An authorized user may intentionally or accidentally transmit sensitive information . TOE1 and TOE2 ensure that all data passing over the subject link passes through the TOE and is checked by it. TOE3 ensures that no data forwarded by the TOE is sensitive.

Threat T3 An authorized or unauthorized person may cause a security breach by an unauthorized restart. As with Threat T1, no unauthorized person has access to the machine (by OE2 and OE3). In the case of authorized users, this is addressed by assumption A3 and OE4, which ensure that the appropriate procedures are in place and are followed.

Threat T4 Integrity of information may be compromised due to hardware errors. This is addressed by the shut-down function (TOE4).

Threat T5 Weaknesses in the design or implementation may transmit unchecked data. This is addressed by the shut-down function (TOE4).

OSP P1 Only data not classified as *secure* or specifically declassified can be transmitted to insecure areas. This is ensured by TOE3 which guarantees that only data in the dictionary is passed to the insecure side of the link, and OE1 which ensures that the dictionary data has been set up (as de-classifiable) by a person with appropriate authorization.

Assump. A1 The TOE is operated within a secure environment, protected from unauthorized access. This is equivalent to the objectives OE2 and OE3 in combination.

Assump. A2 The contents of the reference dictionary is installed correctly and authorized at the appropriate level. This includes ensuring that transmitting items from the dictionary, in any sequence, cannot be used to transmit unauthorized secure information. This is equivalent to the objective OE1.

Appendix 6

Extended Components Definitions

None.

Appendix 7

Security Requirements

7.1 Security Functional Requirements (SFRs)

TOE evaluation is concerned primarily with ensuring that a defined set of security functional requirements (SFRs) is enforced over the TOE resources. The SFRs define the rules by which the TOE governs access to and use of its resources, and thus information and services controlled by the TOE.

The SFRs may, in turn, include multiple Security Function Policies (SFPs). Each SFP has a scope of control, that defines the subjects, objects, resources or information, and operations controlled under the SFP. All SFPs are implemented by the TSF (see below), whose mechanisms enforce the rules defined in the SFRs and provide necessary capabilities.

Those portions of a TOE that must be relied on for the correct enforcement of the SFRs are collectively referred to as the TOE Security Functionality (TSF). The TSF consists of all hardware, software, and firmware of a TOE that is either directly or indirectly relied upon for security enforcement.

7.1.1 Security Function Policy

SFP: SFP-TIF - Trusted Information flow The functional requirement for the *Trusted Filter* device is that only commands in the authorized dictionary are transmitted to the link passing to the insecure side.

- The *subjects* of the policy are:

  A Authorized users of the TOE

  B Any users (unrestricted)

- The *information* under control of the policy is any high-side data stream determined as requiring to be under the control of the TOE by other security policies.

- The *operations* under control of the policy consist of the simple transfer of data by the TOE from (A) to (B).

7.1.2 Security Functional Requirements

SFP: FDP_IFC Information flow control policy

This family identifies the information flow control SFPs (by name) and defines the scope of control for each named information flow control SFP

This scope of control is characterised by three sets: the subjects under control of the policy, the information under control of the policy, and operations which cause controlled information to flow to and from controlled subjects covered by the policy. The criteria allows multiple policies to exist, each having a unique name. This is accomplished by iterating components from this family once for each named information flow control policy.

The rules that define the functionality of an information flow control SFP will be defined by other families such as Information flow control functions (FDP_IFF) and Stored data integrity (FDP_SDI). The names of the information flow control SFPs identified here in Information flow control policy (FDP_IFC) are meant to be used throughout the remainder of the functional components that have an operation that calls for an assignment or selection of an 'information flow control SFP'.

FDP_IFC.2 - complete information flow

The TSF shall enforce the information flow control policy SFP-TIF on users (A) and (B) and any information input to the TOE and all operations that cause that information to flow to and from subjects covered by the SFP.

The TSF shall ensure that all operations that cause any information in the TOE to flow to and from any subject in the TOE are covered by an information flow control SFP.

SFP: FDP_IFF Information flow control functions

This family describes the rules for the specific functions that can implement the information flow control SFPs named in Information flow control policy (FDP_IFC), which also specifies the scope of control of the policy.

It consists of two kinds of requirements: one addressing the common information flow function issues, and a second addressing illicit information flows (i.e. covert channels). This division arises because the issues concerning illicit information flows are, in some sense, orthogonal to the rest of an information flow control SFP. By their nature they circumvent the information flow control SFP resulting in a violation of the policy. As such, they require special functions to either limit or prevent their occurrence.

FDP_IFF.1 Simple security attributes, requires security attributes on information, and on subjects that cause that information to flow and on subjects

that act as recipients of that information. It specifies the rules that must be enforced by the function, and describes how security attributes are derived by the function.

Appendix 8

TOE Summary Specification

The Trusted Filter ensures controlled information flow by acting as a gateway between input and output RS232 ports. All commands pass through the filtering logic, which checks them serially against the contents of an authorized commands dictionary which is held in EPROM. Only commands for which there is a positive match are released to the output port.

If an error occurs the filter shuts down and can only recommence operation after the external reset switch is pressed or power to the device is cycled. Shutdown can be triggered by:

- Inconsistent results from the internal logic.
- The end of dictionary being found without a match.
- An error is detected by either the input or output PIC microprocessors.

When a shut-down state occurs, the processing of further commands is disabled, and the device is locked in an inactive state. A LED connected to the shut-down line alerts the user that the device is in shut-down and therefore needs resetting.

The external reset switch used by the trusted filter is a momentarily on push button switch. By pressing the reset button, the shut-down condition is cleared and the device reactivated.

The list of valid commands that the trusted filter will allow to pass is stored within a dictionary contained in EPROM memory. This dictionary consists of 7 character commands that are terminated by an identifying, end of command byte (FF). Commands less than 7 characters long padded with the end of dictionary byte (DF). If the end of dictionary byte is processed, the trusted filter will have entered an invalid state and will shut down.