NASA Contractor Report 189698

Formal Design Specification of a Processor Interface Unit

David A Fura  
Boeing Defense & Space Group  
Seattle, Washington

Phillip J. Windley  
University of Idaho  
Moscow, Idaho

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

NASA Contract NAS1-18586  
November 1992

NASA Contractor Report 189698

Formal Design Specification of a Processor Interface Unit

David A Fura  
Boeing Defense & Space Group  
Seattle, Washington

Phillip J. Windley  
University of Idaho  
Moscow, Idaho

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

NASA Contract NAS1-18586  
November 1992

(NA24-CR-189698)  
FORMAL DESIGN SPECIFICATION OF A PROCESSOR INTERFACE UNIT (Boeing Military Airplane Development)  

NASA  
National Aeronautics and Space Administration  
Langley Research Center  
Hampton, Virginia 23665-5525

NASA Contractor Report 189698

Formal Design Specification of a Processor Interface Unit

David A Fura  
Boeing Defense & Space Group  
Seattle, Washington

Phillip J. Windley  
University of Idaho  
Moscow, Idaho

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

NASA Contract NAS1-18586  
November 1992

(NA24-CR-189698)  
FORMAL DESIGN SPECIFICATION OF A PROCESSOR INTERFACE UNIT (Boeing Military Airplane Development)  

NASA  
National Aeronautics and Space Administration  
Langley Research Center  
Hampton, Virginia 23665-5525
Preface

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

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

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

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

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

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

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

1 Introduction .......................................................................................................................... 1
  1.1 Informal PIU Description ................................................................................................. 1
    1.1.1 PMM Initialization .................................................................................................... 3
  1.1.2 CPU Accesses to Memory .......................................................................................... 4
    1.1.2.1 To Local Memory ............................................................................................... 4
    1.1.2.2 To Internal Register File ................................................................................... 5
    1.1.2.3 To the C_Bus .................................................................................................... 6
  1.1.3 C_Bus Accesses to Memory ......................................................................................... 6
  1.1.4 Timers and Interrupts ................................................................................................. 6
  1.2 Specification Overview .................................................................................................... 6

2 Generic Interpreter Theory .................................................................................................. 9
  2.1 Introduction .................................................................................................................... 9
  2.2 Formal Microprocessor Modeling .................................................................................. 9
    2.2.1 Microprocessor Specification ................................................................................ 9
    2.2.2 Microprocessor Verification ................................................................................. 10
  2.3 A Formal Model of Interpreters .................................................................................... 10
    2.3.1 Abstract Theories ............................................................................................... 10
    2.3.2 Temporal Abstraction ......................................................................................... 12
    2.3.3 The Abstract Representation .............................................................................. 12
    2.3.4 The Theory Obligations ...................................................................................... 14
    2.3.5 Abstract Theorems .............................................................................................. 15
      2.3.5.1 Defining the Interpreter ............................................................................... 15
      2.3.5.2 Induction on Interpreters ........................................................................... 15
      2.3.5.3 The Implementation is Live ........................................................................ 16
      2.3.5.4 The Correctness Statement .......................................................................... 16
      2.3.5.5 Composing Interpreters Hierarchically ...................................................... 17
  2.4 Parallel Composition ...................................................................................................... 17
  2.5 Conclusion ..................................................................................................................... 17

3 Design Specification .......................................................................................................... 19
  3.1 Gate-Level Structure ...................................................................................................... 19
    3.1.1 Component Descriptions ...................................................................................... 19
      3.1.1.1 Combinational Logic ..................................................................................... 19
      3.1.1.2 Latches ......................................................................................................... 20
      3.1.1.3 Flip-Flops ...................................................................................................... 22
      3.1.1.4 Counters ........................................................................................................ 23
      3.1.1.5 CTR Datapath Block ................................................................................... 23
      3.1.1.6 ICR Datapath Block .................................................................................... 25
      3.1.1.7 CR Datapath Block ..................................................................................... 26
      3.1.1.8 SR Datapath Block ...................................................................................... 26
      3.1.1.9 Finite-State Machines ................................................................................. 26
    3.1.2 Block Diagram Descriptions .................................................................................... 27
      3.1.2.1 P_Port Structure ........................................................................................... 28
      3.1.2.2 M_Port Structure ......................................................................................... 29
      3.1.2.3 R_Port Structure ......................................................................................... 32
      3.1.2.4 C_Port Structure ......................................................................................... 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.4</td>
<td>C Port Specification</td>
<td>198</td>
</tr>
<tr>
<td>D.5</td>
<td>SU_Cont Specification</td>
<td>209</td>
</tr>
<tr>
<td>E</td>
<td>ML Source for the PIU Block-Level Specification</td>
<td>215</td>
</tr>
<tr>
<td>F</td>
<td>ML Source for the PIU Clock-Level Specification</td>
<td>219</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Block Diagram of the Processor-Memory Module (PMM) ....................................................... 2
1.2 Major Blocks of the Processor Interface Unit (PIU) ............................................................... 3
1.3 PIU Specification Hierarchy .................................................................................................. 7

2.1 A Hierarchy of Interpreters .................................................................................................. 11
2.2 The Temporal Abstraction Functions $F$ and $G$ ................................................................ 12

3.1 Two Series Latches Clocked by the Same Phase ................................................................. 21
3.2 Interval Representations ...................................................................................................... 22
3.3 Example D Flip-Flop Constructed With Latches .................................................................. 23
3.4 Functional Block Diagram of a Counter ................................................................................ 24
3.5 Functional Block Diagram of the CTR Datapath Block ....................................................... 24
3.6 Functional Block Diagram of the ICR Datapath Block ....................................................... 25
3.7 Functional Block Diagram of the CR Datapath Block ....................................................... 26
3.8 Functional Block Diagram of the SR Datapath Block ....................................................... 27
3.9 Functional Block Diagram for Finite-State Machines ......................................................... 27
3.10 P_Port Top-Level Block Diagram ....................................................................................... 28
3.11 Block Diagram of P_Port Datapath ..................................................................................... 29
3.12 Block Diagram of P_Port Controller .................................................................................. 30
3.13 M_Port Top-Level Block Diagram ..................................................................................... 30
3.14 Block Diagram of the M_Port Datapath ............................................................................. 31
3.15 Block Diagram of the M_Port Controller .......................................................................... 32
3.16 R_PORT Top-Level Block Diagram .................................................................................... 33
3.17 Block Diagram of Register File Controller .......................................................................... 33
3.18 Block Diagram of the Timer Interrupt Block ...................................................................... 34
3.19 Block Diagram of the Register Interrupt Block .................................................................... 34
3.20 C_PORT Top-Level Block Diagram ................................................................................... 35
3.21 Block Diagram of the C_POINT Datapath ........................................................................ 35
3.22 Block Diagram of the C_POINT Controller (Part A) ......................................................... 36
3.23 Block Diagram of the C_POINT Controller (Part B) .......................................................... 37
3.24 Block Diagram of the Startup Controller PIU-Port Interface ............................................. 38
3.25 Block Diagram of the Startup Controller CPU Interface .................................................. 39

4.1 The View from the CPU ........................................................................................................ 43
4.2 View from the Memory .......................................................................................................... 44
4.3 View from the Network ......................................................................................................... 44
4.4 Abstraction Views for the PIU .............................................................................................. 45
4.5 Modeling the Buses in a Computer System using Tuple Space ........................................... 47
List of Tables

1.1  R_Port Register Definitions ................................................................. 5
2.1  The abstract functions and their types for the generic interpreter model ................. 13
1 Introduction

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

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

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

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

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

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

Section 6 contains a concluding discussion.

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

1.1 Informal PIU Description

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

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

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

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

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

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

---

Figure 1.1: Block Diagram of the Processor-Memory Module (PMM).
The Register Port (R_Port) is the fourth, and final, port residing on the PIU's Internal Bus (I_Bus). It contains a state machine, counters, and various command and status registers used by the local processor to implement timers and interrupts.

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

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

1.1.1 PMM Initialization

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

![Figure 1.2: Major Blocks of the Processor Interface Unit (PIU).](image)
other CPU. During the initialization, the PIU also maintains the inter-PMM synchronization that is initially established by the FTCUs.

The PIU initiates CPU self-test via the CPU reset signals that it controls. To begin the initialization sequence, the PIU resets CPU0, which then goes through a two-phase (Intel 80960) testing process of its own. In the first phase the CPU executes a 47,000-cycle self-test procedure; in the second phase the CPU reads the first eight words of local memory (via the PIU) and performs a checksum test. If either of these tests fail, then the CPU's failure0 pin remains asserted, otherwise it is deasserted.

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

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

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

1.1.2 CPU Accesses to Memory

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

1.1.2.1 To Local Memory

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

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

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

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

Byte accesses of up to four words are also supported to implement cache refilling within the CPU.
1.1.2.2 To Internal Register File

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

The Interrupt Control Register (ICR) supports memory-mapped interrupts to the local processor. The register is divided into four fields. The first two contain the interrupt settings and mask bits for int0, in bits 0 through 7 and 8 through 15, respectively. A logic-1 in both a set location and the associated mask location signifies an active interrupt, which if enabled (external to the R_Port) will generate an active int0_ signal to the processor. Bits 16 through 31 are used in a corresponding way for int3_.

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

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

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

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

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

Table 1.1: R_Port Register Definitions.

<table>
<thead>
<tr>
<th>Register Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interrupt Control Register (ICR) reset</td>
</tr>
<tr>
<td>1</td>
<td>ICR set</td>
</tr>
<tr>
<td>2</td>
<td>General Control Register (GCR)</td>
</tr>
<tr>
<td>3</td>
<td>Communication Control Register (CCR)</td>
</tr>
<tr>
<td>4</td>
<td>Status Register (SR)</td>
</tr>
<tr>
<td>8</td>
<td>Counter 0 in</td>
</tr>
<tr>
<td>9</td>
<td>Counter 1 in</td>
</tr>
<tr>
<td>10</td>
<td>Counter 2 in</td>
</tr>
</tbody>
</table>
Table 1.1: R_Port Register Definitions.

<table>
<thead>
<tr>
<th>Register Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Counter 3 in</td>
</tr>
<tr>
<td>12</td>
<td>Counter 0 out</td>
</tr>
<tr>
<td>13</td>
<td>Counter 1 out</td>
</tr>
<tr>
<td>14</td>
<td>Counter 2 out</td>
</tr>
<tr>
<td>15</td>
<td>Counter 3 out</td>
</tr>
</tbody>
</table>

1.1.2.3 To the C_Bus

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

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

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

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

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

1.1.3 C_Bus Accesses to Memory

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

1.1.4 Timers and Interrupts

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

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

1.2 Specification Overview

Figure 1.3 shows the specification hierarchy developed for the PIU. In constructing this hierarchy much emphasis was placed on maintaining compatibility with existing formal specification methods, particularly the generic interpreter theory described in Section 2. The resulting hierarchy reflects this emphasis, particularly in the lower levels where many of the techniques described in [Win90] are used.
Consistent with established hierarchical specification methods, the levels in the hierarchy of Figure 1.3 are abstractions of the levels below them. Four types of abstraction are used here. Temporal abstraction relates time at a particular level to the time at lower levels; each unit of time at the higher level corresponds to multiple time units at the lower level. Data abstraction relates the states of two levels, with the higher level state being a function (typically a subset) of the state at the lower level. In behavioral abstraction, a structural description at the lower level, defined using the physical interconnection of components or subsystems, is replaced by a purely behavioral description at the higher level. Structural abstraction (or composition) combines subsystems defined at one level to form a higher level.

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

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

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

![Figure 1.3: PIU Specification Hierarchy.](image-url)
The port-level structure is a structural composition of the five individual clock-level port specifications. The port composition is based on the established method of forming a logical conjunction of the individual port descriptions.

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

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

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

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

2.1 Introduction.

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

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

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

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

2.2 Formal Microprocessor Modeling.

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

2.2.1 Microprocessor Specification.

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

1. A representation of the state, $S$.

2. A set of state transition functions, $J$, denoting the behavior of the individual instructions of the microprocessor. Each of these functions takes the state defined in step (1) as an argument and returns the state updated in some meaningful way.

3. A selection function, $N$, that selects a function from the set $J$ according to the current state.

4. A predicate, $I$, relating the state at time $t+1$ to the state at time $t$ by means of $J$ and $N$.

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

2.2.2 Microprocessor Verification.

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

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

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

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

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

2.3 A Formal Model of Interpreters.

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

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

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

2.3.1 Abstract Theories.

A theory is a set of types, definitions, constants, axioms and parent theories. Logics are extended by defining new theories. An abstract theory is parameterized so that some of the types and constants defined
in the theory are undefined inside the theory except for their syntax and a loose algebraic specification of their semantics. Group theory is an example of an abstract theory. The multiplication operator is undefined except for its syntax (a binary operator on type "$\text{group}$") and a loose semantics given by the axioms of group theory.

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

An abstract theory consists of three parts:

1. An abstract representation of the uninterpreted constants and types in the theory. The abstract representation contains a set of abstract operations and a set of abstract objects. (These are sometimes called uninterpreted constants and uninterpreted types.)

2. A set of theory obligations defining relationships between members of the abstract representation. Inside the theory, the obligations represent axiomatic knowledge concerning the abstract representation. Outside the theory, the obligations represent the criteria that a concrete representation must meet if it is to be used to instantiate the abstract theory.

3. A collection of abstract theorems. The theorems are generally based on the theory obligations and can stand alone only after the theory obligations have been met.

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

There are several specification and verification systems that support abstract theories. Some, such as OBJ [Gog88] and EHDM [SR188], offer explicit support. HOL, the verification environment used for the
research reported here, does not explicitly support abstract theories; however, HOL's metalanguage, ML, combined with higher-order logic, provides a framework for implementing abstract theories [Win90a] in a manner that does not degrade the trustworthiness of the theorem prover.

2.3.2 Temporal Abstraction

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

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

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

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

2.3.3 The Abstract Representation

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

The following abstract types are used in the representation.

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

![Figure 2.2: The Temporal Abstraction Functions F and G.](image-url)
Table 2.1: The abstract functions and their types for the generic interpreter model.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>instructions</td>
<td>&quot;:*key-&gt;(*state-*env-*state)&quot;</td>
</tr>
<tr>
<td>select</td>
<td>&quot;:*state-*env-*key&quot;</td>
</tr>
<tr>
<td>output</td>
<td>&quot;:*key-&gt;(*state-*env-*out)&quot;</td>
</tr>
<tr>
<td>substate</td>
<td>&quot;:*state'-*state&quot;</td>
</tr>
<tr>
<td>subenv</td>
<td>&quot;:*env'-*env&quot;</td>
</tr>
<tr>
<td>subout</td>
<td>&quot;:*out'-*out&quot;</td>
</tr>
<tr>
<td>Impl</td>
<td>&quot;:(time'-*state')-(time'-*env')-&gt;bool&quot;</td>
</tr>
<tr>
<td>count</td>
<td>&quot;:*state'-*env'-*key'&quot;</td>
</tr>
<tr>
<td>start</td>
<td>&quot;:*key'&quot;</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

2.3.4 The Theory Obligations

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

The predicate INSTRUCTION_CORRECT expresses the conditions that we require in the instruction correctness lemma:

\[ \text{\texttt{INSTRUCTION\_CORRECT}} \]

\[ \begin{array}{l}
|\text{def} \text{INSTRUCTION\_CORRECT} \text{ gi s' e' inst } = \\
(\text{Impl gi s' e' }) ==>
(\text{!:time'}. \\
\text{let st = (substate gi (s't)) in} \\
\text{let et = (subenv gi (e't)) in} \\
\text{let ft = (count gi (s't)) (e't = (start gi)) in} \\
\text{let k = (select gi (st)) (et)) in} \\
(\text{inst = (instructions gi k)} \land (ft) ==>
?c. \text{Next f(t+c)} \land (\text{inst (st) (et) = (s(t+c))))})
\end{array} \]

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

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

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

\[ \begin{array}{l}
|\text{def} \text{OUTPUT\_CORRECT} \text{ gi s' e' p' k } = \\
(\text{Impl gi s' e' p' }) ==>
(\text{!:time'}. \\
\text{let st = (substate gi (s't)) in} \\
\text{let et = (subenv gi (e't)) in} \\
\text{let pt = (subout gi (p't)) in} \\
\text{let ft = (count gi (s't)) (e't = (start gi)) in} \\
((\text{count gi (s't)} \land (e't) ==>
\text{select gi (st) (et) = k) ==>
(pt = (output gi k) (st) (et))))})
\end{array} \]

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

14
Using \texttt{INSTRUCTION\_CORRECT} and \texttt{OUTPUT\_CORRECT} we can define the theory obligations in our model. The theory obligations are given as a predicate on an abstract representation \( g_i \):

\[
|_{\text{def}} G_{g_i} = (\forall s' e' p' k. \text{INSTRUCTION\_CORRECT} g_i s' e' p' k) \land (\forall s' e' p' k. \text{OUTPUT\_CORRECT} g_i s' e' p' k)
\]

The predicate says that every instruction in the instruction set satisfies the predicate \texttt{INSTRUCTION\_CORRECT} and every output function satisfies the conditions set forth in \texttt{OUTPUT\_CORRECT}.

### 2.3.5 Abstract Theorems

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

#### 2.3.5.1 Defining the Interpreter

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

\[
|_{\text{def}} \text{INTERP} g_i s e p = \\
\forall t. \text{time}.
let k = (\text{select} g_i (s t) (e t)) in
(s(t+1) = (\text{instructions} g_i k (s t) (e t)) \land
(p t = (\text{output} g_i k (s t) (e t))
\]

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

#### 2.3.5.2 Induction on Interpreters

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

\[
|_{\forall} Q. \text{INTERP} g_i s e p => \\\n(Q s0) \land \\
\forall t. \text{let} \text{inst} = (\text{instructions} g_i (\text{select} g_i (s t) (e t)) in \\
Q (s t) => Q (\text{inst} (s t) (e t))) => \\
\forall t. Q (s t)
\]

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

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

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

\[ \text{Impl } gi \ s' \ e' \implies (\forall t. (\text{count } gi \ (s'O \ (e't)) = \text{start } gi \implies (\exists n. \text{Next} (t. \text{count } gi \ (s'I \ (e'I) = \text{start } gi) \ (t, t+n)))} \]

*Next* \(P(t_1, t_2)\) says that \(t_2\) is the next time after \(t_1\) when \(P\) is true.

2.3.5.4 The Correctness Statement

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

\[ \begin{align*}
\text{let } & st = (\text{substate } gi \ (s'O)) \text{ and } \\
& et = (\text{subenv } gi \ (e'I)) \text{ and } \\
& pt = (\text{subout } gi \ (p'I)) \text{ and } \\
& ft = (\text{count } gi \ (s'I \ (e'I) = (\text{start } gi)) \text{ in } \\
\text{let abs} = (\text{Temp_ABS}) \text{ in } \\
\text{(Impl } gi \ s' \ e' \ p') \land \\
(\forall t. ft) \implies \\
(\text{INTERP } gi) \ (s \ o \ abs) \ (e \ o \ abs) \ (p \ o \ abs)
\end{align*} \]

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


2.3.5.5 Composing Interpreters Hierarchically

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

\[
\begin{align*}
\text{let } s't &= (\text{substate } gi 1 (s''t)) \quad \text{and} \\
 e't &= (\text{subenv } gi 1 (e''t)) \quad \text{and} \\
 p't &= (\text{subout } gi 1 (p''t)) \quad \text{and} \\
 ft &= (\text{count } gi 1 (s''t) (e''t) = \text{start } gi 1) \quad \text{in} \\
\text{let } st &= (\text{substate } gi 2 (s't)) \quad \text{and} \\
 e t &= (\text{subenv } gi 2 (e't)) \quad \text{and} \\
 pt &= (\text{subout } gi 2 (p't)) \quad \text{and} \\
 gt &= (\text{select } gi 1 (s't) (e't) = \text{start } gi 2) \quad \text{in} \\
\text{let } abs1 &= (\text{Temp_ABS } f) \quad \text{in} \\
\text{let } abs2 &= abs1 o (\text{Temp_ABS } (g o abs1)) \quad \text{in} \\
(\text{Impl } gi 1 s'' e'' p'') \quad \text{and} \\
(?) t f t &= \Rightarrow \\
(?) (g o abs1) t &= \Rightarrow \\
\text{INTERP } gi 2 (s o abs2) (e o abs2) (p o abs2)
\end{align*}
\]

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

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

2.4 Parallel Composition

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

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

2.5 Conclusion

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

The theory has many important benefits:

- The generic model structures the proof by stating explicitly which definitions must be made (one for each of the members of the abstract representation) and which lemmas need to be proven about these definitions (namely, the theory obligation). This is a substantial improvement over previous microprocessor verifications where these decisions were made on an *ad hoc* basis.

- The generic model insulates users of the model from complex proofs about the data and temporal abstractions. These proofs are done once and then made available to the user by instantiation.

- The use of a generic interpreter model for specifying and verifying microprocessors provides a methodological approach. Making specification and verification methodological is an important step in turning what has been primarily a research activity into an engineering activity.
3 Design Specification

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

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

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

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

3.1 Gate-Level Structure

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

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

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

3.1.1 Component Descriptions

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

3.1.1.1 Combinational Logic

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

\[ \neg \text{AND3_SPEC } a \ b \ c \ z = \forall \ t : \text{time}. \ z \ t = (a \ t) \land (b \ t) \land (c \ t) \]

### 3.1.1.2 Latches

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

The following definition of a D latch demonstrates the specification style that we use for PIU latches. This specification states that the next state \( q_{\text{state}}(t+1) \) equals the input \( d_{\text{in}}(t) \) if the clock \( \text{clk}_{\text{in}}(t) \) is active, otherwise it equals its current value \( q_{\text{state}}(t) \). The latch output \( q_{\text{out}}(t) \) equals the new state.

\[ \neg \text{DLAT_SPEC } d_{\text{in}} \ \text{clk}_{\text{in}} \ q_{\text{state}} \ q_{\text{out}} = \]
\[ \forall \ t : \text{time}. \\
( q_{\text{state}}(t+1) = (\text{clk}_{\text{in}}(t) => d_{\text{in}}(t) \land q_{\text{state}}(t)) \land \\
( q_{\text{out}}(t) = q_{\text{state}}(t+1)) \]

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

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

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

Revisiting fundamental FSM definitions suggests ways to solve this latch modeling problem. In automata theory texts, such as [Koh78], the next-state and present-output of an FSM are said to be functions of
Latch L1

\[ a \xrightarrow{\text{D}} b \xrightarrow{\text{Q}} z \]

Latch L2

\[ \text{phase}_A \]

\[ \text{phase}_A \]

(a) Block diagram.

\[ \text{IMP} = (b(t+1) = \text{phase}_A t \implies a t \lor b t) \land (z(t+1) = \text{phase}_A t \implies b t \lor z t) \]

\[ \text{REQ} = (b(t+1) = \text{phase}_A t \implies a t \lor b t) \land (z(t+1) = \text{phase}_A t \implies a t \lor z t) \]

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

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

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

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

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

(1) Standard approach.

(b) Alternative approach.

3.1.1.3 Flip-Flops

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

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

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

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

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

At the clock level of abstraction we model the state of a flip-flop as the contents of its phase-B latch and
Latch L1  
Latch L2  

\[
\begin{align*}
\text{D} & \quad \text{D} \\
\text{Q} & \quad \text{Q} \\
\text{stateB} & \quad \text{stateA} \\
\text{d_in} & \quad \text{q_out} \\
\end{align*}
\]

(a) Functional block diagram.

\[
\begin{align*}
\text{I- DFF_SPEC d_in phase_A stateA stateB q_out} &= \\
\forall \ t: \text{time}. \\
(stateB \ (t+1) &= \neg (\text{phase_A} \ t) \Rightarrow d_{\text{in}} \ t \ \text{stateB} \ t) \ \land \\
(stateA \ (t+1) &= (\text{phase_A} \ t) \Rightarrow \text{stateB} \ t \ \text{stateA} \ t) \ \land \\
(q_{\text{out}} \ t &= \text{stateA} \ (t+1))
\end{align*}
\]

(b) HOL representation.

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

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

### 3.1.1.4 Counters

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

The inputs \(ld_{\text{in}}\) and \(up_{\text{in}}\) control the operation of this counter. If \(ld_{\text{in}}\) is active then the input \(d_{\text{in}}\) is loaded into the counter, otherwise the current value, incremented or nonincremented according to the \(up_{\text{in}}\) input, is reloaded. The input \(up_{\text{in}}\) also controls the value output by the counter.

### 3.1.1.5 CTR Datapath Block

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

Figure 3.5 shows the functionality of the CTR datapath block. It behaves much like the counter of the previous section, but with additional features such as provisions for carry-in and carry-out and multiple output ports.
Figure 3.4: Functional Block Diagram of a Counter.

Figure 3.5: Functional Block Diagram of the CTR Datapath Block.
Of the 11 latches in this model, the one best representing the counter value is $L_4$, holding the value $ctr$. Latch $L_2$ contains the load-input, controlling whether a new value is loaded or the updated counter value is reloaded. Latches $L_1$ and $L_8$ hold these two values, respectively. Latches $L_5$ and $L_6$ hold values controlling the incrementer itself. For the top half of the 64-bit counters, $L_6$ contains the carry-in from the lower half. Latch $L_7$ holds the carry-out from the counter. Latches $L_9$ and $L_{10}$ implement a flip-flop holding the updated counter value for possible output. The two latches $L_3$ and $L_{11}$ control the writing of latch values onto $Bus_A$, from the input side and output side, respectively.

3.1.1.6 ICR Datapath Block

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

Figure 3.6 shows a functional block diagram of this block. The true ICR value is located in the flip-flop implemented by latches $L_4$ and $L_5$. The flip-flop implemented by $L_1$ and $L_2$ holds the ICR value fed back using $Bus_A$. Latch $L_3$ holds a mask-adjustment value that resets or sets individual mask bits according to the value of input $icr_select$. Latch $L_6$ controls the writing of values onto $Bus_A$ either as part of an ICR read by an external processor or the feedback mentioned above.

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

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

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

3.1.1.8 SR Datapath Block

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

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

3.1.1.9 Finite-State Machines

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

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

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

Figure 3.9: Functional Block Diagram for Finite-State Machines.
Creating superblocks also has the beneficial side effect of simplifying our description of the five PIU ports. Even so, the complexity of the resulting specification remains formidable and a fully-detailed pictorial description of the PIU structure is beyond the scope of this report. The HOL descriptions in Appendix B should be considered the gate-level specification for the five PIU ports; the descriptions in this section are intended only to provide insight so that the HOL is more easily understood. Although considerable care has gone into the construction of these descriptions, they are not complete and contain minor inaccuracies as well.

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

3.1.2.1 P_Port Structure

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

![Figure 3.10: P_Port Top-Level Block Diagram.](image)

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

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

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

The Ctr.Logic block keeps track of the number of words remaining in the current transaction so that the slave port can be notified when the last word is being accessed.
Figure 3.11: Block Diagram of P_Port Datapath.

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

3.1.2.2 M_Port Structure

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

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

The Read_Latches block stores the 32-bit decoded data word read from memory. The Mux_Out_Logic block selects bytes from this stored value or else the word currently on the I_Bus for writing onto the M_Bus. The stored bytes are written back as part of a read-modify-write implementation of byte-write operations.
Figure 3.12: Block Diagram of the P_Port Controller.

Figure 3.13: M_Port Top-Level Block Diagram.
The M_Port controller is shown in Figure 3.15. The left side of the figure is the I_Bus interface. The SE_Logic block determines whether a memory access is to SRAM memory or to EEPROM memory, based on the memory address. It drives the appropriate chip-select signal based on this determination.

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

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

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

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

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

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

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

The \textit{Reg_Sel_Ctr} block contains a 4-bit counter holding the register number for the current access. The \textit{R_srdy_del} latch value is used to increment the counter on multiword accesses. The \textit{Reg_File_Ctl} block

Figure 3.15: Block Diagram of the M_Port Controller.
decodes the register address to create most of the control signals needed by the register file.

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

The latches $R_{c0l\_cout}$ and $R_{c23\_cout}$ hold the carry-out values of the two counters. The $Ctr\_Int\_Logic$ blocks use this information and several bits of the GCR to determine whether the timer interrupts should be enabled or not. The two interrupt outputs, $Int1$ and $Int2$, are active-high signals sent to the local processor.
Figure 3.18: Block Diagram of the Timer Interrupt Block.

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

Figure 3.19: Block Diagram of the Register Interrupt Block.

3.1.2.4 C_Port Structure

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

Figure 3.21 shows the C_Port datapath block diagram. The right side of the figure shows the interface
Figure 3.20: C_Port Top-Level Block Diagram.

Figure 3.21: Block Diagram of the C_Port Datapath.
between the I_Bus and the C_Bus. The Parity_Decode.Logic block decodes the 18-bit parity-encoded data received from the C_Bus data lines. It outputs 16-bit data and a single-bit error-detection flag.

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

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

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

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

Figure 3.22: Block Diagram of the C_Port Controller (Part A).
Figure 3.23 shows part (B) of the C_Port controller. The DP_Ctls PLA block converts output signals from both the master and slave state machines of part (A) into control signals for the datapath. The latches at the output of this block, as well as the Cout_ILeLogic block, provide further processing for the datapath, primarily to control the enabling of the datapath latches.

The CBss_Out_Loc block and the CBms_Out_Loc block determine the master-status and slave-status, respectively, for C_Bus transactions. The Srty_In_Loc block decodes the slave-status input from the C_Bus to determine whether the slave is ready for the next transaction.

The Rdy_Loc block, the ISrty_Out_Loc block, and intervening latches implement the generation and transmission of the I_srty_ signal to the I_Bus. The Iad_En_Loc block controls the enabling for address and data transmissions over the I_Bus.

The Pe_Cnt_Loc block controls the enabling of parity-error counting within the datapath.
3.1.2.5 SU_Cont Structure

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

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

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

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

Figure 3.24: Block Diagram of the Startup Controller PIU-Port Interface.
Figure 3.25 shows the SU_Cont blocks that interact mainly with the local processor. The Cpu_OK block and the Fail_In block together control the loading of four latches holding failure-status information. The Cpu_OK block uses the S_Soft_Cnt signal just discussed and the Failure signals from the local processors. The latch outputs are transmitted to the R_Port where they are stored in the Status Register.

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

3.2 Port Phase-Level Behavior

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

Figure 3.25: Block Diagram of the Startup Controller CPU Interface.
Appendix C contains the HOL phase-level specification. The ports are presented in the order: P Port, M Port, R Port, C Port, and SU_Cont, in Sections C.1 through C.5, respectively. Within each section the next-state function for phase A is presented first, followed by the output function for phase A, and the next-state and output functions for phase B.

3.3 Port Clock-Level Behavior

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

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

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

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

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

3.4 PIU Port-Level Structure

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

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

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

4.1 Introduction.

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

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

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

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

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

• The feature of a bus controller that causes the greatest difficulty in using an interpreter model to describe it is its concurrency—a bus controller does many things at once. For example, most bus controllers contain timers that, in conjunction with an on-board interrupt controller, can interrupt the CPU. These timers operate concurrently with other portions of the bus controller, such as memory and network operations.

• A typical top-level specification of the PIU might include the memory subsystem because this corresponds to the CPU’s view of the PIU (see the next section for a more complete discussion of this). This shared state between the PIU and other devices makes description using an interpreter model difficult.

• The outputs of the PIU do not correspond on a one-to-one basis with the inputs; there is a many-to-one relationship between the outputs and inputs. The interpreter model assumes that the output at a particular time is described by a function on the current state and environment. The PIU may make several outputs in sequence because of a single input request (a block memory read request is a good example).

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

• The notation and semantics should be amenable to embedding and automation in an automatic theorem prover such as HOL.

• The model and notation should be sufficiently general to allow a large number of interesting devices to be described.

• The model and notation should be sufficiently defined to allow a rich set of theorems to be proven about it in isolation of any particular application.
4.2 Abstract Views

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

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

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

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

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

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

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

The problem with the views presented in Figures 4.1–4.3 is that the abstractions include the behavior of the CPU, network, and memory. Our goal is to specify the behavior of the PIU independent of the devices to which it is connected. Each of these views can be thought of as a specification of the abstract interface to one portion of the PIU. As Figure 4.4 shows, we can superimpose the specifications on one another. The union of the PNM, PCN, and PMC specify the behavior of the entire unit. Their intersection, denoted by the shaded area, is meant to represent the behavior that is specific to the PIU.
While we feel that this is a good way to think about the behavior of the PIU in abstract, we are not convinced that it is an appropriate method of specifying the behavior of the PIU. Before such a decision can be made, we will need to do further work. Primarily, we would like to attempt to model the specification of a small device in this way and evaluate the specification for readability and ease of use in verification.

### 4.3 Representing Transaction Systems

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

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

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

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

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

There are a number of candidate notations:

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

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

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

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

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

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

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

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

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

4. We could use a formal model of a coordination language such as LINDA [But91] to model the actions of the system. In this model, the PIU, CPU, memory, and network are modeled as communicating in a
common area called tuple space. Figure 4.5 shows how this would look. In this model, the PIU writes to and reads from tuple space along with the other devices in the system. We can think of tuple space as an abstract model of the bus.

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

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

4.4 Preliminary Transaction Model Design

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

4.4.1 The Transaction Model

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

![Figure 4.5: Modeling the Buses in a Computer System using Tuple Space.](image-url)
4.4.1.1 Ports

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

4.4.1.2 State

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

4.4.1.3 Transactions

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

4.4.1.4 Operation

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

4.4.2 Development Plan and Comments

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

1. Build a function program in ML of the behavior of the PIU based on the model present above. The program will allow us to exercise the model and determine where there are problems. We chose ML since it is close to the syntax of HOL and will be readily converted into HOL when we are satisfied with it.

2. The program built in the previous step will be specific to the PIU. Our plan is to generalize that program into an abstract model of transaction systems. We plan to use the results of the experiments in the previous step to guide a formalization of the general model in HOL. Careful design of the abstraction in the program will make this task easier. Provided that the results of the experiments yield favorable results, we do not anticipate formalization to be a large effort.

3. After the model has been formalized, we will need to use it to assess its utility and determine what lemmas need to be proven in the abstract theory to enable effective reasoning in the concrete model. There is no way to determine what these theories will be until the model is used the first time.

4. As the model is used, there will undoubtedly be refinements and extensions. Our experience with the generic interpreter theory has shown that refining and extending abstract theories is not an arduous task and anticipate that the same will be true of the new model.

There are several areas that may lead to difficulties:

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

- The state is shared and thus may be updated by several ports at once (provided that such updating does not cause interference). We hope that partial specifications of the changes, represented by predicates rather than functions, will solve this problem.

- We have ignored the start-up operation of the PIU in our model. We do not believe that this is a problem since the start-up portion of the chip operates in sequence with the rest of the PIU components. We can model the start-up portion using an interpreter or transaction system (whichever is more appropriate) and choose the behavior of the start-up device or the PIU device depending on the current state.

- The PIU has a number of on-board clocks that serve as interrupt timers. We hope that they can be modeled using the concepts presented in this chapter by looking at the external clock port as another input port with its own set of transactions. One of those transactions will trigger interrupts when the state is correct.

4.5 Conclusions

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

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

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

- The preliminary transaction model must be refined as presented in Section 4.4. The models need to be tested on the PIU design for utility. Furthermore, the model needs to be formalized in HOL.

- Further work must be done on the composition of our abstract-view approach to behavior. We plan a further review of the literature for applicable work and a small test study involving a small device with a simple semantics, but more than one interface, to determine whether composing the abstract behaviors of the interface is sufficient to represent behavior.

- We intend to pursue the formalization of the LINDA coordination language since it seems a likely candidate model for composing the specification of the PIU with the specifications of the CPU, memory, and network. This composition would be used to implement a more abstract view of the system. This work does not have consequences for the top-level specification of the PIU itself but may be important for future compositions.
5 Towards an Integrated Simulation/Verification Environment

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

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

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

• Engineers working on the project are more comfortable with M descriptions than they are with the logic of HOL. This is probably because of the similarity of M to imperative programming languages in which most engineers are schooled.

• M descriptions can be executed. This allows the specifications to be animated, providing a form of simulation. Engineers can observe the operation of the specification in an effort to judge its correctness.

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

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

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

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

5.1 New Datatypes in HOL

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

5.1.1 Arrays

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

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

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

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

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

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

\section*{5.1.2 N-Bit Words}

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

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

There are two major functions for interpreting n-bit words: \textit{VAL} and \textit{WORDN}. \textit{VAL} returns the numeric value of an n-bit word. \textit{WORDN} returns the n-bit word representing a given number.

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

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

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

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

\section*{5.2 An Example in M}

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

```c
#define V1 1
#define V2 2

MODULE test () {
   /* State variables:*/
   MEMORY LOGIC new_A, A;
   MEMORY LOGIC new_B, B;
   MEMORY LOGIC new_C[32], C[32];

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

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

   INITIALIZE {

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

            I_X[31] = new_A
               ? Clock 
               : Rst;
            I_X[30:29] = new_C[1:0];
            I_X[28:0] = new_B
               ? new_C[28:0]
               : I_X[28:0];
            break;
         default:
            PRINT ('\nILLEGAL');
            break;
      }
   }
}
```
5.3 An Example in HOL

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2. The specification hierarchy was outlined in Section 3, but this task did not include the completion of the specification. In particular, the PIU top-level specification remains to be written.
In addition to the work that must be completed to finish the specification, there are a number of open questions that have a direct bearing on how this work is used:

1. The proofs of correspondence between levels in the specification hierarchy should be completed. The specification process itself is useful because it gives designers an abstract view of the device and aids understanding. The detailed examination entailed in the specification is useful for finding errors. However, the primary benefit of a formal specification is that it is amenable to analysis.

2. If we intend to use the top-level specification along with specifications of other devices in the PMM, such as the CPU and memory, to write a specification of the PMM, a model of composition must be developed. Section 4 recommended a formalization of LINDA as that model, but no work has been done to explore the feasibility or utility of this method.

3. The translation between M and HOL is being done in a prototype system written in AWK. A more formal approach, with more confidence in its correctness, would be to embed M in HOL. This would involve defining the syntax of M (or a reasonable subset) in HOL and then defining a formal semantics of M for use in the translation. Because the translation would be done by the verification system itself, we could have increased confidence that the HOL model corresponded to the M model.
7 References


Appendix A  ML Source for Component Specifications.

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

File:         gates_def.ml

Author:      (c) D.A. Fura 1992

Date:        31 March 1992

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

```ml
semit 'rm gates_def.th'';

new_theory 'gates_def'';

map new_parent ['aux_def'];

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

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

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

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

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

58
('OR3_SPEC',
"l a b c z .
OR3_SPEC a b c z =
  (t : time . z t = a t ∨ b t ∨ c t)"
);

let NAND2_SPEC = new_definition
('NAND2_SPEC',
"l a b z .
NAND2_SPEC a b z =
  (t : time . z t = -(a t ∧ b t))"
);

let NAND3_SPEC = new_definition
('NAND3_SPEC',
"l a b c z .
NAND3_SPEC a b c z =
  (t : time . z t = -(a t ∧ b t ∧ c t))"
);

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

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

close_theory();

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

File: latches_def.ml

Author: (c) D.A. Fura 1992

Date: 31 March 1992

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

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

system 'rm latches_def.th';;

new_theory 'latches_def';;

map new_parent ['aux_def'];;
let DLAT_SPEC = new_definition
("DLAT_SPEC",
"! (din:time->bool) clk state qout .
DLAT_SPEC din clk state qout =
  ! t:time .
  (state (t+1) = (clk t) => din t | state t) /
  (qout t = state (t+1))")
);

let DSLAT_SPEC = new_definition
("DSLAT_SPEC",
"! (din:time->bool) set clk state qout .
DSLAT_SPEC din set clk state qout =
  ! t:time .
  (state (t+1) = (clk t) => ((set t) => T | din t | state t) /
  (qout t = state (t+1))")
);

let DRLAT_SPEC = new_definition
("DRLAT_SPEC",
"! (din:time->bool) rst clk state qout .
DRLAT_SPEC din rst clk state qout =
  ! t:time .
  (state (t+1) = (clk t) => ((rst t) => F | din t | state t) /
  (qout t = state (t+1))")
);

let DSRLAT_SPEC = new_definition
("DSRLAT_SPEC",
"! (din:time->bool) set rst clk state qout .
DSRLAT_SPEC din set rst clk state qout =
  ! t:time .
  (state (t+1) = (clk t) => ((set t) => ((set t) => T | din t | state t) /
  (qout t = state (t+1))")
  
state t) /

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

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

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

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

One-bit D-latch, with set, with reset, no enable.
(qout t = state (t+1))

;]

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

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

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

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

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

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

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

let DSRELAT_SPEC = new_definition
('DSRELAT_SPEC',
"| (dm:time->bool) set rst en clk state qout .
DSRELAT_SPEC din set rst en clk state qout =
| t:time .
   (state (t+1) = (clk t \ en 0 =) => ((set t) => T \ din t \ state t) \ (-set t \ rst t) => F \ (-set t \ rst t) => T \ din t \ state t) \ (qout t = state (t+1))"
);

61
let DLA\textsubscript{Tn}\_SPEC = new\_definition
('DLATn\_SPEC',

"I (din:time\rightarrow\text{wordn}) clk state qout .

DLATn\_SPEC din clk state qout =

l t:time .

(state (t+1) = (clk t) \Rightarrow din t \land state t) \land

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

);;

close\_theory();;

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

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

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

let DFF\_SPEC = new\_definition
('DFF\_SPEC',

"I (din:time\rightarrow\text{bool}) clk state0 state1 qout .

DFF\_SPEC din clk state0 state1 qout =

l t:time .

(state0 (t+1) = (\neg clk t) \Rightarrow din t \land state0 t) \land

(state1 (t+1) = (clk t) \Rightarrow state0 t \land state1 t) \land

(qout t = state1 (t+1))";

);;

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

One-bit flip-flop, no set, no reset, no enable.
let DRFF_SPEC = new_definition
  ('DRFF_SPEC',
  "I (din:time->bool) rst clk state0 state1 qout .
  DRFF_SPEC din rst clk state0 state1 qout =
  (t:time . (state0 (t+1) = (~clk t) => (rst t => F | din t) | state0 t) ∧
   (state1 (t+1) = (clk t) => state0 t | state1 t) ∧
   (qout t = state1 (t+1))))");

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

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

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

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

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

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

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

Multi-bit flip-flop, no set, no reset, with enable.
let DEFFn_SPEC = new_definition
(\'DEFFn_SPEC\',
  "1 (din:time->wordn) en clk state0 state1 qout .
  DEFFn_SPEC din en clk state0 state1 qout =
  (t:time . (state0 (t+1) = (\(-clk\) t => din t \& state0 t) \&
  (state1 (t+1) = (clk t \& en t) => state0 t \& state1 t) \&
  (qout t = state1 (t+1)))")
);

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

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

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

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

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

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

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

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

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

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

```
% system 'rm counters_def.th';;
new_theory 'counters_def';;
map new_parent ['aux_def';'array_def';'wordn_def'];;

Up-counter, no reset.

let UPCNT_SPEC = new_definition ('UPCNT_SPEC', "1 size (din:time->wordn) ld up clk state0 statel qout zero.
UPCNT_SPEC size din ld up clk state0 statel qout zero =
 lt:time.

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

Down-counter, no reset.

let DOWNCNT_SPEC = new_definition ('DOWNCNT_SPEC', "1 size (din:time->wordn) ld down clk state0 statel qout zero.
DOWNCNT_SPEC size din ld down clk state0 statel qout zero =
 lt:time.

(state0 (t+1) = (-clk t) =>
 (ld t) => din t !
 (down t) => DECN size (state1 t) ! state1 t) !
 state0 t) /
 (state1 (t+1) = (clk t) => state0 t ! state1 t) /
```

65
(quot t = (down t) => DECN size (state1 (t+1)) | state1 (t+1)) \n(0 t = (down t) => (DECN size (state1 (t+1)) = WORDN 0) | (state1 (t+1) = WORDN 0))
);
%
-- Up-counter, with reset.

let UPRCNT\_SPEC = new\_definition
("UPRCNT\_SPEC",
"| size (din:time->wordn) ld up rst clk state0 state1 qout zero .
UPRCNT\_SPEC size din ld up rst clk state0 state1 qout zero =
l:t:time .

(state0 (t+1) = (-clk t) =>
 (ld t) => din t |
 (up t) => INCN size (state1 t) | state1 t | state0 t) \n
(state1 (t+1) = (clk t) =>
 (rst t) => WORDN 0 | state0 t | state1 t)

(quot t = (up t) => INCN size (state1 (t+1)) | state1 (t+1)) \n
(zero t = (up t) => (INCN size (state1 (t+1)) = WORDN 0) | (state1 (t+1) = WORDN 0))"
);
%
-- Down-counter, with reset.

let DOWNRCNT\_SPEC = new\_definition
("DOWNRCNT\_SPEC",
"| size (din:time->wordn) ld down rst clk state0 state1 qout zero .
DOWNRCNT\_SPEC size din ld down rst clk state0 state1 qout zero =
l:t:time .

(state0 (t+1) = (-clk t) =>
 (ld t) => din t |
 (down t) => DECN size (state1 t) | state1 t | state0 t) \n
(state1 (t+1) = (clk t) =>
 (rst t) => WORDN 0 | state0 t | state1 t)

(quot t = (down t) => DECN size (state1 (t+1)) | state1 (t+1)) \n
(zero t = (down t) => (DECN size (state1 (t+1)) = WORDN 0) | (state1 (t+1) = WORDN 0))"
);

close\_theory();;
%

File: datapaths\_def.ml

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

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

```
% system 'rm datapaths_def.th';
% new_theory 'datapaths_def';
% map loadf ['abstract'];
% map new_parent ['aux_def';'array_def';'wordn_def'];
let rep_ty = abstract_type 'aux_def' 'Andn';

% Counter block used to build timers.

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

DP_CTR_SPEC clkA clkB busB_in cir_wr c_ld cir_rd ce cin csor LD cor_rd
  r_ctr_in r_ctr_mux sel r_ctr irden r_ctr r_ctr ce r_ctr cin r_ctr_cry
  r_ctr_new r_ctr_outA r_ctr_out r_ctr_orden busA_out1 busA_out2 c_out =

| time .

((clkA t) =>
  ((r_ctr_in (t+1) = r_ctr_in t) \n
  (r_ctr_mux_sel (t+1) = r_ctr_mux_sel t) \n
  (r_ctr_irden (t+1) = r_ctr_irden t) \n
  (r_ctr (t+1) = (r_ctr_mux sel t) => r_ctr_in t \ r_ctr_new t) \n
  (r_ctr_ce (t+1) = ce t) \n
  (r_ctr cin (t+1) = cin t) \n
  (r_ctr_cry (t+1) = r_ctr_cry t) \n
  (r_ctr_new (t+1) = r_ctr_new t) \n
  (r_ctr_out (t+1) = r_ctr_out t) \n
  (r_ctr_orden (t+1) = r_ctr_orden t))) \n
((clkB t) =>
  ((r_ctr_in (t+1) = (cir_wr t) => busB_in t \ r_ctr_in t) \n
  (r_ctr_mux_sel (t+1) = c ld t) \n
  (r_ctr_irden (t+1) = cir_rd t) \n
  (r_ctr (t+1) = r_ctr t) \n
  (r_ctr ce (t+1) = r_ctr ce t) \n
  (r_ctr cin (t+1) = r_ctr cin t) \n
  (r_ctr_cry (t+1) = (r_ctr_cry t) \ (r_ctr cin t) \ ONES 31 (r_ctr t)) \n
  (r_ctr_new (t+1) = ((r_ctr ce t) \ (r_ctr cin t)) => INCN 31 (r_ctr t) \ r_ctr t) \n
  (r_ctr_outA (t+1) = r_ctr_outA t) \n
```
Control register used to build General Control Register (GCR) and Communication Control Register (CCR).

let DP_I CR_SPEC = new_definition (‘DP_I CR_SPEC’, "! (rep::rep_ty) clkA clkB (busA_in in time->wordn) busB_in icr_wr_feedback icr_wr icr_select icr_ld icr_rd 
  r_icr oldA r_icr_old r_icr_mask r_icrA r_icr r_icr rden 
  busA_out icr_out . 
  DP_I CR_SPEC rep clkA clkB busA_in busB_in icr_wr_feedback icr_wr icr_select icr_ld icr_rd 
  r_icr oldA r_icr_old r_icr_mask r_icrA r_icr r_icr rden 
  busA_out icr_out = ![time]

  (((clkA t) =>> 
    (r_icr oldA (t+1) = busA_in t) ∨ 
    (r_icr old (t+1) = r_icr_old t) ∨ 
    (r_icr mask (t+1) = r_icr_mask t) ∨ 
    (r icrA (t+1) = (icr select t) => Andn rep (r icr old t, r icr mask t) 
      Orn rep (r icr old t, r icr mask t)) ∨ 
    (r icr (t+1) = r icr t) ∨ 
    (r icr rden (t+1) = r icr rden t)) ∨ 
  (((clkB t) =>> 
    (r icr oldA (t+1) = r icr oldA t) ∨ 
    (r icr old (t+1) = (icr wr_feedback t) => r icr oldA t r icr old t) ∨ 
    (r icr mask (t+1) = (icr wr t) => busB_in t r icr mask t) ∨ 
    (r icrA (t+1) = r icrA t) ∨ 
    (r icr (t+1) = (icr ld t) => r icrA t r icr t) ∨ 
    (r icr rden (t+1) = icr rd t)) ∨ 
  (busA_out t = ((r icr rden (t+1) ∨ (clkA t)) => r icr (t+1) | ARBN)) ∨ 
  (icr out t = r icr (t+1)))")

);

Control register used to build General Control Register (GCR) and Communication Control Register (CCR).
((clkA t) \implies
  (r_cr (t+1) = r_cr t) \land
  (r_cr_rden (t+1) = r_cr_rden t)) \land
((clkB t) \implies
  (r_cr (t+1) = (cr_wr t) \implies busB_in t \land r_cr t) \land
  (r_cr_rden (t+1) = cr_rd t)) \land
((busA_out t = ((r_cr_rden (t+1)) \land (clkA t)) \implies r_cr (t+1) \land ARBN) \land
  (cr_out t = r_cr (t+1)))”
);

%....................................................................................................................

let DP_SR_SPEC = new_definition
  ('DP_SR_SPEC',
   "! clkA clkB (inp:time->wordn) sror ld sr rd
    r sr r sr rden
    busA out .
    DP_SR_SPEC clkA clkB inp sror ld sr rd
    r sr r sr rden
    busA out =
    !t:time .
    ((clkA t) \implies
      (r sr (t+1) = r sr t) \land
      (r sr_rden (t+1) = r sr_rden t)) \land
      (clkB t) \implies
      (r sr (t+1) = (sror ld t) \implies inp t \land r sr t) \land
      (r sr_rden (t+1) = sr_rd t)) \land
    (busA out t = ((r sr_rden (t+1)) \land (clkA t)) \implies r sr (t+1) \land ARBN")
  );;

close_theory();;

%....................................................................................................................

File: buses def.ml
Author: (c) D.A. Fura 1992
Date: 31 March 1992

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

%....................................................................................................................

system 'rm buses_def.th';;
new_theory 'buses_def';;
map new_parent ['aux def'];;
 Specification for a conflict-free bus.

let Bus CF 12 SPEC = new_definition
('Bus CF 12 SPEC',
"1 inE1 inE2 inE3 inE4 inE5 inE6 inE7 inE8 inE9 inE10 inE11 inE12,
Bus CF 12 SPEC inE1 inE2 inE3 inE4 inE5 inE6 inE7 inE8 inE9 inE10 inE11 inE12 = 
lt time .

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

 Specification for a 12-input bus component.

let Bus 12_1 SPEC = new_definition
('Bus 12_1 SPEC',
"1 (inD1:time->*) roD2 inD3 inD4 inD5 inD6 inD7 inD8 inD9 inD10 inD11 inD12
Bus 12_1 SPEC inD1 inD2 inD3 inD4 inD5 inD6 inD7 inD8 inD9 inD10 inD11 inD12 = 
lt time .

(Bus CF 12 SPEC inE1 inE2 inE3 inE4 inE5 inE6 inE7 inE8 inE9 inE10 inE11 inE12) =>

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

70
let BuslA_SPEC = new_definition
('BuslA_SPEC',
"| (in_A:time->*) out_A out_B .
BuslA_SPEC in_A out_A out_B = 
!t:time .
(out_A t = in_A t) ∧
(out_B t = in_A t)"
);

let BuslB_SPEC = new_definition
('BuslB_SPEC',
"| (in_B:time->*) out_A out_B .
BuslB_SPEC in_B out_A out_B = 
!t:time .
(out_A t = in_B (t-1)) ∧
(out_B t = in_B t)"
);

close_theory();;

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

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

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

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

system 'rm aux_def.th';;
new_theory 'aux_def';;
loadf 'abstract';;
new_type_abbrev('time', ":num");
new_type_abbrev('wordn', ":(num->bool)");;
let sc_out_ty = "((wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;

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

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

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

make_inst_thms abs_rep;;

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

close_theory();;

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

File: array_def.ml

Author: (c) P. J. Windley 1992

Description:

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

Modification History:

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

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

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

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

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

ALTER_EQUAL is similar to the EL_SET_EL lemma for lists.

let ALTER_EQUAL = prove_thm ('ALTER_EQUAL', "! x n (f:*->**). (ALTER f n x) n = x", REPEAT GEN_TAC THEN REWRITE_TAC [ALTER_DEF] THEN BETA_TAC THEN REWRITE_TAC []);
ALTER\_NONEQUAL is similar to NOT\_EL\_SET\_EL for lists.

let ALTER\_NONEQUAL = prove_thm
   'ALTER\_NONEQUAL',
   "\( \forall n m (f:*->**) x .
   \neg(n = m) \implies
   (f n \neq (ALTER f m x))\)",
   REPEAT GEN_TAC
   THEN REWRITE_TAC [ALTER\_THM]
   THEN STRIP_TAC
   THEN ASM\_REWRITE\_TAC []
;;

ALTER\_COMMUTES is similar to SET\_EL\_SET\_EL for lists.

let ALTER\_COMMUTE = prove_thm
   ('ALTER\_COMMUTE',
   "\( \forall (d1:*:d2) (f:*->**) (x:**) y .
   \neg(d1 = d2) \implies
   ((ALTER (ALTER f d2 x) d1 y) \neq
   (ALTER (ALTER f d1 y) d2 x))\)",
   REPEAT GEN_TAC
   THEN CONV_TAC (ONCE_DEPTH_CONV FUN\_EQ\_CONV)
   THEN REWRITE_TAC [ALTER\_THM]
   THEN STRIP_TAC
   THEN GEN_TAC
   THEN REPEAT COND\_CASES\_TAC
   THEN ASM\_REWRITE\_TAC []
   THEN UNDISCH_TAC "\neg((d1:*) = d2)"
   THEN ASSUM\_LIST (Xthl. REWRITE_TAC (map SYM\_RULE thl))
   );

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

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

let SUBARRAY\_DEF = new_definition
   ('SUBARRAY\_DEF',
   "\( \forall n m (f:num->*) .
   SUBARRAY f (m,n) = \forall x. ((x+n) \leq m) \implies f(x+n) \uparrow ARB\)"
   );
let SUBARRAY_THM = prove_thm
('SUBARRAY_THM',
"l n m (f: num->*).
SUBARRAY f (m,n) x = ((x+n) <= m) => f(x+n) I ARB",
REPEAT GEN_TAC
THEN REWRITE_TAC [SUBARRAY_DEF]
THEN BETA_TAC
THEN REFL_TAC
);

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

MALTER alters multiple values in an array.

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

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

let MALTER_SUBARRAY_IDENT = prove_thm
('MALTER_SUBARRAY_IDENT',
"l n m (f: num->*).
MALTER f (m,n) (SUBARRAY f (m,n)) = f"
);
let MALTER_SUBARRAY_SUBSCRIPTS = prove_thm
('MALTER_SUBARRAY_SUBSCRIPT',
"in m x (f:num->*) g .
MALTER f (m,n) (SUBARRAY g (m,n)) x =
(n <= x & x <= m) => g x f x",
REPEAT GEN_TAC
THEN CONV_TAC (ONCE_DEPTH_CONV FUN_EQ_CONV)
THEN REWRITE_TAC [MALERTHLM;SUBARRAYTHLM]
THEN REPEAT COND_CASES_TAC
THEN ASM_REWRITE_TAC []
THEN IMP_RES_TAC SUBADD
THEN TRY (UNDISCH_TAC "-((x - n) + n) <= m')
THEN ASM_REWRITE_TAC []
);

close_theory();

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

File: wordn_def.ml

Description:

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

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

Modification History:

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

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

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

---------------------------% Removed 13MAR92. [DAF]

let add_root s = '/users/staff/windley/hoFLibrary/' ^ s;;

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

77
let bv = new_definition
   ('bv',
   "! (b:bool).
   bv b = (b) => 1 1 0"
   );;

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

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

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

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

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

let SETN = new_definition
('SETN',
"\(1 (x:num). \text{SETN} x = \forall (n:num). (n <= x) \Rightarrow T \land \text{ARB}"");

% Equivalent to "WORDN 0" but perhaps more convenient %
let RSTN = new_definition
('RSTN',
"\(1 (x:num). \text{RSTN} x = \forall (n:num). (n <= x) \Rightarrow F \land \text{ARB}"");

let GNDN = new_definition
('GNDN',
"\(1 (x:num) (t:time). \text{GNDN} x t = \forall n (num). (n <= x) \Rightarrow F \land \text{ARB}"");

let NOTN = new_definition
('NOTN',
"\(1 (x:num) (f:wordn). \text{NOTN} x f = \forall n (num). (n <= x) \Rightarrow -f n \land \text{ARB}"");

let INCN = new_definition
('INCN',
"\(1 n f . \text{INCN} n f = (\text{ONES} n f) \Rightarrow \text{RSTN} n \land \text{WORDN} ((\text{VAL} n f) + 1)"");

let DECN = new_definition
('DECN',
"\(1 n f . \text{DECN} n f = (\text{ZEROS} n f) \Rightarrow \text{SETN} n \land \text{WORDN} ((\text{VAL} n f) - 1)"");

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

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

% Removed theorems for now 13MAR92. [DAF]

close_theory();
Appendix B  ML Source for the Gate-Level Specification of the PIU Ports.

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

B.1 P Port Specification

File: p_block.ml

Author: (c) D.A. Fura 1992

Date: 31 March 1992

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

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

system 'rm p_block.th';

new_theory 'p_block';

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

let p_state_ty = ":(p fsm_ty#bool#bool#wordn#bool#wordn#num#bool#bool#pfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool)"

let p_state = "((P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P_dest1, P_be_, P_wr, P_be_n, P_sizeA, P_loadA, P_downA, P_fsm_rst, P_fsm_mrqt, P_fsmmrq, P_fsm sack, P FSM_cgmt, P FSM_crqL, P_fsm_hold, P_fsm_lock, P_qrt, P_size, P_load, P_down, P_lock_, P_lock_inh, P_male_, P_rale)"

let p_env_ty = ":(bool#bool#bool#wordn#bool#bool#bool#bool#bool#bool)"

let p_env = "((ClkA, ClkB, RsL L_ad_m, Lads.., L_den_, L_be_, L_wr, L_lock_, Lady_)"

let p_out_ty = ":(wordn#bool#wordn#wordn#bool#bool#bool#bool#bool#bool#bool)"

let p_out = "((L_ad_out, L_ready, L_ad_data_out, L_ad_addr_out, L_be, L_rale, L_male, L_crqtL, L_cale_, L_mrdy_, L_last, L_hlda, L_lock)"

let Data_Latches_SPEC = new_definition

"Data_Latches_SPEC"

"!(clkA clkB (lad_in:time->(num->bool)) (be_in:time->(num->bool)) (lwr_in:time->bool) en_in be_sel wr_data addr dest1 be wr be_n"
data_out addr_out be_out .

Data_Latches_SPEC clkA clkB lad_in lbe_in lwr_in en_in be_sel
  wr_data addr destl be wr be_n
  data_out addr_out be_out =

\( ((\text{clkA} t) \implies \)
  \begin{align*}
    ((\text{wr\_data} (t+1) = \text{lad\_in} t) & \land \\
    (\text{addr} (t+1) = (\text{en\_in} t) \implies (\text{lad\_in} t) \land (\text{addr} t)) & \land \\
    (\text{destl} (t+1) = (\text{en\_in} t) \implies (\text{ELEMENT} (\text{lad\_in} t) (31)) \land (\text{destl} t)) & \land \\
    (\text{be} (t+1) = (\text{en\_in} t) \implies (\text{lbe\_in} t) \land (\text{be} t)) & \land \\
    (\text{wr} (t+1) = (\text{en\_in} t) \implies (\text{lwr\_in} t) \land (\text{wr} t)) & \land \\
    (\text{be\_n} (t+1) = (\text{lbe\_in} t)) & \land \\
  \end{align*}

\( ((\text{clkB} t) \implies \)
  \begin{align*}
    ((\text{wr\_data} (t+1) = \text{wr\_data} t) & \land \\
    (\text{addr} (t+1) = \text{addr} t) & \land \\
    (\text{destl} (t+1) = \text{destl} t) & \land \\
    (\text{be} (t+1) = \text{be} t) & \land \\
    (\text{wr} (t+1) = \text{wr} t) & \land \\
    (\text{be\_n} (t+1) = \text{be\_n} t)) & \land \\
  \end{align*}

\( ((\text{data\_out} t = \text{wr\_data} (t+1)) & \land \\
    (\text{let} \text{od1} = \text{MALTER} (\text{addr\_out} t) (31, 27) \text{ in} \\
    \text{let} \text{od2} = \text{MALTER} \text{od1} (26) \text{ F in} \\
    \text{let} \text{od3} = \text{MALTER} \text{od2} (25, 24) \text{ in} \\
    \text{let} \text{od4} = \text{MALTER} \text{od3} (23, 0) \text{ in} \\
    (\text{addr\_out} t = (\text{addr\_out} (t+1))) & \land \\
    (\text{be\_out} t = (\text{be\_sel} t) \implies (\text{be} (t+1)) \land (\text{be\_n} (t+1))))\)

\( (l_{time} .
  \begin{align*}
    (\text{req\_inS} t = -(l_{ads} t) & \land (l_{den} t)) \land \\
    (\text{req\_inR} t = \text{reset\_req} t) \land \\
    (\text{req\_inE} t = (\text{req\_inS} t) \lor (\text{req\_inR} t))
  \end{align*}

\)

\( (l_{time} .
  \begin{align*}
    (p_{size} p_{sizeA} p_{load} p_{loadA} p_{down} p_{downA})
  \end{align*}

\)

% Input logic for P\_rqt latch.

% Input logic for P\_size counter.
((clkA t) =>
((p_sizeA (t+1) = p_size t) \land
(p_loadA (t+1) = p_load t) \land
(p_downA (t+1) = p_down t) \land
(p_size (t+1) = p_size t) \land
(p_load (t+1) = p_load t) \land
(p_down (t+1) = p_down t))) \land
((clkB t) =>
((p_sizeA (t+1) = p_sizeA t) \land
(p_loadA (t+1) = p_loadA t) \land
(p_downA (t+1) = p_downA t) \land
(p_size (t+1) = p_size (t+1)) \land
(p_load (t+1) = load_in t) \land
(p_down (t+1) = down_in t)) \land
(zero_cnt t = (p_downA t) \Rightarrow (DECN 2 (p_sizeA (t+1)) = (WORDN 0)) \land (p_sizeA (t+1) = (WORDN 0)))
);

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

let Scat_Logic_SPEC = new_definition('Scat_Logic_SPEC',
(Scat_Logic_SPEC)
);

%---------------------------------------------------------------------%
let Lock_Inputs_SPEC = new_definition
('Lock_Inputs_SPEC',
"! rst fsm_dstate p_male_ p_rale_ lock_inE lock_inh_inE.
Lock_Inputs_SPEC rst fsm_dstate p_male_ p_rale_ lock_inE lock_inh_inE =
lt:time .
(lock_inE t = (rst t) V (fsm_dstate t)) A
(lock_inh_inE t = (rst t) V -(p_male_ t) V -(p_rale_ t))"
);

P-Port controller state machine.

let FSM_SPEC = new_definition
('FSM_SPEC',
"! clkA clkB rst_in mrqt_in sack_in cgnt_in_ crqt_in_ hold_in_ lock_in_
state rst mrqt sack cgnt_ crqt_ hold_ lock_
stateA astate dstate hlda_
astate_out dstate_out hlda_out_.
FSM_SPEC clkA clkB rst_in mrqt_in sack_in cgnt_in_ crqt_in_ hold_in_ lock_in_
state rst mrqt sack cgnt_ crqt_ hold_ lock_
stateA astate dstate hlda_
astate_out dstate_out hlda_out_ =
lt:time .
((clkA t) =>>
((state (t+1) = state t) A
(rst (t+1) = rst t) A
(mrqt (t+1) = mrqt t) A
(sack (t+1) = sack t) A
(cgnt_ (t+1) = cgnt_ t) A
(crqul (t+1) = crqt_ t) A
(hold_ (t+1) = hold_ t) A
(lock_ (t+1) = lock_ t) A
(stateA (t+1) =
(rst t) => PA |
(state t = PH) => ((hold_ t) => PA | PH) |
(state t = PA) => (((mrqt t) V -(cgnt_ t) A -(crqt_ t)) => PD |
(((lock_ t) A -(hold_ t)) => PH | PA)) |
((sack t) A -(hold_ t) A -(lock_ t)) => PA |
((sack t) A -(hold_ t) A -(lock_ t)) => PH | PD) ) A
(astate (t+1) = (stateA (t+1) = PA)) A
(dstate (t+1) = (stateA (t+1) = PD)) A
(hlda_ (t+1) = -(stateA (t+1) = PA)))) A
((clkB t) =>>
((state (t+1) = stateA t) A
(rst (t+1) = rst_in t) A
(mrqt (t+1) = mrqt_in t) A
(sack (t+1) = sack_in t) A
(cgnt_ (t+1) = cgnt_in_ t) A
(crqul (t+1) = crqt_in_ t) A
(hold_ (t+1) = hold_in_ t) A
(lock_ (t+1) = lock_in_ t) A
(stateA (t+1) = stateA t) A
83
(astate \((t+1)\) = astate \(t\)) \land
(dstate \((t+1)\) = dstate \(t\)) \land
(hlda_ \((t+1)\) = hlda_ \(t\))) \land
((astate\_out \(t\) = astate\( (t+1)\)) \land
(dstate\_out \(t\) = dstate\( (t+1)\)) \land
(hlda\_out \(t\) = hlda_ \((t+1)\))))

let P_Block_SPEC = new_definition
('P_Block_SPEC',
"! (P_fsm_stateA P_fsm_state :time->pfsm_ty)
  (P_wr_data P_addr P_be P_be_n P_sizeA P_size :time->wordn)
  (P_fsm_state P_fsm_dstate P_fsm_hlda_ P_destl P_wr P_loadA P_downA P_fsm_rst P_fsm_mrqt
   P_fsm_cgnl P_fsm_cqnt P_fsm_hold P_fsm_lock P_rqt P_load P_down P_lock_
    P_lock_inh P_male_ P_rale_ :time->bool)
  (L_ad_in L_be_ L_ad_in :time->words)
  (ClkA ClkB Rst L_ad_in L_den_ L_wr L_lock_ L_cgnl L_hold_ L_srdy_ :time->bool)
  (L_ad_out L_ad_data_out L_ad_addr_out L_be_ :time->words)
  (L_ready_ L_rale_ L_male_ L_cgnt_ L__hold_ L_last_ L_hlda_ L_lock_ :time->bool)
  P_Block_SPEC (P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_,
   P_wr_data, P_addr, P_destl, P_be_,
   P_wr, P_be_n, P_sizeA, P_loadA, P_downA, P_fsm_state, P_fsm_rst, P_fsm_mrqt, P_fsm_sack,
   P_fsm_cqn, P_fsm_cqnt, P_fsm_cqnt, P_fsm_hold, P_fsm_lock, P_rqt, P_size, P_load, P_down, P_lock_,
   P_lock_inh, P_male_, P_rale_)
  (ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, L_ads_in, L_cgnl, L_hold_, L_srdy_)
  (L_ad_out, L_ready_, L_ad_data_out, L_ad_addr_out, L_be_, L_rale_, L_male_, L_cgnt_, L_cale_,
   L_mrdy_, L_last_, L_hlda_, L_lock_)
  (*)

? fam_stateA fm_stateA\[ fist\( t\)\( \rightarrow\) pfsm\_ty\]
  \(P\_wr\_data P\_addr P\_be\ P\_be\_n P\_sizeA P\_size :time->wordn\)
  \(P\_fm\_state P\_fm\_dstate P\_fm\_hlda_ P\_destl P\_wr P\_loadA P\_downA P\_fm\_rst P\_fm\_mrqt
   P\_fm\_cgnl P\_fm\_cqnt P\_fm\_hold P\_fm\_lock P\_rqt P\_load P\_down P\_lock_
    P\_lock\_inh P\_male_ P\_rale_ :time->bool\)
  \(L\_ad\_in L\_be\_ L\_ad\_in :time->words\)
  \(ClkA ClkB Rst L\_ad\_in L\_den\_ L\_wr L\_lock_ L\_cgnl L\_hold_ L\_srdy_ :time->bool\)
  \(L\_ad\_out L\_ad\_data\_out L\_ad\_addr\_out L\_be_ :time->words\)
  \(L\_ready_ L\_rale_ L\_male_ L\_cgnl_ L\_hold_ L\_last_ L\_hlda_ L\_lock_ :time->bool\).

(Data_Latches_SPEC ClkB L_ad_in L_be_ L_wr rt fsm_ASTate
  P\_wr\_data P\_addr P\_destl P\_be_ P\_wr P\_be\_n
  data\_out addr\_out be\_out)
  \(\
  (TRIBUF\_SPEC data\_out data\_out\_en L\_ad\_data\_out) \land
  (TRIBUF\_SPEC addr\_out data\_out\_en L\_ad\_addr\_out) \land
  (TRIBUF\_SPEC be\_out L\_hlda_ L\_be_) \land
  (Req\_Inputs\_SPEC L\_ad\_ in L\_den\_ reset\_rt\_q\ rqt\_inS rqt\_inR rqt\_inE) \land
  (DSRELAT\_SPEC GND rqt\_inS rqt\_inR rqt\_inE ClkB P\_rqt rqt\_outQ) \land
  (NOT\_SPEC rqt\_outQ reset\_rqt) \land
  (Ctrl\_Logic\_SPEC ClkB L\_ad\_in load\_in L\_in\_zero\_cnt
    P\_size P\_sizeA P\_load P\_loadA P\_down P\_downA) \land
  (Scat\_Logic\_SPEC Rst fsm\_stateA fm\_dstate L\_hlda_ P\_addr P\_wr P\_rqt zero\_cnt L\_srdy_
    data\_out\_en L\_ad\_out\_en L\_rale_ L\_cgnl L\_mrqt L\_mrqt
    L\_hold L\_last_ L\_hlda_ L\_lock_)
  \(\
  (TRIBUF\_SPEC rale_ L\_hlda_ L\_rale_) \land
  (TRIBUF\_SPEC male_ L\_hlda_ L\_male_) \land
  (TRIBUF\_SPEC GND L\_hlda_ L\_mrdy_)
  (NOT\_SPEC zero\_cnt zero\_cnt_)

84
(TRIBUF_SPEC zero_cnt I_hlda I_last_ ) \land 
(NOT_SPEC I_ready I_ready_ ) \land 
(DSELAT_SPEC L_lock Rst lock_inE ClkB P_lock lock_outQ ) \land 
(DSELAT_SPEC L_lock Rst lock_inh_inE ClkB P_lock_inh lock_inh_outQ ) \land 
(Lock_Inputs_SPEC Rst fsm_dstate p_male_outQ p_rst_outQ lock_inE lock_inh_inE ) \land 
(DELAT_SPEC male_fsm_asstate ClkB P_male p_male_outQ ) \land 
(DELAT_SPEC rale_fsm_asstate ClkB P_rst p_rst_outQ ) \land 
(NOT_SPEC lock_outQ lock_outQ ) \land 
(NAND2_SPEC lock_outQ lock_inh_outQ I_lock_ ) \land 
(NOT_SPEC I_cgnt I_cgnt ) \land 
(NAND3_SPEC I_cgnt fsm_asstate I_hold I_cale_ ) \land 
(BUF_SPEC I_ad_in L_ad_out ) \land 
(FSM_SPEC ClkB fsm_rst fsm_mrqt fsm_sack I_cgnt I_crqt I_hold lock_outQ 
P_fsm_state P_fsm_rst P_fsm_mrqt P_fsm_sack P_fsm_cgnt P_fsm_crqt P_fsm_hlda 
P_fsm_lock P_fsm_stateA P_fsm_asstate P_fsm_dstate P_fsm_hlda 
fsm_asstate fsm_dstate I_hlda_ )"

);;

close_theory();;
B.2 M Port Specification

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

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

system 'rm m_block.th';

new_theory 'm_block';

loadf 'abstract';

let m_state_ty = ":(mfsm ty#bool#bool#bool#bool#bool#wordn#wordn#wordn#bool#wordn#mfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#wordn#wordn#bool#bool#bool#wordn#wordn)";

let m_state = _((M_fsm_stateA, M_fram_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable, M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male_, M_fsm_bw, M_fsm_ww, M_fsm_last_, M_fsm_mrty_, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr, M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect):^m_state_ty);

let m_env_ty = ":(bool#bool#bool#bool#bool#wordn#bool#bool#bool#bool)#";

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

let m_out_ty = ":(wordn#bool#bool#bool#bool#bool#bool#bool#bool)#";

let m_out = "((I_ad_out, I_rdy_, MB_addr, MB_data_out, MB_cs_eeprom, MB_cs_sram, MB_we, MB_oe, MB_parity):^m_out_ty);

let rep_ty = abstract_type 'aux_def' 'Andn';

%

SRAM/EEMOR selection logic.

let SE_Logic_SPEC = new_definition
(\"SE_Logic_SPEC\","@! clkA clkB (i_ad, time->wordn) male mem_enable M_se cs e cs e .

86
SE_Logic_SPEC clkA clkB i_ad male mem_enable M_se cs_e_ cs_s_ =
lt:time .

((clkA t) ==> ((M_se (t+1) = M_se t)) \wedge
   ((clkB t) ==> ((M_se (t+1) = (male t) \implies ELEMENT (i_ad t) (23) \mid M_se t))) \wedge
   (cs_e_t = \neg((mem_enable t) \land (M_se (t+1)))) \wedge
   (cs_s_t = \neg((mem_enable t) \land (M_se (t+1))))"
);

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

let WR_Logic_SPEC = new_definition
('WR_Logic_SPEC',
"! clkA clkB i_ad male mem_enable M_wr wr rd_mem wr_mem .
WR_Logic_SPEC clkA clkB i_ad male mem_enable M_wr wr rd_mem wr_mem =
lt:time .

((clkA t) ==> ((M_wr (t+1) = M_wr t)) \wedge
   ((clkB t) ==> ((M_wr (t+1) = (male t) \implies ELEMENT (i_ad t) (27) \mid M_wr t))) \wedge
   (wr t = M_wr (t+1)) \wedge
   (rd_mem t = (mem_enable t) \land (M_wr (t+1))) \wedge
   (wr_mem t = (mem_enable t) \land (M_wr (t+1))))"
);

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

let Addr_Ctr_SPEC = new_definition
('Addr_Ctr_SPEC',
"! clkA clkB i_ad:time->wordn male rdyA M_addr M_addrA addr_out .
Addr_Ctr_SPEC clkA clkB i_ad male rdyA M_addr M_addrA addr_out =
lt:time .

((clkA t) ==> 
   ((M_addr (t+1) = M_addr t)) \wedge
   (M_addrA (t+1) = M_addr t)) \wedge
   ((clkB t) ==> 
   ((M_addr (t+1) = (male t) \implies (SUBARRAY (i_ad t) (18,0)) \mid
     (rdyA t) \implies (INCN 18 (M_addrA t)) \mid (M_addrA t)) \wedge
   (M_addrA (t+1) = M_addrA t))) \wedge
   (addr_out t = (rdyA t) \implies (INCN 18 (M_addrA (t+1))) \mid M_addrA (t+1)))"
);

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

let BE_Logic_SPEC = new_definition
('BE_Logic_SPEC',
"! clkA clkB i_be:time->wordn male srdy wr_mem M_be M_beA be_out ww bw .
BE_Logic_SPEC clkA clkB i_be male srdy wr_mem M_be M_beA be_out ww bw =
lt:time .

((clkA t) ==> 
   ((M_be (t+1) = M_be t)) \wedge

\[
(M_{beA}(t+1) = M_{be}(t)) \land \\
((clkB t) \Rightarrow \\
((M_{be}(t+1) = (male t) \lor (srdy t)) \Rightarrow (i_{be}(t) \lor (M_{be}(t))) \land \\
(M_{beA}(t+1) = M_{beA}(t))) \land \\
((be_{out}(t) = M_{beA}(t+1)) \land \\
(ww_t = (wr_{mem}(t) \land (VAL 3(M_{be}(t+1)) = 15)) \land \\
(bw_t = (wr_{mem}(t) \land (VAL 3(M_{be}(t+1)) = 15)))
\]

;);

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

let Rdy.Logic_SPEC = new_definition
('Rdy.Logic_SPEC',
"'I write read zero_cnt wr_mem rdy .
Rdy.Logic_SPEC write read zero_cnt wr_mem rdy =
\[\text{It:time.
(\text{rdy } t = (\text{write } t) \land (\text{zero_cnt } t) \lor (\text{read } t) \land (\text{zero_cnt } t) \land \neg (\text{wr_mem } t))\]
\]);

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

let Ctr.Logic_SPEC = new_definition
('Ctr.Logic_SPEC',
"'I clkA clkB in dn id M_count M_countA zero_cnt .
Ctr.Logic_SPEC clkA clkB in dn id M_count M_countA zero_cnt =
\[\text{It:time.
((\text{clkA } t) \Rightarrow \\
((M_{count}(t+1) = M_{count}(t)) \land \\
(M_{countA}(t+1) = M_{countA}(t))) \land \\
((\text{clkB } t) \Rightarrow \\
((M_{count}(t+1) = \neg (dn t)) \Rightarrow (\text{in } t) \Rightarrow (\text{WORDN } 1) \lor (\text{WORDN } 2)) \land \\
(dn t) \Rightarrow (\text{DECN } 1 (M_{countA}(t)) \land (M_{countA}(t))) \land \\
(zero_{cnt}(t) = (M_{countA}(t+1) = ((dn t) \Rightarrow (\text{WORDN } 1) \lor (\text{WORDN } 0))))\]
\]);

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

let Enable.Logic_SPEC = new_definition
('Enable.Logic_SPEC',
"'I cs_eeprom rd_mem address read write byte_write wwdel
enable_eeprom disable_writes oe edac le we mb wr_en .
Enable.Logic_SPEC cs_eeprom rd_mem address read write byte_write wwdel
\[\text{disable_eeprom disable_writes oe edac le we mb wr_en =}
\[\text{\text{lt:time.}
(\text{oe } t = \neg ((\text{rd_mem } t) \land (\text{address } t) \lor (\text{read } t))) \land \\
(\text{we } t = \neg (\text{cs_eeprom } t) \land (\text{disable_eeprom } t) \lor (\text{disable_writes } t) \lor \\
(\text{disable_writes } t) \lor \\
\]
\]

88
\[ \neg((\text{write } t) \lor (\text{byte\_write } t) \lor (\text{wwdel } t)) \land \\
(\text{edac\_le } t = \text{read } t) \land \\
(\text{mb\_wr\_en } t = \neg(\text{write } t)) \]

% Generation logic for Srdy_

let Srdy_Logic_SPEC = new_definition
('Srdy_Logic_SPEC',
  "! wr rdy rdy\_outQ srdy_.
Srdy_Logic_SPEC wr rdy rdy\_outQ srdy_ =
  \text{It}:\text{time}.
  srdy_ t = \neg((\text{rdy\_outQ } t) \land \neg(\text{wr } t)) \lor (\text{rdy } t) \land (\text{wr } t))"
);

% Memory decode logic.

let EDAC_Decode_Logic_SPEC = new_definition
('EDAC_Decode_Logic_SPEC',
  "! (rep:Arep\_ty) (mb\_data\_in:time->wordn) edac\_en data\_out detect\_out.
EDAC_Decode_Logic_SPEC rep mb\_data\_in edac\_en data\_out detect\_out =
  \text{It}:\text{time}.
  (data\_out t = (edac\_en t) \Rightarrow (\text{Ham\_Dec rep (mb\_data\_in t)}) \land \\
   (detect\_out t = (edac\_en t) \Rightarrow (\text{Ham\_Det2 rep (mb\_data\_in t)}) \land (\text{WORDN 0}))"
);

% Memory read latches.

let Read_Latches_SPEC = new_definition
('Read_Latches_SPEC',
  "! (rep:Arep\_ty) clkA clkB (data\_inD:time->wordn) edac\_en edac\_le detect\_inD detect\_inE
Read_Latches_SPEC rep clkA clkB data\_inD edac\_en edac\_le detect\_inD detect\_inE
  M\_rd\_data M\_rd\_dataA M\_detect m\_data\_outQ m\_detect\_outQ =
  \text{It}:\text{time}.
  ((\text{clkA } t) \Rightarrow \\
   ((M\_rd\_data t+1) = M\_rd\_data t) \land \\
   (M\_rd\_dataA t+1) = M\_rd\_data t) \land \\
   (M\_detect t+1) = (\text{detect\_inD } t) \Rightarrow (\text{detect\_inE } t) \Rightarrow (M\_detect t)) \land \\
  ((\text{clkB } t) \Rightarrow \\
   ((M\_rd\_data t+1) = (edac\_le t) \Rightarrow (data\_inD t) \lor (M\_rd\_data t)) \land \\
   (M\_rd\_dataA t+1) = M\_rd\_data t) \land \\
   (M\_detect t+1) = M\_detect t) \land \\
   (m\_data\_outQ t = M\_rd\_dataA t+1)) \land \\
   (m\_detect\_outQ t = \text{Ham\_Det2 rep ((M\_detect t+1), (edac\_en t)))})"
);

89
Enable input logic for EDAC correction reporting.

let Detect_Enable_Locic_SPEC = new_definition
("Detect_EnableLogic_SPEC", "I edac_en edac_rd detect_inE.
Detect_Enable_Locic_SPEC edac_en edac_rd detect_inE =
lt time.
(detect_inE \(= (edac_en t) \land (edac_rd t) \lor -(edac_rd t))")
);

% Memory write data multiplexer.

let Mux_OutLogic_SPEC = new_definition
("Mux_OutLogic_SPEC", "I (m_data_outQ:time->wordn) i_ad be mb_data_out.
Mux_OutLogic_SPEC m_data_outQ i_ad be mb_data_out =
lt time.

let odl =
(MALTER (mb_data_out t) (7,0) ((ELEMENT (be t) (0)) => (SUBARRAY (i_ad t) (7,0))
\(= (SUBARRAY (m_data_outQ t) (7,0))))

in
(let od2 =
(MALTER odl (15,8) ((ELEMENT (be t) (1)) => (SUBARRAY (i_ad t) (15,8))
\(= (SUBARRAY (m_data_outQ t) (15,8))))

in
(let od3 =
(MALTER od2 (23,16) ((ELEMENT (be t) (2)) => (SUBARRAY (i_ad t) (23,16))
\(= (SUBARRAY (m_data_outQ t) (23,16))))

in
(let od4 =
(MALTER od3 (31,24) ((ELEMENT (be t) (3)) => (SUBARRAY (i_ad t) (31,24))
\(= (SUBARRAY (m_data_outQ t) (31,24))))

in (mb_data_out t = od4)))")
);

% Data encoding logic.

let Enc_OutLogic_SPEC = new_definition
Enc_OutLogic_SPEC rep mb_data_out mb_edata_out =
lt time.
(mb_edata_out t = Ham_Enc rep (mb_data_out t))")
);

% Input logic for M-parity latch.

C-2
let Memparity_In_Logic_SPEC = new_definition
('Memparity_In_Logic_SPEC',
"srdy mem_enable detect_outQ rst reset_parity memparity_inS memparity_inR memparity_inE .
Memparity_In_Logic_SPEC srdy mem_enable detect_outQ rst reset_parity
memparity_inS memparity_inR memparity_inE =
l:t:time.
(memparity_inS t = (srdy t) \(\land\) (mem_enable t) \(\land\) (detect_outQ t)) \(\land\)
(memparity_inR t = (rst t) \(\lor\) (reset_parity t)) \(\land\)
(memparity_inE t = (memparity_inS t) \(\lor\) (memparity_inR t))"
);

M-Port controller state machine.

let FSM_SPEC = new_definition
('FSM_SPEC',
"clkA clkB male_in rd_in bw_in ww_in last_in _ mrdy_in _ zero_cnt_in rst_in
state male_rd bw ww last_mrdy_zero_cnt rst
stateA address read write byte_write mem_enable
address_out read_out write_out byte_write_out mem_enable_out .
FSM_SPEC clkA clkB male_in rd_in bw_in ww_in last_in _ mrdy_in _ zero_cnt_in rst_in
state male_rd bw ww last_mrdy_zero_cnt rst
stateA address read write byte_write mem_enable
address_out read_out write_out byte_write_out mem_enable_out =
l:t:time.

((clkA t) \(\implies\)
  ((state (t+1) = state t) \(\land\)
    (male_ (t+1) = male_ t) \(\land\)
    (rd (t+1) = rd t) \(\land\)
    (bw (t+1) = bw t) \(\land\)
    (ww (t+1) = ww t) \(\land\)
    (last_ (t+1) = last_ t) \(\land\)
    (mrdy_ (t+1) = mrdy_ t) \(\land\)
    (zero_cnt (t+1) = zero_cnt t) \(\land\)
    (rst (t+1) = rst t) \(\land\)
    (stateA (t+1) =
     ((rst t) \(\implies\) MI I
     (state t = MI) \(\implies\) ((-(male_ t)) \(\implies\) MA I MI)) I
     (state t = MA) \(\implies\) (((mrdy_ t) \(\land\) (ww t)) \(\implies\) MW I
     (-(mrdy_ t) \(\land\) ((rd t) \(\lor\) (bw t))) \(\implies\) MR I MA)) I
     (state t = MR) \(\implies\) (((bw t) \(\land\) (zero_cnt t)) \(\implies\) MBW I
     (last_ t) \(\land\) (rd t) \(\land\) (zero_cnt t)) \(\implies\) MA I
     (-(last_ t) \(\land\) (rd t) \(\land\) (zero_cnt t)) \(\implies\) MRR I MR)) I
     (state t = MRR) \(\implies\) MI I
     (state t = MW) \(\implies\) (((zero_cnt t) \(\land\) (-(last_ t))) \(\implies\) MI I
     ((zero_cnt t) \(\land\) (last_ t)) \(\implies\) MA I MW) I
     (MW)))) \(\land\)
    (address (t+1) = (stateA (t+1) = MA)) \(\land\)
    (read (t+1) = (stateA (t+1) = MR)) \(\land\)
    (write (t+1) = (stateA (t+1) = MW)) \(\land\)
    (byte_write (t+1) = (stateA (t+1) = MBW)) \(\land\)
    (mem_enable (t+1) = -(stateA (t+1) = MI)))) \(\land\)
((clkB t) ===> 
  ((state (t+1) = stateA t) ∧
  (male_ (t+1) = male_in_ t) ∧
  (rd (t+1) = rd_in t) ∧
  (bw (t+1) = bw_in t) ∧
  (ww (t+1) = ww_in t) ∧
  (last_ (t+1) = last_in_ t) ∧
  (rdy_ (t+1) = rdy_in t) ∧
  (zero_cnt (t+1) = zero_cnt_in t) ∧
  (rst (t+1) = rst_in t) ∧
  (stateA (t+1) = stateA t) ∧
  (address (t+1) = address t) ∧
  (read (t+1) = read t) ∧
  (write (t+1) = write t) ∧
  (byte_write (t+1) = byte_write t) ∧
  (mem_enable (t+1) = mem_enable t))) ∧

  ((address_out t = address (t+1)) ∧
  (read_out t = read (t+1)) ∧
  (write_out t = write (t+1)) ∧
  (byte_write_out t = byte_write (t+1)) ∧
  (mem_enable_out t = mem_enable (t+1))))

);;

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

let M_Block_SPEC = new_definition
  "(M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_fsm_rdy)
  (M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_fsm_rdy)
  (M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable, M_fsm_rdy)

M_Block_SPEC (M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable, M_fsm_rdy)

(rep:rep_ty).

? male address read write byte_write mem_enable wr rd_mem wr_mem rdy_outQ srdy be w w bw zero_cnt rdy count_inD count_inLD wwdel_inD wwdel_outQ edac_le rdy_outQ srdy edac_en data_out detect_out data_inD detect_inD detect_inE m_data_outQ m Detect_outQ mb_data_out mb_edata_out mb_wr_en mb_wr_en memparity_inS memparity_inR memparity_inE.
(NOT_SPEC L_male_ male) ∧
(SE_Logic_SPEC ClkB I_ad_in male mem_enable M_se MB_cs_eeprom MB_cs_sram) ∧
(WR_Logic_SPEC ClkB I_ad_in male mem_enable M_wr wr rd_memb wr_mem) ∧
(Addr_Ctr_SPEC ClkB I_ad_in male rdy_outQ M_addr M_addrA MB_addr) ∧
(BE_Logic_SPEC ClkB I_be_male srdy wr_mem M_be M_beA be bw bw) ∧
(Rdy_Logic_SPEC write read zero_cnt wr_memb rdy) ∧
(Ctr_Logic_SPEC ClkB MB_cs_eeprom count_inLD count_inLD M_count M_countA zero_cnt) ∧
(OR2_SPEC write read count_inDN) ∧
(OR2_SPEC address byte_write count_inLD) ∧
(AND2_SPEC wr address wwdel_inD) ∧
(DLAT_SPEC wwdel_inD ClkB M_wwdel wwdel_outQ) ∧
(Enable_Logic_SPEC MB_cs_eeprom rd_mem address read write byte_write wwdel_outQ
  Disable_eeprom Disable_writes MB_oe edac_le MB_we mb wr_en) ∧
(DF SPEC rdy ClkB M_rdy M_rdyA rdy_outQ) ∧
(Srdy Logic SPEC wr rdy rdy_outQ srdy) ∧
(Tribuf SPEC srdy_mem_enable I_srdy) ∧
(NOT_SPEC srdy_srdy) ∧
(NOT_SPEC Edac_en edac_en) ∧
(EDAC_Decode_Logic_SPEC rep MB_data_in edac_en data_out detect_out) ∧
(Read_Latches_SPEC rep ClkB data_inD edac_en edac_le detect_inD detect_inE
  M_rd_data M_rd_dataA M_detect m_data_outQ m_detect_outQ) ∧
(TRIBUF_SPEC m_data_outQ rd_mem I_ad_out) ∧
(Detect_Enable_Logic_SPEC edac_en rd_mem detect_inE) ∧
(Mux_Out_Logic_SPEC m_data_outQ I_ad_in be mb_data_out) ∧
(Enc_Out_Logic_SPEC rep mb_data_out mb_data_out) ∧
(NOT_SPEC mb_wr_men mb_wr_en) ∧
(TRIBUF_SPEC mb_wwdel_outQ mb_wr_en MB_data_out) ∧
(Memparity_In.Logic_SPEC srdy mem_enable m_detect_outQ Rst Reset_parity
  memparity_inS memparity_inR memparity_inE) ∧
(DSRELAT_SPEC GND memparity_inS memparity_inR memparity_inE ClkB
  M_parity M_parity) ∧
(FSM_SPEC ClkB I_male_ rd_mem bw ww I_last I_mrdy_zero_cnt Rst
  M_fsm_state M_fsm_male M_fsm_rd M_fsm_bw M_fsm_ww M_fsm_last M_fsm_mrdy
  M_fsm_zero_cnt M_fsm_rst
  M_fsm_stateA M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable
  address read write byte_write mem_enable) ∧

); close_theory();
B.3

R Port Specification

File:

r_block.ml

Author.

(c) D.A. Fura 1992

Date:

31 March 1992

This file contains
developed

set_search_path

system

the ml source for the gate-level

by the Embedded Processing

(search_patiO)

specification

of the R-Port of the FTEP PIU, an ASIC

Laboratory, Boeing High Technology

Center.

@ ['/home3titan3/dfura/ftep/piu/hol/lib/']);;

'rm r_block.th';;

new_theory

'r_block' ;;

map loadf ['abstract';'buses._deP
map new_parent

];;

['gates_def';'latches_def';'ffs_def';'counters_def;'datapaths_def';'raux_def';

'aux_def';

'array def' ;' wordn_def' ];;
let r._state_ty = ":(rfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#b_o_or_#
bool#bool#wordn#wordn#bool#bool#wordn#wordn#bool#bool#wordn#wordn#bool#boo_
wordn#bool#wordn#wordn#wordn#
rfsm__l#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#boo_o__
bool#bool#bool#wordn#wordn#bool#wordn#bool#bool#bool#wordn#wordn#bool#wor_
bool#bool#bool#wordn#wordn#bool#wordn#bool#bool#bool#wordn#wordn#bcol#bool#
wordn#wordn#woxdn#bool#wordn#bool#wordn#bool#wordn#bool)";;
let r_state = "((R_fsm_stateA,

R_fsm_cntlatch,

R_fsm_stdy_,
R_c23_cout,

R_int0_ca,

R_c01_cout,

R_c01_cout_delA,

R_reg_selA,

R_ca'O, R_cUO_ce, R_cUO_cin. R_ctr0_outA,

R_clrl_outA,

R_ctr2, R._ctr2_ce, R_ctr2_cin,

R_ctr3_outA,

R_icr loadA, R_icr_oldA,

R_fsm_mrdy_,
R_c23_cout_del,
R_ctr0

R_fsm

last_, R_fsm_rst,

R_int0_disA,

R_c23_cout_delA.
R_ctr2_outA,

R_int3_en,

R_cntlatch_deLA.

R_ctrl,

R_ctrl_ce,

R ctrl_cin,

R_clz3. R_ctr3_ce,

R_ctr3_cin,

R_icrA, R_bmA_latch,

R_fsm_stata,

R_int0_dis,

R_c01_cout_del,

R int2_en, R wr, R_cntlatch_del,

R_'mt3_dis,
R_srdy_deL,

R_reg_sel,

mux._sel, R_c,tr0_irden, R._ctr0_cry, R._ctK)_new, R_ctr0_out,

R int3..disA,
R srdy_delA_,

R_fsm_ale_,
R intl_en,

R_ctr0

in,

R_ctrO_orden,

R_ctrl_in,

R_ctrl_mux_sel,

R._ctrl_irden,

R c_rl_cry, R_ctrl._new,

R_ctrl_out,

R._ctrl._orden,

R_ctr2__in,

R ctr2 mux_sel,

R_ctf2_irden,

R_ctr2_cry,

R__ctr2_new, R_ctr2_out,

R_ctr2_orden,

R_ctr3_in,

R ctr3 mux__sel, R_cC3_irden,

R_ctr3_cry,

R_ctr3_new,

R_icr__old, R icr mask, R_icr, R_icr_rden,

R ctr3_out, R_ctr3._orden,

R_ccr, R_ccr_rden,

R icr_load,

R_.gcr, R._gcr_.rden, R._sr,

R_sr_rden)
:"r._state_ty)";;
let r_env_ty

= ":(bool#bool#bool#wordn#bool#bool#wordn#bool#boogtbool#wordn#wordn#b
wordn#wordn#wordn#boo_bool#wordu
)";;

let r_env = "((CIkA,

CIkB, Rst, Lad_m,

Clm_fail,

Reset_cpu,

I..rale._, Llast_,

Piu_fail,

Pmm_fail,

ooI#bool#

I_be_, I mrdy_, Disable__mt, Disable_writes,
S_state, ld, Channel/D,

:Ar_env_ty)";;

94

CB..pafity,

MB__pffirity,C_ss)


let r_out_ty = "(word#bool#bool#bool#bool#bool#word#word#bool#bool)",;;
let r_out = "((I_ad_out, I_srdy_, Int0, Int1, Int2, Ccr, Led, Reset_error, Pmm_invalid)
 :r_out_ty)";;

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

R-Port controller state machine.

let FSM_SPEC = new_definition
  ("FSM_SPEC",
   "! (ClkB:time->bool) ClkB ale_in_ mrdy_in_ last_in_ rst_in
    ale_ mrdy_ last_ rst state
    cntlatch srdy_ (stateA:time->rfsm_ty)
    s0_out s1_out cntlatch_out srdy_out_ .
    FSM_SPEC ClkB ale_in_ mrdy_in_ last_in_ rst_in
    ale_ mrdy_ last_ rst state
    cntlatch srdy_ stateA
    s0_out s1_out cntlatch_out srdy_out_ =
    !:time .
    (((ClkB t) =>
      ((stateA (t+1) = ((rst t) => RI l
      ((state t) = Rl) => ((~ale_ t) => RA l RL) l
      ((state t) = RA) => ((~mrdy_ t) => RD l RA) l
      ((~last t) => RI l RA))) l
      (cntlatch (t+1) = ((state t = RI) L ~ale_ t)) l
      (srdy_ (t+1) = ~(state t = RA) L ~mrdy_ t)) l
      (state (t+1) = state t) l
      (ale_ (t+1) = ~ale_ t) l
      (mrdy_ (t+1) = ~mrdy_ t) l
      (last_ (t+1) = ~last_ t) l
      (rst (t+1) = ~rst t)) l
    (ClkB t) =>
    (((stateA (t+1) = stateA t) l
      (cntlatch (t+1) = cntlatch t) l
      (srdy_ (t+1) = srdy_ t) l
      (state (t+1) = stateA t) l
      (ale_ (t+1) = ale_in_ t) l
      (mrdy_ (t+1) = mrdy_in_ t) l
      (last_ (t+1) = last_in_ t) l
      (rst (t+1) = rst_in t)) l
    ((s0_out (t+1) = (stateA (t+1) = RD))) l
    (s1_out (t+1) = ((stateA (t+1) = RA) V (stateA (t+1) = RD))) l
    (cntlatch_out t = cntlatch (t+1)) l
    (srdy_out_ t = srdy_ (t+1)))")";;

R_wr latch definition.

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

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

let RW_Sigs_SPEC = new_definition
('RW_Sigs_SPEC',
"! wr s0 s1 disable_writes dp_read r_write r_read icr_rd_en srdy_en .
RW_Sigs_SPEC wr s0 s1 disable_writes dp_read r_write r_read icr_rd_en srdy_en =
| t:time .
| (dp_read t = (-r_wr t) \land (s0 t) \lor (s1 t)) \land
| (r_write t = (disable_writes t) \land (r_wr t) \land (s0 t) \land (s1 t)) \land
| (r_read t = (-r_wr t) \land (s0 t) \land (s1 t)) \land
| (icr_rd_en t = (-s0 t) \land (s1 t)) \land
| (srdy_en t = (s0 t) \lor (s1 t)))
|);

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

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

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

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

96
Reg_File_Ctl_SPEC reg_sel write read icr_rd_en

```latex
\begin{align*}
\text{cir_wr01} & \text{ cir_wr23} \\
c0ir_wr & c0ir_rd c0or_rd c1ir_wr c1ir_rd c1or_rd \\
c2ir_wr & c2ir_rd c2or_rd c3ir_wr c3ir_rd c3or_rd \\
icr_wr_feedback & \text{icr_select icr_rd} \\
cwr_wr & \text{crr_rd gcr_wr gcr_rd sr_rd}.
\end{align*}
```

Input logic for R_int1_en, R_int2_en latches.

```latex
\begin{align*}
\textbf{Ctr_Int.Logic_SPEC} \leftarrow \text{new_definition} \\
\text{"Ctr_Int.Logic_SPEC";} \\
\leftarrow \text{"one_shot interrupt reload cout cout_del cirwr}
\int en\_inR int en\_inS int en\_inE \_ld .
\textbf{Ctr_Int.Logic_SPEC} \leftarrow \text{one_shot interrupt reload cout cout_del cirwr}
\int en\_inR int en\_inS int en\_inE \_ld =
\end{align*}
```

\( (t:time) . \)
\begin{align*}
& (\text{cir_wr01} \ t = \text{(write t)} \land ((\text{reg_sel t) = \text{WORDN 8}) \lor ((\text{reg_sel t) = \text{WORDN 9}))}) \land \\
& (\text{cir_wr23} \ t = \text{(write t)} \land ((\text{reg_sel t) = \text{WORDN 10}) \lor ((\text{reg_sel t) = \text{WORDN 11}))}) \land \\
& (\text{c0ir_wr} \ t = \text{(write t)} \land ((\text{reg Sel t) = \text{WORDN 8}}) \land \\
& (\text{c0ir_rd} \ t = \text{(read t)} \land ((\text{reg Sel t) = \text{WORDN 8}}) \land \\
& (\text{c0or_rd} \ t = \text{(read t)} \land ((\text{reg Sel t) = \text{WORDN 12}}) \land \\
& (\text{c1ir_wr} \ t = \text{(write t)} \land ((\text{reg Sel t) = \text{WORDN 9}}) \land \\
& (\text{c1ir_rd} \ t = \text{(read t)} \land ((\text{reg Sel t) = \text{WORDN 9}}) \land \\
& (\text{c1or_rd} \ t = \text{(read t)} \land ((\text{reg Sel t) = \text{WORDN 13}}) \land \\
& (\text{c2ir_wr} \ t = \text{(write t)} \land ((\text{reg Sel t) = \text{WORDN 10}}) \land \\
& (\text{c2ir_rd} \ t = \text{(read t)} \land ((\text{reg Sel t) = \text{WORDN 10}}) \land \\
& (\text{c2or_rd} \ t = \text{(read t)} \land ((\text{reg Sel t) = \text{WORDN 14}}) \land \\
& (\text{c3ir_wr} \ t = \text{(write t)} \land ((\text{reg Sel t) = \text{WORDN 11}}) \land \\
& (\text{c3ir_rd} \ t = \text{(read t)} \land ((\text{reg Sel t) = \text{WORDN 11}}) \land \\
& (\text{c3or_rd} \ t = \text{(read t)} \land ((\text{reg Sel t) = \text{WORDN 15}}) \land \\
& (\text{icr_wr_feedback} \ t = \text{(write t)} \land ((\text{regSel t) = \text{WORDN 0}}) \lor ((\text{regSel t) = \text{WORDN 1}})) \land \\
& (\text{icr_select} \ t = \neg((\text{regSel t) = \text{WORDN 1}})) \land \\
& (\text{icr_rd} \ t = (\text{icr_rd_en t}) \land ((\text{regSel t) = \text{WORDN 0}}) \lor ((\text{regSel t) = \text{WORDN 1}})) \land \\
& (\text{cwr_wr} \ t = (\text{write t}) \land ((\text{regSel t) = \text{WORDN 3}}) \land \\
& (\text{cwr_rd} \ t = (\text{read t}) \land ((\text{regSel t) = \text{WORDN 3}}) \land \\
& (\text{gcr_wr} \ t = (\text{write t}) \land ((\text{regSel t) = \text{WORDN 2}}) \land \\
& (\text{gcr_rd} \ t = (\text{read t}) \land ((\text{regSel t) = \text{WORDN 2}}) \land \\
& (\text{sr_rd} \ t = (\text{read t}) \land ((\text{regSel t) = \text{WORDN 4}}))
\end{align*}
```

\[
\]
% Input logic for R_int0_en, R_int3_en latches.

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

% Generation logic for Int0_, Int3_signals.

let Reg_Int.Logic_SPEC = new_definition
('Reg_Int.Logic_SPEC',
  "| int0_en int0_dis int3_en int3_dis disable_int int0_.int3_ .
  Reg_Int.Logic_SPEC int0_en int0_dis int3_en int3_dis disable_int int0_.int3_ =
  (lt:time.
    (int0_ t = -((int0_en t) A (~int0_dis t) A (~disable_int t))) A
    (int3_ t = -((int3_en t) A (~int3_dis t) A (~disable_int t)))")
  );;

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

let SR_Inputs_SPEC = new_definition
('SR_Inputs_SPEC',
  "| cpu_fail reset_cpu piu_fail pmm_fail s_state
  id channelID cb_parity c_ss mb_parity (sr_inp:time->wordn).
  SR_Inputs_SPEC cpu_fail reset_cpu piu_fail pmm_fail s_state
  id channelID cb_parity c_ss mb_parity sr_inp =
  (lt:time.
    let a1 = (MALTER ARBN (1,0) (cpu_fail t)) in
    let a3 = (MALTER a1 (3,2) (reset_cpu t)) in
    let a5 = (ALTER a3 (8) (piu_fail t)) in

98
let a6 = (ALTER a5 (9) (pmm_fail t)) in
let a7 = (MALTER a6 (15,12) (s_state t)) in
let a8 = (MALTER a7 (21,16) (id t)) in
let a9 = (MALTER a8 (23,22) (channelID t)) in
let a10 = (ALTER a9 (24) (cb_.parity t)) in
let a11 = (MALTER a10 (27,25) (c_ss t)) in
let a12 = (ALTER a11 (28) (mb_.parity t)) in
let sr_inp t = (sr_inp t = a12)"

%
Virtual logic to distribute single GCR output word as several pieces.
%
let GCR_Outputs_SPEC = new_definition
('GCR_Outputs_SPEC',
"1 (gcr_out:thne->wordn)
led reload01 oneshot01 interrupt01 enable01
reload23 oneshot23 interrupt23 enable23 reset_error pmm_invalid.
GCR_Outputs_SPEC gcr_out led reload01 oneshot01 interrupt01 enable01 reload23 oneshot23 interrupt23 enable23 reset_error pmm_invalid =
lt:time .
(led t = SUBARRAY (gcr_out t) (3,0)) ∧
(reload01 t = ELEMENT (gcr_out t) (16)) ∧
(oneshot01 t = ELEMENT (gcr_out t) (17)) ∧
(interrupt01 t = ELEMENT (gcr_out t) (18)) ∧
(enable01 t = ELEMENT (gcr_out t) (19)) ∧
(reload23 t = ELEMENT (gcr_out t) (20)) ∧
(oneshot23 t = ELEMENT (gcr_out t) (21)) ∧
(interrupt23 t = ELEMENT (gcr_out t) (22)) ∧
(enable23 t = ELEMENT (gcr_out t) (23)) ∧
(reset_error t = ELEMENT (gcr_out t) (24)) ∧
(pmm_invalid t = ELEMENT (gcr_out t) (28))"
);
%
Virtual logic to generate the 12 tristate driver enables for datapath Bus A.
%
let Bus_Enab_SPEC = new_definition
('Bus_Enab_SPEC',
"1 clkA r_ctr0_iden r_ctr0_orden r_ctr1_iden r_ctr1_orden r_ctr2_iden r_ctr2_orden
r_ctr3_iden r_ctr3_orden r_icr_orden r_ccr_orden r_gcr_orden r_sr_orden
busA_c0_en1 busA_c0_en2 busA_c1_en1 busA_c1_en2 busA_c2_en1 busA_c2_en2
busA_c3_en1 busA_c3_en2 busA_icr_en busA_ccr_en busA_gcr_en busA_sr_en .
Bus_Enab_SPEC clkA r_ctr0_iden r_ctr0_orden r_ctr1_iden r_ctr1_orden r_ctr2_iden r_ctr2_orden
r_ctr3_iden r_ctr3_orden r_icr_orden r_ccr_orden r_gcr_orden r_sr_orden
busA_c0_en1 busA_c0_en2 busA_c1_en1 busA_c1_en2 busA_c2_en1 busA_c2_en2
busA_c3_en1 busA_c3_en2 busA_icr_en busA_ccr_en busA_gcr_en busA_sr_en =
lt:time .
(busA_c0_en1 t = (clkA t) ∧ (r_ctr0_iden t)) ∧
(busA_c0_en2 t = (clkA t) ∧ (r_ctr0_orden t)) ∧
(busA_c1_en1 t = (clkA t) ∧ (r_ctr1_iden t)) ∧
(busA_c1_en2 t = (clkA t) ∧ (r_ctr1_orden t)) ∧
(busA_c2_en1 t = (clkA t) ∧ (r_ctr2_iden t)) ∧
(busA_c2_en2 t = (clkA t) ∧ (r_ctr2_orden t)) ∧
(busA_c3_en1 t = (clkA t) ∧ (r_ctr3_iden t)) ∧
(busA_c3_en2 t = (clkA t) ∧ (r_ctr3_orden t)) ∧
(busA_icr_en t = (clkA t) ∧ (r_icr_orden t)) ∧
(busA_ccr_en t = (clkA t) ∧ (r_ccr_orden t)) ∧
(busA_gcr_en t = (clkA t) ∧ (r_gcr_orden t)) ∧
(busA_sr_en t = (clkA t) ∧ (r_sr_orden t))"
let R_Block_SPEC = new_definition
('R_Block_SPEC',
"| (rep:rep_ty)
(R_fsm_stateA R_fsm_state :time->rfsm_ty)
(R_reg SelA R_ctrl0 R_ctrl0_ourA R_ctrl1 R_ctrl1_ourA R_ctrl2 R_ctrl2_ourA R_ctrl3 R_ctrl3_ourA R_icr_oldA
R_iaR R_busA latch R_reg_sel R_ctrl0_in R_ctrl0_new R_ctrl1_in R_ctrl1_new R_ctrl2_out R_ctrl2_in R_ctrl2_new R_ctrl3_in R_ctrl3_new R_ctrl3_out R_icr_old R_iaR R_mask R_icr
R_cR R_gR R_sr :time->wordn)
(R_fsm_cnlatch R_fsm_srdy R_int0_en R_int0_disA R_int3_en R_int3_disA R_c01_cout R_c01_cout delA
R_c23_cout R_c23_cout delA R_cnlatch delA R_srdy delA R_ctrl0 ce R_ctrl0_cin R_ctrl1 ce R_ctrl1_cin
R_ctrl2 ce R_ctrl2_cin R_ctrl3 ce R_ctrl3_cin R_iaR R_mask R_iaR R_loadA R_fsm_mrdy R_fsm_last R_fsm_rst
R_int0_dis R_int3_dis R_c01_cout del R_int1_en R_c23_cout del R_int2_en R_wr R_cnlatch del
R_srdy del R_ctrl0 mux sel R_ctrl0_irden R_ctrl0_cry R_ctrl0_orden R_ctrl1 mux sel R_ctrl1_irden
R_ctrl2 mux sel R_ctrl2_irden R_ctrl2_cry R_ctrl2_orden R_ctrl3 mux sel R_ctrl3_irden R_ctrl3_cry R_ctrl3_orden R_iaR R_orden R_iaR R_rden R_icr_load R_iaR R_rden R_cR R_orden R_iaR R_rden R_gR R_rden R_sr
R_sr :time->wordn)
(I_ad_in I_be Cpu fail Reset_cpu S_state Id ChannelID C_ss :time->wordn)
(ClkA ClkB Rst I_rale_ I_last_ I_mrdy_ Disable_int Disable_writes Piu_fail Pnm_fail
C parity MB_parity :time->bool)
(I_ad_out Ccr Led :time->wordn)
(I_srdy_ Int0_ Intl Int2 Int3_ _Reset_error Pnm_invalid :time->bool)
R_Block_SPEC rep
(R_fsm_stateA R_fsm_cnlatch R_fsm_srdy R_int0_en R_int0_disA R_int3_en R_int3_disA R_c01_cout R_c01_cout delA
R_c23_cout R_c23_cout delA R_cnlatch delA R_srdy delA R_ctrl0 ce R_ctrl0_cin R_ctrl1 ce R_ctrl1_cin
R_ctrl2 ce R_ctrl2_cin R_ctrl3 ce R_ctrl3_cin R_iaR R_mask R_iaR R_loadA R_fsm_mrdy R_fsm_last R_fsm_rst
R_int0_dis R_int3_dis R_c01_cout del R_int1_en R_c23_cout del R_int2_en R_wr R_cnlatch del R_srdy del R_reg Sel R_ctrl0_in
R_ctrl0_ourA R_ctrl0_new R_ctrl1_in R_ctrl1_new R_ctrl2_out R_ctrl2_in R_ctrl2_new R_ctrl3_in R_ctrl3_out
R_ctrl3_new R_ctrl3_out R_icr_load R_iaR R_orden R_iaR R_rden R_icr_rden R_cR R_gR R_rden R_sr
R_sr :time->wordn)
(ClkA ClkB Rst I_ad_in I_rale_ I_last_ I_be_ I_mrdy_ Disable_int Disable_writes
Cpu fail Reset_cpu Piu_fail Pnm_fail S_state Id ChannelID C parity MB_parity C_ss
(I_ad_out I_srdy_ Int0_ Intl Int2 Int3_ Ccr Led _Reset_error Pnm_invalid)
? fsm_s0 fsm_s1 fsm_cnlatch fsm_srdy srdy_en wr_inE wr_outQ
This file contains the ml source for the gate-level specification of the C-Port of the FTEP PIU, an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center.

```ml
set_search_path (search_path) @ ["/home/titan3/dfura/ftep/piu/lib/"]);;

system 'rm c_block.th';;
new_theory 'c_block';;
loadf 'abstract';;
map new_parent ['gates_def';'latches_def';'ffs_def';'counters_def';'aux_def';'array_def';'wordn_def'];;

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

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

let c_state_ty = ":(cmfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool#wordn#)

let c_state = "((C_mfsm_stateA,C_mfsm_mabort,C_mfsm_midle,C_mfsm_mrequest,C_mfsm_ma3,C_mfsm_ma2,C_mfsm_ma1, C_mfsm_ma0,C_mfsm_md1,C_mfsm_md0,C_mfsm_iad_en_m,C_mfsm_mcout_sel1,C_mfsm_mcout_sel0, C_mfsm_ms,C_mfsm_rq,C_mfsm_cgt,C_mfsm_cm_en,C_mfsm_abort_le_en,C_mfsm_mparity, C_sfsm_stateA,C_sfsm_ss,C_sfsm_iad_en_s,C_sfsm_sidle,C_sfsm_slot,C_sfsm_sa1,C_sfsm_ssd0, C_sfsm_sale,C_sfsm_ssd1,C_sfsm_sd0,C_sfsm_sack,C_sfsm_sabort,C_sfsm_scoutsel0,C_sfsm_sparity, C_efsm_stateA,C_efsm_srdy_en, C_clkAA,C_sidle_delA,C_mrqf_delA,C_last_inA,C_ssA,C_holdA,C_rd_srdy,C_mcout_0_le_delA,
```

103
let c_env_ty = "((wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool# bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool# bool)";;

let c_env = "((I_ad_in, I_be_in, I_mrdy_in, I_rale_in, I_male_in, I_last_in, I_srdy_in, I_lock, I_cale, I_hlda, I_crqt, CB_rq_in, CB_ad_in, CB_ms_in, CB_ss_in, Rst, ClkB, ClkD, Id, ChannelD, Pmm_failure, Piu_invalid, Ccr, Reset_error)";;

let c_out_ty = "((bool#bool#bool#bool#bool#bool#bool#wordn#wordn#bool#wordn#wordn#wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool# bool)";;

let c_out = "((I_cgnt, I_mrdy_out, I_hold, I_rale_out, I_male_out, I_last_out, I_srdy_out, I_ad_out, I_be_out, CB_rq_out, CB_ms_out, CB_ss_out, CB_ad_out, CB_ss_out, Disable_writes, CB_parity)";;

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

let LastLogic = new_definition ('Last_Logic',
  "'l rst clkD mfsmd,model mfsmb abort last_in_inE.
  LastLogic rst clkD mfsmd,model mfsmb abort last_in_inE =
  \lt time .
  (last_in_inE t = (rst t) V ((clkD t) A (mfsmd model t)) V (mfsmb abort t))"
);;

let HoldLogic = new_definition ('Hold_Logic',
  "'l (cb ms:time->words) clkD sfsm_sal last_out_inS last_out_inR last_out_inE.
  HoldLogic cb_ms clkD sfsm_sal last_out_inS last_out_inR last_out_inE =
  \lt time .
  (last_out_inS t = sfsm_sal t) A
  (last_out_inR t = (clkD t) A ((cb_ms t = ^MEND) V (cb_ms t = ^MABORT))) A
  (last_out_inE t = (last_out_inS t) V (last_out_inR t))"
);;

let LastLogic = new_definition ('Last_Logic',
  "'l rst clkD mfsmd,model mfsmb abort last_in_inE.
  LastLogic rst clkD mfsmd,model mfsmb abort last_in_inE =
  \lt time .
  (last_in_inE t = (rst t) V ((clkD t) A (mfsmd model t)) V (mfsmb abort t))"
);;

let HoldLogic = new_definition ('Hold_Logic',
  "'l (cb ms:time->words) clkD sfsm_sal last_out_inS last_out_inR last_out_inE.
  HoldLogic cb_ms clkD sfsm_sal last_out_inS last_out_inR last_out_inE =
  \lt time .
  (last_out_inS t = sfsm_sal t) A
  (last_out_inR t = (clkD t) A ((cb_ms t = ^MEND) V (cb_ms t = ^MABORT))) A
  (last_out_inE t = (last_out_inS t) V (last_out_inR t))"
);;

let LastLogic = new_definition ('Last_Logic',
  "'l rst clkD mfsmd,model mfsmb abort last_in_inE.
  LastLogic rst clkD mfsmd,model mfsmb abort last_in_inE =
  \lt time .
  (last_in_inE t = (rst t) V ((clkD t) A (mfsmd model t)) V (mfsmb abort t))"
);;

let HoldLogic = new_definition ('Hold_Logic',
  "'l (cb ms:time->words) clkD sfsm_sal last_out_inS last_out_inR last_out_inE.
  HoldLogic cb_ms clkD sfsm_sal last_out_inS last_out_inR last_out_inE =
  \lt time .
  (last_out_inS t = sfsm_sal t) A
  (last_out_inR t = (clkD t) A ((cb_ms t = ^MEND) V (cb_ms t = ^MABORT))) A
  (last_out_inE t = (last_out_inS t) V (last_out_inR t))"
);;
Generation logic for cout_sel signal.

let Cout_Sel_Logic_SPEC = new_definition
('Cout_Sel_Logic_SPEC',
"! sfsm_s_cout_sel0 mfsm_m_cout_sel1 mfsm_m_cout_sel0 sfsm_sd0 sfsm_sd1 (cout_sel:time->wordn).
Cout_Sel_Logic_SPEC sfsm_s_cout_sel0 mfsm_m_cout_sel1 mfsm_m_cout_sel0 sfsm_sd0 sfsm_sd1 cout_sel =

let a1 = (ALTER (cout_sel t) 0 (sfsm_sd0 t))
in (ALTER a1 1 F))

let a2 = (ALTER (cout_sel t) 0 (mfsm_m_cout_sell t))
in (ALTER a2 1 (mfsm_m_cout_sell t))
);

Generation logic for srdy signal.

let Srdy_In.Logic_SPEC = new_definition
('Srdy_In.Logic_SPEC',
"! (cb ss:time->wordn) dfsm srdy.
Srdy_In.Logic_SPEC cb ss dfsm_srdy =

let a1 = (ALTER (cout_sel t) 0 (sfsm_sd0 t))
in (ALTER a1 1 F))

let a2 = (ALTER (cout_sel t) 0 (mfsm_m_cout_sell t))
in (ALTER a2 1 (mfsm_m_cout_sell t))
);

Input logic for C_wrdy, C_rrdy latches.

let Rdy_Logic_SPEC = new_definition
('Rdy_Logic_SPEC',
"! mfsm_md0 mfsm_md1 clkD write srdy wrdy_inD rrdy_inD.
Rdy_Logic_SPEC mfsm_md0 mfsm_md1 clkD write srdy wrdy_inD rrdy_inD =

(wrdy_inD t = (srdy t) \ (write t) \ (mfsm_md0 t) \ (clkD t))\ (rrdy_inD t = (srdy t) \ -(write t) \ (mfsm_md0 t) \ (clkD t))
);

Generation logic for l_srdy_out signal.

let ISrdy_Out_Logic_SPEC = new_definition
('ISrdy_Out_Logic_SPEC',
"! wrdyA_outQ rrdyA_outQ fsm_mabort cale_srdy_en isrdy_inD isrdy_inE.
ISrdy_Out_Logic_SPEC wrdyA_outQ rrdyA_outQ fsm_mabort cale_srdy_en isrdy_inD isrdy_inE =

(isrdy_inD t = (wrtyA_outQ t) V (rrdyA_outQ t) V (fsm_mabort t)) \ (isrdy_inE t = (cale_ t) V (srdy_en t))
);

Generation logic for CBss_out signal.
let CBss_Out_Logic_SPEC = new_definition
('CBss_Out_Logic_SPEC',
 "! (sfsm_ss:time->wordn) pmm_failure piu_valid cbss_out .
 CBss_Out_Logic_SPEC sfsm_ss pmm_failure piu_valid cbss_out =
 !t:time .
 (cbss_out t = (let al = (MALTER (cbss_out t) (1,0) (SUBARRAY (sfsm_ss t) (1,0)))
 in (ALTER al (2) ((ELEMENT (sfsm_ss t) (2)) \ (pmm_failure t) \ (piu_valid t))))")
);

Generation logic for CBms_out signal.

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

Generation logic for cout_1_le signal.

let Cout_1_Le_Logic_SPEC = new_definition
('Cout_1_Le_Logic_SPEC',
 "! (dfsm_master cout_0_le del dfsm_cout_1_le cout_1_le .
 Cout_1_Le_Logic_SPEC dfsm_master cout_0_le del dfsm_cout_1_le cout_1_le =
 !t:time .
 (cout_1_le t = ~(dfsm_master t) \ (dfsm_cout_1_le t) V (dfsm_master t) \ (cout_0_le_del t))")
);

Generation logic for iad_en signal.

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

Generation logic for c_pe_cnt signal.

let Pe_Cnt_Logic_SPEC = new_definition
('Pe_Cnt_Logic_SPEC',

106
"! clkD (sfsm_sparsity:time->bool) mfsm_mparity (cb_ss_in:time->wordn) c_pe_cnt.
Pe_cnt_Llogic_SPEC clkD sfsm_sparsity mfsm_mparity cb_ss_in c_pe_cnt =
|:time .
  (c_pe_cnt t = (clkD t) \A 
   (neg((sfsm_sparsity t) != (mfsm_mparity t)) \V ((SUBARRAY (cb_ss_in t) (1,0)) = WORDN 0)))"
);

% Generation logic for c_grant, c_busy signals.

let Grant_Llogic_SPEC = new_definition
("Grant_Llogic_SPEC",
"! (id:time->wordn) (rqX_time->wordn) busy grant .
Grant_Llogic_SPEC id rqX_busy grant =
|:time .
  (busy t = (ELEMENT (rqX_t) (3)) \V (ELEMENT (rqX_t) (2)) \V (ELEMENT (rqX_t) (1))) \A 
  (grant t = ((SUBARRAY (id t) (1,0)) = WORDN 0) \A (ELEMENT (rqX_t) (0))) \V 
  ((SUBARRAY (id t) (1,0)) = WORDN 1) \A (ELEMENT (rqX_t) (0)) \A (ELEMENT (rqX_t) (1))) \V 
  ((SUBARRAY (id t) (1,0)) = WORDN 2) \A (ELEMENT (rqX_t) (0)) \A (ELEMENT (rqX_t) (1))) \A 
  (ELEMENT (rqX_t) (2))) \V 
  ((SUBARRAY (id t) (1,0)) = WORDN 3) \A (ELEMENT (rqX_t) (0)) \A (ELEMENT (rqX_t) (1)) \A 
  (ELEMENT (rqX_t) (2)) \A (ELEMENT (rqX_t) (3)))"
);

% Generation logic for addressed signal.

let Addressed_Llogic_SPEC = new_definition
("Addressed_Llogic_SPEC",
"! (id:time->wordn) (source:time->wordn) addressed .
Addressed_Llogic_SPEC id source addressed =
|:time .
  (addressed t = (ELEMENT (id t) (0)) = (ELEMENT (source t) (10))) \A 
  (ELEMENT (id t) (1)) = (ELEMENT (source t) (11))) \A 
  (ELEMENT (id t) (2)) = (ELEMENT (source t) (12))) \A 
  (ELEMENT (id t) (3)) = (ELEMENT (source t) (13))) \A 
  (ELEMENT (id t) (4)) = (ELEMENT (source t) (14))) \A 
  (ELEMENT (id t) (5)) = (ELEMENT (source t) (15)))"
);

% Generation logic for Disable_writes signal.

let D_Writes_Llogic_SPEC = new_definition
("D_Writes_Llogic_SPEC",
"! dfsm_slave (chan_id:time->wordn) (source:time->wordn) disable_writes .
D_Writes_Llogic_SPEC dfsm_slave chan_id source disable_writes =
|:time .
  (disable_writes t = (dfsm_slave t) \A (ELEMENT (chan_id t) (0)) \A (ELEMENT (source t) (6))) \A 
  (ELEMENT (chan_id t) (1)) \A (ELEMENT (source t) (7))) \A 
  (ELEMENT (chan_id t) (2)) \A (ELEMENT (source t) (8)))"
\[ \Lambda -((\text{ELEMENT (chan_id t) (3)}) \land (\text{ELEMENT (source t) (9)})) \]

% Generation logic for c_pe signal.

let Parity_Decode_Engine_SPEC = new_definition
('Parity_Decode_Engine_SPEC',
"! rep cad_in cad_in_dec cad_in_det .
Parity_Decode_Engine_SPEC rep cad_in cad_in_dec cad_in_det =
  \begin{align*}
  \text{cad_in_dec t} &= (\text{Par_Dec rep (cad_in t)}) \land \\
  \text{cad_in_det t} &= (\text{Par_Det rep (cad_in t)})
  \end{align*}
);

% Input logic for C_parity latch.

let Parity_Signal_Inputs_SPEC = new_definition
('Parity_Signal_Inputs_SPEC',
"! rst cad_in_dec clkD c_pe_cnt reset_parity
  c_parity_inS c_parityinS c_parity_inE.
Parity_Signal_Inputs_SPEC rst cad_in_dec clkD c_pe_cnt reset_parity
  c_parity_inS c_parityinS c_parity_inE =
  \begin{align*}
  \text{c_parity_inS t} &= (\text{cad_in_det t}) \land \text{(clkD t)} \land (\text{c_pe_cnt t}) \land \\
  \text{c_parity_inR t} &= (\text{rst t}) \lor (\text{reset_parity t}) \land \\
  \text{c_parity_inE t} &= (\text{c_parity_inS t}) \lor (\text{c_parity_inR t})
  \end{align*}
);

% C-Bus input latches.

let CB_In_Latches_SPEC = new_definition
('CB_In_Latches_SPEC',
"! clkA clkB rst cad_in_dec:time->wordn cin_0_le cin_1_le cin_2_le cin_3_le cin_4_le
  source:time->wordn (sizewrbe:time->wordn) iad_preout
  c_source c_data_in c_sizewrbe c_iad_preout .
CB_In_Latches_SPEC clkA clkB rst cad_in_dec cin_0_le cin_1_le cin_2_le cin_3_le cin_4_le
  source sizewrbe iad_preout
  c_source c_data_in c_sizewrbe c_iad_preout =
  \begin{align*}
  \\text{((clkA t) \implies})
  \\text{((c_source t+1) = c_source t) \land} \\
  \\text{(c_data_in t+1) = c_data_in t) \land} \\
  \\text{(c_sizewrbe t+1) = c_sizewrbe t) \land} \\
  \\text{(c_iad_preout t+1) = (cin_2_le t) \implies (c_data_in t) \lor (c_iad_preout t))} \land \\
  \\text{((clkB t) \implies})
  \\text{((c_source t+1) = (rst t) \implies \text{WORDN 0})} \\
  \\text{(cin_3_le t) \implies (cad_in_dec t) \land} \\
  \\text{(c_source t) \land}
  \end{align*}
\)
let BE_Out_Logic_SPEC = new_definition
  ('BE_Out_Logic_SPEC',
   "! (sizewrbe:time->wordn) hlda be_out.
   BE_Out_Logic_SPEC sizewrbe hlda be_out
   = l:t:time.
   (! (hlda t) => (be_out t = SUBARRAY (sizewrbe t) (9,6)))");

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

let CB_Out_Logic_SPEC = new_definition
  ('CB_Out_Logic_SPEC',
   "! rep clkA clkB (iad_in:time->wordn) ccr time->wordn) dfsm_cout_0_le cout_1_le dfsm_mrequest cout sel cad_preout
   C_iad_in C_a1a0 C_a3a2.
   CB_Out_Logic_SPEC rep clkA clkB iad_in ccr dfsm_cout_0_le cout_1_le dfsm_mrequest cout sel cad_preout
   C_iad_in C_a1a0 C_a3a2
   = l:t:time.
   (! (clkA t) =>
   (! (C_iad_in (t+1) = C_iad_in t) \n   

109
let C_Block_SPEC = new_definition
c ("C_Block_QUAL", "C_Block_QUAL", "C_Block_QUAL")

%...
C_efsm_state \ C_efsm_cale_ \ C_efsm_last_ \ C_efsm_male_ \ C_efsm_rale_ \ C_efsm_srdy_ \ C_efsm_rst
C_efsm_stateA \ C_efsm_srdy_en \ efsm_srdy_en \ A

(CDFSM_SPEC dfsm_srdy ClkD clkA_outQ write size wrbe sfsm_sidle sidle_del_outQ sfsm_slock
sfsm_sd1 sfsm_sdl sfsm_sale sfsm_sid sfsm_sd0 sfsm_sack mfsn_midle mfsnodel_outQ
mfsn_sd3 mfsn_sd2 mfsn_sd1 mfsn_sd0 mfsn_md0 mfsn_md1 mfsn_md0 1_cale_ 1_srdy_in
dfsm_master dfsm_slave dfsm_cin_0_le dfsm_cin_1_le dfsm_cin_3_le dfsm_cin_4_le
dfsm_cout_0_le dfsm_cout_1_le dfsm_cad_en dfsm_male, dfsm_rale \ dfsm_srdy_)

);;
close_theory();
B.5 SU_Cont Specification

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

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

system 'rm s_block.th';

new_theory 's_block';

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

let s_state_ty
= ":(sfsm-ty#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#
  bool#bool#wordn#wordn#bool#bool#
  s_fsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
  wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#"
;

let s_state = 
"((S_fsm_stateA,
  S_fsm_sn, S_fsm_so, S_fsm_srcp, S fsm_sdi, S fsm_srp, S fsm_src0, S fsm_src1,
  S fsm_spf, S fsm_sc0f, S fsm_scl1f, S fsm_spnf, S fsm_spn, S fsm_n, S fsm_sns,
  S fsm_sca, S soft_shot, S soft_shot_delA, S soft_cntA, S delayA, S instant, S cpu_histA,
  S fsm_state, S fsm_rst, S fsm_delay6, S fsm_delay17, S fsm_bothbad, S fsm_bypass,
  S soft_shot_del, S soft_cnt, S delay, S bad_cpu0, S bad_cpu1, S reset_cpu0, S reset_cpu1,
  S pmm_fail, S cpu0_fail, S cpu1_fail, S cpu_hist, S piu_fail)
  :s_state_ty);"
;

let s_env_ty
= ":(bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
  bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#"
;

let s_env = "((ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, FailureO., FailureI_)
  :s_env_ty);"
;

let s_out_ty
= ":(wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
  bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#"
;

let s_out = "((S_state, Reset_cport, DisableInt, Reset_piui, Reset_cpu0, Reset_cpu1, Cpu_hist,
  Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail)
  :s_out_ty);"
;

Input logic for S soft_shot latch.

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

114
(soft_cnt_inU t = gcrh t ∧ ~gcr1 t)

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

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

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

let Dis_Int_Out_SPEC = new_definition
('Dis_Int_Out_SPEC',
"! restart normal delay disable_int_in disable_int_out.
Dis_Int_Out_SPEC restart normal delay disable_int_in disable_int_out =
(I t:time. (disable_int_out t = -restart t ∧ normal t ∧ (ELEMENT (delay t) (6)) ∧ disable_int_in t))")

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

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

let Cpu_Ok_SPEC = new_definition
('Cpu_Ok_SPEC',
"! soft_cnt cpu0_fail cpu1_fail failure0_fail failure1_fail cpu0_ok cpu1_ok.
Cpu_Ok_SPEC soft_cnt cpu0_fail cpu1_fail failure0_fail failure1_fail cpu0_ok cpu1_ok =
(1 t:time. (cpu0_ok t = ((soft_cnt t) = WORDN 5) ^ cpu0_fail t ^ failure0 Fail t) ^
(cpu1_ok t = ((soft_cnt t) = WORDN 5) ^ cpu1_fail t ^ failure1 Fail t))"
);

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

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

%...........................................................................................................

Startup-controller controller state machine.
let FSM_SPEC = new_definition
('FSM_SPEC',
"1 clkA clkB rst_in delay_in delay17_in bothbad_in bypass_in
  state delay6 delay17 bothbad bypass
stateA sn so srcp sdi src0 src1 spf sc0f sclf spmf sb src sec srcs
stateA_out sn_out so_out srcp_out sdi_out src0_out src1_out spf_out
  sc0f_out sc1f_out spmf_out sb_out src_out sec_out srcs_out
FSM_SPEC clkA clkB rst_in delay_in delay17_in bothbad_in bypass_in
  state delay6 delay17 bothbad bypass
stateA sn so srcp sdi src0 src1 spf sc0f sclf spmf sb src sec srcs
stateA_out sn_out so_out srcp_out sdi_out src0_out src1_out spf_out
  sc0f_out sc1f_out spmf_out sb_out src_out sec_out srcs_out

!t:time.

((clkA t) =>
  ((state (t+1) = state t) \A
   (rst (t+1) = rst t) \A
   (delay6 (t+1) = delay6 t) \A
   (delay17 (t+1) = delay17 t) \A
   (bothbad (t+1) = bothbad t) \A
   (bypass (t+1) = bypass t) \A
   (stateA (t+1) =
     ((rst t) => SSTART \A
      ((state t) = SSTART) => SRA \A
      ((state t) = SRA) => ((delay6 t) => ((bypass t) => SO \A SPF) \A SRA) \A
      ((state t) = SPF) => SC0F \A
      ((state t) = SC0F) => ((delay17 t) => SC0F \A SC1F) \A
      ((state t) = SC1F) => ST \A
      ((state t) = ST) => SCII \A
      ((state t) = SCII) => ((delay17 t) => SC1F \A SCII) \A
      ((state t) = SC1F) => SS \A
      ((state t) = SS) => ((bothbad t) => SSTOP \A SCS) \A
      ((state t) = SSTOP) => SSTOP \A
      ((state t) = SCS) => ((delay6 t) => SN \A SCS) \A
      ((state t) = SN) => ((delay17 t) => SO \A SN) \A SO)) \A
   (sn (t+1) = (stateA (t+1) = SN)) \A
   (so (t+1) = (stateA (t+1) = SO)) \A
   (srcp (t+1) = (-(stateA (t+1) = SO) \A (-(state t) = SSTOP)) \V ((state t) = SRA)) \A
   (sdi (t+1) = ((-(stateA (t+1) = SO) \A (-(state t) = SSTOP)) \V ((state t) = SRA)) \A
   (src0 (t+1) = (-(stateA (t+1) = SSTOP)) \A (-(state t) = SSTOP)) \V ((state t) = SRA)) \A
   (src1 (t+1) = (-(stateA (t+1) = ST) \A (-(state t) = SSTOP)) \V ((state t) = SRA)) \A
   (spf (t+1) = ((-(stateA (t+1) = ST) \A (-(state t) = SSTOP)) \V ((state t) = SRA)) \A
   (spf (t+1) = (-(stateA (t+1) = ST) \A (-(state t) = SSTOP)) \V ((state t) = SRA)) \A
   (spmf (t+1) = (-(stateA (t+1) = ST) \A (-(state t) = SSTOP)) \V ((state t) = SRA)) \A
   (spmf (t+1) = (-(stateA (t+1) = ST) \A (-(state t) = SSTOP)) \V ((state t) = SRA)) \A
   (sb (t+1) = (-(stateA (t+1) = SSTART)) \A
   (sb (t+1) = (-(stateA (t+1) = SSTART)) \A
   (sb (t+1) = (-(stateA (t+1) = SSTART)) \A
   (sb (t+1) = (-(stateA (t+1) = SSTART)) \A
   (sb (t+1) = (-(stateA (t+1) = SSTART)) \A
   (sb (t+1) = (-(stateA (t+1) = SSTART)) \A
   (sb (t+1) = (-(stateA (t+1) = SSTART)) \A
   (sb (t+1) = (-(stateA (t+1) = SSTOP)) \A

117
(stateA (t+1) = SS) V (((state t) = SCS) \& (delay6 t))) V

(sec (t+1) = (((stateA (t+1) = SSTOP) \& \neg(stateA (t+1) = SO)) V ((state t) = (SN))) \&

(srs (t+1) = (((state t) = SPF) \& \neg(rst t)) V (((state t) = ST) \& \neg(rst t)))) \&

(scs (t+1) = (stateA (t+1) = SCS))) \&

((clkB t) ==>

((state (t+1) = stateA t) \&

(rst (t+1) = rst_in t) \&

(delay6 (t+1) = ELEMENT (delay_in t) (6)) \&

(delay17 (t+1) = delay17_in t) \&

(bothbad (t+1) = bothbad_in t) \&

(bypass (t+1) = bypass_in t) \&

(sn (t+1) = sn t) \&

(so (t+1) = so t) \&

(srcp (t+1) = srcp t) \&

(sdi (t+1) = sdi t) \&

(srp (t+1) = srp t) \&

(src0 (t+1) = src0 t) \&

(src1 (t+1) = src1 t) \&

(spf (t+1) = spf t) \&

(sc0f (t+1) = sc0f t) \&

(sc1f (t+1) = sc1f t) \&

(appmf (t+1) = appmf t) \&

(ab (t+1) = ab t) \&

(arc (t+1) = arc t) \&

(sec (t+1) = sec t) \&

(srs (t+1) = srs t) \&

(scs (t+1) = scs t))) \&

(let a0 = (ALTER (stateA_out t) (0)

((stateA (t+1) = SRA) V (stateA (t+1) = SPF) V (stateA (t+1) = ST) V

(stateA (t+1) = SCI) V (stateA (t+1) = SCS) V (stateA (t+1) = SN) V

(stateA (t+1) = SO)))))

in

(let a1 = (ALTER a0 (1)

((stateA (t+1) = SPF) V (stateA (t+1) = SC0) V (stateA (t+1) = SC0) V

(stateA (t+1) = ST) V (stateA (t+1) = SSTOP) V (stateA (t+1) = SO)))))

in

(let a2 = (ALTER a1 (2)

((stateA (t+1) = SC0) V (stateA (t+1) = ST) V (stateA (t+1) = SCI) V

(stateA (t+1) = SCI) V (stateA (t+1) = SS) V (stateA (t+1) = SSTOP) V

(stateA (t+1) = SCS)))

in

(let a3 = (ALTER a2 (3)

((stateA (t+1) = SS) V (stateA (t+1) = SSTOP) V (stateA (t+1) = SCS) V

(stateA (t+1) = SN) V (stateA (t+1) = SO)))))

in

((stateA_out t = a3)))))) \&

(sn_out t = sn (t+1)) \&

(so_out t = so (t+1)) \&

(srcp_out t = srcp (t+1)) \&

(sdi_out t = sdi (t+1)) \&

(srp_out t = srp (t+1)) \&

(src0_out t = src0 (t+1)) \&

(src1_out t = src1 (t+1)) \&

(spf_out t = spf (t+1)) \&

...
let S_Block_SPEC = new_definition
("S_Block_SPEC",
"S FSM_stateA S FSM_state, (time->sfsm_ty)
S soft_cntinA S soft_cont S delay : (time->wordn)
S FSM sn S FSM so S FSM srcp S FSM_sdi S FSM srp S FSM src0 S FSM src1 S FSM spf S FSM scOf
S FSM sclf S FSM spmf S FSM sb S FSM src S FSM sec S FSM srs S FSM ssc
S soft_shot S soft_shot_delA S instant S cpu_histA
S FSM rst S FSM_delay6 S FSM_delay17 S FSM bothbad S FSM bypass
S soft_shot_del S bad_cpu0 S bad_cpu1 S reset_cpu0 S reset_cpu1 S pmnm_fail S cpu0_fail S cpu1_fail
S piu_fail S cpu Hist : (time->bool))
(ClkA ClkB Rst Bypass Test Gcrh Gcr1 Failure0 Failure1 : (time->bool))
(S state : (time->wordn))
(Reset_cport Disable int Reset_piu Reset_cpu0 Reset_cpu1 Cpu_hist Piu Fail Cpu0_fail Cpu1_fai
Pnmn_fail : (time->bool))
S_Block_SPEC (S FSM stateA, S FSM so, S FSM sn, S FSM srp, S FSM sec, S FSM srs, S FSM src, S FSM src1,
S FSM spf, S FSM scOf, S FSM sclf, S FSM spmf, S FSM sb, S FSM src, S FSM sec, S FSM srs,
S FSM ssc, S soft_shot, S soft_shot_delA, S soft_cntA, S delayA, S instant, S cpu_histA,
S FSM state, S FSM sno S FSM srcp S FSM sdi S FSM srp S FSM src0 S FSM src1 S FSM spf S FSM scOf
S FSM sclf S FSM spmf S FSM sb S FSM src S FSM sec S FSM srs S FSM ssc
S soft_shot S soft_shot_delA S instant S cpu_histA
S FSM rst S FSM_delay6 S FSM_delay17 S FSM bothbad S FSM bypass
S soft_shot_del S soft_cnt S delay S bad_cpu0 S bad_cpu1 S reset_cpu0 S reset_cpu1,
S pmnm_fail S cpu0_fail S cpu1_fail S cpu_hist S piu_fail)
(ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcr1, Failure0, Failure1)
(S state, Reset_cport, Disable int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist,
Piu_fail, Cpu0_fail, Cpu1_fail, Pnmn_fail)
(
? FSM_delay17 FSM bothbad
FSM sn FSM so FSM sdi FSM src0 FSM src1 FSM spf FSM scOf FSM sclf FSM spmf FSM sb
FSM src FSM sec FSM srs FSM ssc NC
soft_shot_inD soft_shot_outQ soft_shot_del_outQ
soft_cnt_inL soft_cnt_inU soft_cnt_inR soft_cnt_outQ
delay_inL delay_inU delay_inR delay_outQ instant_inD instant_outQ
bad_cpu0_inS bad_cpu0_inR bad_cpu0_inE bad_cpu0_outQ reset_cpu0_inD
bad_cpu1_inS bad_cpu1_inR bad_cpu0_inE bad_cpu0_outQ reset_cpu1_inD
bad_cpu1 ok cpu0 ok
pmnm_fail_inS pmnm_fail_inR pmnm_fail_inE cpu0_fail_inS cpu0_fail_inR cpu0_fail_inE
cpu1_fail_inS cpu1_fail_inR cpu1_fail_inE piu_fail_inS piu_fail_inR piu_fail_inE.
(Scnt_In SPEC Gcrh Gcr1 soft_shot_inD soft_cnt_inL)
\)
(DLAT_SPEC soft_shot_inD ClkA S soft_shot soft_shot_outQ)
(DFF_SPEC soft_shot_outQ ClkA S soft_shot_del S soft_shot_delA soft_shot_del_outQ)
(Scnt_In SPEC soft_shot_outQ soft_shot_delA soft_shot_del_outQ)
(UPRCNT_SPEC 2 (GNDN) 2)
soft_outQ NC) \n\n(Delay_In SPEC fsm_scs delay_outQ fsm_src delay inR) \n\n(UPRCNT_SPEC 17 (GNDN 17) delay_inL delay_inU delay inR ClkA S_delay S_delayA delay_outQ NC) \n\n(Muxes_SPEC delay outQ Test instant inD fsm_delay17) \n\n(DLAT_SPEC instant inD ClkA S instant instant outQ) \n\n(Dis_Int_Out SPEC instant outQ fsm sn delay outQ fsm_sdi Disable_int) \n\n(AND2_SPEC Cpu0 fail Cpu1 fail fsm_bothbad) \n\n(Bad_Cpu_In SPEC fsm sn fsm so Cpu0 fail Cpu1 fail fsm sb \n\nbad_cpu0_inS bad_cpu0_inR bad_cpu0_inE \n\nbad_cpu1_inS bad_cpu1_inR bad_cpu1_inE) \n\n(DSRELAT_SPEC GND bad_cpu0_inS bad_cpu0_inR bad_cpu0_inE ClkB S bad_cpu0 bad_cpu0_outQ) \n\n(DSRELAT_SPEC GND bad_cpu1_inS bad_cpu1_inR bad_cpu1_inE ClkB S bad_cpu1 bad_cpu1_outQ) \n\n(AND2_SPEC bad_cpu0_outQ fsm_src0 reset_cpu0_inD) \n\n(AND2_SPEC bad_cpu1_outQ fsm_src1 reset_cpu1_inD) \n\n(DLAT_SPEC reset_cpu0_inD ClkB S reset_cpu0 Reset_cpu0) \n\n(DLAT_SPEC reset_cpu1_inD ClkB S reset_cpu1 Reset_cpu1) \n\n(AND3_SPEC Reset_cpu0 Reset_cpu1 Bypass cpu_hist inD) \n\n(DFF_SPEC cpu_hist inD ClkB S cpu_hist A S cpu_hist B) \n\n(Fail_In SPEC fsm sb fsm_spf fsm_spf Bypass cpu0 ok cpu1 ok \n\npmn_fail_inS pmn_fail_inR pmn_fail_inE cpu0_fail_inS cpu0_fail_inR cpu0_fail_inE \n\ncpu1_fail_inS cpu1_fail_inR cpu1_fail_inE piu_fail_inS piu_fail_inR piu_fail_inE) \n\n(DSRELAT_SPEC GND pmn_fail_inS pmn_fail_inR pmn_fail_inE ClkB S pmn_fail Pmn_fail) \n\n(DSRELAT_SPEC GND cpu0_fail_inS cpu0_fail_inR cpu0_fail_inE ClkB S cpu0_fail Cpu0_fail) \n\n(DSRELAT_SPEC GND cpu1_fail_inS cpu1_fail_inR cpu1_fail_inE ClkB S cpu1_fail Cpu1_fail) \n\n(DSRELAT_SPEC GND piu_fail_inS piu_fail_inR piu_fail_inE ClkB S piu_fail Piu_fail) \n\n(Cpu_Ok_SPEC soft_outQ fsm_sc0f fsm_sc1f Failure0 Failure1 cpu0 ok cpu1 ok) \n\n(FSM_SPEC ClkB Rst delay_outQ fsm_del17 fsm_bothbad Bypass \n\nS fsm state S fsm rst S fsm delay6 S fsm delay17 S fsm bothbad S fsm_bypass \n\nS fsm state A S fsm_sc A S fsm_sc0 S fsm_sc1 S fsm_src S fsm_src0 S fsm_src1 \n\nS fsm_spf S fsm_sc0f S fsm_sc1f S fsm_spnf S fsm sb S fsm src S fsm sec S fsm src \n\nS fsm scs \n\nS state fsm sn fsm so Reset_cport fsm_sdi Reset_piu fsm Src0 fsm Src1 fsm spf \n\nS fsm_sc0f S fsm_sc1f S fsm_src0 S fsm src S fsm sec S fsm src S fsm scs))

);;

close_theory();

120
Appendix C  ML Source for the Phase-Level Specification of the PIU Ports.

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

C.1 P Port Specification

```ml
set_search_path (search_path() @ ["/home/utan3/dfura/ftep/piu/hol/lib/"]);

new_theory 'p_phase';

map new__parent ['paux_def';'aux_def';'affay_def';'wordn_def'];

let p_state ty = ":(pfsm_ty#bool#bool#bool#wordn#wordn#bool#bool#bool#wordn#bool#bool#bool#bool#bool#bool#bool)";;
let p__env_ty = ":(bool#bool#bool#wordn#bool#bool#wordn#bool#bool#wordn#bool#bool#bool)";;
let p_env = "((ClkA, ClkB, Rst, L_ad__in, L_ads_, L_den_, L_be_, L_wr, L_lock_, L_ad__in, L_cga, L_hold_, L_srdy_)
  :bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;

let p_out_ty = ":(wordn#bool#bool#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;
let p_out = "((L_ad__out, L_ready_, L_ad__data_out, L_ad__addr_out, L_be_, L_ral_, L_male_, L_cra, L_cale_,
  L_mrdy_, L_last_, L_hlda_, L_lock_)
  :wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;

let PH_A_inst_def = new_definition
```

121
('PH_A inst',
"1 (P_fsm_state P..fsm__stateA :pfsm_ty)
(P_fsm_astate P__fsm dstate P__fsm_hlda_ P_destl P_wr P_loadA P__downA :bool)
(P_fsm_rst P_fsm_mrqt P fsm_sack P fsm_cngt_ P fsm_cngt_ P fsm_lock_ P___rqt P_load :bool)
(P_down P lock__inh_ P male_ P rale_ :bool)
(P_wr_data P be_ P be_n_ P sizeA P size:wordn)

(PH_A..inst (P._fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda.., P..wr_dat& P. addr, P_destl, P_be,
(P_wr, P_be_n, P.sizeA, P/loadA, P_downA, P_fsm_state, P_fsm_hold_, P_fsm_lock_,
P_fsm__inh_, P._male_, P rale_)

let new_P..fsm_stateA =

((P_fsm_rst) => PA)
((P_fsm_state = PH) => PA)
((P_fsm_state = PA) =>

((P_fsm_mrqt V (~P_fsm_cngt_ A -P_fsm_cngt_)) => PD)
((P_fsm_lock_ A -P_fsm_hold_) => PH)

((P_fsm_state = PD) =>

((P_fsm_sack A P_fsm_hold_) V (P_fsm_sack A -P_fsm_hold_ A -P_fsm_lock_)) => PA)
((P_fsm_sack A -P_fsm_hold_ A P_fsm_lock_) => PH)

(let new_P_fsm_astate = (new_P__fsm_hlda_ = PA) in
let new_P fsm_hlda_ = (new_P_fsm_stateA = PD) in
let new_P_wr_data = L_ad_in in
let new_P_addr = ((-P_rqt) => (SUBARRAY L_ad_in (25,0)) P_addr in
let new_P__destl = ((-P_rqt) => (ELEMENT L_ad_in (31)) P__destl in
let new_P_be_ = ((-P_rqt) => L_be_ P be_ in
let new_P_wr = ((-P_rqt) => L.wr P_wr in
let new_P_be_n = L_be_in
let new_P_loadA = P_load in
let new_P_downA = P_down in
let new_P_sizeA = P.size in
let new_P_fsm_state = P fsm_state in
let new_P_fsm_rst = P fsm_rst in
let new_P fsm_mrqt = P fsm_mrqt in
let new_P fsm_sack = P fsm_sack in
let new_P fsm_cngt_ = P fsm_cngt_ in
let new_P fsm_cngt_ = P fsm_cngt_ in
let new_P fsm_hold_ = P fsm_hold_ in
let new_P fsm_lock_ = P fsm_lock_ in
let new_P rqt = P rqt in
let new_P size = P.size in
let new_P loadA = P_load in
let new_P down = P_down in
let new_P lock__inh_ = P lock__inh_ in
let new_P._male_ = P._male_ in
let new_P rale_ = P rale_ in

(ClkA ClkB Rst L_ads_ L_den_ L_wr L_lock_ L_cgnt_ L_hold_ L_srdy_ :bool) (L_ad_in L_be_ L_ad_in :wordn).

(PH_A inst (P._fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P__destl, P_be_, P_wr, P_be_n, P.sizeA, P_loadA, P__downA, P_fsm_state, P_fsm_hold_, P_fsm_lock_, P_rqt, P_size, P_load, P__down, P_lock_,
P_fsm__inh_, P._male_, P rale_)

ClkB Rst L_ads_ L_den_ L_wr L_lock_ L_cgnt_ L_hold_ L_srdy_ :bool) (L_ad_in L_be_ L_ad_in :wordn).

let new_P..fsm_stateA =

((P_fsm_rst) => PA)
((P_fsm_state = PH) => PA)
((P_fsm_state = PA) =>

((P_fsm_mrqt V (~P_fsm_cngt_ A -P_fsm_cngt_)) => PD)
((P_fsm_lock_ A -P_fsm_hold_) => PH)

((P_fsm_state = PD) =>

((P fsm_sack A P fsm_hold_) V (P fsm_sack A -P_fsm_hold_ A -P fsm_lock_)) => PA)
((P fsm_sack A -P_fsm_hold_ A P fsm_lock_) => PH)

(let new_P fsm_stateA = (new_P fsm_stateA = PA) in
let new_P fsm_stateA = (new_P fsm_stateA = PD) in
let new_P fsm_hlda_ = (new_P fsm_stateA = PH) in
let new_P wr_data = L_ad_in in
let new_P addr = ((-P rqt) => (SUBARRAY L ad_in (25,0)) P addr in
let new_P destl = ((-P rqt) => (ELEMENT L ad_in (31)) P destl in
let new P be_ = ((-P rqt) => L be_ P be_ in
let new P wr = ((-P rqt) => L wr P wr in
let new P be_n = L be_in
let new P loadA = P load in
let new P downA = P down in
let new P sizeA = P size in
let new P fsm_state = P fsm_state in
let new P fsm_rst = P fsm_rst in
let new P fsm_mrqt = P fsm_mrqt in
let new P fsm_sack = P fsm_sack in
let new P fsm_cngt_ = P fsm_cngt_ in
let new P fsm_cngt_ = P fsm_cngt_ in
let new P fsm_hold_ = P fsm_hold_ in
let new P fsm_lock_ = P fsm_lock_ in
let new P rqt = P rqt in
let new P size = P size in
let new P loadA = P load in
let new P down = P down in
let new P lock__inh_ = P lock__inh_ in
let new P._male_ = P._male_ in
let new P rale_ = P rale_ in

122
let PH_A_out_def = new_definition
('PH_A_out',

"! (P_fsm_state P_fsm_stateA :pfsm_ty)
(P_fsm_astate P fsm_dstate P_fsm_hlda_ P_destl P_wr_loadA P_downA :bool)
(P_fsm_rst P_fsm_mrqt P_fsm_sack P_fsm_crgt_ P_fsm_crqt_ P_fsm_hold_ P_fsm_lock_ P_rqt P_load :bool)
(P_down P_lock _P_lock_inh_ P_male_ P_rale :bool)
(P_wr_data P_addr P_be P_be_n_ P_sizeA P_size :wordn)
(CLKA ClkB Rst L_ad_in L_den L_wr L_lock L_crgt_ L_hold_ L_sdry_ :bool) (L_ad_in L_be L_ad_in :wordn).

PH_A_out (P_fsm_stateA, P_fsm_astate, P_fsm_dstate, P_fsm_hlda_, P_wr_data, P_addr, P_destl, P_be_,
P_wr, P_be_n, P_sizeA, P_loadA, P_downA, P_fsm_state, P_fsm_rst, P_fsm_mrqt, P_fsm_sack, P_fsm_crgt_, P_fsm_crqt_, P_fsm_hold_, P_fsm_lock_, P_rqt, P_size, P_load, P_down, P_lock_,
P_lock_inh_, P_male_, P_rale_)
(CLKA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be_, L_wr, L_lock_, L_ad_in, L_crgt_, L_hold_, L_sdry_)

let new_P_fsm_stateA =
((P_fsm_rst) => PA !
 ((P_fsm_state = PH) => ((P_fsm_hold_) => PA ! PH) !
 ((P_fsm_state = PA) =>
   ((P_fsm_mrqt V (~P_fsm_crqt_ V ~P_fsm_crgt_)) => PD !
   ((P_fsm_lock_ V ~P_fsm_hold_) => PH ! PA) !
   ((P_fsm_state = PD) =>
     ((P_fsm_sack V P_fsm_hold_) V (P_fsm_sack V ~P_fsm_hold_ V ~P_fsm_lock_)) => PA !
     ((P_fsm_sack V ~P_fsm_hold_ V ~P_fsm_lock_) => PH ! PD) ! (P_ILL))) in

let new_P_fsm_astate = (new_P_fsm_stateA = PA) in
let new_P_fsm_dstate = (new_P_fsm_stateA = PD) in
let new_P_fsm_hlda_ = ~(new_P_fsm_stateA = PH) in
let new_P_wr_data = L_ad_in in
let new_P_addr = (L_rqt) => (SUBARRAY L_ad_in (25,0)) ! P_addr in
let new_P_destl = (L_rqt) => (ELEMENT L_ad_in (31)) ! P_destl in
let new_P_be = (L_rqt) => L_be_ ! P_be_ in
let new_P_wr = (L_rqt) => L_w ! P_wr in
let new_P_be_n = L_be_ in
let new_P_loadA = P_load in
let new_P_downA = P_down in
let new_P_sizeA = P_size in
let new_P_fsm_state = P_fsm_state in
let new_P_fsm_rst = P_fsm_rst in
let new_P_fsm_mrqt = P_fsm_mrqt in
let new_P_fsm_sack = P_fsm_sack in
let new_P_fsm_crgt_ = P_fsm_crgt_ in
let new_P_fsm_crqt_ = P_fsm_crqt_ in
let new_P_fsm_hold_ = P_fsm_hold_ in
let new_P_fsm_lock_ = P_fsm_lock_ in
let new_P_rqt = P_rqt in

123
let new_P_size = P_size
let new_P_load = P_load
let new_P_down = P_down
let new_P_lock = P_lock
let new_P_lock_inh = P_lock_inh
let new_P_male = P_male
let new_P_nde = P_nde
let
new_P_iock = P_lock
let new_P_lock_inh_ffi = P_lock_inh
let new_P_male = P_male
let new_P_nde = P_nde
let
p_all = (-L_ads_ A L_den_) in
let
p_sack = ((new_P_size A ((new_P_down A => WORDN 1 I WORDN 0)) A -L_srdy_ A new_P_fsm_dstate)
let L_ad_out = ((new_P fsm_astate A P fsm_dstate A new_P_wr) => L_ad out I ARBN)
let I_ad_addr_out = ((new_P fsm_astate) => od4 1 ARBN)
let I_ad_data_out = ((new_P fsm_dstate A new_P_wr) => new_P wr I ARBN)
let I_be_ = ((new_P fsm_hlda_) => ((new_P fsm_a_ate) => new_P_be_ I new_P_be_n_) I ARBN)
let I__rale._ = ((new P fsm_ldda_.) => (~new P_destl A ((SUBARRAY new_P_addr (25,24)) = OVORDN 3)) A new_P_fsm_a_ate A new_P__rqt)
let I_male_ = ((new P fsm_hlda_) => (~new P_destl A (new_P fsm_a_ate) A new_P_fsm_a_ate A new_P_rqt)
let I__male__ = ~(new_P_destl A new_P_rqt)
let I__cale_ = ~(-I_cgnt_ A new P fun astate A I_hold_)
let I_mrdy_ = ((new P fsm_hlda_) => F I ARBN)
let I_last_ = ((new_P fsm_hlda_) => (new_P_size A (new_P fsm_a_ate) => WORDN 1 I WORDN 0)) I ARBN)
let I_lock_ = ~(-new_P lock_ A new_P lock_inh_)

(L_ready_, L_last_, L_be_, L_mrdy_, L_ad_data_out, L_ad_addr_out, L_hlda_, L_lock_, L_cale_, L_male_, L_rale_, L_cqrt_, L_ad_out)"

let PH_B_inst_def = new_definition
'PH_B_inst',

"! (P fsm state P fsm_state A :pfsm ty)
(P fsm astate P fsm_dstate P fsm_hlda P destl P wr P loadA P downA :bool)
(P fsm rst P fsm_mrqt P fsm_sack P fsm cgnt P fsm_cqrt P fsm_hold P fsm_lock P rqt P load :bool)
(P down P lock P lock_inh P male P rale :bool)
(P wr data P addr P be P be_n P sizeA P size B wordn)
(ClkA ClkB Rst L_ads L_den L wr L lock L cgnt L hold L srdy :bool) (L ad in L be L ad in :wordn).

PH_B_inst (P fsm state A, P fsm dstate, P fsm hlda, P destl, P wr, P load, P down, P lock, P lock_inh, P male, P rale, P wr data, P addr, P be, P be_n, P sizeA, P size B wordn)

let p ale = (~L_ads A L_den_ in

let
P size = P size
let
P load = P load
let
P down = P down
let
P lock = P lock
let
P lock_inh = P lock_inh
let
P male = P male
let
P_nde = P_nde
let

p_all = (-L_ads_ A L_den_)

p_sack = ((new_P_size A (new_P_down A => WORDN 1 I WORDN 0)) A -L_srdy_ A new_P_fsm_dstate)

let L_ad_out = ((new_P fsm_astate A new_P_fsm_dstate A new_P_wr) => L_ad out I ARBN)

let I_ad_addr_out = ((new_P fsm_astate) => od4 1 ARBN)

let I_ad_data_out = ((new_P fsm_dstate A new_P_wr) => new_P wr I ARBN)

let I_be_ = ((new_P fsm_hlda_) => ((new_P fsm_astate) => new_P_be_ I new_P_be_n_) I ARBN)

let I__rale._ = ((new P fsm_ldda_.) => (~new P_destl A ((SUBARRAY new_P_addr (25,24)) = OVORDN 3)) A new_P_fsm_a_ate A new_P__rqt)

let I_male_ = ((new P fsm_hlda_) => (~new P_destl A (new_P fsm_a_ate) A new_P_fsm_a_ate A new_P_rqt)

let I__male__ = ~(new_P_destl A new_P_rqt)

let I__cale_ = ~(-I_cgnt_ A new P fun astate A I_hold_)

let I_mrdy_ = ((new P fsm_hlda_) => F I ARBN)

let I_last_ = ((new_P fsm_hlda_) => (new_P_size A (new_P_fsm_a_ate) => WORDN 1 I WORDN 0)) I ARBN)

let I_lock_ = ~(-new_P lock_ A new_P lock_inh_)

(L ready_, L_last_, L_be_, L_mrdy_, L_ad_data_out, L_ad_addr_out, L_hlda_, L_lock_, L_cale_, L_male_, L_rale_, L_cqrt_, L_ad_out)

);;

%----------------------------------------------------------
% Next-state definition for Phase-B instruction.
%----------------------------------------------------------%
let p_sack = ((P_sizeA = ((P_downA) \Rightarrow \text{WORDN 1 | WORDN 0}) \land \neg \text{l_srdy} \land \text{P_fsm_dstate}) \Rightarrow \text{T}) \\
\begin{align*}
\langle \neg \text{p_ale} \land (\text{p_sack} \lor \text{Rst}) \rangle &\Rightarrow \text{F} \\
\langle \neg \text{p_ale} \land (\text{p_sack} \lor \text{Rst}) \rangle &\Rightarrow \text{P_rqt} \lor \text{ARB})
\end{align*}
\text{in}
\\text{let new_P_rqt} = \text{new_P_rqt in}
\\text{let new_P_load} = \text{new_P_load in}
\\text{let new_P_down} = (\neg \text{l_srdy} \land \text{P_fsm_dstate}) \text{in}
\\text{let new_P_size} = (\text{P_loadA}) \Rightarrow (\text{SUBARRAY L_ad_in (1,0)}) \text{in}
\\text{let new_P_male} = (\text{P_fsm_astate}) \Rightarrow \text{<-(p_dest1 \land (\neg (\text{SUBARRAY P_addr (25,24)}) = \text{WORDN 3})) \land \text{new_P_rqt} \lor \text{P_male}) \text{in}
\\text{let new_P_rale} = (\text{P_fsm_astate}) \Rightarrow \text{<-(p_dest1 \land (\text{SUBARRAY P_addr (25,24)}) = \text{WORDN 3}) \land \text{new_P_rqt} \lor \text{P_rale}) \text{in}
\\text{let new_P_lock} = ((\text{Rst}) \Rightarrow \text{T}) \Rightarrow \text{L_lock (P_lock)}) \text{in}
\\text{let new_P_lock_inh} = ((\neg \text{new_P_male} \lor \text{new_P_rale}) \Rightarrow \text{L_lock (P_lock_inh)}) \text{in}
\\text{let new_P_fsm_state} = \text{P_fsm_stateA in}
\\text{let new_P_fsm_rst} = \text{Rst in}
\\text{let new_P_fsm_mrqt} = (\text{P_dest1} \land \text{new_P_rqt}) \text{in}
\\text{let new_P_fsm_sack} = \text{p_sack in}
\\text{let new_P_fsm_cgnt} = \text{L_cgnt in}
\\text{let new_P_fsm_crqt} = (\text{P_dest1} \land \text{new_P_rqt}) \text{in}
\\text{let new_P_fsm_hold} = \text{L_hold in}
\\text{let new_P_fsm_lock} = \text{new_P_lock in}
\\text{let new_P_fsm_stateA} = \text{P_fsm_stateA in}
\\text{let new_P_fsm_astate} = \text{P_fsm_astate in}
\\text{let new_P_fsm_dstate} = \text{P_fsm_dstate in}
\\text{let new_P_fsm_hlda} = \text{P_fsm_hlda in}
\\text{let new_P_wr_data} = \text{P_wr_data in}
\\text{let new_P_addr} = \text{P_addr in}
\\text{let new_P_dest1} = \text{P_dest1 in}
\\text{let new_P_be} = \text{P_be in}
\\text{let new_P_wr} = \text{P_wr in}
\\text{let new_P_be_n} = \text{P_be_n in}
\\text{let new_P_sizeA} = \text{P_sizeA in}
\\text{let new_P_loadA} = \text{P_loadA in}
\\text{let new_P_downA} = \text{P_downA in}
\\text{(new_P_fsm_stateA, new_P_fsm_astate, new_P_fsm_dstate, new_P_fsm_hlda, new_P_wr_data, new_P_addr, new_P_dest1, new_P_be, new_P_wr, new_P_be_n, new_P_sizeA, new_P_loadA, new_P_downA, new_P_fsm_state, new_P_fsm_rst, new_P_fsm_mrqt, new_P_fsm_sack, new_P_fsm_cgnt, new_P_fsm_crqt, new_P_fsm_hold, new_P_fsm_lock, new_P_rqt, new_P_size, new_P_load, new_P_down, new_P_lock, new_P_lock_inh, new_P_male, new_P_rale, ...)}
\text{);}

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

let PH_B_out_def = new_definition
(\text{"PH_B_out",}
\text{\text{
\text{(P_fsm_state \text{P_fsm_stateA} : \text{p fsm ty})}
\text{\text{(P fsm astate P_fsm_dstate P fsm hlda P dest1 P wr P loadA P downA : bool)}}
\text{\text{(P fsm rst P fsm mrqt P fsm sack P fsm cgnt P fsm crqt P fsm hold P fsm lock P rqt P load : bool)}}
\text{)}}
\text{125}
let p_ale = (\L_ads_ \& \L_den_)

let p_sack = (\P_sizeA = (\P_downA \Rightarrow \text{WORDN} 1 \& \text{WORDN} 0)) \& \L_srdy_ \& \P_fsm_dstate)

let new_P_rqt = ((p_ale \& \neg(P_{sack} \lor \text{Rst})) \Rightarrow T !)

let new_P_load = \neg(P_{load})

let new_P_size = ((P_{loadA}) \Rightarrow (\text{SUBARRAY} \L_ad_in (1,0)))

let new_P_male = ((P_{fsm Astate}) \Rightarrow (-P_{destl} \& \neg((-\text{SUBARRAY} P_{addr} (25,24)) = \text{WORDN} 3)) \& \new_P_{rqt} \& \text{P_ male_})

let new_P_rqt = (((\neg p_{ale} \& (P_{sack} \lor \text{Rst})) \Rightarrow F !)

let new_P_{lock} = ((\text{Rst}) \Rightarrow T !)

let new_P_{lock inh} = ((\text{Rst}) \Rightarrow T !)

let new_P_{fsm state} = \P_{fsm stateA}

let new_P_{fsm rst} = \text{Rst}

let new_P_{fsm mrqt} = (-P_{destl} \& \new_P_{rqt})

let new_P_{fsm sack} = p_{sack}

let new_P_{fsm cgnt} = I_{cgnt_}

let new_P_{fsm crqt} = (-P_{destl} \& \new_P_{rqt})

let new_P_{fsm hold} = I_{hold_}

let new_P_{fsm lock} = \new_P_{lock_}

let new_P_{fsm state A} = P_{fsm stateA}

let new_P_{fsm astate} = P_{fsm astate}

let new_P_{fsm dstate} = \P_{fsm dstate}

let new_P_{fsm hlda} = \P_{fsm hlda}

let new_P_{wr data} = P_{wr data}

let new_P_{addr} = P_{addr}

let new_P_{dest1} = P_{dest1}

let new_P_{be} = P_{be_}

let new_P_{wr} = P_{wr}

let new_P_{be n} = P_{be_ n_}

let new_P_{sizeA} = \text{P}_{sizeA}

let new_P_{load A} = \text{P}_{load A}

let new_P_{down A} = \text{P}_{down A}

let L_{ad out} = ((\neg(P_{fsm astate} \& \new_P_{fsm hlda_} \& \neg((\neg(P_{fsm dstate} \& \text{new_P}_{wr})) \Rightarrow \L_{ad in} \& \text{ARBN}) in

let L_{ready_} = ((\neg\L_srdy_ \& \neg((\neg(P_{fsm dstate}))) in

let od0 = \text{ARBN}

let od1 = \text{ALTER od0 (31,27) new_P_{be_}}

let od2 = \text{ALTER od1 (26) F in}
let od3 = MALTER od2 (25,24) (SUBARRAY new_P_addr (1,0)) in
let od4 = MALTER od3 (23,0) (SUBARRAY new_P_addr (25,2)) in
let I_ad_addr_out = ((new_P_fsm_aestate) => od4 | ARBN) in
let I_ad_data_out = ((new_P_fsm_dstate \& new_P_wr) => new_P_wr_data | ARBN) in
let I_be_ = ((new_P_fsm_hlda_) => ((new_P_fsm_aestate) => new_P_be_ \& new_P_be_n_) | ARBN) in
let I_rale_ = ((new_P_fsm_hlda_) =>
~(-new_P_dest1 \& ((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) \& new_P_fsm_aestate \& new_P_rqt) | ARBN) in
let I_male_ = ((new_P_fsm_hlda_) => ~(-new_P_dest1 \& ~((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) \& new_P_fsm_aestate \& new_P_rqt) | ARBN) in
let I_cale_ = (~(-mly_ \& new_P_fsm_aestate \& I_hold_) in
let I_mrdy_ = ((new_P_fsm_hlda_) => F \& ARBN) in
let I_last_ = ((new_P_fsm_hlda_) => (new_P_sizeA = ((new_P_downA) => WORDN 1 \& WORDN 0)) \& ARBN) in
let I_hlda_ = new_P_fsm_hlda_ in
let I_lock_ = ~(-new_P_lock_ \& new_P_lock_inh_) in

(L_ready_, L_last_, I_be_, I_mrdy_, I_ad_data_out, I_ad_addr_out, I_hlda_, I_lock_, I_cale_, I_male_, I_rale_,
I_cqf_, L_ad_out))

);;
close_theory();;

127
C.2 M Port Specification

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

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

system 'rm m_phase.th';

new_theory 're_phase';

loadf 'abstract';

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

let m_state_ty = ":(mfsm ty#bool#bool#bcol#bool#bool#wordn#wordn#wordn#bcol#wordn#
  mfsm_ty#bcol#bool#bool#bool#bool#bool#bool#bool#
  bool#bool#wordn#wordn#wordn#bool#bool#bool#wordn#wordn )",

let instate = "((M fsm_stateA, M_fsm�dress,
  M_fsm read, M_fsm_write, M_fsm__byte_write, M fsm_mer_enable,
  M_addrA, M_beA, M_countA, M_rdyA, M_rd dataA, M fsm_state, M fsm_mal e_, M fsm_rd,
  M fsm_bw, M fsm ww, M fsm last_, M fsm_nurdy_, M fsm_zero_cnt, M fsm_rst, M se, M wr,
  M addr, M be, M count, M rdy, M wwdel, M parity, M rd_data, M detect)
  :'m_state_ty')";  

let m_env_ty = ":(bool#bool#bool#bool#bcol#wordn#bool#bool#wordn#bool#wordn#bool#booly'

let m_env = "((ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_ad_in, I_male_, I last_, I be_,
  I nury, MB_data_in, Edac_en, Reset_parity)
  :'m_env_ty')";  

let m_out_ty = ":(words#bool#bool#bool#bool#bool#bool#bool#bool)"

let m_out = "((I_ad_out, I_srdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_,
  MB parity)
  :'m_out_ty')";

let rep_ty = abstract_type 'aux_def' 'Andn';

let PH_A._inst_def = new_definition

%...........................................................................................-

Next-state definition for Phase-A instruction.

.................................................................................................%
let new_M_fsm_stateA =
  ((M_fsm_state = MW) => new_M_fsm_stateA = MW |)
let new_M_fsm_read = (new_M_fsm_stateA = MR)
let new_M_fsm_write = (new_M_fsm_stateA = MA)
let new_M_fsm_mem_enable = (new_M_fsm_stateA = M1)
let new_M_addrA = M_addr
let new_M_beA = M_be
let new_M_countA = M_count
let new_M_rdyA = M_rdy
let new_M_rd_dataA = M_rd_data
let new_M_fsm_state = M_fsm_state
let new_M_fsm_male_ = M_fsm_male_
let new_M_fsm_rd = M_fsm_rd
let new_M_fsm_bw = M_fsm_bw
let new_M_fsm_ww = M_fsm_ww
let new_M_fsm_last_ = M_fsm_last_
let new_M_fsm_mrdy_ = M_fsm_mrdy_
let new_M_fsm_zero_cnt = M_fsm_zero_cnt
let new_M_fsm_rst = M_fsm_rst
let new_M_se = M_se
let new_M_wr = M_wr
let new_M_addr = M_addr
let new_M_be = M_be

let new_M_fsm_address = new_M_fsm_stateA | MA
let new_M_fsm_read = new_M_fsm_stateA | MR
let new_M_fsm_write = new_M_fsm_stateA | MA
let new_M_fsm_mem_enable = new_M_fsm_stateA | M1
let new_M_count = M_count in
let new_M_rdy = M_rdy in
let new_M_wwdel = M_wwdel in
let new_M_parity = M_parity in
let new_M_rd_data = M_rd_data in
let new_M_detect = M_detect in

(new_M_fsm_stateA, new_M_fsm_address, new_M_fsm_read, new_M_fsm_write, new_M_fsm_byte_write,
new_M_fsm_mem_enable, new_M_addrA, new_M_beA, new_M_countA, new_M_rdyA, new_M_rd_dataA,
new_M_fsm_state, new_M_fsm_male, new_M_fsm_rd, new_M_fsm_bw, new_M_fsm_last,
new_M_fsm_mrdy, new_M_fsm_zero_cnt, new_M_fsm_rst, new_M_se, new_M_wr, new_M_addr, new_M_be,
new_M_count, new_M_rdy, new_M_wwdel, new_M_parity, new_M_rd_data, new_M_detect)
);

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

let PH_A_out_def = new_definition
('PH_A_out',
"%(M_fsm_stateA M_fsm_state :mfsm_ty)
 (M_addrA M_beA M_countA M_rd_dataA M_addrA M_beA M_countA M_rd_dataA M_detect :wordn)
(M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_rdyA
 M_fsm_male M_fsm_rd M_fsm_bw M_fsm_last M_fsm_mrdy M_fsm_zero_cnt M_fsm_rst
 M_se M_wr M_rdy M_wwdel M_parity :bool)
(I_ad_in I_be MB_data_in :wordn)
(ClkA ClkB Rst Disable_eeprom Disable_writes I_male I_last I_mrdy Edac_en Reset_parity :bool)
(rep :rep_ty)
).

PH_A_out (M_fsm_stateA, M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable,
 M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male, M_fsm_rd,
 M_fsm_bw, M_fsm_wm, M_fsm_last, M_fsm_mrdy, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr,
 M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
 (ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_male, I_last, I_mrdy, Edac_en, Reset_parity)
 rep =

let new_M_fsm_stateA =
((M_fsm_rst) => MI 
 ((M_fsm_state = MI) => (M_fsm_male) => MA | MI) 
 ((M_fsm_state = MA) =>
  ((M_fsm_mrdy_ M_fsm_bw) => MW  
  ((M_fsm_mrdy_ (M_fsm_rd V M_fsm_bw)) => MR | MA)) 
 ((M_fsm_state = MR) =>
  ((M_fsm_bw M_fsm_zero_cnt) => MBW  
  ((M_fsm_last_ M_fsm_rd M_fsm_zero_cnt) => MA  
  ((M_fsm_last_ (M_fsm_rd M_fsm_zero_cnt) => MRR | MR))) 
 ((M_fsm_state = MRR) => MI 
 ((M_fsm_state = MW) =>
  ((M_fsm_last M_fsm_zero_cnt) => MI  
  ((M_fsm_last_ M_fsm_zero_cnt) => MA | MW)) 
 ((M_fsm_state = MBW) => MW | M_ILL))))
 in
let new_M_fsm_address = (new_M_fsm_stateA = MA) in
let new_M_fsm_read = (new_M_fsm_stateA = MR) in

130
let new_M_fsm_write = (new_M_fsm_stateA = MW) in
let new_M_fsm_byte_write = (new_M_fsm_stateA = MBW) in
let new_M_fsm_mem_enable = (~new_M_fsm_stateA = MI) in
let new_M_addrA = M_addr in
let new_M_beA = M_be in
let new_M_countA = M_count in
let new_M_rdyA = M_rdy in
let new_M_hd_dataA = M_hd_data in
let new_M_fsm_state = M_fsm_state in
let new_M_fsm_male_ = M_fsm_male_ in
let new_M_fsm_rd = M_fsm_rd in
let new_M_fsm_bw = M_fsm_bw in
let new_M_fsm_ww = M_fsm_ww in
let new_M_fsm_last_ = M_fsm_last_ in
let new_M_fsm_mrdy_ = M_fsm_mrdy_ in
let new_M_fsm_zero_cnt = M_fsm_zero_cnt in
let new_M_fsm_rst = M_fsm_rst in
let new_M_se = M_se in
let new_M_wr = M_wr in
let new_M_addr = M_addr in
let new_M_be = M_be in
let new_M_count = M_count in
let new_M_rdy = M_rdy in
let new_M_wwdel = M_wwdel in
let new_M_parity = M_parity in
let new_M_fd_data = M_fd_data in
let new_MDetect = MDetect in

let m_rdy = ((new_M_fsm_write A (new_M_countA = OVORDN 1))) V (new_M_fsm_read A (new_M_countA = OVORDN 1)) A ~new_M_wr)) in
let m_srdy_ = ~((new_M_rdyA A new_M_wr) V (m_rdy A new_M_wr)) in
let mb_data_7_0 = ((ELEMENT new_M_beA (0)) => (SUBARRAY I_ad_in (7,0)) I (SUBARRAY new_M_fd_dataA (7,0))) in
let mb_data_15_8 = ((ELEMENT new_M_beA (1)) => (SUBARRAY I_ad_in (15,8)) I (SUBARRAY new_M_fd_dataA (15,8))) in
let mb_data_23_16 = ((ELEMENT new_M_beA (2)) => (SUBARRAY I_ad_in (23,16)) I (SUBARRAY new_M_fd_dataA (23,16))) in
let mb_data_31_24 = ((ELEMENT new_M_beA (3)) => (SUBARRAY I_ad_in (31,24)) I (SUBARRAY new_M_fd_dataA (31,24))) in
let mb_data = ((MALTER (MALTER (MALTER ARBN (7,0) mb_data_7_0) (15,8) mb_data_15_8) (23,16) mb_data_23_16) (31,24) mb_data_31_24)) in

let I_ad_out = ((~new_M_wr A new_M_fsm_mem_enable) => new_M_fd_dataA | ARBN) in
let I_srdy_ = ((new_M_fsm_mem_enable) => m_srdy_ | ARB) in
let MB_addr = ((new_M_rdyA) => (INCN 18 new_M_addrA) A new_M_addrA) in
let MB_data_out = ((new_M_fsm_write) => (Ham_Enc rep mb_data) | ARBN) in
let MB_cseprom_ = ~(new_M_fsm_mem_enable A new_M_se) in
let MB_cs_sram_ = ~(new_M_fsm_mem_enable A new_M_se) in
let MB_we_ = ~((new_M_se A ~new_M_fsm_mem_enable A new_M_se) in
let MB_oe_ = ~((new_M_we V ~new_M_fsm_mem_enable) V ~Disable_eeprom)

(I_ad_out, I_srdy_, MB_addr, MB_data_out, MB_cseprom_, MB_cs_sram_, MB_we_, MB_oe_, MB_parity)"
Next-state definition for Phase-B instruction.

let PH_B_inst_def = new_definition
( 'PH_B_inst',
  "'(M_fsm_stateA M_fsm_state :fsm_ty)
   (M_addrA M_beA M_countA M_rd_dataA M_addr M_be M_count M__rd._data M_detect :wordn)
   (M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_rdyA
   M fsm_male M fsm_bw M fsm_ww M fsm_last_ M fsm_mrdy_ M fsm_zero_cnt M fsm_rst
   M_se M_wr M_rdy M_wwdel M parity :bool)
   (L ad_in I be MB data in :wordn)
   (ClkA ClkB Rst Disable_eeprom Disable_writes I male_ I last_ I mrdy Edac en Reset parity :bool)
   (rep:'_rep_ty).
   PH_B_inst (M_fsm_stateA, M_fsm_address, M_fsm_read, Mjsm_write, M_fsm_byte_write, M_fsm_mem_enable, M_addrA, M_beA, M_countA, M_rdA, M_rd_dataA, M_fsm_state, M_fsm_male, M_fsm_rd,
   M fsm_bw, M fsm_ww, M fsm_last_, M fsm_mrdy_, M fsm_zero_cnt, M fsm_rst, M_se, M_wr,
   M_addr, M_be, M_count, M_rdy, M_wwdel, M parity, M_rd_data, M detect)
   (ClkA, ClkB, Rst Disable_eeprom Disable_writes I ad_in, I male, I last, I be, I mrdy, MB data in, Edac en, Reset parity)
   rep =
";

let new_M_se = ((-l_male) => (ELEMENT l ad_in (23)) | M_se) in
let new_M_wr = ((-l_male) => (ELEMENT I ad_in (27)) | M_wr) in
let new_M_addr =
  ((-l_male) => (SUBARRAY I ad_in (18,0)) |
   ((l_rdyA) => (INCN 18 M_addrA) | M_addrA)) in
let new_M_count =
  (((M fsm_address V M fsm_byte_write) => ((new M_se) => (WORDN 1) | (WORDN 2)) |
   ((M fsm_write V M fsm_read) => (DECN 1 M_countA) | M_countA)) in
let m_rdy = ((M fsm_write A new_M_count = (WORDN 0))) |
  (V (M fsm_read A (new M count = (WORDN 0))) A ~new_M wr)) in
let new_M_be =
  ((-l_male V ~m_rdy) => (NOTN 3 I be) | M be) in
let new_M_rdy = m_rdy in
let new_M_wwdel = (M fsm_address A new_M wr A (new M be = (WORDN 15))) in
let new_M_rd_data = ((M fsm_read) => (Ham_Dec rep MB data in) | M rd_data) in
let new_M_detect =
  (((M fsm_read A ~new_M wr) V new_M wr V ~M fsm_mem_enable) =>
   (Edac en) => (Ham_Det1 rep MB data in) | (WORDN 0)) | M detect) in
let new_M_error =
  ((m_error A ~((Rst V Reset parity)) => T) |
   ((m_error A ~((Rst V Reset parity)) => F) |
    ((m_error A ~((Rst V Reset parity)) => M parity | ARB)) in
let new_M_fsm_state = M fsm_stateA in
let new_M_fsm_male = I male in
let new_M_fsm_rd = (~new_M wr A M fsm_mem_enable) in
let new_M_fsm_bw = ((new M be = (WORDN 15)) A new_M wr A M fsm_mem_enable) in
let new_M_fsm_ww = ((new M be = (WORDN 15)) A new_M wr A M fsm_mem_enable) in
let new_M_fsm_last_ = I last_ in

132
let new_M_fsm_mrdy_ = l_mrdy_ in
let new_M_fsm_zero_cnt = (new_M_count = (WORDN 0)) in
let new_M_fsm_rst = Rst in
let new_M_fsm_stateA = M_fsm_stateA in
let new_M_fsm_address = M_fsm_address in
let new_M_fsm_read = M_fsm_read in
let new_M_fsm_write = M_fsm_write in
let new_M_fsm_byte_write = M_fsm_byte_write in
let new_M_fsm_mem_enable = M_fsm_mem_enable in
let new_M_addrA = M_addrA in
let new_M_beA = M_beA in
let new_M_countA = M_countA in
let new_M_rdyA = M_rdyA in
let new_M_rdataA = M_rdataA in
let new_M_rd_dataA = M_rd_dataA in

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

);;

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

let PH_B_out_def = new_definition
  ("PH_B_out",
   "I (M_fsm_stateA M_fsm_state :msm_ty)
    (M_addrA M_beA M_countA M_rd_dataA M_addr M_be M_count M_rd_data M_detect :wordn)
    (M_fsm_address M_fsm_read M_fsm_write M_fsm_byte_write M_fsm_mem_enable M_rdyA
     M_fsm_male M_fsm_rd M_fsm_bw M_fsm_wrw M_fsm_last M_fsm_mrdy M_fsm_zero_cnt M_fsm_rst
     M_se M_wr M_rdy M_wwdel M_parity :bool)
    (ClkA ClkB Rst Disable_eeprom Disable_writes I_male I_last I_mrdy Edac_en Reset_parity :bool)
    (rep:rep_ty).

  PH_B_out (M_fsm_stateA, M_fsm_address, M_fsm_read, M_fsm_write, M_fsm_byte_write, M_fsm_mem_enable, 
    M_addrA, M_beA, M_countA, M_rdyA, M_rd_dataA, M_fsm_state, M_fsm_male, M_fsm_rd, 
    M_fsm_bw, M_fsm_wrw, M_fsm_last, M_fsm_mrdy, M_fsm_zero_cnt, M_fsm_rst, M_se, M_wr, 
    M_addr, M_be, M_count, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect)
    (ClkA, ClkB, Rst, Disable_eeprom, Disable_writes, I_male, I_last, I_mrdy, Edac_en, Reset_parity)
    rep =
  
  let new_M_se = ((-I_male_) => (ELEMENT I_ad_in (23)) \ M_se) in
  let new_M_wr = ((-I_male_) => (ELEMENT I_ad_in (27)) \ M_wr) in
  let new_M_addr = 
    ((-I_male_) => (SUBARRAY I_ad_in (18,0))) 
    ((M_rdyA) => (INCN 18 M_addrA) \ M_addrA)) in
  let new_M_count = 
    ((M_fsm_address \ M_fsm_byte_write) => ((new_M_se) => (WORDN 1) \ (WORDN 2))) 
    ((M_fsm_write \ M_fsm_read) => (DECN 1 M_countA) \ M_countA)) in
  let m_rdy = ((M_fsm_write A (new_M_count = (WORDN 0)))

133
V (M fsm_read ∧ (new_M_count = (WORDN 0)) ∧ ¬new_M_wr)) in

let m_srdy_ = ¬((M_rdyA ∧ ¬new_M_wr) ∨ (m_rdy ∧ new_M_wr)) in

let new_M_be = (((-I_male_ ∨ ¬m_srdy_) (NOTN 1) be) ∧ M_be) in

let new_M_rdy = m_rdy in

let new_M_rwr = = ((M_fsm_address ∧ new_M_wr ∧ (new_M_be = (WORDN 15))) in

let new_M.rd_data = ((M_fsm_read) ⇒ (Ham_Dec rep MB.data_in) ∧ M_rd_data) in

let new_M.detect = 
(((M_fsm_read ∧ ¬new_M_wr) ∨ new_M_wr ∨ ¬M_fsm_mem_enable) −
  (¬Edac_en_ ⇒ (Ham_Dec rep MB.data_in) (WORDN 0)) ∧ M.detect) in

let m.error = ¬((m_srdy ∧ M_fsm_mem_enable ∧ (Ham_Dec rep (new_M.detect, ¬Edac_en_))) in

let new_M.parity = 
((m.error ∧ ¬(Rst ∨ Reset_parity)) ⇒ T ∧
  (¬m.error ∧ (Rst ∨ Reset_parity)) ⇒ F ∧
  (¬m.error ∧ ¬(Rst ∨ Reset_parity)) ⇒ M_parity ∨ ARB)) in

let new_M.fsm_state = M_fsm_stateA in

let new_M.fsm_mem = I_male in

let new_M.fsm.rd = ¬(new_M_wr ∧ M_fsm_mem_enable) in

let new_M.fsm.bw = ((¬new_M.be = (WORDN 15)) ∧ new_M_wr ∧ M_fsm_mem_enable) in

let new_M.fsm.wr = ((new_M.be = (WORDN 15)) ∧ new_M_wr ∧ M fsm_mem_enable) in

let new_M.fsm.last = I_last in

let new_M.fsm.mrdy = I_mrdy in

let new_M.fsm.zero_cnt = (new_M_count = (WORDN 0)) in

let new_M.fsm.rst = Rst in

let new_M.fsm.state = M_fsm_stateA in

let new_M.fsm.address = M_fsm_address in

let new_M.fsm.read = M_fsm_read in

let new_M.fsm.write = M_fsm_write in

let new_M.fsm.byte_write = M_fsm_byte_write in

let new_M.fsm_mem_enable = M_fsm_mem_enable in

let new_M.addrA = M_addrA in

let new_M_beA = M_beA in

let new_M.countA = M_countA in

let new_M_rdyA = M_rdyA in

let new_M_addrA = M_rd_dataA in

let m_rdy = ((new_M_fsm_write ∧ (new_M_countA = (WORDN 1))) ∨
  (new_M_fsm_read ∧ (new_M_countA = (WORDN 1)) ∧ ¬new_M_wr)) in

let m_srdy_ = ¬((new_M_rdyA ∧ ¬new_M_wr) ∨ (m_rdy ∧ new_M_wr)) in

let mb.data_7_0 = ((ELEMENT new_M_beA (0)) ⇒ (SUBARRAY I.ad.in (7,0)) 
  (SUBARRAY new_M.rd.dataA (7,0)))) in

let mb.data_15_8 =

((ELEMENT new_M_beA (1)) ⇒ (SUBARRAY I.ad.in (15,8)) 
  (SUBARRAY new_M.rd.dataA (15,8)))) in

let mb.data_23_16 =

((ELEMENT new_M_beA (2)) ⇒ (SUBARRAY I.ad.in (23,16)) 
  (SUBARRAY new_M.rd.dataA (23,16)))) in

let mb.data_31_24 =

((ELEMENT new_M_beA (3)) ⇒ (SUBARRAY I.ad.in (31,24)) 
  (SUBARRAY new_M.rd.dataA (31,24)))) in

let mb.data = ((MALTER (MALTER (MALTER ARBN (7,0) mb.data_7_0)
  (15,8) mb.data_15_8)
  (23,16) mb.data_23_16)
  (31,24) mb.data_31_24)) in

let I.ad.out = ((¬new_M_wr ∧ new_M_fsm_mem_enable) ⇒ new_M.rd.dataA ∨ ARBN) in

let l.srdy_ = ((new_M_fsm_mem_enable) ⇒ m.srdy_ ∨ ARBN) in

let MB_addr = ((new_M_rdyA) ⇒ (INCN 18 new_M_addrA) ∧ new_M_addrA) in

let MB_data.out = ((new_M_fsm_write) ⇒ (Ham.Enc rep mb.data) ∨ ARBN) in

let MB.cs.eeprom_ = ¬((new_M_fsm_mem_enable ∧ ¬new_M_se) in
let MB_cs_sram_ = \(\neg (\text{new}_M\_fsm\_mem\_enable \land \text{new}_M\_se)\) in
let MB_we_ = \(\neg ((\text{new}_M\_se \lor \neg \text{new}_M\_fsm\_mem\_enable \lor \neg \text{Disable}\_eeprom)\)
\(\land \neg \text{Disable}\_writes\)
\(\land (\text{new}_M\_fsm\_byte\_write \lor \text{new}_M\_fsm\_write \lor \text{new}_M\_wwdel))\) in
let MB_oe_ = \(\neg ((\neg \text{new}_M\_wr \land \text{new}_M\_fsm\_address) \lor \text{new}_M\_fsm\_read)\) in
let MB_parity = \(\text{new}_M\_parity\) in

\((\text{L}_\text{ad\_out}, \text{L}_\text{srdy\_}, \text{MB\_addr, MB\_data\_out, MB\_cs\_eeprom\_, MB\_cs\_sram\_, MB\_we\_, MB\_oe\_, MB\_parity})\) ;

135
C.3 R Port Specification

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

```
set_search_path (search_path () @ ["/home/titan3/dfura/ftep/piu/bol/lib/"]);

system 'rm r_phase.th';

new_theory 'r_phase';

loadf 'abstract';

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

let r__state_ty = _`:(rfsm-ty#bo__#bo__#b___#bo__#b___#bo__#b___#b___#b___#b___#b___F____#w_rdn#w_rdn#

bool#booi#wordn#wordn#bool#bool#wordn#wordn#boolteoool#wordn#wordn#bool#bool#

wordn#bool#wordn#wordn#wordn#rfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#

bool#bool#wordn#word_

bool#boo I#bool#wordn#wordn#bool#wordn#bool#bool#bool#wordn#wordn#bool#booi#wordn#wordn#bool#

wordn#bool#wordn#bool)";;

let r_state = "((R_fsm_stateA, R_fsm_cntlatch, R_fsm_s_dy_, R_int0_en, R

into
disA, R_int3 ea, R_int3_disA, R

c01
cout, R_cOl_cout_delA, R
c23_cout, R_c23_cout_delA, R

cntrlatch_delA, R_srdy_delA, R

reg_selA, R_ctr0, R_ctr0_ce, R_ctr0_cin, R_ctr0_outA, R

ctrl1, R

crtl1_cin, R

cntrl1_outA, R

cntrl2, R

cntrl2_cin, R

cntrl2_outA, R

cntrl3, R

cntrl3_cin, R

cntrl3_outA, R

irc_loadA, R

irc_oldA, R

irc new, R

irc old, R

irc mask, R

irc, R

icrA, R

icr_, latch, R

 fsm state, R

fsn_sle,

 R

 fsm_mrdy, _ R

 fsm_last, _ R

 fsm_stat, R

 int0 dis, R

 int3 dis, R

 e01_cout del, R

 int1 en, R

c23_cout del, R

 int2 en, _ R

 cntlatch del, R

 srdy del, _ R

 reg sel, R

 ctr0 in, R

 ctr0 mux sel, R

 ctr0 irden, R

 ctr0 cry, R

 ctr0 new, R

 ctr0 out, R

 ctr0 orden, R

 ctrl1 in, R

 ctrl1 mux sel, R

 ctrl1 irden, R

 ctrl1 cry, R

 ctrl1 new, R

 ctrl1 out, R

 ctrl1 orden, R

 ctrl2 in, R

 ctrl2 mux sel, R

 ctrl2 irden, R

 ctrl2 cry, R

 ctrl2 new, R

 ctrl2 out, R

 ctrl2 orden, R

 ctrl3 in, R

 ctrl3 mux sel, R

 ctrl3 irden, R

 ctrl3 cry, R

 ctrl3 new, R

 ctrl3 out, R

 ctrl3 orden, R

 ircr load, R

 ircr old, R

 ircr mask, R

 irc new, R

 irc old, R

 ccr, R

 ccr rden, R

 gcr, R

 gcr rden, R

 sr, R

 ss rden)

 :"r_state_ty");;

let r_env_ty = "((bool#bool#bool#wordn#bool#bool#bool#bool#bool#bool#bool#bool#

wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#

wordn#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#

wordn#wordn#bool#bool)";;

let r_env = "((Click, Click, Rst, l_ad_in, l_rate_, l_last_, l_be_, l_mrdy_, Disable_int, Disable_writes,

Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss)

```

136
let r_out_ty = "\((\text{wordn} \times \text{bool} \times \text{bool} \times \text{bool} \times \text{bool} \times \text{bool} \times \text{wordn} \times \text{wordn} \times \text{bool} \times \text{bool})\)";;

let rout = "((I\_ad\_out, I\_srdy\_, Int0\_, Int1\_, Int2\_, Ccr, Led, Reset\_error, Prom\_invalid) -> \((\text{wordn} \times \text{bool})\))";;

let rep\_ty = abstract\_type 'aux\_def 'Andn';;

Next-state definition for Phase-A instruction.

let PH\_A\_inst\_def = new\_definition ('PH\_A\_inst',
  "(! rep\_ty)
  \begin{array}{l}
  (R\_fsm\_stateA R\_fsm\_state\_t\_fsm\_ty) \\
  (R\_reg\_selA R\_ctr0 R\_ctr1\_outA R\_ctr2 R\_ctr2\_outA R\_ctr3 R\_ctr3\_outA R\_icr\_oldA \\
   R\_icrA R\_bus\_latch R\_reg\_sel R\_ctr0\_in R\_ctr0\_new R\_ctr1\_out R\_ctr1\_new R\_ctr1\_out \\
   R\_ctr2\_in R\_ctr2\_new R\_ctr2\_out R\_ctr2\_in R\_ctr2\_new R\_ctr2\_out R\_icr\_old R\_icr\_mask R\_icr \\
   R\_ccr R\_gcr R\_sr :\text{wordn}) \\
  (R\_fsm\_cntlatch R\_fsm\_srdy\_ R\_int0\_en R\_int0\_disA R\_int3\_en R\_int3\_disA R\_co1\_cout R\_co1\_cout\_delA \\
   R\_c23\_cout R\_c23\_cout\_delA R\_cntlatch\_delA R\_srdy\_delA R\_ctr0\_ce R\_ctr0\_cin R\_ctr1\_ce R\_ctr1\_cin \\
   R\_ctr2\_ce R\_ctr2\_cin R\_ctr3\_ce R\_ctr3\_cin R\_icr\_loadA R\_fsm\_ale\_ R\_fsm\_nrdy\_ R\_fsm\_last\_ R\_fsm\_rst \\
   R\_int0\_dis R\_int3\_dis R\_co1\_cout\_del R\_int1\_en R\_c23\_cout\_del R\_int2\_en R\_wr R\_cntlatch\_del \\
   R\_srdy\_del R\_ctr0\_mux\_sel R\_ctr0\_irden R\_ctr0\_cry R\_ctr0\_ord R\_ctr1\_mux\_sel R\_ctr1\_irden \\
   R\_ctr1\_cry R\_ctr1\_ord R\_ctr2\_mux\_sel R\_ctr2\_irden R\_ctr2\_cry R\_ctr2\_ord R\_ctr3\_mux\_sel \\
   R\_ctr3\_irden R\_ctr3\_cry R\_ctr3\_ord R\_icr\_load R\_icr\_rden R\_ccr\_rden R\_ccr\_den R\_ccr\_rden R\_sr \_rden :\text{wordn}) \\
  (I\_ad\_in I\_be\_ Cpu\_fail Reset\_cpu S\_state Id ChannelID C\_ss :\text{wordn}) \\
  (ClkA ClkB Rst I\_rate\_ I\_last\_ I\_mrdy\_ Disable\_int Disable\_writes Piu\_fail Pmm\_fail \\
   CB\_parity MB\_parity :\text{bool})
\end{array}
\)

let new\_R\_fsm\_stateA =
  \begin{array}{l}
  \text{if } R\_fsm\_state = R\_fsm\_rst \text{ then return } R\_fsm\_rst \text{ else continue} \\
  \text{if } R\_fsm\_state = R\_fsm\_last \text{ then return } R\_fsm\_last \text{ else continue} \\
  \text{if } R\_fsm\_ale = RA \text{ then return } RA \text{ else continue} \\
  \text{if } R\_fsm\_ale = RA \text{ then return } RA \text{ else continue}
\end{array}

let new\_R\_fsm\_cntlatch = ((R\_fsm\_state = R\_fsm\_rst) \land \neg R\_fsm\_ale) in
let new_R_fsm_srdy_ = -((R_fsm_state = RA) \&\& \neg R_fsm_mrdy_) in
let new_R_cnlatch_delA = R_cnlatch_del in
let new_R_srdy_delA = R_srdy_del in
let new_R_reg_selA = R_reg_sel in
let r_reg_gel = ((\neg new_R_srdy_delA) \imp (INCN 3 new_R_reg_selA) \&\& \neg R_reg_selA) in
let r_write = (\neg Disable_writes \&\& R_wr \&\& (new_R_fsm_stateA = RD)) in
let r_read = (\neg R_wr \&\& (new_R_fsm_stateA = RA)) in
let new_R_ctr0 = ((R_ctr0_mux_sel) \imp R_ctl0_in \&\& R_ctr0_new) in
let new_R_ctr0_ce = (ELEMENT R_gcr (19)) in
let new_R_ctr0_cin = T in
let new_R_ctr0_outA = R_ctr0_new in
let new_R_ctr1 = ((R_ctr1_mux_sel) \imp R_ctr1_in \&\& R_ctr1_new) in
let new_R_ctr1_ce = T in
let new_R_ctr1_cin = R_ctr0_cry in
let new_R_ctr1_outA = R_ctr1_new in
let new_R_ctr2 = ((R_ctr2_mux_sel) \imp R_ctr2_in \&\& R_ctr2_new) in
let new_R_ctr2_ce = (ELEMENT R_gcr (23)) in
let new_R_ctr2_cin = T in
let new_R_ctr2_outA = R_ctr2_new in
let new_R_ctr3 = ((R_ctr3_mux_sel) \imp R_ctr3_in \&\& R_ctr3_new) in
let new_R_ctr3_ce = T in
let new_R_ctr3_cin = R_ctr2_cry in
let new_R_ctr3_outA = R_ctr3_new in
let new_R_iar_loadA = R_iar_load in
let new_R_iar_oldA = 
  (((new_R_fsm_stateA = RA) \&\& (\neg reg_sel = (WORDN 0)) \&\& (\neg reg_sel = (WORDN 1))) \imp R_iar \&\& R_iar_oldA) in
let new_R_iarA = 
  (\neg (\neg\&\&\& R_iar \&\& R_iar_mask)) \imp R_iar_mask in
let new_R_int0_en = (((ELEMENT R_iar (0)) \&\& (ELEMENT R_iar (8))) V
  ((ELEMENT R_iar (1)) \&\& (ELEMENT R_iar (9))) V
  ((ELEMENT R_iar (2)) \&\& (ELEMENT R_iar (10))) V
  ((ELEMENT R_iar (3)) \&\& (ELEMENT R_iar (11))) V
  ((ELEMENT R_iar (4)) \&\& (ELEMENT R_iar (12))) V
  ((ELEMENT R_iar (5)) \&\& (ELEMENT R_iar (13))) V
  ((ELEMENT R_iar (6)) \&\& (ELEMENT R_iar (14))) V
  ((ELEMENT R_iar (7)) \&\& (ELEMENT R_iar (15))) in
let new_R_int0_disA = R_int0_dis in
let new_R_int3_en = (((ELEMENT R_iar (16)) \&\& (ELEMENT R_iar (24))) V
  ((ELEMENT R_iar (17)) \&\& (ELEMENT R_iar (25))) V
  ((ELEMENT R_iar (18)) \&\& (ELEMENT R_iar (26))) V
  ((ELEMENT R_iar (19)) \&\& (ELEMENT R_iar (27))) V
  ((ELEMENT R_iar (20)) \&\& (ELEMENT R_iar (28))) V
  ((ELEMENT R_iar (21)) \&\& (ELEMENT R_iar (29))) V
  ((ELEMENT R_iar (22)) \&\& (ELEMENT R_iar (30))) V
  ((ELEMENT R_iar (23)) \&\& (ELEMENT R_iar (31))) in
let new_R_int3_disA = R_int3_dis in
let new_R_ctr0_cry = R_ctr0_cry in
let new_R_cnlatch_delA = R_cnlatch_del in
let new_R_cnlatch_delA = R_cnlatch_del in
let new_R_cnlatch_delA = R_cnlatch_del in
let new_R_fsm_state = R_fsm_state in
let new_R_fsm_ale_ = R_fsm_ale_ in
let new_R_fsm_mrdy_ = R_fsm_mrdy_ in
let new_R_fsm_last_ = R_fsm_last_ in
let new_R_fsm_rst = R_fsm_rst in
let new_R_int0_dis = R_int0_dis in
let new_R_int3_dis = R_int3_dis in
let new_R_c01_cout_del = R_c01_cout_del in
let new_R_int1_en = R_int1_en in
let new_R_c23_cout_del = R_c23_cout_del in
let new_R_int2_en = R_int2_en in
let new_R_wr = R_wr in
let new_R_cntlatch_del = R_cntlatch_del in
let new_R_offer = R_offer in
let new_R_reg_sel = R_reg_sel in
let new_R_ctr0_in = R_ctr0_in in
let new_R_ctr0_mux_sel = R_ctr0_mux_sel in
let new_R_ctr0_irden = R_ctr0_irden in
let new_R_ctr0_cry = R_ctr0_cry in
let new_R_ctr0_new = R_ctr0_new in
let new_R_ctr0_out = R_ctr0_out in
let new_R_ctr0_orden = R_ctr0_orden in
let new_R_ctr1_in = R_ctr1_in in
let new_R_ctr1_mux_sel = R_ctr1_mux_sel in
let new_R_ctr1_irden = R_ctr1_irden in
let new_R_ctr1_cry = R_ctr1_cry in
let new_R_ctr1_new = R_ctr1_new in
let new_R_ctr1_out = R_ctr1_out in
let new_R_ctr1_orden = R_ctr1_orden in
let new_R_ctr2_in = R_ctr2_in in
let new_R_ctr2_mux_sel = R_ctr2_mux_sel in
let new_R_ctr2_irden = R_ctr2_irden in
let new_R_ctr2_cry = R_ctr2_cry in
let new_R_ctr2_new = R_ctr2_new in
let new_R_ctr2_out = R_ctr2_out in
let new_R_ctr2_orden = R_ctr2_orden in
let new_R_ctr3_in = R_ctr3_in in
let new_R_ctr3_mux_sel = R_ctr3_mux_sel in
let new_R_ctr3_irden = R_ctr3_irden in
let new_R_ctr3_cry = R_ctr3_cry in
let new_R_ctr3_new = R_ctr3_new in
let new_R_ctr3_out = R_ctr3_out in
let new_R_ctr3_orden = R_ctr3_orden in
let new R_icr_load = R_icr_load in
let new R_icr_old = R_icr_old in
let new R_icr_mask = R_icr_mask in
let new R_icr = R_icr in
let new R_ccr = R_ccr in
let new R_gcr = R_gcr in
let new R_sr = R_sr in
let new R_sr = R_sr in

(new R fsm stateA, new R fsm cntlatch, new R fsm srdy_, new R int0_en, new R int0_disA, new R int3_en,
new R int3_disA, new R c01_cout, new R c01_cout delA, new R c23_cout, new R c23_cout delA,
new R cntlatch delA,
new R srdy delA_, new R reg selA, new R ctrl0, new R ctrl0 outA, new R ctrl0 ce, new R ctrl0 cin, new R ctrl0 outA, new R ctrl, new R ctrl outA, new R ctrl ce, new R ctrl cin, new R ctrl outA, new R ctrl3, new R ctrl3 ce, new R ctrl3 cin, new R ctrl3 outA, new R icr_loadA, new R icr_oldA, new R icrA,
new R busA_latch, new R fsm state, new R fsm ale_, new R fsm mrdy_, new R fsm last_, new R fsm rst,
new R int0 dis, new R int3 dis, new R cntlatch del, new R int0_en, new R c23_cout, new R int2_en,
new R wr,
new R cntlatch del, new R srdy del, new R reg sel, new R ctrl0 in, new R ctrl0 mux sel, new R ctrl0 irden,
new R ctrl0 cry, new R ctrl0 new, new R ctrl0 order, new R ctrl1 in, new R ctrl1 mux sel, new R ctrl1 irden,
new R ctrl1 cry, new R ctrl1 new, new R ctrl1 order, new R ctrl2 in,
new R ctrl2 mux sel, new R ctrl2 irden, new R ctrl2 cry, new R ctrl2 new, new R ctrl2 order, new R ctrl3 in,
new R ctrl3 mux sel, new R ctrl3 irden, new R ctrl3 cry, new R ctrl3 new, new R ctrl3 out,
new R ctrl3 order, new R icr_load, new R icr_old, new R icr mask, new R icr, new R icr rden, new R icr,
new R ccr, new R icr rden, new R icr, new R icr rden)

let PH_A out_def = new definition
('PH_A out',
"("(rep:rep ty)
(R fsm stateA R fsm state:rsfm ty)
(R reg selA R ctrl0 R ctrl0 outA R ctrl1 R ctrl1 outA R ctrl2 R ctrl2 outA R ctrl3 R ctrl3 outA R icr oldA
R icrA R busA_latch R reg sel R ctrl0 in R ctrl0 out R ctrl1 in R ctrl1 out
R ctrl2 in R ctrl2 new R ctrl2 out R ctrl3 in R ctrl3 new R icr outA R icr mask R icr
R ccr R gcr R sr :words)
(R fsm cntlatch R fsm srdy_ R int0_en R int0_disA R int3_disA R c01_cout R c01_cout delA
R c23_cout R c23_cout delA R cntlatch delA R srdy_delA R ctrl0 ce R ctrl0 cin R ctrl1 ce R ctrl1 cin
R ctrl2 ce R ctrl2 cin R ctrl3 ce R ctrl3 cin R icr loadA R fsm ale_ R fsm mrdy R fsm last R fsm rst
R int0 dis R int3 dis R c01 cout del R int1_en R c23 cout del R int2 en R wr R cntlatch del
R srdy del R ctrl0 mux sel R ctrl0 order R ctrl0 cry R ctrl0 order R ctrl1 mux sel R ctrl1 order
R ctrl1 cry R ctrl1 order R ctrl2 mux sel R ctrl2 order R ctrl2 cry R ctrl2 order R ctrl3 mux sel
R ctrl3 order R ctrl3 cry R ctrl3 order R icr load R icr rden R ccr rden R gcr rden
R sr rden :bool)
(R ad in I be_ Cpu_fail Reset_cpu S state Id ChannelID C ss :words)
(C lA C lB R at I rule_ I last_ I mrdy_ Disable_int Disable_writes Piu_fail Pmm_fail
CB_parity MB_parity :bool).

PH_A_out rep

140
let new_R_fsm_stateA =
((R_fsm__rst) => RI 1)
let new_R_fsm_cntlatch = ((R_fsm_state = RI)
A ~R_fsm_ale_) in
let new_R_fsm_srdy_delA = ~((R_fsm_state = RA) A ~R_fsm_mrdy_) in
let new_R_cntlatch_delA = R__cntlatch_del in
let new_R_srdy_delA_ = R_srdy_del_ in
let new_R_reg_selA = R_reg_sel in
let new_R_reg_selA_ = ((~new_R_srdy_delA_) => (INCN 3 new_R_reg_selA) l new_R_reg_selA) in
let r_write = (~DisableWrites l R_wr l (new_R_fsm_stateA = RD)) in
let r_read = (~R_wr l (new_R_fsm_stateA = RA)) in
let r_cir_wr01 = (r_write ((r_reg_sel = (WORDN 8)) V (r_reg_sel = (WORDN 9))) in
let r_cir_wr23 = (r_write ((r_reg_sel = (WORDN 10)) V (r_reg_sel = (WORDN 11))) in
let new_R_ctr0 = ((R_ctr0_mux_sel) => R.cUO_in l R_etr0_new) in
let new_R_ctr0_ce = (ELEMENT R_gcr (19)) in
let new_R_ctr0 Cain = T in
let new_R_ctr0_outA = R_ctr0_new in
let new_R_ctr1 = ((R_ctr1_mux_sel) => R_ctr1_in l R_ctr1_new) in
let new_R_ctr1 Cain = T in
let new_R_ctr1_outA = R_ctr1_cry in
let new_R_ctr2 = ((R_ctr2_mux_sel) => R_ctr2_in l R_ctr2_new) in
let new_R_ctr2_ce = (ELEMENT R_gcr (23)) in
let new_R_ctr2 Cain = T in
let new_R_ctr2_outA = R_ctr2_new in
let new_R_ctr3 = ((R_ctr3_mux_sel) => R_ctr3_in l R_ctr3_new) in
let new_R_ctr3 Cain = T in
let new_R_ctr3_outA = R_ctr3_cry in
let new_R_ctr3_outA = R_ctr3_new in
let new_R_icr_loadA = R_icr_load in
let new_R_icr_oldA =
((new_R_fsm_stateA = RA) l (r_reg_sel = (WORDN 0)) V (r_reg_sel = (WORDN 1))) => R_icr l R_icr_oldA) in
let new_R_icrA =
((~(r_reg_sel = (WORDN 1))) => Andn rep (R_icr_old, R_icr_mask) l Orn rep (R_icr_old, R_icr_mask)) in
let new_R_int0_en = (((ELEMENT R_icr (0)) A (ELEMENT R_icr (8))) V
((ELEMENT R_icr (1)) A (ELEMENT R_icr (9)))) V
141
((ELEMENT R icr (2)) \ (ELEMENT R icr (10))) \ V
((ELEMENT R icr (3)) \ (ELEMENT R icr (11))) \ V
((ELEMENT R icr (4)) \ (ELEMENT R icr (12))) \ V
((ELEMENT R icr (5)) \ (ELEMENT R icr (13))) \ V
((ELEMENT R icr (6)) \ (ELEMENT R icr (14))) \ V
((ELEMENT R icr (7)) \ (ELEMENT R icr (15))) \ V

let new_R int0_disA = R int0_dis in
let new_R int3_en = (((ELEMENT R icr (16)) \ (ELEMENT R icr (24))) \ V
((ELEMENT R icr (17)) \ (ELEMENT R icr (25))) \ V
((ELEMENT R icr (18)) \ (ELEMENT R icr (26))) \ V
((ELEMENT R icr (19)) \ (ELEMENT R icr (27))) \ V
((ELEMENT R icr (20)) \ (ELEMENT R icr (28))) \ V
((ELEMENT R icr (21)) \ (ELEMENT R icr (29))) \ V
((ELEMENT R icr (22)) \ (ELEMENT R icr (30))) \ V
((ELEMENT R icr (23)) \ (ELEMENT R icr (31))) \ V

let new_R int3_disA = R int3_dis in
let new_R_c01_cout = R ctr1_cry in
let new_R c01_cout_delA = R c01_cout_del in
let new_R c23_cout = R ctr3_cry in
let new_R c23_cout_delA = R c23_cout_del in
let new_R_busA_latch =

((((R ctr0_irden) => R ctr0_in) \ l)
((R ctr0_ord) => R ctr0_out) \ l
((R ctr1_irden) => R ctr1_in) \ l
((R ctr1_ord) => R ctr1_out) \ l
((R ctr2_irden) => R ctr2_in) \ l
((R ctr2_ord) => R ctr2_out) \ l
((R ctr3_irden) => R ctr3_in) \ l
((R ctr3_ord) => R ctr3_out) \ l
((R icr_rden) => R icr) \ l
((R ccr_rden) => R ccr) \ l
((R gcr_rden) => R gcr) \ l
((R sr_rden) => R sr \ ARBN)))))))) \ ))

let new_R fsm_state = R fsm_state in
let new_R fsm_ale_ = R fsm_ale_ in
let new_R fsm_mrdy_ = R fsm_mrdy_ in
let new_R fsm_last_ = R fsm_last_ in
let new_R fsm_rst = R fsm_rst in
let new_R int0_dis = R int0_dis in
let new_R int3_dis = R int3_dis in
let new_R c01_cout_del = R c01_cout_del in
let new_R int1_en = R int1_en in
let new_R c23_cout_del = R c23_cout_del in
let new_R int2_en = R int2_en in
let new_R wr = R wr in
let new_R cout latch del = R cout latch del in
let new_R srdy_del_ = R srdy_del in
let new_R reg sel = R reg sel in
let new_R ctr0_in = R ctr0_in in
let new_R ctr0_mux sel = R ctr0_mux sel in
let new_R ctr0_irden = R ctr0_irden in
let new_R ctr0_cry = R ctr0_cry in
let new_R ctr0_new = R ctr0_new in
let new_R ctr0_out = R ctr0_out in
let new_R_c_orden = R_c_orden in
let new_R_c_1_in = R_c_1_in in
let new_R_c_1_mux_sel = R_c_1_mux_sel in
let new_R_c_1_irden = R_c_1_irden in
let new_R_c_1_cry = R_c_1_cry in
let new_R_c_1_new = R_c_1_new in
let new_R_c_1_out = R_c_1_out in
let new_R_c_1_orden = R_c_1_orden in
let new_R_c_2_in = R_c_2_in in
let new_R_c_2_mux_sel = R_c_2_mux_sel in
let new_R_c_2_irden = R_c_2_irden in
let new_R_c_2_cry = R_c_2_cry in
let new_R_c_2_new = R_c_2_new in
let new_R_c_2_out = R_c_2_out in
let new_R_c_2_orden = R_c_2_orden in
let new_R_c_3_in = R_c_3_in in
let new_R_c_3_mux_sel = R_c_3_mux_sel in
let new_R_c_3_irden = R_c_3_irden in
let new_R_c_3_cry = R_c_3_cry in
let new_R_c_3_new = R_c_3_new in
let new_R_c_3_out = R_c_3_out in
let new_R_c_3_orden = R_c_3_orden in
let new_R_icr_load = R_icr_load in
let new_R_icr_old = R_icr_old in
let new_R_icr_mask = R_icr_mask in
let new_R_icr = R_icr in
let new_R_icr_rden = R_icr_rden in
let new_R_crr = R_crr in
let new_R_crr_rden = R_crr_rden in
let new_R_gcr = R_gcr in
let new_R_gcr_rden = R_gcr_rden in
let new_R_sr = R_sr in
let new_R_sr_rden = R_sr_rden in

let I_ad_out = ((~new_R_wr A ((new_R_fsm_stateA = RA) V (new_R_fsm_stateA = RD))) => new_R_busA_latch I ARBN) in
let I_srdy_ = ((new_R_fsm_stateA = RD) V (new_R_fsm_stateA = RA)) => new_R_fsm_srdy_ I ARB) in
let Int0 = (~new_R_int0_en A ~new_R_int0_disA A ~Disable_int) in
let Int1 = (new_R_c01_cout A new_R_int1_en A ~Disable_int) in
let Int2 = (new_R_c23_cout A new_R_int2_en A ~Disable_int) in
let Int3 = (~new_R_int3_en A ~new_R_int3_disA A ~Disable_int) in
let Ccr = new_R_ccr in
let Led = (SUBARRAY new_R_gcr (3,0)) in
let Reset_error = (ELEMENT new_R_gcr (24)) in
let Pmm_invalid = (ELEMENT new_R_gcr (28)) in

(I_ad_out, I_srdy_, Int0, Int1, Int2, Int3, Ccr, Led, Reset_error, Pmm_invalid)"
);

Next-state definition for Phase-B instruction.
let new_R_wr = ((-I_r Ale_) => (ELEMENT I_ad in (27)) \ I R_wr) in
let new_R_srdy_del = R_fsm_srdy_ in
let new_R_reg_sel =
  ((-I_r Ale_) => (SUBARRAY I_ad in (3,0)))
  ((-I_r sdyl_delA) => (INCN 3 R_reg_selA)) in
let new_R_cn latch_del = R fsm_cn latch in
let r_reg_sel = ((-I_r sdyl_delA) => (INCN 3 R_reg_selA)) in
let r_write = (-Disable writes \ new_R_wr \ (R_fsm_stateA = RD)) in
let r_read = (-new_R_wr \ (R_fsm_stateA = RA)) in
let r_cir_wr01 = (r_write \ ((r_reg_sel = (WORDN 8)) \ V \ (r_reg_sel = (WORDN 9)))) in
let r_cir_wr23 = (r_write \ ((r_reg_sel = (WORDN 10)) \ V \ (r_reg_sel = (WORDN 11)))) in
let new_R_ccr = (r_write \ ((r_reg_sel = (WORDN 3))) => I_ad in l R_ccr) in
let new_R_crr_rden = (r_read \ ((r_reg_sel = (WORDN 3})) in
let new_R_gcr = (r_write \ ((r_reg_sel = (WORDN 2))) => I_ad in l R_gcr) in
let new_R_gcr_rden = (r_read \ ((r_reg_sel = (WORDN 2)))) in
let new_R_ctr0_in = ((r_write \ ((r_reg_sel = (WORDN 8)))) => I_ad in l R_ctr0_in) in
let new_R_ctr0_mux_sel = (r_cir_wr01 \ (ELEMENT new_R_gcr (16)) \ R_c01_cout) in
let new_R_ctr0_irden = (r_read \ ((r_reg_sel = (WORDN 8)))) in
let new_R_ctr0_new = (r_reg_sel \ new_R_ctr0_out \ (INCN 31 R_ctr0) l R_ctr0) in
let new_R_ctr0_cry = (r_reg_sel \ new_R_ctr0_in \ (ONES 31 R_ctr0) \ R_ctr0_cry) in
let new R ctr0 out = ((R fsm cnflatch) => R ctr0 outA | R ctr0 out) in
let new R ctr0 orden = (r read & (r reg sel = (WORDN 12))) in
let new R ctr1 in = ((r write & (r reg sel = (WORDN 9))) => l ad_in | R ctr1 in) in
let new R ctr1 mux sel = (r cir wr01 V ((ELEMENT new R gcr (16)) & R c01 cout)) in
let new R ctr1 irden = (r read & (r reg sel = (WORDN 9))) in
let new R ctr1 new = ((R ctr1 ce & R ctr1 cin) => (INCN 31 R ctr1) | R ctr1) in
let new R ctr1 cry = (R ctr1 ce A R ctr1 cin A (ONES 31 R ctr1)) in
let new R ctr1 out = ((R cnflatch delA) => R ctr1 outA | R ctr1 out) in
let new R ctr1 orden = (r read & (r reg sel = (WORDN 13))) in
let new R ctr2 in = ((r write & (r reg sel = (WORDN 10))) => l ad_in | R ctr2 in) in
let new R ctr2 mux sel = (r cir wr23 V ((ELEMENT new R gcr (20)) & R c23 cout)) in
let new R ctr2 irden = (r read & (r reg sel = (WORDN 10))) in
let new R ctr2 new = ((R ctr2 ce & R ctr2 cin) => (INCN 31 R ctr2) | R ctr2) in
let new R ctr2 cry = (R ctr2 ce A R ctr2 cin A (ONES 31 R ctr2)) in
let new R ctr2 out = ((R fsm cnflatch) => R ctr2 outA | R ctr2 out) in
let new R ctr2 orden = (r read & (r reg sel = (WORDN 14))) in
let new R ctr3 in = ((r write & (r reg sel = (WORDN 11))) => l ad_in | R ctr3 in) in
let new R ctr3 mux sel = (r cir wr23 V ((ELEMENT new R gcr (20)) & R c23 cout)) in
let new R ctr3 irden = (r read & (r reg sel = (WORDN 11))) in
let new R ctr3 new = ((R ctr3 ce & R ctr3 cin) => (INCN 31 R ctr3) | R ctr3) in
let new R ctr3 cry = (R ctr3 ce A R ctr3 cin A (ONES 31 R ctr3)) in
let new R ctr3 out = ((R cnflatch delA) => R ctr3 outA | R ctr3 out) in
let new R ctr3 orden = (r read & (r reg sel = (WORDN 15))) in
let new R icr load = (r write & ((r reg sel = (WORDN 0)) V (r reg sel = (WORDN 1)))) in
let new R icr old =
  ((r write & ((r reg sel = (WORDN 0)) V (r reg sel = (WORDN 1))) => R icr oldA | R icr old) in
let new R icr mask =
  ((r write & ((r reg sel = (WORDN 0)) V (r reg sel = (WORDN 1))) => l ad_in | R icr mask) in
let new R icr = ((R icr loadA) => R icrA | R icr) in
let new R icr rdem = ((R fsm stateA = RA) & ((r reg sel = (WORDN 0)) V (r reg sel = (WORDN 1)))) in
let sr28 = (ALTER ARB (28) MB parity) in
let sr28 25 = (MALTER sr28 (27,25) C ss) in
let sr28 24 = (ALTER sr28 25 (24) CB parity) in
let sr28 22 = (MALTER sr28 24 (23,22) ChannelID) in
let sr28 16 = (MALTER sr28 22 (21,16) Id) in
let sr28 12 = (MALTER sr28 16 (15,12) S state) in
let sr28 9 = (ALTER sr28 12 (9) Pmms fail) in
let sr28 8 = (ALTER sr28 9 (8) Piu fail) in
let sr28 2 = (MALTER sr28 8 (3,2) Reset cpu) in
let sr28 0 = (MALTER sr28 2 (1,0) Cpu fail) in
let new R sr = ((R fsm cnflatch) => sr28 0 | R sr) in
let new R sr rdem = (r read & (r reg sel = (WORDN 4))) in
let new R int0 dis = R int0 en in
let new R int3 dis = R int3 en in
let new R c01 cout del = R c01 cout in
let new R c23 cout del = R c23 cout in
let new R int1 en =
  (((ELEMENT new R gcr (18)) & (r cir wr01 V (R c01 cout & (ELEMENT new R gcr (16)))))
  & (-(ELEMENT new R gcr (18)) V ((ELEMENT new R gcr (17)) & R c01 cout del))) => T |
  (((ELEMENT new R gcr (18)) & (r cir wr01 V (R c01 cout & (ELEMENT new R gcr (16)))))
  & (-(ELEMENT new R gcr (18)) V ((ELEMENT new R gcr (17)) & R c01 cout del))) => F |
  (((ELEMENT new R gcr (18)) & (r cir wr01 V (R c01 cout & (ELEMENT new R gcr (16)))))
  & (-(ELEMENT new R gcr (18)) V ((ELEMENT new R gcr (17)) & R c01 cout del))) => R int1 en | ARB)) in
let new R int2 en =

145
let new_R_fsm_state = R fsm stateA in
let new_R fsm ale_ = Ira in
let new R fsm tardy_ = Lmrdy in
let new_R fsm rst = Rst in
let new_R fsm srdy_ = R fsm srdy_ in
let new_R int0_disA = R int0 DisA in
let new_R ctr0 = R ctr0 in
let new_R ctr0 ce = R ctr0 ce in
let new_R ctr0 cin = R ctr0 cin in
let new_R ctr0 ouA = R ctr0 outA in
let new_R ctr1 = R ctr1 in
let new_R ctr1 ce = R ctr1 ce in
let new_R ctr1 cin = R ctr1 cin in
let new_R ctr1 outA = R ctr1 outA in
let new_R ctr2 = R ctr2 in
let new_R ctr2 ce = R ctr2 ce in
let new_R ctr2 cin = R ctr2 cin in
let new_R ctr2 outA = R ctr2 outA in
let new_R ctr3 = R ctr3 in
let new_R ctr3 ce = R ctr3 ce in
let new_R ctr3 cin = R ctr3 cin in
let new_R ctr3 outA = R ctr3 outA in
let new_R icr loadA = R icr loadA in
let new_R icr oldA = R icr oldA in
let new_R icr = R icrA in
let new_R busA latch = R busA latch in

 cú (%) \((\text{new}_R\text{ fsm stateA, new}_R\text{ fsm cntlatch, new}_R\text{ fsm srdy_}, \text{new}_R\text{ int0_disA, new}_R\text{ int0_disA, new}_R\text{ int3_en, new}_R\text{ int3_disA, new}_R\text{ c01_cout, new}_R\text{ c01_cout_delA, new}_R\text{ c23_cout, new}_R\text{ c23_cout_delA, new}_R\text{ cntlatch_delA, new}_R\text{ srdy_delA_ = R srdy_delA_, in let new_R_reg_selA = R_reg_selA in let new_R_ctr0 = R ctr0 in let new_R_ctr0 ce = R ctr0 ce in let new_R_ctr0 cin = R ctr0 cin in let new_R_ctr0 outA = R ctr0 outA in let new_R_ctr1 = R ctr1 in let new_R_ctr1 ce = R ctr1 ce in let new_R_ctr1 cin = R ctr1 cin in let new_R_ctr1 outA = R ctr1 outA in let new_R_ctr2 = R ctr2 in let new_R_ctr2 ce = R ctr2 ce in let new_R_ctr2 cin = R ctr2 cin in let new_R_ctr2 outA = R ctr2 outA in let new_R_ctr3 = R ctr3 in let new_R_ctr3 ce = R ctr3 ce in let new_R_ctr3 cin = R ctr3 cin in let new_R_ctr3 outA = R ctr3 outA in let new_R_icr_loadA = R icr loadA in let new_R_icr oldA = R icr oldA in let new_R_icr = R icrA in let new_R_busA latch = R busA latch in

(cu %)
let PH_B_out_def = new_definition

let PH_B_out = new_definition

let new_R_wr = ((~I_ra/e_) => (ELEMENT I_ad_in (27)) \rightarrow R_wr) in
let new_R_srdy_del_ = R_fsm_srdy_ in
let new_R_reg_sel = ((~I_rale_) => (SUBARRAY I_ad_in (3,0))) \rightarrow

let new_R_ctr0_in = R_fsm_ctr0_in in
let new_R_ctr0_mux_sel = R_fsm_ctr0_mux_sel in
let new_R_ctr0_irden = R_fsm_ctr0_irden in
let new_R_ctr0_cry = R_fsm_ctr0_cry in
let new_R_ctr0_new = R_fsm_ctr0_new in
let new_R_ctr0_out = R_fsm_ctr0_out in
let new_R_ctr0_orden = R_fsm_ctr0_orden in
let new_R_ctr0_ce = R_fsm_ctr0_ce in
let new_R_ctr0_cin = R_fsm_ctr0_cin in
let new_R__ctrl_mux_sel = R_fsm__ctrl_mux_sel in
let new_R_ctrl_orden = R_fsm_ctrl_orden in
let new_R_ctrl_ce = R_fsm_ctrl_ce in
let new_R_ctrl_cin = R_fsm_ctrl_cin in
let new_R_ctrl_new = R_fsm_ctrl_new in
let new_R__ctrl = R_fsm__ctrl in
let new_R__ctrl_irden = R_fsm__ctrl_irden in
let new_R_ctrl_cry = R_fsm_ctrl_cry in
let new_R_controller = R_fsm_controller in

let new_R_ctr2_in = R_fsm_ctr2_in in
let new_R_ctr2_mux_sel = R_fsm_ctr2_mux_sel in
let new_R_ctr2_irden = R_fsm_ctr2_irden in
let new_R_ctr2_cry = R_fsm_ctr2_cry in
let new_R_ctr2_new = R_fsm_ctr2_new in
let new_R_ctr2_out = R_fsm_ctr2_out in
let new_R_ctr2_orden = R_fsm_ctr2_orden in
let new_R_ctr3_in = R_fsm_ctr3_in in
let new_R_ctr3_mux_sel = R_fsm_ctr3_mux_sel in
let new_R_ctr3_irden = R_fsm_ctr3_irden in
let new_R_ctr3_cry = R_fsm_ctr3_cry in
let new_R_ctr3_new = R_fsm_ctr3_new in
let new_R_ctr3_out = R_fsm_ctr3_out in
let new_R_ctr3_orden = R_fsm_ctr3_orden in
let new_R_icr_load = R_fsm_icr_load in
let new_R_icr_mask = R_fsm_icr_mask in
let new_R_icr = R_fsm_icr in
let new_R_icr_rden = R_fsm_icr_rden in
let new_R_ccr = R_fsm_ccr in
let new_R_ccr_rden = R_fsm_ccr_rden in
let new_R_gcr = R_fsm_gcr in
let new_R_gcr_rden = R_fsm_gcr_rden in
let new_R_sr = R_fsm_sr in
let new_R_sr_rden = R_fsm_sr_rden in

let new_R_wr = ((~I_ra/e_) => (ELEMENT I_ad_in (27)) \rightarrow R_wr) in
let new_R_srdy_del_ = R_fsm_srdy_ in
let new_R_reg_sel = ((~I_rale_) => (SUBARRAY I_ad_in (3,0))) \rightarrow

let new_R_wr = ((~I_ra/e_) => (ELEMENT I_ad_in (27)) \rightarrow R_wr) in
let new_R_srdy_del_ = R_fsm_srdy_ in
let new_R_reg_sel = ((~I_rale_) => (SUBARRAY I_ad_in (3,0))) \rightarrow
let new_R_ctrl__orden = (r_read A (r_reg__sel = (WORDN 13))) in
let new_R_ctr2__mux_sel = (Lcir_wr23 V ((ELEMENT new_R_gcr (20)) ^ R_ctr23_cout)) in
let new_R_ctr2_orden = (r_read A (r_reg__sel = (WORDN 14))) in
let new_R_ctr3_in = ((r_write A ((r_reg__sel = (WORDN 8)) V (r_reg__sel = (WORDN 9)))) => I_ad_in l R_ctr3_in) in
let new_R_ctr3_mux_sel = (r_cir_wr23 V ((ELEMENT new_R_gcr (16)) V R_ctr3_in)) in
let new_R_ctr3_new = ((R_ctr3_ce A R_ctr3_cin) => (INCN 31 R_ctr3) I R_ctr3) in
let new_R_ctr3_cry = (R_ctr3_cin A R_ctr3_outA A (ONES 31 R_ctr3)) in
let new_R_ctr3_out = ((R fsm_cntlatch_delA) => R_ctr3_outA I R_ctr3_out) in
let new_R_ctr3_orden = (r_read A (r_reg__sel = (WORDN 15))) in
let new_R_icr_load = (r_write A ((r_reg__sel = (WORDN 0)) V (r_reg__sel = (WORDN 1)))) in
let new_R_icr_old = 
  (((r_write A ((r_reg__sel = (WORDN 0)) V (r_reg__sel = (WORDN 1)))) => R_icr_oldA I R_icr_old) in
let new_R_icr_mask = 
  (((r_write A ((r_reg__sel = (WORDN 0)) V (r_reg__sel = (WORDN 1)))) => I_ad_in l R_icr_mask) in
let new_R_icr_rden = ((R fsm_stateA = RA) => ((r_reg__sel = (WORDN 0)) V (r_reg__sel = (WORDN 1)))) in
let sr28 = (ALTER ARBN (28) MB parity) in
let sr28_25 = (MALTED sr28 (27.25) C_state) in
let sr28_24 = (ALTER sr28_25 (24) CB_parity) in
let sr28_22 = (MALTED sr28_24 (23.22) ChannelID) in
let sr28_16 = (MALTED sr28_22 (21.16) ld) in
let sr28_12 = (MALTED sr28_16 (15.12) S_state) in
let sr28_9 = (ALTER sr28_12 (9) PmM fail) in
let sr28_8 = (ALTER sr28_9 (8) Piu fail) in

148
let sr28_2 = (MALTER sr28_8 (3,2) Reset_cpu) in
let sr28_0 = (MALTER sr28_2 (1,0) Cpu_fail) in
let new R sr = ((R fsm_cntlatch => sr28_0 | R sr) in
let new R sr_rden = (r_read ∧ (r_reg sel = (WORDN 4))) in
let new R int0_dis = R int0_en in
let new R int3_dis = R int3_en in
let new R c01_cout_del = R c01_cout in
let new R c23_cout_del = R c23_cout in
let new R int1_en =
    (((ELEMENT new R gcr (18)) ∧ (r_cir_wr01 V (R c01_cout ∧ (ELEMENT new R gcr (16)))))
    ∧ ¬((ELEMENT new R gcr (18)) V ((ELEMENT new R gcr (17)) ∧ R c01_cout_del))) => T I
    ((−(ELEMENT new R gcr (18)) ∧ (r_cir_wr01 V (R c01_cout ∧ (ELEMENT new R gcr (16)))))
    ∧ ¬((ELEMENT new R gcr (18)) V ((ELEMENT new R gcr (17)) ∧ R c01_cout_del))) => F I
    ((−(ELEMENT new R gcr (18)) ∧ (r_cir_wr01 V (R c01_cout ∧ (ELEMENT new R gcr (16)))))
    ∧ ¬((ELEMENT new R gcr (18)) V ((ELEMENT new R gcr (17)) ∧ R c01_cout_del))) => R int1_en I ARB)) in
let new R int2_en =
    (((ELEMENT new R gcr (22)) ∧ (r_cir_wr23 V (R c23_cout ∧ (ELEMENT new R gcr (20)))))
    ∧ ¬((ELEMENT new R gcr (22)) V ((ELEMENT new R gcr (21)) ∧ R c23_cout_del))) => T I
    ((−(ELEMENT new R gcr (22)) ∧ (r_cir_wr23 V (R c23_cout ∧ (ELEMENT new R gcr (20)))))
    ∧ ¬((ELEMENT new R gcr (22)) V ((ELEMENT new R gcr (21)) ∧ R c23_cout_del))) => F I
    ((−(ELEMENT new R gcr (22)) ∧ (r_cir_wr23 V (R c23_cout ∧ (ELEMENT new R gcr (20)))))
    ∧ ¬((ELEMENT new R gcr (22)) V ((ELEMENT new R gcr (21)) ∧ R c23_cout_del))) => R int2_en I ARB)) in
let new R fsm_state = R fsm_stateA in
let new R fsm_ale_ = l_ale_ in
let new R fsm_mrdy_ = l_mrdy_ in
let new R fsm_last_ = l_last_ in
let new R fsm_rst = Rst in
let new R fsm_stateA = R fsm_stateA in
let new R fsm_cntlatch = R fsm_cntlatch in
let new R fsm_srty_ = R fsm_srty_ in
let new R int0_en = R int0_en in
let new R int0_dis = R int0_dis in
let new R int3_en = R int3_en in
let new R int3_dis = R int3_dis in
let new R c01_cout = R c01_cout in
let new R c01_cout_delA = R c01_cout_delA in
let new R c23_cout = R c23_cout in
let new R c23_cout_delA = R c23_cout_delA in
let new R cntlatch_delA = R cntlatch_delA in
let new R rdy_delA = R rdy_delA in
let new R reg selA = R reg selA in
let new R ctrl0 = R ctrl0 in
let new R ctrl0 ce = R ctrl0 ce in
let new R ctrl0 cin = R ctrl0 cin in
let new R ctrl0 outA = R ctrl0 outA in
let new R ctrl1 = R ctrl1 in
let new R ctrl1 ce = R ctrl1 ce in
let new R ctrl1 cin = R ctrl1 cin in
let new R ctrl1 outA = R ctrl1 outA in
let new R ctrl2 = R ctrl2 in
let new R ctrl2 ce = R ctrl2 ce in
let new R ctrl2 cin = R ctrl2 cin in
let new R ctrl2 outA = R ctrl2 outA in
let new R ctrl3 = R ctrl3 in
let new R_ctr3_ce = R_ctr3_ce in
let new R_ctr3_cin = R_ctr3_cin in
let new R_ctr3_outA = R_ctr3_outA in
let new R_icr_loadA = R_icr_loadA in
let new R_icr_oldA = R_icr_oldA in
let new R_icrA = R_icrA in
let new R_busA_latch = R_busA_latch in

let I_ad_out = ((-new R_wr A ((new_R_fsm_stateA = RA) V (new_R_fsm_stateA = RD))) => new_R_busA_latch I ARBN) in
let I__srdy_ = (((new_R_fsm_stateA = RD) V (new_R_fsm_stateA = RA))) => new_R_fsm_srdy_1 ARB) in
let Int0_ = -(new_R_int0_en A -new_R_int0_disA A ~Disable_int) in
let Int1 = (new_R_c01_cout A new_R_int1_en A ~Disable_int) in
let Int2 = (new_R_c23_cout A new_R_int2_en A ~Disable_int) in
let Int3_ = -(new_R_int3_en A -new_R_int3_disA A ~Disable_int) in
let Ccr = new_R_ccr in
let Led = (SUBARRAY new_R_gcr (3,0)) in
let Reset_error = (ELEMENT new_R_gcr (24)) in
let Pmm_invalid = (ELEMENT new_R_gcr (28)) in

(I_ad_out, I__srdy_, Int0_, Int1, Int2, Int3_, Ccr, Led, Reset_error, Pmm_invalid))

close_theory();
C.4 C Port Specification

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

```ml
set_searchpath (search_path) @ ['/home/titan3/dfura/tep/piu/hol/lib/']);;

system 'rm c_phase.th';;
new_theory 'c_phase';;
loadf 'abstract';;
map new_parent ['caux_def'; 'aux_def'; 'array_def'; 'wordn_def'];:

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

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

let c_state_ty = "(cmfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
wordn#bool#bool#bool#bool#bool#
cfsm_ty#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
cfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
cfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
cfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
cfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
cfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
cfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
cfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#
cfsm_ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#"

let c_state = "((C_mfsm_stateA, C_mfsm_mabort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
C_sfsm_sale, C_sfsm_sdl1, C_sfsm_sd0, C_sfsm_sack, C_sfsm_sabort, C_sfsm_s_out_set0, C_sfsm_sparity,
C_sfsm_stateA, C_sfsm_rdy_en,
C_clkAA, C_sidle_delA, C_mrqt_delA, C_last_inA, C_ssaA, C_holdA, C_cout_0_le_delA,
C_cin_2_leA, C_mrdy_delA, C_iad_en_s_delA, C_wrdyA, C_rdbyA, C_iad_outC_a1a0, C_a3a2,
C_mfsm_stateC, C_mfsm_rdy_en, C_mfsm_rst, C_mfsm_busy, C_mfsm_write,
C_mfsm_cqtn, C_mfsm_hold, C_mfsm_lastC_mfsm_lock, C_mfsm_ss, C_mfsm_invalid,
C_sfsm_stateC, C_mfsm_sdl1, C_mfsm_said, C_mfsm_swait, C_mfsm_swait0, C_mfsm_swait1, C_mfsm_swait2,
C_mfsm_stateC, C_mfsm_cqtn, C_mfsm_rst, C_mfsm_write, C_mfsm_addressed, C_mfsm_hlda, C_mfsm_ms,
C_mfsm_stateC, C_mfsm_cqtn, C_mfsm_male, C_mfsm_male, C_mfsm_rst, C_mfsm_rst,
C_mfsmapellido, C_mfsm_request, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male, C_mfsm_male,
C._efsm._cale_ C._efsm_last_ C._efsm_male_ C._efsm_rale_ C._efsm_srdy_ C._efsm_rst
C._wry C._clkA C._sidle_del C._mrqt_del C._last_in_ C._lock_in_ C._last_out_
C._hold_ C._cout_0_le_del C._cin_2_le C._rardy_del C._iad_en_s_del C._wrdy
C._rardy C._parity = :bool

(l._rardy_in_ l._rale_in_ l._male_in_ l._last_in_ l._srdy_in_ l._lock_ l._cale_ l._idle del
l._mrqt C._mrqt C._pmm_failure Piu._invalid Reset_error :bool)
(l._cgnt_ l._ruddy_out_ l._hold_ l._rale_out_ l._male_out_ l._last_out_ l._srdy_out_ C._rqt_out
Disable_writes CB._parity : bool).

PH_A inst rep
(C_mfsm_stateA, C_mfsm_abort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
C_mfsm_ma1, C_mfsm_ms0, C_mfsm_ms0, C_mfsm_mrdy, C_mfsm_cid, C_mfsm_mcout_sel1,
C_mfsm_mcout_sel0, C_mfsm_ms, C_mfsm_rqt, C_mfsm_cgo, C_mfsm_cm_en, C_mfsm_abort_le_en,
C_mfsm_mparity, C_sfsm_stateA, C_sfsm_ss, C_sfsm_iad_en_s, C_sfsm_sidle, C_sfsm_slock,
C_sfsm_scout, C_sfsm_saval, C_sfsm_sval, C_sfsm_srd, C_sfsm_smad, C_sfsm_sabort,
C_sfsm_scout0, C_sfsm_sparity, C_efsm_stateA, C_efsm_srdy_en, C_clkAA, C_sidle_delA,
C_mrqt_delA, C_last_inA, C_ssA, C_holdA, C_srdyA, C_rddyA, C_iad_out, C_a3a2, C_mfsm_state,
C_mfsm_srdy_en, C_mfsm_D, C_mfsm_grant, C_mfsm_rst, C_mfsm_busy, C_mfsm_write, C_mfsm_cqt,
C_mfsm_hold, C_mfsm_last, C_mfsm_lock, C_mfsm_ss, C_mfsm_invalid, C_sfsm_state, C_sfsm_D,
C_sfsm_grant, C_sfsm_rst, C_sfsm_write, C_sfsm_addressed, C_sfsm_hlda, C_sfsm_ms,
C_efsm_state, C_efsm_cale, C_efsm_last, C_efsm_male, C_efsm_rale, C_efsm_srdy,
C怦rms, C.wr, C.sidl, C.erq, C.mrdy_del, C_last_in, C_lock_in,
C_ss, C.last_out, C.hold, C.cout_0_le_del, C.cin_2_le, C.mrdy_del, C.iad_en_s_del, C.wrdy,
C._rdy, C._parity, C._source, C_data_in, C_iad_in)
(l._ad_in_ l.be_in_ l.mrdy_in_ l.rale_in_ l.male_in_ l.last_in_ l.srdy_in_ l.lock_,
LA_cale_ l._idle del, L._rqt del, CB._rqt_in, CB.ad_in, CB.ms_in, CB.ss_in, Rst, ClikA, ClikB,
ClikD, Id, ChannelID, Pmm_failure Piu._invalid Ccr Reset_error :bool)

let new_C_mfsm_stateA =
((C_mfsm_rst) => CM1 | CMI
((C_mfsm_state = CMI) => (C_mfsm_D A -C_mfsm_crt A -C_mfsm_busy A -C_mfsm_invalid) => CMR | CM1 | CMI
((C_mfsm_state = CMR) => (C_mfsm_D A C_mfsm_grant A C_mfsm_hold) => CMA3 | CMR | CM1
((C_mfsm_state = CMA3) => (C_mfsm_D => CMA1 | CMA3 | CM1
((C_mfsm_state = CMA1) =>
(C_mfsm_D A (C_mfsm_ss = ^SRDY)) => CMA0 | CM0
(C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT | CMA1
((C_mfsm_state = CM0) =>
(C_mfsm_D A (C_mfsm_ss = ^SRDY)) => CMA0 | CM0
(C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT | CMA0
((C_mfsm_state = CMA2) =>
(C_mfsm_D A (C_mfsm_ss = ^SRDY)) => CMD1 | CM1
(C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT | CMD1
((C_mfsm_state = CMD1) =>
(C_mfsm_D A (C_mfsm_ss = ^SRDY)) => CMD0 | CM0
(C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT | CMD0
((C_mfsm_state = CMD0) =>
(C_mfsm_D A (C_mfsm_ss = ^SRDY) A C_mfsm_last) => CMD1 | CM0
(C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT | CMD1
((C_mfsm_state = CMW) =>
(C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT | CMW
(C_mfsm_D A (C_mfsm_ss = ^SACK) A C_mfsm_lock) => CM1
(C_mfsm_D A (C_mfsm_ss = ^SRDY) A -C_mfsm_lock A -C_mfsm_crt) => CMA3 | CMW

153
let new_C_mfsm_mabort = (new_C_mfsm_stateA = CMABT) in
let new_C_mfsm_midle = (new_C_mfsm_stateA = CMI) in
let new_C_mfsm_mrequest = (new_C_mfsm_stateA = CMR) in
let new_C_mfsm_m2 = (new_C_mfsm_stateA = CMRA) in
let new_C_mfsm_m0 = (new_C_mfsm_stateA = CMD0) in
let new_C_mfsm_m10 = (new_C_mfsm_stateA = CMD1) in
let new_C_mfsm_m1 = (new_C_mfsm_stateA = CMDI) in
let new_C_mfsm_m00 = (new_C_mfsm_stateA = CMD0) in
let new_C_mfsm_iadr_en_m = (((new_C_mfsm_stateA = CMDI) ∧ ¬C_mfsm_write ∧ C_mfsm_srdy_en)
V ((new_C_mfsm_stateA = CMD0) ∧ ¬C_mfsm_write ∧ C_mfsm_srdy_en)
V ((new_C_mfsm_stateA = CMW) ∧ (C_mfsm_state = CMD0) ∧ ¬C_mfsm_write
∧ C_mfsm_srdy_en)) in
let new_C_mfsm_m_cnt_sell = ((new_C_mfsm_stateA = CMIA) V (new_C_mfsm_stateA = CM2)) in
let new_C_mfsm_m_cnt_sell0 = ((new_C_mfsm_stateA = CMIA) V (new_C_mfsm_stateA = CM1)
V (new_C_mfsm_stateA = CMD1)) in
let ms2 = (ALTER ARBN (2) ((new_C_mfsm_stateA = CMIA) V (new_C_mfsm_stateA = CM2))
V (new_C_mfsm_stateA = CM0) V (new_C_mfsm_stateA = CM2)
V (new_C_mfsm_stateA = CMW) V (new_C_mfsm_stateA = CMABT)) in
let ms1 = (ALTER ms2 (1) ((new_C_mfsm_stateA = CMIA) V (new_C_mfsm_stateA = CM2)
V (new_C_mfsm_stateA = CM0) V (new_C_mfsm_stateA = CMABT)) in
let ms0 = (ALTER ms1 (0) ((new_C_mfsm_stateA = CMD0) ∧ ¬C_mfsm_last_)
V (new_C_mfsm_stateA = CMW) ∧ C_mfsm_lock_) in
let new_C_mfsm_ms = ms0 in
let new_C_mfsm_rq_ = ¬¬(new_C_mfsm_stateA = CMI) in
let new_C_mfsm_cgnl_ = ¬¬(new_C_mfsm_stateA = CMA3) in
let new_C_mfsm_abort = (¬¬(new_C_mfsm_stateA = CMIA) ∧ ¬¬(new_C_mfsm_stateA = CM2)) in
let new_C_mfsm_impair = ((new_C_mfsm_stateA = CMIA) V (new_C_mfsm_stateA = CM2)
V (new_C_mfsm_stateA = CM0) V (new_C_mfsm_stateA = CM2)
V (C_mfsm_state = CMA1) V (C_mfsm_state = CMA0)
V (C_mfsm_state = CMA2) V (C_mfsm_state = CMD1)) in

let new_C_sfsm_stateA =
((C_sfsm_rst) ⇒ CSI)
(C_sfsm_state = CSI) ⇒ ((C.sfsm_D ∧ C.sfsm_ms = MSTART)
∧ ¬C_sfsm_grant ∧ C_sfsm_addressed) ⇒ CSA1 | CSI)
(C_sfsm_state = CSL) ⇒
((C.sfsm_D ∧ C.sfsm_ms = MSTART) ∧ ¬C_sfsm_grant ∧ C_sfsm_addressed) ⇒ CSA1 | CSI)
((C.sfsm_D ∧ C.sfsm_ms = MSTART) ∧ ¬C_sfsm_grant ∧ ¬C.sfsm_addressed) ⇒ CSI | CSL)
(C_sfsm_state = CSA1) ⇒
((C.sfsm_D ∧ C.sfsm_ms = MRDY) ⇒ CSA01)
(C.sfsm_D ∧ C.sfsm_ms = MRDY) ⇒ CSA01 | CSI)
((C.sfsm_D ∧ C.sfsm_ms = MBORT) ⇒ CSA01 | CSI)
(C.sfsm_state = CSA0) ⇒
((C.sfsm_D ∧ C.sfsm_ms = MRDY) ∧ ¬C.sfsm_hlda_) ⇒ CSALE |
(C.sfsm_D ∧ C.sfsm_ms = MRDY) ∧ C.sfsm_hlda_ ⇒ CSA0W |
(C.sfsm_D ∧ C.sfsm_ms = MBORT) ⇒ CSABT | CSA0)
let ss2 = (ALTER ARBN (2) ((\(new\_C\_sfsm\_stateA = CSI\)) \& \((\neg (new\_C\_sfsm\_stateA = CSABT))\))) in
let ss1 = (ALTER ss2 (1) ((\(new\_C\_sfsm\_stateA = CSI\)) \& \((\neg (new\_C\_sfsm\_stateA = CSACK))\)) \& \((\neg (new\_C\_sfsm\_stateA = CSABT))\))) in
let ss0 = (ALTER ss1 (0) ((\(new\_C\_sfsm\_stateA = CSAW\)) \& \((\neg (new\_C\_sfsm\_stateA = CSLE))\)) \& \((\neg (new\_C\_sfsm\_stateA = CSACK))\)) in
let new\_C\_sfsm\_ss = ss0 in
let new\_C\_sfsm\_isad\_en_s = ((\(new\_C\_sfsm\_stateA = CSLE\)) \& \((\neg C\_sfsm\_write)\)) in
let new\_C\_sfsm\_sidle = (new\_C\_sfsm\_stateA = CSI) in
let new\_C\_sfsm\_slock = (new\_C\_sfsm\_stateA = CSL) in
let new\_C\_sfsm\_ss = (new\_C\_sfsm\_stateA = CSA1) in
let new\_C\_sfsm\_ss0 = (new\_C\_sfsm\_stateA = CSA0) in
let new\_C\_sfsm\_sle = (new\_C\_sfsm\_stateA = CSLE) in
let new\_C\_sfsm\_sd0 = (new\_C\_sfsm\_stateA = CSD0) in
let new\_C\_sfsm\_sack = (new\_C\_sfsm\_stateA = CSACK) in
let new\_C\_sfsm\_sab = (new\_C\_sfsm\_stateA = CSABT) in
let new\_C\_sfsm\_squote = (new\_C\_sfsm\_stateA = CSI) \& \((\neg (new\_C\_sfsm\_stateA = CSACK))\) in
let new\_C\_sfsm\_stateA =
((C\_efsm\_rst) \& \((\neg C\_sfsm\_write)\)) \& \((\neg C\_sfsState)\) \& \((\neg C\_sfsm\_write)\) in
let cout\_sel0 = (ALTER ARBN (0) ((\(new\_C\_sfsm\_sd1 \& \(new\_C\_sfsm\_sd0\))\)) in
let cout_sell = (ALTER cout_sell (1)) ((new C_sfsm_sd1 V new C_sfsm_sd0) => F1 new C_mfsm_m_cout_sell)) in
let c_cout_sell = cout_sell in
let c_busy = (~((SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
let c_grant = (((SUBARRAY CB_ (I,0)) = (WORDN 0)) \ (ELEMENT CB_rqt_in_ (0))) \ (ELEMENT CB_rqt_in_ (1))) \ (ELEMENT CB_rqt_in_ (2))) \ (ELEMENT CB_rqt_in_ (3))) in
let c_write = ((new C_mfsm_cm_en) --> C_wr (ELEMENT C_sizewrbe (3))) in
let new C_clkAA = C_clkA in
let new C_sidle_delA = C_sidle_del in
let new_C_mrqt_delA = C_mrqt_del in
let c_dfsm_srdy = (CB_ss_in = $SRDY$) in
let c_dfsm_slave = (-new C_sfsm_sidle \ new C_sfsm_slock) in
let c_dfsm_cin_0_le = (ClkD \ ((new C_mfsm_md0 \ c_dfsm_srdy \ c_write) \ (new C_sfsm_sd0 V new C_sfsm_md0))) in
let c_dfsm_cin_1_le = (ClkD \ ((new C_mfsm_md1 \ c_dfsm_srdy \ c_write) \ (new C_sfsm_sd1 \ c_write) \ (new C_sidle V new C_sfsm_sd0))) in
let c_dfsm_cin_3_le = (ClkD \ (new C_sidle V new C_sfsm_sd0)) in
let c_dfsm_cin_4_le = (new C_clkAA \ new C_sfsm_sd0) in
let c_dfsm_cout_0_le = (C_wr \ (ELEMENT C_sizewrbe (3))) in
let c_dfsm_cad_en = (new C_clkAA \ new C_sfsm_sd1) in
let c_dfsm_cla = (-new C_sfsm_sidle \ ((SUBARRAY C_sizewrbe (1,0)) = (WORDN 3))) \ new C_clkAA) in
let c_dfsm_cla = (-new C_sfsm_sidle \ ((SUBARRAY C_sizewrbe (1,0)) = (WORDN 3))) \ new C_clkAA) in
let c_dfsm_cla = (-new C_sfsm_sd0) \ new C_clkAA) in
let new C_last_inA = 1_last_in in
let new C_ssA = CB_ss in
let new C_holdA = (ClkD) => C_hold \ C_holdA in
let new C_cout_0_le_delA = C_cout_0_le_del in
let new C_cin_2_leA = C_cin_2_le in
let new C_mrdy_delA = C_mrdy_del in
let new C_iad_en_s_delA = (ClkD) => C_iad_en_s_del \ C_iad_en_s_delA in
let new C_wr_0A = C_wr_0 in
let new C_iad_out = (new C_cin_2_leA) => C_data_in \ C_iad_out in
let new C_a1a0 = ((c_dfsm_master \ new C_cout_0_le_delA) \ (-c_dfsm_master \ c_dfsm_cout_1_le)) => C_iad_in \ C_a1a0 in
let new C_mfsm_state = C_mfsm_state in
let new C_mfsm_srdy_en = C_mfsm_srdy_en in
let new C_mfsm_srdy = C_mfsm_srdy in
let new C_mfsm_grant = C_mfsm_grant in
let new C_mfsm_busy = C_mfsm_busy in
let new C mfsm_write = C mfsm_write in
let new C mfsm crqt_ = C mfsm crqt_ in
let new C mfsm hold_ = C mfsm hold_ in
let new C mfsm last_ = C mfsm last_ in
let new C mfsm lock_ = C mfsm lock_ in
let new C mfsm ss = C mfsm ss in
let new C mfsm invalid = C mfsm invalid in
let new C sfsm state = C sfsm state in
let new C sfsm D = C sfsm D in
let new C sfsm grant = C sfsm grant in
let new C sfsm rst = C sfsm rst in
let new C sfsm_write = C sfsm write in
let new C sfsm addressed = C sfsm addressed in
let new C sfsm hlda_ = C sfsm hlda_ in
let new C sfsm ms = C sfsm ms in
let new C sfsm state = C sfsm state in
let new C sfsm_cale_ = C sfsm_cale_ in
let new C sfsm_last_ = C sfsm_last_ in
let new C sfsm_male_ = C sfsm_male_ in
let new C sfsm rale_ = C sfsm_rale_ in
let new C sfsm srty_ = C sfsm srty_ in
let new C sfsm rst = C sfsm rst in
let new C wr = C wr in
let new C sizewrbe = C sizewrbe in
let new C clkA = C clkA in
let new C sidle_del = C sidle_del in
let new C mrqt_del = C mrqt_del in
let new C last_in_ = C last_in_ in
let new C lock_in_ = C lock_in_ in
let new C ss = C ss in
let new C last_out_ = C last_out_ in
let new C hold_ = C hold_ in
let new C cout_0_le_del = C cout_0_le_del in
let new C cin_2_le = C cin_2_le in
let new C mrdy_del_ = C mrdy_del_ in
let new C iad_en_s_del = C iad_en_s_del in
let new C wrdy = C wrdy in
let new C rrdy = C rrdy in
let new C parity = C parity in
let new C source = C source in
let new C data_in = C data_in in
let new C iad_in = C iad_in in

(new C mfsm_stateA, new C mfsm_mabort, new C mfsm_midle, new C mfsm_mrequest, new C mfsm_ma3,
new C mfsm ma2, new C mfsm ma1, new C mfsm ma0, new C mfsm md1, new C mfsm md0,
new C mfsm iad_en_m,
new C mfsm m cout sel1, new C mfsm m cout sel0, new C mfsm ms, new C mfsm rqt_, new C mfsm cgot_,
new C mfsm cm_en, new C mfsm abort_le_en_, new C mfsm_mparity, new C sfsm stateA, new C sfsm ss,
new C sfsm iad_en_s, new C sfsm sidle, new C sfsm slck, new C sfsm sa1, new C sfsm sa0,
new C sfsm sale, new C sfsm sel1, new C sfsm sd1, new C sfsm sack, new C sfsm sabort,
new C sfsm s cout sel0, new C sfsm_sparity, new C sfsm_stateA, new C sfsm srty_en, new C clkAA,
new C sidle delA, new C mrqt delA, new C last_inA_, new C ssA, new C holdA_,
new C cout_0_le delA, new C cin_2_leA, new C mrdy delA_, new C iad_en_s delA, new C wrdyA, new C rrdyA,
new C iad out, new C a1a0, new C a3 a2, new C mfsm state, new C mfsm srty_en, new C mfsm D)
Output definition for Phase-A instruction.

let PH_A_out_def = new_definition
('PH_A_out',
"(rep:_cp_ty)
(C_mfsm_stateA C_mfsm_state :c_fsm_ty)
(C_sfsm_stateA C_sfsm_state :csfsm_ty)
(C_efsm_stateA C_efsm_state :cefsm_ty)
(C_mfsm_ms C_sfsm_ms C_ssA C_sfsm_ms C_sizewrbe C_ss
C_source C_data_in C_iad_in)"
);
let new_C_mfsm_stateA =
  ((C_mfsm_rst) => CMR)
  (C_mfsm_state = CMI) => (C_mfsm_D & ~C_mfsm_cqkt & ~C_mfsm_busy & ~C_mfsm_invalid) => CMR
  (C_mfsm_state = CRM) => (C_mfsm_D & C_mfsm_grant & C_mfsm_hold) => CMA3
  (C_mfsm_state = CMA3) => (C_mfsm_D) => CMA1
  (C_mfsm_state = CMA1) =>
    (C_mfsm_D & (C_mfsm_ss = ^SRDY)) => CMA0
    (C_mfsm_D & (C_mfsm_ss = ^SABORT)) => CMABT
  (C_mfsm_state = CMA0) =>
    (C_mfsm_D & (C_mfsm_ss = ^SRDY)) => CMA2
    (C_mfsm_D & (C_mfsm_ss = ^SABORT)) => CMABT
  (C_mfsm_state = CMA2) =>
    (C_mfsm_D & (C_mfsm_ss = ^SRDY)) => CMD1
    (C_mfsm_D & (C_mfsm_ss = ^SABORT)) => CMABT
  (C_mfsm_state = CMD1) =>
    (C_mfsm_D & (C_mfsm_ss = ^SRDY)) => CMD0
    (C_mfsm_D & (C_mfsm_ss = ^SABORT)) => CMABT
  (C_mfsm_state = CMD0) =>
    (C_mfsm_D & (C_mfsm_ss = ^SRDY) & C_mfsm_last) => CMD1
    (C_mfsm_D & (C_mfsm_ss = ^SRDY) & ~C_mfsm_last) => CMW
    (C_mfsm_D & (C_mfsm_ss = ^SABORT)) => CMABT
  (C_mfsm_state = CMW) =>
    (C_mfsm_D & (C_mfsm_ss = ^SABORT)) => CMABT
    (C_mfsm_D & (C_mfsm_ss = ^SACK) & C_mfsm_lock) => CMI
    (C_mfsm_D & (C_mfsm_ss = ^SRDY) & ~C_mfsm_lock & ~C_mfsm_cqkt) => CMA3
  (((C_mfsm_last) => CMI) & (CMABT)))

let new_C_mfsm_mabort = (new_C_mfsm_stateA = CMABT) in
let new_C_mfsm_midle = (new_C_mfsm_stateA = CMI) in
let new_C_mfsm_mrequest = (new_C_mfsm_stateA = CRM) in
let new_C_mfsm_ma3 = (new_C_mfsm_stateA = CMA3) in
let new_C_mfsm_ma2 = (new_C_mfsm_stateA = CMA2) in
let new_C_mfsm_ma1 = (new_C_mfsm_stateA = CMA1) in
let new_C_mfsm_ma0 = (new_C_mfsm_stateA = CMA0) in
let new_C_mfsm_md1 = (new_C_mfsm_stateA = CMD1) in
let new_C_mfsm_md0 = (new_C_mfsm_stateA = CMD0) in
let new_C_mfsm_iad_en_m = ((((new_C_mfsm_stateA = CMD1) & ~C_mfsm_write & C_mfsm_srty_en)) V
  ((new_C_mfsm_stateA = CMD0) & ~C_mfsm_write & C_mfsm_srty_en)) V
  ((new_C_mfsm_stateA = CMW) & (C_mfsm_state = CMD0) & ~C_mfsm_write & C_mfsm_srty_en))

let new_C_mfsm_mcout_sel1 = ((new_C_mfsm_stateA = CMA3) V (new_C_mfsm_stateA = CMA2)) in
let new_C_mfsm_mcout_sel0 = ((new_C_mfsm_stateA = CMA3) V (new_C_mfsm_stateA = CMA1)) V
  (new_C_mfsm_stateA = CMD1) in
let ms2 = (ALTER ARBN (2) (new_C_mfsm_stateA = CMA3) V (new_C_mfsm_stateA = CMA2) V
  (new_C_mfsm_stateA = CMA1) V (new_C_mfsm_stateA = CMD1) V (new_C_mfsm_stateA = CMD0)
(new_C_mfsm_stateA = CMW) V (new_C_mfsm_stateA = CMABT)) in
let msl = (ALTER ms2 (I)((new_C_mfsm_stateA = CMA1) V (new_C_mfsm_stateA = CMA0) V
(new_C_mfsm_stateA = CAM2) V (new_C_mfsm_stateA = CMD1) V
((new_C_mfsm_stateA = CMD0) A C_mfsm_last) V (new_C_mfsm_stateA = CMW) V
(new_C_mfsm_stateA = CMABT)) in
let ms0 = (ALTER msl (0) (((new_C_mfsm_stateA = CMD0) A C_mfsm_last) V
(new_C_mfsm_stateA = CMABT))) in
let aew_C_m fsm_ms = ms0 in
let new_C_mfsm_rq = -((new_C_mfsm_stateA = CMD0) V
(new_C_mfsm_stateA = CMABT)) in
let new_C_mfsm_cm = ((~(new_C_mfsm_stateA = CMA3) A C_mfsm_last) V
(new_C_mfsm_stateA = CMABT)) in
let new_C_mfsm_abort_le = ((new_C_mfsm_stateA = CMABT) V
(new_C_mfsm_stateA = CMA3)) in
let new_C_mfsm_mparity = ((new_C_mfsm_stateA = CMA3) V (new_C_mfsm_stateA = CMA0)
V (new_C_mfsm_stateA = CMD1) V (new_C_mfsm_stateA = CMD0)
V (C_mfsm_state = CMA1) V (C_mfsm_state = CMA0)
V (C_mfsm_state = CMA2) V (C_mfsm_state = CMD1)) in

let new_C_m fsm_stateA =
  ((C_sfsm_reset) => CSI I
   (C_sfsm_state = CSI) => ((C_sfsm_D A (C_sfsm_state = ^MSTART) A -C_sfsm_grant
       A C_sfsm_addressed) => CSA1 | CSI) I
   (C_sfsm_state = CSL) =>
     ((C_sfsm_D A (C_sfsm_state = ^MSTART) A -C_sfsm_grant A C_sfsm_addressed) => CSA1 I
     (C_sfsm_D A (C_sfsm_state = ^MSTART) A -C_sfsm_grant A -C_sfsm_addressed) => CSI I
     (C_sfsm_D A (C_sfsm_state = ^MABORT)) => CSABT | CSL) I
     (C_sfsm_state = CSA1) =>
       ((C_sfsm_D A (C_sfsm_state = ^MRDY)) => CSA0 I
       (C_sfsm_D A (C_sfsm_state = ^MABORT)) => CSABT | CSA0) I
       (C_sfsm_state = CS0) =>
         ((C_sfsm_D A (C_sfsm_state = ^MRDY) A -C_sfsm_hlda) => CSALI
         (C_sfsm_D A (C_sfsm_state = ^MRDY) A C_sfsm_hlda) => CSAW I
         (C_sfsm_D A (C_sfsm_state = ^MABORT)) => CSABT | CS0) I
         (C_sfsm_state = CSA0) =>
           ((C_sfsm_D A (C_sfsm_state = ^MRDY) A -C_sfsm_hlda) => CSAL I
           (C_sfsm_D A (C_sfsm_state = ^MABORT)) => CSABT | CSAO) I
           (C_sfsm_state = CSAL) =>
             ((C_sfsm_D A C_sfsm_write A (C_sfsm_state = ^MRDY)) => CS1 I
             (C_sfsm_D A -C_sfsm_write A (C_sfsm_state = ^MRDY)) => CSR I
             (C_sfsm_D A (C_sfsm_state = ^MABORT)) => CSABT | CSAL) I
             (C_sfsm_state = CSR) =>
               ((C_sfsm_D A -(C_sfsm_state = ^MABORT)) => CS1 I
               (C_sfsm_D A (C_sfsm_state = ^MABORT)) => CSABT | CSR) I
               (C_sfsm_state = CSD1) =>
                 ((C_sfsm_D A (C_sfsm_state = ^MRDY)) => CS0 I
                 (C_sfsm_D A (C_sfsm_state = ^MABORT)) => CSABT | CSD1) I
                 (C_sfsm_state = CSD0) =>
                   ((C_sfsm_D A (C_sfsm_state = ^MEND)) => CSACK I
                   (C_sfsm_D A (C_sfsm_state = ^MRDY)) => CS1 I
                   (C_sfsm_D A (C_sfsm_state = ^MABORT)) => CSABT | CSD0) I
                   (C_sfsm_state = CSACK) =>
                     ((C_sfsm_D A (C_sfsm_state = ^MRDY)) => CSL I
                     (C_sfsm_D A (C_sfsm_state = ^MWAIT)) => CSI I
                     (C_sfsm_D A (C_sfsm_state = ^MABORT)) => CSABT | CSACK) I

160
let ss2 = (ALTER ARBN (2) (-(new_C_sfsm_stateA = CSI)) \ (- (new_C_sfsm_stateA = CSABT)))) in
let ss1 = (ALTER ss2 (1) ((- (new_C_sfsm_stateA = CSI)) \ (- (new_C_sfsm_stateA = CSACK)))\n  \ (- (new_C_sfsm_stateA = CSABT)))) in
let ss0 = (ALTER ss1 (0) (new_C_sfsm_stateA = CSA0) \ (-C_sfsm_write) \ (- (new_C_sfsm_stateA = CSACK))) in
let new_C_sfsm_ss = ss0 in
let new_C_sfsm_dde = (new C sfsm_stateA = CSI) in
let new_C sfsm_slock = (new_C_sfsm_stateA = CSL) in
let new_C sfsm_sal = (new_C sfsm_stateA = CSAI) in
let new_C_sfsm_sa0 = (new_C sfsm_stateA = CSA0) in
let new_C_sfsm_sale = (new C sfsm_stateA = CSALE) in
let new_C sfsm_sdl = (new_C sfsm_stateA = CSDI) in
let new_C sfsm_sd0 = (new_C sfsm_stateA = CSDO) in
let new_C sfsm_sack = (new_C sfsm_stateA = CSACK) in
let new_C sfsm_sabort = (new C sfsm_stateA = CSABT) in
let new_C sfsm_s cout SelO = (new_C sfsm_stateA = CSDI) in
let new_C sfsm_sparity = ((- (new_C sfsm_stateA = CSI)) \ (- (new_C sfsm_stateA = CSACK))\n\n  \ (- (new_C sfsm_stateA = CSABT))) in
let new_C_efsm_stateA = ((C_efsm_rst) \ CEI) \ \n(C_sfsm_stateA = CEI) \ CEE \ CEI \ \n\n  \n(C_efsm_stateA = CEI) \ CEI \ \n(C_efsm_stateA = CEE \ CEI) \ \n\n  \n(C_efsm_stateA = CEI) \ CEI \ \n(C_efsm_stateA = CEE) \ CEI \ \n\n  \n(C_efsm_last \ -C_efsm_srdy) \ -C_efsm_male \ -C_efsm_rale \ CEI \ CEI) in
let new_C Efsm_srdy_en = (new_C sfsm_stateA = CEE) \ (C_sfsm_stateA = CEE)) in
let cout sel0 = (ALTER ARBN (0) ((new_C sfsm_sd1 V new_C sfsm_sd0) \ \nnew_C sfsm_s cout sel0 \ new_C mfsm_m cout sel0)) in
let cout sel10 = (ALTER cout sel0 (1) ((new_C sfsm_sd1 V new_C sfsm_sd0) \ \nnew_C mfsm_m cout sel1)) in
let c_busy = (-((SUBARRAY CB_rqt_in (3,1)) = (WORDN 7))) in
let c_grant = (((SUBARRAY Id (1,0)) = (WORDN 0)) \ - (ELEMENT CB_rqt_in (0))) \ \nV (((SUBARRAY Id (1,0)) = (WORDN 1)) \ - (ELEMENT CB_rqt_in (0)) \ (ELEMENT CB_rqt_in (1))) \ \nV (((SUBARRAY Id (1,0)) = (WORDN 2)) \ - (ELEMENT CB_rqt_in (0)) \ (ELEMENT CB_rqt_in (1)) \ \n (ELEMENT CB_rqt_in (2))) in
let c_write = ((new_C mfsm_cm_en) \ C_wr) \ (ELEMENT C_sizewrbe (5))) in
let new_C clkAA = C_clkA in
let new_C sidle_delA = C sidle del in
let new_C mrqt delA = C mrqt del in
let c_dfsm_srdy = (CB ss in = ^SRDY) in
let c_dfsm_master = (new_C mfsm ma3 V new_C mfsm ma2 V new_C mfsm ma1 V \new_C mfsm ma0 V new_C mfsm md1 V new_C mfsm md0) in
let c_dfsm_slave = (~new_C sfsm sidle \ ~new_C sfsm stock) in
let c_dfsm_cin_0_le = (ClkD \ ((new_C mfsm md0 \ c_dfsm_srdy \ ~c_write) \ \n(new_C sfsm ss0) \ (new_C sfsm sd0 \ c_write))) in
let c_dfsm_cin_1_le = (ClkD \ ((new_C mfsm md1 \ c_dfsm_srdy \ ~c_write) \ \nV

161
(new_C_sfsam_sal1) V (new_C_sfsam_sd1 & c_write)) in
let c_dfsm_cin_3_le = (ClkD & (new_C_sfsam_sidle V new_C_sfsam_slock)) in
let c_dfsm_cin_4_le = (new_C_clkAA & new_C_sfsam_sd0) in
let c_dfsm_cout_0_le = ((L_cale_) V (l_rdy_in_ & ~c_write)
  V (new_C_mfsm_mem0 & c_dfsm_rdy & c_write & ClkD))
  V (new_C_mfsm_mem0 & c_write & c_dfsm_rdy & ClkD)) in
let c_dfsm_cout_1_le = (new_C_clkAA & new_C_sfsam_sd1) in
let c_dfsm_cad_en = ~(new_C_mfsm_mem3) V (new_C_mfsm_mem1) V (new_C_mfsm_mem0)
  V (new_C_mfsm_mem2) V (c_write & (new_C_mfsm_mrdy1 V new_C_mfsm_mrdy0))
  V (~c_write & new_C_clkAA & new_C_mfsm_mem0) in
let new_C_last_inA_ = 1_last_in in
let new_C_holdA_ = ((ClkD) => C_hold | C_holdA_) in
let new_C_cout_0_le_delA = C_cout_0_le_del in
let new_C_cin_2_leA = C_cin_2_le in
let new_C_mrdy_delA_ = C_mrdy_del in
let new_C_iad_en_s_delA = ((ClkD) => C_iad_en_s_del | C_iad_en_s_delA) in
let new_C_wrdyA = C_wrdy in
let new_C_rrdyA = C_rrdy in
let new_C_iad_out = ((new_C_cin_2_leA) => C_data_in | C_iad_out) in
let new_C_a3a20 = (((c_dfsm_master & new_C_cout_0_le_delA) V (~c_dfsm_master & c_dfsm_cout_1_le)) => C_iad_in | C_a3a20) in
let new_C_mfsm_state = C_mfsm_state in
let new_C_mfsm_srdy_en = C_mfsm_srdy_en in
let new_C_mfsm_D = C_mfsm_D in
let new_C_mfsm_grant = C_mfsm_grant in
let new_C_mfsm_rst = C_mfsm_rst in
let new_C_mfsm_busy = C_mfsm_busy in
let new_C_mfsm_write = C_mfsm_write in
let new_C_mfsm_crqt_ = C_mfsm_crqt_ in
let new_C_mfsm_hold_ = C_mfsm_hold_ in
let new_C_mfsm_last_ = C_mfsm_last_ in
let new_C_mfsm_lock_ = C_mfsm_lock_ in
let new_C_mfsm_ss = C_mfsm_ss in
let new_C_mfsm_invalid = C_mfsm_invalid in
let new_C_sfsam_state = C_sfsam_state in
let new_C_sfsam_D = C_sfsam_D in
let new_C_sfsam_grant = C_sfsam_grant in
let new_C_sfsam_rst = C_sfsam_rst in
let new_C_sfsam_write = C_sfsam_write in
let new_C_sfsam_addressed = C_sfsam_addressed in
let new_C_sfsam_hlda_ = C_sfsam_hlda_ in
let new_C_sfsam_ms = C_sfsam_ms in
let new_C_efsm_state = C_efsm_state in
let new_C_efsm_cale_ = C_efsm_cale_ in
let new_C_efsm_last_ = C_efsm_last_ in
let new_C_efsm_male_ = C_efsm_male_ in
let new_C_efsm_rale_ = C_efsm_rale_ in

162
let new C_efsm_srdy_ = C_efsm_srdy_ in
let new C_efsm_rst = C_efsm_rst in
let new C_wr = C_wr in
let new C_sizewrbe = C_sizewrbe in
let new C_clkA = C_clkA in
let new C_sidle_del = C_sidle_del in
let new C_mrqtl_del = C_mrqtl_del in
let new C_last_in_ = C_last_in_ in
let new C_lock_in_ = C_lock_in_ in
let new C_ss = C_ss in
let new C_last_out_ = C_last_out_ in
let new C_hold_ = C_hold_ in
let new C_cout_0_le_del = C_cout_0_le_del in
let new C_cin_2_le = C_cin_2_le in
let new C_mrdy_del_ = C_mrdy_del_ in
let new C_iad_en_s_del = C_iad_en_s_del in
let new C_wrdy = C_wrdy in
let new C_rdy = C_rdy in
let new C_parity = C_parity in
let new C_source = C_source in
let new C_data_in = C_data_in in
let new C_iad_in = C_iad_in in
let new C_mfsm_cgnt_ in
let I_mrdy_out_ = ((~i_hlda_) => new C_mrdy_delA_ l ARB) in
let I_hold_ = new C_holdA_ in
let I_rale_out_ = ((~i_hlda_) => c_dfsm_i_rale_ l ARB) in
let I_male_out_ = ((~i_hlda_) => c_dfsm_i_male_ l ARB) in
let I_last_out_ = ((~i_hlda_) => new C_last_out_ l ARB) in
let I_srdy_out_ =
  ((~i_rale_ V new C_efsm_srdy_en) => ~(new C_wrdyA V new C_rdyA V new C_mfsm_mabort) l ARB) in
let I_be_out_ = ((~i_hlda_) => (SUBARRAY new C_sizewrbe (9,6)) l ARB) in
let I_ad_out_ =
  ((new C_iad_en_s_delA V new C_mfsm_iad_en_m V new C_sfm_iad_en_s) => new C_iad_out l ARB) in
let CB_rqOut_ = new C_mfsm_rqOut in
let cbms10 = (MALTER ARBN (1,0) (SUBARRAY new C_mfsm_ms (1,0))) in
let cbmx210 = (ALTER cbms10 (2) (ELEMENT new C_mfsm_ms (2)) V ~Pmm_failure V ~Pnu_invalid)) in
let CB_ms_out = ((~new C_mfsm_cm_en) => cbmx210 l ARBN) in
let cbssl0 = (MALTER ARBN (1,0) (SUBARRAY new C_sfsm_ss (1,0))) in
let CB_sss_out = ((~new C_sfsm_sss (2)) => cbssl0 l ARBN) in
let CB_iad_out = ((c_dfsm_iad_en) =>
  ((c_cout.sel = (WORDN 0)) => Par_Enc rep ((SUBARRAY new C_sia0 (15,0)))) in
  ((c_cout.sel = (WORDN 1)) => Par_Enc rep ((SUBARRAY new C_sia0 (31,16)))) in
  ((c_cout.sel = (WORDN 2)) => Par_Enc rep ((SUBARRAY new C_sia2 (15,0)))) in
  Par_Enc rep ((SUBARRAY new C_sia2 (31,16)))) in
let CB_parity = new C_parity in
let Disable_writes = (c_dfsm_slave A ~((ChannelID = (WORDN 0)) V (ELEMENT new C_mfsm_sabort) => cbss210 l ARBN)) in
let CB_par_out = new C_mfsm_parOut in
let C_ss_out = new C_ss in
let Disable_writes = (c_dfsm_slave A ~((ChannelID = (WORDN 0)) V (ELEMENT new C_mfsm_sabort) => cbss210 l ARBN)) in
let CB_parity = new C_parity in
Next-state definition for Phase-B instruction.

let PH_B_inst_def = new_definition
('PH_B_inst',
"! (rep:"rep_ty)
(C_mfsm_stateA C_mfsm_state :cmfsm_ty)
(C_sfsm_stateA C_sfsm_state :csfsm_ty)
(C_mfsm_ss C_sfsm_ss A C_a1a0 C_a3a2 C_mfsm_ms C_sizewrbe C_ss
C_source C_data_in C_iad_in :wordn)
(C_mfsm_mabort C_mfsm_midle C_mfsm_mrequest C_mfsm_ma3 C_mfsm_ma2 C_mfsm_mal
C_mfsm_rai0 C_mfsm_md0 C_mfsm_iad_en_m C_mfsm_mcout_sel1 C_mfsm_mcout_sel0
C_mfsm_rqts C_mfsm_cgot C_mfsm_cm_en C_mfsm_abort_le_en C_mfsm_mparity
C_mfsm_iad_en_s C_sfsm_sidle C_sfsm_slock C_sfsm_sal C_sfsm_sa0
C_sfsm_sale C_sfsm_sdk0 C_sfsm_sack C_sfsm_sabort C_sfsm_scout_sel0 C_sfsm_sparity
C_efsm_srdy_en
C_clkAA C_sidle_delA C_mrqt_delA C_last_inA C_holdA C_cout_0_le_delA
C_in_2_leA C_mrdy_delA C_iad_en_s_delA C_wrdyA C_rddyA
C_mfsm_srdy_en C_mfsm_D C_mfsm_grant C_mfsm_rst C_mfsm_busy C_mfsm_write
C_mfsm_cqnt C_mfsm_hold C_mfsm_last C_mfsm_lock C_mfsm_invalid
C_sfsm_D C_sfsm_grant C_sfsm_rst C_sfsm_write C_sfsm_addressed C_sfsm_blda
C_efsm_cale C_efsm_last C_efsm_male C_efsm_male C_ksfsm_rsdyc C_ksfsm_rst
C_mrty C_clkAA C_sidle_del C_mrqt_del C_last_in C_lock_in C_last_out
C_hold C_cout_0_le_del C_cin_2_le C_mrdy_del C_iad_en_s_del C_wrdy
C_parity :bool)

(Rst ClkA ClkB ClkD Pmm_failure Piu_invalid Reset_error :bool)

let PH_B_inst_rep
(C_mfsm_stateA, C_mfsm_mabort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
C_mfsm_ma1, C_mfsm_ma0, C_mfsm_md1, C_mfsm_md0, C_mfsm_iad_en_m, C_mfsm_mcout0
C_mfsm_mcout1, C_mfsm_ms, C_mfsm_rqts, C_mfsm_cgot, C_mfsm_cm_en, C_mfsm_abort_le_en,
C_mfsm_mparity, C_sfsm_stateA, C_sfsm_ss, C_sfsm_iad_en_s, C_sfsm_sidle, C_sfsm_slock,
C_sfsm_sa1, C_sfsm_sa0, C_sfsm_sale, C_sfsm_sdk0, C_sfsm_slock, C_sfsm_sack, C_sfsm_sabort,
C_sfsm_scout0, C_sfsm_sparity, C_efsm_stateA, C_efsm_srdy_en, C_clkAA, C_sidle_delA,
C_mrqt_delA, C_last_inA, C_ssA, C_holdA, C_cout_0_le_delA, C_cin_2_leA,
C_mrdy_delA, C_iad_en_s_delA, C_wrdyA, C_rddyA, C_iad_out, C_a1a0 C_a3a2 C_mfsm_stateA,
C_mfsm_srdy_en C_mfsm_D C_mfsm_grant C_mfsm_rst C_mfsm_busy C_mfsm_write C_mfsm_cqnt,
C_mfsm_hold C_mfsm_last C_mfsm_lock C_mfsm_ms C_mfsm_invalid C_sfsm_stateA C_sfsm_ss,
C_mfsm_grant C_sfsm_rst C_sfsm_write C_sfsm_addressed C_sfsm_blda C_sfsm_ms,
C_sfsm_stateA, C_sfsm_cale C_efsm_last C_efsm_male C_efsm_rale C_efsm_srdy
C_sfsm_rst C_mrty C_clkAA C_sidle_del C_mrqt_del C_last_in C_lock_in
C_ssidle C_last_out C_hold C_cout_0_le_del C_cin_2_le C_mrdy_del C_iad_en_s_del C_wrdy,
C_parity C_source C_data_in C_iad_in)

(Rst ClkA, ClkB, ClkD)

164
ClkD, Id, ChannelID, Pmm_failure, Piu_invalid, Ccr, Reset_error =

let new_C_wr = ((~Lcale_ => (ELEMENT I_ad_in (27)) \ C_wr) in
let new_C_sizewrbe = ((Rst) => ARBN !
  ((C_sfsm_sa0 \ C_clkAA) => (SUBARRAY C_data_in (31,22) \ C_sizewrbe)) in
let c_write = ((C_mfsm_cm_en) => new_C_wr (ELEMENT new_C_sizewrbe (5))) in
let cout_sel0 = (ALTER ARBN (0) ((C_sfsm_sd1 \ C_sfsm_sd0) =>
    C sfsm_s_cout_sel0 (C_mfsm_m_cout_sel0))) in
let cout_sel10 = (ALTER cout_sel0 (1) ((C_sfsm_sd1 \ C_sfsm_sd0) => F \ C_mfsm_m_cout_sel1)) in
let c_busy = (-(SUBARRAY CB_rqLin_in_ (3,1)) = (WORDN 7))) in
let c_grant = (((SUBARRAY Id (1,0)) = (WORDN 0)) \ (ELEMENT CB_rqLin_in_ (0))
  V (SUBARRAY Id (1,0)) = (WORDN 1) \ (ELEMENT CB_rqLin_in_ (0))
  \ (ELEMENT CB_rqLin_in_ (1))
  V (SUBARRAY Id (1,0)) = (WORDN 2) \ (ELEMENT CB_rqLin_in_ (0))
  \ (ELEMENT CB_rqLin_in_ (1))
  \ (ELEMENT CB_rqLin_in_ (2))
  V (SUBARRAY Id (1,0)) = (WORDN 3) \ (ELEMENT CB_rqLin_in_ (0))
  \ (ELEMENT CB_rqLin_in_ (1))
  \ (ELEMENT CB_rqLin_in_ (2))
  \ (ELEMENT CB_rqLin_in_ (3))) in
let c_dfsm_srdy = (CB_ss_in = ^SRDY) in
let c_dfsm_master = (C_mfsm_ma3 \ C_mfsm_ma1 \ C_mfsm_ma0 \ V C_mfsm_md1 \ V C_mfsm_md0) in
let c_dfsm_slave = (C_clkAA \ C_sfsm_sa0) in
let c_dfsm_out_0_le = ((I_cale_ \ (I_sdry_in_ \ C_clkAA) \ c_write)
    V (C_mfsm_ma0 \ c_dfsm_srdy \ c_write \ C_clkAA)
    V (C_mfsm_md0 \ c_write \ C_clkAA \ C_mfsm) in
let c_dfsm_out_1_le = ((C_clkAA \ C_sfsm_sd1) in
let c_dfsm_cad_en = (C_mfsm_md1 \ C_mfsm_md0) \ (c_write \ C_clkAA) in
let c_dfsm_i_mrdy_ = (-(~c_write \ CB_ms_in = ^MABORT)
  V (c_write \ C_clkAA \ c_write)) in
let c_dfsm_i_tale._ = (C_sfsm_sale \ ((SUBARRAY new_C_sizewrbe (1,0)) = (WORDN 3)) \ C_clkAA) in
let c_dfsm_i_male_ = (C_mfsm_mabort \ C_mfsm_real \ C_mfsm ma0 \ C_mfsm ma2 \ C_mfsm ma3 \ C_mfsm md1 \ C_mfsm md0) in
let c_dfsm_i_de_ = (C_mfsm_mrequest \ C_mfsm_mrequest) in
let c_dfsm_i_last_in_ = ((Rst) => F \ C_last_inA_ \ C_last_in_)) in
let c_dfsm_i_lock_in_ = (MEND) in
let c_dfsm_i_mabort = (CB_ms_in = ^MABORT) in
let c_dfsm_i_last_out_ = (C_sfsm sd1 \ C_sfsm_sd1) in
let c_dfsm_i_hold_ = (C_sfsm_sd1) in
let new_C_cout_0_le_del = c_dfsm_cout_0_le in
let new_C_cin_2_le = c_dfsm_cin_0_le in
let new_C_mrdy_del_ = c_dfsm_i_mrdy_ in
let new_C_iad_en_s_del = C_sfsm_iad_en_s in
let new_C_wrdy = (c_dfsm_srdy A c_write A C_mfsm_mdl A ClkD) in
let new_C_rrdy = (c..dfsm_srdy A -c_write A C..mfsm mdO A CIkD) in
let c_pe = (Par_Det rep CB_ad_in) in
let c_pe_cnt = (ClkD A ((-(C_mfsm_mparity = C_sfsm_sparsity)) V ((SUBARRAY CB_ss_in (1,0)) = (WORDN 0)))) in
let new_C_parity =
((ClkD A c_pe A c_pe_cnt) A I_cale_) => T |
((-ClkD A c_pe A c_pe cnt) A -I_cale_) => F |
((-ClkD A c_pe A c_pe_cnt) A I_cale_) => C_parity ! ARB)) in
let new_C_source = ((Rst) => (WORDN 0) |
((c_dfsm_cin_3_le) => Par_Dec rep (CB_ad_in) ! C_source)) in
let data_in31_16 = (MALTER ARBN (31,16) ((Rst) => (WORDN 0) |
((c_dfsm_cin_1_le) => Par_Dec rep (CB_ad_in) !
(SUBARRAY C_data_in (31,16)))))) in
let data_in31_0 = (MALTER data_in31_16 (15,0) ((Rst) => (WORDN 0) |
((c_dfsm_cin_0_le) => Par_Dec rep (CB_ad_in) !
(SUBARRAY C_data_in (15,0)))))) in
let new_C_data_in = data_in31_0 in
let new_C_iad_in = ((c_dfsm_cout_0_le) => I_ad_in ! C_iad_in) in
let new_C_mfsm_state = C_mfsm_stateA in
let new_C_mfsm_srty_en = C_efsm_srty_en in
let new_C_mfsm_D = ClkD in
let new_C_mfsm_grant = c_grant in
let new_C_mfsm_rst = Rst in
let new_C_mfsm_busy = c_busy in
let new_C_mfsm_write = c_write in
let new_C_mfsm_cro_ = I_cro_ in
let new_C_mfsm_hold_ = C_holdA_ in
let new_C_mfsm_last_ = new_C_last_in_ in
let new_C_mfsm_lock_ = new_C_lock_in_ in
let new_C_mfsm_ss = CB_ss_in in
let new_C_mfsm_invalid = Piu_invalid in
let new_C_sfsm_state = C_sfsm_state in
let new_C_sfsm_D = ClkD in
let new_C_sfsm_grant = c_grant in
let new_C_sfsm_rst = Rst in
let new_C_sfsm_write = c_write in
let new_C_sfsm_addressed = (Id = (SUBARRAY new_C_source (15,10)))) in
let new_C_sfsm_hlda_ = I_hlda_ in
let new_C_sfsm_ms = CB_ms_in in
let new_C_sfsm_state = C_sfsm_state in
let new_C_sfsm_cale_ = I_cale_ in
let new_C_sfsm_last_ = I_last_in_ in
let new_C_sfsm_male_ = I_male_in_ in
let new_C_sfsm_rale_ = I_rale_in_ in
let new_C_sfsm_srty_ = I_srty_in_ in
let new_C_sfsm_rst = Rst in
let new_C_mfsm_stateA = C_mfsm_stateA in
let new_C_mfsm_mabort = C_mfsm_mabort in
let new_C_mfsm_midle = C_mfsm_midle in
let new_C_mfsm_mrequest = C_mfsm_mrequest in
let new C mfsm_ma3 = C mfsm_ma3 in
let new C mfsm_ma2 = C mfsm_ma2 in
let new C mfsm_ma1 = C mfsm_ma1 in
let new C mfsm_ma0 = C mfsm_ma0 in
let new C mfsm_md1 = C mfsm_md1 in
let new C mfsm_md0 = C mfsm_md0 in
let new C mfsm_iad_en_m = C mfsm_iad_en_m in
let new C mfsm_mcout_sel1 = C mfsm_mcout_sel1 in
let new C mfsm_mcout_sel0 = C mfsm_mcout_sel0 in
let new C mfsm_ms = C mfsm_ms in
let new C mfsm_rqtn = C mfsm_rqtn in
let new C mfsm_cgnlt = C mfsm_cgnlt in
let new C mfsm_cm_en = C mfsm_cm_en in
let new C mfsm_abrtn_le_en = C mfsm_abrtn_le_en in
let new C mfsm_mparity = C mfsm_mparity in
let new C sfsm_stateA = C sfsm_stateA in
let new C sfsm_ss = C sfsm_ss in
let new C sfsm_iad_en_s = C sfsm_iad_en_s in
let new C sfsm_sidle = C sfsm_sidle in
let new C sfsm_slock = C sfsm_slock in
let new C sfsm_sal = C sfsm_sal in
let new C sfsm_sa0 = C sfsm_sa0 in
let new C sfsm_sale = C sfsm_sale in
let new C sfsm_sd1 = C sfsm_sd1 in
let new C sfsm_sd0 = C sfsm_sd0 in
let new C sfsm_sack = C sfsm_sack in
let new C sfsm_sabort = C sfsm_sabort in
let new C sfsm_s_cout_sel0 = C sfsm_s_cout_sel0 in
let new C sfsm_sparity = C sfsm_sparity in
let new C sfsm_stateA = C sfsm_stateA in
let new C sfsm_srdy_en = C sfsm_srdy_en in
let new C clkAA = C clkAA in
let new C sidle_delA = C sidle_delA in
let new C mrqtdelA = C mrqtdelA in
let new C last_inA = C last_inA in
let new C ssA = C ssA in
let new C holdA = C holdA in
let new C cout_0_le_delA = C cout_0_le_delA in
let new C cin_2_leA = C cin_2_leA in
let new C mrdy_delA = C mrdy_delA in
let new C iad_en_s_delA = C iad_en_s_delA in
let new C wrdyA = C wrdyA in
let new C rdyA = C rdyA in
let new C iad_out = C iad_out in
let new C ala0 = C ala0 in
let new C a3a2 = C a3a2 in

(let new C mfsm_stateA, new C mfsm_abrtn, new C mfsm_midle, new C mfsm_mrquest, new C mfsm_ma3,
new C mfsm_ma2, new C mfsm_ma1, new C mfsm_ma0, new C mfsm_md1, new C mfsm_md0,
new C mfsm_iad_en_m,
new C mfsm_mcout_sel1, new C mfsm_mcout_sel0, new C mfsm_ms, new C mfsm_rqtn,
new C mfsm_cm_en, new C mfsm_abrtn_le_en, new C mfsm_mparity, new C sfsm_stateA, new C sfsm_ss,
new C sfsm_iad_en_s, new C sfsm_sidle, new C sfsm_slock, new C sfsm_sal, new C sfsm_sa0,
new C sfsm_sale, new C sfsm_sd1, new C sfsm_sd0, new C sfsm_sack, new C sfsm_sabort,

167
new_C_sfsm_s_cout_sel0, new_C_sfsm_sparity, new_C_efsm_stateA, new_C_efsm_srdy_en, new_C_clkAA,
new_C_sidle_delA, new_C_mrqt_delA, new_C_last_inA, new_C_ssa, new_C_holdA,
new_C_cout_0_le_delA, new_C_cin_2_leA, new_C_mrdy_delA, new_C_iad_en_s_delA, new_C_wrdyA, new_C_rdyA,
new_C_iad_out, new_C_a1a0, new_C_a3a2, new_C_mfsm_state, new_C_mfsm_srdy_en, new_C_mfsm_D,
new_C_mfsm_grant, new_C_mfsm_rst, new_C_mfsm_busy, new_C_mfsm_write, new_C_mfsm_cqnt,
new_C_mfsm_hold, new_C_mfsm_last, new_C_mfsm_lock, new_C_mfsm_ss, new_C_mfsm_invalid,
new_C_sfsm_state, new_C_sfsm_D, new_C_sfsm_grant, new_C_sfsm_rst, new_C_sfsm_write,
new_C_sfsm_addressed, new_C_sfsm_hlda, new_C_sfsm_ms, new_C_sfsm_state, new_C_sfsm_cale,
new_C_sfsm_last, new_C_sfsm_male, new_C_sfsm_rale, new_C_sfsm_srdy, new_C_sfsm_rst, new_C_wr,
new_C_sizewrbe, new_C_clkA, new_C_cidle_del, new_C_mrqt_del, new_C_last_in, new_C_lock_in,
new_C_ss, new_C_last_out, new_C_hold, new_C_cout_0_le_del, new_C_cin_2_le, new_C_mrdy_del,
new_C_iad_en_s_del, new_C_wrdy, new_C_rdy, new_C_parity, new_C_source, new_C_data_in, new_C_iad_in")
);

% Output definition for Phase-B instruction.

let PH_B_out_def = new_definition
('PH_B_out',
"!(rep:*rep_ty)
(C_mfsm_stateA C_mfsm_state :cmfsm_ty)
(C_sfsm_stateA C_sfsm_state :csfsm_ty)
(C_efsm_stateA C_efsm_state :cefsm_ty)
(C_mfsm_ms C_sfsm_ss C_ssA C_iad_out C_a1a0 C_a3a2 C_sfsm_ss C_mfsm_ms C_sizewrbe C_ss
C_source C_data_in C_iad_in :worda)
(C_mfsm_mabort C_mfsm_midle C_mfsm_mrequest C_mfsm_ma3 C_mfsm_ma2 C_mfsm_ma1
C_mfsm_meta C_mfsm_mdt C_mfsm_mrd C_mfsm_iad_en_m C_mfsm_mcout_selt C_mfsm_mcout.sel0
C_mfsm_rqt C_mfsm_cqnt C_mfsm_cm_en C_mfsm_abort_le_en C_mfsm_mparity
C_sfsm_iad_en_s C_sfsm_sidle C_sfsm_slock C_sfsm_sal C_sfsm_sal0
C_sfsm_sdata C_sfsm_sd1 C_sfsm_sd0 C_sfsm_sack C_sfsm_sabort C_sfsm_scout.sel0 C_sfsm_sparsity
C_sfsm_srdy_en C_clkAA C_sidle_delA C_mrqt_delA C_last_inA C_holdA C_cout_0_le_delA
C_cin_2_leA C_mrdy_delA C_iad_en_s_delA C_wr C_mrdy_del C_mrdy_del C_last_in C_lock_in C_last_out
C_hold C_cout_0_le C_cin_2_le C_mrdy_del C_iad_en_s_del C_wrdy
C_rdy C_parity :bool)
(Rt C_mfsm_rqt C_mfsm_busy C_mfsm_write C_mfsm_lock C_mfsm_invalid
C_mfsm_cqnt C_mfsm_grant C_mfsm_rst C_mfsm_addressed C_sfsm_hlda
C_mfsm_cale C_mfsm_last C_mfsm_male C_mfsm_rale C_mfsm_srdy C_mfsm_rst
C_wr C_clkA C_sidle_del C_mrqt_del C_last_in C_lock_in C_last_out
C_hold C_cout_0_le C_cin_2_le C_mrdy_del C_iad_en_s_del C_wrdy
C_rdy C_parity :bool))

PH_B_out rep
(C_mfsm_stateA, C_mfsm_mabort, C_mfsm_midle, C_mfsm_mrequest, C_mfsm_ma3, C_mfsm_ma2,
C_mfsm_meta, C_mfsm_mdt, C_mfsm_mrd, C_mfsm_iad_en_m, C_mfsm_mcout_selt,
C_mfsm_mcout.sel0, C_mfsm_ms, C_mfsm_rqnt, C_mfsm_cm_en, C_mfsm_abort_le_en,
C_mfsm_mparity, C_sfsm_stateA, C_sfsm_sdata, C_sfsm_sidle, C_sfsm_slock,
C_sfsm_sdata C_sfsm_sd1 C_sfsm_sd0 C_sfsm_sack C_sfsm_sabort,
C_sfsm_scout.sel0 C_sfsm_sparsity, C_sfsm_stateA, C_sfsm_srdy_en, C_clkAA, C_sidle_delA,
C_mrqt_delA, C_last_inA, C_ssA, C_holdA, C_cout_0_le_delA, C_cin_2_leA,

168
let new_C_wr = (~(C_sfdm_sd1 V C_sfdm_sd0)) in
let new_C_sizewrbe = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_write = (C_sfdm_sd1 V C_sfdm_sd0) in
let cout_sel0 = (C_sfdm_sd1 V C_sfdm_sd0) in
let cout_sel10 = (C_sfdm_sd1 V C_sfdm_sd0) in
let cout_sel = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_busy = (~((C_sfdm_sd1 V C_sfdm_sd0) V ~c_write)) in
let c_grant = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_srdy = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_master = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_slave = (~((C_sfdm_sd1 V C_sfdm_sd0) V C_sfdm_sd0) in
let c_dfsm_cin_0_le = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_cin_1_le = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_cin_3_le = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_cin_4_le = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_cin_5_le = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_cout_0_le = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_cout_1_le = (C_sfdm_sd1 V C_sfdm_sd0) in
let c_dfsm_cad_en = ~((C_sfdm_sd1 V C_sfdm_sd0) V (C_sfdm_sd1 V C_sfdm_sd0) V (C_sfdm_sd1 V C_sfdm_sd0)) in
let c_dfsm_i_male = ~((C_sfdm_sd1 V C_sfdm_sd0) V (C_sfdm_sd1 V C_sfdm_sd0)) in
let c_dfsm_i_rale = ~((C_sfdm_sd1 V C_sfdm_sd0) V (C_sfdm_sd1 V C_sfdm_sd0)) in
let c_dfsm_i_mrdy = ~((C_sfdm_sd1 V C_sfdm_sd0) V (C_sfdm_sd1 V C_sfdm_sd0)) in
let new_C_clkD = (C_sfdm_sd1 V C_sfdm_sd0) in
let new_C_sidle_del = (C_sfdm_sd1 V C_sfdm_sd0) in
let new_C_mrq_del = (C_sfdm_sd1 V C_sfdm_sd0) in

169
let new_C_last_in_ = ((Rst) => F !
  ((C_mfsm_mabort V C_mfsm_end1 & ClkD) => C_last_inA_ | C_last_in)) in
let new_C_lock_in_ = ((Rst) => F ! ((C_mfsm_mal) => l_lock_ | C_lock_in_)) in
let new_C_ss = ((C_mfsm_abort_le_en_) => C_ssA | C_ss) in
let mend = (CB_mal_in = ^MBND) in
let mabort = (CB_mal_in = ^MABORT) in
let new_C_lasLout.. = ((C_lfsm_sal A CIkD A (mand V mabort))) => T ! ((--C_lfsm_sal A CIkD A (mand V mabort))) => F ! ((.--C_lfsm..sal A CIkD A (mand V mabort))) => C_lasLout._ I ARB)) in
let new_C_hold_ = C_sfr, m sidle in
let new_C_couLO..le_del = c_dfsm_cout..0_le in
let new_C_cin_2._le = c_dfsm_cin_0_le in
let new_C_mrdy_del_ = c_dfsm_lmdy_ in
let new_C_iad_an_Ldel = C_sfsm_iad_en_s in
let new_C_mrdy -- (c_dfsm_srdy A c_write A C_mfsm_rdy0 A ClkD) in
let c_pe -- (Par Det rep CB_ms_in) in
let c_pe_cnt = (ClkD A c_pe A c_pe_cnt | _cale_) => T ! ((--(ClkD A c_pe A c_pe_cnt | _cale_)) => F ! ((--(ClkD A c_pe A c_pe_cnt | _cale_)) => C_pe'rity | ARB)) in
let new_C_source = ((Rst) => (WORDN 0) | ((c_dfsm_cin_3_le | Par Dec rep (CB_ad_in | C_source)) in
let data_in31_16 = (MALTER ARBN (31,16) ((Rst) => (WORDN 0) | (c_dfsm_cin_1_le | Par Dec rep (CB_ad_in | (SUBARRAY C_data_in (31,16)))))) in
let data_in31_0 = (MALTER data_in31_16 (15,0) ((Rst) => (WORDN 0) | (c_dfsm_cin_0_le | Par Dec rep (CB_ad_in | (SUBARRAY C_data_in (15,0)))))) in
let new_C_data_in = data_in31_0 in
let new_C_iad_in = ((c_dfsm_cout_0_le | l_ad_in | C_iad_in) in
let new_C_mfsm_state = C_mfsm_stateA in
let new_C_mfsm_srdy_en = C_mfsm_srdy_en in
let new_C_mfsm_D = ClkD in
let new_C_mfsm_grant = c_grant in
let new_C_mfsm_rst = Rst in
let new_C_mfsm_busy = c_busy in
let new_C_mfsm_write = c_write in
let new_C_mfsm_crt_ = l_crt_ in
let new_C_mfsm_hold_ = C_holdA_ in
let new_C_mfsm_last_ = new_C_last_in_ in
let new_C_mfsm_lock_ = new_C_lock_in_ in
let new_C_mfsm_ss = CB_ss_in in
let new_C_mfsm_invalid = Piu_invalid in
let new_C_sfsm_state = C_sfsm_state in
let new_C_sfsm_D = ClkD in
let new_C_sfsm_grant = c_grant in
let new_C_sfsm_rst = Rst in
let new_C_sfsm_write = c_write in
let new_C_sfsm_addressed = ((id = (SUBARRAY new_C_source (15,10)))) in
let new_C_sfsm_hlda_ = l_hlda_in
let new_C_sfsm_ms = CB_ms_in in
let new_C.efsm_state = C.efsm_state in
let new_C.efsm_cale_ = I.cale_ in
let new_C.efsm_last_ = I.last_in_ in
let new_C.efsm_male_ = I.male_in_ in
let new_C.efsm_rale_ = I.rale_in_ in
let new_C.efsm_srty_ = I.srty_in_ in
let new_C.efsm_rs: = Rst in
let new_C.mfsm_stateA = C.mfsm_stateA in
let new_C.mfsm_mabort = C.mfsm_mabort in
let new_C.mfsm_midle = C.mfsm_midle in
let new_C.mfsm_mrequest = C.mfsm_mrequest in
let new_C.mfsm_ma3 = C.mfsm_ma3 in
let new_C.mfsm_ma2 = C.mfsm_ma2 in
let new_C.mfsm_ma1 = C.mfsm_ma1 in
let new_C.mfsm_ma0 = C.mfsm_ma0 in
let new_C.mfsm_md1 = C.mfsm_md1 in
let new_C.mfsm_md0 = C.mfsm_md0 in
let new_C.mfsm_iad_en_m = C.mfsm_iad_en_m in
let new_C.mfsm_mcout.sel1 = C.mfsm_mcout.sel1 in
let new_C.mfsm_mcout.sel0 = C.mfsm_mcout.sel0 in
let new_C.mfsm_ms = C.mfsm_ms in
let new_C.mfsm_rqU = C.mfsm_rqU in
let new_C.mfsm_cm_en = C.mfsm_cm_en in
let new_C.mfsm_abort_le_en = C.mfsm_abort_le_en in
let new_C.mfsm_mparity = C.mfsm_mparity in
let new_C.sfsf_stateA = C.sfsf_stateA in
let new_C.sfsf_ss = C.sfsf_ss in
let new_C.sfsf_iad_en_s = C.sfsf_iad_en_s in
let new_C.sfsf_sidle = C.sfsf_sidle in
let new_C.sfsf_slock = C.sfsf_slock in
let new_C.sfsf_sal = C.sfsf_sal in
let new_C.sfsf_sa0 = C.sfsf_sa0 in
let new_C.sfsf_sale = C.sfsf_sale in
let new_C.sfsf_sd1 = C.sfsf_sd1 in
let new_C.sfsf_sd0 = C.sfsf_sd0 in
let new_C.sfsf_sack = C.sfsf_sack in
let new_C.sfsf_sabort = C.sfsf_sabort in
let new_C.sfsf_scout.sel0 = C.sfsf_scout.sel0 in
let new_C.sfsf_sparsity = C.sfsf_sparsity in
let new_C.sfsf_stateA = C.sfsf_stateA in
let new_C.sfsf_srty_en = C.sfsf_srty_en in
let new_C.clkAA = C.clkAA in
let new_C.sidle_delA = C.sidle_delA in
let new_C.mrqU_delA = C.mrqU_delA in
let new_C.last_inA_ = C.last_inA_ in
let new_C.ssA = C.ssA in
let new_C.holdA_ = C.holdA_ in
let new_C.cout_0_le_delA = C.cout_0_le_delA in
let new_C.cin_2_leA = C.cin_2_leA in
let new_C.mrdy_delA_ = C.mrdy_delA_ in
let new_C.iad_en_s_delA = C.iad_en_s_delA in
let new_C.wrdyA = C.wrdyA in
let new_C.rdyA = C.rdyA in
let new_C_iad_out = C_iad_out in
let new_C_ala0 = C_ala0 in
let new_C_a3a2 = C_a3a2 in

let Lcgnt__ = new_C_mfsm_cgnt_ in
let Lmrdy_out_ = ((~L_hlda_) => new_C_mrdy_delA_ ! ARB) in
let L_bold_ = new_C_boldA_ in
let L_rale_out_ = ((~L_hlda_) => c_dfsm_i_rale_ ! ARB) in
let L_male_out_ = ((~L_hlda_) => c_dfsm_i_male_ ! ARB) in
let L_last_out_ = ((~L_hlda_) => new_C_last_out_ ! ARB) in
let L_srdy_out_ =
  ((~L_cale_ V new_C_efsrm_srdy_en) => -(new_C_wrdyA V new_C_rddyA V new_C_mfsm_mabort) ! ARB) in
let L_be_out_ = ((~L_hlda_) => (SUBARRAY new_C_sizewrbe (9,6)) ! ARB) in
let L_ad_out_ =
  ((new_C_iad_en_s_delA V new_C_efsrm_iad_en_m V new_C_sfsrm_iad_en_s) => new_C_iad_out ! ARB) in
let CB_rqt_out_ = new_C_mfsm_rqt_ in

let cbms10 = (MALTER ARBN (1,0) (SUBARRAY new_C_mfsm_ms (1,0))) in
let cbms210 = (ALTER cbms10 (2) ((ELEMENT new_C_mfsm_ms (2)) ∧ ¬Pmm_failure ∧ ¬Piu_invalid)) in
let CB_ms_out_ = ((~new_C_mfsm_cm_en) => cbms210 ! ARBN) in
let cbss10 = (MALTER ARBN (1,0) (SUBARRAY new_C_sfsrm_ss (1,0))) in
let cbss210 = (ALTER cbss10 (2) ((ELEMENT new_C_sfsrm_ss (2)) ∧ ¬Pmm_failure ∧ ¬Piu_invalid)) in
let CB_ss_out_ = ((~new_C_sfsrm_sidle ∧ ¬new_C_sfsrm_sabort) => cbss210 ! ARBN) in
let CB_ad_out_ = (c_dfsm_cad_en =>
  ((c_cout_sel = (WORDN 0)) => Par_Enc rep ((SUBARRAY new_C_a1a0 (15,0)))
  ((c_cout_sel = (WORDN 1)) => Par_Enc rep ((SUBARRAY new_C_a1a0 (31,16)))
  ((c_cout_sel = (WORDN 2)) => Par_Enc rep ((SUBARRAY new_C_a3a2 (15,0)))
  ARBN
in
let CB_parity = new_C_parity in

let Disable_writes = (c_dfsm_slave ∧
  ((ChannelID = (WORDN 0)) ∧ (ELEMENT new_C_source (6)))
  ∧ ¬((ChannelID = (WORDN 1)) ∧ (ELEMENT new_C_source (7)))
  ∧ ¬((ChannelID = (WORDN 2)) ∧ (ELEMENT new_C_source (8)))
  ∧ ¬((ChannelID = (WORDN 3)) ∧ (ELEMENT new_C_source (9)))) in

let CB_parity = new_C_parity in

(L_cgnL, L_mrdy_out_, L_bold_, L_rale_out_, L_male_out_, L_last_out_, L_srdy_out_, L_ad_out_, L_be_out_,
CB_rqt_out_, CB_ms_out, CB_ss_out, CB_ad_out, CB_ss_out, Disable_writes, CB_parity)"

);;

close_theory();
C.5 SU_Cont Specification

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

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

```ml
set_search_path (search_pathO @ ['/home/titan3/dfura/ftep/piu/holflib/']);

system 'rm s_block.th';

new_theory 's_block';

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

let s__state_ty = ":(sfsm-ty#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#b___#
bool#bool#wordn#wordn#bool#bool#
s fsm_ty#bool#bool#
bool#bool#bool#bool#bool#bool#bool#bool");

let s_state = "((S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_srcp, S__fsm._sdi, S_fsm_srp, S_fsm_src0, S_fsm_srcl,
S_fsm_spf, S_fsm_sc0f, S_fsm_sc1f, S_fsm_spmf, S_fsm_sh, S_fsm_sc, S_fsm_scu, S_fsm_srs,
S_fsm_scs, S_soft_shot, S_soft_shot_delA, S_soft_cntA, S_delayA, S_instart, S_cpu_histA,
S_fsm_state, S_fsm_rst, S._fsm delay6, S_fsm._delayl7, S_fsm_bothbad, S fsm_bypass,
S soft shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1,
S_pnn_fail, S_cpu0_fail, S_cpu1_fail, S_cpu_hist, S_piu_fail)
:^s_state_ty)";;

let s_env_ty = ":(bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool
bool#bool#bool#bool#bool#bool)";;

let s_env = "((ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0, Failure1) :
:^s_env_ty)";;

let s_out_ty = ":(words#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool
bool#bool#bool#bool#bool#bool)";;

let s_out = "((S_state, Reset_cport, Disable_int, Reset_pi0, Reset_cpu0, Reset_cpu1, Cpu_hist,
Piu_fail, Cpu0_fail, Cpu1_fail, Pnn_fail)
:^s_out_ty)";;

Next-state definition for Phase-A instruction.

let PH_A_inst_def = new_definition
('PH_A_inst')
```

173
let new_S_fsm_stateA =
(S_fsm_rst) => SSTART
((S_fsm_state = SSTART) => SRA)
((S_fsm_state = SRA) => ((S_fsm_delay6) => ((S_fsm_bypass) => SO | SPF | SRA))
((S_fsm_state = SPF) => SCO)
((S_fsm_state = SCO) => ((S_fsm_delay17) => SCOF | SCO))
((S_fsm_state = SCO) => ST)
((S_fsm_state = ST) => SCII)
((S_fsm_state = SCII) => ((S_fsm_delay17) => SCIF | SCII))
((S_fsm_state = SCIF) => SS)
((S_fsm_state = SS) => ((S_fsm_bothbad) => SSTOP | SCS))
((S_fsm_state = SSTOP) => SSTOP)
((S_fsm_state = SCS) => ((S_fsm_delay6) => SN | SCS))
((S_fsm_state = SN) => ((S_fsm_delay17) => SO | SN))
((S_fsm_state = SO) => SO | SSTALL))
in
let new_S_fsm_so = (new_S_fsm_stateA = SN) in
let new_S_fsm_sn = (new_S_fsm_stateA = SO) in
let new_S_fsm_srcp = (((new_S_fsm_stateA = SO)) \ (~((S_fsm_state = SSTOP))) \ V (S_fsm_state = SRA)) in
let new_S_fsm_sdi = (((new_S_fsm_stateA = SO)) \ (~((S_fsm_state = SSTOP))) \ V (S_fsm_state = SRA)) in
let new_S_fsm_src = ((new_S_fsm_stateA = SSTART) \ V (new_S_fsm_stateA = SRA)) \ V (new_S_fsm_stateA = ST)
V (new_S_fsm_stateA = SCIF) \ V (new_S_fsm_stateA = SS)
V (new_S_fsm_stateA = SCS)) in
let new_S_fsm_src0 = ((new_S_fsm_stateA = SPF)) \ (~((new_S_fsm_stateA = SC0))) in
let new_S_fsm_sc0f = ((new_S_fsm_stateA = ST)) \ (~((new_S_fsm_stateA = SCII))) in
let new_S_fsm_sc1f = (new_S_fsm_stateA = SC1f) in
let new_S_fsm_srcp = (new_S_fsm_stateA = SO) in
let new_S_fsm_src0 = (new_S_fsm_stateA = SSTART) in
let new_S_fsm_src = ((new_S_fsm_stateA = SSTART) \ V ((S_fsm_state = SRA) \ S_fsm_delay6)
V (new_S_fsm_stateA = SCOF) \ V (new_S_fsm_stateA = ST)
V (new_S_fsm_stateA = SCIF) \ V (new_S_fsm_stateA = SS)
V ((S_fsm_state = SCS) \ S_fsm_delay6)) in
let new_S_fsm_src = (((new_S_fsm_stateA = SSTOP)) \ (~((new_S_fsm_stateA = SO))) \ V (S_fsm_state = SN)) in
let new_S_fsm_src = (((S_fsm_state = SPF) \ ~S_fsm_rst) \ V ((S_fsm_state = ST) \ ~S_fsm_rst)) in
let new_S_fsm_src = ((new_S_fsm_stateA = SCS)) in
let new_S_soft_shot = (~Gcrh \ Gcrh) in
let new_S_soft_shot_delA = S_soft_shot_del in
let new_S_soft_cntA = ((new_S_fsm_srs) => (WORDN 0) \ S_soft_cnt) in
let s_delay_out = ((S_fsm_sec) => (INCN 17 S_delayA) \ S_delayA) in
let new_S_delayA = ((new_S_fsm_src V (new_S_fsm_scs (ELEMENT s_delay_out (6)))) => (WORDN 0) \ S_delay) in
let s_delayout = ((new_S_fsm_sec) => (INCN 17 S_delayA) \ S_delayA) in
let new_S_instart = ((Test) => (ELEMENT s_delayout (5)) \ (ELEMENT s_delayout (16))) in
let new_S_cpu_histA = (S_resets_cpu0 \ S_resets_cpu1 \ Bypass) in
let new_S_fsm_state = S_fsm_state in
let new_S_fsm_rst = S_fsm_rst in
let new_S_fsm_delay6 = S_fsm_delay6 in
let new_S_fsm_delay17 = S_fsm_delay17 in
let new_S_fsm_bothbad = S_fsm_bothbad in
let new_S_fsm_bypass = S_fsm_bypass in
let new_S_soft_shot_del = S_soft_shot_del in
let new_S_soft_cnt = S_soft_cnt in
let new_S_delay = S_delay in
let new_S_bad_cpu0 = S_bad_cpu0 in
let new_S_bad_cpu1 = S_bad_cpu1 in
let new_S_resets_cpu0 = S_resets_cpu0 in
let new_S_resets_cpu1 = S_resets_cpu1 in
let new_S_pmm_fail = S_pmm_fail in
let new_S_cpu_hist = S_cpu_hist in
let new_S_piur_fail = S_piur_fail in

Output definition for Phase-A instruction.
(ClkA ClkB Rst Bypass Test Gcrh Gcrl Failure0_ Failure1_ :bool).

PH_A_out (S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_srcp, S_fsm_sdi, S_fsm_src, S_fsm Src0, S_fsm_src1, S_fsm_spf, S_fsm_scof, S_fsm_scll, S_fsm_spmf, S_fsm sb, S_fsm Src, S_fsm sec, S_fsm srs, S_fsm_scs, S_soft_shot, S_soft_shot_delA, S_soft_cntA, S_delayA, S_instance, S_cpu_histA, S_fsm_state, S_fsm_rst, S_fsm delay6, S_fsm delay17, S_fsm bothbad, S_fsm_bypass, S_soft_shot del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1, S_pmm fail, S_cpu0 fail, S_cpu1 fail, S_cpu hist, S_piu fail)

(ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0, Failure1_ _) =

let new_S_fsm_stateA =

((S fsm_rst) => SSTART I
 (S fsm_state = SSTART) => SRA I
 (S fsm_state = SRA) => ((S fsm delay6) => ((S fsm_bypass) => SPF) I SRA) I
 (S fsm_state = SPF) => SCOF I
 (S fsm_state = SCOF) => ST I
 (S fsm_state = ST) => SCII I
 (S fsm_state = SCII) => ((S fsm_delay17) => SCIF I SCII) I
 (S fsm_state = SCIF) => SS I
 (S fsm_state = SS) => ((S fsm bothbad) => SSTOP I SCS) I
 (S fsm state = SSTOP) => SSTOP I
 (S fsm state = SCS) => ((S fsm delay6) => SN I SCS) I
 (S fsm state = SN) => ((S fsm delay17) => SO I SN) I
 (S fsm state = SO) => SO I (S ill))))))))) in

let new_S_fsm_sn = (new_S fsm stateA = SN) in

let new_S_fsm_so = (new_S fsm stateA = SO) in

let new_S_fsm_srcp = (S fsm stateA = SPF) I S fsm_delay6 I S fsm_rst)

let new_S_fsm_spmf = (S fsm_stateA = SCS) in

let new_S_fgm_spf = ((S fsm state = SRA) I S fsm_delay6 I S fsm_rst)

let new_S_fsm Src = ((new_S fsm stateA = SSTART) V ((S fsm state = SRA) A S fsm_delay6) V (new_S fsm stateA = SCS)) in

let new_S_fsm_src = (((S fsm_state = SPF) ^ ~S fsm_rst) V ((S fsm_state = ST) ^ ~S fsm_rst)) in

let new_S_fsm_scs = (new_S fsm stateA = SCS) in

let new_S_soft_shot = (Gcrh Gcrl) in

let new_S_soft_shot_delA = S soft_shot_del in

let new_S_soft_cntA = (new_S fsm_srcs) => (WORDN 0) I S soft_cnt in

let s_delay_out = (S fsm sec) => (INCN 17 S delayA) I S delayA in

let s_delay_out = (new_S fsm sec) => (INCN 17 new_S delayA) I new_S delayA) in

let new_S_instance = (Test) => (ELEMENT s_delay out (5)) I (ELEMENT s_delay out (16)) in

let s_soft_cnt out = ((S soft_shot A ~new_S soft_shot_delA) =>

176
(INCN 2 new_S_soft_cntA) \ new_S_soft_cntA) in
let s_cpu0_ok = (new_S_fsm_sc0f \ Failure0 \ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu1_ok = (new_S_fsm_sc1f \ Failure1 \ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu0_select = ((new_S_fsm_sn V new_S_fsm_so) \ s_cpu0_fail) in
let s_cpu1_select = ((new_S_fsm_sn V new_S_fsm_so) \ s_cpu1_fail) in
let new_S_cpu_histA = (S_reset_cpu0 \ S_reset_cpu1 \ Bypass) in
let new_S_fsm_state = S_fsm_state in
let new_S_fsm_rst = S_fsm_rst in
let new_S_fsm_delay6 = S_fsm_delay6 in
let new_S_fsm_delay17 = S_fsm_delay17 in
let new_S_fsm_bothbad = S_fsm_bothbad in
let new_S_fsm_bypass = S_fsm_bypass in
let new_S_soft_shot_del = S_soft_shot_del in
let new_S_soft_cnt = S_soft_cnt in
let new_S_delay = S_delay in
let new_S_bad_cpu0 = S_bad_cpu0 in
let new_S_bad_cpu1 = S_bad_cpu1 in
let new_S_reset_cpu0 = S_reset_cpu0 in
let new_S_reset_cpu1 = S_reset_cpu1 in
let new_S_pmm_fail = S_pmm_fail in
let new_S_cpu0_fail = S_cpu0_fail in
let new_S_cpu1_fail = S_cpu1_fail in
let new_S_cpu_hist = S_cpu_hist in
let new_S_piu_fail = S_piu_fail in
let ss0 = (ALTER ARBN (0) ((new_S_fsm_stateA = SS) V (new_S_fsm_stateA = SSTOP)
V (new_S_fsm_stateA = SCS) V (new_S_fsm_stateA = SN)
V (new_S_fsm_stateA = SO))) in
let ss1 = (ALTER ss0 (1) ((new_S_fsm_stateA = SCOF) V (new_S_fsm_stateA = ST)
V (new_S_fsm_stateA = SCIF) V (new_S_fsm_stateA = SCII) V (new_S_fsm_stateA = SSTOP)
V (new_S_fsm_stateA = SCS))) in
let ss2 = (ALTER ss1 (2) ((new_S_fsm_stateA = SPF) V (new_S_fsm_stateA = SCOI)
V (new_S_fsm_stateA = SCOF) V (new_S_fsm_stateA = SCIF)
V (new_S_fsm_stateA = SSTOP) V (new_S_fsm_stateA = SO))) in
let ss3 = (ALTER ss2 (3) ((new_S_fsm_stateA = SRA) V (new_S_fsm_stateA = SPF)
V (new_S_fsm_stateA = ST) V (new_S_fsm_stateA = SCII)
V (new_S_fsm_stateA = SCS) V (new_S_fsm_stateA = SN)
V (new_S_fsm_stateA = SO))) in
let S_state = ss3 in
let Reset_cport = new_S_fsm_srcp in
let Disable_int = (~new_S_instart \ (new_S_fsm_sn \ (ELEMENT s_delay_out (6))) \ new_S_fsm_sdi) in
let Reset_piu = new_S_fsm_srp in
let Reset_cpu0 = new_S_reset_cpu0 in
let Reset_cpu1 = new_S_reset_cpu1 in
let Cpu_hist = new_S_cpu_hist in
let Piu_fail = new_S_piu_fail in
let Cpu0_fail = new_S_cpu0_fail in
let Cpu1_fail = new_S_cpu1_fail in
let Pmm_fail = new_S_pmm_fail in

(S_state, Reset_cport, Disable_int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist, Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail)"
);;
let PH_B_inst_def = new_definition
('PH_B_inst',
"! (S_fsm_stateA S_fsm_state :sfsm_ty)
(S_soft_cntA S_delayA S_soft_cnt S_delay :wordn)
(S_fsm_sn S_fsm_so S_fsm_srcp S_fsm_sdi S_fsm_src0 S_fsm Src1 S_fsm_spf S_fsm_scOf
S_fsm_sc1f S_fsm_spmf S_fsm_smp S_fsm_src S_fsm_sec S_fsm_srs S fsm_scs S_soft_shot S_soft_shot_delA
S instant S_cpu_histA S fsm_rst S fsm_delay6 S_fsm_delay17 S fsm bothbad S fsm_bypass
S_soft_shot_del S_bad_cpuO S_bad_cpu1 S reset_cpu0 S reset_cpu1 S_pmm fail S cpu0 fail S cpu1 fail
S_cpu_hist S_piu fail :bool)

PH_B_inst (S_fsm stateA, S fsm so, S fsm_so, S fsm src p, S fsm sdi, S fsm sdi, S fsm src0, S fsm src1,
S fsm_spf, S fsm_scOf, S fsm_sc1f, S fsm_spmf, S fsm_smp, S fsm src, S fsm sec, S fsm srs,
S fsm_scs, S soft_shot, S soft_shot_delA, S soft_cntA, S delayA, S instant, S cpu_histA,
S instant, S fsm_rst, S fsm_delay6, S fsm_delay17, S fsm bothbad, S fsm_bypass,
S soft_shot_del, S soft_cnt, S delay, S bad_cpu0, S bad_cpu1, S reset_cpu0, S reset_cpu1,
S_pmm fail, S cpu0 fail, S cpu1 fail, S cpu_hist, S_piu fail)

(ClkA, CkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0, Failure1_ :bool)

let s_soft_cnt_out = ((S_soft_shot_b S_soft_shot_delA) => (INCN 2 S_soft_cntA) I S_soft_cntA) in
let s_delay_out = ((S_fsm_sec) => (INCN 17 S_delayA) I S_delayA) in
let s_cpu0 ok = ((S fsm_scOf b Failure0_ b (s_soft_cnt out = (WORDN 5))) in
let s_cpu1 ok = ((S fsm_sc1f b Failure1_ b (s_soft_cnt out = (WORDN 5))) in
let new_S_soft_shot_del = S_soft_shot in
let new_S soft_cnt = ((~Gcrh b ~Gcrl) => (WORDN 0) I s_soft_cnt out) in
let new_S delay = s_delay out in
let new_S_pmm fail =
((S_fsm sb b S fsm spmf) => T I
((S_fsm sb b S fsm spmf) => F I
((~S_fsm sb b S fsm spmf) => S_pmm fail I ARB)) in
let new S_cpu0 fail =
((S_fsm sb b S_fsm spmf) => T I
((S_fsm sb b S fsm spmf) => F I
((~S_fsm sb b S fsm spmf) => S_cpu0 fail I ARB)) in
let new S_cpu1 fail =
((S_fsm sb b ~S_cpu1 ok b Bypass) => T I
((S_fsm sb b ~S_cpu1 ok b Bypass) => F I
((~S_fsm sb b ~S_cpu1 ok b Bypass) => S_cpu1 fail I ARB)) in
let new_S_piu fail =
((S_fsm sb b ~S_fsm spf b Bypass) => T I
((S_fsm sb b ~S_fsm spf b Bypass) => F I
((~S_fsm sb b ~S_fsm spf b Bypass) => S_piu fail I ARB)) in
let s_cpu0 select = ((S fsm sn V S fsm so) b new S_cpu0 fail) in
let s_cpu1 select = ((S fsm sn V S fsm so) b new S_cpu0 fail b new S_cpu1 fail) in
let new_S bad_cpu0 =
((S_fsm sb b ~S_cpu0 select) => T I
((S_fsm sb b ~S_cpu0 select) => F I
((~S_fsm sb b ~S_cpu0 select) => S_bad_cpu0 I ARB)) in
let new_S bad_cpu1 =
((S_fsm sb b ~S_cpu1 select) => T I
((S_fsm sb b ~S_cpu1 select) => F I
((~S_fsm sb b ~S_cpu1 select) => S_bad_cpu1 I ARB)) in

178
let new_SReset_cpu0 = (new_Sbad_cpu0 \& S_fsm_src0) in
let new_SReset_cpu1 = (new_Sbad_cpu1 \& S_fsm_src1) in
let new_S_cpu_hist = S_cpu_histA in
let new_Sfsm_state = S_fsm_stateA in
let new_Sfsm_rst = Rst in
let new_Sfsm_delay6 = (ELEMENT s_delay_out (6)) in
let new_Sfsm_delay17 = ((Test) \: \rightarrow (ELEMENT s_delay_out (6)) \& (ELEMENT s_delay_out (17))) in
let new_SfsmBothbad = (new_Scpu0_fail \& new_Scpu1_fail) in
let new_SfsmBypass = Bypass in
let new_Sfsm_stateA = S_fsm_stateA in
let new_Sfsm_sn = S_fsm_sn in
let new_Sfsm_so = S_fsm_so in
let new_Sfsm_srcp = S_fsm_srcp in
let new_Sfsm_sdi = S_fsm_sdi in
let new_Sfsm_srp = S_fsm_srp in
let new_Sfsm_src0 = S_fsm_src0 in
let new_Sfsm_src1 = S_fsm_src1 in
let new_Sfsm_spf = S_fsm_spf in
let new_Sfsm_sc0f = S_fsm_sc0f in
let new_Sfsm_sc1f = S_fsm_sc1f in
let new_Sfsm_spnf = S_fsm_spnf in
let new_Sfsm_sbf = S_fsm_sbf in
let new_Sfsm_src = S_fsm_src in
let new_Sfsm_sc = S_fsm_sc in
let new_Sfsm_srs = S_fsm_srs in
let new_Sfsm_scs = S_fsm_scs in
let new_Ssoft_shot = S_soft_shot in
let new_Ssoft_shot_delA = S_soft_shot_delA in
let new_Ssoft_cntA = S_soft_cntA in
let new_SdelayA = S_delayA in
let new_Sinstant = S_instant in
let new_Scpu_histA = S_cpu_histA in

(new_Sfsm_stateA, new_Sfsm_sn, new_Sfsm_so, new_Sfsm_srcp, new_Sfsm_sdi, new_Sfsm_srp,
new_Sfsm_src0, new_Sfsm_src1, new_Sfsm_spf, new_Sfsm_sc0f, new_Sfsm_sc1f, new_Sfsm_spnf,
new_Sfsm_sbf, new_Sfsm_src, new_Sfsm_sc, new_Sfsm_srs, new_Sfsm_scs, new_Ssoft_shot,
new_Ssoft_shot_delA, new_Ssoft_cntA, new_SdelayA, new_Sinstant, new_Scpu_histA, new_Sfsm_state,
new_Sfsm_rst, new_Sfsm_delay6, new_Sfsm_delay17, new_SfsmBothbad, new_SfsmBypass,
new_Ssoft_shot_del, new_Ssoft_cnt, new_Sdelay, new_Sbad_cpu0, new_Sbad_cpu1, new_Sreset_cpu0,
new_Sreset_cpu1, new_Spmm_fail, new_Scpu0_fail, new_Scpu1_fail, new_Scpu_hist, new_Spiu_fail)";)

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

let PH_B_out_def = new_definition
('PH_B_out',
"!(S_fsm_stateA S fsm_state : sfsm_ty)
(S_soft_cntA S_delayA S_soft_cnt S_delay :wordn)
(S_fsm_sn S_fsm_so S_fsm_srcp S_fsm_sdi S_fsm_srp S_fsm_src0 S_fsm_src1 S_fsm_spf S_fsm_sc0f
S_fsm_sc1f S_fsm_spnf S_fsm_sbf S_fsm_src S_fsm_sc S_fsm_srs S_fsm_scs S_soft_shot S_soft_shot_delA
S_instant S_cpu_histA S_fsm_rst S_fsm_delay6 S_fsm_delay17 S_fsmBothbad S_fsmBypass
S_soft_shot_del S_bad_cpu0 S_bad_cpu1 S_reset_cpu0 S_reset_cpu1 S_pmm_fail S_cpu0_fail S_cpu1_fail")

179
S_cpu_hist S_piu_fail :bool
)(CIA CikB Rst Bypass Test Gcrh Gcrl Failure0_Failure1 :bool).

{PH_B_out (S_fsm_stateA, S_fsm_sn, S_fsm_so, S_fsm_spf, S_fsm_sp, S_fsm_sdi, S_fsm_src0, S_fsm_src1,
S_fsm_scof, S_fsm_sclf, S_fsm_spmf, S_fsm_sc, S fsm_row, S_fsm_row0, S_fsm_row1, S_fsm_row2, S_fsm_row3,
S_fsm_row4, S_fsm_row5, S_fsm_row6, S_fsm_row7, S_fsm_row8, S_fsm_row9, S_fsm_row10, S_fsm_row11,
S_fsm_row12, S_fsm_row13, S_fsm_row14, S_fsm_row15, S_fsm_row16, S_fsm_row17, S_fsm_row18, S_fsm_row19,
S_fsm_row20, S_fsm_row21, S_fsm_row22, S_fsm_row23, S_fsm_row24, S_fsm_row25, S_fsm_row26, S_fsm_row27,
S_fsm_row28, S_fsm_row29, S_fsm_row30, S_fsm_row31, S_fsm_row32, S_fsm_row33, S_fsm_row34, S_fsm_row35,
S_fsm_row36, S_fsm_row37, S_fsm_row38, S_fsm_row39, S_fsm_row40, S_fsm_row41, S_fsm_row42, S_fsm_row43,
S_fsm_row44, S_fsm_row45, S_fsm_row46, S_fsm_row47, S_fsm_row48, S_fsm_row49, S_fsm_row50, S_fsm_row51,
S_fsm_row52, S_fsm_row53, S_fsm_row54, S_fsm_row55, S_fsm_row56, S_fsm_row57, S_fsm_row58, S_fsm_row59,
S_fsm_row60, S_fsm_row61, S_fsm_row62, S_fsm_row63, S_fsm_row64, S_fsm_row65, S_fsm_row66, S_fsm_row67,
S_fsm_row68, S_fsm_row69, S_fsm_row70, S_fsm_row71, S_fsm_row72, S_fsm_row73, S_fsm_row74, S_fsm_row75,
S_fsm_row76, S_fsm_row77, S_fsm_row78, S_fsm_row79, S_fsm_row80, S_fsm_row81, S_fsm_row82, S_fsm_row83,
S_fsm_row84, S_fsm_row85, S_fsm_row86, S_fsm_row87, S_fsm_row88, S_fsm_row89, S_fsm_row90, S_fsm_row91,
S_fsm_row92, S_fsm_row93, S_fsm_row94, S_fsm_row95, S_fsm_row96, S_fsm_row97, S_fsm_row98, S_fsm_row99,
S_fsm_row100, S_fsm_row101, S_fsm_row102, S_fsm_row103, S_fsm_row104, S_fsm_row105, S_fsm_row106,
S_fsm_row107, S_fsm_row108, S_fsm_row109, S_fsm_row110, S_fsm_row111, S_fsm_row112, S_fsm_row113,
S_fsm_row114, S_fsm_row115, S_fsm_row116, S_fsm_row117, S_fsm_row118, S_fsm_row119, S_fsm_row120,
S_fsm_row121, S_fsm_row122, S_fsm_row123, S_fsm_row124, S_fsm_row125, S_fsm_row126, S_fsm_row127,
S_fsm_row128, S_fsm_row129, S_fsm_row130, S_fsm_row131, S_fsm_row132, S_fsm_row133, S_fsm_row134,
S_fsm_row135, S_fsm_row136, S_fsm_row137, S_fsm_row138, S_fsm_row139, S_fsm_row140, S_fsm_row141,
S_fsm_row142, S_fsm_row143, S_fsm_row144, S_fsm_row145, S_fsm_row146, S_fsm_row147, S_fsm_row148,
S_fsm_row149, S_fsm_row150, S_fsm_row151, S_fsm_row152, S_fsm_row153, S_fsm_row154, S_fsm_row155,
S_fsm_row156, S_fsm_row157, S_fsm_row158, S_fsm_row159, S_fsm_row160, S_fsm_row161, S_fsm_row162,
S_fsm_row163, S_fsm_row164, S_fsm_row165, S_fsm_row166, S_fsm_row167, S_fsm_row168, S_fsm_row169,
S_fsm_row170, S_fsm_row171, S_fsm_row172, S_fsm_row173, S_fsm_row174, S_fsm_row175, S_fsm_row176,
S_fsm_row177, S_fsm_row178, S_fsm_row179, S_fsm_row180, S_fsm_row181, S_fsm_row182, S_fsm_row183,
S_fsm_row184, S_fsm_row185, S_fsm_row186, S_fsm_row187, S_fsm_row188, S_fsm_row189, S_fsm_row190,
S_fsm_row191, S_fsm_row192, S_fsm_row193, S_fsm_row194, S_fsm_row195, S_fsm_row196, S_fsm_row197,
S_fsm_row198, S_fsm_row199, S_fsm_row200, S_fsm_row201, S_fsm_row202, S_fsm_row203, S_fsm_row204,
S_fsm_row205, S_fsm_row206, S_fsm_row207, S_fsm_row208, S_fsm_row209, S_fsm_row210, S_fsm_row211,
S_fsm_row212, S_fsm_row213, S_fsm_row214, S_fsm_row215, S_fsm_row216, S_fsm_row217, S_fsm_row218,
S_fsm_row219, S_fsm_row220, S_fsm_row221, S_fsm_row222, S_fsm_row223, S_fsm_row224, S_fsm_row225,
S_fsm_row226, S_fsm_row227, S_fsm_row228, S_fsm_row229, S_fsm_row230, S_fsm_row231, S_fsm_row232,
S_fsm_row233, S_fsm_row234, S_fsm_row235, S_fsm_row236, S_fsm_row237, S_fsm_row238, S_fsm_row239,
S_fsm_row240, S_fsm_row241, S_fsm_row242, S_fsm_row243, S_fsm_row244, S_fsm_row245, S_fsm_row246,
let new_S_fsm_so = S_fsm_so in
let new_S_fsm_srcp = S_fsm_srcp in
let new_S_fsm_sdi = S_fsm_sdi in
let new_S_fsm_srp = S_fsm_srp in
let new_S_fsm_src0 = S_fsm_src0 in
let new_S_fsm_src1 = S_fsm_src1 in
let new_S_fsm_spf = S_fsm_spf in
let new_S_fsm_sc0f = S_fsm_sc0f in
let new_S_fsm_sc1f = S_fsm_sc1f in
let new_S_fsm_spmf = S_fsm_spmf in
let new_S_fsm_sbf = S_fsm_sbf in
let new_S_fsm_scf = S_fsm_scf in
let new_S_fsm_spmf = S_fsm_spmf in
let new_S_fsm_sbf = S_fsm_sbf in
let new_S_fsm_src = S_fsm_src in
let new_S_fsm_sec = S_fsm_sec in
let new_S_fsm_srs = S_fsm_srs in
let new_S_fsm_scs = S_fsm_scs in
let new_S_soft_shot = S_soft_shot in
let new_S Mathf_shot_delA = S_mathf_shot_delA in
let new_S_soft_catA = S_soft_catA in
let new_S_delayA = S_delayA in
let new_S_instart = S_instart in
let new_S_cpu_histA = S_cpu_histA in
let ss0 = (ALTER ARBN (0) ((new_S_fsm_stateA = SS) V (new_S_fsm_stateA = SSTOP))
V (new_S_fsm_stateA = SCS) V (new_S_fsm_stateA = SN)
V (new_S_fsm_stateA = SO))) in
let ss1 = (ALTER ss0 (1) ((new_S_fsm_stateA = SCOF) V (new_S_fsm_stateA = ST)
V (new_S_fsm_stateA = SCII) V (new_S_fsm_stateA = SSTOP)
V (new_S_fsm_stateA = SCII)) in
let ss2 = (ALTER ss1 (2) ((new_S_fsm_stateA = SPF) V (new_S_fsm_stateA = SCO1)
V (new_S_fsm_stateA = SCS)) in
let ss3 = (ALTER ss2 (3) ((new_S_fsm_stateA = SRA) V (new_S_fsm_stateA = SPF)
V (new_S_fsm_stateA = ST) V (new_S_fsm_stateA = SCII)
V (new_S_fsm_stateA = SCS) V (new_S_fsm_stateA = SN)
V (new_S_fsm_stateA = SO))) in
let S_state = ss3 in
let Reset_cport = new_S_fsm_srcp in
let Disable_int = (~new_S_instart A ~(new_S_fsm_sn A (ELEMENT s_delay out (6))) A new_S_fsm_sdi) in
let Reset_pi0 = new_S_fsm_srp in
let Reset_cpu0 = new_S_reset_cpu0 in
let Reset_cpu1 = new_S_reset_cpu1 in
let Cpu_hist = new_S_cpu_hist in
let Piu_fail = new_S_piu_fail in
let Cpu0_fail = new_S_cpu0_fail in
let Cpu1_fail = new_S_cpu1_fail in
let Pnm_fail = new_S_pnm_fail in

(S_state, Reset_cport, Disable_int, Reset_pi0, Reset_cpu0, Reset_cpu1, Cpu_hist, Piu_fail, Cpu0_fail,
Cpu1_fail, Pnm_fail))

);;

close_theory();;

181
Appendix D ML Source for the Clock-Level Specification of the PIU Ports.

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

D.1 P Port Specification

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

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

```
next-state definition for EXEC instruction.
''
```

```
let pEXEC_inst_def = new_definition ('pEXEC_inst',
    "! (P fsm_state :fsm_ty)
    (P addr P be P size :wordn)
    (P destl P wr P fsm_rst P fsm_sack P fsm_cgnt P fsm_hold P rqnt P size P down P lock P lock_in P male P tale P)
    (:pc_state_ty)";
```

182
let new_P_fsm_state =
  ((P_fsm_rst) => PA |
  ((P_fsm_state = PH) => ((~P_fsm_hold_) => PH | PA)) |
  ((P_fsm_state = PA) =>
    (((P_rqt ∧ P_destl) ∨ (P_rqt ∧ P_fsm_cgnU)) => PD) |
    ((~P_fsm_hold_ ∧ P_lock_) => PH | PA)) |
  ((P_fsm_state = PD) =>
    (((P fsm_sack ∧ P_fsm_hold_) ∨ (P fsm_sack ∧ ~P_fsm_hold_ ∧ ~P_lock_)) => PA |
    ((P_fsm_sack ∧ ~P_fsm_hold_ ∧ P_lock_) => PH | PD)) | PILL)))) in
let new_P_addr = ((~P_rqt) => (SUBARRAY L_ad_in (25,0)) | P_addr) in
let new_P_destl = ((~P_rqt) => (ELEMENT L_ad_in (31)) | P_destl) in
let new_P_be_ = ((~P_rqt) => L_be_ | P_be) in
let new_P_wr = ((~P_rqt) => L_wr | P_wr) in
let new_P_size =
  ((~P_rqt) => (SUBARRAY L_ad_in (1,0)) |
  ((P_down) => (DECN 1 | P_size)) | P_size)) in
let new_P_be_ = (~L_be_ ∧ L_den_) in
let new_P_addr = ((~P_rqt) => L_be_ | P_be) in
let new_P_size =
  ((~P_rqt) => (SUBARRAY L_ad_in (1,0)) |
  ((P_down) => (DECN 1 | P_size)) | P_size)) in
let new_P_be_ = (~L_be_ ∧ L_den_) in
let p_sack = ((P_size = ((P_down) => (WORDN 1) | (WORDN 0))) ∧ ~L_srdy ∧ (new_P_fsm_state = PD)) in
let new_P_rqt =
  ((p_sack ∨ (p_sack ∨ P_lst)) => T |
  ((~p_sack ∨ (p_sack ∨ P_lst)) => F |
  (~p_sack ∨ (p_sack ∨ P_lst)) => P_rqt | ARB))) in
let new_P_down = (~L_srdy ∧ (new_P_fsm_state = PD)) in
let new_P_male_ = ((new_P_fsm_state = PA) =>
  (~new_P_destl ∧ (SUBARRAY new_P_addr (25,24))) ∧ new_P_rqt) | P_male_ in
let new_P_rslt = ((new_P_fsm_state = PA) =>
  (~new_P_destl ∧ (SUBARRAY new_P_addr (25,24))) ∧ new_P_rqt) | P_rslt in
let new_P_lock_ =
  ((P_lst) => T |
  ((new_P_fsm_state = PD) => L_lock_ | P_lock_)) in
let new_P_lock_inh_ =
  ((P_lst) => T |
  (~new_P_male_ ∨ ~new_P_rslt) => L_lock_ | P_lock_inh_)) in
let new_P_fsm_rst = P_lst in
let new_P_fsm_sack = P_sack in
let new_P_fsm_cgnU = P_cgnU in
let new_P_fsm_hold_ = P_hold_ in

(new_P_addr, new_P_destl, new_P_be_, new_P_wr, new_P_fsm_state, new_P_fsm_rst, new_P_fsm_sack,
new_P_fsm_cgnU, new_P_fsm_hold_, new_P_lst, new_P_size, new_P_down, new_P_lock_, new_P_lock_inh_,
new_P_male_, new_P_rslt)"
let pEXEC_out_def = new_definition
('pEXEC_out').

"1 (P_fsm_state :pfsm_ty)
(P_addr P_be_P_size :wordn)
(P_destl P_wr P_fsm_rst P_fsm_sack P_fsm_cgnt P_fsm_hold P_rqt P_down P_lock_ 
P_lock_inh P_male P_rale :bool)
(L_ad_l_in L_be L_ad_in:wordn)
(ClkA ClkB Rst L_ads_ L_den_ L.wr L_lock_ L_cgnt_ L_hold_ L_srdy_ :bool).

pEXEC_out (P_addr, P_destl, P_be, P_wr, P_fsm_state, P_fsm_rst, P_fsm_sack, P_fsm_cgnt, P_fsm_hold, 
P_rqt, P_size, P_down, P_lock_, P_lock_inh_, P_male_, P_rale_)
(ClkA, ClkB, Rst, L_ad_in, L_ads_, L_den_, L_be, L_wr, L_lock_, L_ad_in, L_cgnt_, L_hold_, 
L_srdy_) =

let new_P_fsm_state =

(P_fsm_rst => PA) |
((P_fsm_state = PH) => ((-P_fsm_hold_) => PH | PA)) |
((P_fsm_state = PA) =>
  ((P_rqt AND -P_destl) V (P_rqt AND P_destl AND -P_fsm_cgnt_)) => PD |
  ((-P fsm_hold_ AND P_lock_) => PH | PA)) |
((P_fsm_state = PD) =>
  ((P_fsm_sack AND -P_fsm_hold_ AND P_lock_) => PH | PD) | P_ILL)))

let new_P_addr = ((-P_rqt) => (SUBARRAY L_ad_in (25,0)) | P_addr) in 
let new_P_destl = ((-P_rqt) => (ELEMENT L_ad_in (31)) | P_destl) in 
let new_P_be = ((-P_rqt) => L_be | P_be_ in 
let new_P_wr = ((-P_rqt) => L_wr | P_wr) in 
let new_P_size =

((P_down) => (DECN 1 P_size) | P_size) in 
let p_ale = (-L_ads_ | L_den_) in 
let p_sack = ((new_P_size = (P_down) => (WORDN 1) 1 (WORDN 0)) AND -L_srdy_ AND (new_P_fsm_state = PD)) in 
let new_P_rqt =

((p_ale AND -p_sack V Rst)) => T |
((p_ale AND (p_sack V Rst)) => F) |
((p_ale AND -(p_sack V Rst)) => P_rqt | ARB)) in 
let new_P_down = (-L_srdy_ AND (new_P_fsm_state = PD)) in 
let new_P_male = ((new_P_fsm_state = PA) =>
  (~(-new_P_destl AND (SUBARRAY new_P_addr (25,24)) = (WORDN 3)) AND new_P_rqt) | P_male_) in 
let new_P_rale = ((new_P_fsm_state = PA) =>
  (~(-new_P_destl AND (SUBARRAY new_P_addr (25,24)) = (WORDN 3)) AND new_P_rqt) | P_rale_) in 
let new_P_lock =

((Rst) => T) |
((new_P_fsm_state = PD) => L_lock_ | P_lock_) in 
let new_P_lock_inh =

((Rst) => T) |
((~new_P_male V ~new_P_rale) => L_lock_ | P_lock_inh)) in 
let new_P_fsm_rst = Rst in 
let new_P_fsm_sack = p_sack in 
let new_P_fsm_cgnt_ = L_cgnt_ in 
let new_P_fsm_hold_ = L_hold_ in 
let L_ad_out = (((~new_P_fsm_state = PA)) AND (~new_P_fsm_state = PH))
\[
\begin{align*}
\lambda & \sim ((\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PD}) \land \text{new}\_\text{P}\_\text{wr}) \Rightarrow \text{l}\_\text{ad}\_\text{in} \mid \text{ARBN}) \text{ in} \\
\text{let}\ & \text{L}\_\text{ready} = \sim (\text{l}\_\text{srty} \land (\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PD})) \text{ in} \\
\text{let}\ & \text{od}0 = \text{ARBN} \text{ in} \\
\text{let}\ & \text{od}1 = (\text{MALTER}\ \text{od}0 (31,27) \text{ new}\_\text{P}\_\text{be}) \text{ in} \\
\text{let}\ & \text{od}2 = (\text{ALTER}\ \text{od}1 (26) \ F) \text{ in} \\
\text{let}\ & \text{od}3 = (\text{MALTER}\ \text{od}2 (25,24) (\text{SUBARRAY}\ \text{new}\_\text{P}\_\text{addr} (1,0))) \text{ in} \\
\text{let}\ & \text{od}4 = (\text{MALTER}\ \text{od}3 (23,0) (\text{SUBARRAY}\ \text{new}\_\text{P}\_\text{addr} (25,2))) \text{ in} \\
\text{let}\ & \text{L}\_\text{ad}\_\text{addr}\_\text{out} = ((\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PA}) \Rightarrow \text{od}4 \mid \text{ARBN}) \text{ in} \\
\text{let}\ & \text{L}\_\text{ad}\_\text{data}\_\text{out} = (((\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PD}) \land \text{new}\_\text{P}\_\text{wr}) \Rightarrow \text{L}\_\text{ad}\_\text{in} \mid \text{ARBN}) \text{ in} \\
\text{let}\ & \text{L}\_\text{be} = (((\sim (\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PH})) \Rightarrow ((\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PA}) \Rightarrow \text{new}\_\text{P}\_\text{be} \mid \text{L}\_\text{be}) \mid \text{ARBN}) \text{ in} \\
\text{let}\ & \text{L}\_\text{role} = (((\sim (\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PH})) \Rightarrow \\
\sim (\text{new}\_\text{P}\_\text{dest}) \land ((\text{SUBARRAY}\ \text{new}\_\text{P}\_\text{addr} (25,24)) = (\text{WORDN} \ 3)) \land ((\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PA}) \land (\text{new}\_\text{P}\_\text{rqt}) \mid \text{ARBN}) \text{ in} \\
\text{let}\ & \text{L}\_\text{male} = ((\sim (\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PH})) \Rightarrow \\
\sim (\text{new}\_\text{P}\_\text{dest}) \land ((\text{SUBARRAY}\ \text{new}\_\text{P}\_\text{addr} (25,24)) = (\text{WORDN} \ 3)) \land ((\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PA}) \land (\text{new}\_\text{P}\_\text{rqt}) \mid \text{ARBN}) \text{ in} \\
\text{let}\ & \text{L}\_\text{crqt} = \sim (\text{new}\_\text{P}\_\text{dest} \land (\text{new}\_\text{P}\_\text{rqt}) \mid \text{in} \\
\text{let}\ & \text{L}\_\text{cale} = \sim (\sim \text{l}\_\text{cnt} \land (\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PA}) \land \text{l}\_\text{hold}) \mid \text{in} \\
\text{let}\ & \text{L}\_\text{mrdy} = \sim (\sim (\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PH})) \Rightarrow (\text{F} \mid \text{ARBN}) \mid \text{in} \\
\text{let}\ & \text{L}\_\text{last} = ((\sim (\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PH})) \Rightarrow (\text{P}\_\text{size} = ((\text{P}\_\text{down}) \Rightarrow (\text{WORDN} \text{ 1} \mid (\text{WORDN} \text{ 0})) \mid \text{ARBN}) \mid \text{in} \\
\text{let}\ & \text{L}\_\text{hida} = \sim (\text{new}\_\text{P}\_\text{fsm}\_\text{state} = \text{PH}) \mid \text{in} \\
\text{let}\ & \text{L}\_\text{lock} = \sim (\sim \text{new}\_\text{P}\_\text{lock} \land (\text{new}\_\text{P}\_\text{lock}\_\text{inh}) \mid \text{in} \\
\text{(L}\_\text{ad}\_\text{out}, \text{L}\_\text{ready}, \text{L}\_\text{ad}\_\text{data}\_\text{out}, \text{L}\_\text{ad}\_\text{addr}\_\text{out}, \text{L}\_\text{be}, \text{L}\_\text{role}, \text{L}\_\text{male}, \text{L}\_\text{crqt}, \text{L}\_\text{cale}, \text{L}\_\text{mrdy}, \text{L}\_\text{last}, \text{L}\_\text{hida}, \text{L}\_\text{lock}) \text{"} \\
\)\); \\
\text{close}\_\text{theory}(); \\
\)
D.2 M Port Specification

File: m_clock1.ml

Author: (c) D.A. Fura 1992

Date: 31 March 1992

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

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

system 'rm m_clock1.th';;

new_theory 'm_clock1';;

load 'abstract';;

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

let mc_state_t = "((M_fsm_state, M_fsm_male_, M_fsm_last_, M_fsm_mrdy_, M_fsm_rst, M_count, M_se, M_wr, M_addr, M_be, M_rdy, M_wwdel, M_parity, M_rd_data, M_detect):^mc..state_ty)";

let mc_env_t = "((ClkA, ClkB, Rst, Disable_eeprom, DisableWrites, I_ad_in, I_male_, I_last_, I_be_, I_mrdy_, MB_data_in, Edac_en_, Reset_parity):^mc_env_ty)";

let mc_out_t = "((Lad_out, Lsrdy_, MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_, MB_parity):^mc_out_ty)";

let rep_t = abstract_type 'aux_def' 'Andn';;

% Next-state definition for EXEC instruction.

let mEXEC_inst_def = new_definition ('mEXEC_inst',
  "I (M_fsm_state :msfm_ty)
   (M_count M_addr M_be M_rdy_data M_detect :wordn)
   (M_fsm_male_ M_fsm_last_ M_fsm_mrdy_ M_fsm_rst M_se M_wr M_rdy M_wwdel M_parity :bool)
   (I_ad_in I_be_ MB_data_in :wordn)
  "

-- 186 --
let m_bw = ((~(M_be = (WORDN 15))) \& M_wr \& (M_fsm_state = MI))) in
let m_ww = ((M_be = (WORDN 15)) \& M_wr \& (M_fsm_state = MI))) in
let new_M_fsm_state =
((M_fsm_rst) => MI)
((M_fsm_state = MI) => ((~M_fsm_male_) => MA \& MI))
((M_fsm_state = MA) =>
((~M_fsm_mrdy_ \& m_ww) => MW)
((~M_fsm_mrdy_ \& (M_wr \& (~M_fsm_state = MI)) \& m_bw) => MR \& MA))
((M_fsm_state = MR) =>
((m_bw \& (M_count = (WORDN 0))) => MBW)
((M_fsm_last_ \& ~M_wr \& (~M_fsm_state = MI)) \& (M_count = (WORDN 0))) => MA)
((~M_fsm_last_ \& ~M_wr \& (~M_fsm_state = MI)) \& (M_count = (WORDN 0))) => MRR \& MR)
((M_fsm_state = MRR) => MI)
((M_fsm_state = MW) =>
((~M_fsm_last_ \& (M_count = (WORDN 0))) => MI)
((M_fsm_last_ \& (M_count = (WORDN 0))) => MA \& MW))
((M_fsm_state = MBW) => MW \& M_ILL)) in
let new_M_se = ((~I_male_) => (ELEMENT I_ad_in (23)) \& M_se) in
let new_M_wr = ((~I_male_) => (ELEMENT I_ad_in (27)) \& M_wr) in
let new_M_addr =
((~I_male_) => (SUBARRAY I_ad_in (18,0)))
((M_rdy) => (INCN 18 M_addr) \& M_addr)) in
let new_M_count =
((new_M_fsm_state = MA) \& (new_M_fsm_state = MBW)) =>
((new_M_se) => (WORDN 1)) \& (WORDN 2))
((new_M_fsm_state = MBW) \& (new_M_fsm_state = MR)) =>
(DECN 2 M_count) \& M_count)) in
let m_rdy_ = ((new_M_fsm_state = MW) \& (new_M_count = (WORDN 0)))
V ((new_M_fsm_state = MR) \& (new_M_count = (WORDN 0)) \& ~new_M_wr) in
let m_srdy_ = ~((m_rdy_ \& ~new_M_wr) \& (m_rdy_ \& new_M_wr)) in
let new_M_be = ((~I_male_ \& ~m_srdy_) => (NOTN 3 I_be_)) \& M_be in
let new_M_rdy = m_rdy in
let new_M_wwdel = ((new_M_fsm_state = MA) \& new_M_wr \& (new_M_be = (WORDN 15))) in
let new_M_rddata = (((new_M_fsm_state = MR)) => (Ham_Dec rep MB_data_in) \& M_rddata) in
let new_M_detect =
(((new_M_fsm_state = MR) \& ~new_M_wr) \& new_M_wr \& (new_M_fsm_state = MI))) =>
((~Edac_en_) => (Ham_Det1 rep MB_data_in) \& WORDN 0) \& M_detect) in
let m_error = (~m_srdy_ \& (new_M_fsm_state = MI)) \& Ham_Det2 rep (new_M_detect, ~Edac_en_) in
let new_M_parity =
((m_error \& ~new_M_wr) \& (new_M_fsm_state = MI)) \& (m_error \& ~new_M_wr) \& (new_M_fsm_state = MI))
\& (Ham_Det1 rep MB_data_in) \& WORDN 0) \& M_detect) in
let m_error = (~m_srdy_ \& (new_M_fsm_state = MI)) \& Ham_Det2 rep (new_M_detect, ~Edac_en_) in
let new_M_parity =
((m_error \& ~new_M_wr) \& (new_M_fsm_state = MI)) \& (m_error \& ~new_M_wr) \& (new_M_fsm_state = MI))
\& (m_error \& ~new_M_wr) \& (new_M_fsm_state = MI))
\& (m_error \& ~new_M_wr) \& (new_M_fsm_state = MI))
\& (m_error \& ~new_M_wr) \& (new_M_fsm_state = MI))
\& (m_error \& ~new_M_wr) \& (new_M_fsm_state = MI))
new_M_se, new_M_wr, new_M_addr, new_M_be, new_M_rdy, new_M_wwdel, new_M_parity, new_M_rd_data, new_M_detect)

Output definition for EXEC instruction.

let mEXEC_out_def = new_definition
('mEXEC_out',
  "(M_fsm_state : mfsm_ty)
   (M_count M_add* M_be M_rdy M_rd_data M_fsm_male_ M_fsm_Jast_ M_fsm_mrdy_ M_fsm_rst M_so M_wr M_wwdel M_parity M_n:l_data M_detect)
   (ClkA ClkB Rst Disable_eeprom Disable_writes I_male_ I_last_ I_mrdy_ Edac_en_ Reset_parity : bool)
   (rep:'rep_ty).

let m_bw = ((-(M_be = (WORDN 15))) A M_wr A (~(M_fsm_state = MI))) in
let m_ww = ((M_be = (WORDN 15)) A M_wr A (~(M_fsm_state = MI))) in
let new_M_fsm_state =
  ((M_fsm_state = MI) => ((-M_fsm_male_) => MA I MI)) I
  ((M_fsm_state = MA) =>
    ((-M_fsm_mrdy_ A m_ww) => MW I
     ((-M_fsm_mrdy_ A ((-M_wr A (-(M_fsm_state = MI))) V m_bw)) => MR I MA)) I
  ((M_fsm_state = MR) =>
    ((m_bw A (M_count = (WORDN 0))) => MBW I
     ((M_fsm_last_ A -M_wr A ((-M_fsm_state = MI))) I (M_count = (WORDN 0))) => MA I
     ((-M_fsm_last_ A ~M_wr A ((-M_fsm_state = MI))) I (M_count = (WORDN 0))) => MRR I MR)) I
  ((M_fsm_state = MRR) => MI I
   ((M_fsm_state = MW) =>
     ((-M_fsm_last_ A (M_count = (WORDN 0))) => MI I
      ((M_fsm_last_ A (M_count = (WORDN 0))) => MA I MW)) I
     ((M_fsm_state = MBW) => MW I M_ILL)))))) in
let new_M_se = ((-I_male_ => (ELEMENT I_ad_in (23))) I M_se) in
let new_M_wr = ((-I_male_ => (ELEMENT I_ad_in (27))) I M_wr) in
let new_M_addr =
  ((-I_male_ => (SUBARRAY I_ad_in (18,0))) I
   ((I_rdy) => (INCN 18 M_addr) I M_addr)) in
let new_M_count =
  (((new_M_fsm_state = MA) V (new_M_fsm_state = MBW)) => ((new_M_se) => (WORDN 1)) I (WORDN 2)) I
  (((new_M_fsm_state = MW) V (new_M_fsm_state = MR)) => (DECN 2 M_count) I M_count)) in
let m_rdy = (((new_M_fsm_state = MW) A (new_M_count = (WORDN 0))) V ((new_M_fsm_state = MR) A (new_M_count = (WORDN 0) I -new_M_wr))) in
let m_mrdy_ = ((-m_rdy A new_M_wr) V (m_rdy A new_M_wr)) in
let new_M_be = ((-I_male_ A -m_mrdy_) => (NOTN 3 I_be) I M_be) in
let new_M_rdy = m_rdy in
let new_M_wwdel = ((new_M_fsm_state = MA) A new_M_wr A (new_M_be = (WORDN 15))) in
let new_M_rd_data = (((new_M_fsm_state = MR)) => (Ham_Dec rep MB_data_in) I M_rd_data) in

188
let new_M_detect =

\(((\text{new\_M\_fsm\_state} = \text{MR}) \land \neg \text{new\_M\_wr}) \lor \text{new\_M\_wr} \lor (\text{new\_M\_fsm\_state} = \text{MI})) \implies

((\neg \text{Edac\_en}) \implies (\text{Ham\_Det1 rep MB\_data\_in} \lor \text{WORDN} 0) \lor \text{M\_detect})\) in

let m_error = \((-\text{m\_srdy} \land \neg (\text{new\_M\_fsm\_state} = \text{MI})) \land \text{Ham\_Det2 rep (new\_M\_detect, \neg \text{Edac\_en})}\) in

let new_M_parity =

\((\text{m\_error} \land \neg (\text{Rst} \lor \text{Reset\_parity})) \implies T \|

((\text{m\_error} \land (\text{Rst} \lor \text{Reset\_parity})) \implies F \|

((\text{m\_error} \land \neg (\text{Rst} \lor \text{Reset\_parity})) \implies \text{M\_parity} \lor \text{ARB}))\) in

let new_M_fsm_male_ = I\_male\_ in

let new_M_fsm_last_ = I\_last\_ in

let new_M fsm_mrdy_ = I\_mrdy\_ in

let new_M_fsm_rst = Rst in

let I\_ad\_out = ((\neg \text{new\_M\_wr} \land (\neg (\text{new\_M\_fsm\_state} = \text{MI}))) \implies \text{M\_rd\_data} \lor \text{ARBN})\) in

let I\_srdy\_ = (((\neg (\text{new\_M\_fsm\_state} = \text{MI}))) \implies \text{m\_srdy} \lor \text{ARB})\) in

let MB\_addr = ((\text{M\_rdy}) \implies (\text{INCN} 18 \text{ M\_addr}) \lor \text{M\_addr})\) in

let MB\_data\_7\_0 = ((\text{ELEMENT M\_be (0)}) \implies (\text{SUBARRAY} \text{I\_ad\_in (7,0)}) \lor (\text{SUBARRAY} \text{M\_rd\_data (7,0)}))\) in

let MB\_data\_15\_8 = ((\text{ELEMENT M\_be (1)}) \implies (\text{SUBARRAY} \text{I\_ad\_in (15,8)}) \lor (\text{SUBARRAY} \text{M\_rd\_data (15,8)}))\) in

let MB\_data\_23\_16 = ((\text{ELEMENT M\_be (2)}) \implies (\text{SUBARRAY} \text{I\_ad\_in (23,16)}) \lor (\text{SUBARRAY} \text{M\_rd\_data (23,16)}))\) in

let MB\_data\_31\_24 = ((\text{ELEMENT M\_be (3)}) \implies (\text{SUBARRAY} \text{I\_ad\_in (31,24)}) \lor (\text{SUBARRAY} \text{M\_rd\_data (31,24)}))\) in

let MB\_data = ((\text{M\_ALTER} ((\text{M\_ALTER} ((\text{M\_ALTER} \text{M\_ALTER} \text{ARBN (7,0)} \text{MB\_data\_7\_0}) \text{(15,8)} \text{MB\_data\_15\_8}) \text{(23,16)} \text{MB\_data\_23\_16}) \text{(31,24)} \text{MB\_data\_31\_24}))\) in

let MB\_data\_out = ((\text{new\_M\_fsm\_state} = \text{MW}) \implies (\text{Ham\_Enc rep MB\_data} \lor \text{ARBN}))\) in

let MB\_cs\_eeprom\_ = (\neg ((\text{new\_M\_fsm\_state} = \text{MI}) \land \neg \text{new\_M\_se})\) in

let MB\_cs\_sram\_ = (\neg ((\text{new\_M\_fsm\_state} = \text{MI}) \land \neg \text{new\_M\_se})\) in

let MB\_we\_ = (\neg \text{new\_M\_se} \lor (\neg (\text{new\_M\_fsm\_state} = \text{MI})) \lor \text{Disable\_eeprom})

\land (\text{m\_error} \land (\text{new\_M\_fsm\_state} = \text{MW}) \lor \text{new\_M\_wwdel}))\) in

let MB\_oe\_ = (\neg \text{new\_M\_wr} \land (\text{new\_M\_fsm\_state} = \text{MA}) \lor (\text{new\_M\_fsm\_state} = \text{MR}))\) in

let MB\_parity = \text{new\_M\_parity} in

(I\_ad\_out, I\_srdy\_, MB\_addr, MB\_data\_out, MB\_cs\_eeprom\_, MB\_cs\_sram\_, MB\_we\_, MB\_oe\_, MB\_parity)"

);;

close\_theory();;
D.3 R Port Specification

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

```ml
set_search_path (search_pathO @ ['/home/titan31dfura/ftep/piu/hol/lib/']);

system 'nn r__clockl.th';

new_theory 'r_clockl';

loadf 'abstract';

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

let rc_state..ty = ":(rfsm-ty#b_ol#bo_l#boo_#bo_#bo_#wordn#boo_#w_rdn#boo_#w_rdn#_r__r_#
wordn#bool#wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#wordn_or_l#
wordn#bo_#w_rdn#b_#w_rdn#b_o_#b_#w_rdn#b_o_#b_#w_rdn#b_o_#b_#w_rdn#b_o_#b_#w_rdn#b_o_#
wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#b
```
let rEXEC_inst_def = new_definition
('rEXEC_inst',
"' (rep 'rep_ty)
(R_fsm_state :fsm_ty)
(R_crt0 in R_crt0 new R_crt0_out R_crt1 in R_crt1 new R_crt1_out R_crt2 in R_crt2 new
R_crt2_out R_crt3 in R_crt3 new R_crt3_out R_irc old R_irc_mask R_irc R_ccr R_gcr R_SR R_reg sel
R_busA_latch :words)
(R_fsm_ale R_fsm_mrdy R_fsm_last R_fsm_rst R_crt0_mux_sel R_crt0_irden R_crt0_cry R_crt0_orden
R_crt1_mux_sel
R_crt1_irden R_crt1_cry R_crt1_ordest R_crt2_mux_sel R_crt2_irden R_crt2_cry R_crt2_orden R_crt2_mux_sel
R_crt3_irden R_crt3_cry R_crt3_orden R_irc_load R_irc_rden R_gcr_rden R_sr_rden R_int0_dis
R_int3_dis R_c01_cout_del R_int1_en R_c23_cout_del R_int2_en R_wr R_cntlatc_del R_srdy_del :bool)
(I_ad_in I_irden I_last_ruffly Disable_int Disable_writes Piu_fail Pmm_fail CB_parity MB_parity :bool)
(ClkA Rst I_rale I_last I_mrdy Disable_int Disable_writes Piu_fail Pmm_fail CB_parity MB_parity :bool).
rEXEC_inst rep
(R_fsm_state, R_fsm_ale, R_fsm_mrdy, R_fsm_last, R_fsm_rst, R_crt0_in, R_crt0_out, R_crt0_orden,
R_crt0_irden, R_crt0_cry, R_crt0_mux_sel, R_crt0, R_crt1_in, R_crt1_out, R_crt1_orden, R_crt1_irden,
R_crt1_cry, R_crt1_mux_sel, R_crt2_in, R_crt2_out, R_crt2_orden, R_crt2_irden, R_crt2_cry, R_crt2_mux_sel,
R_crt3_in, R_crt3_out, R_crt3_orden, R_crt3_irden, R_crt3_cry, R_crt3_mux_sel,
R_irc, R_irc_mask, R_irc_cry, R_irc_rden, R_gcr, R_gcr_rden, R_sr, R_sr_rden, R_int0_dis,
R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatc_del, R_srdy_del,
R_reg sel, R_busA_latch)
(ClkA, Rst, I_ad_in, I_rale, I_last, I_mrdy, Disable_int, Disable_writes, Piu_fail, Pmm_fail, CB_parity, MB_parity, :bool).

let new_R_fsm_state =
((R_fsm_state = RI) => (R_fsm_ale) => (R_fsm_mrdy) => (R_fsm_last) => (R_fsm_rst) =>)
let r_fsm_cntlatc = ((R_fsm_state = RI) => (R_fsm_crt0) => (R_fsm_crt1) => (R_fsm_crt2) => (R_fsm_crt3) =>
(R_fsm_ale) => (R_fsm_mrdy) => (R_fsm_last) => (R_fsm_rst) => (R_crt0_in) => (R_crt0_out) => (R_crt0_orden) =>
(R_crt0_irden) => (R_crt0_cry) => (R_crt0_mux sel) => (R_crt1_in) => (R_crt1_out) => (R_crt1_orden) =>
(R_crt1_irden) => (R_crt1_cry) => (R_crt1_mux sel) => (R_crt2_in) => (R_crt2_out) => (R_crt2_orden) =>
(R_crt2_irden) => (R_crt2_cry) => (R_crt2_mux sel) => (R_crt3_in) => (R_crt3_out) => (R_crt3_orden) =>
(R_crt3_irden) => (R_crt3_cry) => (R_crt3_mux sel) => (R_irc) => (R_irc_mask) => (R_irc_cry) =>
(R_irc_rden) => (R_gcr) => (R_gcr_rden) => (R_sr) => (R_sr_rden) => (R_int0_dis) =>
(R_int3_dis) => (R_c01_cout_del) => (R_int1_en) => (R_c23_cout_del) => (R_int2_en) => (R_wr) =>
(R_cntlatc_del) => (R_srdy_del) => (R_reg sel) => (R_busA_latch))

let new_R_reg sel =
((I_rale) => (SUBARRAY I_ad_in (3,0)) =>
(I_srdy_del) => (INCN 3 R_reg sel) =>
let r_reg sel = ((R_srdy_del) => (INCN 3 R_reg sel) =>
let r_wr A = (R_reg sel RD) =>
let r_writeB = (R_reg sel RD) =>
let r_readA = (R_reg sel RD) =>
let r_readB = (R_reg sel RD) =>
let r_cir wr01A = ((r_writeA (R_reg sel WORDN 8)) =>
let r_cir wr01B = ((r_writeB (R_reg sel WORDN 8)) =>
let r_cir wr23A = ((r_writeA (R_reg sel WORDN 10)) =>
let r_cir wr23B = ((r_writeB (R_reg sel WORDN 10)) =>
let new_R_crr = (r_writeB (R_reg sel WORDN 10)) =>
let new_R_crr_rden = (r_readB (R_reg sel WORDN 3)) =>

191
let new_R_gcr = ((r_writeB \& (r_reg_sel = (WORDN 2))) => I_ad in \ R_gcr) in
let new_R_gcr_rden = (r_readB \& (r_reg_sel = (WORDN 2))) in
let new_R_c01_cout_del = R_ctrl_cry in
let new_R_intl_en = (((ELEMENT new_R_gcr (18)) \& (r_cmds_wr01B \& (R_ctrl_cry \& (ELEMENT new_R_gcr (16))))) \&
(-(ELEMENT new_R_gcr (18)) \& (r_cmds_wr01B \& (R_ctrl_cry \& (ELEMENT new_R_gcr (16))))) => T
((-(ELEMENT new_R_gcr (18)) \& (r_cmds_wr01B \& (R_ctrl_cry \& (ELEMENT new_R_gcr (16))))) \&
(-(ELEMENT new_R_gcr (18)) \& (r_cmds_wr01B \& (R_ctrl_cry \& (ELEMENT new_R_gcr (16))))) => F
((-(ELEMENT new_R_gcr (18)) \& (r_cmds_wr01B \& (R_ctrl_cry \& (ELEMENT new_R_gcr (16))))) \&
(-(ELEMENT new_R_gcr (18)) \& (r_cmds_wr01B \& (R_ctrl_cry \& (ELEMENT new_R_gcr (16))))) => R
goal
let new_R_c23_cout_del = R_ctrl3_cry in
let new_R_int2_en = (((ELEMENT new_R_gcr (22)) \& (r_cmds_wr23B \& (R_ctrl3_cry \& (ELEMENT new_R_gcr (20))))) \&
(-(ELEMENT new_R_gcr (22)) \& (r_cmds_wr23B \& (R_ctrl3_cry \& (ELEMENT new_R_gcr (20))))) => T
((-(ELEMENT new_R_gcr (22)) \& (r_cmds_wr23B \& (R_ctrl3_cry \& (ELEMENT new_R_gcr (20))))) \&
(-(ELEMENT new_R_gcr (22)) \& (r_cmds_wr23B \& (R_ctrl3_cry \& (ELEMENT new_R_gcr (20))))) => F
goal
((-(ELEMENT new_R_gcr (22)) \& (r_cmds_wr23B \& (R_ctrl3_cry \& (ELEMENT new_R_gcr (20))))) \&
(-(ELEMENT new_R_gcr (22)) \& (r_cmds_wr23B \& (R_ctrl3_cry \& (ELEMENT new_R_gcr (20))))) => R
goal
let new_R_ctr0_in = ((r_writeB \& (r_reg_sel = (WORDN 8))) => I_ad in \ R_ctr0_in) in
let new_R_ctr0_mux_sel = (r_cmds_wr01B \& (ELEMENT new_R_gcr (16)) \& R_ctr0_cry)) in
let new_R_ctr0_iden = (r_readB \& (r_reg_sel = (WORDN 8))) in
let new_R_ctr0 = ((R_ctr0_mux_sel) \& R_ctr0_in \ R_ctr0_new) in
let new_R_ctr0_new = ((ELEMENT new_R_gcr (19))) => (INCN 31 R_ctr0) \& R_ctr0_new) in
let new_R_ctr0_cry = (R_ctr0_new \& (R_ctr0_cry)) in
let new_R_ctr0_ordem = (r_readB \& (r_reg_sel = (WORDN 12))) in
let new_R_ctr1_in = ((r_writeB \& (r_reg_sel = (WORDN 9))) => I_ad in \ R_ctr1_in) in
let new_R_ctr1_mux_sel = (r_cmds_wr01B \& (ELEMENT new_R_gcr (16)) \& R_ctr1_cry)) in
let new_R_ctr1_iden = (r_readB \& (r_reg_sel = (WORDN 9))) in
let new_R_ctr1 = ((R_ctr1_mux_sel) \& R_ctr1_in \ R_ctr1_new) in
let new_R_ctr1_new = ((R_ctr0_cry) => (INCN 31 R_ctr1) \& R_ctr1_new) in
let new_R_ctr1_cry = (R_ctr1_new \& (R_ctr1_cry)) in
let new_R_ctr1_out = ((R_ctr1_mux_sel) \& R_ctr1_new \ R_ctr1_out) in
let new_R_ctr1_ordem = (r_readB \& (r_reg_sel = (WORDN 13))) in
let new_R_ctr2_in = ((r_writeB \& (r_reg_sel = (WORDN 10))) => I_ad in \ R_ctr2_in) in
let new_R_ctr2_mux_sel = (r_cmds_wr23B \& (ELEMENT new_R_gcr (20)) \& R_ctr3_cry)) in
let new_R_ctr2_iden = (r_readB \& (r_reg_sel = (WORDN 10))) in
let new_R_ctr2 = ((R_ctr2_mux_sel) \& R_ctr2_in \ R_ctr2_new) in
let new_R_ctr2_new = ((ELEMENT new_R_gcr (23))) => (INCN 31 R_ctr2) \& R_ctr2_new) in
let new_R_ctr2_cry = (R_ctr2_new \& (R_ctr2_cry)) in
let new_R_ctr2_hid = (r_readB \& (r_reg_sel = (WORDN 14))) in
let new_R_ctr2_out = ((R_ctr2_hid) \& R_ctr2_new \ R_ctr2_out) in
let new_R_ctr2_ordem = (r_readB \& (r_reg_sel = (WORDN 15))) in
let new_R_ctr2_in = ((r_writeB \& (r_reg_sel = (WORDN 11))) => I_ad in \ R_ctr3_in) in
let new_R_ctr3_mux_sel = (r_cmds_wr23B \& (ELEMENT new_R_gcr (20)) \& R_ctr3_cry)) in
let new_R_ctr3_iden = (r_readB \& (r_reg_sel = (WORDN 11))) in
let new_R_ctr3 = (R_ctr3_mux_sel \& R_ctr3_in \ R_ctr3_new) in
let new_R_ctr3_new = ((R_ctr2_cry) => (INCN 31 R_ctr3) \& R_ctr3_new) in
let new_R_ctr3_cry = (R_ctr3_new \& (R_ctr3_cry)) in
let new_R_ctr3_out = ((R_ctr3_hid) \& R_ctr3_new \ R_ctr3_out) in
let new_R_ctr3_ordem = (r_readB \& (r_reg_sel = (WORDN 15))) in
let new_R_ctr3_load = (r_writeB \& (r_reg_sel = (WORDN 0)) \& (r_reg_sel = (WORDN 1)))) in
let new_R_ctr3_iden = ((r_writeB \& (r_reg_sel = (WORDN 0)) \& (r_reg_sel = (WORDN 1)))) => R_ctr3 \& R_ctr3_load) in
let new_R_ctr3_mask =
let new_R_icr =
  (R_icr_load =>
   (¬(r_reg_sel = (WORDN 1))) => (And rep (R_icr_old, R_icr_mask)) | (Or rep (R_icr_old, R_icr_mask)))
let new_R_icr_rden = (new_R fsm_state = RA) ∧ (r_reg_sel = (WORDN 0)) ∨ (r_reg_sel = (WORDN 1)) in
let sr28 = (ALTER ARBN (28) MB_parity) in
let sr28_25 = (ALTER sr28 (27,25) C_ss) in
let sr28_24 = (ALTER sr28_25 (24) CB_parity) in
let sr28_22 = (ALTER sr28_24 (23,22) ChannelID) in
let sr28_16 = (ALTER sr28_22 (21,16) Id) in
let sr28_12 = (ALTER sr28_16 (15,12) State) in
let sr28_9 = (ALTER sr28_12 (9) Pmm_fail) in
let sr28_8 = (ALTER sr28_9 (8) Pmu_fail) in
let sr28_2 = (ALTER sr28_8 (3,2) Reset_cpu) in
let sr28_0 = (ALTER sr28_2 (1,0) Cpu_fail) in
let new_R_sr = (r_readB => sr28_01 R sr) in
let new_R_sr_rden = (r_writeB ∧ (r_reg_sel = (WORDN 4))) in
let r_int0_en = (((ELEMENT R_icr (0)) ∧ (ELEMENT R_icr (8))) ∨
  ((ELEMENT R_icr (1)) ∧ (ELEMENT R_icr (9))) ∨
  ((ELEMENT R_icr (2)) ∧ (ELEMENT R_icr (10))) ∨
  ((ELEMENT R_icr (3)) ∧ (ELEMENT R_icr (11))) ∨
  ((ELEMENT R_icr (4)) ∧ (ELEMENT R_icr (12))) ∨
  ((ELEMENT R_icr (5)) ∧ (ELEMENT R_icr (13))) ∨
  ((ELEMENT R_icr (6)) ∧ (ELEMENT R_icr (14))) ∨
  ((ELEMENT R_icr (7)) ∧ (ELEMENT R_icr (15))) in
let new_R_int0_dis = r_int0_en in
let new_R_int3_dis = r_int3_en in
let new_R_busA_latch =
  (R_ctr0_inrden => R_ctr0_in1)
  (R_ctr0_ndern => R_ctr0_out1)
  (R_ctr1_inrden => R_ctr1_in1)
  (R_ctr1_ndern => R_ctr1_out1)
  (R_ctr2_inrden => R_ctr2_in1)
  (R_ctr2_ndern => R_ctr2_out1)
  (R_ctr3_inrden => R_ctr3_in1)
  (R_ctr3_ndern => R_ctr3_out1)
  (R_icr_rden => new_R_icr1)
  (R_ccr_rden => R_ccr1)
  (R_gcr_rden => R_gcr1)
  (R_sr_rden => R_sr1 ARB))))) in
let new_R fsm_ale_ = I_rale_ in
let new_R fsm_mrdy_ = I_mrdy_ in
let new_R fsm_last_ = I_last_ in
let new_R fsm_rst = Rst in

193
NEW_R_fsm_state, new_R_fsm_ale_, new_R_fsm_mrdy_, new_R_fsm_last_, new_R_fsm_rst, new_R_ctr0_in, new_R_ctr0_mux_sel, new_R_ctr0, new_R_ctr0_irden, new_R_ctr0_new, new_R_ctr0_cry, new_R_ctr0_out, new_R_ctr0_order, new_R_ctr1_in, new_R_ctr1_mux_sel, new_R_ctr1, new_R_ctr1_irden, new_R_ctr1_new, new_R_ctr1_cry, new_R_ctr1_out, new_R_ctr1_order, new_R_ctr2_in, new_R_ctr2_mux_sel, new_R_ctr2, new_R_ctr2_irden, new_R_ctr2_new, new_R_ctr2_cry, new_R_ctr2_out, new_R_ctr2_order, new_R_ctr3_in, new_R_ctr3_mux_sel, new_R_ctr3, new_R_ctr3_irden, new_R_ctr3_new, new_R_ctr3_cry, new_R_ctr3_out, new_R_ctr3_order, new_R_icr_load, new_R_icr_old, new_R_icr_mask, new_R_icr_irden, new_R_icr, new_R_ccr, new_R_ccr_irden, new_R_gcr, new_R_gcr_irden, new_R_sr, new_R_sr_irden, new_R_int0_dis, new_R_int0_dis, new_R_c01_cout_del, new_R_int1_en, new_R_c23_cout_del, new_R_int2_en, new_R_wr, new_R_cntlatch_del, new_R_srdy_del_, new_R_reg_sel, new_R_busA_latch)

%;

% Output definition for EXEC instruction.
---------------------------------------------------------------------%

let rEXEC_out_def = new_definition
('rEXEC_out',
"1 (rep :rep_ty)
(R_fsm_state :rfsm_ty)
(R_ctr0_in R_ctr0 R_ctr0_new R_ctr0_out R_ctr1_in R_ctr1_R_ctr1_new R_ctr1_out R_ctr2_in R_ctr2_new
R_ctr2_out R_ctr3_in R_ctr3_R_ctr3_new R_ctr3_out R_icr old R_icr_mask R_icr R_gcr R_sr R_reg sel
R_busA_latch :wordn)
(R_fsm_ale R_fsm_mrdy R_fsm_last R_fsm_rst R_ctr0_in R_ctr0_mux_sel R_ctr0_irden R_ctr0_cry R_ctr0_order
R_ctr1_mux_sel
R_ctr1_irden R_ctr1_cry R_ctr1 order R_ctr2_mux_sel R_ctr2_irden R_ctr2_cry R_ctr2_order R_ctr3_mux_sel
R_ctr3_irden R_ctr3_cry R_ctr3_order R_icr_load R_icr_irden R_gcr_irden R_sr_irden R_int0_dis
R_int3_dis R_c01_cout del R_int1_en R_c23_cout del R_int2_en R_wr R_cntlatch del R_srdy del_ :bool)
(I_ad_in I_be Cpu_fail Reset_cpu S_state Id ChannelID C_ss :wordn)
(CIkA Rst I_rale I_last I_mrdy_ Disable_int Disable_writes Piu fail Pmm fail CB parity MB_parity :bool) .

rEXEC_out rep
(R_fsm_state, R_fsm_ale_, R_fsm_mrdy_, R_fsm_last_, R_fsm_rst, R_ctr0_in, R_ctr0_mux_sel, R_ctr0,
R_ctr0_irden, R_ctr0_cry, R_ctr0_out, R_ctr0_order, R_ctr1_in, R_ctr1_mux_sel, R_ctr1,
R_ctr1_irden, R_ctr1_cry, R_ctr1_order, R_ctr2_in, R_ctr2_mux_sel, R_ctr2,
R_ctr2_irden, R_ctr2_cry, R_ctr2_order, R_ctr3_in, R_ctr3_mux_sel, R_ctr3,
R_ctr3_irden, R_ctr3_cry, R_ctr3_order, R_icr_load, R_icr_old,
R_icr_mask, R_icr_irden, R_gcr, R_gcr_irden, R_sr, R_sr_irden, R_int0_dis,
R_int3_dis, R_c01_cout_del, R_int1_en, R_c23_cout_del, R_int2_en, R_wr, R_cntlatch_del, R_srdy_del_,
R_reg_sel, R_busA_latch)
(CIkA, Rst, I_ad_in, I_rale, I_last, I_be, I_mrdy, Disable_int Disable_writes, Cpu_fail, Reset_cpu, Piu_fail, Pmm_fail, S_state, Id, ChannelID, CB_parity, MB_parity, C_ss) =

let new_R_fsm_state =
((R_fsm_rst) => RI1
((R_fsm_state = RI) => (-R_fsm_ale_ => RA | RI1)
((R_fsm_state = RA) => (-R_fsm_mrdy_ => RD | RA1)
((R_fsm_last_) => RJ | RA))) in
let r_fsm_cntlatch = ((R_fsm_state = RI1) \ -R_fsm_ale_1 in
let r fsm_srdy_ = -(R_fsm_state = RA) \ -R_fsm_mrdy_ in
let new_R_wr = (I_rale => (ELEMENT I_ad_in (27)) | R_wr) in

194
let new_R_ctr2_new = (((ELEMENT new_R_gcr (23))) => (INCN 31 R_ctr2) \ R ctr2) in
let new_R_ctr2_cry = ((ONES 31 R_ctr2) \ (ELEMENT new_R_gcr (23))) in
let new_R_ctr2_out = ((r fsm_cntlatch) => R ctr2_new \ R ctr2_out) in
let new_R_ctr2_command = (r readB \ (r reg sel = (WORDN 14))) in
let new_R_ctr3_in = ((r writeB \ (r reg sel = (WORDN 11))) => I ad in \ R ctr3 in) in
let new_R_ctr3 mux sel = ((r cir wr23B => (ELEMENT new_R_gcr (23))) => INCN 31 R ctr2) in
let new_R_ctr3_orden = ((r readB \ (r reg sel = (WORDN 15))) => I R ctr3 new) in
let new_R_iicr_load = ((r writeB \ (r reg sel = (WORDN 0))) \ (r reg sel = (WORDN 1))) in
let new_R_iicr_mask = ((r writeB \ (r reg sel = (WORDN 0))) \ (r reg sel = (WORDN 1))) => I ad in \ R iicr_mask) in
let new_R_iicr = ((r readB \ (r reg sel = (WORDN 1))) => (Andn rep (R iicr old, R iicr_mask)) \ (Orn rep (R iicr old, R iicr_mask)) \ R iicr) in
let new_R_iicr_ren = (new R fsm_state = RA) \ (r reg sel = (WORDN 0)) \ (r reg sel = (WORDN 1))) in
let sr28 = (ALTER ARBN (28) MB parity) in
let sr28_25 = (MALTER sr28 (27,25) C sel) in
let sr28_24 = (ALTER sr28 (24) CB parity) in
let sr28_22 = (MALTER sr28 (23,22) ChannelID) in
let sr28_16 = (MALTER sr28 (21,16) Id) in
let sr28_12 = (MALTER sr28 (15,12) S state) in
let sr28_9 = (ALTER sr28 (9) Pnm fail) in
let sr28_8 = (ALTER sr28 (8) Piu fail) in
let sr28_2 = (MALTER sr28 (3,2) Reset cpu) in
let sr28_0 = (MALTER sr28 (1,0) Cpu fail) in
let new_R_sr = ((r fsm_cntlatch) => sr28_0 \ R sr) in
let new_R_ssr_ren = (r readB \ (r reg sel = (WORDN 4))) in
let r int0_en = (((ELEMENT R icr (0)) \ (ELEMENT R icr (8))) \ V (ELEMENT R icr (1)) \ (ELEMENT R icr (9))) \ V (ELEMENT R icr (2)) \ (ELEMENT R icr (10))) \ V (ELEMENT R icr (3)) \ (ELEMENT R icr (11))) \ V (ELEMENT R icr (4)) \ (ELEMENT R icr (12))) \ V (ELEMENT R icr (5)) \ (ELEMENT R icr (13))) \ V (ELEMENT R icr (6)) \ (ELEMENT R icr (14))) \ V (ELEMENT R icr (7)) \ (ELEMENT R icr (15))) \ V
let new_R_int0_dis = r int0_en in
let r int3_en = (((ELEMENT R icr (16)) \ (ELEMENT R icr (24))) \ V (ELEMENT R icr (17)) \ (ELEMENT R icr (25))) \ V (ELEMENT R icr (18)) \ (ELEMENT R icr (26))) \ V (ELEMENT R icr (19)) \ (ELEMENT R icr (27))) \ V (ELEMENT R icr (20)) \ (ELEMENT R icr (28))) \ V (ELEMENT R icr (21)) \ (ELEMENT R icr (29))) \ V (ELEMENT R icr (22)) \ (ELEMENT R icr (30))) \ V (ELEMENT R icr (23)) \ (ELEMENT R icr (31))) in
let new_R_int3_dis = r int3_en in
let new_R_busA_latch = (r ctrl0_ren) => R ctrl0 in
let new_R_fsm_ale = I_ale in
let new_R_fsm_mrdy = I_mrdy in
let new_R_fsm_last = I_last in
let new_R_fsm_rst = Rst in

let Lad_out = ((~R_wr A ((new_R_fsm_state = RA) V (new_R_fsm_state = RD))) => new_R_busA_latch | ARBN) in
let Lsrdy =
    (((new_R_fsm_state = RA) V (new_R_fsm_state = RD)) => -((R_fsm_state = RA) A (new_R_fsm = RD)) | ARB) in
let IntO_ = -(r_int0_en A Rint0dis A Disable_int) in
let Intl = (R_ctrl_cry A new_R_intl_en A Disable_int) in
let Int2 = (R_ctr3_cry A new_R_int2_en A Disable_int) in
let Int3_ = -(r_int3__en A RinG_dis A Disable_int) in
let Ccr = R_ccr in
let Led = (SUBARRAY new_R_gcr (3,0)) in
let Reset_error = (ELEMENT new_R_gcr (24)) in
let Pmm_invalid = (ELEMENT new_R_gcr (28)) in

(Lad_out, Lsrdy, IntO_, Intl, Int2, Int3, Ccr, Led, Reset_error, Pmm_invalid)
This file contains the ml source for the clock-level specification of the C-Port of the FTEP PIU, an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center. The bulk of this code was translated from an M-language simulation program using a translator written by P.J. Windley at the University of Idaho.
let cc_env = "((I_ad_in, I_be_in, I_mrdy_in, I_rale_in, I_male_in, I_last_in, I_ndry_in, 
  I_lock, I_cale, I_hlda, I_crqt, 
  CB_rqt_in, CB_ad_in, CB_ms_in, CB_ss_in, 
  Rst, ClikA, ClikB, ClikD, Id, ChannelID, Pnm_failure, Pnu_invalid, Ccr, 
  Reset_error) 
  :cc_env_ty)";;

let cc_out_ty = ":(bool#bool#bool#bool#bool#bool#bool#wordn#wordn#
  bool#wordn#wordn#wordn#wordn#bool#bool)";;

let cc_out = "((I_cgnt, I_mrdy_out, I_hold, I_rale_out, I_male_out, I_last_out, I_ndry_out, 
  I_ad_out, I_be_out, 
  CB_rqt_out, CB_ms_out, CB_ss_out, CB_ad_out, C_ss_out, Disable_writes, CB_parity) 
  :cc_out_ty)";;

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

let cEXEC_def = new_definition ('cEXEC_inst', 
  "! (rep:rep_ty)
  (C_mfsm_state:cmfsm_ty) (C_sfsm_state:csfsm_ty) (C_efsm_state:cefsm_ty) 
  (C_mfsm_ss C_sfsm_ms C_sizewrbe C_ss C_source C_data_in C_iad_out C_iad_in C_a1a0 C_a3a2 :wordn) 
  (C_mfsm_D C_sfsm_rst C_efsm_cqtr C_mfsm_hold. C_mfsm_inval C_sfm Дм C_smf_rst C_sfsm_hlda_ 
  C_efsm_cale C_efsm_last C_efsm_male C_efsmקולנוע C_efsm_srdy C_efsm rst 
  C_wr C_clkA C_last_in C_lock_in C_last_out C_hold C_holdA C_out_0_le del C_cin_2_le 
  C_mrdy del C_iad en s del C_iad en s delA C_wrzy C_crqt C parity :bool) 
  (I_ad in I_be in CB_rqt in CB_ad in CB_ms in CB_ss in Id ChannelID Ccr :wordn) 
  (I_mrdy in I_rale in I_male in I_last in I_ndry in I_lock I_cale I_hlda I_crqt_ 
  Rst ClikA ClikB ClikD Id ChannelID Pnm_failure Pnu_invalid Reset error :bool) . 
  cEXEC_inst rep 
  (C_mfsm_state, C_mfsm_D, C_mfsm_rst, C_mfsm_cqtr, C_mfsm_hold, C_mfsm_inval, 
  C_sfsm_state, C_sfsm_D, C_sfsm_rst, C_sfsm_hlda, C_sfsm_ms, 
  C_efsm_state, C_efsm_cale, C_efsm_last, C_efsm_male, C_efsm_rale, C_efsm_srdy, C_efsm rst, 
  C_wr, C_sizewrbe, C_clkA, C_last_in, C_lock_in, C_ss, C_last_out, 
  C_hold, C_holdA, C_out_0_le del, C_cin_2_le, C_mrdy del, C_iad en s del, C_iad en s delA, 
  C_wrzy, C_rsdry, C.pageY, C source, C data_in, C_iad out, C_iad in, C_a1a0, C_a3a2) 
  (I_ad in, I_be in, I_mrdy in, I_rale in, I_male in, I_last in, I_ndry in, 
  I_lock, I_cale, I_hlda, I_crqt, CB_rqt in, CB_ad in, CB_ms in, CB_ss in, 
  Rst, ClikA, ClikB, ClikD, Id, ChannelID, Pnm_failure, Pnu invalid, Ccr, Reset error) = 
  
  let c_write = (((-C_mfsm_state = CMI)) \ (-C_mfsm_state = CMR))) \ C_wr 1 (ELEMENT C_sizewrbe (5)) in 
  let c_busy = ((SUBARRAY CB_rqt_in (3,1)) = (WORDN 7)) in 
  let c_grant = (((SUBARRAY Id (1,0)) = (WORDN 0)) \ (-ELEMENT CB_rqt_in_ (0))) 
  \ V ((SUBARRAY Id (1,0)) = (WORDN 1)) \ (-ELEMENT CB_rqt_in_ (0)) 
  \ (ELEMENT CB_rqt_in_ (1)) 
  \ V ((SUBARRAY Id (1,0)) = (WORDN 2)) \ (-ELEMENT CB_rqt_in_ (0)) 
  \ (ELEMENT CB_rqt_in_ (1)) 
  \ (ELEMENT CB_rqt_in_ (2)) 
  \ V ((SUBARRAY Id (1,0)) = (WORDN 3)) \ (-ELEMENT CB_rqt_in_ (0)) 
  \ (ELEMENT CB_rqt_in_ (1))";

199
let c_addressed = (Id = (SUBARRAY C_source (15,10))) in

let c_mfsm_stateA =
((C_mfsm_rst) => CMI)
((C_mfsm_state = CMI) =>
  (C_mfsm_D \& \& C_mfsm_crgnt \& c_busy \& \& C_mfsm_invalid) => CMR \& CMI)
((C_mfsm_state = CMR) \&\& (C_mfsm_D \&\& c_grant \&\& C_mfsm_hold) => CMA3 \& CMR)
((C_mfsm_state = CMA3) \&\& ((C_mfsm_D) => CMA1 \& CMA3))
((C_mfsm_state = CMA1) =>
  (C_mfsm_D \&\& (C_mfsm_ss = ^SRDY)) => CMA0 \&
  (C_mfsm_D \&\& (C_mfsm_ss = ^SABORT)) => CMABT \& CMA1)
((C_mfsm_state = CMA0) =>
  (C_mfsm_D \&\& (C_mfsm_ss = ^SRDY)) => CMA2 \&
  (C_mfsm_D \&\& (C_mfsm_ss = ^SABORT)) => CMABT \& CMA0)
((C_mfsm_state = CMD1) =>
  (C_mfsm_D \&\& (C_mfsm_ss = ^SRDY)) => CMD0 \&
  (C_mfsm_D \&\& (C_mfsm_ss = ^SABORT)) => CMABT \& CMD1)
((C_mfsm_state = CMD0) =>
  (C_mfsm_D \&\& (C_mfsm_ss = ^SRDY) \&\& C_last_in_) => CMD1 \&
  (C_mfsm_D \&\& (C_mfsm_ss = ^SABORT) \&\& C_last_in_) => CMW \&
  (C_mfsm_D \&\& (C_mfsm_ss = ^SABORT)) => CMABT \& CMD1)
((C_mfsm_state = CMW) =>
  (C_mfsm_D \&\& (C_mfsm_ss = ^SABORT)) => CMABT \&
  (C_mfsm_D \&\& (C_mfsm_ss = ^SACK) \&\& C_lock_in_) => CMI \&
  ((C_last_in_) => (CMI \& CMABT)))
)

let c_sfsm_stateA =
((C_sfsm_rst) => CSI)
((C_sfsm_state = CSI) =>
  ((C_sfsm_D \&\& (C_sfsm_ms = ^MSTART) \&\& c_grant \&\& c_addressed) => CSA1 \& CSI))
((C_sfsm_state = CSL) =>
  ((C_sfsm_D \&\& (C_sfsm_ms = ^MSTART) \&\& c_grant \&\& c_addressed) => CSA1)
  (C_sfsm_D \&\& (C_sfsm_ms = ^MSTART) \&\& \& c_grant \&\& \& c_addressed) => CSI)
  ((C_sfsm_D \&\& (C_sfsm_ms = ^MABORT)) => CSABT \& CSL))
((C_sfsm_state = CSA1) =>
  ((C_sfsm_D \&\& (C_sfsm_ms = ^MRDY)) => CSA0)
  (C_sfsm_D \&\& (C_sfsm_ms = ^MABORT)) => CSABT \& CSA1))
((C_sfsm_state = CSA0) =>
  ((C_sfsm_D \&\& (C_sfsm_ms = ^MRDY) \&\& \& c_sfsm_hlda) => CSALE)
  (C_sfsm_D \&\& (C_sfsm_ms = ^MRDY) \&\& \& c_sfsm_hlda) => CSAW)
  (C_sfsm_D \&\& (C_sfsm_ms = ^MABORT)) => CSABT \& CSA0)
((C_sfsm_state = CSAW) =>
  ((C_sfsm_D \&\& (C_sfsm_ms = ^MRDY) \&\& \& c_sfsm_hlda) => CSALE)
  (C_sfsm_D \&\& (C_sfsm_ms = ^MABORT)) => CSABT \& CSAW)
((C_sfsm_state = CSALE) =>
  ((C_sfsm_D \&\& c_write \&\& (C_sfsm_ms = ^MRDY)) => CSD1)
  (C_sfsm_D \&\& c_write \&\& (C_sfsm_ms = ^MRDY)) => CSRR)
  (C_sfsm_D \&\& (C_sfsm_ms = ^MABORT)) => CSABT \& CSALE)
((C_sfsm_state = CSRR) =>
  ((C_sfsm_D \&\& \& c_write \&\& (C_sfsm_ms = ^MABORT)) => CSD1)
  (C_sfsm_D \&\& \& c_write \&\& (C_sfsm_ms = ^MABORT)) => CSRR)
(\text{c\_sfsm\_D} \land (\text{c\_sfsm\_ms} = \text{\textasciitilde MABORT})) \Rightarrow \text{CSABT} \lor \text{CSRR}) \\
(\text{c\_sfsm\_state} = \text{CSD1}) \Rightarrow \\
((\text{c\_sfsm\_D} \land (\text{c\_sfsm\_ms} = \text{\textasciitilde MRDY})) \Rightarrow \text{CSDO}) \\
(\text{c\_sfsm\_D} \land (\text{c\_sfsm\_ms} = \text{\textasciitilde MABORT})) \Rightarrow \text{CSABT} \lor \text{CSD1}) \\
(\text{c\_sfsm\_state} = \text{CSD0}) \Rightarrow \\
((\text{c\_sfsm\_D} \land (\text{c\_sfsm\_ms} = \text{\textasciitilde MEND})) \Rightarrow \text{CSACK}) \\
(\text{c\_sfsm\_D} \land (\text{c\_sfsm\_ms} = \text{\textasciitilde MRDY})) \Rightarrow \text{CSD1} \\
(\text{c\_sfsm\_D} \land (\text{c\_sfsm\_ms} = \text{\textasciitilde MABORT})) \Rightarrow \text{CSABT} \lor \text{CSD0}) \\
(\text{c\_sfsm\_state} = \text{CSACK}) \Rightarrow \\
((\text{c\_sfsm\_D} \land (\text{c\_sfsm\_ms} = \text{\textasciitilde MRDY})) \Rightarrow \text{CSL}) \\
(\text{c\_sfsm\_D} \land (\text{c\_sfsm\_ms} = \text{\textasciitilde MWAIT})) \Rightarrow \text{CSI} \\
(\text{c\_sfsm\_D} \land (\text{c\_sfsm\_ms} = \text{\textasciitilde MABORT})) \Rightarrow \text{CSABT} \lor \text{CSACK}) \\
(\text{c\_sfsm\_D}) \Rightarrow \text{CSI} \lor \text{CSABT})$

let \text{c\_efsm\_stateA} = \\
((\text{c\_efsm\_rst}) \Rightarrow \text{CEI}) \\
(\text{c\_efsm\_state} = \text{CEI}) \Rightarrow ((\text{\textasciitilde c\_efsm\_cal_e}) \Rightarrow \text{CEE} \lor \text{CEI}) \\
(((\text{c\_efsm\_last}) \land \text{c\_efsm\_mdry}) \lor (\text{c\_efsm\_cale} \land \text{c\_efsm\_rale}) \Rightarrow \text{CEI} \lor \text{CEE})$

let c\_srdy\_en = ((\text{c\_efsm\_stateA} = \text{CEE}) \lor (\text{c\_efsm\_state} = \text{CEE})) \Rightarrow \\
((\text{c\_mfsf\_stateA} = \text{CMA3}) \lor (\text{c\_mfsf\_stateA} = \text{CMA1}) \\
\lor (\text{c\_mfsf\_stateA} = \text{CMD1}))$

let cout\_sel\_0 = (\text{ALTER ARBN} (0)) (((\text{c\_sfsm\_stateA} = \text{CSD1}) \lor (\text{c\_sfsm\_stateA} = \text{CSDO})) \Rightarrow \\
((\text{c\_sfsm\_stateA} = \text{CSD1}) \lor \\
(\text{c\_mfsf\_stateA} = \text{CMA3}) \lor (\text{c\_mfsf\_stateA} = \text{CMA1}) \\
\lor (\text{c\_mfsf\_stateA} = \text{CMD1})))$

let c\_cout\_sel = cout\_sel\_0 in
let new\_C\_wr = ((\text{\textasciitilde c\_ms\_in}) \Rightarrow (\text{ELEMENT L\_d\_in (27)}) \lor \text{C\_wr}) in
let new\_C\_sizewrbe = ((\text{Rst}) \Rightarrow (\text{WORDN O}) \lor \\
(((\text{c\_mfsf\_stateA} = \text{CSA0}) \land \text{C\_clkD}) \Rightarrow (\text{SUBARRAY C\_data\_in (31,22)}) \lor \text{C\_sizewrbe})) in
let c\_new\_write = (((\text{\textasciitilde c\_mfsf\_stateA} = \text{CMA1}) \land (\text{\textasciitilde c\_mfsf\_stateA} = \text{CMR}))) \Rightarrow \\
\text{new\_C\_wr} \lor (\text{ELEMENT new\_C\_sizewrbe (5)})$

let new\_C\_clkA = \text{C\_clkD} in
let new\_C\_last\_in = ((\text{Rst}) \Rightarrow F \lor \\
((\text{c\_mfsf\_stateA} = \text{CMABT}) \lor (\text{c\_mfsf\_stateA} = \text{CMD1}) \land \text{C\_clkD}) \Rightarrow \text{I\_last\_in_1} \lor \\
\text{C\_last\_in_1}) in
let new\_C\_lock\_in = ((\text{Rst}) \Rightarrow F \lor \\
((\text{c\_mfsf\_stateA} = \text{CMA1}) \Rightarrow \text{I\_lock_1} \lor \\
\text{C\_lock\_in_1}) in
let new\_C\_ss = (((\text{\textasciitilde c\_mfsf\_stateA} = \text{CMABT}) \land (\text{\textasciitilde c\_mfsf\_stateA} = \text{CMI}))) \Rightarrow \text{CB\_ss\_in_1} \lor \text{C\_ss}) in
let c\_mend = (\text{CB\_ms\_in} = \text{\textasciitilde MEND}) in
let c\_mabort = (\text{CB\_ms\_in} = \text{\textasciitilde MABORT}) in
let new\_C\_last\_out = 
((((\text{c\_mfsf\_stateA} = \text{CSA1}) \land (\text{\textasciitilde C\_clkD} \land (\text{c\_mend} \lor \text{c\_mabort})))) \Rightarrow \text{T} \lor 
(((\text{c\_mfsf\_stateA} = \text{CSA1}) \land (\text{\textasciitilde C\_clkD} \land (\text{c\_mend} \lor \text{c\_mabort})))) \Rightarrow \text{F} \lor 
(((\text{c\_mfsf\_stateA} = \text{CSA1}) \land (\text{\textasciitilde C\_clkD} \land (\text{c\_mend} \lor \text{c\_mabort})))) \Rightarrow \text{C\_last\_out_1 ARBN})$ in
let c\_srdy\_en = (\text{\textasciitilde C\_ss\_in} \Rightarrow \text{\textasciitilde SRDY}) in
let c\_dfsm\_master = ((\text{c\_mfsf\_stateA} = \text{CMA3}) \lor (\text{c\_mfsf\_stateA} = \text{CMA2}) \lor (\text{c\_mfsf\_stateA} = \text{CMA1}) \\
\lor (\text{c\_mfsf\_stateA} = \text{CMD0}) \lor (\text{c\_mfsf\_stateA} = \text{CMA0}) \lor (\text{c\_mfsf\_stateA} = \text{CMRI})) in
let c\_dfsm\_cad\_en = (\text{\textasciitilde c\_mfsf\_stateA} = \text{CMA3}) \lor (\text{c\_mfsf\_stateA} = \text{CMA2}) \\
\lor (\text{\textasciitilde c\_new\_write} \land (\text{\textasciitilde c\_mfsf\_stateA} = \text{CMD1}) \lor (\text{c\_mfsf\_stateA} = \text{CMA0}) \\
\lor (\text{\textasciitilde c\_new\_write} \land (\text{\textasciitilde c\_mfsf\_stateA} = \text{CMD0})) \\
\lor (\text{\textasciitilde c\_new\_write} \land (\text{\textasciitilde c\_mfsf\_stateA} = \text{CSD1}) \lor (\text{c\_mfsf\_stateA} = \text{CSD0})))$ in
let new\_C\_hold = (\text{c\_mfsf\_stateA} = \text{CSI}) in
let new_C_holdA_ = ((ClkD) => C_holdA_ in
let new_C_cout_0_le_del = (I_called) V (I_srdy_in \land \neg c_new_write)
  V ((c_mfsm_stateA = CMA0) \land c_srdy \land c_new_write \land ClkD)
  V ((c_mfsm_stateA = CMD0) \land c_new_write \land c_srdy \land ClkD) in
let new_C_cin_2_le = (ClkD \land ((c_mfsm_stateA = CMD0) \land c_srdy \land \neg c_new_write) V
  ((c_mfsm_stateA = CSA0)) \land
  ((c_sfm_stateA = CSD0) \land \neg c_new_write) in
let new_C_srdy_del_ = -(c_new_write \land ClkD) \land (c_sfm_stateA = CSALE) V (c_sfm_stateA = CSD1)) V
  (\neg c_new_write \land C clkA \land (c_sfm_stateA = CSAK)) \land
  (\neg c_new_write \land ClkD \land (c_sfm_stateA = CSD0)) in
let new_C_cout_0_le_del = ((c_sfm_stateA = CMA0) \land \neg c_new_write) V
  ((c_sfm_stateA = CMD0) \land \neg c_new_write \land c_mfsm_stateA = CMA0) V
  ((c_sfm_stateA = CMD1) \land (c_mfsm_stateA = CMA1)) V
  (\neg c_new_write \land ClkD \land (c_mfsm_stateA = CMD0)) V
  (c_mfsm_stateA = CMA1) \land (c_mfsm_stateA = CMA2) \land
  (c_mfsm_stateA = CMD0) \land (c_mfsm_stateA = CMD1) in
let c_sparity = ((\neg (c_sfm_stateA = CSM)) \land (\neg (c_sfm_stateA = CSM)) 
  \land (\neg (c_sfm_stateA = CSABT))) in
let c_pe_cnt = (ClkD \land ((\neg c_marity = c_sparity)) \land (SUBARRAY CB_se_in (1,0)) = (WORDN 0))) in
let new_C_parity = (((ClkD \land c_pe \land c_pe_cnt) \land \neg Reset_error) => T I
  ((\neg ClkD \land c_pe \land c_pe_cnt) \land Reset_error) => F I
  ((\neg ClkD \land c_pe \land c_pe_cnt) \land \neg Reset_error) => C_parity \land ARB))) in
let new_C_source =
  ((Rst) => (WORDN 0)) \land
  (ClkD \land ((c_marity = c_sparity)) \land (SUBARRAY CB_se_in (1,0)) = (WORDN 0))) in
let data_in31_16 = (MALTER ARBN (31,16) ((Rst) => (WORDN 0)) \land
  (ClkD \land ((c_marity = c_sparity)) \land (SUBARRAY C_data_in (31,16)))) in
let data_in31_0 = (MALTER data_in31_16 (15,0) ((Rst) => (WORDN 0)) \land
  ((new_C_cin_2_le) => Par_Dec rep (CB_ad_in) \land C_source))) in
let new_C_data_in = data_in31_0 in
let new_C_iad_out = (C_cin_2_le) \land C_data_in \land C_iad_out in
let new_C_iad_in = ((new_C_cout_0_le_del) \land \neg C_ad_in \land \neg C_iad_in) in
let new_C_a1a0 = (((c_dfsm_master \land C_cout_0_le_del) \land
  (\neg c_dfsm_master \land C clkA \land (c_sfm_stateA = CSD1))) => C_iad_in \land C_a1a0) in
let new_C_a3a2 = (c_mfsm_stateA = CMR) => C_or (C_a3a2) in
let new_C_mfsm_state = c_sfm_stateA in
let new_C_mfsm_D = ClkD in
let new_C_mfsm_rst = Rst in
let new_C_mfsm_cqpt_ = I_cqpt_ in
let new_C_mfsm_hold_ = new_C_holdA_ in

202
let new_C mfsm_ss = CB_ss_in in
let new_C mfsm_invalid = Pinlvalid in
let new_C sfsm state = c_sfsm_stateA in
let new C sfsm_D = CIkD in
let new_C sfsm_rst = Rst in
let new_C sfsm_hlda_ = Lhldt in
let new_C sfsm_ms = CB_ms_in in
let new_C efsm cale = I cale_in in
let new_C efsm_last_ = I_last_in_ in
let new_C efsm_male = I_male_in_ in
let new_C efsm_rale_ = I_rale_in_ in
let new_C efsm_srdy_ = Isrdy_in_ in
let new_C efsm_rst = Rst in

(C mfsm_state, C mfsm_D, C mfsm_rst, C mfsm_hold_, C mfsm_crqt_, C mfsm_sizewrbe, C mfsm_ss, C mfsm_source, C data_in, C iad_in, C iad_out, C iad_outA, C a0, C a3a2)

let cEXEC out def = new_definition
('cEXEC_out',
"/ (rep:rep_ty) (C mfsm_state:cmfsm_ty) (C efsm_state:cefsam_ty)
(C mfsm_ss C mfsm_ms C sizewrbe C ss C source C data_in C iad_out C iad_in C a10 C a3a2 :wordn)
(C mfsm_D C mfsm_rst C mfsm_crqt_ C mfsm_hold_ C mfsm_invalid C sfsm_state C efsm_state C efsm_crqt_ C efsm_hold_ C efsm_sizewrbe C efsm_mono
C efsm_male_ C efsm_rale_ C efsm_srdy_ C efsm_rst
C wr C clkA C last_in_ C lock_in_ C last_out_ C hold_ C holdA_ C cout_0_le_del_ C cin_2_le C mrdy_del_ C iad_en_s_del
C iad_en_s_delA C wrdy_ C rrdy_ C parity_ C source_ C data_in_ C iad_out_ C iad_in_ C a10 C a3a2)

let c_write = (\!(\!(C mfsm_state = CMI) \&\& (C mfsm_state = CMR))) => C wr \! (ELEMENT C sizewrbe (5)) in
let c_busy = (\!(\!(SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
let c_grant = (\!(\!(SUBARRAY Id (1,0)) = (WORDN 0)) \&\& (ELEMENT CB_rqt_in_ (0)))
V (\!(\!(SUBARRAY Id (1,0)) = (WORDN 1)) \&\& (ELEMENT CB_rqt_in_ (0)))
\& (ELEMENT CB_rqt_in_ (1)))
V (\!(\!(SUBARRAY Id (1,0)) = (WORDN 2)) \&\& (ELEMENT CB_rqt_in_ (0)))
\[(\text{ELEMENT } \text{CB}_\text{rqt}_\text{in}_\text{(1)}) \land (\text{ELEMENT } \text{CB}_\text{rqt}_\text{in}_\text{(2)}) \land ((\text{SUBARRAY } \text{Id}(1,0)) = (\text{WORDN } 3) \land (\text{ELEMENT } \text{CB}_\text{rqt}_\text{in}_\text{(0)}) \land (\text{ELEMENT } \text{CB}_\text{rqt}_\text{in}_\text{(1)}) \land (\text{ELEMENT } \text{CB}_\text{rqt}_\text{in}_\text{(2)}) \land (\text{ELEMENT } \text{CB}_\text{rqt}_\text{in}_\text{(3)})) \)

let \(c\_\text{addressed} = (\text{Id} = (\text{SUBARRAY } \text{C}_\text{source}(15,10)))\) in

let \(c\_\text{msfm}_\text{state}A =\)

\[
((\text{C}_\text{msfm}_\text{rst}) \Rightarrow \text{CMI}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMI}) \Rightarrow
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{crqt} \land \neg \text{c}_\text{busy} \land \neg \text{C}_\text{msfm}_\text{invalid}) \Rightarrow \text{CMR} \land \text{CMI}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMR}) \Rightarrow (\text{C}_\text{msfm}_\text{D} \land \text{c}_\text{grant} \land \text{C}_\text{msfm}_\text{hold} ) \Rightarrow \text{CMA3} \land \text{CMR}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMA3}) \Rightarrow ((\text{C}_\text{msfm}_\text{D}) \Rightarrow \text{CMA1} \land \text{CMA3}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMA1}) \Rightarrow
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SRDY})) \Rightarrow \text{CMA0}) \land
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SABORT})) \Rightarrow \text{CMA0} \land \text{CMA1}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMA0}) \Rightarrow
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SRDY})) \Rightarrow \text{CMA2}) \land
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SABORT})) \Rightarrow \text{CMA2} \land \text{CMA1}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMA2}) \Rightarrow
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SRDY})) \Rightarrow \text{CMD1}) \land
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SABORT})) \Rightarrow \text{CMD1} \land \text{CMA2}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMD1}) \Rightarrow
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SRDY})) \Rightarrow \text{CMD0}) \land
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SABORT})) \Rightarrow \text{CMD0} \land \text{CMD1}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMD0}) \Rightarrow
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SRDY}) \land \text{c}_\text{last}_\text{in} ) \Rightarrow \text{CMD1}) \land
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SRDY}) \land \neg \text{c}_\text{last}_\text{in} ) \Rightarrow \text{CMW}) \land
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SABORT})) \Rightarrow \text{CMD0} \land \text{CMD1}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMW}) \Rightarrow
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SABORT})) \Rightarrow \text{CMA0} \land \text{CMD0}) \land
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SACK}) \land \text{C}_\text{lock}_\text{in} ) \Rightarrow \text{CMI}) \land
((\text{C}_\text{msfm}_\text{state} = \text{CMW}) \Rightarrow
(\text{C}_\text{msfm}_\text{D} \land (\text{C}_\text{msfm}_\text{ss} = \text{SABORT}) \land \text{c}_\text{grant} \land \text{c}_\text{addressed}) \Rightarrow \text{CMA3} \land \text{CMW} \land
((\neg \text{c}_\text{last}_\text{in} ) \Rightarrow \text{CMI} \land \text{CMW}))))) in

let \(c\_\text{sfsm}_\text{state}A =\)

\[
((\text{C}_\text{sfsm}_\text{rst}) \Rightarrow \text{CSI}) \land
(\text{C}_\text{sfsm}_\text{state} = \text{CSI}) \Rightarrow
((\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MSTART}) \land \neg \text{c}_\text{grant} \land \text{c}_\text{addressed}) \Rightarrow \text{CSA1} \land \text{CSI}) \land
(\text{C}_\text{sfsm}_\text{state} = \text{CSL}) \Rightarrow
((\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MSTART}) \land \neg \text{c}_\text{grant} \land \text{c}_\text{addressed}) \Rightarrow \text{CSA1}) \land
(\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MSTART}) \land \neg \text{c}_\text{grant} \land \neg \text{c}_\text{addressed}) \Rightarrow \text{CSI}) \land
(\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MABORT}) \Rightarrow \text{CSABT} \land \text{CSL}) \land
(\text{C}_\text{sfsm}_\text{state} = \text{CSA1}) \Rightarrow
((\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MRDY}) \land \neg \text{c}_\text{grant} \land \text{c}_\text{addressed}) \Rightarrow \text{CSA0}) \land
(\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MABORT}) \Rightarrow \text{CSABT} \land \text{CSA1}) \land
(\text{C}_\text{sfsm}_\text{state} = \text{CSA0}) \Rightarrow
((\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MRDY}) \land \neg \text{C}_\text{sfsm}_\text{hlda} ) \Rightarrow \text{CSALE}) \land
(\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MRDY}) \land \text{C}_\text{sfsm}_\text{hlda} ) \Rightarrow \text{CSA0W}) \land
(\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MABORT}) \Rightarrow \text{CSABT} \land \text{CSA0}) \land
(\text{C}_\text{sfsm}_\text{state} = \text{CSA0W}) \Rightarrow
((\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MRDY}) \land \neg \text{C}_\text{sfsm}_\text{hlda} ) \Rightarrow \text{CSALE}) \land
(\text{C}_\text{sfsm}_\text{D} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MABORT}) \Rightarrow \text{CSABT} \land \text{CSA0W}) \land
(\text{C}_\text{sfsm}_\text{state} = \text{CSALE}) \Rightarrow
((\text{C}_\text{sfsm}_\text{D} \land \text{c}_\text{write} \land (\text{C}_\text{sfsm}_\text{ms} = \text{MRDY}) \Rightarrow \text{CSD1}) \land
204
let c_efsm_stateA =
(C_efsm_rst) => CEI
(C_efsm_state = CEI) => (~C_efsm_cale_) => CEE (C Efsm)
let c_srdy_en = ((c Efsm_stateA = CEI) (C efsm_state = CEI)) in
let cout_sel0 = (ALTERN ARBN 0) ((c Efsm_stateA = CSD1) (C Efsm_stateA = CSD0)) =>
(c Efsm_stateA = CSD1)
(c Efsm_stateA = CMA3) (C Efsm_stateA = CMA1)
(c Efsm_stateA = CSD0)
let cout_sel10 = (ALTERN cout_sel0 1) ((c Efsm_stateA = CSD1) (C Efsm_stateA = CSD0)) =>
F
(c Efsm_stateA = CMA3) (C Efsm_stateA = CMA2)) in
let c_cout_sel = cout sel10 in
let new_C_wr = ((~C cale) => (ELEMENT I ad_in (27)) \ C wr) in
let new_C_size wrbe = ((Rst) => (WORDN 0) \ ((c Efsm_stateA = CSA0) \ C clk) \ C)
let c new write = (((c Efsm_stateA = CM2) (C Efsm_stateA = CMM)) =>
new_C wr (ELEMENT new_C size wrbe (5))) in
let new_C clk = ClkD in
let new_C last in_ = ((Rst) => F \ ((c Efsm stateA = CM2) \ C lock_in_)) in
let new_C lock_in_ = ((Rst) => F \ ((c Efsm stateA = CMA1) \ I lock_1)
C lock_in_)) in
let new_C ss = (((c Efsm stateA = CM2)) \ (C Efsm stateA = CM2)) \ CB ss in \ C ss) in
let c_mend = (CB ms in = *MEND) in
let c_mabort = (CB ms in = *MABORT) in
let new_C last out_ =
(((c Efsm stateA = CSA1) \ (~C ClkD \ (c mend V c mabort))) => T \ ((c Efsm stateA = CSA1) \ (C ClkD \ (c mend V c mabort))) => F \ ((c Efsm stateA = CSA1) \ (~C ClkD \ (c mend V c mabort))) => C last out_ \ ARB)) in
let c srdy = (CB ss in = *SRDY) in
let c dfsm_master = (c Efsm stateA = CMA3) \ (c Efsm stateA = CMA2) V (c Efsm stateA = CMA1)
V (c Efsm stateA = CMA0) \ (c Efsm stateA = CMD1) \ (c Efsm stateA = CMD0) in
let c dfsm cad en = ((c Efsm stateA = CMA3) \ (c Efsm stateA = CMA2) \ (c Efsm stateA = CMA1)
V (c Efsm stateA = CMA0) \ (c Efsm stateA = CMA2)
\[
V (c._new_write \wedge (c._mfsm_stateA = CMD1)) \vee (c._mfsm_stateA = CMD0))\]
\[
V (-c._new_write \wedge (c._sfsm_stateA = CMD1)) \vee (c._sfsm_stateA = CSD0))\]
let new_C_hold = (c._sfsm_stateA = CSI) in
let new_C_holdA = ((ClkD) => C_hold \l C_holdA) in
let new_C_cout_0_le_del = ((c.cale) \wedge (c.srdy \wedge \neg c._new_write)) \wedge CliKD\]
\[
V (c._mfsm_stateA = CMA0) \wedge c.srdy \wedge \neg c._new_write \wedge CliKD\]
\[
V ((c._mfsm_stateA = CMD0) \wedge \neg c.srdy \wedge \neg c._new_write)\]
let new_C_cin_2_le = (ClkD) \wedge (((c._mfsm_stateA = CMD0) \wedge \neg c.srdy \wedge \neg c._new_write) \vee \neg c._new_write) \wedge CliKD\]
\[
V ((c._mfsm_stateA = CSA0)) \vee ((c._mfsm_stateA = CSD0)) \wedge \neg c._new_write\]
\[
V (((c._mfsm_stateA = CSALE) \wedge \neg (c._sfsm_stateA = CSD0) )\wedge \neg c._new_write)) \wedge CliKD\]
\[
V ((c._mfsm_stateA = CSD1) \wedge \neg c._new_write \wedge (c._sfsm_stateA = CSRR)) \wedge CliKD\]
\[
V ((c._mfsm_stateA = CS0) \wedge \neg c._new_write) \wedge CliKD\]
\[
V ((c._mfsm_stateA = CSA0) \wedge CliKD) \wedge (c._sfsm_stateA = CMD1) \wedge CliKD\]
let new_C_wrdy = (c.srdy \wedge \neg c._new_write \wedge (c._mfsm_stateA = CMD1) \wedge CliKD) in
let new_C_srdy = (c.srdy \wedge \neg c._new_write \wedge (c._mfsm_stateA = CMD0) \wedge CliKD) in
let c.pe = (Par_Dec (CB.ad_in)) in
let c.mparity = ((c._mfsm_stateA = CMA3) \wedge (c._mfsm_stateA = CMA1) \wedge (c._mfsm_stateA = CMA0) \wedge (c._mfsm_stateA = CMA2)) \wedge (c._mfsm_stateA = CMD1) \wedge (c._mfsm_stateA = CMD0) \wedge (c._mfsm_stateA = CMA1) \wedge (c._mfsm_stateA = CMA0) \wedge (c._mfsm_stateA = CMA2) \wedge (c._mfsm_stateA = CMD1) \wedge (c._mfsm_stateA = CMD0) \wedge (c._mfsm_stateA = CS0) \wedge (c._mfsm_stateA = CSA0) \wedge (c._mfsm_stateA = CS1) \wedge (c._mfsm_stateA = CSL)) \wedge (Par_Dec (CB.ad_in)) \wedge (C.ssource)\]
let new_C_csource = ((Rst) => (WORD0))\]
\[
V ((ClkD) \wedge (c._pe \wedge c._pe \wedge \neg c._new_write) \wedge (c._sfsm_stateA = CSD0)) \wedge (c._sfsm_stateA = CSI) \wedge (c._sfsm_stateA = CSL))) \wedge (Par_Dec (CB.ad_in)) \wedge (C.ssource)\]
let data.in31_16 =\]
\[
(\text{MALTER ARBN (31, 16)) (\text{Rst}) => (\text{WORD0}) \wedge (\text{ClkD} \wedge ((\text{c.mfsm.stateA} = \text{CMD1}) \wedge c.srdy \wedge \neg c._new_write)) \vee ((\text{c.mfsm.stateA} = \text{CSA})) \vee ((\text{c.mfsm.stateA} = \text{CSD1}) \wedge c._new_write)) \wedge \text{Par_Dec (CB.ad_in) \wedge (SUBARRAY C.data.in (31, 16)))}\]
let data.in31_0 =\]
\[
(\text{MALTER data.in31_16 (15, 0)) (\text{Rst}) => (\text{WORD0}) \wedge (\text{new.C.cin_2_le} \Rightarrow \text{Par_Dec (CB.ad_in) \wedge (SUBARRAY C.data.in (15, 0)))}\]
let new_C.data .in = data. in31_0 in
let new_C.res.out = ((C.cin_2_le) => C.data .in \wedge C.res.out) in
let new_C.res.in = ((new.C.cin_2_le) => C.data .in \wedge C.res.in) in
let new_C.s1a0 =\]
\[
((c._dfsm_master \wedge c.cou.t_0 .le .del) \wedge c.mfsm.stateA = CS0) \wedge c.dfsm_master \wedge c.cou.t_0 .le .del) \wedge c.mfsm.stateA = CMD1) \wedge c.srdy \wedge \neg c._new_write) \wedge CliKD\]
\[
V ((c._mfsm_stateA = CSD1) \wedge c._new_write)) \Rightarrow \text{Par_Dec (CB.ad_in) \wedge (SUBARRAY C.data.in (15, 0)))}\]
\[
let new_C.data .in = C.data .in \wedge C.res.out) in
let new_C.res.in = ((new.C.cin_2_le) \wedge C.data .in \wedge C.res.in) in
let new_C.s1a0 =\]
\[
((c._dfsm_master \wedge C.cou.t_0 .le .del) \wedge \neg c._new_write) \wedge CliKD\]
\[
let new_C.s3a2 = ((C.mfsm.stateA = CMR) \Rightarrow C.s3a2) \wedge c.mfsm.state = c.mfsm.stateA in
let new_C.mfsm.D = CliKD in
\[
\]
let new C_mfsm_rst = Rst in
let new C_mfsm_cq = I_cq in
let new C_mfsm_hold = new C_holdA in
let new C_mfsm_ss = CB_ss in
let new C_mfsm_invalid = Piu_invalid in
let new C_sfsm_state = c_sfsm_stateA in
let new C_mfsm_D = ClkB in
let new C_sfsm_rst = Rst in
let new C_mfsm_blda = I_blda in
let new C_mfsm_ms = CB_ms in
let new C_esfsm_cale = I_cale in
let new C_esfsm_last = I_last in
let new C_esfsm_male = I_male in
let new C_esfsm_rale = I_rale in
let new C_esfsm_srdy = I_srdy in
let new C_mfsm_rst = Rst in

let I_cq = ~((c_mfsm_stateA = CMA3)) in
let I_mrdy_out = ((~I_blda) => C_mrdy_del | ARB) in
let I_hold = new C_holdA in
let I_rale_out =
  ((~I_blda) =>
   ~((c_sfsm_stateA = CSALE)) 
   | ((SUBARRAY new C_sizewrbe (1,0)) = (WORDN 3)) 
   | C_clkA | ARB) in
let I_male_out =
  ((~I_blda) =>
   ~((c_sfsm_stateA = CSALE)) 
   | ~((SUBARRAY new C_sizewrbe (1,0)) = (WORDN 3)) 
   | C_clkA | ARB) in
let I_last_out =
  ((~I_blda) => C_last_out | ARB) in
let I_srdy_out = ((~I_cale) V c_srdy_en) => ~((C_wrdy V C_rdy V (c_mfsm_stateA = CMABT))) | ARB in
let I_be_out = ((~I_blda) => (SUBARRAY new C_sizewrbe (9,6)) | ARB) in
let I_ad_out =
  (new C_iad_en_s_delA
   V ((c_mfsm_stateA = CMD1) 
   | c_new_write V c_srdy_en)
   V (c_mfsm_stateA = CMD0) 
   | c_new_write V c_srdy_en)
   V (c_mfsm_stateA = CMW) 
   | (C_mfsm_state = CMD0) 
   | ~c_new_write V c_srdy_en)
   V (c_mfsm_stateA = CSALE) 
   | ~(C_sfsm_state = CSALE))
   V (c_mfsm_stateA = CSD1) 
   | c_new_write V ~((C_sfsm_state = CSRR))
   V (c_mfsm_stateA = CSD0) 
   | c_new_write)
   V (c_sfsm_stateA = CSACK) 
   | c_new_write)) => new C_iad_out | ARB) in
let CB_rq_out = ~((c_mfsm_stateA = CMD)) in
let ms0 = (ALTER ARB (0) ((c_mfsm_stateA = CMD0) 
          | ~C_last_in) V
          (c_mfsm_stateA = CMW) 
          | C_lock_in) V
          (c_mfsm_stateA = CMABT)) in
let ms10 = (ALTER ms0 (1) ((c_mfsm_stateA = CMA1) V 
          (c_mfsm_stateA = CMA2) V (c_mfsm_stateA = CMD1) V
          (c_mfsm_stateA = CMA0) V (c_mfsm_stateA = CMD0) 
          | C_last_in) V (c_mfsm_stateA = CMW) V
          (c_mfsm_stateA = CMABT)) in
let ms210 = (ALTER ms10 (2) ((c_mfsm_stateA = CMA3) V 
          (c_mfsm_stateA = CMA0) V (c_mfsm_stateA = CMA2) V
          (c_mfsm_stateA = CMD1) V (c_mfsm_stateA = CMD0) V
          (c_mfsm_stateA = CMW) V (c_mfsm_stateA = CMABT) 
          | ~Pmm_failure | ~Piu_invalid)) in
let CB_mq_out = ((~(c_mfsm_stateA = CM1)) 
          | ~(c_mfsm_stateA = CMR)) => ms210 | ARB) in
let ss0 = (ALTER ARB (0) ((c_sfsm_stateA = CSA0W) V
          (c_sfsm_stateA = CSALE) | ~c_new_write) V

207
(c.sfsm_stateA = CSACK)) in
let ss10 = (ALTER ss0 (1) -(c.sfsm_stateA = CSACK)) in
let ss210 = (ALTER ss10 (2) (~Pmm_failure \&\& ~Piu_invalid)) in
let CB_ss_out = (((-c.sfsm_stateA = CSI)) \&\& (-c.sfsm_stateA = CSABT)) => ss210 | ARBN) in
let CB_ad_out = ((c.sfsm_stateA = CSI)) \&\& (-c.sfsm_stateA = CSABT)) => ss210 | ARBN) in

let C_ss_out = new_C_ss in
let Disable_writes = ((-c.sfsm_stateA = CSI)) \&\& (-c.sfsm_stateA = CSABT)) \&
-((ChannelID = (WORDN 0)) \&\& (ELEMENT Csource (6))) \&
-((ChannelID = (WORDN 1)) \&\& (ELEMENT Csource (7))) \&
-((ChannelID = (WORDN 2)) \&\& (ELEMENT Csource (8))) \&
-((ChannelID = (WORDN 3)) \&\& (ELEMENT Csource (9))) in

let CB_parity = new_C_parity in

(l_cgnt, l_mrdy_out, l_hold, l_rale_out, l_male_out, l_last_out, l_srdy_out, l_ad_out, l_be_out,
CB_rqLout, CB_ms_out, CB_ss_out, CB_ad_out, C_ss_out, Disable_writes, CB_parity)

);;
close_theory();

208
This file contains the ML source for the clock-level specification of the startup controller of the FTEP PIU, an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center. The bulk of this code was translated from an M-language simulation program using a translator written by P.J. Windley at the University of Idaho.

```
set_search_path (search_path0 @ ['/home/fitan3/dfura/ftep/piu/lib']);

system 'rm s_clock.th';

new_theory 's_clock';

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

let sc_stam._ty = ',:(sfsm_ty#bool#bool#bool#bool#bool#bool#wordn#wordn#
  bool#bool#bool#bool#bool#bool#bool#bool#bool)';

let sc_stam = "((S fsm state, S fsm rst, S fsm delay6, S fsm delay7, S fsm bothbad, S fsm bypass,
  S soft shot del, S soft cnt, S delay, S bad cpu0, S bad cpu1, S reset cpu0, S reset cpu1,
  S cpu hist, S pmn fail, S cpu0 fail, S cpu1 fail, S piu fail)
  :sc_state_ty)";;

let sc_env_ty = ":(bool#bool#bool#bool#bool#bool#bool#bool#bool)";;

let sc_env = "((ClkA, ClkB, Rst, Bypass, Test, Gcrh, Gcrl, Failure0_, Failure1_)
  :sc_env_ty)";;

let sc_out_ty = ":(wordn#bool#bool#bool#bool#bool#bool#bool#bool#bool)";;

let sc_out = "((S state, Reset cport, Disable int, Reset piu, Reset cpu0, Reset cpu1, Cpu hist,
  Piu fail, Cpu0 fail, Cpu1 fail, Pmm fail)
  :sc_out_ty)";;

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

Next-state definition for EXEC instruction.
%-----------------------------------------------

let sEXEC_inst_def = new_definition
('sEXEC_inst',
  "! (S fsm state:sfsm_ty)
   (S soft cnt S delay :wordn)
   (S fsm rst S fsm delay6 S fsm delay7 S fsm bothbad S fsm bypass S soft shot del S bad cpu0
   S bad cpu1 S reset cpu0 S reset cpu1 S cpu hist S pmn fail S cpu0 fail S cpu1 fail
   S piu fail :bool)
   (ClkA ClkB Rst Bypass Test Gcrh Gcrl Failure0_, Failure1_ :bool)."
```

209
sEXEC_inst (S_fsm_state, S_fsm_rst, S_fsm_delay6, S_fsm_delay17, S_fsm_bothbad, S_fsm_bypass, S_soft_shot_del, S_soft_cnt, S_delay, S_bad_cpu0, S_bad_cpu1, S_reset_cpu0, S_reset_cpu1, S_cpu_hist, S_pmm_fail, S_cpu0_fail, S_cpu1_fail, S_pi0_fail)

let new_S_fsm_state =
  ((S_fsm_rst) => SSTART !
   ((S_fsm_state = SSTART) => SRA !
    ((S_fsm_state = SRA) => ((S_fsm_delay6) => (S_fsm_bypass) => SO ! SPF) ! SRA) !
    ((S_fsm_state = SPF) => SCO !
     ((S_fsm_state = SCO) => ((S_fsm_delay17) => COF ! COF) !
      ((S_fsm_state = COF) => ST !
        ((S_fsm_state = ST) => SCI !
         ((S_fsm_state = SCI) => (S_fsm_delay17) => SCI ! SCI) !
          ((S_fsm_state = SCI) => (S_fsm_bothbad) => SSTOP ! SCS) !
           ((S_fsm_state = SSTOP) => SSTOP !
             ((S_fsm_state = SSTOP) => SC1 !
              ((S_fsm_state = SC1) => (S_fsm_delay17) => SSTOP ! SSTOP) !
               ((S_fsm_state = SSTOP) => SSTOP !
                 ((S_fsm_state = SSTOP) => SC2 !
                  ((S_fsm_state = SC2) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                   ((S_fsm_state = SSTOP) => SSTOP !
                     ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                      ((S_fsm_state = SCI) => (S_fsm_delay17) => SCI ! SCI) !
                       ((S_fsm_state = SCI) => (S_fsm_bothbad) => SSTOP ! SCS) !
                        ((S_fsm_state = SSTOP) => SSTOP !
                          ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                           ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                            ((S_fsm_state = SSTOP) => SSTOP !
                              ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                               ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                ((S_fsm_state = SSTOP) => SSTOP !
                                  ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                   ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                    ((S_fsm_state = SSTOP) => SSTOP !
                                      ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                       ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                        ((S_fsm_state = SSTOP) => SSTOP !
                                          ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                           ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                            ((S_fsm_state = SSTOP) => SSTOP !
                                              ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                               ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                                ((S_fsm_state = SSTOP) => SSTOP !
                                                  ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                                   ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                                    ((S_fsm_state = SSTOP) => SSTOP !
                                                      ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                                       ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                                        ((S_fsm_state = SSTOP) => SSTOP !
                                                          ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                                           ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                                            ((S_fsm_state = SSTOP) => SSTOP !
                                                              ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                                               ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                                                ((S_fsm_state = SSTOP) => SSTOP !
                                                                  ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                                                   ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                                                    ((S_fsm_state = SSTOP) => SSTOP !
                                                                      ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                                                       ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                                                        ((S_fsm_state = SSTOP) => SSTOP !
                                                                          ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                                                           ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                                                            ((S_fsm_state = SSTOP) => SSTOP !
                                                                             ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                                                              ((S_fsm_state = SCI) => (S_fsm_delay17) => SSTOP ! SSTOP) !
                                                                               ((S_fsm_state = SSTOP) => SSTOP !
                                                                                ((S_fsm_state = SSTOP) => SCI ! SSTOP) !
                                                                             )

let new_S_soft_shot_del = (Gcrh ! Gcrcl !)
let new_S_soft_cnt = (Gcrh ! Gcrcl !)
let new_S_soft_cnt_out =
  ((s_fsm_srcs) =>
    ((Gcrh ! Gcrh ! S_soft_shot_del) => (WORDN 1) ! (WORDN 0) !
     ((Gcrh ! Gcrh ! S_soft_shot_del) => (INCN 2 S_soft_cnt ! S_soft_cnt)) !
  )
let new_S_soft_cnt = (Gcrh ! Gcrcl !)
let new_S_delay_out =
  ((s_fsm_src V (S_fsm_sc6s ! (ELEMENT S_delay)))) =>
    ((S_fsm_sec) => (WORDN 1) ! (WORDN 0) !
     ((S_fsm_sec) => (INCN 17 S_delay ! S_delay)) !
  )
let new_S_delay = s_delay_out !
let s_cpu0_ok = (s_fsm_sc6f ! Failure0 ! (s_soft_cnt_out = (WORDN 5)) !
let s_cpu1_ok = (s_fsm_sc1f ! Failure1 ! (s_soft_cnt_out = (WORDN 5)) !
let new_S_pmm_fail =
(\langle s_{\text{fsm\_sb}} \land \neg s_{\text{fsm\_spmf}} \rangle \Rightarrow T I
(\langle s_{\text{fsm\_sb}} \land s_{\text{fsm\_spmf}} \rangle \Rightarrow F I
(\langle s_{\text{fsm\_sb}} \land \neg s_{\text{fsm\_spmf}} \rangle \Rightarrow \text{S\_pmn\_fail} (\text{ARB}))) in

let \text{new\_S\_cpuO\_fail} =
(\langle s_{\text{fsm\_sb}} \land \neg (s_{\text{cpuO\_ok}} \lor \text{Bypass}) \rangle \Rightarrow T I
(\langle s_{\text{fsm\_sb}} \land (s_{\text{cpuO\_ok}} \lor \text{Bypass}) \rangle \Rightarrow F I
(\langle s_{\text{fsm\_sb}} \land \neg (s_{\text{cpuO\_ok}} \lor \text{Bypass}) \rangle \Rightarrow \text{S\_cpuO\_fail} (\text{ARB}))) in

let \text{new\_S\_cpul\_fail} =
(\langle s_{\text{fsm\_sb}} \land \neg (s_{\text{cpul\_ok}} \lor \text{Bypass}) \rangle \Rightarrow T I
(\langle s_{\text{fsm\_sb}} \land (s_{\text{cpul\_ok}} \lor \text{Bypass}) \rangle \Rightarrow F I
(\langle s_{\text{fsm\_sb}} \land \neg (s_{\text{cpul\_ok}} \lor \text{Bypass}) \rangle \Rightarrow \text{S\_cpul\_fail} (\text{ARB}))) in

let \text{new\_S\_piu\_fail} =
(\langle s_{\text{fsm\_sb}} \land \neg (s_{\text{fsm\_spf}} \lor \text{Bypass}) \rangle \Rightarrow T I
(\langle s_{\text{fsm\_sb}} \land (s_{\text{fsm\_spf}} \lor \text{Bypass}) \rangle \Rightarrow F I
(\langle s_{\text{fsm\_sb}} \land \neg (s_{\text{fsm\_spf}} \lor \text{Bypass}) \rangle \Rightarrow \text{S\_piu\_fail} (\text{ARB}))) in

let \text{s\_cpu1\_select} = ((s_{\text{fsm\_sn}} \lor s_{\text{fsm\_so}}) \land \neg S\_cpu0\_fail) in
let \text{s\_cpul\_select} = ((s_{\text{fsm\_sn}} \lor s_{\text{fsm\_so}}) \land S\_cpu0\_fail \land \neg S\_cpu1\_fail) in

let \text{new\_S\_bad\_cpu0} =
(\langle s_{\text{fsm\_sb}} \land \neg s_{\text{cpu0\_select}} \rangle \Rightarrow T I
(\langle s_{\text{fsm\_sb}} \land s_{\text{cpu0\_select}} \rangle \Rightarrow F I
(\langle s_{\text{fsm\_sb}} \land \neg s_{\text{cpu0\_select}} \rangle \Rightarrow \text{S\_bad\_cpu0} (\text{ARB}))) in

let \text{new\_S\_bad\_cpul} =
(\langle s_{\text{fsm\_sb}} \land \neg s_{\text{cpu1\_select}} \rangle \Rightarrow T I
(\langle s_{\text{fsm\_sb}} \land s_{\text{cpu1\_select}} \rangle \Rightarrow F I
(\langle s_{\text{fsm\_sb}} \land \neg s_{\text{cpu1\_select}} \rangle \Rightarrow \text{S\_bad\_cpul} (\text{ARB}))) in

let \text{new\_S\_reset\_cpu0} = (\text{new\_S\_bad\_cpu0} \land s_{\text{flsm\_src0}}) in
let \text{new\_S\_reset\_cpul} = (\text{new\_S\_bad\_cpul} \land s_{\text{flsm\_src1}}) in
let \text{new\_S\_cpu\_hist} = (S\_reset\_cp0 \land S\_reset\_cp1 \land \text{Bypass}) in
let \text{new\_S\_fsm\_rst} = \text{Rst} in
let \text{new\_S\_fsm\_delay6} = (\text{ELEMENT s\_delay\_out (6)}) in
let \text{new\_S\_fsm\_delay17} = (\text{Test} \Rightarrow (\text{ELEMENT s\_delay\_out (6)}) \land (\text{ELEMENT s\_delay\_out (17)})) in
let \text{new\_S\_fsm\_bothbad} = (\text{new\_S\_cpu0\_fail} \land \text{new\_S\_cpu1\_fail}) in
let \text{new\_S\_fsm\_bypass} = \text{Bypass in}

(new\_S\_fsm\_state, new\_S\_fsm\_rst, new\_S\_fsm\_delay6, new\_S\_fsm\_delay17, new\_S\_fsm\_bothbad,
new\_S\_fsm\_bypass, new\_S\_soft\_shot\_del, new\_S\_soft\_cnt, new\_S\_delay, new\_S\_bad\_cp0, new\_S\_bad\_cp1,
new\_S\_reset\_cp0, new\_S\_reset\_cp1, new\_S\_cpu\_hist, new\_S\_pmn\_fail, new\_S\_cpu0\_fail, new\_S\_cpu1\_fail,
new\_S\_piu\_fail))

);;

%---------------------------------------------------------------------
% Output definition for EXEC instruction.
%---------------------------------------------------------------------

let \text{SEXC\_out\_def} = \text{new\_definition}
('S\_EXEC\_out',
"1 (S\_fsm\_state :sfsm\_ty)
(S\_soft\_cnt S\_delay :wordn)
(S\_fsm\_rst S\_fsm\_delay6 S\_fsm\_delay17 S\_fsm\_bothbad S\_fsm\_bypass S\_soft\_shot\_del S\_bad\_cp0
S\_bad\_cp1 S\_reset\_cp0 S\_reset\_cp1 S\_cpu\_hist S\_pmn\_fail S\_cpu0\_fail S\_cpu1\_fail
S\_piu\_fail :bool)
(CIA\_1 CIB\_1 Rst\_Bypass\_Test Gcrh\_Gcr\_Failure0\_\_Failure1 :bool).
S\_EXEC\_out\_def (S\_fsm\_state, S\_fsm\_rst, S\_fsm\_delay6, S\_fsm\_delay17, S\_fsm\_bothbad, S\_fsm\_bypass,
S\_soft\_shot\_del, S\_soft\_cnt, S\_delay, S\_bad\_cp0, S\_bad\_cp1, S\_reset\_cp0, S\_reset\_cp1,
let new_S_fsm_state =
  ((S_fsm_rst) => SSTART)
  ((S_fsm_state = SSTART) => SRA)
  ((S_fsm_state = SRA) => ((S_fsm_delay6) => (S_fsm_bypass) => SO | SPF) | SRA)
  ((S_fsm_state = SPF) => SCO)
  ((S_fsm_state = SCO) => ((S_fsm_delay17) => SCOF | SCO)
  ((S_fsm_state = SCOF) => ST)
  ((S_fsm_state = ST) => SCI)
  ((S_fsm_state = SCI) => ((S_fsm_delay17) => SCI1 | SCI)
  ((S_fsm_state = SCI1) => (S_fsm_bothbad) => SSTOP | SCS)
  ((S_fsm_state = SCS) => SSTOP)
  ((S_fsm_state = SCS) => ((S_fsm_bothbad) => SSTOP | SCS)
  ((S_fsm_state = SSTOP) => SSTOP)
  ((S_fsm_state = SCS) => ((S_fsm_delay6) => SO | SCS)
  ((S_fsm_state = SO) => SO)
  ((S_fsm_state = SCS) => SCS)

let s_fsm_sn = (new_S_fsm_state = SN) in
let s_fsm_so = (new_S_fsm_state = SO) in
let s_fsm_srcp = (((~(new_S_fsm_state = SO)) V (S_fsm_state = SSTOP))) V (S_fsm_state = SRA)) in
let s_fsm_sdi = (((~(new_S_fsm_state = SO)) V (S_fsm_state = SSTOP))) V (S_fsm_state = SRA)) in
let s_fsm_srp = (new_S_fsm_state = SSTART) V (new_S_fsm_state = SRA)
  V (new_S_fsm_state = SPF) V (new_S_fsm_state = SRA)
let s_fsm_spf = (new_S_fsm_state = SRA) V (S_fsm_delay6) ~S_fsm_rst in
let s_fsm_sc0f = (new_S_fsm_state = SCOF) in
let s_fsm_sc1f = (new_S_fsm_state = SCI1) in
let s_fsm_spnf = (new_S_fsm_state = SO) in
let s_fsm_sc0 = (new_S_fsm_state = SCS) in
let new_S_fsm_src = (S_fsm_state = SSTOP) V (S_fsm_state = SCS)
let s_fsm_sc0f = (new_S_fsm_state = SPF) ~S_fsm_rst)
let new_S_fsm_sc0f = (new_S_fsm_state = SCS) in
let new_S_soft_shot_del = (~Gcrh ^ Gcrh) in
let s_soft_cnt_out =
  ((s_fsm_src) =>
  ((Gcrh ^ ~Gcrh) => (WORDN 0) | (WORDN 1)))
let new_S_soft_cnt = (~Gcrh ^ Gcrh) => (WORDN 0) | s_soft_cnt_out)
let s_delay_out =
  ((s_fsm_src V (s_fsm_sc0f ^ (ELEMENT S_delay (6)))) =>
  ((s_fsm_src) => (WORDN 0)) | (WORDN 1))
let new_S_delay = s_delay_out in
let s_cpu0_ok = (s_fsm_sc0f ^ Failure0_ ^ (s_soft_cnt_out = (WORDN 5))) in
let s_cpu1_ok = (s_fsm_sc1f ^ Failure1_ ^ (s_soft_cnt_out = (WORDN 5))) in
let new_S_pmm_fail =
  ((s_fsm_sc0 ^ s_fsm_sc0f) => T)
  (~s_fsm_sc0 ^ s_fsm_sc0f) => F
let new _S_cpu0_fail = 
  ((s_fsm_sb ∧ ¬(s_cpu0_ok V Bypass)) => T) 
  ((s_fsm_sb ∧ (s_cpu0_ok V Bypass)) => F) 
  ((s_fsm_sb ∧ ¬(s_cpu0_ok V Bypass)) => S_cpu0_fail l ARB)) in

let new _S_cpu1_fail = 
  ((s_fsm_sb ∧ ¬(s_cpu1_ok V Bypass)) => T) 
  ((s_fsm_sb ∧ (s_cpu1_ok V Bypass)) => F) 
  ((s_fsm_sb ∧ ¬(s_cpu1_ok V Bypass)) => S_cpu1_fail l ARB)) in

let new _S_piu_fail = 
  ((s_fsm_sb ∧ ¬(s_fsm_spf V Bypass)) => T) 
  ((s_fsm_sb ∧ (s_fsm_spf V Bypass)) => F) 
  ((s_fsm_sb ∧ ¬(s_fsm_spf V Bypass)) => S_piu_fail l ARB)) in

let _s_cpu0_select = ((s_fsm_sn V s fsm_so) ∧ ¬S_cpu0_fail) in

let _s_cpu1_select = ((s_fsm_sn V s fsm_so) ∧ S_cpu0_fail ∧ ¬S_cpu1_fail) in

let new _S_bad_cpu0 = 
  ((s_fsm_sb ∧ ¬_s_cpu0_select) => T) 
  ((s_fsm_sb ∧ ¬_s_cpu0_select) => F) 
  ((¬s_fsm_sb ∧ ¬_s_cpu0_select) => S_bad_cpu0 1 ARB)) in

let new _S_bad_cpu1 = 
  ((s_fsm_sb ∧ ¬_s_cpu1_select) => T) 
  ((s_fsm_sb ∧ ¬_s_cpu1_select) => F) 
  ((¬s_fsm_sb ∧ ¬_s_cpu1_select) => S_bad_cpu1 1 ARB)) in

let new _S_reset_cpu0 = (new _S_bad_cpu0 ∧ s_fsm_src0) in

let new _S_reset_cpu1 = (new _S_bad_cpu1 ∧ s_fsm_src1) in

let new _S_cpu_hist = (S_reset_cpu0 ∧ S_reset_cpu1 ∧ Bypass) in

let new _S_fsm_rst = Rst in

let new _S_fsm_delay6 = (ELEMENT s_delay_out (6)) in

let new _S_fsm_delay17 = ((Test) => (ELEMENT s_delay_out (6) l (ELEMENT s_delay_out (17)))) in

let new _S_fsm_bothbad = (new _S_cpu0_fail ∧ new _S_cpu1_fail) in

let new _S_fsm_bypass = Bypass in

let ss0 = (ALTER ARBN (0) ((new _S_fsm_state = SS) V (new _S_fsm_state = SSTOP) 
  V (new _S_fsm_state = SCS) V (new _S_fsm_state = SN) 
  V (new _S_fsm_state = SO))) in

let ss1 = (ALTER ss0 (1) ((new _S_fsm_state = SCOF) V (new _S_fsm_state = ST) 
  V (new _S_fsm_state = SC11) V (new _S_fsm_state = SC1P) 
  V (new _S_fsm_state = SS) V (new _S_fsm_state = SSTOP) 
  V (new _S_fsm_state = SCS))) in

let ss2 = (ALTER ss1 (2) ((new _S_fsm_state = SPF) V (new _S_fsm_state = SC01) 
  V (new _S_fsm_state = SCOF) V (new _S_fsm_state = ST) 
  V (new _S_fsm_state = SSTOP) V (new _S_fsm_state = SO))) in

let ss3 = (ALTER ss2 (3) ((new _S_fsm_state = SRA) V (new _S_fsm_state = SPF) 
  V (new _S_fsm_state = ST) V (new _S_fsm_state = SC11) 
  V (new _S_fsm_state = SCS) V (new _S_fsm_state = SN) 
  V (new _S_fsm_state = SO))) in

let _S_state = ss3 in

let _Reset_cport = s_fsm_srcp in

let Disable_int = (¬(s_fsm_sn ∧ (ELEMENT s_delay_out (6))) ∧ s_fsm_sdi ∧ ((Test) => ¬(ELEMENT s_delay_out (5)) l ¬(ELEMENT s_delay_out (16)))) in

let _Reset_piup = s_fsm_srcp in

let _Reset_cpu0 = new _S_reset_cpu0 in

let _Reset_cpu1 = new _S_reset_cpu1 in

let _Cpu_hist = new _S_cpu_hist in

let _Piufail = new _S_piufail in

213
let Cpu0_fail = new_S_cpu0_fail in
let Cpu1_fail = new_S_cpu1_fail in
let Pmm_fail = new_S_pmm_fail in

(S_state, Reset_cport, Disable_int, Reset_piu, Reset_cpu0, Reset_cpu1, Cpu_hist, Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail)"

);;
close_theory();;
Appendix E ML Source for the PIU Block-Level Specification.

This appendix contains the HOL model for the PIU block-level structural specification.

---

File: piu_block.ml

Author: (c) D.A. Fura 1992

Date: 31 March 1992

This file contains the ml source for the block-level specification of the FTEP PIU, an ASIC developed by the Embedded Processing Laboratory, Boeing High Technology Center. At this level the blocks correspond to the four PIU ports and the startup controller.

---

```ml
set_search_path (search_path) @ ['/home/titan3/dfura/ftep/piu/hol/lib/';
   '/home/titan3/dfura/ftep/piu/hol/pport/';
   '/home/titan3/dfura/ftep/piu/hol/eport/';
   '/home/titan3/dfura/ftep/piu/hol/import/';
   '/home/titan3/dfura/ftep/piu/hol/export/';
   '/home/titan3/dfura/ftep/piu/hol/sucont/']));

system 'rm piu_block.th';

new_theory 'piu_block';

load'abstract';

map new_parent ['aux_def';'p_clock1';'c_clock1';'m_clock1';'c_clock1';'s_clock1'];

let rep_ty = abstract_type 'aux_def' 'Andn';

let PIU_Block_SPEC = new_definition
   ('PIU_Block_SPEC',
    "1 (rep:rep_ty)
    (P_fsm_state :pfsm_ty)
    (P_addr P_be P_size :wordn)
    (P_dest P_wr P_fsm_rst P_fsm sacked P_fsm_hold P_rqt P_down P_lock_
    P_lock_inh P_male P_rale :bool)
    (C_mfsm_state :cmfsm_ty) (C_sfsm_state :csfsm_ty) (C_esfsm_state :cesfsm_ty)
    (C_mfsm ss C_sfsm ms C_msizewrbe C_ss C_source C_data_in C_iad_out C_iad_in C_s1a0 C_s3a2 :wordn)
    (C_mfsm D C_mfsm rst C_mfsm criq C_mfsm_hold C_mfsm_invalid C_esfsm D C_esfsm rst C_esfsm hida_
    C_esfsm cale C_esfsm last C_esfsm male C_esfsm rale C_esfsm_srdy C_esfsm rst
    C_wr C_clkA C_last_in C_lock_in C_last_out C_hold C_holdA C_cout0_out_cle del C_cin2_le
    C_mrdy del C_iad en s del C_iad en s del A C_wrdy C_rdy C_parity :bool)
    (M_fsm_state :mfsn_ty)
    (M_count M_addr M_be M_rd_data M_detect :wordn)
    (M_fsm male M_fsm last M_fsm mrdy M_fsm rst M_se M_wr M_rdy M_wwdel M_parity :bool)
    (R_fsm_state rfsn ty)
    (R_ctr0 Init R_ctl0 out R_ctr1 in R_ctl1 R_ctl1 new R_ctr1 out R_ctr2 in R_ctl2 R_ctl2 new
    R_ctr2 out R_ctr3 in R_ctl3 R_ctl3 new R_ctr3 out R_ctr old R_icr mask R_icr R_ccr R_gcr R_sr
```

215
(s_state, reset_cport, disable_int, reset_piu, reset_cpu0, reset_cpu1, Cpu_hist, piu_fail, cpu0_fail, cpu1_fail, pmm_fail))

); close_theory();


Appendix F ML Source for the PIU Clock-Level Specification.

This appendix contains the HOL model for the clock-level specification of the PIU.

---

File: piu_clock.ml

Author: (c) D.A. Fura 1992

Date: 31 March 1992

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

---

```ml
set_search_path (search_path @ ['faome/fitan3/dftn/ftep/piu/hol/lib/';
   'faome/titan3/dfura/ftep/piu/hol/';
   'faome/titan3/dfura/ftep/piu/hol/cport/';
   '/home/titan3/dfura/ftep/piu/hol/cco/']);

system 'rm piu_clock1.th';

new theory 'piu_clock1';

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

loadf 'abstract';

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

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

let piu_state_ty = ":(w_rd_#b_#w_rdn#b_#pfsm--ty#b_#b_#b_#b_#b_#w_rd_#b_#b_#b_#b_#b_#cmfsm ty#bool#bool#bool#bool#wordn#bool#bool#bool#bool#bool#cmfsm ty#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#false

219
let piu_state = "((P_addr, P_dest, P_be, P_wr, P fsm_state, P fsm rst, P fsm_sack, P fsm_cqtn, P fsm_hold,
P_qtl, P size, P down, P lock, P lock_inh, P male, P rale,
C_fsm_state, C_fsm_D, C_fsm rst, C_fsm_cqtn, C_fsm_hold, C_fsm ss, C_fsm_invalid,
C_fsm_state, C_fsm_D, C_fsm rst, C_fsm_hld, C_fsm_ms,
C_esf_state, C esf_cale, C esf_last, C esf_male, C esf_rale, C esf_srdy, C esf rst,
C wr, C szwrbr, C clkA, C fast in, C lock_in, C ss, C last out,
C_hold, C hold_A, C cout_0, le_del, C cin, 2, le, C rdfy_del, C jad_en, s, del, C jad_en, s, delA,
C wrdy, C rdfy, C parity, C source, C data_in, C jad out, C jad_in, C salo, C salo2,
M fsm_state, M fsm male, M fsm last, M fsm_mrdy, M fsm rst, M count, M se, M wr, M addr,
M be, M rdy, M wwdel, M parity, M rd data, M detect,
R fsm state, R fsm ale, R fsm mrdy, R fsm last, R fsm rst, R ctrl0 in, R ctrl0 mux sel, R ctrl0,
R ctrl0 irden, R ctrl0_new, R ctrl0_cry, R ctrl0 out, R ctrl0 orden, R ctrl in, R ctrl1 mux sel,
R ctrl1, R ctrl1 irden, R ctrl1 new, R ctrl1_cry, R ctrl1 out, R ctrl1 orden, R ctrl2 in, R ctrl2 mux sel,
R ctrlz, R ctrl2 irden, R ctrlz new, R ctrlz_cry, R ctrlz out, R ctrlz orden, R ctrl3 in, R ctrl3 mux sel,
R ctrl3, R ctrl3 irden, R ctrl3 new, R ctrl3_cry, R ctrl3 out, R ctrl3 orden, R irden, R irden load, R irden old,
R icr mask, R icr rden, R icr, R ccr, R ccr rden, R gc, R gc rden, R sr, R sr rden, R int0 dis,
R int3 dis, R c01 cout del, R int1 en, R c23 cout del, R int2 en, R wr, R c0latch del, R srdy del,
R reg sel, R busA latch,
S fsm state, S fsm rst, S fsm delay6, S fsm delay17, S fsm bothbad, S fsm bypass,
S soft slot del, S soft cnt, S delay, S bad cpul, S bad_cqtn, S reset cpul, S reset cpul, S reset cpul,
S cpu hist, S pmr_fails, S cpu0 fail, S cpu1 fail, S cpu2 fail, S cpu fail)
: piu_state ty)";;

let piu_env_ty = "((bool#bool#bool#false#bool#false#false#false#false)
wordn#wordn#wordn#wordn#wordn#wordn#wordn#wordn#wordn#wordn#wordn)
wordn#bool#
bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#bool#b
let piuEXEC\_inst\_def = new\_definition
('piuEXEC\_inst',

"1 (rep\_0\_rep\_ty)
(P\_fsm\_state :fsym\_ty)
(P\_addr P\_be\_P\_size :wordsn)
(P\_dest1 P\_wr P\_fsm\_rst P\_fsm\_sack P\_fsm\_cgnt_ P\_fsm\_hold_ P\_rqt P\_down P\_lock_
P\_lock\_inh_ P\_male_ P\_rare :bool)
(C\_mfsm\_state :cmfsm\_ty) (C\_mfsm\_state :cmfsm\_ty) (C\_efsm\_state :cefsym\_ty)
(C\_mfsm\_ss C\_mfsm\_ms C\_sizewrbe C\_ss C\_source C\_data\_in C\_iad\_out C\_iad\_in C\_al\_0 C\_al\_a2 :wordn)
(C\_mfsm\_D C\_mfsm\_rst C\_mfsm\_crqt C\_mfsm\_hold_ C\_mfsm\_invalid C\_sfsm\_D C\_sfsm\_rst C\_sfsm\_hlda_
C\_efsm\_cale_ C\_efsm\_last_ C\_efsm\_male_ C\_efsm\_rare_ C\_efsm\_srty_ C\_efsm\_rst
C\_wr C\_clkA C\_last\_in_ C\_lock\_in_ C\_last\_out_ C\_hold_ C\_holdA C\_c0t\_0\_le\_del C\_cin\_2\_le
C\_mdry\_del_ C\_iad\_en\_s\_del C\_iad\_en\_s\_delA C\_wrdy C\_rdy C\_parity :bool)
(M\_fsm\_state :mfsm\_ty)
(M\_count M\_addr M\_be M\_rd\_data M\_detect :wordsn)
(M\_fsm\_male_ M\_fsm\_last_ M\_fsm\_mdry_ M\_fsm\_rst M\_se M\_wr M\_rdy M\_wwdel M\_parity :bool)
(R\_fsm\_state :rfsm\_ty)
(R\_ctr0 in R\_ctr0 new R\_ctr0\_out R\_ctr1 in R\_ctr1 new R\_ctr1\_out R\_ctr2 in R\_ctr2 new
R\_ctr2\_out R\_ctr3 in R\_ctr3 new R\_ctr3\_out R\_icer old R\_icer mask R\_icer R\_cr new R\_cr
R\_reg\_sel R\_bus\_A latch :wordsn)
(S\_fsm\_state :sfsm\_ty)
(S\_soft\_cnt S\_delay :wordsn)
(S\_fsm\_rst S\_fsm\_delay6 S\_fsm\_bothbad S\_fsm\_bypass S\_soft\_shot\_del S\_bad\_cpu0 S\_bad\_cpu1
S\_reset\_cpu0 S\_reset\_cpu1 S\_cpu\_hist S\_pmn\_fail S\_cpu\_fail S\_cpu\_fail S\_piu\_fail :bool)
(L\_ad\_in L\_be :wordsn)
(Clk\_A ClkB Rst L\_ads L\_den L\_wr L\_lock :bool)
(C\_b\_rq\_in_ C\_b\_ad\_in C\_b\_ms\_in C\_b\_ss\_in Id ChannelID :wordsn)
(Clk\_D :bool)
(MB\_data\_in :wordsn)
(Edac\_en :bool)
(Bypass Test Failure0 Failure1 :bool).

piuEXEC\_inst rep
(P\_addr P\_dest1 P\_be\_P\_wr P\_fsm\_state P\_fsm\_rst P\_fsm\_sack P\_fsm\_cgnt_ P\_fsm\_hold_
P\_rqt P\_size P\_down P\_lock_ P\_lock\_inh_ P\_male_ P\_rare ,
C\_mfsm\_state, C\_mfsm\_D, C\_mfsm\_rst, C\_mfsm\_crqt, C\_mfsm\_hold_, C\_mfsm\_ss, C\_mfsm\_invalid,
C\_sfsm\_state, C\_sfsm\_D, C\_sfsm\_rst, C\_sfsm\_hlda_, C\_sfsm\_ms,
C\_efsm\_state, C\_efsm\_cale_ C\_efsm\_last_, C\_efsm\_male_ C\_efsm\_rare_ C\_efsm\_srty_ C\_efsm\_rst,
C\_wr, C\_sizewrbe, C\_clkA, C\_last\_in_ C\_lock\_in_ C\_ss, C\_last\_out_,
C\_hold_ C\_holdA C\_c0t\_0\_le\_del C\_cin\_2\_le, C\_mdry\_del_ C\_iad\_en\_s\_del C\_iad\_en\_s\_delA
C\_wrdy C\_rdy C\_parity, C\_source C\_data\_in C\_iad\_out, C\_iad\_in C\_al\_0 C\_al\_a2,
M\_fsm\_state, M\_fsm\_male_ M\_fsm\_last_ M\_fsm\_mdry_ M\_fsm\_rst M\_count M\_se M\_wr M\_addr,
M\_be M\_rdy M\_wwdel M\_parity M\_rd\_data M\_detect,
R\_fsm\_state, R\_fsm\_ale_ R\_fsm\_mdry_ R\_fsm\_last_ R\_fsm\_rst R\_ctrl0 in R\_ctrl0 new R\_ctrl0\_cry R\_ctrl0\_out R\_ctrl0\_orden R\_ctr1 in R\_ctr1\_irden R\_ctr1 new R\_ctr1\_cry R\_ctr1\_orden R\_ctr2 in R\_ctr2\_irden R\_ctr2 new R\_ctr2\_cry R\_ctr2\_orden
R\_ctr3 in R\_ctr3\_irden R\_ctr3\_cry R\_ctr3\_orden R\_ctr3\_latch R\_ctr3\_hold_ R\_ctr3\_in R\_ctr3\_out
R\_ctr3\_mask R\_gcr\_orden R\_sr\_orden
R\_int\_0\_dis R\_int\_1\_dis R\_c01\_c0ut\_del R\_int\_1\_en R\_c23\_c0ut\_del R\_int\_2\_en R\_wr R\_cnt\_latch\_del R\_sr\_d\_del_:bool)
(S\_fsm\_state :sfsm\_ty)
(S\_soft\_cnt S\_delay :wordsn)
(S\_fsm\_rst S\_fsm\_delay6 S\_fsm\_bothbad S\_fsm\_bypass S\_soft\_shot\_del S\_bad\_cpu0 S\_bad\_cpu1
S\_reset\_cpu0 S\_reset\_cpu1 S\_cpu\_hist S\_pmn\_fail S\_cpu\_fail S\_cpu\_fail S\_piu\_fail :bool)

221
let new P_fsm_state =

(P fsm_state = PH) => ((-P_fsm_hold_) => PH I PA) !
((P fsm_state = PA) =>
  (((P_rq A ~P_destl) V (P_rqt A P_destl A ~P fsm_cgnt_)) => PD) !
  ((-P fsm_hold_ A P_lock_) => PH I PA)) !
((P fsm_state = PD) =>
  (((P fsm_sack A P fsm_hold_) V (P fsm_sack A ~P fsm_hold_ A P lock_)) => PA) !
  ((P fsm_sack A ~P fsm_hold_ A P lock_) => PH I PA)) !
)

let c_write = (((-C_mfsm_state = CMI)) A (-(C_mfsm_state = CMR))) => C_wr I (ELEMENT C_sizewrbe (5)) in
let c_busy = (-(SUBARRAY CB_rqt_in_ (3,1)) = (WORDN 7))) in
let c_grant = (((SUBARRAY Id (1,0)) = (WORDN 0)) A (ELEMENT CB_rqt_in_ (0))) V (((SUBARRAY Id (1,0)) = (WORDN 1)) A (ELEMENT CB_rqt_in_ (1))) V (((SUBARRAY Id (1,0)) = (WORDN 2)) A (ELEMENT CB_rqt_in_ (2))) V (((SUBARRAY Id (1,0)) = (WORDN 3)) A (ELEMENT CB_rqt_in_ (3))) in

let c_addressed = (Id = (SUBARRAY C_source (15,10))) in
let new C_mfsm_state =

(C_mfsm_rst) => CMI !
(C_mfsm_state = CMI) =>
  (C_mfsm_D A ~C_mfsm_cqnt_ A ~c_busy A ~C_mfsm_invalid) => CMR ! CMI !
((C_mfsm_state = CMR) => ((C_mfsm_D => CMR) => CMR) => (C_mfsm_D => CMR) => CMR) !
((C_mfsm_state = CMA3) => ((C_mfsm_D => CMA1) => CMA3) !
((C_mfsm_state = CMA1) =>
  (C_mfsm_D A (C_mfsm_ss = ^SRDY)) => CMA0 !
  (C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT ! CMA1 !
  (C_mfsm_state = CMA0) !
((C_mfsm_D A (C_mfsm_ss = ^SRDY)) => CMA2 !
  (C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT ! CMA0 !
((C_mfsm_state = CMA2) =>
  (C_mfsm_D A (C_mfsm_ss = ^SRDY)) => CMA0 !
  (C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT ! CMA2 !
((C_mfsm_state = CMD1) =>
  (C_mfsm_D A (C_mfsm_ss = ^SRDY)) => CMD0 !
  (C_mfsm_D A (C_mfsm_ss = ^SABORT)) => CMABT ! CMD1 !
let new_C_mfsm_state =

((C_mfsm_state = CMD0) =>
  (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ C_last_in_) => CMD1 l
  (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ¬C_last_in_) => CMW l
  (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT l CMD0 l
)

((C_mfsm_state = CMW) =>
  (C_mfsm_D ∧ (C_mfsm_ss = ^SABORT)) => CMABT l
  (C_mfsm_D ∧ (C_mfsm_ss = ^SACK) ∧ C_lock_in_) => CMI l
  (C_mfsm_D ∧ (C_mfsm_ss = ^SRDY) ∧ ¬C_lock_in_ ∧ ¬C_mfsm_creq_l) => CMA3 l CMW l
  (¬C_last_in_) => CMI l CMABT))))))) in

let new_C_sfsm_state =

((C_sfsm_rst) => CSI l
  (C_sfsm_state = CSI) =>
    ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ¬c_grant ∧ c_addressed) => CSA1 l CSI l
  (C_sfsm_state = CSL) =>
    ((C_sfsm_D ∧ (C_sfsm_ms = ^MSTART) ∧ ¬c_grant ∧ c_addressed) => CSA1 l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT l CSL l
  (C_sfsm_state = CSA1) =>
    ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSA0 l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT l CSA1 l
  (C_sfsm_state = CSA0) =>
    ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ¬C_sfsm_hlda_) => CSALE l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ C_sfsm_hlda_) => CSA0W l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT l CSA0 l
  (C_sfsm_state = CSA0W) =>
    ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY) ∧ ¬C_sfsm_hlda_) => CSALE l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT l CSA0W l
  (C_sfsm_state = CSALE) =>
    ((C_sfsm_D ∧ c_write ∧ (C_sfsm_ms = ^MRDY)) => CS1 l
    (C_sfsm_D ∧ ¬c_write ∧ (C_sfsm_ms = ^MRDY)) => CSRR l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT l CSALE l
  (C_sfsm_state = CSRR) =>
    ((C_sfsm_D ∧ ¬(C_sfsm_ms = ^MABORT)) => CSD1 l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT l CSRR l
  (C_sfsm_state = CSD1) =>
    ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD0 l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT l CSD1 l
  (C_sfsm_state = CSD0) =>
    ((C_sfsm_D ∧ (C_sfsm_ms = ^MEND)) => CSACK l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSD1 l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT l CSD0 l
  (C_sfsm_state = CSACK) =>
    ((C_sfsm_D ∧ (C_sfsm_ms = ^MRDY)) => CSL l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MWAIT)) => CS1 l
    (C_sfsm_D ∧ (C_sfsm_ms = ^MABORT)) => CSABT l CSACK l
  (C_sfsm_D) => CSI l CSABT) in

let m_bw = ((¬(M_be = (WORDN 15))) ∧ M_wr ∧ ((¬M_fsm_state = MI))) in
let m_ww = ((M_be = (WORDN 15)) ∧ M_wr ∧ (¬(M_fsm_state = MI))) in
let new_M_fsm_state =
   (M_fsm_rst) -> MI |
   (M_fsm_state = MI) -> (¬M_fsm_male_) -> MA | MI | MI
   (M_fsm_state = MA) ->
      (¬M_fsm_mrdy_ ⊓ m_ww) -> MW |
      (¬M_fsm_mrdy_ ⊓ (¬M_wr ⊓ (¬(M_fsm_state = MI)))) ∨ m_bw) -> MR | MA | MA
   (M_fsm_state = MR) ->
      (m_bw ∧ (M_count = (WORDN 0))) -> MBW |
      (¬(¬M_fsm_last_ ∧ ¬M_wr ∧ (¬M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) -> MA | MA
   (¬(¬M_fsm_last_ ∧ ¬M_wr ∧ (¬M_fsm_state = MI)) ∧ (M_count = (WORDN 0))) -> MRR | MR | MR
   (M_fsm_state = MRR) -> MI | MI
   (M_fsm_state = MW) ->
      (¬M_fsm_last_ ∧ (M_count = (WORDN 0))) -> RI | MI | MA
   (¬(¬M_fsm_last_ ∧ (M_count = (WORDN 0))) ∧ (M_count = (WORDN 0))) -> MA | MW | MW
   (¬(¬M_fsm_last_ ∧ (M_count = (WORDN 0))) ∧ (M_count = (WORDN 0))) ⊓ MA | RW
   (¬(¬M_fsm_last_ ∧ (M_count = (WORDN 0))) ∧ (M_count = (WORDN 0))) ⊓ MA | MR
   (M_fsm_state = MBW) -> MW | MW | MW

let new_R_fsm_state =
   (R_fsm_rst) -> RI | RI
   (R_fsm_state = RI) -> (¬R_fsm_name_) -> RA | RA | RI
   (R_fsm_state = RA) -> (¬R_fsm_mrdy_) -> RD | RA | RA
   (¬R_fsm_last_) -> RI | RI | RI

let new_S_fsm_state =
   (S_fsm_rst) -> SSTART | SSTART | SSTART
   (S_fsm_state = SSTART) -> SRA | SRA
   (S_fsm_state = SRA) -> (¬S fsm_delay6) -> ((¬S_fsm_bypass) -> SO | SPF) | SRA | SRA
   (S_fsm_state = SPF) -> SC0 | SC0
   (S_fsm_state = SC0) -> (¬S_fsm_delay17) -> SCOF | SC0 | SC0
   (S_fsm_state = SCOF) -> ST | ST
   (S_fsm_state = ST) -> SC1 | SC1
   (S_fsm_state = SC1) -> (¬S_fsm_delay17) -> SC1F | SC1 | SC1
   (S_fsm_state = SC1F) -> SS | SS
   (S_fsm_state = SS) -> (¬S_fsm_bothbad) -> SSTOP | SCS | SCS
   (S_fsm_state = SSTOP) -> SSTOP | SSTOP
   (S_fsm_state = SCS) -> (¬S_fsm_delay6) -> SN | SCS | SCS
   (S_fsm_state = SN) -> (¬S_fsm_delay17) -> SO | SN | SN
   (S_fsm_state = SO) -> SO | SO | SILL | SILL | SILL | SILL

let s_fsm_sn = (new_S_fsm_state = SN) in
let s_fsm_so = (new_S_fsm_state = SO) in
let reset_cport = (((¬(¬S fsm_state = SO)) ∧ (¬S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let s_fsm_sdi = (((¬(¬S fsm_state = SO)) ∧ (¬S_fsm_state = SSTOP))) ∨ (S_fsm_state = SRA)) in
let reset_piu = (new_S_fsm_state = SSTART) ∨ (new_S_fsm_state = SRA)
   ∨ (new_S_fsm_state = SC0) ∨ (new_S_fsm_state = ST)
   ∨ (new_S_fsm_state = SC1) ∨ (new_S_fsm_state = SS) ∨ (new_S_fsm_state = SCS)
   ∨ (¬S_fsm_state = SPF) ∧ (¬S_fsm_state = SC0) ∧ (¬S_fsm_state = ST) ∧ (¬S_fsm_state = SC1) ∧ (¬S_fsm_state = SC0) ∧ (¬S_fsm_state = SO)
   ∨ (new_S_fsm_state = SC0) ∨ (new_S_fsm_state = SC1) ∨ (new_S_fsm_state = SC0)
   ∨ (new_S_fsm_state = SPF) ∨ (new_S_fsm_state = SO)
   ∨ (new_S_fsm_state = SC0)
   ∨ (new_S_fsm_state = SC1)
   ∨ (new_S_fsm_state = SPF)
   ∨ (new_S_fsm_state = SO)
   ∨ (new_S_fsm_state = SC0)
   ∨ (new_S_fsm_state = SC1)

let s_fsm_src0 = ((¬S_fsm_state = SPF)) ∧ (¬(¬S_fsm_state = SC0)) in
let s_fsm_src1 = ((¬S_fsm_state = ST)) ∧ (¬(¬S_fsm_state = SC1)) in
let s_fsm_spf = (S_fsm_state = SRA) ∧ (¬S_fsm_delay6 ∧ ¬S_fsm_rst) in
let s_fsm_sc0f = (new_S_fsm_state = SC0) in
let s_fsm_sc1f = (new_S_fsm_state = SC1) in
let s_fsm_spnf = (new_S_fsm_state = SO) in
let s_fsm_sd = (new_S_fsm_state = SSTART) in

224
let s_fsm_src = ((new_S_fsm_state = SSTART) V (S_fsm_state = SRA) \ S_fsm_delay6) \\
V (new_S_fsm_state = SCOF) V (new_S_fsm_state = ST) V (new_S_fsm_state = SC1F) \\
V (new_S_fsm_state = SS) V (S_fsm_state = SCS) \ S_fsm_delay6) in \\
let s_fsm_sec = (((\(new_S fsm_state = SST0\)) \ S_fsm_state = SO)) V (S_fsm_state = SN)) in \\
let s_fsm_srs = ((S_fsm_state = SPF) \ S_fsm_rst) V (S_fsm_state = ST) V (new_S_fsm_state = SS) V (S_fsm_state = SCS) \ S_fsm_delay6) in \\
let new P_addr -- ((-P_rqt) => (SUBARRAY L_ad_in (25,0)) \ P_addr) in \\
let new_P_destl = ((~P_rqt) => (ELEMENT L_ad_in (31)) \ P_destl) in \\
let newP__be._ = ((~P_rqt) => L_be_ \ P_be_) in \\
let new P_wr = ((-P_rqt) => L_wr \ P_wr) in \\
let newP_size = ((-P._rqt) => (SUBARRAY L_ad_in (1,0)) \ P_size) in \\
let new_C_holdA_ = ((CikD) \ C_holdA_) in \\
let i_cale_ = -((new_C_mfsm_state = CMA3) A (new_P_fsm_state = PA) \ newC_holdA_) in \\
let new_M_count = (((new_M_fsm_state = MA) \ new_M_fsm_state = MFW)) => ((M_se) => (WORDN 1) \ (WORDN 2))) in \\
let m_rdy = ((new_M_fsm_state = MW) \ new_M fsm_count) => (DECN 2 M_count) \ M_count) in \\
let m_srdy_ = ((M_rdy \ M_wr) \ P_male_) in \\
let new_M_fsm_state = ((new_M_fsm_state = MW) \ new_M_fsm_state = MW) => (SUBARRAY new P_addr (25,24)) \ new_M_fsm_state = MABT)) in \\
let new_M_count = (new_M_fsm_state = MD) \ m_srdy_ \ \\
let new_P_rqt = ((p_ale \ p_sack \ new_P_fsm_state = (P_size = ((P_down) => (P_size) \ new_P_rqt) \ (P_size) \ ARB))) in \\
let new_P_down = ((new_P_fsm_state = PD) \ new_P_addr = (P_size) \ P_down) \ P_down) \ ARB) in \\
let new_P_male_ = ((new_P_fsm_state = PD) \ P_male_) in \\
let new_P_rale_ = ((new_P_fsm_state = PM) \ new_P_rqt) \ P_rale_) in \\
let new_P_lock_ = \\
let new_P_lock_inh = \\
let pod31_27 = (MALTER ARBN (31,27) new_P_be_) in \\
let pod31_26 = (MALTER pod31_27 (26) F) in \\
let pod31_24 = (MALTER pod31_26 (25,24) (SUBARRAY new P_addr (1,0))) in \\
let new_C_iad_ea_s_delA = (CikD) => C_iad_out A C_iad_en_s_delA in \\
let new_C_iad_out = (((new_C_fsm_state = CMABT)) \ ((new_C_fsm_state = CMABT)) \ ((new_C_fsm_state = CMABT)) \ (ELEMENT new_C_iad_out (31,22))) \ C_iad_out) in \\
let r_reg_sel = ((-R_srdy_del_) => (INCN 3 R_reg_sel) \ R_reg_sel) in \\
225
let new_R_icr = 
((R_icr_load) => 
 (-(r_reg.sel = (WORDN 1))) => (And rep (R_icr_old, R_icr_mask)) I (Or rep (R_icr_old, R_icr_mask)))) ! R_icr) in
let new_R_busALatch = 
((R_c0_irend) => R_c0_in I 
 (r_reg_ordeu) => R_c0_out I 
 (r_reg1_irend) => R_c01_in I 
 (r_reg2_irend) => R_c02_in I 
 (r_reg3_irend) => R_c03_in I 
 (r_reg4_irend) => new_R_icr I 
 (r_reg5_irend) => R_c5 I 
 (r_reg6_irend) => r_c6 I 
 (r_reg7_irend) => R_ctr2_in I 
 (r_reg8_irend) => R_c8 I 
 (r_reg9_irend) => R_ctr3_in I 
 (r_reg10_irend) => R_c10 I 
 (r_reg11_irend) => R_c11 I 
 (r_reg12_irend) => R_c12 I 
 (r_reg13_irend) => R_c13 I 
 (r_reg14_irend) => R_c14 I 
 (r_reg15_irend) => R_c15 I 
 (r_reg16_irend) => R_c16 I 
 (r_reg17_irend) => R_c17 I 
 (r_reg18_irend) => R_c18 I 
 (r_reg19_irend) => R_c19 I 
 (r_reg20_irend) => R_c20 I 
 (r_reg21_irend) => R_c21 I 
 (r_reg22_irend) => R_c22 I 
 (r_reg23_irend) => R_c23 I 
 (r_reg24_irend) => R_c24 I 
 (r_reg25_irend) => R_c25 I 
 (r_reg26_irend) => R_c26 I 
 (r_reg27_irend) => R_c27 I 
 (r_reg28_irend) => R_c28 I 
 (r_reg29_irend) => R_c29 I 
 (r_reg30_irend) => R_c30 I 
 (r_reg31_irend) => R_c31 I) => L_ad I
I (new_C_iad_en_s.delA V (new_C_mfsm_state = CMD1) A -c_new_write A c_srdy_en V (new_C_mfsm_state = CMD0) A c_new_write V (new_C_mfsm_state = CSALE) A c_new_write V (new_C_mfsm_state = CSRR) A c_new_write V (new_C_mfsm_state = CSD1) A c_new_write V (new_C_mfsm_state = CSD0) A c_new_write V (new_C_mfsm_state = CSD3) A c_new_write V (new_C_mfsm_state = CSACK) A c_new_write) => new_C_iad_out I

M_wr A (new_M_fsm_state = M1) => M_rd_data I 

(-R_wr A (new_R_fsm_state = RA) V (new_R_fsm_state = RD)) => new_R_busALatch I ARB) in
let disable_writes = (-(new_C_fsm_state = CSI) V (new_C_fsm_state = CSL)) A 
(CHANNELID = (WORDN 0)) A ELEMENT C_source (6)) A 
(CHANNELID = (WORDN 1)) A ELEMENT C_source (7)) A 
(CHANNELID = (WORDN 2)) A ELEMENT C_source (8)) A 
(CHANNELID = (WORDN 3)) A ELEMENT C_source (9))) in
let i_rule = 
((-new_P_fsm_state = PH) => 
 ~(new_P_fsm_state = PH)) A 
((SUBARRAY new_P_addr (25,24)) = (WORDN 3)) A (new_P_fsm_state = PA) A new_P_reg) I 
(CHANNELID = (WORDN 3)) A ELEMENT C_source (9))) in
let new_R_wr = ((~i_rule_.) => (ELEMENT i_ad (27)) I R_wr) in
let r_writeB = (new_R_wr A (new_R_fsm_state = RA)) in
let new_R_gcr = (r_writeB A (r_reg.sel = (WORDN 2))) => i_ad I R_gcr) in
let new_R_gcrd = (r_readB A (r_reg.sel = (WORDN 2))) in
let gcr = (ELEMENT new_R_gcr (0)) in
let gcrh = (ELEMENT new_R_gcr (1)) in
let reset_error = (ELEMENT new_R_gcr (24)) in
let pin_invalid = (ELEMENT new_R_gcr (28)) in
let cout_sel0 = (ALTER ARBN (0) ((new_C_fsm_state = CSD1) V (new_C_fsm_state = CSD0)) => 
 (new_C_fsm_state = CSD1) I 
 (new_C_fsm_state = CMA3) V (new_C_fsm_state = CMA1) V (new_C_fsm_state = CMD1)) in
let cout_sel = (ALTER cout_sel0 (1) ((new_C_fsm_state = CSD1) V (new_C_fsm_state = CSD0)) => 

226
let newC_hold_ = (newC_sfsm._state = CSI) in
let newC_wr = (((~i.cale_) => (ELEMENT i.ad (27)) | C.wr) in
let newC clkA = ClkD in
let i_last_ =
  (newP_fsm_state = PH) =>
  (P.size = ((P.down) => (WORDN 1) | (WORDN 0))))
  C_last_out_ in
let newC_last_in_ = ((reset cport) =>
  C_last_in_ in
let newC ss = (((~(newC mfsm_state = CMABT)) & (newC_mfsm_state = CMD1)) =>
  newC_ss in | C_ss) in
let newC_last_out_ =
  (newC_sfm_state = CSA1) & (ClkD & ((CB_ms_in = ^MEND) V (CB_ms_in = ^MABORT))) =>
  T in
  ((~(newC_sfm_state = CSA1) & (ClkD & ((CB_ms_in = ^MEND) V (CB_ms_in = ^MABORT)))) =>
  F in
  ((~(newC_sfm_state = CSA1) & (ClkD & ((CB_ms_in = ^MEND) V (CB_ms_in = ^MABORT)))) =>
  C_last_out_ | ARB) in
let c_srdy = (CB_ss.in = ^SRDY) in
let c_dfsm_master = ((newC_mfsm_state = CMA3) V (newC_mfsm_state = CMA2) V (newC_mfsm_state = CMA1)
  V (newC_mfsm_state = CMA0) V (newC_mfsm_state = CMD1) V (newC_mfsm_state = CMD0)) in
let c_dfsm_cad_en = ((newC_mfsm_state = CMA3) V (newC_mfsm_state = CMA2) V (newC_mfsm_state = CMA1)
  V (newC_mfsm_state = CMA0) V (newC_mfsm_state = CMD1) V (newC_mfsm_state = CMD0)) in
let c__new_write = (c_new_write = (newC_mfsm_state = CMA3) V (newC_mfsm_state = CMA2) V (newC_mfsm_state = CMA1)
  V (newC_mfsm_state = CMA0) V (newC_mfsm_state = CMD1) V (newC_mfsm_state = CMD0)) in
let c_state = (c_state = (newC_mfsm_state = CMA3) V (newC_mfsm_state = CMA2) V (newC_mfsm_state = CMA1)
  V (newC_mfsm_state = CMA0) V (newC_mfsm_state = CMD1) V (newC_mfsm_state = CMD0)) in
let c_parity = (c_parity = (newC_mfsm_state = CMA3) V (newC_mfsm_state = CMA2) V (newC_mfsm_state = CMA1)
  V (newC_mfsm_state = CMA0) V (newC_mfsm_state = CMD1) V (newC_mfsm_state = CMD0)) in
let c_pe = (c_pe = (newC_mfsm_state = CMA3) V (newC_mfsm_state = CMA2) V (newC_mfsm_state = CMA1)
  V (newC_mfsm_state = CMA0) V (newC_mfsm_state = CMD1) V (newC_mfsm_state = CMD0)) in
let c_pe_cnt = (c_pe_cnt = (newC_mfsm_state = CMA3) V (newC_mfsm_state = CMA2) V (newC_mfsm_state = CMA1)
  V (newC_mfsm_state = CMA0) V (newC_mfsm_state = CMD1) V (newC_mfsm_state = CMD0)) in
let newCparity =
  (CIkD & c_pe & c_pe_cnt) =>
  (reset_error) =>
  (newC_mfsm_state = CMA3) V (newC_mfsm_state = CMA2) V (newC_mfsm_state = CMA1)
  V (newC_mfsm_state = CMA0) V (newC_mfsm_state = CMD1) V (newC_mfsm_state = CMD0)
let new_C_source =
  ((reset_cport) => (WORDN 0) |
    ((ClkD ∧ ((new_C_sfm_state = CMD1) V (new_C_sfm_state = CSL))) => Par_Dec rep (CB_ad_in) I C_source)) in
let data_in31_16 =
  (MALTER ARBN (31,16) ((reset_cport) => (WORDN 0) |
    ((ClkD ∧ ((new_C_sfm_state = CMD1) V (new_C_sfm_state = CSL))) => Par_Dec rep (CB_ad_in) I C_source)) in
let new_C_data_in =
  ((new_C_cin_2_le) => Par_Dec rep (CB_ad_in)) I
let new_C_iad_m = ((new_C_cout_OJe_del) => i_ad I C_iad_in) in
let new_C_alaO =
  (((c_dfsm_master A C cout_O le del) V (new_C_sfm_state = CSNI)) => C_iad_in I C_alaO) in
let new_C_a3a2 =
  ((new_C_m.fsm_state = CMR) => R_ccr I C_a3a2) in
let i be = ((new_P fsm_state = PA) => new_P_be_ I
  (new_P_fsm_state = PD) => L_be_ I SUBARRAY new_C_sizewrbe (9,6)) in
let imale_ =
  ((~(new_P_fsm state = PH) => (SUBARRAY i_ad (18,0))) |
    (INCN 18 M_addr) I M_addr)) in
let new_M_addr =
  ((~i_male_) => (SUBARRAY i_ad (18,0)) |
    (INCN 18 M_addr) I M_addr)) in
let new_M_be = ((~m_srdy_ A (~(new_M_fsm state = MI)) => T) I
  (m_srdy_ A (~m_srdy_ => F)) in
let r_writeA = (~disable_writes A R_wr A (new_R_fsm_state = RD)) in
let r_readA = (~R_wr A (new_R_fsm_state = RA)) in
let r_cir_wr01A = ((r_writeA A (r_reg_sel = (WORDN 8)) V (r_reg_sel = (WORDN 9)))) in
let r_cir_wr01B = ((r_writeB A (r_reg_sel = (WORDN 8)) V (r_reg_sel = (WORDN 9)))) in
let r_cir_wr23A = ((r_writeA A (r_reg_sel = (WORDN 10)) V (r_reg_sel = (WORDN 11))) in
let r_cir_wr23B = ((r_writeB A (r_reg_sel = (WORDN 10)) V (r_reg_sel = (WORDN 11))) in
let new_R_ctrl_orden = (r_writeB A (r_reg_.sel = (WORDN 10))) => R_ctr2_in I R_ctr2_new in
let new_R_ctrl0_in = ((r_writeB A (r_reg_sel = (WORDN 8)))) => i_ad I R_ctrl0_in in
let new_R_ctrl0mux_sel = (r_cir_wr0B V ((ELEMENT new_R_gcr (16)) \ R_ctr1_cry)) in
let new_R_ctrl0_orden = (r_readB A (r_reg_sel = (WORDN 8))) in
let new_R_ctrl0 = (R_ctrl0.mux_sel => R_ctrl0_in I R_ctrl0_new) in
let new_R_ctrl0_new = ((ELEMENT new_R_gcr (19))) => (INCN 31 R_ctr0) I R_ctr0 in
let new_R_ctrl0_cry = ((ONES 31 R_ctr0) \ (ELEMENT new_R_gcr (19))) in
let new_R_ctrl0_out = ((r fsm_clk latch) => R_ctrl0_new I R_ctrl0_out in
let new_R_ctrl0_order = ((r_readB A (r_reg_sel = (WORDN 12))) in
let new_R_ctr1_in = ((r_writeB A (r_reg_sel = (WORDN 9))) => i_ad I R_ctr1_in in
let new_R_ctr1mux_sel = (r_cir_wr0B V ((ELEMENT new_R_gcr (16)) \ R_ctr1_cry)) in
let new_R_ctr1_orden = (r_readB A (r_reg_sel = (WORDN 9))) in
let new_R_ctr1 = ((R_ctr1.mux_sel => R_ctr1_in I R_ctr1_new) in
let new_R_ctr1_new = ((R_ctr0_cry) => (INCN 31 R_ctr1) I R_ctr1 in
let new_R_ctr1_cry = ((ONES 31 R_ctr1) \ R_ctr0_cry) in
let new_R_ctr1_out = ((R_clk latch_del) => R_ctr1_new I R_ctr1_out in
let new_R_ctr1_order = (r_readB A (r_reg_sel = (WORDN 13))) in
let new_R_ctr2_in = ((r_writeB A (r_reg_sel = (WORDN 10))) => i_ad I R_ctr2_in in
let new_R_ctr2mux_sel = (r_cir_wr23B V ((ELEMENT new_R_gcr (20)) \ R_ctr3_cry)) in
let new_R_ctr2_orden = (r_readB A (r_reg_sel = (WORDN 10))) in
let new_R_ctr2 = (R_ctr2.mux_sel => R_ctr2_in I R_ctr2_new) in
let new_R_ctr2_new = ((ELEMENT new_R_gcr (23))) => (INCN 31 R_ctr2) I R_ctr2 in
let new_R_ctr2_cry = ((ONES 31 R_ctr2) \ (ELEMENT new_R_gcr (23))) in
let new_R_ctr2_out = ((r fsm_clk latch) => R_ctr2_new I R_ctr2_out in
let new_R_ctr2_order = (r_readB A (r_reg_sel = (WORDN 14))) in
let new_R_ctr3_in = ((r_writeB A (r_reg_sel = (WORDN 11))) => i_ad I R_ctr3_in in
let new_R_ctr3mux_sel = (r_cir_wr23B V ((ELEMENT new_R_gcr (20)) \ R_ctr3_cry)) in
let new_R_ctr3_orden = (r_readB A (r_reg_sel = (WORDN 11))) in
let new_R_ctr3 = (R_ctr3.mux_sel => R_ctr3_in I R_ctr3_new) in
let new_R_ctr3_new = (R_ctr2_cry) => (INCN 31 R_ctr3) I R_ctr3 in
let new_R_ctr3_cry = ((ONES 31 R_ctr3) \ R_ctr3_cry) in
let new_R_ctr3_out = ((R_clk latch_del) => R_ctr3_new I R_ctr3_out in
let new_R_ctr3_order = (r_readB A (r_reg_sel = (WORDN 15))) in
let new_R_icr_load = (r_writeB A (r_reg_sel = (WORDN 0)) V (r_reg_sel = (WORDN 1))) in
let new_R_icr_old =
(r_writeB A (r_reg_sel = (WORDN 0)) V (r_reg_sel = (WORDN 1))) => R_icr \ R_icr_old in
let new_R_icr_mask =
(r_writeB A (r_reg_sel = (WORDN 0)) V (r_reg_sel = (WORDN 1))) => i_ad I R_icr_mask in
let new_R_icr_irden = ((new_R_fsm_state = RA) \ ((r_reg_sel = (WORDN 0)) V (r_reg_sel = (WORDN 1)))) in
let r_int0_en = (((ELEMENT R_icr (0)) \ (ELEMENT R_icr (8))) \ ((ELEMENT R_icr (1)) \ (ELEMENT R_icr (9))) \ ((ELEMENT R_icr (2)) \ (ELEMENT R_icr (10))) \ ((ELEMENT R_icr (3)) \ (ELEMENT R_icr (11))) \ ((ELEMENT R_icr (4)) \ (ELEMENT R_icr (12))) \ ((ELEMENT R_icr (5)) \ (ELEMENT R_icr (13))) \ ((ELEMENT R_icr (6)) \ (ELEMENT R_icr (14))) \ ((ELEMENT R_icr (7)) \ (ELEMENT R_icr (15)))) in

let new_R_int0_dis = r_int0_en in

let r_int3_en = (((ELEMENT R_icr (16)) \ (ELEMENT R_icr (24))) \ ((ELEMENT R_icr (17)) \ (ELEMENT R_icr (25))) \ ((ELEMENT R_icr (18)) \ (ELEMENT R_icr (26))) \ ((ELEMENT R_icr (19)) \ (ELEMENT R_icr (27))) \ ((ELEMENT R_icr (20)) \ (ELEMENT R_icr (28))) \ ((ELEMENT R_icr (21)) \ (ELEMENT R_icr (29))) \ ((ELEMENT R_icr (22)) \ (ELEMENT R_icr (30))) \ ((ELEMENT R_icr (23)) \ (ELEMENT R_icr (31)))) in

let new_R_int3_dis = r_int3_en in

let new_S_soft_shot_del = (-gcrh \ gcrl) in

let s_soft_cat_out = ((s_fsm_srs) => ((gcrl \ ~gcrh \ -S_soft_shot_del) \ (WORDN 1) \ (WORDN 0)) \ ((gcrl \ ~gcrh \ -S_soft_shot_del) \ (INCN 2 S_soft_cnt) \ S_soft_cnt)) in

let s_delay_out = ((s fsm_src \ (s fsm_sca \ (ELEMENT S_delay (6)))) => ((s fsm_sec) \ (WORDN 1) \ (WORDN 0)) \ ((s fsm_sec) \ (INCN 17 S_delay) \ S_delay)) in

let new_S_delay = s_delay_out in

let s_cpu0_ok = (s fsm_sca \ Failure_0 \ (s_soft_cat_out = (WORDN 5))) in

let new_S_pmm_fail = ((s fsm_sbf \ (s fsm_spmf) \ T) \ ((s fsm_sbf \ (s fsm_spmf) \ F) \ ((s fsm_sbf \ (s fsm_spmf) \ S_pmm_fail \ ARB))) in

let new_S_cpu0_fail = ((s fsm_sbf \ (s cpu0 ok V Bypass)) \ T) \ ((s fsm_sbf \ (s cpu0 ok V Bypass)) \ F) \ ((s fsm_sbf \ (s cpu0 ok V Bypass)) \ S_cpu0_fail \ ARB)) in

let new_S_cpu1_fail = ((s fsm_sbf \ (s cpu1 ok V Bypass)) \ T) \ ((s fsm_sbf \ (s cpu1 ok V Bypass)) \ F) \ ((s fsm_sbf \ (s cpu1 ok V Bypass)) \ S_cpu1_fail \ ARB)) in

let new_S_piu_fail = ((s fsm_sbf \ (s fsm_spf V Bypass)) \ T) \ ((s fsm_sbf \ (s fsm_spf V Bypass)) \ F) \ ((s fsm_sbf \ (s fsm_spf V Bypass)) \ S_piu_fail \ ARB)) in

let s_cpu0_select = (s fsm_ssa V s fsm_sso) \ S_cpu0 fail in

let s_cpu0_select = (s fsm_ssa V s fsm_sso) \ S_cpu0 fail \ S_cpu1 fail in

let new_S_bad_cpu0 = ((s fsm_sbf \ ~s_cpu0 select) \ T) \ ((s fsm_sbf \ ~s_cpu0 select) \ F) \ ((s fsm_sbf \ ~s_cpu0 select) \ S_bad_cpu0 \ ARB)) in

230
let new_S_bad_cpu1 = 
((s_fsm_sb \land \neg s_cpu1_select) => T \
((s_fsm_sb \land s_cpu1_select) => F \
((\neg s_fsm_sb \land \neg s_cpu1_select) => S_bad_cpu1 \land ARB)) in
let new_S_reset_cpu0 = (new S_bad_cpu0 \land s_fsm_src0) in
let new_S_reset_cpu1 = (new S_bad_cpu1 \land s_fsm_src1) in
let new_S_cpu_hist = (S_reset_cpu0 \land S_reset_cpu1 \land Bypass) in
let ss0 = (ALTER ARBN (0) ((new_S_fsm_state = SS) \lor (new_S fsm_state = SSTOP) 
\lor (new_S fsm_state = SC) \lor (new_S fsm_state = SN) 
\lor (new_S fsm_state = SO))) in
let ss1 = (ALTER ss0 (1) ((new_S fsm_state = SC0F) \lor (new_S fsm_state = ST) 
\lor (new_S fsm_state = SCIF) \lor (new_S fsm_state = SSTOP) 
\lor (new_S fsm_state = SC))) in
let ss2 = (ALTER ss1 (2) ((new_S fsm_state = SPF) \lor (new_S fsm_state = SC0I) 
\lor (new_S fsm_state = SC0F) \lor (new_S fsm_state = ST) 
\lor (new_S fsm_state = SSTOP) \lor (new_S fsm_state = SO))) in
let ss3 = (ALTER ss2 (3) ((new_S fsm_state = SRA) \lor (new_S fsm_state = SPF) 
\lor (new_S fsm_state = ST) \lor (new_S fsm_state = SCII) 
\lor (new_S fsm_state = SC) \lor (new_S fsm_state = SN) 
\lor (new_S fsm_state = SO))) in

let s_state = ss3 in
let sr28 = (ALTER ARBN (28) new_M_parity) in
let sr28_25 = (MALTER sr28 (27,25) new_C_ss) in
let sr28_24 = (ALTER sr28_25 (24) new_C_parity) in
let sr28_22 = (MALTER sr28_24 (23,22) ChannelID) in
let sr28_16 = (MALTER sr28_22 (21,16) id) in
let sr28_12 = (ALTER sr28_16 (15,12) s_state) in
let sr28_9 = (ALTER sr28_12 (9) new_S_pmm_fail) in
let sr28_8 = (ALTER sr28_9 (8) new_S_piup_fail) in
let sr28_3 = (ALTER sr28_8 (3) new_S_reset_cpu1) in
let sr28_2 = (ALTER sr28_3 (2) new_S_reset_cpu0) in
let sr28_1 = (ALTER sr28_2 (1) new_S_cpu1_fail) in
let sr28_0 = (ALTER sr28_1 (0) new_S_cpu0_fail) in
let new_R_st = ((r_fsm_ctlatch) => sr28_01R_sr) in
let new_R_sr_rden = (r_readB \land (r_reg_sel = (WORDN 4))) in

let new_P fsm rst = reset_piu in
let new_P fsm_sack = p_sack in
let new_P fsm_cgnl_ = (new_C_mfsm_state = CMA3) in
let new_P fsm_hold_ = new_C_holdA_ in
let new_C_mfsm_D = ClkD in
let new_C_mfsm_rst = reset_cport in
let new_C_mfsm_cqrt_ = (new_P_destl \land new_P_rqt) in
let new_C_mfsm_hold_ = new_C_holdA_ in
let new_C_mfsm_ss = CB_ss_in in
let new_C_mfsm_invalid = piu_invalid in
let new_C_sfm_D = ClkD in
let new_C_sfm_rst = reset_cport in
let new_C_sfm_hlda_ = (new_P_fsm_state = PH) in
let new_C_sfm_ms = CB_ms_in in
let new_C_sfm_cale_ = i_cale_in
let new_C_sfm_last = i_last_in
let new_C_sfm_male_ = i_male_in

231
let new_C_efsm_rale_ = i_rale_ in
let new_C_efsm_srdy_ = i_srdy_ in
let new_C_efsm_rst = reset_cport in
let new_M_fsm_male_ = i_male_ in
let new_M_fsm_last = i_last_ in
let new_M_fsm_mrdy_ = ((P fsm_state = PH)) => (C_mrdy_del) in
let new_M_fsm_rst = reset_piu in
let new_R_fsm_male_ = i_male_ in
let new_R_fsm_last = i_last_ in
let new_R_fsm_mrdy_ = ((P fsm_state = PH)) => (C_mrdy_del) in
let new_R_fsm_rst = reset_piu in
let new_S_fsm_rst = Rst in
let new_S_fsm_delay6 = (ELEMENT s_delay_out (6)) in
let new_S_fsm_delay17 = (ELEMENT s_delay_out (6)) (ELEMENT s_delay_out (17)) in
let new_S_fsm_bothbad = (new_S_cpu0 fail & new_S_cpu1 fail) in
let new_S_fsm_bypass = Bypass in

(new_P_addr, new_P_dest1, new_P_be, new_P_wr, new_P fsm_state, new_P fsm_rst, new_P fsm_sack,
  new_P fsm_cqnt, new_P fsm_hold, new_P_rq, new_P size, new_P down, new_P lock, new_P lock inh,
  new_P male, new_P rale,
new_C_mfsm_state, new_C_mfsm_D, new_C_mfsm_rst, new_C_mfsm_cqnt, new_C_mfsm_hold, new_C_mfsm_ss,
new_C_mfsm_invalid, new_C_sfsm_state, new_C_sfsm_D, new_C_sfsm_rst, new_C_sfsm_hlda, new_C_sfsm_ms,
new_C_efsm_state, new_C_efsm_cale, new_C_efsm_last, new_C_efsm_male, new_C_efsm_rale, new_C_efsm_srdy,
new_C_efsm_rst, new_C wr, new_C sizewrbe, new_C clkA, new_C last_in, new_C lock io, new_C ss,
new_C_last_out, new_C hold, new_C holdA, new_C cin2 le, new_C cin2 le, new_C mrdy_del,
new_C_iad_en_s del, new_C_iad_en_r delA, new_C wrdy, new_C rdy, new_C parity, new_C source, new_C data in,
new_C iad out, new_C iad in, new_C a1a0, new_C a3a2,
new_M_fsm_state, new_M_fsm_male, new_M_fsm_last, new_M_fsm_mrdy, new_M_fsm_rst, new_M fsm cnt,
new_M se, new_M wr, new_M addr, new_M be, new_M rdy, new_M wwdel, new_M parity, new_M rd data,
new_M detect,
new_R_fsm_state, new_R_fsm_male_, new_R_fsm_mrdy_, new_R_fsm_last, new_R_fsm_rst, new_R ctr0_in,
new_R ctr0 mux sel, new_R ctr0, new_R ctr0 irden, new_R ctr0 cry, new_R ctr0 out,
new_R ctr0_orden, new_R ctr1_in, new_R ctr1 mux sel, new_R ctr1, new_R ctr1 irden, new_R ctr1 new,
new_R ctr1 cry,
new_R ctr1 out, new_R ctr1_orden, new_R ctr2 in, new_R ctr2 mux sel, new_R ctr2, new_R ctr2 irden,
new_R ctr2 new,
new_R ctr2 cry, new_R ctr2 out, new_R ctr2_orden, new_R ctr3_in, new_R ctr3 mux sel, new_R ctr3,
new_R ctr3 irden,
new_R ctr3 new, new_R ctr3 cry, new_R ctr3 out, new_R ctr3_orden, new_R icr load, new_R icr old,
new_R icr mask,
new_R icr rden, new_R icr, new_R ccr, new_R ccr rden, new_R gcr, new_R gcr rden, new_R sr, new_R sr rden,
new_R int0 dis, new_R int3 dis, new_R c01 cout_del, new_R int1 en, new_R c23 cout del, new_R int2 en,
new_R wr,
new_R cuilatch del, new_R mrdy del, new_R reg sel, new_R busA latch,
new_S_fsm_state, new_S fsm rst, new_S fsm_delay6, new_S fsm delay17, new_S fsm bothbad,
new_S fsm bypass, new_S soft shot del, new_S soft cnt, new_S delay, new_S bad cpu0, new_S bad cpu1,
new_S reset cpu0, new_S reset cpu1, new_S cpu_hist, new_S pmm fail, new_S cpu0 fail, new_S cpu1 fail,
new_S piu fail)

);
let piuEXEC_out_def = new_definition
("piuEXEC_out",

"!
(rep:\rep_ty

(P fsm state :fsfm_ty)
(P addr :Word)
(P dest1 P wr P fsm rst P fsm sack P fsm cgnt P fsm hold P rqt P down P lock
P lock inb P male P rale :bool)
(C mfsm state :csfm_ty) (C fsm state :csfm_ty) (C efsm state :csfm_ty)
(C mfsm ss C fsm ms C size wr C ss C source C data in C iad out C iad in C a1s0 C a3a2
C fsm rstate C efsm male C efsm rstate C efsm rstate
C mw C rstate C lock in C last out C hold A C rstate C 0 le C del C cin 2 C
C mrdy del C iad en s del C iad en s del A C wrdy C rdy C parity :bool)
(M fsm state :mfsm_ty)
(M count M addr M be M rd data M detect :word)
(M fsm male M last M fsm mrdy M fsm rst M se M wr M rdy M wwdel M parity :bool)
(R fsm state :rfsm_ty)
(R ctrl0 new R ctrl0 out R ctrl1 new R ctrl1 out R ctrl2 new R ctrl2 out
R ctrl3 new R ctrl3 out R icr old R icr mask R icr ccr R gcr R sr
R reg sel R busa latch :word)
(S fsm state :sfsm_ty)
(S soft cnr S delay :word)
(S fsm rst S fsm delay17 S fsm both bad S fsm bypass S soft shor C del S bad cpu0 S bad cpu1
S reset cpu0 S reset cpu1 S cpu hist S pmm fail S cpu0 fail S cpu1 fail S piu fail :bool)

(L ad in L be :word)
(ClkA ClkB Rst L ads L den L wr L lock :bool)
(CB rqt in CB ad in CB ms in CB ss in Id ChannelID :word)
(ClkD :bool)
(MB data in :word)
(Edac_eo :bool)
(Bypass Test Failure0 Failure1 :bool)

piuEXEC out rep

(P addr, P dest1, P be, P wr, P fsm state, P fsm rst, P fsm sack, P fsm cgnt, P fsm hold
P rqt, P size, P down, P lock, P lock inb, P male, P rale,
C mfsm state, C mfsm D, C mfsm rst, C mfsm cgnt, C mfsm hold, C mfsm ss, C mfsm invalid,
C fsm state, C fsm D, C fsm rst, C fsm hlda, C fsm ms,
C efsm state, C efsm D, C efsm rstate, C efsm male, C efsm rstate, C efsm rstate
C wr, C size wr, C clk A, C last in, C lock in, C ss, C last out,
C hold A C rstate C 0 le C del C cin 2 C mrdy del C iad en s del C iad en s del A,
C wrdy, C rdy, C parity, C source, C data in, C iad out, C iad in, C a1s0 C a3a2,
M fsm state, M fsm male, M fsm last, M fsm mrdy, M fsm rst, M count, M se, M wr, M addr,
M be, M rdy, M wwdel, M parity, M rd data, M detect,
R fsm state, R fsm ale, R fsm mrdy, R fsm last, R fsm rst, R ctrl0 in, R ctrl0mux sel, R ctrl0,
R ctrl0 inren, R ctrl0 new, R ctrl0 cry, R ctrl0 out, R ctrl0 orden, R ctrl0 in, R ctrl1mux sel,
R ctrl1, R ctrl1 inren, R ctrl1 new, R ctrl1 cry, R ctrl1 out, R ctrl1 orden, R ctrl2 in, R ctrl2mux sel,
R ctrl2, R ctrl2 inren, R ctrl2 new, R ctrl2 cry, R ctrl2 out, R ctrl2 orden, R ctrl3 in, R ctrl3mux sel,
R ctrl3, R ctrl3 inren, R ctrl3 new, R ctrl3 cry, R ctrl3 out, R ctrl3 orden, R icr load, R icr old,
R icr mask, R icr rden, R icr ccr, R gcr, R gcr rden, R sr, R sr rden, R int0 dis,
let new_P_fsm_state =
((P_fsm_rst) => PA |
((P_fsm_state = PH) => ((-P_fsm_hold_) => PH | PA) | 
((P_fsm_state = PA) =>
  ((P_rq_v -P_dest1) V (P_rq_v P_dest1 -P_fsm_cntl)) => PD | 
  ((-P_fsm_hold_ P_lock_) => PH | PA) | 
((P_fsm_state = PD) =>
  ((P_fsm_sack -P_fsm_hold_) V (P_fsm_sack -P_fsm_hold_ -P_lock_) => PA | 
  ((P_fsm_sack -P fsm_hold_ -P lock_) => PH | PD)) | P_LL))) in

let c_write = (((-C_m fsm_state = CMI)) | (-C_m fsm_state = CMR)) => C_writer (ELEMENT C_size wrbe (5)) in
let c_busy = (-((SUBARRAY CB_rq_bot_ (3,1)) = (WORDN 7))) in
let c_grant = (((SUBARRAY Id (1,0)) = (WORDN 0)) | -(ELEMENT CB_rq_bot_ (0)))
  V (((SUBARRAY Id (1,0)) = (WORDN 1)) | -(ELEMENT CB_rq_bot_ (0))
  | (ELEMENT CB_rq_bot_ (1)) | (ELEMENT CB_rq_bot_ (2)))
  V (((SUBARRAY Id (1,0)) = (WORDN 2)) | -(ELEMENT CB_rq_bot_ (0))
  | (ELEMENT CB_rq_bot_ (1)) | (ELEMENT CB_rq_bot_ (2)))
  V (((SUBARRAY Id (1,0)) = (WORDN 3)) | -(ELEMENT CB_rq_bot_ (0))
  | (ELEMENT CB_rq_bot_ (1)) | (ELEMENT CB_rq_bot_ (2)) | (ELEMENT CB_rq_bot_ (3)) in

let c_addressed = (Id = (SUBARRAY C_source (15,10))) in
let new_C_m fsm_state =
((C_m fsm rst) => CMI |
  (C_m fsm state = CMI) =>
  (C_m fsm D -C_m fsm req -c_busy -C_m fsm invalid) => CMR CMI |
  (C_m fsm state = CMR) => (C_m fsm D C_grant C_m fsm hold_) => CMA3 CMI |
  (C_m fsm state = CMA3) => (C_m fsm D => CMA1 | CMA3) |
  (C_m fsm state = CMA1) =>
  (C_m fsm D (C_m fsm ss ^SRDY)) => CMA0 |
  (C_m fsm D (C_m fsm ss ^SABORT)) => CMABT | CMA1 |
  (C_m fsm state = CMA0) =>
  (C_m fsm D (C_m fsm ss ^SRDY)) => CMA1 |
  (C_m fsm D (C_m fsm ss ^SABORT)) => CMABT | CMA0 |
  (C_m fsm state = CMA2) =>
  (C_m fsm D (C_m fsm ss ^SRDY)) => CMD1 |
  (C_m fsm D (C_m fsm ss ^SABORT)) => CMABT | CMA2 |
  (C_m fsm state = CMD1) =>
  (C_m fsm D (C_m fsm ss ^SRDY)) => CMD0 |
  (C_m fsm D (C_m fsm ss ^SABORT)) => CMABT | CMD1 |
  (C_m fsm state = CMD0) =>
  (C_m fsm D (C_m fsm ss ^SRDY) C_last_in_) => CMD1)
((C_mfsm_D \land (C_mfsm_ss = ^SRDY) \land \neg C_{last\_in_} ) \Rightarrow CMW \lor
(C_mfsm_D \land (C_mfsm_ss = ^SABORT)) \Rightarrow CMABT \lor CMD0 \lor
((C_mfsm_state = CMW) \Rightarrow
(C_mfsm_D \land (C_mfsm_ss = ^SABORT)) \Rightarrow CMABT \lor
(C_mfsm_D \land (C_mfsm_ss = ^SACK) \land C_{lock\_in_} ) \Rightarrow CMI \lor
(C_mfsm_D \land (C_mfsm_ss = ^SRDY) \land C_{lock\_in_} \land \neg C_{mfsm\_cqqt_} ) \Rightarrow CMA3 \lor CMW \lor
((\neg C_{last\_in_} ) \Rightarrow CMI \lor CMABT))) ))) in

let new_C_sfsm_state =
((C_sfsm_rst) \Rightarrow CSl \lor
(C_sfsm_state = CSl) \Rightarrow
((C_sfsm_D \land (C_sfsm_ms = ^MSTART) \land \neg c_grant \land c_addressed) \Rightarrow CSAI \lor CSl) \lor
(C_sfsm_state = CSA1) \Rightarrow
((C_sfsm_D \land (C_sfsm_ms = ^MRDY)) \Rightarrow CSA0 \lor
(C_sfsm_D \land (C_sfsm_ms = ^MABORT)) \Rightarrow CSABT \lor CSA1) \lor
(C_sfsm_state = CSA0W) \Rightarrow
((C_sfsm_D \land (C_sfsm_ms = ^MRDY) \land \neg C_{sfsm\_hlda_} ) \Rightarrow CSALE \lor
(C_sfsm_D \land (C_sfsm_ms = ^MABORT)) \Rightarrow CSABT \lor CSA0W) \lor
(C_sfsm_state = CSALE) \Rightarrow
((C_sfsm_D \land c_write \land (C_sfsm_ms = ^MRDY)) \Rightarrow CSD1 \lor
(C_sfsm_D \land \neg c_write \land (C_sfsm_ms = ^MRDY)) \Rightarrow CSRR \lor
(C_sfsm_D \land (C_sfsm_ms = ^MABORT)) \Rightarrow CSABT \lor CSALE) \lor
(C_sfsm_state = CSRR) \Rightarrow
((C_sfsm_D \land \neg (C_sfsm_ms = ^MABORT)) \Rightarrow CSD1 \lor
(C_sfsm_D \land (C_sfsm_ms = ^MABORT)) \Rightarrow CSABT \lor CSRR) \lor
(C_sfsm_state = CSD1) \Rightarrow
((C_sfsm_D \land (C_sfsm_ms = ^MRDY)) \Rightarrow CSD0 \lor
(C_sfsm_D \land (C_sfsm_ms = ^MABORT)) \Rightarrow CSABT \lor CSD1) \lor
(C_sfsm_state = CSD0) \Rightarrow
((C_sfsm_D \land (C_sfsm_ms = ^MEND)) \Rightarrow CSACK \lor
(C_sfsm_D \land (C_sfsm_ms = ^MRDY)) \Rightarrow CSD1 \lor
(C_sfsm_D \land (C_sfsm_ms = ^MABORT)) \Rightarrow CSABT \lor CSD0) \lor
(C_sfsm_state = CSACK) \Rightarrow
((C_sfsm_D \land (C_sfsm_ms = ^MRDY)) \Rightarrow CSL \lor
(C_sfsm_D \land (C_sfsm_ms = ^MWAIT)) \Rightarrow CSI \lor
(C_sfsm_D \land (C_sfsm_ms = ^MABORT)) \Rightarrow CSABT \lor CSACK) \lor
(C_sfsm_D) \Rightarrow CSl \lor CSABT) in

let new_C_efsm_state =
((C_efsm_rst) \Rightarrow CEI \lor
(C_efsm_state = CEI) \Rightarrow ((\neg C_{efsm\_cale_} ) \Rightarrow CEE \lor CEI) \lor
((\neg C_{efsm\_last_} \land \neg C_{efsm\_aridy_}) \lor C_{efsm\_male_} \lor \neg C_{efsm\_rale_} ) \Rightarrow CEI \lor CEE) in

let m_bw = ((\neg (M_be = (WORDN 15))) \land M_wr \land (\neg (M_fsm_state = MI))) in
let m_ww = ((M_be = (WORDN 15)) \land M_wr \land (\neg (M_fsm_state = MI))) in
let new_M_fsm_state =
let new_R_fsm_state =
(R_fsm_rst) => RI |
((R_fsm_state = RI) => ((-R_fsm_ale_)) => RA | RI) |
((R_fsm_state = RA) => ((-R_fsm_mrdy_)) => RD | RA) |
((-R_fsm_last_)) => RI | RA)) in
let r_fsm_cntlatch = ((R_fsm_state = RI) A -R_fsm_mrdy_) in
let r_fsm_srdy_ = ![un decipherable latex code]

let new_S_fsm_state =
(S_fsm_rst) => SSTART |
((S_fsm_state = SSTART)) => SRA |
((S_fsm_state = SRA) => ((S_fsm_delay6) => ((S_fsm_bypass)) => SO | SPF) | SRA) |
((S_fsm_state = SPF) => SC0F |
((S_fsm_state = SC0I) => ((S_fsm_delay17) => SCOF | SC0I) |
((S_fsm_state = SCOF) => ST |
((S_fsm_state = ST) => SC11 |
((S_fsm_state = SC1I) => ((S_fsm_bothbad)) => SSTOP | SC lieu) |
((S_fsm_state = SSTOP) => SSTOP |
((S_fsm_state = SCS) => ((S_fsm_delay6) => SN | SC lieu) |
((S_fsm_state = SN) => ((S_fsm_delay17) => SO | SN) |
((S_fsm_state = SO) => SO | S_ILL)))))))) in
let s_fsm_sn = (new_S_fsm_state = SN) in
let s_fsm_so = (new_S_fsm_state = SO) in
let reset_sport = ((((-new_S_fsm_state = SO)) A (-S_fsm_state = SSTOP)) V (S_fsm_state = SRA)) in
let s_fsm_scli = ((((-new_S_fsm_state = SO)) A (-S_fsm_state = SSTOP)) V (S_fsm_state = SRA)) in
let reset_pi = ((new_S_fsm_state = SSTART) V (new_S_fsm_state = SRA)
V (new_S_fsm_state = SC0F) V (new_S_fsm_state = ST)
V (new_S_fsm_state = SC1F) V (new_S_fsm_state = SC1I) V (new_S_fsm_state = SSTOP) V (new_S_fsm_state = SCS) in
let s_fsm_src0 = (((new_S_fsm_state = SC0F)) A (-new_S_fsm_state = SC0I)) in
let s_fsm_src1 = (((new_S_fsm_state = ST)) A (-new_S_fsm_state = SC1I)) in
let s_fsm_spf = ((S_fsm_state = SRA) A S_fsm_delay6 A -S_fsm_rst) in
let s_fsm_scof = (new_S_fsm_state = SC0F) in
let s_fsm_sclf = (new_S_fsm_state = SC1F) in
let s fsm spmf = (new_S_fsm_state = SO) in
let s_fsm sb = (new_S_fsm_state = SSTART) in
let s_fsm src = (((new_S_fsm_state = SSTART) V (S_fsm_state = SRA) A S_fsm_delay6)
V (new_S_fsm_state = SC0F) V (new_S_fsm_state = ST) V (new_S_fsm_state = SC1F)
```ml
V (new_S_fsm_state = SS) \land (S_fsm_state = SCS) \land S_fsm_delay6)) in
let s_fsm_sec = (((new_S_fsm_state = SSTOP) \land (S_fsm_state = SN)) \lor (S_fsm_state = ST)) \land (-S_fsm_rst) \lor (S_fsm_state = ST) \land -S_fsm_rst)) in
let s_fsm_scs = (new_S_fsm_state = SSTOP) \land (new_S_fsm_state = SO) \lor (S_fsm_state = SN) \land S_fsm_delay6)) in
let s_fsm = (((-P_rqt) => (SUBARRAY L_ad_in (25,0)) \lor (P_addr)) \lor (P_down) \lor (P_fsm_state = PA)) \lor (new_S_fsm_state = SSTOP) \land (new_S_fsm_state = SO) \lor (S_fsm_state = SN) \land S_fsm_delay6)) in
let s_fsm_rst = (((-P_rqt) => (SUBARRAY L_ad_in (25,0)) \lor (P_addr)) \lor (P_down) \lor (P_fsm_state)) \lor (new_S_fsm_state = SSTOP) \land (new_S_fsm_state = SO) \lor (S_fsm_state = SN) \land S_fsm_delay6)) in
let s_fsm_cscs = (new_S_fsm_state = SC) \land (new_S_fsm_state = SCS) \land S_fsm_delay6)) in
let s_fsm_delsy6 = (new_S_fsm_state = SSTOP) \land (new_S_fsm_state = SO) \lor (S_fsm_state = SN) \land S_fsm_delay6)) in
```

((~(r_reg sel = (WORDN 1))) => (And rep (R_icr old, R_icr mask)) | (Orn rep (R_icr old, R_icr mask)))

R_icr in
let new R_busA latch =
((R_ctr0 rden) => R_ctr0 in |
((R_ctr0 orden) => R_ctr0 out |
((R_ctr1 rden) => R_ctrl in |
((R_ctr1 orden) => R_ctr1 out |
((R_ctr2 rden) => R_ctr2 in |
((R_ctr2 orden) => R_ctr2 out |
((R_ctr3 rden) => R_ctr3 in |
((R_ctr3 orden) => R_ctr3 out |
((R_ctr3 rden) => new R_icr |
((R_ccr rden) => R_ccr |
((R_.gcr rden) => R_gcr |
((R sr_rde_)

let new R_busA latch =
((R_ctr0 rden) => R_ctr0 in |
((R_ctr0 orden) => R_ctr0 out |
((R_ctr1 rden) => R_ctrl in |
((R_ctr1 orden) => R_ctr1 out |
((R_ctr2 rden) => R_ctr2 in |
((R_ctr2 orden) => R_ctr2 out |
((R_ctr3 rden) => R_ctr3 in |
((R_ctr3 orden) => R_ctr3 out |
((R_ctr3 rden) => new R_icr |
((R_ccr rden) => R_ccr |
((R_.gcr rden) => R_gcr |
((R sr_rde_)}

let disable_writes = ((~(new C_sfsm state = PA) => pod31_24 |
((new P fsm state = FD) \ new P wr) => i_ad in |
(new C_iad_en_s delA V
((new C_mfsm state = CMD1) \ ~c_new_write \ c_srdy_en) V
((new C_mfsm state = CMD0) \ ~c_new_write \ c_srdy_en) V
((new C_mfsm state = CMW) \ (c_mfsm state = CMD0) \ ~c_new_write \ c_srdy_en) V
((new C_sfm state = CSALE) \ (~c_sfm state = CSALE)) V
((new C_sfm state = CSD1) \ ~c_new_write) V
((new C_sfm state = CSD0) \ c_new_write) V
((new C_sfm state = CSACK) \ ~c_new_write) => new C_iad out |
(M_wr \ ~new M fsm state = MI)) => M rd data |
(~R wr \ ((new R fsm state = RA) \ (new R fsm state = RD))) => new R_busA latch | ARB in

let i_rale =
((~(new P fsm state = PH)) =>
~((new P dst1 (SUBARRAY new P addr (25,24)) = (WORDN 3)) \ (new P fsm state = PA) \ new P rqt) |
~((new C_sfm state = CSALE) \ (SUBARRAY new C sizewrbe (1,0)) = (WORDN 3)) \ C clkA) in

let new R wr = ((~i_rale) => (ELEMENT i_ad (27)) \ R wr) in
let r writeB = (~disable writes \ new R wr \ (new R fsm state = RA)) in
let r readB = (~new R wr \ (new R fsm state = RA)) in
let new R gcr = ((r writeB \ (r reg sel = (WORDN 2))) => i_ad \ R gcr) in
let new R gcr rden = (r readB \ (r reg sel = (WORDN 2))) in
let gcr = (ELEMENT new R gcr (0)) in
let gcrb = (ELEMENT new R gcr (1)) in
let reset error = (ELEMENT new R gcr (24)) in
let piu_invalid = (ELEMENT new R gcr (28)) in
let cout sel0 = (ALTER ARBN (0) \ ((new C_sfm state = CSD1) \ (new C_sfm state = CSD0)) =>
(new C_sfm state = CSD1) |
(new C_mfsm state = CMA3) \ (new C_mfsm state = CMA1) \ V (new C_mfsm state = CMD1)) in
let cout sel = (ALTER cout sel0 (1) \ ((new C_sfm state = CSD1) \ (new C_sfm state = CSD0)) =>
F | (new C_mfsm state = CMA3) \ (new C_mfsm state = CMA2)) in
let new C hold = (new C sfm state = CSI) in
let new_C_wr = ((~i_cale_) => (ELEMENT i_ad (27)) | C_wr) in
let new_C_clkA = ClkD in
let i_last_ =
  (~(new_P_fsm_state = PH) =>
   (P_size = ((P_down) => (WORDN 1) | (WORDN 0))) | C_last_out_) in
let new_C_last_in_ = ((reset_cport) => F !
  ((new_C_m fsm_state = CMDABT) V (new_C_m fsm_state = CMD1) A ClkD) => i_last_ !
  C_last_in_)) in
let new_C_lock_in_ = ((reset_cport) => F !
  ((new_C_m fsm_state = CMABT) V (new_C_m fsm_state = CMDI) A ClkD) => i_last_ !
  C_last_in_)) in
let new_C_ss = (((~ (new_C_m fsm state = CMABT)) A (_(new_C n fsm__state = CMI))) => CB_ss_in !
  C._ss) in
let new_C_lastout_ = ((new_C_sfsm_state = CSA1) A ~(CIkD A ((CB_ms_in = AMEND) V
  (CB_ms_mffi MABORT)))) => T !
  ((~(new_C_sfsm_state = CSA1) A (CIkD A ((CB_ms_in = AMEND) V
  (CB_ms_in = MABORT)))) => F !
  ((~(new_C_sfsm_state = CSA1) A (CIkD A ((CB_ms_in = AMEND) V
  (CB_ms_in = MABORT)))) => C_last_out_ !
  ARB)) in
let c_srdy = (CB_ss_in = ~SRDY) in
let c_dfsm_master = ((new_C_m fsm_state = CMA3) V (new_C_m fsm_state = CMA2) V (new_C_m fsm_state = CMA1)
  V (new_C_m fsm_state = CMD1) V (new_C_m fsm_state = CMD0)) in
let c_dfsm_cad_en = ~((new_C_m fsm_state = CMA3) V (new_C_m fsm_state = CMA1) V (new_C_m fsm_state = CMA0)
  V (new_C_m fsm state = CMA2)
  V (c_new_write A ((new_C_m fsm_state = CMD1) V (new_C_m fsm_state = CMD0))
  V (c_new_write A ((new_C_s fsm state = CSD1) V (new_C_s fsm state = CSD0)))) in
let new_C_cout_0_le_del_ = (i_cale_) V (i_srdy_ A ~(c_new_write)
  V (new_C_m fsm_state = CMA0) A c_srdy A c_new_write A ClkD
  V (new_C_m fsm_state = CMD0) A c_new_write A c_srdy A ClkD)) in
let new_C_cin_2_le_ = (ClkD A (((new_C_m fsm_state = CMD0) A c_srdy A ~(c_new_write)) V
  ((new_C_s fsm_state = CSA0)) V
  ((new_C_s fsm_state = CSD0) A c_new_write))) in
let new_C_mrdy_del_ = ~((~c_new_write A ClkD A ((new_C_s fsm_state = CSAL) V (new_C_s fsm_state = CSD1))) V
  (~c_new_write A ClkD A (new_C_s fsm_state = CSACK)) V
  (c_new_write A ClkD A (new_C_s fsm state = CSD0)) in
let new_C_iad_en_s._del_ = ((c_new_write A ClkD A (((new_C_s fsm_state = CSALE) A ~(ClkD A ((CB_ms_in = AMEND) V
  (CB_ms_in = MABORT)))) => C_last_out_ ARB)) in
let new_C_wrdy = (c_srdy A c_new_write A (new_C_m fsm_state = CMD1) A ClkD) in
let new_C_rrdy = (c_srdy A ~c_new_write A (new_C_m fsm_state = CMD0) A ClkD) in
let c_pe = (Par_Det rep (CB_ad_in)) in
let c_mparity = ((new_C_m fsm_state = CMA3) V (new_C_m fsm_state = CMA1) V (new_C_m fsm state = CMA0)
  V (new_C_m fsm_state = CMA2) V (new_C_m fsm_state = CMD1) V (new_C_m fsm_state = CMD0)
  V (C_m fsm state = CMA1) V (C_m fsm_state = CMA0) V (C_m fsm_state = CMA2)
  V (C_m fsm_state = CMD1)) in
let c_sparity = ((~(new_C_s fsm state = CSl)) A (~(new_C_s fsm state = CSACK)) A (new_C_s fsm state = CSABT))) in
let c_pe_cnt = (ClkD A ((~(c_mparity = c_sparity)) V ((SUBARRAY CB_ss_in (1,0)) = (WORDN 0)))) in
let new_C_parity =
  (((ClkD A c_pe A c_pe_cnt) A ~reset_error) => T !
  ((~(ClkD A c_pe A c_pe_cnt) A reset_error) => F !
  ((~(ClkD A c_pe A c_pe_cnt) A ~reset_error) => C_parity ARB))) in
let new_C_source =
  ((reset_cport) => (WORDN 0) !
  239
let d__in31_16 = (MALTER ARBN (31,16) ((resot_cport) --> (WORDN 0))
(let (new_C_mfsm_state = CMD1) A c_srdy A -e_new_write)
V ((new_C_dsm._state = CSDI) A c_new_write) => Par_Dec rep (CB_ad_in)
I (SUBARRAY C_data_in (31,16)))) in

let newC_data_in = (MALTER data_in3 I_I (15,0)
(let (reset_cport) => (WORDN 0))) (ELEMENT i_ad (23)) I M_so
in
let new_M_so =((-imale_) => (SUBARRAY i_ad (23))) in

let new_M_rdy = m_rdy in
let new__M_wwdel = ((new_M_fsm_state = MA) A new_M_wr A (newM_be = (WORDN 15)))
in
let new_M_.rd_data = (((new_Mjsm_state = MR)) => (Ham_Dec rep M_B_data
in) I M rd data)
in
let new_M_detect = (((new M fsm state = MR) A ~new_M_wr) V new M_wr V (new M fsm state = MI))
=> ((Ham_Det2 rep (new_Mdetect, ~Edac_en_)) in

let new_R_srdy_del_ = r_fsm_srdy_ in
let new_R_reg_.sol =((-i_rale_) => (SUBARRAY i ed (3,0)),
((-R_srdy_del_) => (INCN 3 i be_)) in

let r_writeA = (-disable writes A R_wr A (new_R_fsm_state ffi RD)) in
let r...readA = (-Rwr A (new_R_fsm_state = RA)) in
let r_cir_wr01A ffi ((r._writeA A ((r_reg_sol ffi (WORDN 8)) V (r._reg_sol ffi (WORDN 9))))) in
let r_cir_wr23A = ((r_writeB A ((r_reg_sol = (WORDN 8)) V (r_reg_sel = (WORDN 9))))) in
let r_cir_wr23B = ((r_writeB A ((r_reg_sel = (WORDN 8)) V (r_reg_sel = (WORDN 9))))) in
let new_R_cir_wr01B = ((r_writeB A ((r_reg_sel = (WORDN 8)) V (r_reg_sel = (WORDN 9))))) in
let new_R_ccr = ((r..writeB A (r._reg_sol = (WORDN 3))) => lad I R_c_) in
let new_R_ccr_rden = (r_read8 A (r_reg_sol ffi (WORDN 3))) in
let new_R_cOl_cout_del = r_ctrl_cry in
let new_R_int1_en =

240
let new_R_int0_dis = r_int0_en
let r_int3_en = (((ELEMENT R._icr (16)) A (ELEMENT R_icr (24))) V (ELEMENT R_icr (17)) A (ELEMENT R icr (25))) V (ELEMENT R icr (18)) A (ELEMENT R icr (26))) V (ELEMENT R icr (19)) A (ELEMENT R icr (27))) V (ELEMENT R icr (20)) A (ELEMENT R icr (28))) V (ELEMENT R icr (21)) A (ELEMENT R icr (29))) V (ELEMENT R icr (22)) A (ELEMENT R icr (30))) V (ELEMENT R icr (23)) A (ELEMENT R licr (31))))

let new_R_int3_dis = r_int3_en

let new_S soft_shot = ((-gcrl A gcrl) A gcrl) in
let s soft_cnt_out = ((s_fsm_m) => ((gcrl A -gcrb A -S_soft_shot del) => (INCN 2 S_soft_cnt) I S_soft_cnt)) in
let new_S_delay = s_delay_out in
let s.spu0_ok = (s._fsm._so0f A Failure0_ A (s soft cnt._out = (WORDN 5))) in
let new_S_cpo0_fail = ((s_fsm_sb A -(s_cpu0._ok V Bypass)) => T I ((~s_fsm sb A -(s_cpu0 Select V Bypass)) => F I ((~s_fsm sb A -(s_cpu0 ok V Bypass)) => S_cpu0__fail I ARB))) in
let new_S__piu_fail = ((s_fsm_sb A -(s_fsm_spf V Bypass)) => T I ((~s_fsm sb A -(s_fsm spf V Bypass)) => F I ((~s fsm sb A -(s fsm spf V Bypass)) => S_piu_fail I ARB))) in
let new_S_bad_cpu0 = ((s_fsm_sb A -(s_cpu0 Select) => T I ((~s fsm sb A -(s_cpu0 Select) => F I ((~s fsm sb A -(s_cpu0 Select) => S_bad_cpu0 I ARB)))) in
let new_S_bad_cpu1 = ((s_fsm sb A -(s_cpu1 Select) => T I ((~s fsm sb A -(s_cpu1 Select) => F I ((~s fsm sb A -(s_cpu1 Select) => S_bad_cpu1 I ARB)))) in

242
let new_S_reset_cpu0 = (new_S_bad_cpu0 ∧ s_fsm_src0) in
let new_S_reset_cpu1 = (new_S_bad_cpu1 ∧ s_fsm_src1) in
let new_S_cpu_hist = (S_reset_cpu0 ∧ S_reset_cpu1 ∧ ARB) in
let ss0 = (ALTER ARBN (0) (new_S fsm_state = SS) V (new_S fsm_state = SSTOP)
V (new_S fsm_state = SCS) V (new_S fsm_state = SN)
V (new_S fsm_state = SO)) in
let ss1 = (ALTER ss0 (1) (new_S fsm_state = SCOF) V (new_S fsm_state = ST)
V (new_S fsm_state = SC11) V (new_S fsm_state = SC1F)
V (new_S fsm_state = SS) V (new_S fsm_state = SSTOP)
V (new_S fsm_state = SCS)) in
let ss2 = (ALTER ss1 (2) (new_S fsm_state = SPF) V (new_S fsm_state = SC01)
V (new_S fsm_state = SC0F) V (new_S fsm_state = ST)
V (new_S fsm_state = SSTOP) V (new_S fsm_state = SO)) in
let ss3 = (ALTER ss2 (3) (new_S fsm_state = SRA) V (new_S fsm_state = SPF)
V (new_S fsm_state = ST) V (new_S fsm_state = SC11)
V (new_S fsm_state = SCS) V (new_S fsm_state = SN)
V (new_S fsm_state = SO)) in
let s_state = ss3 in
let sr28 = (ALTER ARBN (28) new_M_parity) in
let sr28_25 = (MALTER sr28 (27,25) new_C_ss) in
let sr28_24 = (ALTER sr28_25 (24) new_C_parity) in
let sr28_22 = (MALTER sr28_24 (23,22) ChannelID) in
let sr28_16 = (MALTER sr28_22 (21,16) Id) in
let sr28_12 = (MALTER sr28_16 (15,12) s_state) in
let sr28_9 = (ALTER sr28_12 (9) new_S_pmm fail) in
let sr28_8 = (ALTER sr28_9 (8) new_S_piu fail) in
let sr28_3 = (ALTER sr28_8 (3) new_S_reset_cpu1) in
let sr28_2 = (ALTER sr28_3 (2) new_S_reset_cpu0) in
let sr28_1 = (ALTER sr28_2 (1) new_S_cpu1 fail) in
let sr28_0 = (ALTER sr28_1 (0) new_S_cpu0 fail) in
let new_R_sr = ((r fsm_CNTLATCH) => sr28_0 !R_sr) in
let new_R_sr_rden = (r_reg_sel = (WORDN 4)) in
let new_P fsm_rst = reset_piu in
let new_P fsm_sack = p_sack in
let new_P fsm_cgnl_ = (new_C n_fsm_state = CMA3) in
let new_P fsm_hold_ = new_C holdA_ in
let new_C n_fsm_D = ClkD in
let new_C n_fsm_rst = reset_cport in
let new_C n_fsm_cq_t_ = (new_P destl ∧ new_P rqt) in
let new_C n_fsm_hold_ = new_C holdA_ in
let new_C n_fsm_ss = CB ss in
let new_C n_fsm_invalid = piu invalid in
let new_C n_fsm_D = ClkD in
let new_C n_fsm_rst = reset_cport in
let new_C n_fsm_blda_ = (new_P fsm_state = PH) in
let new_C n_fsm_ms = CB ms in
let new_C n_fsm_cale_ = i cale in
let new_C n_fsm_last_ = i last in
let new_C n_fsm_male_ = i male in
let new_C n_fsm_rale_ = i rale in
let new_C n_fsm_srty_ = i srty in

243
let new_C__efsm_rst = reset_cport in
let new_M fsm_male = i_male_ in
let new_M fsm_last_ = i_last_ in
let new_M fsm_mrdy_ = (-(P_fsm_state = PH)) => F | C_mrdy_del_ in
let new_R fsm_rst = reset_piu in
let new_R fsm_mrdy_ = (-(P_fsm_state = PH)) => F | C_mrdy_del_ in
let new_S fsm_bypa_ = Bypass in
let L_ad out = (((-(new_P_fun_state = PA)) A (--(new P fsm_state = PH)) A -((new P fsm_state = PD) A new_P_wr)) => i_ad I ARBN) in
let L_ready_ = ~(-i_srdy_ A (new_P fsm_state = PD)) in
let CB_rqt_out.. = (-(new_C_mfsm_state = CMIV) in
let ms0 = (ALTER ARBN (0) (((new_C_mfsm_state = CMD0) A -C_last_in_) V ((new_C_mfsm_state = CMW) A C_lock_in_) V (new_C_mfsm_state = CMABT)) in
let ms10 = (ALTER ms0 (1) (((new_C_mfsm_state = CMA1) V (new_C_mfsm_state = CMA0) V (new_C_mfsm_state = CMA2) V (new_C_mfsm_state = CMD1) V ((new_C_mfsm_state = CMD0) A _C_last_in_) V (new_C_mfsm_state = CMW) V (new_C_mfsm_state = CMABT)))) in
let ms210 = (ALTER ms10 (2) (((new_C_mfsm_state = CMA3) V (new_C_mfsm_state = CMA1) V (new_C_mfsm_state = CMA0) V (new_C_mfsm_state = CMA2) V (new_C_mfsm_state = CMD1) V (new_C_mfsm_state = CMW) V (new_C_mfsm_state = CMABT)) A -new_S pmm..fail A -(ELEMENT new_R gcr (28))) in
let CB ms..out = (-(new_C_mfsm_state = CMI) A (new C_mfsm_state = CMR)) => ms2101 ARBN) in
let ss0 = (ALTER ARBN (0) (((new_C sfsm_state = CSAO) V ((new_C sfsm_state = CSALE) A -c_new_write) V (new_C sfsm_state = CSACK))) in
let ss10 = (ALTER ss0 (1) A (new_C sfsm_state = CSACK)) in
let ss210 = (ALTER ss10 (2) ~new_S pmm..fail A -(ELEMENT new_R gcr (28))) in
let CB ss out = (((new_C sfsm_state = CSI) A (new_C sfsm_state = CSABT)) => ss2101 ARBN) in
let CB_ad out = (c__dfsm_cad_en) =>
               (c_cout_sel = (WORDN 0)) => Par_Enc rep (SUBARRAY new_C_a1a0 (15,0)) I
               (c_cout_sel = (WORDN 1)) => Par_Enc rep (SUBARRAY new_C_a1a0 (31,16)) I
               (c_cout_sel = (WORDN 2)) => Par_Enc rep (SUBARRAY new_C_a3a2 (15,0)) I
               Par_Enc rep (SUBARRAY new_C_a3a2 (31,16))! ARBN) in
let MB_addr = (M_rdy) => ((INCN 18 M_addr) I M_addr) in
let mb_data_7_0 = (((ELEMENT M_be (0)) => (SUBARRAY i_ad (7,0)) I (SUBARRAY M_rd_data (7,0))) in
let mb_data_15_8 = (((ELEMENT M_be (1))) => (SUBARRAY i_ad (15,8)) I (SUBARRAY M_rd_data (15,8))) in
let mb data_23_16 = (((ELEMENT M_be (2))) => (SUBARRAY i_ad (23,16)) I (SUBARRAY M_rd_data (23,16))) in
let mb data_31_16 = (((ELEMENT M_be (3))) => (SUBARRAY i_ad (31,24)) I (SUBARRAY M_rd_data (31,24))) in
let mb data = ((MALTER (MALTER (MALTER (MALTER MB ARBN (7,0) mb data_7_0)
(15,8) mb data_15_8)
(23,16) mb data_23_16)
(31,24) mb data_31_24)) in
let MB data out = ((new_M fsm_state = MW) => (Ham_Enc rep mb data) | ARBN) in
let MB_cs_eeprom_ = ~((new_M_fsm_state = MI)) \land \neg new_M_se in
let MB_cs_sram_ = ~((new_M_fsm_state = MI)) \land new_M_se in
let MB_we_ = ~((new_M_se \lor \neg(new_M_fsm_state = MI)) \lor \neg reset_cport)
\land \neg disable_writes
\land ((new_M_fsm_state = MBW) \lor (new_M_fsm_state = MW) \lor new_M_wwdel)) in
let MB_oe_ = ~((new_M_wr \land (new_M_fsm_state = MA)) \lor (new_M_fsm_state = MR)) in
let disable_int = ~((s fsm = (ELEMENT s_delay_out (6))) \land s fsm = sdi
\lor \neg test = \neg (ELEMENT s_delay_out (16)))) in
let Int0_ = ~((r int0_en \land \neg R_int0_dis \land \neg disable_int) in
let Int1 = (R_ctrl1_cry \land new_R_int1_en \land \neg disable_int) in
let Int2 = (R_ctrl3_cry \land new_R_int2_en \land \neg disable_int) in
let Int3_ = ~((r_int3_en \land \neg R_int3_dis \land \neg disable_int) in
let Led = (SUBARRAY new_R_gcr (3,0)) in
let Reset_cpu0 = new_S_reset_cpu0 in
let Reset_cpu1 = new_S_reset_cpu1 in
let Cpu_hist = new_S_cpu_hist in
let Piu_fail = new_S_piu_fail in
let Cpu0_fail = new_S_cpu0_fail in
let Cpu1_fail = new_S_cpu1_fail in
let Pmm_fail = new_S_pmm_fail in

(L_ad_out, L_ready_,
CB_rqt_out, CB_ms_out, CB_ss_out, CB_ad_out,
MB_addr, MB_data_out, MB_cs_eeprom_, MB_cs_sram_, MB_we_, MB_oe_,
Int0, Int1, Int2, Int3, Led,
Reset_cpu0, Reset_cpu1, Cpu_hist, Piu_fail, Cpu0_fail, Cpu1_fail, Pmm_fail)
);;

close_theory();

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