ATM Modeling Example for SpecGen Evaluation

Jon Kleinsmith
Jianwen Zhu
Daniel D. Gajski

Technical Report ICS-97-47
October, 1997

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
(714) 824-8059

jon@ics.uci.edu
jzhu@ics.uci.edu
gajski@ics.uci.edu

Abstract

In this report we discuss the specification of an ATM cell filter. A behavioral model of the filter is supplied using two specification languages, VHDL, and SpecC, a new language under development by the CADLAB at the University of California, Irvine. A description of the functionality of the filter model is supplied in addition to language features and problems encountered during the specification process. This model is to be used as an example for the evaluation of the SpecGen system, a simulation and synthesis environment also under development at the CADLAB.

Contents

1 Motivation

Under development at the CADLAB, University of California, Irvine is a new language and system-level design environment, **SpecC** and **SpecGen** respectively. The new language is based upon the C programming language and incorporates constructs that facilitate system-level specification. SpecGen is a codesign environment that will allow designers to rapidly explore their design space by allocating system-level components, partitioning and scheduling the specification on the selected architecture and generating a refined specification representing synthesis decisions.

In order to exercise the functionality of these new synthesis tools, several design examples, representing industrial strength applications are being generated. This paper introduces one such design example from the communications domain. The **CADLAB Development Frame Filter (cdf2atm)** accepts a a bit stream representing data formatted as a CADLAB Frame (cdf) and generates ATM packets. This filter is based upon existing industry products in this domain and exercises a significant number of language constructs and synthesis procedures in our new synthesis tools.

We have specified the filter using both VHDL and SpecC in order to compare and contrast strengths and weakness of both languages with respect to this application domain and allow the reader the opportunity to understand the syntax and semantics of this new language in relation to VHDL. For completeness, a comparable C model is included in the Appendices which describes the original specification.

In Section 2 we offer a detailed description of the filter, outlining process by process its functionality. Next, Section 3 presents the specification decisions needed to capture the design using VHDL. We also explain the various strengths and weaknesses of the language, as used within this application domain, encountered during specification. Similarly in Section 4, we present the modeling effort utilizing the SpecC language and discuss its merits and shortcomings with respect to the communications domain. In the last section, we offer some short conclusions based upon our specification experiences.

Finally, in Appendix A and Appendix B, we have included the VHDL and SpecC descriptions of the *cdf2atm* filter, respectively. Appendix C contains the filter described in *C*.

2 Description

The *cdf2atm* filter is a generalized model of an ATM filter, designed to accept as input *CDFrames*, a LAN digital packet format devised for this example and based upon industrial formats, reformats this input into ATM cells, and sends these cells as output.

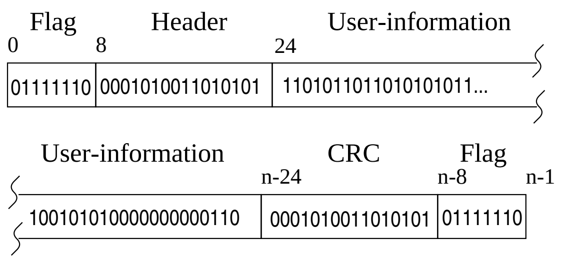

Figure 1: CD Frame Format

The *CDFrame* is a variable length communication string with a maximum length of 5000 bits. The frame is composed of an 8-bit opening flag followed by a 16-bit header containing routing and network information. The next series of bits from position 25 to $n - 24$, where n is the length of the frame, are dedicated to the actual data payload of the frame and are referred to collectively as the user-information. The last 24 bits of the frame contain a 16-bit **cyclic redundancy check** (CRC) number and an 8-bit closing flag, identical to the opening flag. This frame format is presented graphically in Figure 1.

The behavior of the *cdf2atm* filter may be divided into a series of tasks or processes used to capture, decompose and reformat a *CDFrame* in order to output ATM cells. The flow-graph of this filter can be seen in Figure 2. In the following sections we describe the functionality of each of these seven processes in detail.

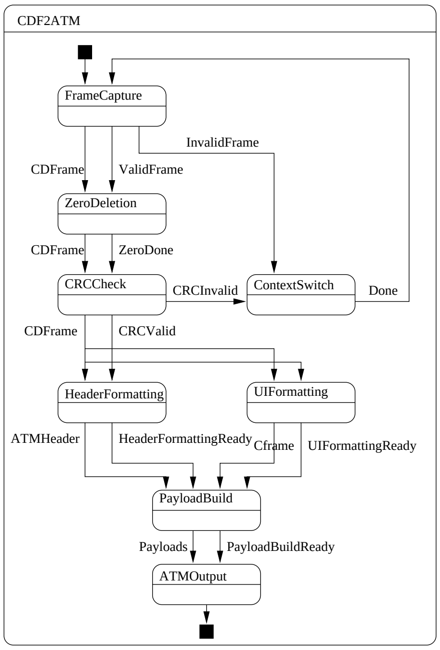

Figure 2: CDF2ATM Graphical Specification

2.1 FrameCapture

As stated previously, the filter captures data serially, one bit at a time, from a data-stream. Valid data bits are recognized by an accompanying clock signal. When the clock signal is high, data is read from the data port. In the specification, we refer to the clock and data ports as *ICDFCLK* and *ICDFDATA*, respectively. Upon encountering a specific bit sequence representing the opening flag of a *CDFrame*, the *FrameCapture* process begins capturing the data-stream as a potentially valid frame. Capture continues until either a closing or abort flag is recognized or the frame length limit is exceeded. If a closing flag is encountered, the frame's length is validated. After passing length validation, a zero deletion procedure in initiated to remove additional zeros that may have been added "upstream" in order to avoid passing false opening or closing flags, the details are described in the following section. The frame is then passed to the *CRCCheck* process. If the frame length is exceeded or the frame is found to be too short, it is assumed that the frame is invalid and the *ContextSwitch* process is initiated. If an abort flag is encountered, the current frame is abandoned and *FrameCapture* resumes by seeking the next opening flag.

2.2 ZeroDeletion

In order to ensure that data within the frame is not mistaken for an opening or closing flag, the frame had additional zeros inserted by the sender. For example, within the original frame the string "...00111111011..." may be present. Upon capture, the substring "01111110" will be mistaken for a closing flag and the resulting frame will have been truncated. To avoid this, a zero is inserted wherever five consecutive ones are found within the frame. Thus, following zero insertion, the string above would be transformed into "...0011111**0**1011..." the bold zero being the bit added to the string. With this in mind, the receiver must remove a zero following any string of five ones. After capturing a frame, this process will perform zero deletion as described above and then pass the corrected frame to the *CRCCheck* process for validation.

2.3 ContextSwitch

This process keeps a tally of the number of invalid frames encountered. Each invalid frame encountered triggers the execution of this process, which in turn, increments an invalid frame counter. Should the limit of invalid frames be exceeded, it is assumed that frame capture has been operating out of context, that is, since the opening and closing flags of a frame are identical and junk data or "noise" may be encountered between closing flags and the next opening flag, a closing flag may be misinterpreted as an opening flag and invalid data may be capture inadvertently. The notion of frame context can be seen in Figure 3.

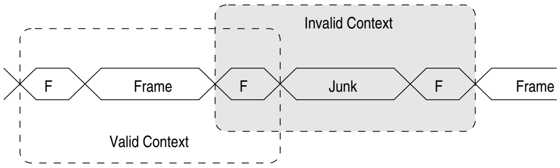

Figure 3: Valid and Invalid Capture Contexts

Next, the datastream is scanned until an opening flag is encountered and it is discarded, in effect switching the valid frame context. *FrameCapture* then resumes.

2.4 CRCCheck

After a potentially valid frame is captured and extra zeros have been removed, it is checked for data integrity using the 16-bit CRC included at the end of the frame. The CRC computation is essentially, 17-bit CRC equation divided into the data to be checked with a previously computed CRC appended. The frame is considered error-free if the result of the division has no remainder. In our specification we use a temporary variable and repeatedly use the exclusive-OR function, shifting the frame buffer whenever the most-significant bit is "0". If an error had occurred in transmission, a non-zero remainder would result from the division. If the frame is invalid, the *ContextSwitch* process is called. If the CRC passes, the frame is passed on for decomposition and formatting, specifically, the *HeaderFormatting* and *UIFormatting* processes are initiated.

2.5 HeaderFormatting

The *HeaderFormatting* process maps information in the *CDFrame* header into a 40-bit ATM header format. Various fields in the *CDFrame* header will be needed in the ATM header to be output. The payload type (*PT2*) is mapped, along with the cell's relative priority (*CLP2*). The virtual channel and path (*VCI2*) and (*VPI2*) are exchanged as well. Diagrammatically, this mapping can be seen in Figure 4.

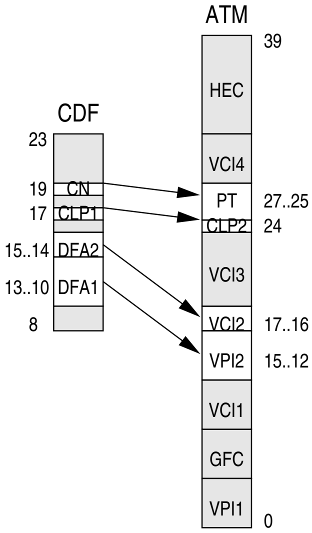

Figure 4: Header Mapping

2.6 UIFormatting

Concurrent with *HeaderFormatting*, the user-information of the *CDFrame* must be broken into 48-byte payloads in order to fit within one or more ATM cells. The *UIFormatting* process begins by determining how many ATM payloads may be obtained from the present *CDFrame*. In order to ensure that the frame divides evenly, padding is added to the frame and its length is adjusted accordingly. At this point, the frame is considered to be an intermediate format and is referred to in the specification as a *CFrame*. Next, the length of the *CFrame* is converted to a binary representation and stored within the body of the *CFrame*. Finally, CRC-32 is generated and the resulting remainder is attached to the *CFrame*. This *CFrame* is then forwarded to the *PayloadBuild* process for decomposition.

2.7 PayloadBuild

PayloadBuild waits for the completion of both the *HeaderFormatting* and *UIFormatting* processes. At this point, the *CFrame* is ready to be divided into 48-byte payloads for inclusion in the ATM cells. Additionally, an 8-bit CRC is calculated for the ATM header and stored within the *HEC* field dedicated for this result. The revised header and payload array are then sent for output via the *ATMOutput* process.

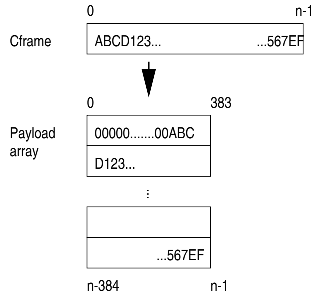

Figure 5: Payload Structure

2.8 ATMOutput

Finally, the array of ATM cell payloads is output byte-wise serially through the filter's output ports. Iteratively, the master ATM header is sent to the output ports followed by a payload. When the last cell is reached, the last bit of the part field in the ATM header is changed to indicate that this is the last of a set of cells and the final header and payload are output.

3 VHDL Specification

The specification of the *cdf2atm* was initially captured using VHDL. The decision to use VHDL as the initial or primary specification language was motivated by the simulation and synthesis tools available and the need for a base model with which to compare and contrast the new specification language, as well as evaluate the new simulation and synthesis tools under development at the CADLAB.

The functionality of the filter was specified as a number of processes in order to add complexity to the design. This complexity will in turn allow us to more completely exercise synthesis tasks such as partitioning and scheduling in the developmental tools. It should be noted that the design methodology behind the new tools is directed at the system-level and as such, the smallest indivisible unit is the task or process.

Since the design was to be divided amongst a number of processes, running concurrently, a global memory and communication scheme were employed. A group of control signals were introduced for the purpose of termination notification and for simplicity, each process was given its own signal(s) to eliminate the need for resolution functions or arbitration at this time. It should be noted that the division of the filter's functionality into the represented processes was not determined through any complex analysis and should not be considered optimal, rather, the operations were grouped as the designer deemed logical, ie. those operation necessary for capturing a frame were grouped and specified as a single process.

A Synopsys synthesizable subset of VHDL was used, again to facilitate evaluation at later stages of this project. The specification proceeded in a rather straight-forward manner with little difficulty. VHDL was well suited to the task of capturing the functionality of this communications example, facilitating bit examination and manipulation while maintaining a high-level view of the filter.

4 SpecC Specification

The translation of the design model from VHDL to SpecC is straightforward, if the following steps are followed:

- each VHDL **process** declaration is converted to a SpecC **behavior** declaration;
- each global signal accessed by the process is declared as a port of the corresponding behavior;
- each imperative statement such as assignment, loop or condition is converted to the corresponding SpecC statement in the *main* method of the behavior;
- each temporal statement such as *wait for* is converted to the corresponding temporal statement in the main method of the behavior.
- each behavior is instantiated in a top-level behavior with the appropriate port mapping.
- a *par* statement is added in the *main* method of the top behavior to invoke concurrent behaviors.

For most of the ATM model, we follow the above procedures to obtain the SpecC model from the VHDL model. However, we changed a few items in order to exercise some features of SpecC, and which may lead to more efficient simulation and synthesis.

The most notable change was the representation of frame data from the *bitvector* in VHDL into a channel *CBitBuffer* in SpecC. A set of methods defined on the *CBitBuffer* allow the concatenation, extraction and viewing (*peek*) of bits, which are potentially cheaper to implement than the corresponding *bitvector* operations.

The other change was to the control structure of the top-level behavior. In VHDL, concurrent processes can only be declared at the first level of hierarchy. *Fork/join* behavior can only be simulated by using extra signals between processes. In SpecC, such a restriction is relaxed by the presence of the *par* statement. For example, the *FrameCapture* behavior is sequentially followed by a *par* statement, which in turn contains concurrent behaviors *HeaderFormatting* and *UIFormatting*.

5 Conclusion

As shown in the previous sections, the behavioral description of the ATM filter is control dominated and is not computationally expensive. Rather, the filter requires rapid data acquisition and high-speed memory access to ensure data throughput. Both VHDL and SpecC offered language constructs, types and operators that allow the designer to specify the filter at the behavioral level.

Each language expressed the design in a similar fashion and resulted in specifications of nearly identical length. VHDL offered many built-in types and operators for the manipulation of bit-streams and bit-vectors, speeding specification. While SpecC currently lacks such built-in utilities, they were easily coded. Furthermore, by encapsulating communication between tasks or processes in channel objects, SpecC may prove advantageous at later stages of synthesis, since it is difficult to extract control and data communication in the VHDL model without expressing it explicitly through common signal names or procedures. Offering built-in language and tool support for the channel object should facilitate interface generation during communication synthesis.

6 Acknowledgment

We would like to acknowledge the support of this work by Hitachi Ltd. Japan. Their funding and personnel have greatly aided in the continued research in this area. In particular, we wish to extend our thanks to Tatsuya Umezaki of Hitachi, for his efforts in providing an industrial ATM example for our experiments, assisting and guiding the development of the various models and creating a test suite for model verification.

7 Appendix A

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
--***************************************************************************--
--         Title: CaDFrame to ATM filter
--    Written by: Jon Kleinsmith
--          Site: CADLAB, University of California, Irvine
-- Last Modified: 970105
--   Description: Filter accepts a stream of CDFrames bitwise serial
--                Performs format and crc checks and then produces
--                ATM packets to be output bytewise serial.
--***************************************************************************--

entity cdf2atm is
  port(ICDFCLK  : in  bit;
       ICDFDATA : in  bit;
       OATMCLK  : out bit := '0';
       OATMDATA : out bit_vector(7 downto 0));
end cdf2atm;

architecture behavioral of cdf2atm is
  -- OBJECT LENGTHS
  constant CDFMAXLENGTH        : integer := 5000;
  constant MINCDFLENGTH        : integer := 40;
  constant CPCSPDUMAXLENGTH    : integer := 4992;
  constant ATMCELLDATALENGTH   : integer := 384;
  constant ATMCELLHEADERLENGTH : integer := 40;
  constant CONTEXTSWITCHTIME   : integer :=  2;
  constant PADMAXLENGTH        : integer := ATMCELLDATALENGTH;
  constant BYTE                : bit_vector(7 downto 0)  := "00000000";

  -- FLAGS
  constant CDFFLAG             : bit_vector(7 downto 0) := "01111110";
  constant ABORTFLAG           : bit_vector(7 downto 0) := "01111111";

  -- CRC8 OBJECTS
  constant CRC8Eqn             : bit_vector(8 downto 0) := "100000111";
  constant CRC8EqnPad          : bit_vector(7 downto 0) := "00000000";

  -- CRC16 OBJECTS
  constant CRC16Eqn            : bit_vector(16 downto 0) :=
    "10001000000100001";
```

```
  constant CRC16ValidEqn       : bit_vector(15 downto 0) :=
    "0000000000000000";

  -- CRC32 OBJECTS
  constant CRC32Eqn            : bit_vector(32 downto 0) :=
    "100000100110000010001110110110111";

  constant CRC32ValidEqn       : bit_vector(31 downto 0) :=
    "00000000000000000000000000000000";

  -- DATA SIGNALS
  subtype Payload                 is bit_vector(ATMCELLDATALENGTH-1 downto 0);
  type    PayloadsArray           is array (5 downto 0) of Payload;

  signal  CDFrame               : bit_vector(CDFMAXLENGTH-1 downto 0);
  signal  CDFrameLength         : integer := 0;
  signal  CPCSPDUFrame          : bit_vector(CPCSPDUMAXLENGTH-1 downto 0);
  signal  CPCSPDUFrameLength    : integer := 0;
  signal  ATMCellHeaderFrame    : bit_vector(ATMCELLHEADERLENGTH-1 downto 0);
  signal  ATMCellHeader_0       : bit_vector(ATMCELLHEADERLENGTH-1 downto 0);
  signal  ATMCellHeader_1       : bit_vector(ATMCELLHEADERLENGTH-1 downto 0);
  signal  Payloads              : PayloadsArray;
  signal  PayloadsNumber        : integer := 0;

  -- CONTROL SIGNALS
  signal  ValidFrame            : bit := '0';
  signal  InvalidFrame          : bit := '0';
  signal  CRCValid              : bit := '0';
  signal  CRCFail               : bit := '0';
  signal  UIFormatingReady      : bit := '0';
  signal  HeaderFormattingReady : bit := '0';
  signal  PayloadsReady         : bit := '0';
  signal  Hault_Capturing       : bit := '0';

begin
--******************************************************************--
-- FrameCapture
-- Input:    ICDFDATA
-- Output:   ValidFrame
--           InvalidFrame
--           CDFrame
--           CDFrameLength
-- Function: Captures bitstream and assembles CDFrame after recognizing
--           opening and closing flags.  If frame is valid, CRCChecking
```

```
--          is commenced, otherwise, context switching occurs and
--          process resumes.
--*****************************************************************--
FrameCapture : process
    variable TmpFrame           : bit_vector(CDFrame'range) := (others => '0');
    variable TmpFrameLength     : integer :=  0;
    variable WrkFrame           : bit_vector(CDFrame'range) := (others => '0');
    variable WrkFrameLength     : integer :=  0;
    variable Done               : bit     := '0';
--*****************************************************************--
-- ZeroDel
-- Function: 0 is inserted into CDFrame after consecutive 5 '1'
-- to avoid confusion between flags and user information.
-- To delete above '0' and restore original user information
--*****************************************************************--
procedure zerodel(
I_Frame : in bit_vector(CDFMAXLENGTH-1 downto 0);
I_Length    : in integer;
O_Frame : out bit_vector(CDFMAXLENGTH-1 downto 0);
O_Length    : out integer ) is
variable TMPbuf : bit_vector(CDFMAXLENGTH-1 downto 0);
variable K    : integer;
variable J    : integer;
variable I    : integer;
variable one  : integer;
variable caution : integer;
begin
  caution := 0;
  one := 0;
  I := I_Length-1;
  J := 0;

  while I >= 0 loop
    if(I_Frame(I) = '1') then
      one := one + 1;
      assert(caution = 0)
report "Zero Insertion Error"
severity Failure;
      TMPbuf(J) := I_Frame(I);
      J := J + 1;
    elsif(I_Frame(I) = '0') then
      if(caution = 0) then
TMPbuf(J) := I_Frame(I);
J := J + 1;
```

```
      end if;
      one := 0;
      caution := 0;
    else
      TMPbuf(J) := I_Frame(I);
      J := J + 1;
    end if;
    if(one = 5) then
      caution := 1;
    end if;
    I := I - 1;
  end loop;
  O_Length := J;
  K := J - 1;

  while K >= 0 loop
    O_Frame(K) := TMPbuf(J-1-K);
    K := K - 1;
  end loop;
end ZeroDel;
-- end ZeroDel

begin
  ValidFrame <= '0';
  InvalidFrame <= '0';
  TmpFrame := (others => '0');
  TmpFrameLength := 0;
  Done := '0';
  wait for 0 ns;

-- capture data stream
-- recognize opening flag
  OpenFlag:loop
    TmpFrame(7 downto 1) := TmpFrame(6 downto 0);
    wait until (ICDFCLK = '1');
    TmpFrame(0) := ICDFDATA;
    if TmpFrame(7 downto 0) = CDFFLAG then
      exit;
    end if;
  end loop OpenFlag;

-- capture data stream
-- assemble potential frame
  If Hault_Capturing = '0' then
```

```
    TmpFrameLength := 0;
    CloseFlag:loop
      TmpFrame(TmpFrame'high downto 1) :=
TmpFrame(TmpFrame'high-1 downto 0);
      wait until (ICDFCLK = '1');
      TmpFrame(0)                      := ICDFDATA;
      TmpFrameLength                   := TmpFrameLength + 1;

      -- check for frame overflow
      if Done /= '1' then
if (TmpFrameLength > CDFMAXLENGTH) then
  Done := '1';
end if;
      end if;
      -- check for abortflag
      if Done /= '1' then
if (TmpFrame(ABORTFLAG'high downto 0) = ABORTFLAG) then
  Done := '1';
end if;
      end if;

      -- check for closign flag
      if (TmpFrame(7 downto 0) = CDFFLAG) then
if Done = '1' then
  InvalidFrame <= '1';
else
  TmpFrame(TmpFrameLength-9 downto 0) :=
    TmpFrame(TmpFrameLength-1 downto 8);
  TmpFrameLength := TmpFrameLength - 8;
  ZeroDel(TmpFrame,TmpFrameLength,WrkFrame,WrkFrameLength);
  if (WrkFrameLength > MINCDFLENGTH) then
    CDFrame <= WrkFrame;
    CDFrameLength <= WrkFrameLength;
    ValidFrame                         <= '1';
  else
    InvalidFrame  <= '1';
  end if;
end if;
exit;
      end if;
    end loop CloseFlag;
    wait for 0 ns;
  end if;
end process FrameCapture;
```

```
--*****************************************************************--
-- ContextSwitch
-- Input:    InvalidFrame
--           CRCFail
-- Output:   none
-- Function: Keeps track of invalid frames and after encountering too many
--           invalid frames, scans the data stream for a new flag.
--*****************************************************************--
ContextSwitch : process
  variable InvalidFrameCount : integer := 0;
  variable LoopTime          : integer := 0;
  variable ExitTime          : integer := 0;
  variable TmpFlag           : bit_vector(7 downto 0) := (others => '1');
begin
  if (InvalidFrame'event and InvalidFrame = '1') or
    (CRCFail'event and CRCFail = '1') then
    InvalidFrameCount := InvalidFrameCount + 1;
  end if;
  if (CRCValid'event and CRCValid = '1') then
    InvalidFrameCount := 0;
  end if;
  if ( InvalidFrameCount >= CONTEXTSWITCHTIME ) then
    Hault_Capturing <= '1';
    LoopTime := 0;
    ExitTime := 0;
    CTS:            loop
      LoopTime := LoopTime + 1;
      TmpFlag(7 downto 1) := TmpFlag(6 downto 0);
      wait until ICDFCLK = '1';
      TmpFlag(0) := ICDFDATA;
      if  TmpFlag = CDFFLAG  then
ExitTime := LoopTime + 1;
      end if;
      if  ExitTime = LoopTime then
Hault_Capturing <= '0';
InvalidFrameCount := 0;
TmpFlag := (others => '1');
exit;
      end if;
    end loop CTS;
  end if;
  wait on InvalidFrame,CRCFail,CRCValid;
end process ContextSwitch;
```

```
--*****************************************************************--
-- CRCCheck
-- Input:    CDFrame
--           CDFrameLength
--           ValidFrame
-- Output:   CRCValid
--           CRCFail
-- Function: Performs 16-bit cyclic redundancy check on a valid CDFrame
--           and returns whether the frame has valid CRC or has failed the
--           check.
--*****************************************************************--
CRCCheck : process
  variable i           : integer := 0;
  variable TmpBuffer   : bit_vector(CRC16Eqn'range) := (others => '0');
  variable TmpCDFrame  : bit_vector(CDFMAXLENGTH-1 downto 0) := (others => '0');
begin
  CRCFail  <= '0';
  CRCValid <= '0';
  if ValidFrame = '1' then
    TmpCDFrame(CDFrameLength+15 downto 16):=
      CDFrame(CDFrameLength-1 downto 0);
    -- copy a window of data from frame
    TmpBuffer := TmpCDFrame(CDFrameLength+15 downto CDFrameLength-1);

    -- perform CRC16
    i := 1;
    while i < CDFrameLength loop
      if (TmpBuffer(TmpBuffer'high) = '0') then
TmpBuffer := TmpBuffer(TmpBuffer'high-1 downto 0)&
     TmpCDFrame(CDFramelength-1-i);
i := i + 1;
      else
TmpBuffer := TmpBuffer xor CRC16Eqn;
      end if;
    end loop;
    if (TmpBuffer(TmpBuffer'high) = '1') then
      TmpBuffer := TmpBuffer xor CRC16Eqn;
    end if;
      -- check for pass or fail
    if (TmpBuffer(15 downto 0) = CRC16ValidEqn) then
      CRCValid <= '1';
    else
      CRCFail <= '1';
```

```
    end if;
  end if;
  wait on ValidFrame;
end process CRCCheck;

--*******************************************************************--
-- HeaderFormatting
-- Input:    CDFrame
--           CRCValid
-- Output:   ATMHeaderFrame
-- Function: Maps CDFrame header information to an ATM header
--*******************************************************************--
HeaderFormatting: process
begin
  if CRCValid = '1' then
    HeaderFormattingReady <= '0';
    ATMCellHeaderFrame <=
      "00000000"
      &"0000"
      &CDFrame(CDFrameLength-3 downto CDFrameLength-6)
      &CDFrame(CDFrameLength-7 downto CDFrameLength-8)
      &"000000"
      &CDFrame(CDFrameLength-10)
      &'0'
      &CDFrame(CDFrameLength-12)
      &'0'
      &CDFrame(CDFrameLength-13 downto CDFrameLength-16)
      &"00000000";
    HeaderFormattingReady <= '1';
  end if;
  wait on CRCValid;
end process HeaderFormatting;

--*******************************************************************--
-- UIFormatting
-- Input:    CDFrame
--           CDFrameLength
--           CRCValid
-- Output:   UIFormattingReady
--           CPCSPDUFrame
-- Function: Calculates binary length of the data
--           Calculates CRC32 for the data
--*******************************************************************--
UIFormatting : process
```

```
  variable PayloadLength      : integer;
  variable TmpPayloadLength   : integer;
  variable PADLength          : integer;
  variable CFrame             : bit_vector(CPCSPDUFrame'range);
  variable WrkCFrame          : bit_vector(CPCSPDUFrame'range);
  variable TmpCFrame          : bit_vector(CPCSPDUMAXLENGTH+31 downto 0);
  variable CFrameLength       : integer;
  variable i                  : integer;
  variable j                  : integer;
  variable TmpBuffer          : bit_vector(CRC32Eqn'range);
  variable LengthBuffer_1     : bit_vector(15 downto 0);
  variable LengthBuffer_2     : bit_vector(15 downto 0);
begin
  if CRCValid = '1' then
    UIFormatingReady <= '0';
    wait for 0 ns;
    -- Determine payload, frame and padding lengths and
    -- initialize the payload frame
    PayloadLength := CDFrameLength - 32;
    PADLength  := ATMCELLDATALENGTH-((PayloadLength+64)
     rem (ATMCELLDATALENGTH));
    -- remove address and crc
    CFrame(PayloadLength-1 downto 0) := CDFrame(CDFrameLength-17 downto 16);
    CFrameLength                     := PayloadLength + PADLength + 64;
    CPCSPDUFrameLength               <= CFrameLength;

    WrkCFrame := (others => '0');
    WrkCFrame := CFrame;
    CFrame := (others => '0');
    CFrame(CFrameLength-1 downto CFrameLength-PayloadLength)
      := WrkCFrame(PayloadLength-1 downto 0);
    -- Calculate and store the binary length of the payload
    TmpPayloadLength := PayloadLength;

    i := 0;
    LengthBuffer_1 := (others => '0');
    LengthBuffer_2 := (others => '0');
    while (TmpPayloadLength /= 0) loop
      if ( (TmpPayloadLength rem 2 ) = 1 ) then
LengthBuffer_1(LengthBuffer_1'high-i) := '1';
      end if;
      TmpPayloadLength := TmpPayloadLength / 2;
      i := i + 1;
    end loop;
```

```
    j := 0;
    while j <= i-1 loop
      LengthBuffer_2(i-1-j):=LengthBuffer_1(LengthBuffer_1'high+1-i+j);
      j := j + 1;
    end loop;

    CFrame(CFrameLength-1-PayloadLength-Padlength-16
   downto CFrameLength-1-PayloadLength-Padlength-31) :=
      LengthBuffer_2;

    -- calculate CRC32 for the payload
    TmpBuffer := CFrame(CFramelength-1 downto CFrameLength-33);
    i := 1;
    while i < (CFrameLength-32) loop
      if (TmpBuffer(TmpBuffer'high) = '0') then
TmpBuffer := TmpBuffer(TmpBuffer'high-1 downto 0)&
     CFrame(CFramelength-33-i);
i := i + 1;
      else
TmpBuffer := TmpBuffer xor CRC32Eqn;
      end if;
    end loop;
    if (TmpBuffer(TmpBuffer'high) = '1') then
      TmpBuffer := TmpBuffer xor CRC32Eqn;
    end if;
    CFrame(TmpBuffer'high-1 downto 0):=
      TmpBuffer(TmpBuffer'high-1 downto 0);
    CPCSPDUFrame <= CFrame;
    UIFormatingReady <= '1';
  end if;
  wait on CRCValid;
end process UIFormatting;

--*****************************************************************--
-- PayloadBuild
-- Input:    ATMCellHeaderFrame
--           CPCSPDUFrame
--           UIFormattingReady
-- Output:   ATMCellHeader
--           Payloads
--           PayloadsNumber
--           PayloadsReady
-- Function: Decomposes CPCSPDU structure into and array of ATM payloads
--           Calculates CRC8 for the ATM header
```

```
--*****************************************************************--
PayloadBuild : process
  variable i              : integer := 0;
  variable k              : integer := 0;
  variable TmpCellHeader  : bit_vector(ATMCellHeaderFrame'range);
  variable TmpPayloads    : PayloadsArray;
  variable TmpBuffer      : bit_vector(ATMCellHeaderFrame'range);

--*****************************************************************--
-- Input:    I_CellHeader
-- Output:   O_CellHeader
-- Function: CRC 8 Caliculation
--*****************************************************************--
  procedure crc_8(
I_CellHeader : in bit_vector(39 downto 0);
O_CellHeader : out bit_vector(39 downto 0)
) is
    constant CRC8Eqn     : bit_vector(8 downto 0) := "100000111";
    variable TmpBuffer   : bit_vector(CRC8Eqn'range);

  begin
    -- copy a window of data from frame
    TmpBuffer := I_CellHeader(39 downto 31);
    -- perform CRC8
    i := 1;
    while i < 32 loop
      if (TmpBuffer(TmpBuffer'high) = '0') then
TmpBuffer := TmpBuffer(TmpBuffer'high-1 downto 0)
     &I_CellHeader(31-i);
i := i + 1;
      else
TmpBuffer := TmpBuffer xor CRC8Eqn;
      end if;
    end loop;
    if (TmpBuffer(TmpBuffer'high) = '1') then
      TmpBuffer := TmpBuffer xor CRC8Eqn;
    end if;
    O_CellHeader := I_CellHeader(39 downto 8)&TmpBuffer(7 downto 0);
  end crc_8;

begin
  if UIFormatingReady = '1' and HeaderFormattingReady = '1' then
    PayloadsReady <= '0';
    wait for 0 ns;
```

```
    TmpCellHeader := ATMCellHeaderFrame;
    -- decompose CPCSPDUFrame into payloads
    i := 0;
    k := 0;
    while (i < CPCSPDUFrameLength) loop
      TmpPayloads(k) :=
CPCSPDUFrame(CPCSPDUFrameLength-1-i
     downto CPCSPDUFrameLength-1-(ATMCELLDATALENGTH+i-1));
      i := i + ATMCELLDATALENGTH;
      k := k + 1;
      assert k < PayloadsArray'length
report "PayloadsArray Overflow"
severity Failure;
    end loop;
    -- calculate CRC8 for the ATM header and store
    crc_8(TmpCellHeader,TmpBuffer);
    ATMCellHeader_0      <= TmpBuffer;

    TmpCellHeader(14) := '1';
    crc_8(TmpCellHeader,TmpBuffer);
    ATMCellHeader_1      <= TmpBuffer;

    PayloadsNumber     <= k;
    Payloads           <= TmpPayloads;
    PayloadsReady      <= '1';
  end if;
  wait on UIFormatingReady,HeaderFormattingReady;
end process PayloadBuild;

--*****************************************************************--
-- ATMOutput
-- Input:    ATMCellHeader
--           Payloads
--           PayloadsNumber
--           PayloadsReady
-- Output:   OATMCLK
--           OATMDATA
-- Function: Outputs ATM packets bitwise serial
--*****************************************************************--
ATMOutput : process
  variable i                 : integer := 0;
  variable k                 : integer := 0;
  variable TmpPayloadsNumber : integer := 0;
  variable TmpCellHeader     : bit_vector(ATMCellHeaderFrame'range);
```

```
  variable TmpPayloads       : PayloadsArray;
begin
  if PayloadsReady = '1' then
    TmpPayloads       := Payloads;
    TmpPayloadsNumber := PayloadsNumber;

    -- loop through PayloadArray
    -- output header then payload
    for k in 0 to TmpPayloadsNumber-1 loop
      if (k + 1 = TmpPayloadsNumber) then
TmpCellHeader     := ATMCellHeader_1;
      else
TmpCellHeader     := ATMCellHeader_0;
      end if;

      -- send ATM header
      i := 0;
      while (i < TmpCellHeader'high) loop
OATMCLK  <= '1';
OATMDATA <= TmpCellHeader(TmpCellHeader'high-i
  downto TmpCellHeader'high-(i+7));
wait for 5 ns;
OATMCLK  <= '0';
wait for 5 ns;
i := i + 8;
      end loop;

      -- send ATM payload
      i := 0;
      while (i < Payload'high) loop
OATMCLK  <= '1';
OATMDATA <= TmpPayloads(k)(Payload'high-i
   downto Payload'high-(i+7));
wait for 5 ns;
OATMCLK  <= '0';
wait for 5 ns;
i := i + 8;
      end loop;
    end loop;
  end if;
  wait on PayloadsReady;
end process ATMOutput;
end behavioral; --cdf2atm
```

```
---------------------------------------------------------------------
library STD;
use STD.TEXTIO.all;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;

entity sim_top is
end sim_top;

architecture behv of sim_top is

  file TVF : TEXT is in "invector";
  file DOF : TEXT is out "outvector";

  component cdf2atm port(
ICDFCLK  : in  bit;
ICDFDATA : in  bit;
OATMCLK  : out bit := '0';
OATMDATA : out bit_vector(7 downto 0) );
  end component;


signal IICDFCLK  : bit;
signal IICDFDATA : bit;
signal IOATMCLK  : bit := '0';
signal IOATMDATA : bit_vector(7 downto 0);

constant CYCLE : Time := 10 ns;
constant PULSE : Time := 5 ns;

begin
  u0 : cdf2atm
       port map(
IICDFCLK,
IICDFDATA,
IOATMCLK,
IOATMDATA
       );

--DATAREAD PROCESS
       READ_PROCESS:
       process
```

```
          variable vICDFCLK  : bit ;
          variable vICDFDATA : bit ;
          variable INLI : line;

       begin
               while not endfile(TVF) loop
                  READLINE(TVF,INLI);
                  READ(INLI,vICDFCLK);
                  READ(INLI,vICDFDATA);
                  IICDFCLK <= vICDFCLK;
                  IICDFDATA <= vICDFDATA;
                  wait for PULSE;
                  IICDFCLK <= not IICDFCLK;
                  wait for CYCLE-PULSE;
          end loop;
          wait;
  end process;

--DATA WRITE PROCESS
WRITE_PROCESS:
process(IOATMCLK)

variable OUTLI : Line;

begin
if(IOATMCLK'event and IOATMCLK = '1') then
                  write(OUTLI,NOW,left,15);
                  write(OUTLI,IOATMDATA,right,9) ;
                  writeline(DOF,OUTLI);
end if;
end process;
end behv; -- HSSI_testbench


configuration sim_cfg of sim_top is
       for behv
       end for;
end sim_cfg;
```

8 Appendix B

```
//////////////////////////////////////////////////////////////////////////
//        Title: CaDFrame to ATM filter
//   Written by: Jon Kleinsmith
//  Modified by: Jianwen Zhu
//  Modified by: Tatsuya Umezaki
//         Site: CADLAB, University of California, Irvine
// Last Modified: 971120
//  Description: Filter accepts a stream of CDFrames bitwise serial
//               Performs format and crc checks and then produces
//               ATM packets to be output bitwise serial.
/////////////////////////////////////////////////////////////////////////

// OBJECT LENGTHS
#define MAX_CDFLENGTH      5000;
#define MIN_CDFLENGTH        48;
#define MAX_CPCSPDULENGTH  4992;
#define ATMCELLDATALENGTH  48*8;
#define ATMCELLHEADERLENGTH  40;
// FLAGS
#define CDFFLAG   0x7e;
#define ABORTFLAG 0xffff;
// CRCS
#define CRC8EQN  0x07;
#define CRC16EQN 0x0811;
#define CRC32EQN 0xdb71641;
typedef unsigned int    uint;
typedef unsigned long   ulong;
typedef signed   char   byte;
typedef unsigned char   ubyte;
typedef bool            boolean;
#include <assert.h>
#include <stdlib.h>

channel CBitBuffer( void ) {
  int     fLength, fByteCount;
  int     fHead, fTail;
  ubyte*  fData;

  void  init( int maxLength ) {
    fLength = fHead = fTail = 0;
    fByteCount = (maxLength + 7) / 8;
    fData = (ubyte *)malloc( sizeof(ubyte) * fByteCount );
    }
```

```
int  getLength( void ) {
  return fLength;
  }

void  reset( void ) {
  fLength = fHead = fTail = 0;
  }

// set from offset the given number of bits
void  set( int offset, int size, uint val ) {
  int  curBit = (fHead + offset) % (fByteCount << 3);
  int  curByte = curBit >> 3;
  int  firstByteSize;
  byte  firstByteMask;
  assert( size <= 32 );
  firstByteSize = 8 - (curBit & 0x07);
  firstByteMask = 0xff << firstByteSize;
  fData[curByte] = (fData[curByte] & firstByteMask) |
    (byte)(val << (4*8-size) >> (4*8-firstByteSize));
  curByte ++;
  for( curBit = firstByteSize; curBit < size; curBit += 8, curByte ++ ) {
    if( curByte > fByteCount ) curByte = 0;
    fData[curByte] = val << (4*8-size+curBit) >> 3*8;
    }
  }

// add from tail the given number of bits
void put( int size, uint val ) {  // put after tail
  int  oldTail = fTail;
  assert( size <= 32 );
  fTail += size; fLength += size;
  assert( fLength <= (fByteCount << 3) );
  if( fTail > (fByteCount << 3) )
    fTail -= (fByteCount << 3);
  set( oldTail, size, val );
  }

// add from tail the given number of bits in another buffer
void put( CBitBuffer another, int from, int size ) {
  // put a buffer after tail
  int  buffer, i;
  for( i = 0; i < size; i += 4*8 ) {
    buffer = another.peek( from+i, 4*8 );
```

```
    if( i + 4*8 < size )
      put( 4*8, buffer );
    else
      put( size-i, buffer );
    }
  }

// add from head the given number of bits
void putHead( int size, uint val ) { // put before head
  assert( size <= 32 );
  fHead -= size; fLength += size;
  assert( fLength <= (fByteCount << 3) );
  if( fHead < 0 )
    fHead += (fByteCount << 3);
  set( fHead, size, val );
  }

// check from offset the given number of bits
uint  peek( int offset, int size ) {  // peek from offset
  uint rtn = 0, buffer;
  int  curOffset = (fHead + offset) % (fByteCount << 3);
  int  curByte = curOffset >> 3;
  int i;
  assert( size <= 32 );
  curOffset &= 0x7; buffer = fData[curByte] << curOffset;
  for( i = 0; i < size; i ++ ) {
    rtn <<= 1;
    if( curOffset == 8 ) {
            curOffset = 0;
      curByte ++;
      if( curByte > fByteCount ) curByte = 0;
      buffer = fData[curByte];
      }
    rtn |= (buffer & 0x80) ? 1 : 0;
    buffer <<= 1;
    curOffset ++;
    }
  return rtn;
  }

// check from tail the given number of bits
uint  peek( int size ) {  // peek from tail
  assert( size <= 32 );
  size = fLength < size ? fLength : size;
```

```
    if( size > 0 )
      return peek( fLength-size, size );
    return 0;
    }

  // remove from tail the given number of bits
  uint  get( int size ) {
    uint  buffer;
    assert( size <= 32 );
    buffer = peek( size );
    fTail -= size;
    if( fTail < 0 ) fTail += (fByteCount << 3);
    fLength -= size;
    return buffer;
    }

  // remove from head the given number of bits
  int  getHead( int size ) {  // get from head
    uint  buffer;
    assert( size <= 32 );
    buffer = peek( 0, size );
    fHead += size;
    if( fHead > (fByteCount << 3) )
      fHead -= (fByteCount << 3);
    fLength -= size;
    assert( fLength >= 0 );
    return buffer;
    }

  // Execute zero deletion
  uint ZeroDel(CbitBuffer another){
    int length = another.getlength();
    int one = 0;
    bit headBit;
    CbitBuffer buffer;
    for(i=0;i<length;i++){
      headBit = another.peek(length-1-i,1);
      if(headBit == 1){
one++;
buffer.put(1,headBit);
      }
      else{
if(caution == 0) buffer.put(1,headBit);
one = 0;
```

```
caution = 0;
      }
      if(one == 5) caution = 1:
    }
    length = buffer.getlength();
    another.put(length,buffer.peek(0,length));
  }

  // Execute crc8 caliculation
  void crc8(CbitBuffer Inbuffer,CbitBuffer Outbuffer){
    int length;
    ushort tmpFirst;
    ubyte tmp;
    length = Inbuffer.getLength() - 8;
    tmpFirst = Inbuffer.peek(length,1);
    tmp = Inbuffer.peek(length-9,8);
    i = 0;
    while(i<length){
      if(tmpFirst == 1){
tmpFirst = (tmpFirst^0x1)&0x1;
tmp ^= CRC8EQN;
      }
      else{
tmpFirst = tmp & 0x80;
tmp = (tmp << 1)| Inbuffer.peek(length-9-i,1);
i++;
      }
    }
    if(tmpFirst == 1){
      tmpFirst = (tmpFirst^0x1)&0x1;
      tmp ^= CRC8EQN;
    }
    Outbuffer.put(8,tmp);
    Outbuffer.put(length,Inbuffer.peek(9,length));
  }
};

behavior EAtmFilter(in bit pCDData,
    out ubyte pATMData){
  boolean fIsValidCRC,
    fIsValidFrame,
    fIsHault_Capturing,
    fIsInvalidFrame,
    fIsFailCRC;
```

```
  CbitBuffer fCDFrame(),fATMCellHeader_0(),fATMCellHeader_1(),
    fATMCellHeaderFrame(),fPayloads[6];
  int fCDFrameLength,fCPCSPDUFrameLength,fPayloadNumber;

  AtmFrameCapture frameCapture(pCDData,fIsHault_Capturing,
       fIsInvalidFrame,fIsValidFrame,
       fCDFrame,fCDFrameLength);
  AtmCrcCheck crcCheck(fCDFrame,fCDFrameLength,fIsFailCRC,
       fIsValidCRC);
  AtmRecovery recovery(pCDData,fIsInvalidFrame,fIsFailCRC,
       fIsValidCRC,fIsHault_Capturing);
  AtmHeaderFormating headerFormating(fCDFrame,fATMCellHeaderFrame);
  AtmUiFormating uiFormating(fCDFrame,fCDFrameLength,
     fCPCSPDUFrameLength);
  AtmPayloadBuild payloadBuild(fATMCellHeaderFrame,fCDFrame,
       fCPCSPDUFrameLength,
       fATMCellHeader_0,
       fATMCellHeader_1,fPayloads,
       fPayloadNumber);
  AtmOutput atmOutput(fATMCellHeader_0,fATMCellHeader_1,
      fPayloads,fPayloadNumber,fATMData);

  void init(void){
    int i;
    fCDFrame.init(MAX_CDFLENGTH);
    fATMCellHeaderFrame.init(ATMCELLHEADERLENGTH);
    for(i=0;i<6;i++) fPayloads[i].init(ATMCELLDATALENGTH);
  }
  void main(void){
    while(fIsValidCRC == false){
      par{
{
  frameCapture.main();
  if(fIsValidFrame == true)
    crcCheck.main();
}
recovery.main();
      }
    }
    par{
      headerFormating.main();
      uiFormating.main();
    }
    payloadBuild.main();
```

```
    atmOutput.main();
  }
};



behavior AtmCrcCheck(in CbitBuffer pCDFrame,
     in int pCDFrameLength,
     out boolean pIsFailCRC,
     out boolean pIsValidCRC){
  void main(void){
    ushort tmpFirst;
    int i,tmp;
    pIsValidCRC = false;
    pIsFailCRC  = false;
    tmpFirst = pCDFrame.peek(pCDFrameLength,1);
    tmp = pCDFrame.peek(pCDFrameLength-17,16);
    i = 1;
    while(i<pCDFrameLength){
      if(tmpFirst == 1){
tmpFirst = (tmpFirst^0x1)&0x1;
tmp ^= CRC16EQN;
      }
      else{
tmpFirst = tmp & 0x1000;
tmp = (tmp << 1)|pCDFrame.peek(pCDFrameLength-17-i,1);
i++;
      }
    }
    if(tmpFirst == 1){
      tmpFirst = (tmpFirst^0x1)&0x1;
      tmp ^= CRC16EQN;
    }
    if(tmp == 0){
      pIsValidCRC = true;
    }
    else{
      pIsFailCRC = true;
    }
    return;
  }
};

behavior AtmFrameCapture(in bit pCDData,
```

```
 in boolean pIsHault_Capturing,
 out boolean pIsInvalidFrame,
 out boolean pIsValidFrame,
 out CbitBuffer pCDFrame,
 out int pCDFrameLength){
  void main(void){
    int flag;
    boolean done;
    done = false;

    while(pCDFrame.peek(8) != CDFFLAG){
      pCDFrame.put(1,pCDData);
      waitfor(1);
    }
    pCDFrame.reset();
    pIsValidFrame = false;
    pIsInvalidFrame = false;
    if (pIsHault_Capturing == false){
      while(true){
pCDFrame.put(1,pCDData);
if(done == true){
  if(pCDFrame.getLength()>MAX_CDFLENGTH)
    done = true;
  flag = pCDFrame.peek(8);
  if(flag==ABORTFLAG) done = true;
}
if(flag==CDFFLAG){
  if(done == true){
    pIsInvalidFrame = true;
    break;
  }
  else{
    pCDFrame.get(8);
    pCDFrame.zerodel(pCDFrame);
    if(pCDFrame.getLength()>MIN_CDFLENGTH){
      pIsValidFrame = true;
      break;
    }
    else{
      pIsInvalidFrame = true;
      break;
    }
  }
}
```

```
waitfor(1);
      }
      return;
    }
  }
};

behavior AtmHeaderFormating(in CbitBuffer pCDFrame,
    out CbitBuffer pATMHeaderFrame){
  void main(void){
    int length = pCDFrame.getLength();
    pATMHeaderFrame.put(0,8);
    pATMHeaderFrame.put(pCDFrame.peek(length-16,4),4);
    pATMHeaderFrame.put(0,1);
    pATMHeaderFrame.put(pCDFrame.peek(length-12,1),1);
    pATMHeaderFrame.put(0,1);
    pATMHeaderFrame.put(pCDFrame.peek(length-10,1),1);
    pATMHeaderFrame.put(0,6);
    pATMHeaderFrame.put(pCDFrame.peek(length-8,2),2);
    pATMHeaderFrame.put(pCDFrame.peek(length-6,4),4);
    pATMHeaderFrame.put(0,12);
  }
};

behavior AtmOutput(in CBitBuffer pATMCellHeader_0,
   in CBitBuffer pATMCellHeader_1,
   in CBitBuffer pPayloads[],
   in int pPayloadNumber,
   out ubyte pATMData){
  void main(void){
    int i,k;
    for(k=0;k<pPayloadNumber;k++){
      if(k+1 == pPayloadNumber){
pATMCellHeaderFrame = pATMCellHeaderType_1;
      }
      else{
pATMCellHeaderFrame = pATMCellHeaderType_0;
      }
     for(i=0;i<ATMCELLHEADERLENGTH;i=i+8){
       pATMData = pATMCellHeaderFrame.peek(ATMCELLHEADERLENGTH-i,8);
       waitfor(1);
     }
     for(i=0;i<ATMCELLDATALENGTH;i=i+8){
       pATMData = pPayloads[k].peek(ATMCELLDATALENGTH-i,8);
```

```
      waitfor(1);
     }
    }
    return;
  }
};

behavior AtmPayloadBuild(in CBitBuffer pATMCellHeaderFrame,
 in CBitBuffer pCDFrame,
 in int pCPCSPDUFrameLength,
 out CBitBuffer pATMCellHeaderType_0,
 out CBitBuffer pATMCellHeaderType_1,
 out CBitBuffer pPayloads[],
 out int pPayloadNumber){
  void main(void){
    int i=0,k=0;
    byte tmp,tmpFirst;
    while(i<pCPCSPDUFrameLength){
      pPayloads[k].put(pCDFrame,i,ATMCELLDATALENGTH);
      i += ATMCELLDATALENGTH;
      k++;
    }
    pATMCellHeaderFrame.crc8(pATMCellHeaderFrame,pATMCellHeaderType_x);
    pATMCellHeaderFrame.set(15,1,1);
    pATMCellHeaderFrame.crc8(pATMCellHeaderFrame,pATMCellHeaderType_x);
  }
};

behavior AtmRecovery(in bit pCDData,
     in boolean pIsInvalidFrame,
     in boolean pIsFailCRC,
     in boolean pIsValidCRC,
     out boolean pIsHaultCapturing){
  void main(void){
    int ExitTime,LoopTime,ValidFrameCount;
    CbitBuffer Flag;
    if(pIsInvalidFrame==true||pIsCRCFail==true){
      ValidFrameCount++;
    }
    if(pIsCRCValid==true){
      ValidFrameCount = 0;
    }
    if(ValidFrameCount >= CONTEXTSWITCHTIME){
      pIsHaultCapturing = true;
```

```
      ExitTime = 0;
      LoopTime = 0;
      while(true){
LoopTime = LoopTime + 1;
Flag.put(1,pCDData);
if(flag == CDFFLAG){
  ExitTime = LoopTime + 1;
}
if(ExitTime == LoopTime){
  pIsHaultCapturing = false;
  ValidFrameCount = 0;
  break;
}
      }
    }
    return;
  }
};

behavior AtmUiFormating(inout CbitBuffer pCDFrame,
in int pCDFrameLength,
out int pCPCSPDUFrameLength){
  void main(void){
    int payloadLength,padLength,i;
    ushort tmpFirst;
    ulong tmp;
    int length = pCDFrame.getLength();
    payloadLength = length - 32;
    padLength = ATMCELLDATALENGTH-((payloadLength+64)%(ATMCELLDATALENGTH));
    pCDFrame.get(16);
    pCDFrame.getHead(16);
    pCDFrame.putHead(0,padLength+64);
    pCPCSPDUFrameLength = payloadLength+padLength+64;
    i = 0;
    while(payloadLength != 0){
      pCDFrame.set(pCPCSPDUFrameLength-32-i,1,payloadlength&0x0001);
      i++;
      payloadLength>>=1;
    }
    tmpFirst = pCDFrame.peek(pCPCSPDUFrameLength-32,1);
    tmp = pCDFrame.peek(pCDFrameLength-65,32);
    i++;
    while(i<pCPCSPDUFrameLength-32){
      if(tmpFirst == 1){
```

```
tmpFirst = (tmpFirst^0x1)&0x1;
tmp ^= CRC32EQN;
      }
      else{
tmpFirst = tmp&0x10000000;
tmp = (tmp << 1)|pCDFrame.peek(pCDFrameLength-33-i,1);
i++;
      }
    }
    if(tmpFirst == 1){
      tmpFirst = (tmpFirst^0x1)&0x1;
      tmp ^= CRC32EQN;
    }
    pCDFrame.set(pCPCSPDUFrameLength-32,32,tmp);
  }
};


behavior Main( void ) {
  int         fCDData;
  int         fATMData;
  EAtmFilter  fFilter( fCDData, fATMData );

  void  main( void ) {
    par {
      ...
      fFilter.main();
      }
    }
  };
```

9 Appendix C

```
/**********************************************************************/
/*         Title: HDLCFrame to ATM filter         */
/*    Written by: Tatsuya Umezaki */
/*          Site: CADLAB, University of California, Irvine */
/* Last Modified: 961016 */
/*   Description: Filter accepts a stream of HDLCFrames bitwise serial */
/*                Performs format and crc checks and then produces */
/*                ATM packets to be output bitwise serial. */
/**********************************************************************/

#include<stdio.h>
#include<string.h>

void shift(str)
/**************************************************/
/*NAME:SHIFT                                      */
/*DATE:Sat Sep  7 14:35:41 PDT 1996               */
/*DATE:Sat Sep  7 14:35:41 PDT 1996               */
/*DATE:Sat Sep  7 14:35:41 PDT 1996               */
/**************************************************/
char* str;
{
   char* cp;
   cp = str + 1;
   strcpy(str,cp);
}

int xor(str)
/**************************************************/
/*NAME:SHIFT                                      */
/*DATE:Sat Sep  7 14:35:41 PDT 1996               */
/*DATE:Sat Sep  7 14:35:41 PDT 1996               */
/*DATE:Sat Sep  7 14:35:41 PDT 1996               */
/**************************************************/
char* str;
{
        char crc[20];
        int i;
        int crclength = 17 ;

        crc[0] = '1';crc[1] = '0';crc[2] = '0';crc[3] = '0';
        crc[4] = '1';crc[5] = '0';crc[6] = '0';crc[7] = '0';
        crc[8] = '0';crc[9] = '0';crc[10]= '0';crc[11]= '1';
```

```
        crc[12]= '0';crc[13]= '0';crc[14]= '0';crc[15]= '0';
        crc[16]= '1';crc[17]= '\0';

        for(i=0;i<crclength;i++){
                if((str[i]=='0')&&(crc[i]=='0')) str[i] = '0';
                else if((str[i]=='0')&&(crc[i]=='1')) str[i] = '1';
                else if((str[i]=='1')&&(crc[i]=='0')) str[i] = '1';
           else str[i] = '0';
   }
}

void crc16(p_HDLC,BUF)
/***************************************************/
/*NAME:SHIFT                                       */
/*DATE:Sat Sep  7 14:35:41 PDT 1996                */
/*DATE:Sat Sep  7 14:35:41 PDT 1996                */
/*DATE:Sat Sep  7 14:35:41 PDT 1996                */
/***************************************************/
char* p_HDLC;
char BUF[17];
{
   char str[7000];
   char c;
   int i,strlength;

   strcpy(str,p_HDLC);
   strlength = strlen(str);
   strcat(str,"0000000000000000");
   i = 0;

   while(i<strlength){
           if(str[0] == '0'){
                   shift(str);
                   i++;
           }
           else{
                   xor(str);
           }
   }
   if(strlen(str)!=16){
           printf("FAIL FOR CRC CALICULATION\n");
           exit(1);
   }
   strcpy(BUF,str);
```

```
}
zeroinst(FRAME)
/**************************************************/
/*NAME:ZEROINST                                   */
/*DATE:Sat Oct 25 14:08:32 JST 1997               */
/*Function:ZERO INSERTION TO HDLC_FRAME           */
/*AUTHER NAME:Tatsuya Umezaki                     */
/**************************************************/
char* FRAME;
{
   int one,i,j;
   char BUF[7000];

   one = 0;
   i = 0;
   j = 0;

   while(FRAME[i] != '\0'){
           if(FRAME[i] == '1') one++;
           if(FRAME[i] == '0') one = 0;
           BUF[j] = FRAME[i];
           i++;
           j++;
           if(one == 5){
                   BUF[j] = '0';
                   one = 0;
                   j++;
           }
   }
   BUF[j] = '\0';
   strcpy(FRAME,BUF);
}
void main()
/**************************************************/
/*NAME:HDLC_GEN                                   */
/*DATE:Tue Sep  9 16:12:17 JST 1997               */
/*Function:HDLC FRAME GENERATION                  */
/*AUTHER NAME:Tatsuya Umezaki                     */
/**************************************************/
{
   int max_length,bit_length;
   char c;
   char BUF_16[17];
   char FRAME[7000];
```

```
   max_length = 10000;
                                /* DATA INPUT BIT BY BIT and LENGTH
CHECK */
   bit_length = 0;
   while((c=getchar())!=EOF){
           if(c != '\n'){
                   FRAME[bit_length] = c;
                   bit_length++;
                   if(bit_length>max_length){
                           printf("BIT LENGTH ERROR\n");
                           exit(1);
                   }
           }
   }
   FRAME[bit_length] = '\0';

                                /* ADDITION of CRC16 */
   crc16(FRAME,BUF_16);
   strcat(FRAME,BUF_16);

                                /* 0 INSERTION */
   zeroinst(FRAME);

                                /* START FLAG */
        printf("01111110\n");
                                /* HDLC BODY */
        printf("%s\n",FRAME);
                                /* START FLAG */
        printf("01111110\n");
}

====================================================
head_tr.c
====================================================
#include<stdio.h>
#include<string.h>
main()
/*************************************************/
/*NAME:HEADER_TRANSFORMATION                      */
/*DATE:Tue Oct 28 18:33:24 JST 1997               */
/*Function:HEADER_TRANSFORM FROM HDLC TO ATM      */
/*AUTHER NAME:Tatsuya Umezaki                     */
/*************************************************/
```

```
{
   int i,j,k;
   char ATM_HEAD[5][9];
   char c;
   char HDLC_HEAD[2][9];

   i = 0;
   j = 0;
/* INITIATE ATM_HEAD */
   for(k=0;k<5;k++){
           strcpy(ATM_HEAD[k],"00000000");
   }

/* INPUT CHARACTER */
   while((c=getchar())!=EOF){

/* IGNORE RETURN */
                if(c != '\n'){

/* INPUT CHARACTER CHECK 0/1 */
/*
                   if((c == '0')||(c == '1')){
                   }
                   else{
                           printf("NON 0/1 CHARACTER APPEARED\n");
                           exit(1);
                   }
*/
/* HDLC_HEADER STORING */
                   if((j >= 0)&&(j < 8)){
                           i = 0;
                           HDLC_HEAD[i][j] = c;
                           HDLC_HEAD[i][j+1] = '\0';
                           j++;
                   }
                   else if((j >= 8)&&(j < 16)){
                           i = 1;
                           HDLC_HEAD[i][j-8] = c;
                           HDLC_HEAD[i][j-8+1] = '\0';
                           j++;
                   }
                   else if(j == 16) break;
                   else{
                           printf("HDLC_HEAD INDEX ERROR\n");
```

```
                           exit(1);
                   }
                }
        }
/* MAPPING FROM DFA TO VCI 1 */
   ATM_HEAD[2][0] = HDLC_HEAD[0][6];
   ATM_HEAD[2][1] = HDLC_HEAD[0][7];
/* MAPPING FROM DFA TO VPI */
   ATM_HEAD[1][4] = HDLC_HEAD[0][2];
   ATM_HEAD[1][5] = HDLC_HEAD[0][3];
   ATM_HEAD[1][6] = HDLC_HEAD[0][4];
   ATM_HEAD[1][7] = HDLC_HEAD[0][5];
/* MAPPING FROM DFA TO VCI 2 */
   ATM_HEAD[3][4] = HDLC_HEAD[1][4];
   ATM_HEAD[3][5] = HDLC_HEAD[1][5];
   ATM_HEAD[3][6] = HDLC_HEAD[1][6];
   ATM_HEAD[3][7] = HDLC_HEAD[1][7];
/* MAPPING FROM CN TO EFCI */
   ATM_HEAD[3][2] = HDLC_HEAD[1][3];
/* MAPPING FROM CLP TO CLP */
   ATM_HEAD[3][0] = HDLC_HEAD[1][1];

/* OUTPUT ATM_HEAD */
   for(k=0;k<5;k++){
           printf("%s\n",ATM_HEAD[k]);
   }
}

====================================================
cpcs_gen.c
====================================================

#include<stdio.h>
#include<string.h>
void lentobit(LENGTH,BUF)
/*************************************************/
/*NAME:LENTOBIT                                  */
/*DATE:Thu Sep  4 13:07:28 JST 1997              */
/*Function:CONVERT LENGTH INTO BIT STRINGS       */
/*AUTHER NAME:Tatsuya Umezaki                    */
/*************************************************/
int LENGTH;
char* BUF;
{
```

```
   char TMP[17],OBJ[17];
   int i,REST,strlength;

   i=0;
   while(LENGTH!=0){
           REST=LENGTH%2;
           if(REST == 1) TMP[i]='1';
           else TMP[i]='0';
           LENGTH=LENGTH/2;
           i++;
   }
   TMP[i]='\0';

   strlength = strlen(TMP);
   strcpy(OBJ,"0000000000000000");
   for(i=0;i<strlength;i++){
           OBJ[15-i]=TMP[i];
   }
   strcpy(BUF,OBJ);
}
/**************************************************/
/*LOAD MODULE NAME:crc32                          */
/*DATE:Sat Sep  7 14:35:41 PDT 1996               */
/*Function:CRC32 caliculation                     */
/*AUTHER NAME:Tatsuya Umezaki                     */
/**************************************************/
void shift(str)
/**************************************************/
/*NAME:SHIFT                                      */
/*DATE:Sat Sep  7 14:35:41 PDT 1996               */
/*Function:1 bit left shift                       */
/*AUTHER NAME:Tatsuya Umezaki                     */
/**************************************************/
char* str;
{
   char* cp;
   cp = str + 1;
   strcpy(str, cp);
}

int xor(str)
/**************************************************/
/*NAME:XOR for CRC32                              */
/*DATE:Sat Sep  7 14:35:41 PDT 1996               */
```

```
/*Function:XOR for CRC32                              */
/*AUTHER NAME:Tatsuya Umezaki                         */
/*****************************************************/
char* str;
{
   char crc[34];
   int i;
   int crclength = 33 ;


   crc[0] = '1';crc[1] = '0';crc[2] = '0';crc[3] = '0';crc[4] = '0';
   crc[5] = '0';crc[6] = '1';crc[7] = '0';crc[8] = '0';crc[9] = '1';
   crc[10]= '1';crc[11]= '0';crc[12]= '0';crc[13]= '0';crc[14]= '0';
   crc[15]= '0';crc[16]= '1';crc[17]= '0';crc[18]= '0';crc[19]= '0';
   crc[20]= '1';crc[21]= '1';crc[22]= '1';crc[23]= '0';crc[24]= '1';
   crc[25]= '1';crc[26]= '0';crc[27]= '1';crc[28]= '1';crc[29]= '0';
   crc[30]= '1';crc[31]= '1';crc[32]= '1';crc[33]= '\0';

   for(i=0;i<crclength;i++){
           if((str[i]=='0')&&(crc[i]=='0')) str[i] = '0';
           else if((str[i]=='0')&&(crc[i]=='1')) str[i] = '1';
           else if((str[i]=='1')&&(crc[i]=='0')) str[i] = '1';
           else str[i] = '0';
   }
}

void crc32(p_CPCSPDU,BUF)
/*****************************************************/
/*NAME:crc32                                          */
/*DATE:Sat Sep  7 14:35:41 PDT 1996                   */
/*Function:MAIN CONTROLER for CRC32                   */
/*AUTHER NAME:Tatsuya Umezaki                         */
/*****************************************************/
char* p_CPCSPDU;
char BUF[33];
{
   char str[5000];
   char c;
   int i,strlength;

   strcpy(str,p_CPCSPDU);
   strlength = strlen(str);
   strcat(str,"00000000000000000000000000000000");
        i = 0;
```

```
   while(i<strlength){
           if(str[0] == '0'){
                   shift(str);
                   i++;
           }
           else{
                   xor(str);
           }
   }
   if(strlen(str)!=32){
           printf("FAIL FOR CRC CALUCULATION\n");
           exit(1);
   }
   strcpy(BUF,str);
}
CPCS(HDLC_READY,PAYLOAD_LENGTH,HDLC_FRAME,p_CPCS_BUSY,p_CPCS_READY,
p_CPCSPDU,p_CPCSPDU_LENGTH)
/*****************************************************/
/*NAME:CPCS                                          */
/*DATE:Thu Sep  4 18:03:44 JST 1997                  */
/*Function:CPCS PROCESS                              */
/*AUTHER NAME:Tatsuya Umezaki                        */
/*****************************************************/
char HDLC_READY;
int PAYLOAD_LENGTH;
char HDLC_FRAME[5000];
char *p_CPCS_BUSY;
char *p_CPCS_READY;
char *p_CPCSPDU;
int *p_CPCSPDU_LENGTH;
{
   int HDLC_PAYLOAD_LENGTH;
   int PAD_LENGTH;
   int i;
   char BUF_17[17];
   char BUF_33[33];

   *p_CPCS_READY = '0';
   if(HDLC_READY==1){
           *p_CPCS_BUSY = '1';

/* PAD STAFFING */
           HDLC_PAYLOAD_LENGTH = PAYLOAD_LENGTH ;
```

```
          PAD_LENGTH = 384 - ((HDLC_PAYLOAD_LENGTH+64)%384);
          strncpy(p_CPCSPDU,HDLC_FRAME,HDLC_PAYLOAD_LENGTH);
          *p_CPCSPDU_LENGTH = HDLC_PAYLOAD_LENGTH + PAD_LENGTH + 64;
          for(i=0;i<PAD_LENGTH;i++) strcat(p_CPCSPDU,"0");

/* CPCS-UU & CPI */
          strcat(p_CPCSPDU,"00000000"); /* CPCS-UU */
          strcat(p_CPCSPDU,"00000000"); /* CPI */

/* LENGTH ADDITION */
          lentobit(HDLC_PAYLOAD_LENGTH,BUF_17);
          strcat(p_CPCSPDU,BUF_17);

/* CRC CALICULATION & ADDITION */
          crc32(p_CPCSPDU,BUF_33);
          strcat(p_CPCSPDU,BUF_33);

          *p_CPCS_READY = '1';
          *p_CPCS_BUSY = '0';
   }
}
void main()
/*************************************************/
/*NAME:MAIN                                      */
/*Thu Sep  4 13:07:28 JST 1997                   */
/*Function:TEST BENCH FOR CPCS                   */
/*AUTHER NAME:Tatsuya Umezaki                    */
/*************************************************/
{
   int   i;
   char  c;
   char HDLC_READY;
   int FRAME_LENGTH;
   int PAYLOAD_LENGTH;
   char *p_HDLC_FRAME;
   char HDLC_FRAME[5000];
   char CPCS_BUSY;
   char CPCS_READY;
   int CPCSPDU_LENGTH;
   char CPCSPDU[5000];
   char LINE[20];

/* INPUT FOR CPCS PROCESS*/
   HDLC_READY = 1;
```

```
/* SET HDLC_FRAME BUFFA */
   FRAME_LENGTH = 0;
   PAYLOAD_LENGTH = 0;
   while((c = getchar())!=EOF){
           if(c != '\n'){
                   if(FRAME_LENGTH < 16){
                   }
                   else{
                           HDLC_FRAME[PAYLOAD_LENGTH] = c;
                           PAYLOAD_LENGTH++;
                   }
                   FRAME_LENGTH++;
           }
   }
   HDLC_FRAME[PAYLOAD_LENGTH] = '\0';
/* CPCS PROCESS */
   CPCS(HDLC_READY,PAYLOAD_LENGTH,HDLC_FRAME,&CPCS_BUSY,&CPCS_READY,
CPCSPDU,&CPCSPDU_LENGTH);

/* OUTPUT FROM CPCS PROCESS */
   i = 0;
   while(CPCSPDU[i]!='\0'){
           putchar(CPCSPDU[i]);
           i++;
           if((i%8) == 0) printf("\n");
   }
   printf("\n");
}

==================================================
atm_seg.c
==================================================
#include<stdio.h>
#include<string.h>

void shift(str)
/*************************************************/
/*NAME:SHIFT                                     */
/*DATE:Sat Sep  7 14:35:41 PDT 1996              */
/*Function:SHIFT OPERATION                       */
/*AUTHER NAME:Tatsuya Umezaki                    */
/*************************************************/
char* str;
{
```

```
        char* cp;
        cp = str + 1;
        strcpy(str,cp);
}

int xor(str)
/*************************************************/
/*NAME:XOR                                      */
/*DATE:Sat Sep  7 14:35:41 PDT 1996             */
/*Function:XOR OPERATION                        */
/*AUTHER NAME:Tatsuya Umezaki                   */
/*************************************************/
char* str;
{
        char crc[20];
        int i;
        int crclength = 9;

        crc[0] = '1';crc[1] = '0';crc[2] = '0';crc[3] = '0';
        crc[4] = '0';crc[5] = '0';crc[6] = '1';crc[7] = '1';
        crc[8] = '1';crc[9] = '\0';

        for(i=0;i<crclength;i++){
                if((str[i]=='0')&&(crc[i]=='0')) str[i] = '0';
                else if((str[i]=='0')&&(crc[i]=='1')) str[i] = '1';
                else if((str[i]=='1')&&(crc[i]=='0')) str[i] = '1';
                else str[i] = '0';
        }
}

void crc8(p_HDLC,BUF)
/*************************************************/
/*NAME:CRC_8                                    */
/*DATE:Sat Sep  7 14:35:41 PDT 1996             */
/*Function:CRC_8 CALICULATION                   */
/*AUTHER NAME:Tatsuya Umezaki                   */
/*************************************************/
char* p_HDLC;
char BUF[9];
{
        char str[5000];
        char c;
        int i,strlength;
```

```
        strcpy(str,p_HDLC);
        strlength = strlen(str);
        strcat(str,"00000000");
        i = 0;

        while(i<strlength){
                if(str[0] == '0'){
                        shift(str);
                        i++;
                }
                else{
                        xor(str);
                }
        }
        if(strlen(str)!=8){
                printf("FAIL FOR CRC CALICULATION\n");
                exit(1);
        }
        strcpy(BUF,str);
}

void main()
/**************************************************/
/*NAME:ATM_GEN                                    */
/*DATE:Wed Oct 29 16:33:09 JST 1997               */
/*Function:ATM SEGMENTATION                       */
/*AUTHER NAME:Tatsuya Umezaki                     */
/**************************************************/
{
   int length;
   int cell_num;
   int i,j;
   char c;
   char BUF[5000];
   char HEAD_BUF[50];
   char TMP_BUF[50];
   char CPCS_BUF[5000];
   char BUF_8[9];
   char HEAD_TYPE_0[50];
   char HEAD_TYPE_1[50];
   char CELL_BUF[400];

/* CAT_F INPUT */
   length = 0;
```

```
        while((c=getchar())!=EOF){
                if(c != '\n'){
                        BUF[length] = c;
                        length++;
                }
        }
        BUF[length] = '\0';

/* LENGTH CHECK */
   if(length%384 != 40){
                printf("HEAD+CPCS LENGTH CHECK ERROR:LENGTH:%d\n",
length);
                exit(1);
   }

/* SEPARATE HEAD AND CPCS */
   strncpy(HEAD_BUF,BUF,32);
   strcpy(CPCS_BUF,BUF+40);

/* MAKE HEAD OF SDU TYPE_0 */
   strcpy(TMP_BUF,HEAD_BUF);
   TMP_BUF[25] = '0';
   crc8(TMP_BUF,BUF_8);
   strcat(TMP_BUF,BUF_8);
   strcpy(HEAD_TYPE_0,TMP_BUF);

/* MAKE HEAD OF SDU TYPE_1 */
   strcpy(TMP_BUF,HEAD_BUF);
   TMP_BUF[25] = '1';
   crc8(TMP_BUF,BUF_8);
   strcat(TMP_BUF,BUF_8);
   strcpy(HEAD_TYPE_1,TMP_BUF);

/* OUTPUT ATM CELL */
   cell_num = (length-40)/384;
   for(i=0;i<cell_num;i++){
           if(i == cell_num-1) printf("%s\n",HEAD_TYPE_1);
           else printf("%s\n",HEAD_TYPE_0);
           strncpy(CELL_BUF,(CPCS_BUF+i*384),384);
           printf("%s\n",CELL_BUF);
```