SpecC RTL Methodology

Pei Zhang, Dongwan Shin, Haobo Yu, Qiang Xie, Daniel D. Gajski
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

Abstract

This report describes the SpecC RTL methodology, using a specific example (one's counter). We first begin with the introduction of one's counter. Then the behavioral view and the structural view of the SpecC implementation model are given to implement the one's counter function and mappings in different RTL style. The source codes are also included in the Appendix.

1 Introduction

The SpecC's Methodology [GZDG00][GERS00] uses SpecC system-style design language to implement a system from the specification model to manufacturing-ready RTL implementation model. In this report, we mainly focus on the backend of whole system processing, from communication model to implementation model. In the architecture of SpecC methodology, it is called SpecC RTL methodology.

In this report, we use a specific example, one's counter, to explain the SpecC RTL methodology. There are five styles [GAJS00] and mainly two views in the implementation model [GERS00]. We will explain how to use SpecC RTL methodology to implement the mappings of these five styles and the full functions of two views in the following sections.

2 One's Counter

In order to setup the RTL design flow in SpecC environment, we experiment the RTL implementation model using one's counter which computes the number of one in specified input. We also implement behavioral RTL model and structural RTL model for one's counter in well-defined procedure.

The function of one's counter is to count the number of one in an integer number. Figure 1 shows the Finite-State Machine (FSM) specification of one's counter. The FSM has eight states and transitions from one state to another under the control of external signal *Start* and the status signal *Date*. Here, we use variable assignment statement to indicate changes in variable values describe in the datapath operation. It is called an FSM with data, or FSMD.

One's counter specification also can be expressed by a **state-action table**, which is reduced from a **state and output table** [GAJS00].

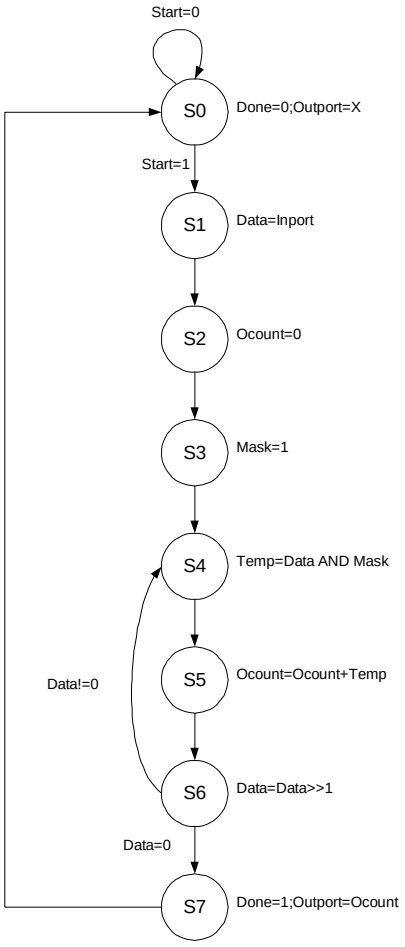

Figure 1 One's counter specification

3 RTL Description

3.1 Introduction of RTL

In the RTL implementation model, there are two parts: a controller and datapath [GAJS00], just as showed in figure 2. The datapath consists of sequential storage units and combinatorial units. Usually, in one clock cycle, the datapath takes operand from storage units, performs the computation in the combinatorial units, and returns the results to the storage units during every state. The controller controls the datapath's action by setting proper values of datapath control signals. The datapath also affects some status signals that will be used in controller.

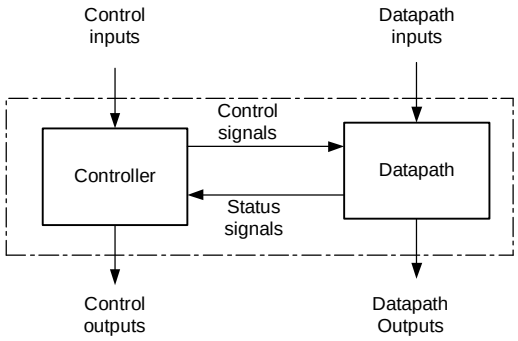

Figure 2 RTL Design Model

3.2 Five styles of RTL

There are five different RTL styles that represent different mappings from high layer to implementation layer.

Style 1: Behavioral (unmapped RTL)
In the behavioral RTL, the variables are divided into ports and internal variables, while ports are further divided into control and data ports, where each could be an input, output or input-output port. Behavioral RTL only specifies the change of values for some variables in each state. The order in which assignments are executed is determined by control dependencies, that is, the order written in the description. States, transitions and assignment statements are in no way related to any implementation. The variables do not represent registers or busses and functions or operations do not represent any functional units.

Style 2: Storage mapped RTL
The variables in style 1 can be of two types. One type is variables whose value is used in the same state in which that value is assigned. These variables will be implemented as wires or busses in the final implementation. The other type is variables whose values are assigned in one state and used in other state. The states between the value assignment and its last usage define the lifetime of each variable. These variables must be mapped to storage units such as register, register files, and memories in the final implementation. Thus style 2 represents RTL description in which the second type of variables with non-overlapping lifetimes are grouped and assigned to storage units. In other words, a group of internal variables in replaced by a new variable of type storage.

Style 3: Function mapped RTL
In style 3, the operators and/or functions with non-overlapping lifetimes are grouped into functional units, and a control encoding is assigned to each operation in the functional unit. Therefore, in style 3 we must identify the operation performed by each function unit in each state. Style 3 is the same as style 2 with functions replaced by multi-operation functional units. Note, that original functions and functions representing functional units use the same syntax.

Style 4: Connection mapped RTL
Similarly to style 2, the variables, with non-overlapping life times, that represent wires as well as inputs and outputs to storage elements and functional units are grouped and assigned to busses. Syntactically, there is no difference between wires and buses. The only difference is in additional bus drivers that must be inserted in style 5. Similarly, we can merge (multiplex) ports if they are not used at the same time.

Style 5: Control mapped RTL
In style 5, the FSMD implementation is described in two parts: netlist of datapath components and a controller that assign a constant to each control variable in each state. The control variables specify the operation for each storage, functional or bus component in the datapath.

In fact, a FSMD is mostly like a behavioral RTL view (style 1) of the implementation model, and

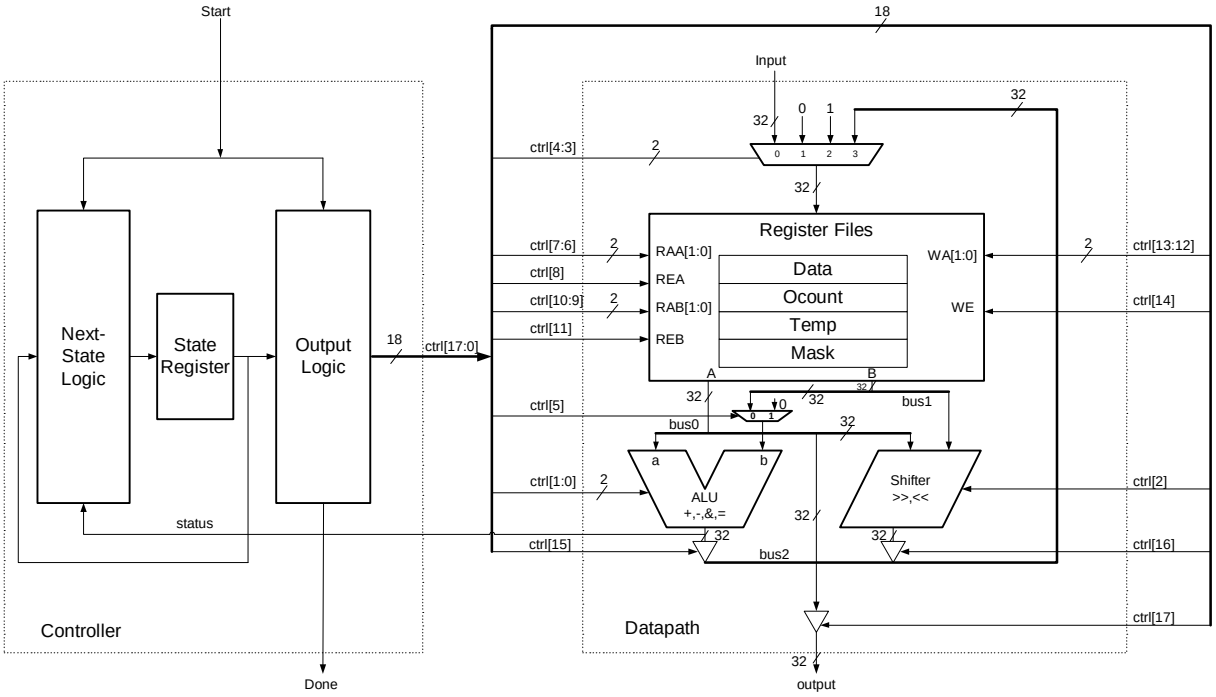

Figure 3 Diagram of the one's counter's structural RTL view

can be implemented easily by C, VHDL or SpecC. A SpecC implementation of behavioral view is discussed in the section 3.3.

In the structural RTL view (after style 5), The controller decides the state transition and sends control signals to the datapath in every state and the datapath takes the all the computation using register files, ALU and shifter and returns a status signal (*Data*) to the controller. The detail of the structural RTL will be discussed in the section 3.4.

3.3 Behavioral RTL View

The operation of the one's counter is showed in figure 1. It is specified by an FSMD, representing the control unit and a set of variable assignments representing register transfers in the datapath. The FSMD has eight states and transitions from one state to another under the control of the external signal Start and the status signal (Data=0). In each state, the FSMD assigns values to a set of datapath control signals, which completely specifies the behavior of the datapath.

We implement the Behavioral RTL model for one's counter in SpecC languange, which is shown in Appendix A. It is composed of two part: the one is for FSMD behavioral model of one's counter (one_counter behavior), whose operations are just like the FSMD in the figure 1, and the other is testbench which assigns the input to Inport(Input behavior) and get an output from Outport to verify the operation of one's counter(Output behavior). Several 32-b bit type local variables are used to store the some value: Data/Ocount/Temp/Mask. Channel idone is used to notify the done event to the test bench. Channel iStart is used to synchronize between to get input from standard input and to generate output to standard output.

3.4 Structural RTL View

The one's counter structural RTL model consists of two parts: the control unit and the datapath. The control unit is an FSM, which contains a state register and two combinatorial blocks computing the next state and output functions. The input to the control unit is the start signal. The output is done signal and control signals for the datapath.

The datapath unit computes the number of one's in given input data. It accepts the inport data as its input and calculates the number of one's in the inport data. The datapath is composed of three

parts: the storage unit, the functional unit and the busses as the connection between them. In the one's counter, we used one ALU and one shift as the functional unit and we use register file with one write port and two read ports to store the four variables (Data / Ocount / Temp / Mask). Also there are two busses to connect register file read port with the input of ALU and shifter, and one bus to connect the output of ALU and shifter with write port of register file. The diagram of the structural RTL view is showed in figure 3 and the source code is included in Appendix B.

STATE TRANSAC-TIONS	STYLE 1 BEHAVIORAL (UNMAPPED) RTL	STYLE 2 STORAGE MAPPED RTL	STYLE 3 FUNCTION MAPPED RTL	STYLE 4 CONNECTION MAPPED RTL	STYLE 5 CONTROL MAPPED RTL
S0	Done=0 Start=0	Done=0 Start=0	Done=0 Start=0	Done=0 Start=0	0X_XXXX_XXXX_XXXX_XXXX
S1	Data=Inport	RF(0)=Inport	RF(0)=Inport	RF(0)=Inport	00_0100_XXXX_XXX0_0XXX
S2	Ocount=0	RF(1)=0	RF(1)=0	RF(1)=0	00_0101_XXXX_XXX0_1XXX
S3	Mask=1	RF(3)=1	RF(3)=1	RF(3)=1	00_0111_XXXX_XXX1_0XXX
S4	Temp=Data&Mask	RF(2)= f&(RF(0),RF(3))	RF(2)= ALU(&,RF(0),RF(3))	Bus0=RF(0) Bus1=RF(3) Bus2=ALU(&,bus0,bus1) RF(2)=bus2	00_1110_1111_0001_1X10
S5	Ocount=Ocount+Temp	RF(1)= f+(RF(1),RF(2))	RF(1)=ALU(+,RF(1), RF(2))	Bus0=RF(1) Bus1=RF(2) Bus2=ALU(+,bus0,bus1) RF(1)=bus2	00_1101_1101_0101_1X00
S6	Data=Data>>1	RF(0)= f>>(RF(0),RF(3))	RF(0)=shift(>>,RF(0),RF(3))	Bus0=RF(0) Bus1=RF(3) Bus2=shift(>>,bus0,bus1) RF(0)=bus2	01_0100_1111_0011_1011
S7	Done=1 Outport=Ocount	Done=1 Outport=RF(1)	Done=1 Outport=RF(1)	Done=1 Outport=RF(1)	1X_XXXX_XXX1_01XX_XXXX
MAPPINGS	Storage mappings {Data,Ocount,Temp ,Mask}=RF	Function mappings {f+,f-,f&,f|}=ALU {>>,<<}=shift	Connection mappings RF_to_ALUL=bus0 RF_to_ALUR=bus1 RF_to_shiftL=bus0 RF_to_shiftR=bus1 ALU_to_RF=bus2 shift_to_RF=bus2	Control mappings ALU=C0,C1 Shift=C2 Inport_to_RF=C3,C4 Bus2_to_RF=C3,C4 0_to_RF=C3,C4 1_to_RF=C3,C4 Bus1_to_ALU=C5 0_to_ALU=C5 RF=C6,C7,C8,C9,C10,C11,C12,C13,C14 RF_to_bus0=C6,C7,C8 RF_to_bus1=C9,C10,C11 ALU_to_bus2=C15 Shift_to_bus2=C16 Bus2_to_Outport=C17	

Figure 4. State diagram with different styles and mappings

3.5 Styles mappings

Five different RTL styles for the one's counter implementation and necessary mappings [GAJS00] to refine one style to the other are shown in Figure 4.

Style 1: Behavioral (unmapped RTL)
Just as we mentioned before, style 1 (behavioral RTL) is equivalent to the programming languagecode with exception that such code is divided into states, with conditional transition between states.

Style 2: Storage mapped RTL
We assigned the four variables Data, Ocount, Temp, Mask to the four registers in the register file with 2 read ports and 1 write port. This assignment is shown in storage mapping table at the bottom of Style 1 in Figure 4. Note that in Style 2 description we used, for uniformity sake, functional notation for all operators.

Style 3: Function mapped RTL
We used two functional units: ALU perform addition, subtraction, AND operation, OR operation, Shift performs the right shift and left shift operation.

Style 4: Connection mapped RTL
In our example, connections from register file to ALUL and shiftL are assigned to bus0, connections from register file to ALUR and shiftR are assigned to bus1, while connections from ALU and shift to register file are assigned to bus2.

Style 5: Control mapped RTL
Control mappings to perform the style 4 assignments are shown in control mapping table. Thus the transfers and operations are replaced by assignments to control signals for all storage, functional and bus units. In style 5, we inserted six additional bus drivers to control the bus.

4 Summary and Conclusions

In this report, we apply SpecC RTL design methodology into the design of a one's counter. We implement the behavioral view and the structural view of SpecC implementation model using SpecC language. The mapping table of five different RTL styles for the one's counter implementation is also given. The correctness of output demonstrates the correctness of our model and design.

References

[GZDG00] D. Gajski er al. *SpecC: Specification Language and Design Methodology*, Kluwer Academic Publishers, 2000
[GAJS00] D. Gajski *RTL Design and Methodology*, University of California, Irvine, Technical Report ICS-00-35, November 2000
[GERS00] A. Gerstlauer *SpecC Modeling Guidelines*, University of California, Irvine, Technical Report ICS-00-xx, September 2000
[GAJS97] D. Gajski Principles of Digital Design, Prentice-Hall, Inc, 1997

Appendix A: SpecC code for one's counter (Behavioral RTL)

A.1 bus.sc

```
/*************************************************************************
*  Title: bus.sc
*  Description: Bus Definition
*************************************************************************/
#ifndef __BUS_
#define __BUS_

//Signal channel for representation of control signal
interface iOSignal
{
      void assign ( int v ) ;
};

interface iISignal
{
      int val() ;
      void waitval ( int v ) ;
};

channel cSignal()
      implements iISignal, iOSignal
{
      int value=0;
      event ev;

      void assign ( int v )   // assign a value
      {
            value = v ;
            notify ( ev ) ;
      }

      int val()   // return a value
      {
            return value ;
      }

      void waitval ( int v )  // wait for a value
      {
            while ( value != v )
                  wait ( ev ) ;
      }
};

#endif
```

A.2 chann.sc

```
/*************************************************************************
*  Title: chann.sc
*  Description: channel Definition
*************************************************************************/
```

```
#ifndef __CHANNEL__
#define __CHANNEL__

#include <stdio.h>
#include <stdlib.h>

interface iSendInt {
      void send(int val);
};

interface iRecvInt {
      int receive(void);
};

channel cSyncInt(void) implements iSendInt, iRecvInt
{
      int message;
      bool valid=false;
      event sent, received;

      void send(int val){
            message = val;
            valid=true;
            notify(sent);
            if(valid)
                  wait(received);
      }

      int receive(void){
            int local_message;

            if(!valid)
                  wait(sent);
            local_message = message;
            valid=false;
            notify(received);
            return local_message;
      }
};

#endif
```

A.3 clock_gen.sc

```
/*********************************************************************
*  Title: clock_gen.sc
*  Description: clock generator
*********************************************************************/

#include "global.sh"
import "chann";

behavior clock_gen(out event clk)
{
      void main(void) {
            while (1) {
```

```
                waitfor(HW_CLOCK_PERIOD);
                printf("\nclock event!!!\n");
                notify(clk);
          }
     }
};
```

A.4 global.sh

```
/*********************************************************************
*  Title: global.sh
*  Description: Global Definition
*********************************************************************/
#ifndef __GLOBAL_H__
#define __GLOBAL_H__

#define HW_CLOCK_PERIOD 10

#endif
```

A.5 io.sc

```
/*********************************************************************
*  Title: io.sc
*  Description: input/outport for testbench
*********************************************************************/

#include "global.sh"

import "bus";
import "chann";

// get 32-bit bit vector from stdin
behavior Input(out bit[0:0] rst, out bit[31:0] inport,
     iOSignal start, iRecvInt isync)
{
     void main(void) {
          char buf[16];

          rst = 1b;         // reset all storage elements in design
during 2 clock cycles
          start.assign(0); // maintain start signal low during 2
clock cycle
          waitfor ( HW_CLOCK_PERIOD );
          waitfor ( HW_CLOCK_PERIOD );

          rst = 0b;         // deassign reset

          while (1) {
               printf("Input for one's counter: ");
               gets(buf);
               inport = atoi(buf);

               start.assign(1);  // now, design calculates num. of
one in inport
```

```
                isync.receive();  // I/O synchronization
            }
      }
};

// write ocount to stdout
behavior Output(in event clk, in bit[31:0] outport, iOSignal start,
iISignal done, iSendInt isync)
{
      void main(void) {
            while (1) {
                  done.waitval(1);
                  printf("output = %d\n", (int)outport);
                  wait(clk);
                  start.assign(0);
                  isync.send(1);  // I/O synchronization
            }
      }
};
```

A.6 one_counter.sc

```
/*****************************************************************
*  Title: one_counter.sc
*  Description: input/outport for testbench
*****************************************************************/
#ifndef __ONE_COUNTER__
#define __ONE_COUNTER__

import "bus";
import "chann";

behavior One_Counter(in event clk, in bit [0:0] rst, in bit[31:0]
Inport,
      out bit[31:0] Outport, iISignal start, iOSignal done)
{
      void main(void) {
            bit[31:0] Data;
            bit[31:0] Ocount;
            bit[31:0] Mask;
            bit[31:0] Temp;

            enum state { S0, S1, S2, S3, S4, S5, S6, S7 } state;

            state = S0;

            while (1) {
                  wait(clk);
                  if (rst == 1b) {
                        Outport = 0x0000;
                        state = S0;
                  }
                  switch (state) {
                        case S0 :
                              printf("S0\n");
                              done.assign(0);
```

```
                        Outport = -1;
                        if (start.val() == 1)
                              state = S1;
                        else
                              state = S0;
                        break;
                  case S1:
                        printf("S1\n");
                        Data = Inport;
                        state = S2;
                        break;
                  case S2:
                        printf("S2\n");
                        Ocount = 0;
                        state = S3;
                        break;
                  case S3:
                        printf("S3\n");
                        Mask = 1;
                        state = S4;
                        break;
                  case S4:
                        printf("S4\n");
                        Temp = Data & Mask;
                        state = S5;
                        break;
                  case S5:
                        printf("S5\n");
                        Ocount = Ocount + Temp;
                        state = S6;
                        break;
                  case S6:
                        printf("S6\n");
                        Data = Data >> 1;
                        if (Data == 0)
                              state = S7;
                        else
                              state = S4;
                        break;
                  case S7:
                        printf("S7\n");
                        Outport = Ocount;
                        done.assign(1);
                        state = S0;
                        break;
            }
      }
   }
};

#endif
```

A.7 tb.sc

```
/*****************************************************************
*  Title: tb.sc
```

```
*  Description: Testbench for one_counter
*********************************************************************/
import "io";
import "clock_gen";
import "one_counter";

behavior Main
{
      // Channels
      bit[31:0] inport;
      bit[31:0] outport;
      bit[0:0] rst;
      cSignal start;
      cSignal done;
      cSyncInt isync;
      event clk;

      Input input(rst, inport, start, isync);
      Output output(clk, outport, start, done, isync);
      clock_gen   clk_gen(clk);
      One_Counter one_counter(clk, rst, inport, outport, start, done);

      int main (void)
      {
            // Command line arguments
            par {
                  input.main();
                  output.main();
                  clk_gen.main();
                  one_counter.main();
            }
            return 0;
      }
};
```

Appendix B: SpecC code for one's counter (Structural RTL)

B.1 alu.sc

```
/*************************************************************************
*   Title: alu.sc
*   Description: ALU has add/sub/cmp function
*   ctrl    function
*    00      addition
*    01      subtraction
*    10      bitwise and
*    11      eq(sum=1 if a == b, otherwise sum = 0)
*************************************************************************/
behavior alu(in bit[31:0] a, in bit[31:0] b, out bit[31:0] sum, in
bit[1:0] ctrl,
      in event _a, in event _b, out event _sum, in event _ctrl)
{
      void main(void) {
            while (1) {
                  wait (_a, _b, _ctrl);
                  switch(ctrl) {
                        case 00b:   // addition
                              sum = a+b;
                              break;
                        case 01b:   // subtraction
                              sum = a-b;
                              break;
                        case 10b:   // bitwise and
                              sum = a&b;
                              break;
                        case 11b:   // equal
                              if (a == b)
                                    sum = 0x0001;
                              else
                                    sum = 0x0000;
                              break;
                  }
                  notify(_sum);
            }
      }
};
```

B.2 assign.sc

```
/*************************************************************************
*  Title: assign.sc
*  Description: implementation of assign statement in verilog
*************************************************************************/

import "chann";
behavior assign(in bit[0:0] di, out bit[0:0] dout, in event _di, out
event _dout)
{
      void main(void) {
            bit[0:0] data;
            while (1) {
```

```
                wait(_di);
                data = di&1b;
                dout = data;
                notify(_dout);
          }
     }
};
```

B.3 buff.sc

```
/*********************************************************************
*  Title: buff.sc
*  Description: buffer with enable(high active)
*********************************************************************/

behavior buff(in bit[31:0] di, out bit[31:0] dout, in bit[0:0] en, in
event _di,
     out event _dout, in event _en)
{
     void main(void) {
          while (1) {
                wait (_di, _en);
                if (en == 1b) {
                     dout = di;
                }
                notify(_dout);
          }
     }
};
```

B.4 chann.sc

```
/*********************************************************************
*  Title: chann.sc
*  Description: channel definition
*********************************************************************/

#include <stdio.h>
#include <stdlib.h>

interface iSendInt {
     void send(int val);
};

interface iRecvInt {
     int receive(void);
};

channel cSyncInt(void) implements iSendInt, iRecvInt
{
     int message;
     bool valid=false;
     event sent, received;

     void send(int val){
          message = val;
          valid=true;
          notify(sent);
```

```
            if(valid)
                  wait(received);
      }

      int receive(void){
            int local_message;

            if(!valid)
                  wait(sent);
            local_message = message;
            valid=false;
            notify(received);
            return local_message;
      }
};
```

B.5 clock_gen.sc

```
/*********************************************************************
*  Title: clock_gen.sc
*  Description: two phase clock generator
*********************************************************************/

#include "global.sh"
import "chann";

behavior clock_gen(out event clk)
{
      void main(void) {
            while (1) {
                  waitfor(HW_CLOCK_PERIOD);
                  printf("\nclock event!!!\n");
                  notify(clk);
            }
      }
};
```

B.6 controller.sc

```
/*********************************************************************
*  Title: controller.sc
*  Description: controller for one's counter(hamming encoded state
register)
*********************************************************************/

import "state_reg";
import "output_logic";
import "nstate_logic";

behavior controller(in event clk, in bit[0:0] rst, in bit[0:0] start,
      in bit[0:0] status, out bit[0:0] done, out bit[17:0] ctrl, in
event _start, in event _status, out event _done, out event _ctrl)
{
      bit[2:0] state, nstate;
      event _state, _nstate;

      state_reg sr(clk, rst, nstate, state, _state);
      output_logic ol(state, ctrl, done, _state, _ctrl, _done);
```

```
      nstate_logic nsl(state, nstate, start, status, _state, _nstate,
_start, _status);

      void main(void) {
            par {
                  sr.main();
                  ol.main();
                  nsl.main();
            }
      }
};
```

B.7 datapath.sc

```
/*********************************************************************
*  Title: datapath.sc
*  Description: datapath has alu, shift, regs and glue logic
*********************************************************************/
import "regfile_2p";
import "alu";
import "shift";
import "buff";
import "mux2x1";
import "mux4x1";
import "assign";

behavior datapath(in event clk, in bit[0:0] rst, in bit[31:0] inport,
      out bit[31:0] outport, out bit[0:0] status, in bit[17:0] ctrl,
      in event _inport, out event _outport, out event _status, in event
_ctrl)
{
      bit[31:0] const_zero= 0x0000;
      bit[31:0] const_one= 0x0001;

      bit[31:0] bus0, bus1, bus2;
      bit[31:0] rf_in, alu_b;
      bit[31:0] alu_out, shift_out;

      event _bus0, _bus1, _bus2, _rf_in, _alu_b, _alu_out, _shift_out;

      // register files(data/ocount/ocount/mask)
      regfile_2p rf_2p00(clk, rst, rf_in, ctrl[7:6], ctrl[10:9],
ctrl[8], ctrl[11],
        ctrl[13:12], ctrl[14], bus0, bus1, _rf_in, _ctrl, _ctrl, _ctrl,
_ctrl,
            _bus0, _bus1);

      // functional units(ALU/SHIFTER)
      alu alu00(bus0, alu_b, alu_out, ctrl[1:0], _bus0, _alu_b,
_alu_out, _ctrl);
      shift shift00(bus0, bus1[4:0], ctrl[2], shift_out, _bus0, _bus1,
_ctrl, _shift_out);

      // input mux for register file
      mux4x1 mux00(inport, const_zero, const_one, bus2, rf_in,
ctrl[4:3],
            _inport, _ctrl, _ctrl, _bus2, _rf_in, _ctrl);
```

```
      // input mux for alu
      mux2x1 mux01(bus1, const_zero, alu_b, ctrl[5], _bus1, _ctrl,
_alu_b, _ctrl);

      // output buffers of functional units
      buff buff00(alu_out, bus2, ctrl[15], _alu_out, _bus2, _ctrl);
      buff buff01(shift_out, bus2, ctrl[16], _shift_out, _bus2, _ctrl);

      // buffer for outport
      buff buff02(bus0, outport, ctrl[17], _bus0, _outport, _ctrl);

      assign ass00(alu_out[0], status, _alu_out, _status);

      void main(void) {
            par {
                  rf_2p00.main();

                  alu00.main();
                  shift00.main();

                  mux00.main();
                  mux01.main();

                  buff00.main();
                  buff01.main();

                  buff02.main();

                  ass00.main();
            }
      }
};
```

B.8 global.sh

```
/*********************************************************************
*  Title: global.sh
*  Description: Global Definition
*********************************************************************/
#ifndef __GLOBAL_H__
#define __GLOBAL_H__

#define HW_CLOCK_PERIOD 10

#endif
```

B.9 io.sc

```
/*********************************************************************
*  Title: io.sc
*  Description: input/outport for testbench
*********************************************************************/

#include "global.sh"

import "chann";
```

```
// get 32-bit bit vector from stdin
behavior Input(out bit[0:0] rst, out bit[31:0] inport, out bit[0:0]
start,
      out event _inport, out event _start, iRecvInt isync)
{
      void main(void) {
            char buf[16];

            rst = 1b;         // reset all storage elements in design
during 2 clock cycles
            start = 0b; // maintain start signal low during 2 clock
cycle
            notify(_start);
            waitfor ( HW_CLOCK_PERIOD );
            waitfor ( HW_CLOCK_PERIOD );

            rst = 0b;         // deassign reset

            while (1) {
                  printf("Input for one's counter: ");
                  gets(buf);
                  inport = atoi(buf);
                  notify(_inport);
                  waitfor(5);
                  start = 1b; // now, design calculates num. of one in
inport
                  notify(_start);
                  isync.receive();  // I/O synchronization
            }
      }
};

// write ocount to stdout
behavior Output(in bit[31:0] outport, in bit[0:0] done, in event
_outport, in event _done, iSendInt isync)
{
      void main(void) {
            while (1) {
                  wait(_done, _outport);
                  waitfor(1);
                  if (done == 1b) {
                        printf("output = %d\n", (int)outport);
                        isync.send(1);  // I/O synchronization
                  }
            }
      }
};
```

B.10 mux2x1.sc

```
/*****************************************************************
*  Title: mux2x1.sc
*  Description: 2x1 multiplexer
*****************************************************************/

behavior mux2x1(in bit[31:0] d0, in bit[31:0] d1, out bit[31:0]  dout,
```

```
      in bit[0:0] sel, in event _d0, in event _d1, out event _dout, in
event _sel)
{
      void main(void) {
            while (1) {
                  wait (_d0, _d1, _sel);
                  switch (sel) {
                        case 00b:
                              dout = d0;
                              break;
                        case 01b:
                              dout = d1;
                              break;
                  }
                  notify(_dout);
            }
      }
};
```

B.11 mux4x1.sc

```
/*********************************************************************
*  Title: mux4x1.sc
*  Description: 4x1 multiplexer
*********************************************************************/

behavior mux4x1(in bit[31:0] d0, in bit[31:0] d1, in bit[31:0] d2,
      in bit[31:0] d3, out bit[31:0] dout, in bit[1:0] sel, in event
_d0,
      in event _d1, in event _d2, in event _d3, out event _dout, in
event _sel)
{
      void main(void) {
            while (1) {
                  wait (_d0, _d1, _d2, _d3, _sel);
                  switch (sel) {
                        case 00b:
                              dout = d0;
                              break;
                        case 01b:
                              dout = d1;
                              break;
                        case 10b:
                              dout = d2;
                              break;
                        case 11b:
                              dout = d3;
                              break;
                  }
                  notify(_dout);
            }
      }
};
```

B.12 nstate_logic.sc

```
/*********************************************************************
*  Title: nstate_logic.sc
```

```
*  Description: next state  logic(hamming encoded state register)
*********************************************************************/

behavior nstate_logic(in bit[2:0] state, out bit[2:0]  nstate, in
bit[0:0] start,
      in bit[0:0] status, in event _state, out event _nstate, in event
_start,
      in event _status)
{
      void main(void) {
            while (1) {
                  wait(_state, _start, _status);
                  switch (state) {
                        case 000b:        // S0
                              if (start == 0b)
                                    nstate = 000b;
                              else
                                    nstate = 001b;
                              break;
                        case 001b:        // S1
                              nstate = 011b;
                              break;
                        case 011b:        // S2
                              nstate = 010b;
                              break;
                        case 010b:        // S3
                              nstate = 110b;
                              break;
                        case 110b:        // S4
                              nstate = 111b;
                              break;
                        case 111b:        // S5
                              nstate = 101b;
                              break;
                        case 101b:        // S6
                              if (status == 1b) // Data == 0
                                    nstate = 100b;
                              else
                                    nstate = 110b;
                              break;
                        case 100b:        // S7
                              nstate = 000b;
                              break;
                  }
                  notify(_nstate);
            }
      }
};
```

B.13 one_counter.sc

```
/*********************************************************************
*  Title: one_counter.sc
*  Description: top-style structural RTL for one's  counter(bus-based
arch.)
*********************************************************************/
```

```
import "controller";
import "datapath";

behavior one_counter(in event clk, in bit[0:0] rst, in bit[31:0]
inport,
      out bit[31:0] outport, in bit[0:0] start, out bit[0:0] done,
      in event _inport, out event _outport, in event _start, out event
_done)
{
      bit[17:0] ctrl;
      bit[0:0] status;
      event _ctrl, _status;

      controller U1(clk, rst, start, status, done, ctrl, _start,
_status, _done,
            _ctrl);
      datapath U2(clk, rst, inport, outport, status, ctrl, _inport,
_outport,
            _status, _ctrl);

      void main(void) {
            par {
                  U1.main();
                  U2.main();
            }
      }
};
```

B.14 output_logic.sc

```
/*****************************************************************
*  Title: output_logic.sc
*  Description: output logic for controller
*****************************************************************/
import "chann";

behavior output_logic(in bit[2:0] state, out bit[17:0] ctrl, out
bit[0:0] done,
      in event _state, out event _ctrl, out event _done)
{
      void main(void) {
            while (1) {
                  wait(_state);
                  switch (state) {
                        case 000b:        // S0
                              printf("S0(o)\n");
                              ctrl = 000000000000000000b;   //
0X_XXXX_XXXX_XXXX_XXXX
                              done = 0b;
                              break;
                        case 001b:        // S1
                              printf("S1(o)\n");
                              ctrl = 000100000000000000b;   //
00_0100_XXXX_XXX0_0XXX
                              done = 0b;
                              break;
                        case 011b:        // S2
```

```
                              printf("S2(o)\n");
                              ctrl = 0001010000000001000b;   //
00_0101_XXXX_XXX0_1XXX
                              done = 0b;
                              break;
                        case 010b:        // S3
                              printf("S3(o)\n");
                              ctrl = 0001110000000010000b;   //
00_0111_XXXX_XXX1_0XXX
                              done = 0b;
                              break;
                        case 110b:        // S4
                              printf("S4(o)\n");
                              ctrl = 0011101111000011010b;   //
00_1110_1111_0001_1X10
                              done = 0b;
                              break;
                        case 111b:        // S5
                              printf("S5(o)\n");
                              ctrl = 0011011101010110000b;   //
00_1101_1101_0101_1X00
                              done = 0b;
                              break;
                        case 101b:        // S6
                              printf("S6(o)\n");
                              ctrl = 0101001111001110 11b;   //
01_0100_1111_0011_1011
                              done = 0b;
                              break;
                        case 100b:        // S7
                              printf("S7(o)\n");
                              ctrl = 1000000001010000000b;   //
1X_XXXX_XXX1_01XX_XXXX
                              done = 1b;
                              break;
                  }
                  notify(_ctrl, _done);
            }
      }
};
```

B.15 regfile_2p.sc

```
/*********************************************************************
*  Title: regfile_2p.sc
*  Description: risging edge-triggered 32bit two-port register file
*                        with synchronous reset(high active)
*                        00: data reg
*                        01: ocount reg
*                        10: temp reg
*                        11: mask reg
*********************************************************************/
behavior reg_read(in bit[31:0] tmp_reg[4], in bit[1:0] raA, in bit[1:0]
raB,
      in bit[0:0] reA, in bit[0:0] reB, out bit[31:0] outA, out
bit[31:0] outB,
```

```
      in event _raA, in event _raB, in event _reA, in event _reB, out
event _outA,
      out event _outB)
{
      bit[31:0] mask = 0x0003;

      void main(void) {
            while (1) {
                  wait(_raA, _raB, _reA, _reB);
                  if (reA == 1b)
                        outA = tmp_reg[raA&mask];
                  if (reB == 1b)
                        outB = tmp_reg[raB&mask];
                  notify(_outA, _outB);
            }
      }
};

behavior reg_write (in event clk, in bit[0:0] rst, in bit[31:0] inp,
      in bit[31:0] tmp_reg[4], in bit[1:0] wa, in bit[0:0] we)
{
      bit[31:0] mask = 0x0003;

      void main(void) {
            while (1) {
                  wait(clk);
                  if (rst == 1b) {  // reset
                        tmp_reg[0] = 0x0000;
                        tmp_reg[1] = 0x0000;
                        tmp_reg[2] = 0x0000;
                        tmp_reg[3] = 0x0000;
                  }
                  else { // write data from input port( inp)
                        if (we == 1b)
                              tmp_reg[wa&mask] = inp;
                  }
            }
      }
};

behavior regfile_2p(in event clk, in bit[0:0] rst, in bit [31:0] inp,
      in bit[1:0] raA, in bit[1:0] raB, in bit[0:0] reA, in bit[0:0]
reB,
      in bit[1:0] wa, in bit[0:0] we, out bit[31:0] outA, out bit[31:0]
outB,
      in event _inp, in event _raA, in event _raB, in event _reA, in
event _reB,
      out event _outA, out event _outB)
{
      bit[31:0] tmp_reg[4];

      reg_read read_mode(tmp_reg, raA, raB, reA, reB, outA, outB, _raA,
_raB,
            _reA, _reB, _outA, _outB);
      reg_write write_mode(clk, rst, inp, tmp_reg, wa, we);

      void main(void) {
```

```
            par {
                  write_mode.main();
                  read_mode.main();
            }
      }
};
```

B.16 shift.sc

```
/*****************************************************************
*  Title: shift.sc
*  Description: Shifter Unit has right/left shift function
*   ctrl    function
*    0      right shift
*    1      left shift
*****************************************************************/

behavior shift(in bit[31:0] si, in bit[4:0] amount, in bit[0:0] ctrl,
out bit[31:0] so,
      in event _si, in event _amount, in event _ctrl, out event _so)
{
      void main(void) {
            while (1) {
                  wait(_si, _amount, _ctrl);
                  switch(ctrl) {
                        case 0b:    // right shift
                              so = si >> amount;
                              break;
                        case 1b:          // left shift
                              so = si << amount;
                              break;
                  }
                  notify(_so);
            }
      }
};
```

B.17 state_reg.sc

```
/*****************************************************************
*  Title: state_reg.sc
*  Description: state register(hamming encoded)
*****************************************************************/

import "chann";

behavior state_reg(in event clk, in bit[0:0] rst, in bit[2:0] next,
      out bit[2:0] cur, out event _cur)
{
      bit[2:0] state;

      void main(void) {
            while (1) {
                  wait(clk);

                  if (rst == 1b)
                        state = 000b;
                  else
```

```
                        state = next;

                cur = state;
                notify(_cur);
            }
        }
};
```

B.18 tb.sc

```
/*********************************************************************
*  Title: tb.sc
*  Description: testbench for one counter
*********************************************************************/

import "io";
import "clock_gen";
import "one_counter";

behavior Main
{
      bit[31:0] inport;
      bit[31:0] outport;
      bit[0:0] start;
      bit[0:0] done;
      bit[0:0] rst;
      event clk, _inport, _outport, _start, _done;
      cSyncInt isync;

      Input input(rst, inport, start, _inport, _start, isync);
      Output output(outport, done, _outport, _done, isync);
      clock_gen clkgen(clk);
      one_counter ones(clk, rst, inport, outport, start, done, _inport,
_outport,
            _start, _done);

      int main (void)
      {
            par {
                  clkgen.main();    // clock generator
                  input.main();           // get 32-bit vector from
stdin
                  ones.main();            // structural RTL for one
counter
                  output.main();    // write ocount to stdout
            }
            return 0;
      }
};
```