RTL Modeling in C++

Technical Report ICS-01-18
April 30, 2001

Shuqing Zhao

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
(949)824-8059

szhao@ics.uci.edu
http://www.ics.uci.edu/~szhao

Abstract

In this report we present the implementation of a C++ class library aimed at providing basic support for modeling RTL semantics in C++. We first review the inherent features of the RTL modeling in a formal FSMD approach. The corresponding C++ implementation for non-pipelined state-based (Moore) machine is given with the illustration of its usage for two examples.

Contents

List of Figures

RTL Modeling in C++

Shuqing Zhao

CECS, Department of ICS
University of California, Irvine
Irvine, CA 92697-3425, USA

Abstract

In this report we present the implementation of a C++ class library aimed at providing basic support for modeling RTL semantics in C++. We first review the inherent features of the RTL modeling in a formal FSMD approach. The corresponding C++ implementation for non-pipelined state-based (Moore) machine is given with the illustration of its usage for two examples.

1 Introduction

Hardware modeling in C/C++ has emerged as a promising alternative to HDL. There are many advantages associated with this approach. For example, it can significantly improve the simulation performance. Also it provides inherent support for Hardware/Software co-verification capability without extra PLI overhead as in HDL approach.

C/C++ support for hardware modeling can be delivered at various levels and in different flavors of implementation[1, 2, 3]. In this report we try to provide C++ support only at the register-transfer-level(RTL) which is well understood by engineers and employed by current generation of EDA tools. As proposed in [4], FSMD[6] is adopted in this report as a formal model for abstraction of the RTL description. The rationale is that this could make it easier for the synthesis and verification tools to analyze the input model as opposed to other implicit input descriptions.

In the report I first go over a list of styles used in FSMD modeling [4]. Then I present the corresponding implementation of a C++ class library for each specific style. An example of a one's counter is used throughout the five FSMD styles and an example of a parity checker is used for hierarchical/communicating FSMDs to illustrate the usage of the proposed RTL C++ library.

2 RTL modeling with FSMD

RTL modeling could be done in different flavors but they essentially all can be abstracted to FSMD. Some RTL synthesis tools accept implicit FSMD modeling for coding convenience purpose especially when the state number is just a couple. An explicit FSMD modeling flavor is used in this report in order to ease the compilation and analysis of the input description for the future synthesis and verification tool. For details about FSMD, the reader is referred to [6].

The modeling of FSMD can have different styles depending on how much detail of the implementation information it contains [4]. These FSMD modeling styles, are briefly reviewed in the several following paragraphs. They represent results of different stages during the RTL synthesis process including scheduling, storage unit allocation/binding, functional units allocation/binding, bus units allocation/binding, and control unit generation. All these styles will have its corresponding C++ support in our implementation. This means that after each synthesis step the refined FSMD model can still have a C++ image which is simulatable. The simulation result then can be compared to the result before this synthesis step to verify its correctness.

2.1 Five styles of FSMD modeling

The common features for all five styles include ports and a state. All five styles of FSMD have a state memory to keep track of its running status in each clock cycle. They also all have ports. Individual FSMD gets input from and outputs results through ports. Ports are also the means of communication by which FSMDs exchange information. The ports keep intact for all five styles of FSMD. It means no ports will be added or removed during all synthesis steps. The total states number remain unchanged among all five styles of FSMD as well. These two features guarantee the design's interfacing characteristic will remain the same if compared cycle by cycle after all synthesis steps. This is a very important property because it means the simulation result after each synthesis step can always be checked against the result before the specific step to verify the correctness of the refinement.

2.1.1 Unmapped FSMD(style 1)

Besides ports, in the unmapped FSMD there only exist internal variables which have not yet been implemented with hardware meaning: storages or wires. The variable assignments are executed in the written order sequentially. The assigned values of the variables can be used immediately after the assignments.

2.1.2 Storage mapped FSMD(style 2)

The only difference this style FSMD has from style 1 is that there are two types of variables in the storage-mapped FSMD. One is the **storage variable** which has a life time across the state boundary. The other is **wire variable** (normal C++ variable) which only temporarily exist in one state and is not visible in the other states.

2.1.3 Function mapped FSMD(style 3)

In this FSMD style, the arithmetic operators used before has been replaced with invocation of "functional units" that are used in the implementation data path. For example, a "+" and "-" operator could be substituted by a "ALU" with control input "1" representing the *add* operation and "0" representing the *minus* operation.

2.1.4 Connection mapped FSMD(style 4)

In this style, the wire variables could be grouped into "buses". The buses here are conceptual. In the final implementation they can be wires plus tri-state drivers or they can be realized by multiplexors.

2.1.5 Control mapped FSMD(style 5)

In this style, the FSMD is ready for the logic synthesis. It consists of two parts: a datapath containing the netlist of functional units and bus units; and a control unit which generates the data-path control signals in each clock cycles. The control unit is still described in a FSM in which the state actions are only control output assignments of binary values.

2.2 Communicating and Hierarchical FSMDs

Rarely any realistic design can be completed by only one FSMD. Usually two or more FSMDs need to be combined and connected via ports to form communicating FSMDs. Each individual FSMD is defined by a set of ports and a description in any of the styles mentioned above. These communicating FSMDs can be added another layer of container to form a hierarchical FSMD. Also FSMDs can be used in other FSMD as functional or storage unit in a hierarchical FSMD. The FSMDs used in this report are all Moore type which is state-based.

3 C++ Implementation for RTL Modeling

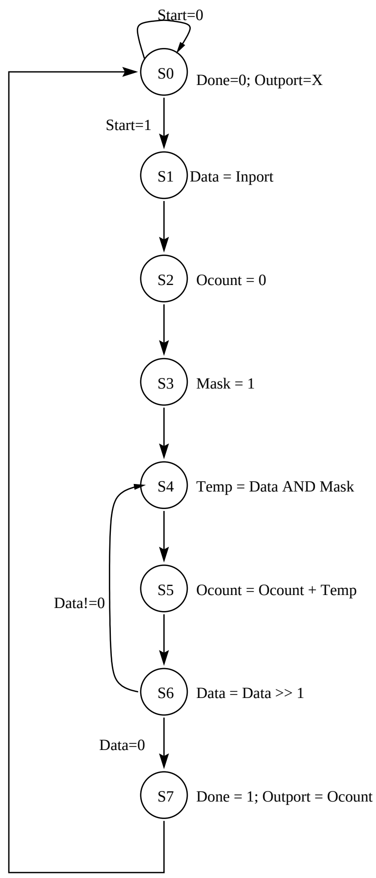

Figure 1: One's Counter FSMD.

An one's counter example from [5] is used to illustrate the implementation of C++ library support for RTL modeling. Its FSMD state transition diagram

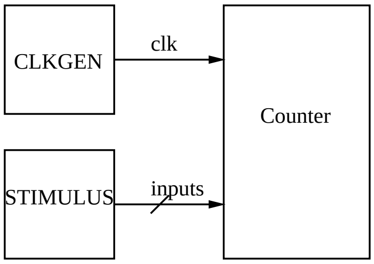

Figure 2: The Testbench of One's Counter.

is shown in Figure 1. To model this kind of description, some coding guidelines or conventions must be enforced. For example, the FSMD has a built-in state variable and if there is no naming convention it would be hard for the synthesis tool to find it out of so many other variables. Also for the state transition description, it can be described with various ways for example a nested if-statement can be used instead of the case statement we are going to suggest.

The top level structure of the testbench for this one's counter design is shown in Figure 2. In general similar testbench structure can be applied to other designs as well. A more generic structure of the testbench for a design under test is shown in Figure 3. Apparently, to model this structure some kind of concurrent execution support must be provided.

3.1 Basic support

3.1.1 Multi-thread C++ support

If the design is only limited to one clock domain or at least multiple clocks derived from one common clock source, the parallel execution of multiple hardware components can be implemented in C++ without need of any "real" multi-thread support. However, a more realistic premise of this report is that multiple unrelated clocks and asynchronous stimulus input are inevitable in a design environment supporting multiple application domains. Thus in this implementation the quick thread C++ library is used to support concurrency in C++ as in SpecC and SystemC. The multi-thread API is taken from SpecC simulation library but it is independent of SpecC language [7]. First it provides an interface to the multi-thread library to make it easy to use. For concurrent execution of threads, there is a simple API: *void Par(module* first, ...);*. Secondly it provides a timed discrete event scheduler to support method calls: *void waitfor(long long delay); void wait(event first); void notify(event first);*. These should be easily replaced with other C++ based simulation support API set.

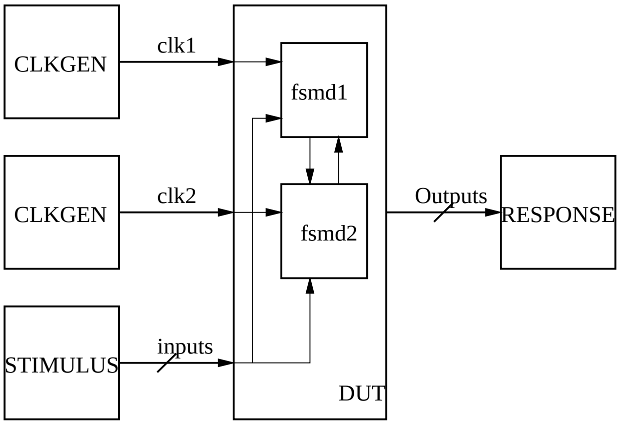

Figure 3: A Generic Structure of A Test Bench.

3.1.2 Module class

Module class is like *behavior* concept in SpecC language, *entity* in VHDL, or *module* in Verilog. It is the common container for FSMDs, datapath components, clock generators and stimulus generators. It can have any number of ports. See Section 3.1.3 for details about this type of class. Every Module object have a *main()* function which will be mapped to one C++ thread during execution. Like VHDL *process* concept, within the module *main()* function everything is executed sequentially but among multiple modules those *main()* functions are executed concurrently. It can be hierarchically composed which means that other modules can be instantiated inside it.

3.1.3 Port class

Instead of using C++ variables as parameters for passing information among concurrently running modules, a port class is needed for solving the non-deterministic exchange of values among them. As in VHDL and Verilog, a *delta cycle* concept is introduced in the port class. Or in another word, if you do not like the word *delta cycle*, the order of simultaneous read and write to the same port variable is fixed. That is, all the read to the port will always be executed before any of the write of the port although in the specification they seem to happen at the same time. Two methods: *read()* and *write()* are used for access to the value of the port variable. Another feature of the port class is that after the port value is being updated it generates an event to notify all the waiting parties on this event. This is useful for style-5 – control mapped

FSMD modeling. Ports can be further categorized to input, output or inout types by their I/O direction. In this report these distinctions has yet to be implemented.

The definition of the class is listed in Appendix A from line 41 to 85.

3.2 Style-1 FSMD class

Not surprisingly the class has a built-in variable called *state* which denote the current state of the FSMD along the time line. Usually this variable is being described with *enum* type like $\{S0,\ S1,\ ...\}$ defined by users in HDL. Due to difficulty of generalizing this type definition for all FSMD the state of the FSMD is being modeled with an integer type in this report. To make sure that the state number is always valid, this class accepts one template parameter: total state number. The state value range is between one and one less than this total state number. Run-time assertion is inserted in the class to check the next state value against this valid range to ensure the validity of state transitions. The state numbers have nothing to do with the final state encoding. They can still be think as symbolically equivalent to the enum type $\{S0,\ S1,\ ...\}$. The final state encoding can be determined by synthesis tool to be one-hot, one-cold or Gray etc. depending on the synthesis constraints like area, delay etc.

Style-1 FSMD class has two built-in ports: clock and reset. Clock is a special port. For Moore type FSMD all FSMD actions and state transitions are synchronized to the clock. The reactivity is implemented with a *while(true)*{...} infinite loop. The pseudo code for the class *main()* function is as follows:

```
1   void fsmd::main(void){
2     while(true)
3     {
4       wait(clk);
5       if(reset)
6       {
7           init();
8       }
9       else
10      {
11          assert(state > 0 && state < snum);
12          trans_act();
13      }
14
15      wait for delta;
16      update all the ports being written;
17    }
18  }
```

The FSMD class only requires users to specify two methods: *init()* for reset actions, *trans_act()* for normal state transitions and actions. On the clock edge the synchronous reset port is checked first (line 5) to determine whether reset actions should be taken (line 7) or a normal state transition/action (line 12) should be carried. Then the thread is suspended to wait for other threads reading its ports (line 15). After this it updates all its ports to new values (line 16).

There are two types of reset but in this report only synchronous reset is being implemented. However, support for asynchronous reset can be easily as follows.

```
void fsmd::main(void){
  while(true)
  {
    wait(clk || reset.e);
    if(reset)
    {
        init();
    }
    else
    {
        assert(state > 0 && state < snum);
        trans_act();
    }

    wait for delta;
    update all the ports being written;
  }
}
```

As for the actual description of state transitions and actions, the convention is to use C++ case statement as follows:

```
void Design::trans_act(void)
{
  switch(state)
    {
    case 0:
        state actions 0;
        state = x;
        break;
    case 1:

        ...

    case n:
        state actions n;
        state = y;
        break;
    }
}
```

All of the above said definitions and conventions are to ensure the rules of this style FSMD being always followed if designers use this class to build FSMD. The definition of the class is listed in Appendix A from line 123 to 180. An example using this class is listed in Appendix B.2.

3.3 Style-2 FSMD class

Style-2 FSMD inherits all features from style-1 FSMD. In addition for describing this storage-mapped FSMD, a new storage class named *stor* is introduced for those variables that have been mapped to storage elements such as registers or memory cells. This class is defined in Appendix A from line 94 to 121. Two methods: *read()* and *write()* are used for access to the value of the storage variable. The value written to the storage variables can only be used in the next clock cycle. This is being done through two steps in the FSMD2 class. First when the value is assigned to the storage variable, the value is stored in a temporary variable. Only until next clock cycle this new value will be updated for read access.

Compared to style-1 FSMD the class *main()* function is slightly adjusted as follows:

```
void fsmd2::main(void){
  while(true)
  {
    wait on clock edge;
    update all registers being written;
    if(reset)
    {
        init();
    }
    else
    {
        assert(state >= 0 && state < snum);
        trans_act();
    }

    wait for delta;
    update all the ports being written;
  }
}
```

The complete definition of the class is listed in Appendix A from line 181 to 235. An example using this class is listed in Appendix B.3.

3.4 "Functional units" for style-3 FSMD

For describing this style of FSMD, no extra classes are introduced. Style-2 FSMD class is still used as a template for modeling. Only difference is that the arithmetic operators has been replaced with "functional units" which are actually just a set of C++ predefined functions. For example, as listed in Appendix B.5, operators "+" and "&" can be replaced with a C++ function *int alu(int ctrl, int oprnd1, int oprnd2);* where *ctrl* is the control encoding for this functional unit to decide which operation to perform.

Standard functional units, such as ALU, multiplier and so on, can be provided in the library in order to ease the synthesis task later on. However, the generalization of the functional units is very difficult. For example a generic ALU has to include all functions such as $+, -, \&$ and $||$ etc. In addition it has to accommodate all possible bit-width. Realistically the library can be like the *Synopsys DesignWare* which contains only the units recognizable by the synthesis tool.

This style of FSMD can be used as a result format after the functional units allocation synthesis step. The simulation result should remain the same as before this synthesis step. This replacement of functional units does not affect the simulation also in the sense of event scheduling. It may slow down the simulation speed a little bit because of the function calling overhead.

The definition of the functional units used in the one's counter example is listed in Appendix A from line 405 to 410. The complete source code for the one's counter style-3 FSMD is listed in Appendix B.4.

3.5 "Bus units" for style-4 FSMD

In this style FSMD, the wire variables can be grouped into one common "bus unit" which is essentially just another variable with different name. The wire variables could be defined explicitly or implicitly inferred. For example, as in Appendix B.5, register Data and Ocount output are not defined as wire variables but they both are connected to the alu operand 1 input(see Figure 5). Indicated in the description there are two calls: $alu(0, Data, Mask)$ and $alu(1, Ocount, Temp)$ in style-3 FSMD. Indeed these wire variables have to be replaced with a bus variable, we call "Bus1" here. The same thing happens to the register Mask and Temp outputs which are replaced with a bus variable "Bus2". Correspondingly, the ALU call has been changed to *alu(ctrl, Bus1, Bus2);* where *ctrl* is the actual control encoding constant for the ALU to decide which operation to perform. These two definition of the "bus" variables used in the one's counter example is inserted line 30 of Appendix B.5.

This style of FSMD can be used as a result format after the interconnection allocation synthesis step. This replacement of bus variables does not affect the simulation at all. The simulation result should remain the same as before this synthesis step. Also the event scheduling should also remain unchanged.

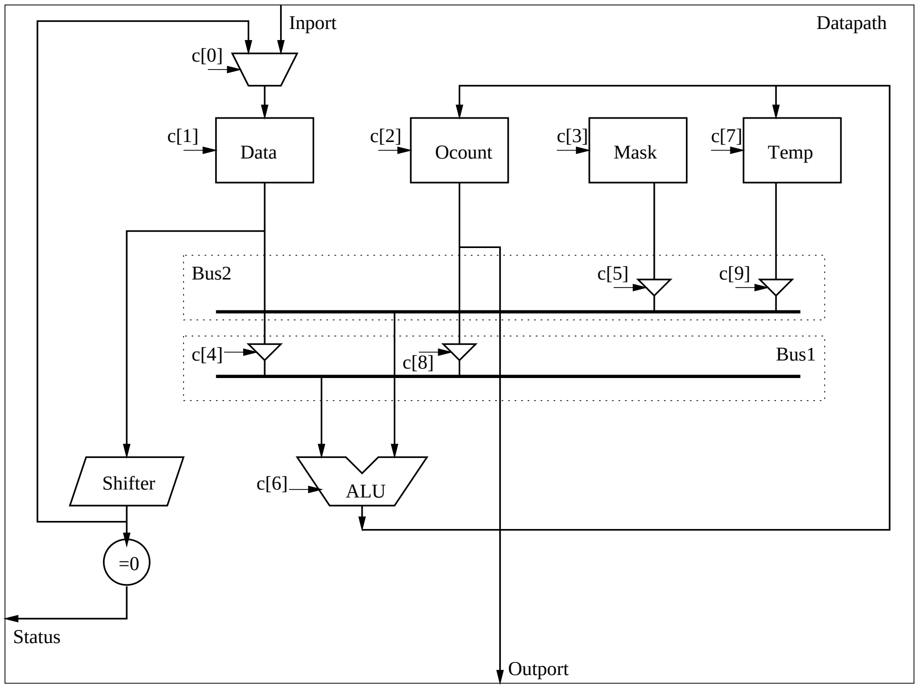

Figure 5: One's Counter Datapath.

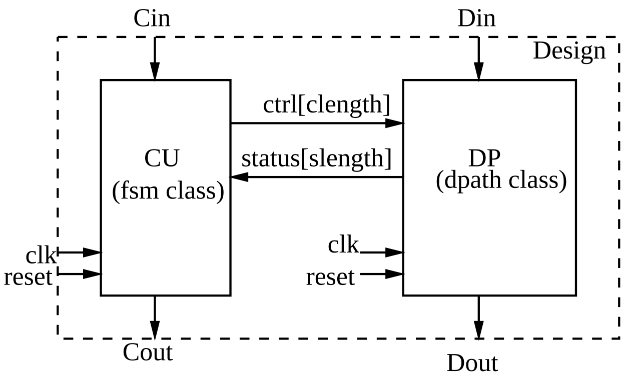

Figure 4: The Structure Template for Style-5 FSMD.

3.6 Supporting style-5 FSMD

In this style of FSMD, as shown in Figure 4 it contains two objects: a datapath object which is a netlist of datapath components objects and a controller object which is a FSM with a control variable output and a status variable input. In each state of this FSM, a control encoding vector is assigned to the control variable and the next state computation is being calculated based on the current state and maybe the status input. For simulation of this concurrent controller and all datapath components, including synchronous ones as registers and asynchronous ones as combinational functional units, an event-driven scheme is used as in traditional HDL. The controller will generate the control variables along with events to trigger the execution of corresponding datapath components which are sensitive to these events. As shown in Section 3.6.2, synchronous components such as *REGISTER* will be executed only once in each state. Asynchronous components such as *ALU* could be executed multiple times responding to the control input events or their other combinational input changing events.

3.6.1 FSM class

The FSM class for the controller inherits all the features of style-1 FSMD. It has state variable and two built-in ports: clock and reset. In addition it has two more fixed ports: control outputs and datapath status inputs. The class accepts two more template parameters: the control output bit-length and the status input bit-length. The class is defined in Appendix A from line 236 to line 248.

3.6.2 Functional unit class

The class *main()* function template is as follows:

```
void fu::main(void){
  while(true)
  {
```

```
    func();
    wait for delta;
    update all the ports being written;
  }
}
```

The functionality of the unit *func()* is left for the user to define. For synchronous components, the first line of the *func()* should be a wait statement on the clock event. For combinational components, the first line of the *func()* should be a wait statement of the sensitivity list of the unit. The list should include all the events of the input ports. Examples of ALU and REGISTER *func()* is shown below.

```
void ALU::func(void ) {
  wait(op.e || alu_in1.e || alu_in2.e);
    if(op==1)
        assign(alu_out, alu_in1 + alu_in2);
    else
        assign(alu_out, alu_in1 & alu_in2);
}

void REGISTER::func(void ) {
        wait(clk);
        if(reset==1)    //sync. reset
                assign(Dout, 0);
        else if(ld==1)
                assign(Dout, Din);
}
```

3.6.3 "Bus unit"

In traditional HDL, bus is modeled by wires plus explicit bus driver components. The bus wire is written simultaneously by multiple drivers and the value is obtained by a resolution function. The bus conflict is solved by control of the individual bus drivers so that there is only one driver which has the binary value while others exhibiting high-impedance value. In this approach designers have to make sure there is only one driver of the bus at any moment. We model bus by integrating the bus drivers together with the wires as shown in Figure 5. The bus unit can then be used as a component to connect to its driving component and receiving component. One of the advantage is that the unit can check the bus conflict at the run time to ensure there is no bus conflict. The definition of the bus units used in the one's counter datapath is listed in Appendix B.6 from line 90 to 108.

The source code of the one's counter style-5 FSMD example is listed in Appendix B.6. For better understanding of the structure of the design, a schematic of the datapath is shown in Figure 5.

3.7 Supporting Communicating and Hierarchical FSMDs

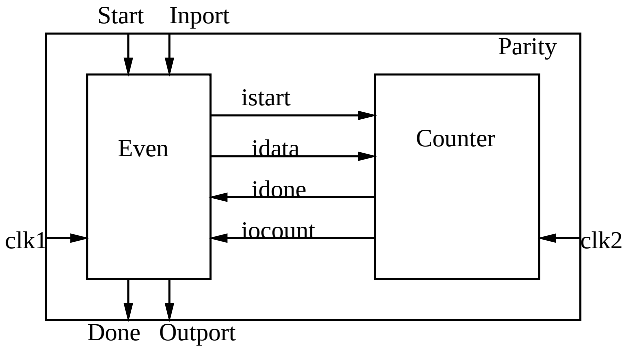

Figure 6: Parity Checker Communicating/Hierarchical FSMD.

No more classes were introduced for communicating FSMDs and hierarchical FSMDs. Communicating FSMDs are simply connected via their ports assuming a proper protocol(e.g. handshake) exists to avoid miscommunication. Hierarchical FSMD is constructed by instantiating FSMD objects. We illustrate the composition of a communicating/hierarchical FSMD with above mentioned C++ classes by using an example called parity checker. In Figure 6 we show the one's counter is used as a component inside the parity checker FSMD. The one's counter is running concurrently with another FSMD: even number checker. They communicate by the ports connecting them. The FSMD *Even* takes the role of the master whose Start is the Start of the composite and whose Done is the Done of the composite. Of course there should be a protocol to ensure the correct synchronization. The actual timing is shown in Figure 7. The protocol is part of the design FSMD in this RT level. There is no clear separation between the computation and the communication especially when the FSMD is fully optimized by the scheduling algorithm during the synthesis process.

The source code of the parity checker example illustrating communicating and hierarchical FSMDs is listed in Appendix C.

3.8 Miscellaneous support

3.8.1 Clock generator class

Clock is defined as an event in this implementation. A clock generator class is needed to model the clock generator which periodically generates the clock event

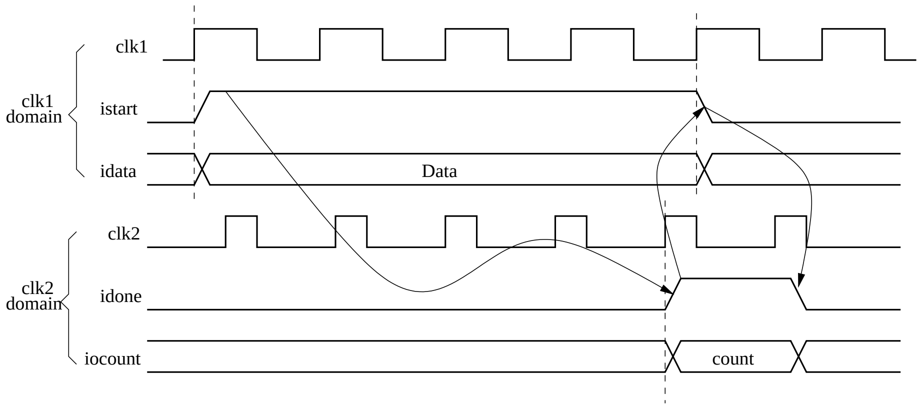

Figure 7: Parity Checker Communication Protocol.

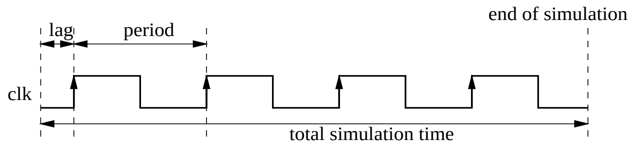

Figure 8: Clock waveform generated by the clock generator class.

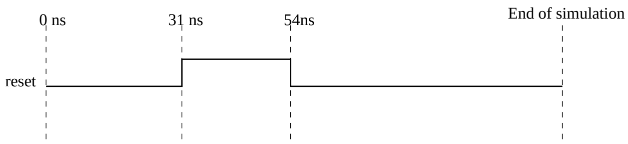

Figure 9: Reset wire waveform for testing the one's counter.

to trigger the executions of all the hardware units synchronized to it. As shown in Figure 8 this class can be configured with two parameters: period and start time, where $period = 1/frequency$ and start time denotes when the clock start ticking. Duty cycle is not included currently since only one edge of the clock is used for any FSMD in this report. An instance of the class $CLKGEN\ ckg(clk, 10, 0)$ will generate events of clock *clk* at a period of 10 time units starting at 0 time unit. This parameterized clock class would make it easier for synthesis tool to recognize clock domains and generate corresponding logic synthesis scripts.

This class will accept a total simulation run time parameter *SIMTIME* from a configuration file called ".sim" and depend on that to terminate the simulation. For example, if the file has a line: *simulation_time = 400*, the simulation will simulate for 400 time units and stop. The benefit of this implementation is that it does not need re-compilation for simulation iterations differing only in total run time.

3.8.2 Stimulus generator class

It is very common that stimuli need to be provided to test the design. A stimulus generation class is supplied to generate the waveform i.e. values varying in time. It inherits the *module* class and contains one *waveform* class object called *stimulus*. One such class object can generate the waveform for one wire and inside the module *main()* function C++ stream syntax is used for compactness purpose. The format is: $stimulus << value0 << delay0 << value1 << delay1 ...;$. It means the initial value of the stimulus is $value0$, after $delay0$ it changes to $value1$, and so on and so forth. For example, to generate the waveform shown in Figure 9 the statement should be: $stimulus << 0 << 31 << 1 << 23 << 0;$. The initial value of the *reset* signal is 0, after 31 time units it becomes 1, and after another 23 time units it changes back to 0 again and keep this value until the end of the simulation.

3.8.3 Function *main()*

The top level of the hierarchy C++ function *main()* has already been defined in the library to hide some simulation related functions such as checking the existence of the file ".sim" and starting the scheduler etc. The definition is listed in Appendix A from line 377 to 402. Users are required to write their top level function as *testbench()*.

3.9 Class relationship

In summary, a UML-like class relationships diagram is shown in Figure 10 to illustrate the class relationships like inheritance, use etc.

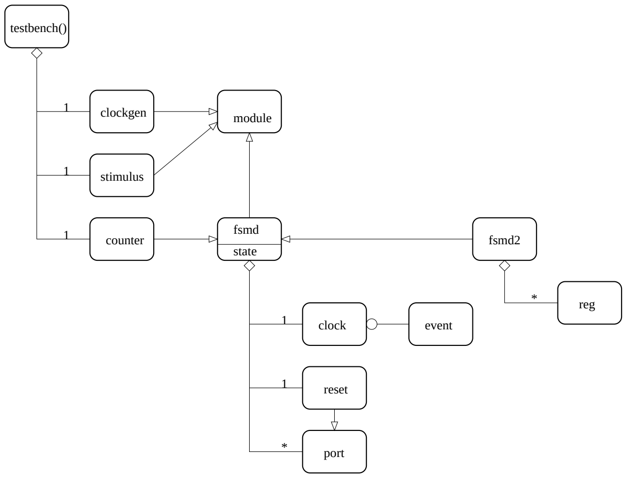

Figure 10: One's Counter class relationship.

4 Summary

In this report we have present the implementation of a C++ class library aimed at providing basic support for modeling RTL semantics in C++. The above explained C++ classes and other definitions provide a foundation for RTL modeling with FSMD approach. We have tested its application with two examples. The complete implementation and the examples source code are listed in the following appendices.

5 Acknowledgment

I would like to thank Professor Daniel Gajski for his leadership and generous contribution to this work. I also appreciate the help from Andreas Gerstalauer and Rainer Doemer with my C++ language related questions. I am also grateful to all the students in the UCI ICS259 class – Haobo Yu, Qiang Xie, Pei Zhang, Dongwan Sin, Fred Ducet, and Mukesh Rajan – for their contribution of their time testing the code and their engagement in the discussion leading to this report.

References

[1] http://www.specc.org

[2] http://www.systemc.org

[3] http://www.cynapps.com

[4] http://wwww.accellera.org, *RTL Semantics Draft Specification*, Februray 2001.

[5] D. Gajski, *Principles of Digital Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1997

[6] D. Gajski, N. Dutt, C.H. Wu, Y.L. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers, Boston, Massachusetts, 1991

[7] D. Gajski, J. Zhu, R. Döemer, A. Gerstlauer, S. Zhao, *SpecC: Specification language and Methodology*, Kluwer Academic Publisher, Norwell, Massachussetes, 2000

A C++ source listing for RTL modeling support

```
#ifndef RTLC
#define RTLC

#include <cola/lang/Xevent.h>
#include <cola/specc/XModule.h>

class _rtlc
{
        public:
                typedef Xevent          event;
                typedef XModule         module;

                static void start( void );
                static void end( void );

                static void Par( module* first , ... );

                static void waitfor( long long delay );
                static void wait( event first );
                static void notify( event first );

};

#define module _rtlc::module
#define event _rtlc::event
#define waitfor _rtlc::waitfor
#define wait _rtlc::wait
#define notify _rtlc::notify
#define Par _rtlc::Par

//------------------above: specc simulator support-----------------//

//------------------below: added RTL C++ support -----------------//
#include <assert.h>
#include <queue>
#include <fstream.h>

int SIMTIME; //global variable for total simulation time
typedef event clock;             //clock is only an event?

class base_port
{
        public:
                event e;

                base_port(void){}
                virtual ~base_port(void){}
                virtual void update(void) = 0;
};

template<class T> class port: public base_port
{
        protected:
                T cur_val, next_val;

        public:

                port(void) {cur_val=next_val=0;}
                port(T i) {next_val=i;}

                T read(void)
                {
                        return cur_val;
                }
```

```
                operator const T&()
                {
                        return cur_val;
                }

                void write(T val)
                {
                        next_val = val;
                }

                void update()
                {
                        if(cur_val != next_val) {
                        cur_val = next_val;
                        notify(e);
                        }
                }

};

#define net port                                        //for connecting ports

class base_reg
{
        public:
                base_reg(void){}
                virtual ~base_reg(void){}
                virtual void update(void) = 0;
};

template<class T> class reg : public base_reg
{
        protected:
                T cur_val, next_val;

        public:
                reg(void) {cur_val=next_val=0;}

                T read(void)
                {
                        return cur_val;
                }

                operator const T&()
                {
                        return cur_val;
                }

                void write(T val)
                {
                        next_val = val;
                }

                void update()
                {
                        cur_val = next_val;
                }
};

template<int snum> class fsmd : public module
{

        protected:
                queue<base_port*> pq;

```

```
		int state;
		clock (&clk);
		port<bool> (&reset);

		fsmd(clock (&clk_local), port<bool> (&reset_local))
			: clk(clk_local), reset(reset_local)
		{
			state=0;
		}

	public:
		virtual void init(void) = 0;
		virtual void trans_act(void) = 0;

		template<class T> void assign(port<T>& r, int val)
		{
			r.write(val);
			pq.push(&r);
		}

		template<class T> void assign(port<T>& r1, port<T>& r2)
		{
			r1.write(r2.read());
			pq.push(&r1);
		}


		void main(void)
		{
			while(true)
			{
				wait(clk);
				if(reset.read() == 1)
				{
					init();
				}
				else
				{
					assert(state >= 0 && state < snum);
					trans_act();
				}

				waitfor(0);
				while(!pq.empty())
				{
					pq.front()->update();
					pq.pop();
				}
			}
		}
};

template<int snum> class fsmd2 : public fsmd<snum>
{
	protected: queue<base_reg*> q;

	public:
		fsmd2(clock (&clk_local), port<bool> (&reset_local)):
		fsmd<snum>(clk_local, reset_local)
		{
			state=0;
		}

		template<class T> void write(reg<T>& r, int val)
		{
			r.write(val);
```

```
                q.push(&r);
        }

        template<class T> void write(reg<T>& r1, reg<T>& r2)
        {
                r1.write(r2.read()); //using r: no problem?
                q.push(&r1);
        }

        void main(void)
        {
                while(true)
                {
                        wait(clk);
                        while(!q.empty()) //what about async reset?
                        {
                                q.front()->update();
                                q.pop();
                        }

                        if(reset.read() == 1)
                        {
                                init();
                        }
                        else
                        {
                                assert(state >= 0 && state < snum);
                                trans_act();
                        }

                        waitfor(0);
                        while(!pq.empty())
                        {
                                pq.front()->update();
                                pq.pop();
                        }
                }
        }

};

template<int snum, int clength, int slength> class fsm: public fsmd<snum>
{
    protected:
        port<bool> (&ctrl)[clength]; //ctrl bits
        port<bool> (&status)[slength]; //status bits

        fsm(clock (&clk_local), port<bool> (&reset_local),
                port<bool> (&ctrl0)[clength], port<bool> (&status0)[slength])
                : fsmd<snum>(clk_local, reset_local), ctrl(ctrl0), status(status0)
        {
        }

};

template<int clength, int slength> class dpath : public module
{
protected:

    clock (&clk); port<bool> (&reset);
    port<bool> (&ctrl)[clength];
    port<bool> (&status)[slength];

public:
        dpath(clock (&clk_local), port<bool> (&reset_local),
                port<bool> (&ctrl0)[clength], port<bool> (&status0)[slength])
```

```
		: clk(clk_local), reset(reset_local),
		ctrl(ctrl0), status(status0)
		{}
};

//functional unit for style 5
class fu : public module
{

	protected:
		queue<base_port*> pq;

		fu(void) {};

	public:
		virtual void func(void) = 0;

		template<class T> void assign(port<T>& r, int val)
		{
			r.write(val);
			pq.push(&r);
		}

		template<class T> void assign(port<T>& r1, port<T>& r2)
		{
			r1.write(r2.read());
			pq.push(&r1);
		}


		void main(void)
		{
			while(true)
			{
				func();

				waitfor(0);
				while(!pq.empty())
				{
					pq.front()->update();
					pq.pop();
				}
			}
		}
};

class CLKGEN : public module
{
	private:

		event (&clk);
		int period;
		int lag;
	public:

		CLKGEN(event (&clk), int period, int lag)
			:clk(clk), period(period), lag(lag)
			{};

		void main(void);
};

void CLKGEN::main(void)
{
	int j = 0;
	int cycles = (SIMTIME-lag)/period;
```

```
        int resid = (SIMTIME-lag)%period;

        waitfor(lag);
        do
        {
                notify(clk);
                waitfor(period);
        }while(j++ < cycles);
        waitfor(resid);
        exit(1);
}

template<class T> class wav
{
        private:
                port<T>& sig;
                bool even;
        public:
                wav<T> (port<T> &sig)
                : sig(sig)
                {
                even = false;
                };

                template<class C> wav<T>& wav<T>::operator<<(const C v)
                {
                if (!even)
                        {sig.write(v);
                        waitfor(0);
                        sig.update(); even = true;}
                else {waitfor(v); even = false;}
                return *this;
                }

};

template<class T> class STI : public module
{
        protected:
                wav<T> stimulus;

        public:
                STI(port<T>& sig)
                : stimulus(sig)
                {};

};

extern "C" int testbench(int argc, char* argv[]);

int main(int argc, char* argv[] )
{
        int status;
        char envar[20], equ;
        ifstream inf(".sim");

        if (!inf)
        {
        cerr<<"File .sim does not exist.\n";
        exit(1);
        }

        inf >> envar >> equ >> SIMTIME;
        if((strcmp(envar, "simulation_time")) || (equ != '='))
        {
        cout<<"The env variable setup is: " << envar <<' ' << equ << endl;
```

```
        cerr<<"File .sim format is wrong.\n";
        exit(1);
        }

        _rtlc::start();
        status = testbench(argc, argv);
        _rtlc::end();

        return status;
}

//functional unit for style 3,4:
int alu(int ctrl, int op1, int op2)
{
        if(ctrl==1) return op1+op2;
        else return op1&op2;
}

extern "C" long long int now(void);
extern "C" int printf(const char *, ...);

#endif
```

B C++ source listing for the one's counter example

B.1 Testbench and stimulus generation

```
/////////////////////////////////////////////////////////////////////
// Design:   one's counter
// Author:   Shuqing Zhao
// Purpose: RTL C++ semantics experiment
//           - Testbench
// File:     tb.cc
// Time:     Thu Jan 25 2001
/////////////////////////////////////////////////////////////////////

#include "rtlc.h"
#include "sti.h"
#include "count.cc"

int testbench(int argc, char* argv[])
{

        const int clk_period = 10;

        // instantiation of wires
        clock clk;
        net<bool> reset;
        net<bool> start0, done0;
        net<int> indata0, outdata0;

        // instantiation and connection of components
        CLKGEN ckg(clk, clk_period, 0);
        STIGEN sti0(reset, start0, indata0);
        Count count0(clk, reset, start0, indata0, done0, outdata0);

        // parallel execution
        Par(&ckg, &sti0, &count0, 0);

        return(0);
}

/////////////////////////////////////////////////////////////////////
// End of file tb.cc
/////////////////////////////////////////////////////////////////////
```

```
/////////////////////////////////////////////////////////////////////
// Design:   one's counter
// Author:   Shuqing Zhao
// Purpose: RTL C++ semantics experiment
//           - Stimulus generation
// File:     sti.h
// Time:     Thu Jan 25 2001
/////////////////////////////////////////////////////////////////////

#include "rtlc.h"

class STI_RESET : public STI<bool>
{
        public:
                STI_RESET(port<bool>& reset)
                : STI<bool>(reset)
                {};

                void main(void)
                {
                stimulus << 0 << 31 << 1 << 23 << 0;
                };
```

```
};

class STI_START : public STI<bool>
{
        public:
                STI_START(port<bool>& start)
                : STI<bool>(start)
                {};

                void main(void)
                {
                stimulus << 0 << 5 << 1;
                };

};

class STI_DATA : public STI<int>
{
        public:
                STI_DATA(port<int>& data)
                : STI<int>(data)
                {};

                void main(void)
                {
                stimulus << 0 << 5 << 7;
                };

};

class STIGEN : public module
{
        private:

                port<bool> (&reset);
                port<bool> (&start);
                port<int> (&data);

                STI_RESET sti_reset;
                STI_START sti_start;
                STI_DATA  sti_data;
        public:

                STIGEN(port<bool> (&reset), port<bool> (&start), port<int> (&data))
                        : reset(reset), start(start), data(data),
                          sti_reset(reset), sti_start(start), sti_data(data)
                        {};
                void main(void );
};

void STIGEN::main(void)
{
        Par(&sti_reset, &sti_start, &sti_data, 0);
}

//////////////////////////////////////////////////////////////////////
// End of file sti.h
//////////////////////////////////////////////////////////////////////
```

B.2 Style 1 FSMD

```
/////////////////////////////////////////////////////////////////////
// Design: one's counter
// Author: Shuqing Zhao
// File:   count.h
// Time:   Tue Jan 16 18:09:50 2001
/////////////////////////////////////////////////////////////////////

#include "rtlc.h"

class Count : public fsmd<8>
{
        private: //ports

                port<bool> (&Start);
                port<int> (&Inport);
                port<bool> (&Done);
                port<int> (&Outport);

        public:
                Count(clock (&clk), port<bool> (&reset),
                        port<bool> (&Start), port<int> (&Inport),
                        port<bool> (&Done), port<int> (&Outport))
                        : fsmd<8> (clk, reset),
                        Start(Start), Inport(Inport), Done(Done), Outport(Outport)
                        {};

                void init(void);
                void trans_act(void);

        private: // internal variables

                int Data, Mask, Ocount, Temp;

};

/////////////////////////////////////////////////////////////////////
// End of file count.h
/////////////////////////////////////////////////////////////////////
```

```
/////////////////////////////////////////////////////////////////////
// Design:  one's counter
// Author:  Shuqing Zhao
// Purpose: RTL C++ semantics experiment
//          - Style 1 FSMD
// File:    count.cc
// Time:    Thu Jan 25 2001
/////////////////////////////////////////////////////////////////////

#include "rtlc.h"
#include "count.h"

void Count::init(void)
{
        state = 0;
        assign(Done, 0);
        assign(Outport, 0);
        Data = 0;
        printf("%3lld ns: (Count) Oops, reset!\n", now());
}


void Count::trans_act(void)
{
        switch(state)
```

```
	{
		case 0:
			assign(Done, 0); assign(Outport, 0);
			if (Start)
				state = 1;
			break;
		case 1:
			Data = Inport; //style 1
			//or explicitly: Data = Inport.read();
			state = 2;
			break;
		case 2:
			Ocount = 0;
			state = 3;
			break;
		case 3:
			Mask = 1;
			state = 4;
			break;
		case 4:
			Temp = Data & Mask; // style 1
			state = 5;
			break;
		case 5:
			Ocount = Ocount + Temp;
			state = 6;
			break;
		case 6:
			Data=Data>>1;
			if (Data == 0)
				state = 7;
			else
				state = 4;
			//Data = Data >> 1;
			break;
		case 7:
			assign(Done, 1);
			assign(Outport, Ocount);
			if (Start)
				state = 7;
			else
				state = 0;
			break;
	}

	printf("%3lld ns: (Count) reset/%d Start/%d Done/%d Inport/%d Outport/%d [state/%d Data/%2d].\n",
		now(), reset.read(), Start.read(), Done.read(), Inport.read(), Outport.read(), state, Data);
}

////////////////////////////////////////////////////////////////////////
// End of file count.cc
////////////////////////////////////////////////////////////////////////
```

B.3 Style 2 FSMD

```
/////////////////////////////////////////////////////////////////////
// Design: one's counter
// Author: Shuqing Zhao
// File:   count.h
// Time:   Tue Jan 16 18:09:50 2001
/////////////////////////////////////////////////////////////////////

#include "rtlc.h"

class Count : public fsmd2<8>
{
        private:                                             //ports

                port<bool> (&Start); port<int> (&Inport); port<bool> (&Done); port<int> (&Outport);

        public:
                Count(clock (&clk), port<bool> (&reset),
                        port<bool> (&Start), port<int> (&Inport),
                        port<bool> (&Done), port<int> (&Outport))
                        : fsmd2<8> (clk, reset),
                        Start(Start), Inport(Inport), Done(Done), Outport(Outport)
                        {};

                void init(void);
                void trans_act(void);

        private:

                reg<int> Data, Mask, Ocount, Temp;

};

/////////////////////////////////////////////////////////////////////
// End of file count.h
/////////////////////////////////////////////////////////////////////
```

```
/////////////////////////////////////////////////////////////////////
// Design:  one's counter
// Author:  Shuqing Zhao
// Purpose: RTL C++ semantics experiment
//          - Style 2 FSMD
// File:    count.cc
// Time:    Thu Jan 25 2001
/////////////////////////////////////////////////////////////////////

#include "rtlc.h"
#include "count.h"

void Count::init(void)
{
        state = 0;
        assign(Done, 0);
        assign(Outport, 0);
        write(Data, 0);
        printf("%3lld ns: (Count) Oops, reset!\n", now());
}


void Count::trans_act(void)
{
        switch(state)
        {
                case 0:
                        assign(Done, 0); assign(Outport, 0);
```

```
                if (Start)
                    state = 1;
                break;
            case 1:
                write(Data, Inport);
                state = 2;
                break;
            case 2:
                write(Ocount, 0);
                state = 3;
                break;
            case 3:
                write(Mask, 1);
                state = 4;
                break;
            case 4:
                write(Temp, Data & Mask); // style 2
                state = 5;
                break;
            case 5:
                write(Ocount, Ocount + Temp); //style 2
                state = 6;
                break;
            case 6:
                // Data=Data>>1;
                write(Data, Data >> 1);
                if (Data == 0)
                    state = 7;
                else
                    state = 4;
                //Data = Data >> 1;
                break;
            case 7:
                assign(Done, 1); assign(Outport, Ocount);
                if (Start)
                    state = 7;
                else
                    state = 0;
                break;
        }

        printf("%3lld ns: (Count) reset/%d Start/%d Done/%d Inport/%d Outport/%d [state/%d Data/%2d].\n",
            now(), reset.read(), Start.read(), Done.read(),
            Inport.read(), Outport.read(), state, Data.read());
}

//////////////////////////////////////////////////////////////////////////
// End of file count.cc
//////////////////////////////////////////////////////////////////////////
```

B.4 Style 3 FSMD

```
////////////////////////////////////////////////////////////////////
// Design: one's counter
// Author: Shuqing Zhao
// File :   count.h
// Time:    Tue Jan 16 18:09:50 2001
////////////////////////////////////////////////////////////////////

#include "rtlc.h"

class Count : public fsmd2<8>
{
        private:                                        //ports

                port<bool> (&Start); port<int> (&Inport); port<bool> (&Done); port<int> (&Outport);

        public:
                Count(clock (&clk), port<bool> (&reset),
                        port<bool> (&Start), port<int> (&Inport),
                        port<bool> (&Done), port<int> (&Outport))
                        : fsmd2<8> (clk, reset),
                        Start(Start), Inport(Inport), Done(Done), Outport(Outport)
                        {};

                void init(void);
                void trans_act(void);

        private:

                reg<int> Data, Mask, Ocount, Temp;

};

////////////////////////////////////////////////////////////////////
// End of file count.h
////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////
// Design:   one's counter
// Author:   Shuqing Zhao
// Purpose: RTL C++ semantics experiment
//          - Style 3 FSMD
// File:     count.cc
// Time:     Thu Jan 25 2001
////////////////////////////////////////////////////////////////////

#include "rtlc.h"
#include "count.h"

void Count::init(void)
{
        state = 0;
        assign(Done, 0);
        assign(Outport, 0);
        write(Data, 0);
        printf("%3lld ns: (Count) Oops, reset!\n", now());
}


void Count::trans_act(void)
{
        switch(state)
        {
                case 0:
                        assign(Done, 0); assign(Outport, 0);
```

```
                        if (Start)
                                state = 1;
                        break;
                case 1:
                        write(Data, Inport);
                        state = 2;
                        break;
                case 2:
                        write(Ocount, 0);
                        state = 3;
                        break;
                case 3:
                        write(Mask, 1);
                        state = 4;
                        break;
                case 4:
                        write(Temp, alu(0, Data, Mask)); // style 3
                        state = 5;
                        break;
                case 5:
                        write(Ocount, alu(1, Ocount, Temp)); //style 3
                        state = 6;
                        break;
                case 6:
                        // Data=Data>>1;
                        write(Data, Data >> 1);
                        if (Data == 0)
                                state = 7;
                        else
                                state = 4;
                        //Data = Data >> 1;
                        break;
                case 7:
                        assign(Done, 1); assign(Outport, Ocount);
                        if (Start)
                                state = 7;
                        else
                                state = 0;
                        break;
        }

        printf("%3lld ns: (Count) reset/%d Start/%d Done/%d Inport/%d Outport/%d [state/%d Data/%2d].\n",
                now(), reset.read(), Start.read(), Done.read(),
                Inport.read(), Outport.read(), state, Data.read());
}

////////////////////////////////////////////////////////////////////////
// End of file count.cc
////////////////////////////////////////////////////////////////////////
```

B.5 Style 4 FSMD

```
//////////////////////////////////////////////////////////////////////
// Design: one's counter
// Author: Shuqing Zhao
// File :   count.h
// Time:    Tue Jan 16 18:09:50 2001
//////////////////////////////////////////////////////////////////////

#include "rtlc.h"

class Count : public fsmd2<8>
{
        private:                                                  //ports

                port<bool> (&Start); port<int> (&Inport); port<bool> (&Done); port<int> (&Outport);

        public:
                Count(clock (&clk), port<bool> (&reset),
                        port<bool> (&Start), port<int> (&Inport),
                        port<bool> (&Done), port<int> (&Outport))
                        : fsmd2<8> (clk, reset),
                        Start(Start), Inport(Inport), Done(Done), Outport(Outport)
                        {};

                void init(void);
                void trans_act(void);

        private:

                reg<int> Data, Mask, Ocount, Temp;
                int Bus1, Bus2; // introduced in style 4

};

//////////////////////////////////////////////////////////////////////
// End of file count.h
//////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////
// Design:  one's counter
// Author:  Shuqing Zhao
// Purpose: RTL C++ semantics experiment
//          - Style 4 FSMD
// File :   count.cc
// Time:    Thu Jan 25 2001
//////////////////////////////////////////////////////////////////////

#include "rtlc.h"
#include "count.h"

void Count::init(void)
{
        state = 0;
        assign(Done, 0);
        assign(Outport, 0);
        write(Data, 0);
        printf("%3lld ns: (Count) Oops, reset!\n", now());
}


void Count::trans_act(void)
{
        switch(state)
        {
                case 0:
```

```
                        assign(Done, 0); assign(Outport, 0);
                        if (Start)
                                state = 1;
                        break;
                case 1:
                        write(Data, Inport);
                        state = 2;
                        break;
                case 2:
                        write(Ocount, 0);
                        state = 3;
                        break;
                case 3:
                        write(Mask, 1);
                        state = 4;
                        break;
                case 4:
                        Bus1 = Data;
                        Bus2 = Mask;
                        write(Temp, alu(0, Bus1, Bus2));
                        state = 5;
                        break;
                case 5:
                        Bus1 = Ocount;
                        Bus2 = Temp;
                        write(Ocount, alu(1, Bus1, Bus2));
                        state = 6;
                        break;
                case 6:
                        // Data=Data>>1;
                        write(Data, Data >> 1);
                        if (Data == 0)
                                state = 7;
                        else
                                state = 4;
                        //Data = Data >> 1;
                        break;
                case 7:
                        assign(Done, 1); assign(Outport, Ocount);
                        if (Start)
                                state = 7;
                        else
                                state = 0;
                        break;
        }

        printf("%3lld ns: (Count) reset/%d Start/%d Done/%d Inport/%d Outport/%d [state/%d Data/%2d].\n",
                now(), reset.read(), Start.read(), Done.read(),
                Inport.read(), Outport.read(), state, Data.read());
}

//////////////////////////////////////////////////////////////////////
// End of file count.cc
//////////////////////////////////////////////////////////////////////
```

B.6 Style 5 FSMD

```
//////////////////////////////////////////////////////////////////////
// Design: one's counter
// Author: Shuqing Zhao
// File :    count.h
// Time:     Tue Jan 16 18:09:50 2001
//////////////////////////////////////////////////////////////////////

#include "rtlc.h"

class CU : public fsm<8, 10, 1>
{
private:
        port<bool> (&Start); port<bool> (&Done);

public:
        CU(clock (&clk_local), port<bool> (&reset_local),
                port<bool> (&ctrl0)[10], port<bool> (&status0)[1],
                port<bool> &Start, port<bool> &Done)
                : fsm<8, 10, 1>(clk_local, reset_local, ctrl0, status0), Start(Start), Done(Done)
                {}

        void init(void);
        void trans_act(void);
};

class COMP : public fu
{
private:

        port<int> (&comp_in);
        port<bool> (&comp_out);

public:
        COMP(port<int> (&comp_in), port<bool> (&comp_out))
        : comp_in(comp_in), comp_out(comp_out)
        {}

        void func(void);
};

class SHIFTER : public fu
{
private:

        port<int> (&shift_in);
        port<int> (&shift_out);

public:
        SHIFTER(port<int> (&shift_in), port<int> (&shift_out))
        : shift_in(shift_in), shift_out(shift_out)
        {}

        void func(void);
};

class ALU : public fu
{
private:

        port<bool> (&op);
        port<int> (&alu_in1); port<int> (&alu_in2);
        port<int> (&alu_out);

public:
```

```
        ALU(port<bool> (&op), port<int> (&alu_in1), port<int> (&alu_in2), port<int> (&alu_out))
        : op(op), alu_in1(alu_in1), alu_in2(alu_in2), alu_out(alu_out)
        {}

        void func(void);
};

class MUX : public fu
{
private:

        port<bool> (&en1);
        port<int> (&driver1); port<int> (&driver2);
        port<int> (&bus_val);

public:
        MUX(port<bool> (&en1),
                port<int> (&driver1), port<int> (&driver2),
                port<int> (&bus_val))
        : en1(en1), driver1(driver1), driver2(driver2), bus_val(bus_val)
        {}

        void func(void);
};

class BUS : public fu
{
private:

        port<bool> (&en1);
        port<bool> (&en2);
        port<int> (&driver1); port<int> (&driver2);
        port<int> (&bus_val);

public:
        BUS(port<bool> (&en1), port<bool> (&en2),
                port<int> (&driver1), port<int> (&driver2),
                port<int> (&bus_val))
        : en1(en1), en2(en2), driver1(driver1), driver2(driver2), bus_val(bus_val)
        {}

        void func(void);
};

class REGISTER : public fu
{
private:

        clock (&clk); port<bool> (&reset);
        port<bool> (&ld);
        port<int> (&Din); port<int> (&Dout);

public:
        REGISTER(clock (&clk), port<bool> (&reset), port<bool> (&ld), port<int> (&Din), port<int> (&Dout))
        : clk(clk), reset(reset), ld(ld), Din(Din), Dout(Dout)
        {}

        void func(void);
};

class DP : public dpath<10, 1>
{
private:
        port<int> (&Inport); port<int> (&Outport);

        net<int> shifted;
```

```
        net<int> Data;
        net<int> Data_in;
        net<int> Ocount; port<int> Mask;
        net<int> Mask_in;
        net<int> alu_in1; port<int> alu_in2; port<int> alu_out;
        net<int> Temp;

        SHIFTER shifter;
        COMP comp;
        REGISTER data_reg, Ocount_reg, Mask_reg, Temp_reg;
        BUS bus1, bus2;
        MUX mux_data;
        ALU alu;

public:
                DP(clock (&clk_local), port<bool> (&reset_local),
                        port<bool> (&ctrl0)[10], port<bool> (&status0)[1],
                        port<int> &Inport, port<int> &Outport)
                : dpath<10, 1>(clk_local, reset_local, ctrl0, status0),
                Inport(Inport), Outport(Outport), Mask_in(1),
                shifter(Data, shifted),
                comp(shifted, status[0]),
                data_reg(clk, reset, ctrl[1], Data_in, Data),
                Ocount_reg(clk, reset, ctrl[2], alu_out, Ocount),
                Mask_reg(clk, reset, ctrl[3], Mask_in, Mask),
                Temp_reg(clk, reset, ctrl[7], alu_out, Temp),
                bus1(ctrl[4], ctrl[8], Data, Ocount, alu_in1),
                bus2(ctrl[5], ctrl[9], Mask, Temp, alu_in2),
                mux_data(ctrl[0], Inport, shifted, Data_in),
                alu(ctrl[6], alu_in1, alu_in2, alu_out)
                {
                }

        void main(void);

};

class Count : public module
{
private:

        clock (&clk); port<bool> (&reset);
        port<bool> (&Start); port<int> (&Inport); port<bool> (&Done); port<int> (&Outport);
        port<bool> ctrl0[10];
        port<bool> status0[1];

        CU cu;
        DP dp;

public:

        Count(clock (&clk), port<bool> (&reset),
                port<bool> (&Start), port<int> (&Inport),
                port<bool> (&Done), port<int> (&Outport))
        : clk(clk), reset(reset), Start(Start), Inport(Inport), Done(Done), Outport(Outport),
        cu(clk, reset, ctrl0, status0, Start, Done),
        dp(clk, reset, ctrl0, status0, Inport, Outport)
        { /* nothing */ };

        void main(void);
};

///////////////////////////////////////////////////////////////////////
// End of file count.h
///////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////
// Design:   one's counter
// Author:   Shuqing Zhao
// Purpose : RTL C++ semantics  experiment
//             -  Style  5 FSMD
// File :      count.cc
// Time:       Thu Jan 25 2001
//////////////////////////////////////////////////////////////////////

//#include <stdio.h>
#include "rtlc.h"
#include "count.h"

void SHIFTER::func(void ) {
        wait(shift_in.e);
        //shift_out =shift_in >> 1;
        assign(shift_out, shift_in >> 1);
        printf("%3lld ns: shift_in/%d shifted_data/%d ",
                now(), shift_in.read(), shift_out.read());
}

void ALU::func(void ) {
        wait(op.e || alu_in1.e  || alu_in2.e);
        if(op==1)
                //alu_out =alu_in1.read() + alu_in2.read();
                assign(alu_out, alu_in1 + alu_in2);
        else
                assign(alu_out, alu_in1 & alu_in2);
        printf("%3lld ns: alu_in1/%d alu_in2/%d alu_out/%d\n",
                now(), alu_in1.read(), alu_in2.read(), alu_out.read());
}

void COMP::func(void ) {
        wait(comp_in.e);
        assign(comp_out, comp_in == 0);
        printf("comp_in/%d.\n", comp_in.read());
}

void MUX::func(void ) {
        wait(en1.e || driver1.e || driver2.e);
        if(en1==0)
                assign(bus_val, driver1);
        else
                assign(bus_val, driver2);
        printf("%3lld ns: DATA_MUX - Inport/%d shifted/%d, Data_in/%d\n",
                now(), driver1.read(), driver2.read(), bus_val.read());
}

void BUS::func(void ) {
        wait(en1.e || en2.e || driver1.e || driver2.e);
        if(en1==1 && en2==1) printf("Bus conflict\n");
        else {if(en1==1)
                assign(bus_val, driver1.read());
        else if(en2==1)
                assign(bus_val, driver2.read());
        printf("%3lld ns: Bus: driver1/%d driver2/%d, alu_in/%d\n",
                now(), driver1.read(), driver2.read(), bus_val.read());
        }
}

void REGISTER::func(void ) {
        wait(clk);
        if(reset==1)      //sync. reset
                assign(Dout, 0);
        else if(ld==1)
                { assign(Dout, Din);
```

```
                printf("%3lld ns: reg: ld/%d Din/%d Dout/%d\n",
                now(), ld.read(), Din.read(), Dout.read());
                }
}

void DP::main(void) {
        Par(
        &mux_data,
        &comp,
        &shifter,
        &alu,
        &bus1,
        &bus2,
        &Mask_reg,
        &Temp_reg,
        &Ocount_reg,
        &data_reg,
        0);
}

void Count::main(void) {
        Par(&dp, &cu, 0);
}

void CU::init(void) {
        state = 0;
        assign(Done, 0);
        //Outport = 0; // being done by datapath reset
        assign(ctrl[0], 0);
        assign(ctrl[1], 0);
        assign(ctrl[2], 0);
        assign(ctrl[3], 0);
        assign(ctrl[4], 0);
        assign(ctrl[5], 0);
        assign(ctrl[6], 0);
        assign(ctrl[7], 0);
        assign(ctrl[8], 0);
        assign(ctrl[9], 0);
        printf("%3lld ns: Oops, reset!\n", now());
}

void CU::trans_act(void) {
        switch(state) {
        case 0:
        assign(Done, 0);
        //Outport = 0;
        if (Start)
        state = 1;
        break;
        case 1:
        // Data = Inport; //style 1
        // write(Data, Inport);
        assign(ctrl[0], 0); //data reg mux sel inport
        assign(ctrl[1], 1); //data reg load
        state = 2;
        break;
        case 2:
        assign(ctrl[1], 0); //data reg load disable

        //write(Ocount, 0);
        //ctrl[2].assign(1); //Ocount reg reset
        state = 3;
        break;
        case 3:
        //ctrl[2].assign(0); //Ocount reg reset off
```

```
        //write(Mask, 1);
        assign(ctrl[3], 1); //Mask reg set
        state = 4;
        break;
        case 4:
        //Bus1 = Data.read();
        //Bus2 = Mask.read();
        //write(Temp, alu(0, Bus1, Bus2));
        assign(ctrl[4], 1); //Data reg output enable
        assign(ctrl[5], 1); //Mask reg output enable
        assign(ctrl[6], 0); //alu & op
        assign(ctrl[7], 1); //Temp reg load
        assign(ctrl[3], 0); //Mask reg set off
        assign(ctrl[1], 0); //data reg load off --from state 6
        state = 5;
        break;
        case 5:
        assign(ctrl[7], 0); //Temp reg load disable
        //Bus1 = Ocount.read();
        //Bus2 = Temp.read();
        //write(Ocount, alu(1, Bus1, Bus2));
        assign(ctrl[4], 0); //Data reg output disable
        assign(ctrl[8], 1); //Ocount reg output enable
        assign(ctrl[5], 0); //Mask reg output disable
        assign(ctrl[9], 1); //Temp reg output enable
        assign(ctrl[6], 1); //alu + op
        assign(ctrl[2], 1); //Ocount reg load
        state = 6;
        break;
        case 6:
        assign(ctrl[8], 0); //Ocount reg output disable
        assign(ctrl[9], 0); //Temp reg output disable
        assign(ctrl[2], 0); //Ocount reg load disable
        // Data=Data>>1;

        //write(Data, Data.read() >> 1);
        assign(ctrl[0], 1); //Data reg mux sel
        assign(ctrl[1], 1); //data reg load

        //if (Data.read() == 0)
        if (status[0] == 1)
        state = 7;
        else
        state = 4;
        break;
        case 7:
        assign(ctrl[1], 0); //data reg load disable

        assign(Done, 1);
        //Outport = Ocount.read(); // directly connected
        state = 0;
        break;}

        printf("%3lld ns: reset/%d Start/%d Done/%d ctrl/%d%d%d%d%d%d%d%d%d%d status/%d, [state/%d].\n",
               now(), reset.read(), Start.read(), Done.read(),
               ctrl[9].read(), ctrl[8].read(), ctrl[7].read(), ctrl[6].read(),
               ctrl[5].read(), ctrl[4].read(), ctrl[3].read(), ctrl[2].read(),
               ctrl[1].read(), ctrl[0].read(), status[0].read(),
               state);
}

////////////////////////////////////////////////////////////////////
// End of file count.cc
////////////////////////////////////////////////////////////////////
```

C C++ source listing for parity checker example(communicating FSMDs)

```
////////////////////////////////////////////////////////////////////////
// Design:   Parity Checker
// Author:   Shuqing Zhao
// Purpose : RTL C++ semantics  experiment
//           - Testbench
// File :    tb.cc
// Time:     Thu Jan 25 2001
////////////////////////////////////////////////////////////////////////

#include "rtlc.h"
#include "sti.h"
#include "parity.cc"

int testbench(int argc, char* argv[])
{

        const int clk1_period = 10;
        const int clk2_period = 7;

        clock clk1, clk2;
        net<bool> reset;

        net<bool> start0; net<bool> done0; //wires
        net<int> indata0; net<int> outdata0;

        CLKGEN ckg1(clk1, clk1_period, 0);
        CLKGEN ckg2(clk2, clk2_period, 0);
        STIGEN sti0(reset, start0, indata0);

        Parity parity0(clk1, clk2, reset, start0, indata0, done0, outdata0);

        Par(&ckg1, &ckg2, &sti0, &parity0, 0);

        return(0);
}
////////////////////////////////////////////////////////////////////////
// End of file tb.cc
////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////
// Design: parity checker
// Author: Shuqing Zhao
// File :   parity.h
// Time:    Tue Jan 16 18:09:50 2001
////////////////////////////////////////////////////////////////////

#include "rtlc.h"

class Even : public fsmd<6>
{
        private:                                            //ports

                port<bool> (&Start); port<int> (&Inport); port<bool> (&Done); port<int> (&Outport);
                port<int> (&idata); port<int> (&iocount); port<bool> (&istart); port<bool> (&idone);

        public:
                Even(clock (&clk), port<bool> (&reset),
                        port<bool> (&Start), port<int> (&Inport), port<bool> (&Done),
                        port<int> (&Outport), port<int> &idata, port<int> &iocount,
                        port<bool> &istart, port<bool> &idone)
                        : fsmd<6> (clk, reset),
                        Start(Start), Inport(Inport), Done(Done), Outport(Outport),
                        idata(idata), iocount(iocount), istart(istart), idone(idone)
                        {};

                void init(void);
                void trans_act(void);

        private:                                            //internal variables

                int mask; int ocount;
};

class Count : public fsmd2<8>
{
        private:                                            //ports

                port<bool> (&Start); port<int> (&Inport); port<bool> (&Done); port<int> (&Outport);

        public:
                Count(clock (&clk), port<bool> (&reset),
                        port<bool> (&Start), port<int> (&Inport),
                        port<bool> (&Done), port<int> (&Outport))
                        : fsmd2<8> (clk, reset),
                        Start(Start), Inport(Inport), Done(Done), Outport(Outport)
                        {};

                void init(void);
                void trans_act(void);

        private:

                reg<int> Data, Mask, Ocount, Temp;
                int Bus1, Bus2;

};

class Parity : public module
{
        private:                                            //ports

                clock (&clk1), &clk2;
                port<bool> (&reset);
                port<bool> (&Start); port<int> (&Inport); port<bool> (&Done); port<int> (&Outport);
```

```
    private:                                        //internal wires
        net<bool> cstart, cdone;
        net<int> data, ocount;

        Count count0;
        Even even0;

    public:

        Parity(clock (&clk1), clock &clk2, port<bool> (&reset),
            port<bool> (&Start), port<int> (&Inport),
            port<bool> (&Done), port<int> (&Outport))
            : clk1(clk1), clk2(clk2), reset(reset),
            Start(Start), Inport(Inport), Done(Done), Outport(Outport),
            count0(clk1, reset, cstart, data, cdone, ocount),
            even0(clk2, reset, Start, Inport, Done, Outport, data, ocount, cstart, cdone)
        {                                                   /* nothing */
        };

        void main(void);
};

//////////////////////////////////////////////////////////////////////////
// End of file parity.h
//////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////
// Design:   parity checker
// Author:   Shuqing Zhao
// Purpose: RTL C++ semantics experiment
//           - communicating/hierarchical FSMDs
// File:     parity.cc
// Time:     Thu Jan 25 2001
//////////////////////////////////////////////////////////////////////

#include "rtlc.h"
#include "parity.h"

void Count::init(void)
{
        state = 0;
        assign(Done, 0);
        assign(Outport, 0);
        write(Data, 0);
        printf("%3lld ns: (Count) Oops, reset!\n", now());
}


//style 4 FSMD
void Count::trans_act(void)
{
        switch(state)
        {
                case 0:
                        assign(Done, 0); assign(Outport, 0);
                        if (Start.read())
                                state = 1;
                        break;
                case 1:
                        write(Data, Inport.read());
                        state = 2;
                        break;
                case 2:
                        write(Ocount, 0);
                        state = 3;
                        break;
                case 3:
                        write(Mask, 1);
                        state = 4;
                        break;
                case 4:
                        Bus1 = Data.read();
                        Bus2 = Mask.read();
                        write(Temp, alu(0, Bus1, Bus2));
                        state = 5;
                        break;
                case 5:
                        Bus1 = Ocount.read();
                        Bus2 = Temp.read();
                        write(Ocount, alu(1, Bus1, Bus2));
                        state = 6;
                        break;
                case 6:
                        // Data=Data>>1;
                        write(Data, Data.read() >> 1);
                        if (Data.read() == 0)
                                state = 7;
                        else
                                state = 4;
                        //Data = Data >> 1;
                        break;
                case 7:
```

```
                        assign(Done, 1); assign(Outport, Ocount.read());
                        if (Start.read())
                                state = 7;
                        else
                                state = 0;
                        break;
        }

        printf("%3lld␣ns:␣(Count)␣reset/%d␣Start/%d␣Done/%d␣Inport/%d␣Outport/%d␣[state/%d␣Data/%2d].\n",
                now(), reset.read(), Start.read(), Done.read(),
                Inport.read(), Outport.read(), state, Data.read());
}

void Even::init(void)
{
        state = 0;
        ocount = 0;
        assign(Done, 0); assign(Outport, 0);
        assign(istart, 0); assign(idata, 0);
        printf("%3lld␣ns:␣(Even)␣␣Oops,␣reset!\n", now());
}


// style 1 FSMD
void Even::trans_act(void)
{
        switch(state)
        {
                case 0:
                        assign(Done, 0); assign(Outport, 0);
                        if (Start.read())
                                state = 1;
                        else
                                state = 0;
                        break;
                case 1:
                        mask = 1;
                        state = 2;
                        break;
                case 2:
                        assign(idata, Inport);
                        assign(istart, 1);
                        state = 3;
                        break;
                case 3:
                        if(idone.read() == 1)
                                state = 4;
                        else
                                state = 3;
                        break;
                case 4:
                        ocount = iocount.read();
                        assign(istart, 0);
                        state = 5;
                        break;
                case 5:
                        assign(Outport, ocount & mask);
                        assign(Done, 1);
                        state = 0;
                        break;
        }

        printf("%3lld␣ns:␣(Even)␣␣reset/%d␣istart/%d␣idone/%d␣Inport/%d␣Outport/%d␣[state/%d␣ocount/%d].\n",
                now(), reset.read(), istart.read(), idone.read(),
                Inport.read(), Outport.read(), state, ocount);
```

```
}

void Parity::main(void)
{
        Par(&count0, &even0, 0);
}

////////////////////////////////////////////////////////////////////////
// End of file parity.cc
////////////////////////////////////////////////////////////////////////
```