Prerequisites

- This lecture assumes a basic understanding of C++ and Verilog-2001
  - For more information on Verilog, see
  - For more information on Verilog-2001, see
  - For more information on C++, see
    - http://www.cplusplus.com
      → Documentation → C++ Language Tutorial
Today’s Lecture

- Verilog RTL Version of an FIR Filter
- Introduction to SystemC
- SystemC RTL Version of an FIR Filter

Direct Form FIR Filter
Design Hierarchy

FIR_cascade

\[ Z^{-1} \]

16 bits for \( rk \) values

32 bits for \( sk \) values

RTL Code for FIR_cell (1)

```
module FIR_cell #(parameter rw=16, sw=32, bw=16)
  (input [rw-1:0] rkm1, input [sw-1:0] skm1, input [bw-1:0] bk,
   input clock, reset, output reg [rw-1:0] rk, output [sw-1:0] sk);

wire [sw-1:0] rkm1_ext, bk_ext;

assign rkm1_ext={{sw-rw{rkm1[rw-1]}},rkm1};
assign bk_ext={{sw-bw{bk[bw-1]}},bk};
assign sk=(rkm1_ext*bk_ext)+skm1;
```

- Why do we define the signals \( rkm1_{-ext} \) and \( bk_{-ext} \)?
**RTL Code for FIR_cell (2)**

```verilog
class always@ (posedge clock)
    if (reset)
        rk<=0;
    else
        rk<=rkm1;
endmodule
```

- Is this a synchronous or asynchronous reset?

**RTL Code for FIR_cascade (1)**

```verilog
module FIR_cascade #(parameter b_Len=9, rw=16, sw=32, bw=16)
    (input [rw-1:0] xk, input [b_Len*bw-1:0] bk, input clock, reset, output [sw-1:0] yk);

    wire [sw-1:0] gnd = 0;
    wire [b_Len*rw-1:0] r;
    wire [(b_Len-1)*sw-1:0] s;
endmodule
```

- Why are the dimensions of \( bk \), \( r \), and \( s \) multiplied by \( b_{Len} \)?
generate Statements

- **generate** statements are a convenient way to make arrays of instances with for loops in Verilog 2001

- Unfortunately, the internal signals of the instances are not visible with **simvision** (Need to use the **NC-Verilog** GUI for that)

- It’s best to use them to create instances of modules that you’ve already verified completely

---

**RTL Code for FIR_cascade (2)**

```verilog
generate
genvar i;
for (i=0; i<b_Len; i=i+1)
begin:fir
    if (i==0)
        FIR_cell #(rw,sw,bw) fir (xk,gnd,bk[bw-1:0],
                        clock,reset,r[rw-1:0],s[sw-1:0]);
    else if (i==b_Len-1)
        FIR_cell #(rw,sw,bw) fir (r[i*rw-1:(i-1)*rw],
                        s[i*sw-1:(i-1)*sw], bk[(i+1)*bw-1:i*bw],
                        clock,reset,r[(i+1)*rw-1:i*rw],yk);
    else
        FIR_cell #(rw,sw,bw) fir (r[i*rw-1:(i-1)*rw],
                        s[i*sw-1:(i-1)*sw],bk[(i+1)*bw-1:i*bw],
                        clock,reset,r[(i+1)*rw-1:i*rw],
                        s[(i+1)*sw-1:i*sw]);
end
endgenerate
endmodule
```
Today’s Lecture

- Verilog RTL Version of an FIR Filter
- Introduction to SystemC
- SystemC RTL Version of an FIR Filter

History of SystemC

- Synopsys began development (1996)
  » originally called “Scenic”

- Version 2.0.1 released on SourceForge.net (May 31, 2000)

- Version 2.1.v1 released on SystemC.org by OSCl (Sept. 19, 2005)

- IEEE Std. 1666-2005 approved (Mar. 28, 2006)

- Version 2.2.05 released on SystemC.org (June 6, 2006)
  » more closely adheres to IEEE Std. 1666-2005
Parts of a Verilog Module

- **Header:** `module <module_name> (<port_list>);
- **Parameter, Port, & Internal Variable declarations**
- **Functionality description**
  - **Structural**
    - Instantiations of basic gates
    - Instantiations of lower-level modules
  - **Behavioral**
    - Data-Flow (continuous assignments)
    - Procedural (initial & always blocks)
- **Terminator:** `endmodule`

Parts of a SystemC Module

- **Declaration:** `class <module_name>: sc_module`
- **Port & Internal Variable declarations**
- **Instance & Process declarations**
- **Constructor**
  - **Structural Functionality Description**
    - Instantiations of lower-level modules
  - **Procedural Functionality Description**
    - Process re-declaration (thread & method processes)
- **Procedural Functionality Description**
  - Process Definitions
Declaring a module in SystemC

- Modules in SystemC are classes that are derived from the `sc_module` class

```
class FIR_cascade : sc_module
{

```

- You will often see the macro `SC_MODULE`, which is replaced with the code above at compile time

```
SC_MODULE(FIR_cascade)
{

```

Declaring Ports

- Ports are member instances of classes derived from `sc_port` (such as `sc_in` and `sc_out`)

```
sc_uint<> types are used to model arbitrary word lengths (see chapter 7 of the IEEE 1666-2005 standard for more)
```

```
sc_out< sc_uint<32> > yk;
sc_in< sc_uint<16> > xk,
sc_in< sc_uint<16> > bk[9];
sc_in<bool> clock, reset;
```

Note that arrays of ports are legal in SystemC.

See the cplusplus.com tutorial on Templates.
Declaring Internal Variables

- Internal Variables (like `wire` and `reg` variables in Verilog) are member instances of other classes

  » If the variable will be used with procedural descriptions (i.e., processes), then the basic data type may be declared.

    ```
    sc_uint<16> r[9];
    sc_uint<32> s[9];
    ```

  » If the variable will be used with structural descriptions (i.e., instantiations) or sensitivity lists, then it must be declared as an `sc_signal<>` type.

    ```
    sc_signal< sc_uint<16> > r[9];
    sc_signal< sc_uint<32> > s[9];
    ```

Declaring Instances & Processes

- Instances of lower-level modules must have member variables to access them

  ```
  FIR_cell *cell[9];
  ```

- SystemC processes are like `always` blocks in Verilog
  
    » They are essentially class methods with no arguments or return value
    
    » They must be declared in the class definition

  ```
  void unit_delay ();
  ```
Parts of a SystemC Module

- Declaration: class \textit{<module\_name>: sc\_module}
  - Port & Internal Variable declarations
  - Instance & Process declarations

- Constructor
  - Structural Functionality Description
    - Instantiations of lower-level modules
  - Procedural Functionality Description
    - Process re-declaration (thread & method processes)

- Procedural Functionality Description
  - Process Definitions

Constructor Declaration

- Before the constructor is declared, the type \textit{SC\_CURRENT\_USER\_MODULE} is defined to the current module name
  - This type is used during process re-declaration

\begin{verbatim}
typedef FIR_cascade SC\_CURRENT\_USER\_MODULE;
FIR_cascade(sc\_module\_name)
\end{verbatim}

- SystemC defines the macro \textit{SC\_CTOR} to represent this code more conveniently

\begin{verbatim}
SC\_CTOR(FIR\_cascade)
\end{verbatim}

- SystemC Also defines the macro \textit{SC\_HAS\_PROCESS} to represent only the \texttt{typedef} statement
  - This is helpful if the constructor needs extra arguments, as is the case with parameterized modules

\begin{verbatim}
SC\_HAS\_PROCESS(FIR\_cascade);
FIR\_cascade(sc\_module\_name)
\end{verbatim}
**Instantiations**

- Module instances are created in the constructor
- The first argument is always the instance name

```
cell[1] = new FIR_cell("cell[1]");
```

- Ports are bound to internal signals by calling the port constructors for the instance with the name of the internal signal (this is called “binding”)

```
cell[1] -> rk(r[1]);
cell[1] -> sk(s[1]);
cell[1] -> rkml(r[0]);
cell[1] -> skml(s[0]);
cell[1] -> bk(bk[1]);
cell[1] -> clock(clock);
cell[1] -> reset(reset);
```

**Process Re-Declaration**

- Although processes were declared with the class, they must also be declared to the SystemC scheduler at run-time. (I call this re-declaration)
- SystemC defines the macros `SC_METHOD` and `SC_THREAD` for re-declaration
  - thread processes are like Verilog `always` blocks without a sensitivity list
    ```sc
    SC_THREAD(unit_delay);
    ```
  - method processes are like Verilog `always` blocks with a sensitivity list
    ```sc
    SC_METHOD(unit_delay);
    sensitive_pos << clock;
    ```
  - the `sensitive_pos` function is used to declare the signals in the sensitivity list that trigger the process on their rising edges.
Parts of a SystemC Module

- Declaration: class `<module_name>`: `sc_module`
  » Port & Internal Variable declarations
  » Instance & Process declarations

- Constructor
  » Structural Functionality Description
    – Instantiations of lower-level modules
  » Procedural Functionality Description
    – Process re-declaration (thread & method processes)

- Procedural Functionality Description
  » Process Definitions

Process Definitions

- Finally, write the C++ code to define the processes

```cpp
void FIR_cell::unit_delay ()
{
    if (reset.read()) {
        rk = 0;
    }
    else {
        rk = rkm1.read();
    }
}
```

when referencing the value of a port, the read() method should be used

an assignment to an output port is considered to be a write to that port.
Today’s Lecture

- Verilog RTL Version of an FIR Filter
- Introduction to SystemC
- SystemC RTL Version of an FIR Filter

RTL Code for FIR_cell.h

```c
#include "systemc.h"
SC_MODULE (FIR_cell)
{
    sc_out< sc_uint<16> > rk;
    sc_out< sc_uint<32> > sk;

    sc_in< sc_uint<16> > rkml, bk;
    sc_in< sc_uint<32> > skml;
    sc_in<bool> clock, reset;

    void unit_delay ();
    void mult_add ();

    SC_CTOR (FIR_cell)
    {
        SC_METHOD(unit_delay);
        sensitive_pos << clock;
        SC_METHOD(mult_add);
        sensitive << rkml << skml << bk;
    }
};
```
### RTL Code for FIR_cell.cc

```cpp
#include "FIR_cell.h"

void FIR_cell::unit_delay ()
{
    if (reset.read()) {
        rk = 0;
    }
    else {
        rk = rkm1.read();
    }
}

void FIR_cell::mult_add ()
{
    sc_uint<32> tmp;
    tmp=(sc_int<16>)rkm1.read()*(sc_int<16>)bk.read();
    sk=tmp+skm1.read();
}
```

### RTL Code for FIR_cascade.h (1)

```cpp
#include "systemc.h"
#include "FIR_cell.h"
#include <stdio.h> // to define sprintf()

SC_MODULE (FIR_cascade)
{
    sc_out< sc_uint<32> > yk;
    sc_in< sc_uint<16> > xk, bk[9];
    sc_in<bool> clock, reset;

    sc_signal< sc_uint<16> > r[9];
    sc_signal< sc_uint<32> > s[9], gnd;

    FIR_cell *cell[9];
```
**RTL Code for FIR_cascade.h (2)**

```
SC_CTOR (FIR_cascade)
{
    int i;
    char buf[9];
    gnd=0;
    for (i=0; i<9; i++) {
        sprintf(buf,"cell[%d]",i);
        cell[i] = new FIR_cell(buf);
        if (i==0) {
            cell[i] -> rk(r[i]);
            cell[i] -> sk(s[i]);
            cell[i] -> rkm1(xk);
            cell[i] -> skm1(gnd);
            cell[i] -> bk(bk[i]);
            cell[i] -> clock(clock);
            cell[i] -> reset(reset);
        } else if (i==8) {
            cell[i] -> rk(r[i]);
            cell[i] -> skm1(s[i-1]);
            cell[i] -> bk(bk[i]);
            cell[i] -> clock(clock);
            cell[i] -> reset(reset);
        } else {
            cell[i] -> rk(r[i]);
            cell[i] -> sk(s[i]);
            cell[i] -> rkml(r[i-1]);
            cell[i] -> skml(s[i-1]);
            cell[i] -> bk(bk[i]);
            cell[i] -> clock(clock);
            cell[i] -> reset(reset);
        }
    }
}
```

**Questions**

- Why does FIR_cell have two processes?

- Why is there no FIR_cascade.cc file?

- In practice, would we want to write a SystemC description like this one?