

BEE Design Flow Tutorials

Lesson 3.1: Control Logic Design

Chen Chang
Kevin Camera
Hayden So

Lesson Goals

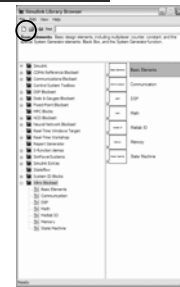
- Use a simple counter as example explore 3 different ways construct finite state machine:
 - Using M-code block in XSG v3.1
 - Using StateFlow
 - Using VHDL blackbox
- VHDL co-simulation in Simulink with ModelSim

What do you need to start?

- Successful completion of BEE Design Flow Tutorials Lesson 1.1: Flow Basics

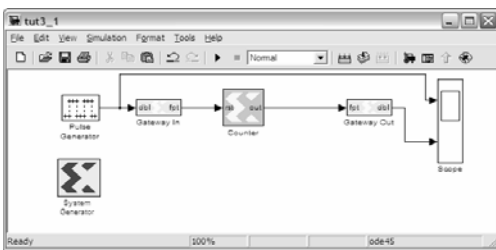
Starting Matlab/Simulink

- First start Matlab, then type "simulink" in the command window
- Type "path" in the command window and verify [WhitzDesigns\BEE\mlib](http://whitzdesigns.com/BEE/mlib) is included
- Check the "Xilinx Blockset" toolbox is installed in the library browsing window
- Create a new model
 - Files -> New -> Model



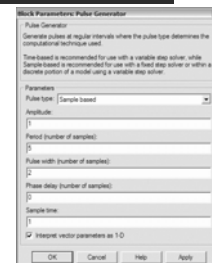
Construct the new system

- Construct the design as shown below, and save as "tut3.mdl"



Configure the Pulse Generator

- Pulse type: Sample based
- Amplitude: 1
- Period: 5
- Pulse width: 2
- Phase delay: 0
- Sample time: 1



Configure the Gateway In

- Output Data Type: Boolean
- Sample Period: 1



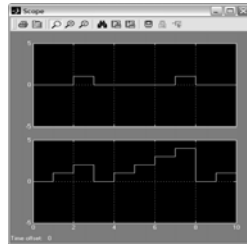
Configure the Counter

- Counter Type: Free Running
- Number of Bits: 8
- Binary Point Position: 0
- Arithmetic Type: Unsigned
- Initial Value: 0
- Step: 1
- Check "Provide Reset Port"



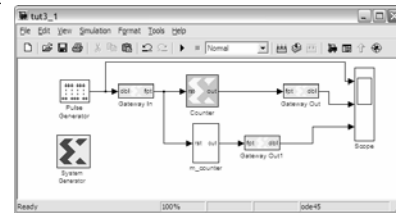
Simulink simulation #1

- Now simulate the design, the scope block should show a wave form like the one on the right.
- The first wave is the input reset pulse
- The second is the output of the counter, which responds to the reset input pulse



Adding a new subsystem

- Add a new subsystem and renamed it "m_counter", and connect through a new Xilinx Gateway Out to the scope
- Rename the m_counter subsystem input as "rst" and output as "out"



Steps to use Mcode block

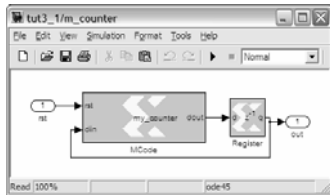
- Create the Matlab script file that defines the asynchronous state/output update logic function in M language
- Insert the Mcode block from Xilinx Blockset / Control Logic
- Insert state register and any necessary output registers
- Connect the Mcode block with the registers

The Mcode M script

- Create a M script named "my_counter.m" as below, and save it in the same directory as the Simulink MDL file "tut3_1.mdl"
- ```
function dout = my_counter(rst,din)
if rst == xfix({xlBoolean},1)
 dout = 0;
else
 dout = din + 1;
end
dout = xfix({xlUnsigned,8,0},dout);
```

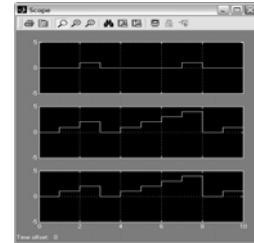
## Construct the Mcode state machine

- Descend into the "m\_counter" subsystem, and add the Mcode and Register as shown below
- Configure the MCode to use the Matlab function "my\_counter", and Register to use explicit sample period of 1



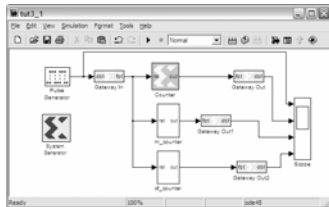
## Simulink simulation #2

- Now simulate the design, the scope block should show a wave form like the one on the right.
- Note the newly added MCode state machine behave exactly the same as the Xilinx counter



## Adding a new subsystem

- Add a new subsystem and renamed it "sf\_counter", and connect through a new Xilinx Gateway Out to the scope
- Rename the m\_counter subsystem input as "rst" and output as "out"



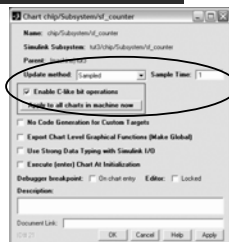
## Add a StateFlow Chart

- Decent into tut3/sf\_counter
- Add a StateFlow chart object from the Simulink/Stateflow library



## Modify Chart Options

- Double click on the "Chart" object
- Click on File->Chart Properties
- Check the "Enable C-like bit operations" option
- This option is required to make the Stateflow simulation match the hardware behavior
- Use "Sampled" Update method, and make the Sample Time matching the input sample time, in this case is the default "1"



## Add Chart Inputs

- Click on Add->Data->Input from Simulink...
- Create the new input "reset" as Type "boolean"
- This correspond to a single bit boolean input port



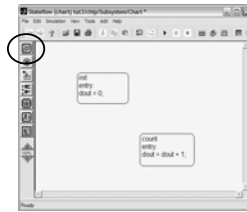
## Add Chart Outputs

- Click on Add->Data->Output from Simulink...
- Create the new output "dout" as Type "uint16" with Min = 0 and Max = 255
- This correspond to an 8-bit unsigned integer output in hardware
- The Min/Max value are used to determine the hardware bitwidth, not the Type "uint16", which only indicates the data type of unsigned integer



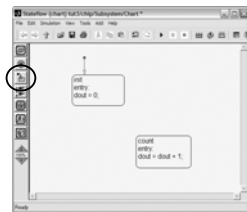
## Add States to the Chart

- Add two states: init, count
- The "entry:" label indicate what the state will do on entry



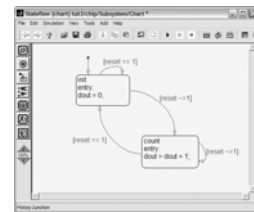
## Add default transition

- Label the "init" state as the default state by dragging the default transition arrow to it.
- The "init" state is required for all charts. It should contain all initial conditions, such as the output value.



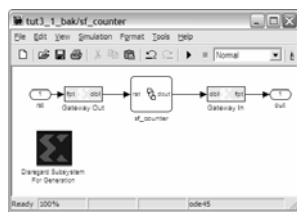
## Add Transitions

- Add transitions by click and drag from the source to the destination state to create the arrow.
- Add the transition condition encapsulated by "[", "]"
- Make sure that all transitions are explicitly defined, especially the self-state transition.
- Also the transition conditions need to cover all possible input/state combinations.



## Connect StateFlow to Simulink

- Save and exist the StateFlow diagram editor
- Change the chart name to "sf\_counter" in Simulink
- Add Xilinx Gateway In, Gateway Out, and Discard Subsystem
- Connect as shown below



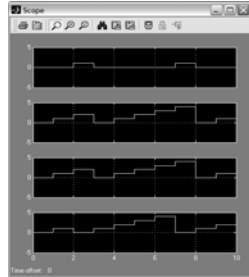
## Modify the "Gateway In"

- Since the StateFlow connects to the Simulink environment using floating point number representation, the Xilinx gateways are need to convert them to fix-point representation, but in the opposition direction as in Lesson 1
- Output of the StateFlow chart need to be translated, so the "Gateway In" need to be modified.
- Make sure that the data type and bitwidth in the gateway matches the StateFlow data type and Min/Max values.
- In this case, make the Output Data Type "Unsigned", and 8 bits with 0 binary point.
- Sample period should match the StateFlow chart "Sample time" options as well, so in this case is "1"



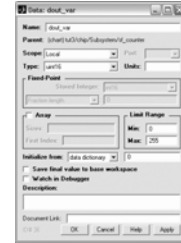
### Simulink simulation #3

- Now simulate the design.
- Notice that the sf\_counter is exactly one cycle ahead of the Xilinx counter
- This is due to the fact that StateFlow is a Mealy machine.



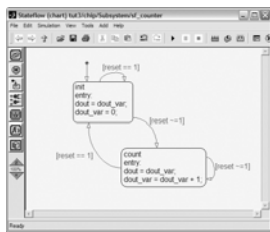
### Add a local variable

- To make the StateFlow run as a Moore machine, local variables are needed.
- Go back to the StateFlow chart editor, click on Add->Data->local
- Add a local variable named "dout\_var" with the same type and Min/Max value as the output "dout"



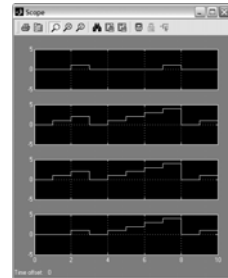
### Modify the state actions

- Change the state actions as below, basically update the "dout\_var" as before, but only output the previous cycle "dout\_var" value to the output "dout"



### Simulink simulation #4

- Now simulate the design again
- The output from sf\_counter matches the Xilinx counter after the first reset pulse
- The initial condition is difference in StateFlow than other Xilinx blocks



### Runing SF2VHD

- Run the following command to generate the VHDL from the StateFlow chart  
`bee_sf2vhd('tut3_1', './', '3.1')`
- This creates the VHDL file "sf\_counter.vhd" in the current direct (same as the MDL file)

### Add the VHDL as Blackbox

- Add a Blackbox block to the top-level Simulink design, a diagram will pop up to select the VHDL file, choose "sf\_counter.vhd"



## Blackbox Configuration

- After adding the blackbox a Matlab configuration file "sf\_counter\_config.m" is automatically created
- If the state machine designed in StateFlow is a Moore machine, find and comment out the following line  

```
this_block.tagAsCombinational;
```
- Find the line  

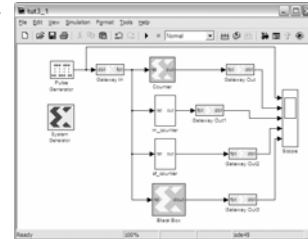
```
"this_block.addFile('sf_counter.vhd');"
```

and add the following line above it:  

```
this_block.addFile('sf2vhd_helper.vhd');
```

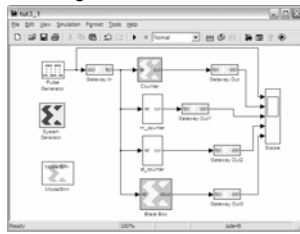
## Connecting the Blackbox

- Finish connecting the blackbox with the rest of the system.



## VHDL co-simulation with ModelSim

- To simulate the VHDL blackbox, add a ModelSim block to the design



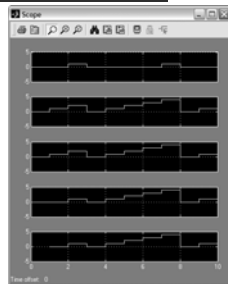
## Configure the blackbox to co-simulate

- Configure the blackbox block for ModelSim co-simulation
- Simulation Mode: Use HDL Co-Simulation
- HDL Co-Simulator: ModelSim



## Simulink simulation #4

- Now simulate the design again
- The VHDL co-simulation result matches the StateFlow result
- The first cycle difference is due to unassigned initial signal values in VHDL



## Hardware Generation

- The rest of the FPGA/ASIC generation matches identically as described in Lesson 1
- The StateFlow subsystem is tag as "Discard Subsystem" and will not be generated in the final hardware
- Optionally, "Simulation Multiplexer" blocks can be used to merge the outputs between the StateFlow subsystem and the VHDL Blackbox. In this case, the co-simulation result will not be used.

## Congratulations

- You have just finished the third BEE Design Flow tutorial lesson
- For more detailed instruction on how to use the tools introduced, please read the user's guide or the manual of the tools from the [BEE web site](#)