ECE 274 – Digital Logic

Venlog

Venlog for Digital Design (Vahid, Lysecky): Ch. 5

High-Level State Machine Behavior

- Module ports same as FSM
- Same two-procedure approach as FSM
  - One for combinational logic
  - Registers now include explicit registers (\( \text{Cnt} \))
    - Two reg variables per explicit register (\( \text{Cnt} \) and \( \text{CntNext} \)), just like for state register

Simulating the HLSM

- Use same testbench as in Chapter 3
- Waveforms below also show \( \text{Cnt} \) and \( \text{State} \) variables, even though not a port on the LaserTime module
  - Simulators allow one to zoom into modules to select internal variables/nets to show

Note: Writes are to “current” variable, reads are from “next” variable. See target architecture to understand why.

LaserTime module variables, even though not a port on the LaserTime module

Simulators allow one to zoom into modules to select internal variables/nets to show

- Note reset behavior
  - Until \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) and \( \text{State} \) undefined
  - Upon \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) set to 0, and \( \text{State} \) set to S_Off (which is defined as 0)

- Note how system enters S_On on first rising clock after \( \text{B} \) becomes 1, causing \( \text{Cnt} \) to be initialized to 2
- \( \text{Cnt} \) is decremented in S_On

- Until \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) and \( \text{State} \) undefined
- \( \text{Simulators} \) allow one to zoom into modules to select internal variables/nets to show

- Note reset behavior
  - Until \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) and \( \text{State} \) undefined
  - Upon \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) set to 0, and \( \text{State} \) set to S_Off (which is defined as 0)

- Note how system enters S_On on first rising clock after \( \text{B} \) becomes 1, causing \( \text{Cnt} \) to be initialized to 2
  - \( \text{Cnt} \) is decremented in S_On
  - \( \text{Cnt} \) wrapped from 0 to 3, but the 3 was never used

- Use same testbench as in Chapter 3
- Waveforms below also show \( \text{Cnt} \) and \( \text{State} \) variables, even though not a port on the LaserTime module
  - Simulators allow one to zoom into modules to select internal variables/nets to show

- Note reset behavior
  - Until \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) and \( \text{State} \) undefined
  - Upon \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) set to 0, and \( \text{State} \) set to S_Off (which is defined as 0)

- Note how system enters S_On on first rising clock after \( \text{B} \) becomes 1, causing \( \text{Cnt} \) to be initialized to 2
  - \( \text{Cnt} \) is decremented in S_On
  - \( \text{Cnt} \) wrapped from 0 to 3, but the 3 was never used

- Use same testbench as in Chapter 3
- Waveforms below also show \( \text{Cnt} \) and \( \text{State} \) variables, even though not a port on the LaserTime module
  - Simulators allow one to zoom into modules to select internal variables/nets to show

- Note reset behavior
  - Until \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) and \( \text{State} \) undefined
  - Upon \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) set to 0, and \( \text{State} \) set to S_Off (which is defined as 0)

- Note how system enters S_On on first rising clock after \( \text{B} \) becomes 1, causing \( \text{Cnt} \) to be initialized to 2
  - \( \text{Cnt} \) is decremented in S_On
  - \( \text{Cnt} \) wrapped from 0 to 3, but the 3 was never used

- Use same testbench as in Chapter 3
- Waveforms below also show \( \text{Cnt} \) and \( \text{State} \) variables, even though not a port on the LaserTime module
  - Simulators allow one to zoom into modules to select internal variables/nets to show

- Note reset behavior
  - Until \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) and \( \text{State} \) undefined
  - Upon \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) set to 0, and \( \text{State} \) set to S_Off (which is defined as 0)

- Note how system enters S_On on first rising clock after \( \text{B} \) becomes 1, causing \( \text{Cnt} \) to be initialized to 2
  - \( \text{Cnt} \) is decremented in S_On
  - \( \text{Cnt} \) wrapped from 0 to 3, but the 3 was never used

- Use same testbench as in Chapter 3
- Waveforms below also show \( \text{Cnt} \) and \( \text{State} \) variables, even though not a port on the LaserTime module
  - Simulators allow one to zoom into modules to select internal variables/nets to show

- Note reset behavior
  - Until \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) and \( \text{State} \) undefined
  - Upon \( \text{Rst} = 1 \) and rising clock, \( \text{Cnt} \) set to 0, and \( \text{State} \) set to S_Off (which is defined as 0)

- Note how system enters S_On on first rising clock after \( \text{B} \) becomes 1, causing \( \text{Cnt} \) to be initialized to 2
  - \( \text{Cnt} \) is decremented in S_On
  - \( \text{Cnt} \) wrapped from 0 to 3, but the 3 was never used
Top-Down Design: HLSM to Controller and Datapath

- Recall from Chapters 2 & 3
  - Top-down design
    - Capture behavior, and simulate
    - Capture structure (circuit), simulate again
    - Gets behavior right first, unfettered by complexity of creating structure

Top-Down Design: HLSM to Controller and Datapath

- Deriving a datapath from the HLSM
  - Replace HLSM by FSM that uses the datapath

Top-Down Design: HLSM to Controller and Datapath

- Describe controller and datapath in VHDL
  - One option: structural datapath, behavioral (FSM) controller
  - Let’s instead describe both behaviorally

Describe a Datapath Behaviorally

- Two procedures
  - Combinational part and register part
  - Current and next signals shared between the two parts
    - Just like for FSM behavior

Verilog for Digital Design
Copyright © 2007
Frank Vahid and Roman Lysecky
Describing the Controller

- **Behaviorally**
  - Standard approach for describing FSM
    - Two procedures
  - Inputs: B, Cnt_Eq_0; Outputs: X, Cnt_Sel, Cnt_Ld

Controller and Datapath

- Result is one module with four procedures
  - Datapath procedures (2)
    - Combinational logic
      - Registers
  - Controller procedures (2)
    - Combinational logic
      - Registers

One Procedure FSM Description

- Likewise, can describe FSM using one procedure rather than two
  - Same approach
    - One variable for state (rather than two)
  - Same issues as with HLSM
    - Simpler, cleaner code: one procedure per FSM
  - Timing may change, makes inputs synchronous, may introduce extra cycles

Timing Differences Between Two and One Procedure Descriptions

- Previous two procedure description
  - Change in B immediately noticed by combinational logic procedure, which configures next state to be S_On. State changes to S_On (1) on next rising clock (setting X=1)
- One procedure description
  - Change in B not noticed until next rising clock, so next state will be S_Off (0). After rising clock, procedure configures next state to be S_On (1). State changes to S_On on the next rising clock (setting X=1) – second rising clocks after b changed.

Synchronization of input can delay impact of input changes
Improving Timing Realism

- Real components have delays
- Suppose comparator has 7 ns delay

Simulation with Timing Delays Added

- Waveforms show key internal signals:
  - Without comparator delay
  - With comparator delay (note shift)

Algorithmic-Level Behavior for SAD

- Most easily described as an algorithm:
  - Input is two arrays A and B
  - Wait until Go is '1'
  - Use for loop to step through each array item
    - Adds absolute difference to running sum
    - Update output after loop completes
- Note: No intention of providing this description to synthesis tool
- Merely used for early system simulation

Delay Control on Right Side of Assignment Statements

- Delays can be described by including delay control in assignment statement
- Example:
  - assign x = #50 y;
  - This function named "ABS" returns integer value
    - Returns absolute value of input
    - Should be used as 
      - Add absolute value to running sum
      - Add absolute value to new number

Algorithmic-Level Behavior for SAD

- Description uses several language features to be described on next several slides
- User-defined function:
  - This function named "ABS" returns integer value
  - Each input argument, integer
  - 256-byte array
  - Contents assign return value to be used in an expression
    - Returns absolute value
      - Will compute absolute value
      - Absolute value of input
  - Inside function, integer
    - Integer array
    - Absolute value
    - Absolute value of input

Simulation with Timing Delays Added

Algorithmic-Level Behavior for SAD

Algorithmic-Level Behavior for SAD

- Description declares A and B as 256-element arrays of bytes.
- Initializes those arrays using built-in system task.
- Reads file of hex numbers (first argument).
- Each number placed into subsequent element of array as assigned to each.
- Number of numbers in file should match number of elements.
- Separated by white space.
- e.g. (DE FB A4 01).
- Simulation generates SAD_out.
- Testbench: Reads file of binary numbers.
- Note: Called as only statement of "initial" procedure.
- Could have used begin-end block: initial.
- SAD_Out is uninitialized at first, resulting in unknown value.
- Activate SAD with Go_s <= 1.

```
module SAD(Go, SAD_Out, Clk, Rst);
`timescale 1 ns/1 ns

// Vector Procedure
parameter S0 = 0, S1 = 1;
input Clk, Rst;
input Go;

// Clock Procedure
always begin
    #((256*2+3)*20) if (SAD_Out_s != 4) begin
        Go_s <= 0;
    end
    if (Rst==1) begin
        SAD_Reg <= Sum;
        #50;
    end
    else if (Go==1) begin
        SAD_Reg <= Sum;
        #50;
    end
end
```

Event Control with an Expression

- Uses @(Go=1) — Event control with an expression.
- If Go not 1, wait until Go becomes 1.
- Previous event control expressions were either one event, such as @(X) or @(posedge Clk), or a list of events such as @(X,Y).
- But expression can be a longer expression too, like @(Go==1).

```
always begin
    if (Go==1) begin
        #50;  // Delay of 50 ns added just so that result doesn't appear instantaneously
    end
end
```

Convert Algorithm to HLSM

Local registers: Sum, SAD_Reg (32 bits); I (integer)
```
time=10,290 ns
module Testbench();
reg Go;
reg Clk;
reg [31:0] Sum, SAD_Reg;
reg [31:0] SAD_Out;
reg [7:0] SAD_Reg_s, Sum_s;

// Clock Procedure
always begin
    #10
    if (Clk'z_event) begin
        Clk_s <= 1;
        #10
        Clk_s <= 0;
    end
end
```

Describe HLSM in Verilog

```
time=10,290 ns
module SAD(Go, SAD_Out, Clk, Rst);
    // Clock Procedure
    always @(posedge Clk)
        if (Go) begin
            // HLSM estimates better clock cycle accuracy than algorithmic
            // Implementation -- Proceed to convert HLSM to controller/datapath, as before
            SAD_Out <= Sum;
            #50;
        end
endmodule
```

Describe HLSM in Verilog

```
time=10,290 ns
module SAD(Go, SAD_Out, Clk, Rst);
    // Clock Procedure
    always @(posedge Clk)
        if (Go) begin
            // HLSM estimates better clock cycle accuracy than algorithmic
            // Implementation -- Proceed to convert HLSM to controller/datapath, as before
            SAD_Out <= Sum;
            #50;
        end
endmodule
```
Automated Synthesis from the Algorithmic Level

- Behavioral synthesis (or high-level synthesis) tools
  - Introduced in the 1990s and 2000s
  - Automatically synthesize algorithmic-code to RTL code or directly to circuits
  - Not yet adopted as widely as RTL synthesis tools

- Each tool imposes strong requirements and restrictions on the algorithmic-level code

- Higher-level descriptions yield faster simulations
  - Due to fewer procedures, fewer variables, fewer events, ...
  - 10x difference means difference between tens of minutes versus hours; 100x could mean days
  - High-level descriptions (e.g., algorithmic) preferred early in design, and for integrated systems with many (hundreds) of sub-systems

- One-procedure clocked procedure state machine is also faster than procedure approach where one procedure is combinational
  - Again, fewer procedures, fewer variables, fewer events, ...

- Differences become significant for very large systems, or for very lengthy testbenchs

Simulation Speed

- Differences become significant for very large systems, or for very lengthy testbenchs

Accessing Memory

- Memory may be initially accessed as array for simplicity
  - SAD example
    - Declared two 256-item arrays
    - Extra state necessary
    - Closer to real implementation

- Simple read-only memory
  - Addr and data ports only
  - Declares array named Memory for storage
  - Procedure uses continuous assignment statement to always output Memory data element corresponding to current address input value

Simple Memory Entity

- Simple read-only memory
  - Addr and data ports only
  - Declares array named Memory for storage
  - Procedure uses continuous assignment statement to always output Memory data element corresponding to current address input value

Accessing Memory

- Modify SAD’s HLSM with extra state (S3a) for reading the memories
  - Extra state necessary due to use of fully-synchronous HLSM (modeled as one procedure sensitive only to clock)
    - A_addr & B_addr are output ports connected to memory address inputs
    - A_data and B_data are input ports connected to memory data outputs

- Local registers: Sum, SAD_Reg (32 bits); I (integer)
Testbench

- Instantiate and connect modules for SAD, SADMemA, and SADMemB
  - Note two instances of same memory
- Initialize memories
  - Note how we can access SADMemA.Memory (and B) from testbench
- Vector procedure adjusted from earlier to account for extra state
  - \((256^2+3)/20\) changed to \((256^3+3)/\text{ClkPeriod}\)
  - Also note use of parameter ClkPeriod – allows us to change period in one place

```verilog
module Testbench();
  reg Go_s;
  wire [7:0] A_Addr_s, B_Addr_s;
  wire [7:0] A_Data_s, B_Data_s;
  reg Clk_s, Rst_s;
  wire [31:0] SAD_Out_s;
  parameter ClkPeriod = 20;
  SAD CompToTest(Go_s, A_Addr_s, A_Data_s, B_Addr_s, B_Data_s, SAD_Out_s, Clk_s, Rst_s);
  SADMem SADMemA(A_Addr_s, A_Data_s);
  SADMem SADMemB(B_Addr_s, B_Data_s);
  // Clock Procedure
  always begin
    Clk_s <= 0; #(ClkPeriod/2);
    Clk_s <= 1; #(ClkPeriod/2);
  end
  // Initialize Arrays
  initial $readmemh("MemA.txt", SADMemA.Memory);
  initial $readmemh("MemB.txt", SADMemB.Memory);
  // Vector Procedure
  initial begin
    ... // Reset behavior not shown
    Go_s <= 1;
    @(posedge Clk_s);
    Go_s <= 0;
    if (SAD_Out_s != 4) begin
      $display("SAD failed -- should equal 4");
    end
  end
endmodule
```

Waveforms

- Waveforms now include address and data lines with memories
- We also added some internal signals, State, I, Sum, and SAD_Reg
- SAD_Out result appears later than previously, due to extra state