Intro to Digital Design

Project Tips, Memory

Instructor: Justin Hsia

Teaching Assistants:
Emilio Alcantara  Eujean Lee
Naoto Uemura     Pedro Amarante
Wen Li
Lab 8 – Project

- 2 weeks to work on it – don’t wait to start!
  - Reports due Friday, May 31 @ 11:59 pm
  - Lab 8 check-in due next week during demo slot
  - Demos can be scheduled outside of the lab hours by making a private post on Ed Discussion

- 8 suggested projects, or get your own approved
  - Most use LED breakout board – included in your lab kit
  - Not all are worth the same number of points (“full credit” is 150)
  - Think carefully about what you want to tackle (e.g., complex FSM, LED board, multiple “clock speeds”)

- Bonus points for adding cool features and early finish
  - Up to 20 points for extra features; up to 10 points for early finish
Practice

- Implement a **counter** that goes through the state sequence 000→001→011→010→110→111→101→100→000→...
  - Include an Enable signal to count and a Reset signal (to 000)

<table>
<thead>
<tr>
<th>$P_2$</th>
<th>$P_1$</th>
<th>$P_0$</th>
<th>$N_2$</th>
<th>$N_1$</th>
<th>$N_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- $N_2 = P_2P_0 + P_1\overline{P_0}$
- $N_1 = \overline{P_2}P_0 + P_1\overline{P_0}$
- $N_0 = \overline{P_2}P_1 + P_2P_1$
Outline

❖ Project Tips
  ▪ “Multiple clocks”
  ▪ Verilog generate
  ▪ SystemVerilog Arrays

❖ Computer Components
  ▪ Memory/RAM
Comparator (Multibit)

❖ Equality (A == B)
  ▪ XNOR corresponding bits of A and B, then AND together
  ▪ NOR all bits of A–B

❖ Comparator (A < B, A == B, A > B)
  ▪ A < B:  MSB of A–B
  ▪ A == B:  NOR of all bits of A–B
  ▪ A > B:  NOT of MSB of A–B
“Multiple Clocks” Via Counters

- The `clock_divider` module is a 32-bit up counter
  - All output bits update at same time ($t_{C2Q}$)
  - Output bits get us powers of 2 differences in speed
- Still want to use *single* clock for all state elements
  - We will instead control actions using the Enable signal
- Use comparator on a counter as Enable signal
  - May need to feedback into Reset signal on counter
Advanced Verilog: \textit{generate}

- Condense your code using loops and conditionals
  - Often used with \texttt{assign} and module instantiation

- Details:
  - Loop variables must be declared as \texttt{genvar} outside of \texttt{generate} statement
  - Block statements (\texttt{for/if}) \textit{must} have \texttt{begin} and \texttt{end} and be labeled

```
genvar <loop_var>;

generate
    for (<init>; <cond>; <update>) begin : <label>
        // do something with <loop_var>
    end
endgenerate
```
Add/Sub in Verilog (parameterized)

- Variable-width add/sub (with overflow, carry)

```verilog
module addN #(parameter N=32) (OF, CF, S, sub, A, B);
output logic OF, CF;
output logic [N-1:0] S;
input logic sub;
input logic [N-1:0] A, B;
logic [N-1:0] D;  // possibly flipped B
logic C2;  // second-to-last carry-out

always_comb begin
    D = B ^ {N{sub}};  // replication operator
    OF = CF ^ C2;
end
endmodule
```
Add/Sub in Verilog (generate)

❖ Generate produces N fulladd modules

```verilog
module addNgen #(parameter N=32) (OF, CF, S, sub, A, B);
  output logic OF, CF;   // overflow and carry flags
  output logic [N-1:0] S; // sum output bus
  input logic sub;       // subtract signal
  input logic [N-1:0] A, B; // input busses
  logic [N:0] C;         // carry signals between modules
```

❖ Reminder: module fulladd (cout, s, cin, a, b);
SystemVerilog Arrays

- A bus is known as a vector or packed array
  - e.g., `logic [31:0] divided_clocks;`
  - Can only be made of single bit datatypes

- “Regular” array syntax is known as an unpacked array
  - e.g., `logic an_unpacked_array[4:0];`
  - Can be made of any datatype

- Multidimensional arrays can be combinations of packed and unpacked dimensions
  - e.g., `logic [3:0] two_D_array[4:0];`
  - Accessed left to right, starting with unpacked dimensions
Outline

❖ Project Tips
  ▪ “Multiple clocks”
  ▪ Verilog generate
  ▪ SystemVerilog Arrays

❖ Computer Components
  ▪ Memory/RAM
Five Components of a Computer

- Components a computer needs to work:
  - Control
  - Datapath
  - Memory
  - Input
  - Output
 Executing an Instruction

❖ Depends on ISA, but generally:
  ▪ Instruction Fetch
  ▪ Instruction Decode
  ▪ Data Fetch
  ▪ Computation
  ▪ Store Result

❖ Basic Datapath Components (idealized)
  ▪ Register File
  ▪ Memory Management Unit
  ▪ Arithmetic Logic Unit (ALU)
  ▪ Routing Elements
Storage Element: Idealized Memory

- Memory (idealized)
  - One input bus: DataIn
  - One output bus: DataOut
  - In reality, often combined

- Memory access:
  - Read: Data at Address placed on DataOut
  - Write: If WriteEnable = 1, DataIn written to Address

- For N addresses, need Address input to be \( (\log_2 N) \)-bits wide
- Clock (CLK) is a factor ONLY during write operation
8x4 RAM

WriteEnable

<table>
<thead>
<tr>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>In3</td>
<td>In2</td>
<td>In1</td>
<td>In0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3-bit DEMUX

S2 S1 S0

A2 A1 A0

Out3 Out2 Out1 Out0
RAM Cell

❖ Requirements:
  ▪ Store one bit of data
  ▪ Change data based on input when row is selected

❖ Just a controlled register!
  ▪ No need to Reset
  ▪ Use RowSelect as Enable
Verilog Memories

```
module memory16x8 (data_out, data_in, addr, write, clk);

output logic [7:0] data_out;
input  logic [7:0] data_in;
input  logic [3:0] addr;
input  logic write, clk;

logic [7:0] mem [15:0];  // array of vectors

assign data_out = mem[addr];

always @(posedge clk)
  if (write)
    mem[addr] <= data_in;

endmodule
```