Lecture 7
Verilog HDL, Part 2

Xuan 'Silvia' Zhang

Washington University in St. Louis

http://classes.engineering.wustl.edu/ese461/

A Verilog Counter

- 50MHz clock
- Count from 0 to 7, reset, then start from 0 again

Recap Verilog Basics

- Identifier
  - [a-z A-Z_][a-z A-Z_$]
- Logic values
  - ‘0’, ‘1’, ‘x’, ‘z’
- Data types
  - reg, wire, tri, …
  - integer, time, event, real
  - array, vector, memory
  - numbers, strings
- Operators

Arrays, Vectors, and Memories

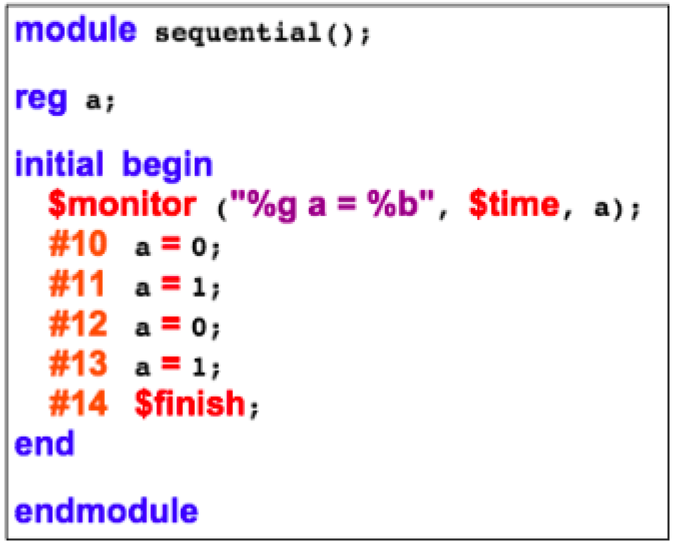
```
//*** Arrays for integer, time, reg, and vectors of reg ****************
integer i[3:0];  //integer array with a length of 4
time    x[20:1]; //time array with length of 19
reg     r[7:0];  //scalar reg array with length of 8

c = r[3]; //the 3rd reg value in array r is assigned to c

//*** Vectors are multi-bit words of type reg or net (wire)************
reg  [7:0] MultiBitWord1;    // 8-bit reg vector with MSB=7 LSB=0
wire [0:7] MultiBitWord2;    // 8-bit wire vector with MSB=0 LSB=7
reg [3:0] bitslice;
reg a;                      // single bit vector often referred to as a scalar
....    //referencing vectors
a = MultiBitWord1[3]; //applies the 3rd bit of MultiBitWord1 to a
bitslice = MultiBitWord1[3:0]; //applies the 3-0 bits of MultiBitWord1 to bitslice


//*** Memories are arrays of vector reg ********************************
reg [7:0] ram[0:4095];      // 4096 memory cells that are 8 bits wide

               //code excerpt from Chapter 2 SRAM model
input [11:0] ABUS;     // 12-bit address bus to access all 4096 memory cells
inout [7:0] DATABUS;   // 8-bit data bus to wite into and out of a memory cell
reg  [7:0] DATABUS_driver;
wire [7:0] DATABUS = DATABUS_driver; //inout must be driven by a wire
....
for (i=0; i < 4095; i = i + 1)  // Setting individual memory cells to 0
  ram[i] = 0;
end
....
ram[ABUS] = DATABUS;  //writing to a memory cell
....
DATABUS_driver =  ram[ABUS]; //reading from a memory cell
```

Recap Verilog Basics

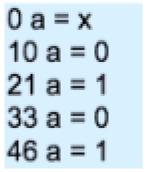
- Hierarchy
  - module interface
  - ports: input, output, inout
- Procedure
  - always or initial statement
  - task
  - function
- Task
  - called from another procedure
  - has inputs and outputs; no return value
- Function
  - procedure used in any expression
  - at least one input, no output, and a single return value

Procedural Blocks

- Block contains more than one statement
- Sequential block
  - between begin and end
  - considered a statement; can be nested
  - appear in always statement; execute repeatedly
  - appear in initial statement; execute once
  - evaluate sequentially



```
module sequential();

reg a;

initial begin
  $monitor ("%g a = %b", $time, a);
  #10  a = 0;
  #11  a = 1;
  #12  a = 0;
  #13  a = 1;
  #14  $finish;
end

endmodule
```



```
0 a = x
10 a = 0
21 a = 1
33 a = 0
46 a = 1
```

Blocking and Non-blocking Statements

- Blocking Statement
  - use = operator
  - sequential code, execute when called
- Non-blocking Statement
  - use <= operator
  - same time execution

```
// Section 1: Blocking statements execute sequentially
#5 a = b;  // waits 5 time units, evaluates and applies value to a
   c = d;  // evaluates and applies value to c

// Section 2: Non-Blocking statements execute concurrently
#5 a <= b; // waits 5 time units, evaluates, schedules apply for end of current time
   c <= d; // evaluate, schedules apply for end of current time
           // At end of current time both a and c receive their values
```

```
#5 x = 1'b0;    // blocks for 5 time units, applies value to x, then next line goes
y = 1'b1;       // blocks, sets y to 1 now, then next statement goes
y <=  #3 1'b0; // evaluates now, schedules apply y=0 in 3 time units, and next line goes
#5 x <= y;      // waits for 5 time units, evaluates,
                 //  schedules apply at end of current time, and next line goes
```

Blocking and Non-blocking Statements

```
module delay;
reg a,b,c,d,e,f,g,bds,bsd;
initial begin
a = 1; b = 0; // No delay control.
#1 b = 1; // Delayed assignment.
c = #1 1; // Intra-assignment delay.
#1; // Delay control.
d = 1; //
e <= #1 1; // Intra-assignment delay, nonblocking assignment
#1 f <= 1; // Delayed nonblocking assignment.
g <= 1; // Nonblocking assignment.
end
initial begin #1 bds = b; end // Delay then sample (ds).
initial begin bsd = #1 b; end // Sample then delay (sd).
initial begin ("t a b c d e f g bds bsd");
("%g",,,a,,b,,c,,d,,e,,f,,g,,bds,,,,bsd); end
endmodule
```

Procedural Groups

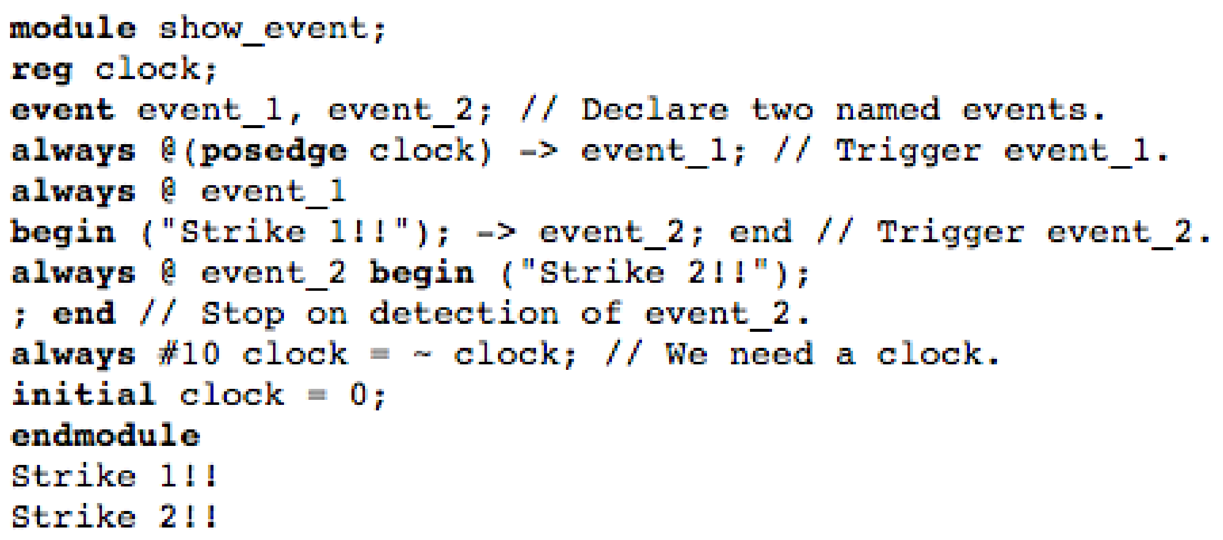
- Parallel block
  - "fork join"
  - evaluated in parallel

```
module parallel();

reg a;

initial
fork
  $monitor ("%g a = %b", $time, a);
  #10  a = 0;
  #11  a = 1;
  #12  a = 0;
  #13  a = 1;
  #14  $finish;
join

endmodule
```

```
0 a = x
10 a = 0
11 a = 1
12 a = 0
13 a = 1
```

Verilog Timing

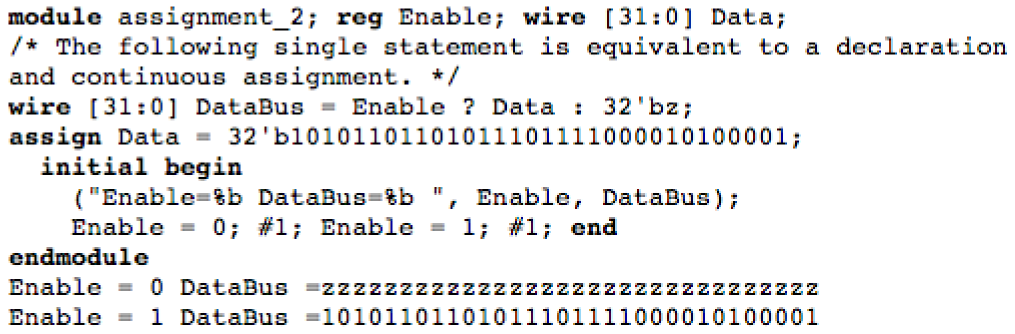
- Timing control
  - delay control: delays an assignment by #amount
  - e.g.: x = #1 y; vs #1 x = y;
  - event control: delay until a specified event occurs
  - e.g. @posedge Clk; @negedge
  - events can be declared, triggered, detected



```
module show_event;
reg clock;
event event_1, event_2; // Declare two named events.
always @(posedge clock) -> event_1; // Trigger event_1.
always @ event_1
begin ("Strike 1!!"); -> event_2; end // Trigger event_2.
always @ event_2 begin ("Strike 2!!");
; end // Stop on detection of event_2.
always #10 clock = ~ clock; // We need a clock.
initial clock = 0;
endmodule
Strike 1!!
Strike 2!!
```

Procedure in More Depth

- Continuous assignment
  - appear outside procedure
  - assign a value to a wire



```
module assignment_2; reg Enable; wire [31:0] Data;
/* The following single statement is equivalent to a declaration
and continuous assignment. */
wire [31:0] DataBus = Enable ? Data : 32'bz;
assign Data = 32'b10101101101011101111000010100001;
  initial begin
    ("Enable=%b DataBus=%b ", Enable, DataBus);
    Enable = 0; #1; Enable = 1; #1; end
endmodule
Enable = 0 DataBus =zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
Enable = 1 DataBus =10101101101011101111000010100001
```

Continuous assignment

```
wire d;
and a1(d, a, b);
not n1(c, d);
```

```
assign c = !(a && b); //notice that wire d was not used here
```

```
wire d;
assign d = a || b; //continuous assignment
```

```
wire c;
assign #5 c = a && b; //delay in the continuous assignment

wire #5 c = a && b;   //delay in the implicit assignment

wire #5 c;  //delay in the wire declaration
assign c = a && b;
```

Procedure in More Depth

- Procedural assignment

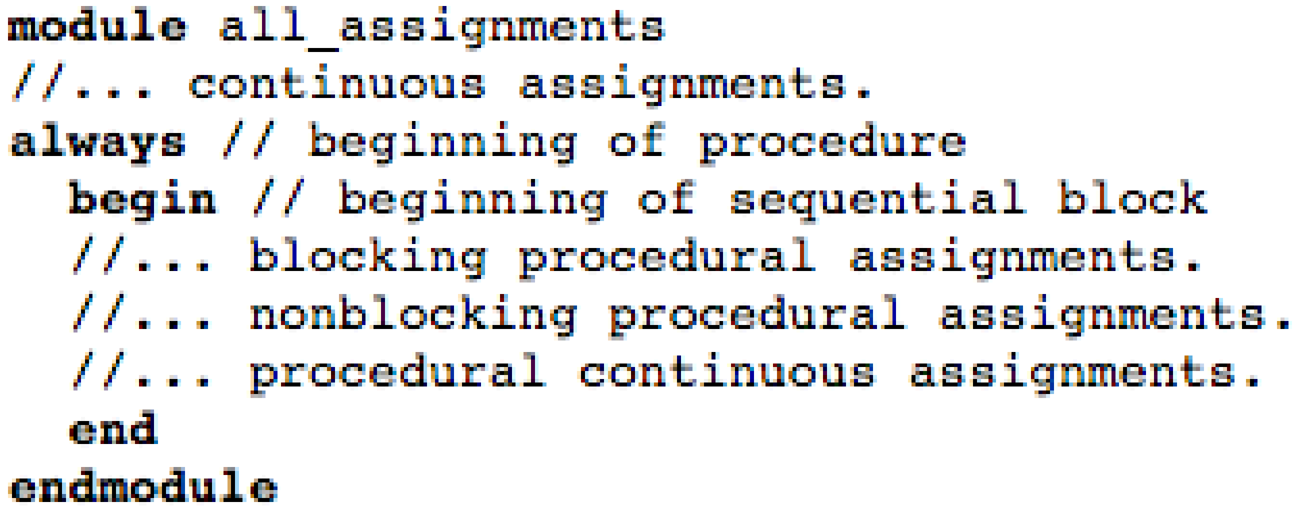
```
module procedural_assign; reg Y, A;
always @(A)
  Y = A; // Procedural assignment.
initial begin A=0; #5; A=1; #5; A=0; #5; ; end
initial ("T=%2g",,,"A=",A,,,"Y=",Y);
endmodule
T= 0 A=0 Y=0
T= 5 A=1 Y=1
T=10 A=0 Y=0
```

- Procedural continuous assignment
  - assign statement inside a sequential block

```
always @(Enable)
if(Enable) assign Q
= D;
else deassign Q;
```

Procedure in More Depth

- In summary



```
module all_assignments
//... continuous assignments.
always // beginning of procedure
  begin // beginning of sequential block
  //... blocking procedural assignments.
  //... nonblocking procedural assignments.
  //... procedural continuous assignments.
  end
endmodule
```

User-Defined Primitives (UDP)

- UDP truth table specification
- First port must be the only output port
- Only specify ‘0’, ‘1’, and ‘x’

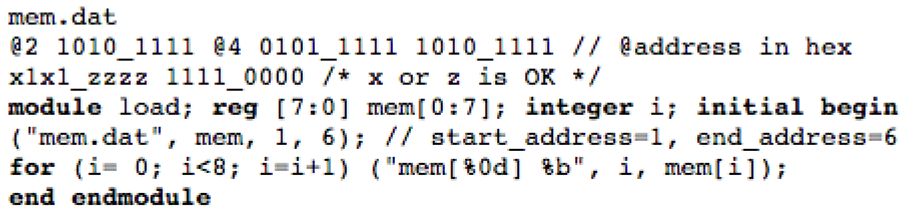
```
primitive DLatch(Q, Clock, Data);
output Q; reg Q; input Clock, Data;
table
//inputs : present state : output (next state)
1 0 : ? : 0; // ? represents 0,1, or x (input or present state).
1 1 : b : 1; // b represents 0 or 1 (input or present state).
1 1 : x : 1; // Could have combined this with previous line.
0 ? : ? : -; // - represents no change in an output.
endtable
endprimitive
```

- `*` is `(??)`
- `r` is `(01)`
- `f` is `(10)`
- `p` is `(01)`, `(0x)`, or `(x1)`
- `n` is `(10)`, `(1x)`, or `(x0)`

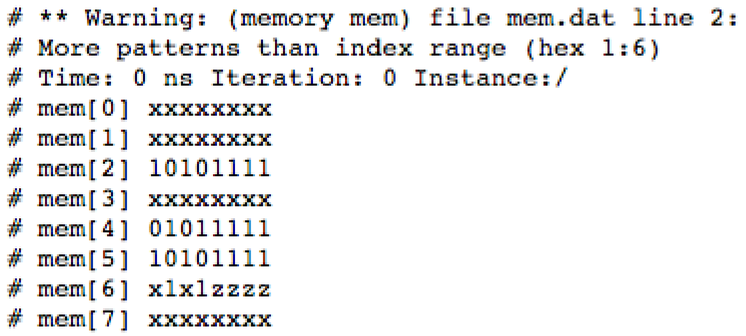
```
primitive DFlipFlop(Q, Clock, Data);
output Q; reg Q; input Clock, Data;
table
//inputs : present state : output (next state)
r    0 : ? : 0 ; // rising edge, next state = output = 0
r    1 : ? : 1 ; // rising edge, next state = output = 1
(0x) 0 : 0 : 0 ; // rising edge, next state = output = 0
(0x) 1 : 1 : 1 ; // rising edge, next state = output = 1
(?0) ? : ? : - ; // falling edge, no change in output
? (??) : ? : - ; // no clock edge, no change in output
endtable
endprimitive
```

Useful Verilog Features

- Display tasks
  - $display, $displayb (h, o) in binary, hex, and octal
  - $write, $strobe, $monitor
- File I/O tasks
  - $fopen, $fclose
  - $fdisplay, $fwrite, $fstrobe, $fmonitor
  - $readmemb, $readmemh: read a text file into memory



```
mem.dat
@2 1010_1111 @4 0101_1111 1010_1111 // @address in hex
x1x1_zzzz 1111_0000 /* x or z is OK */
module load; reg [7:0] mem[0:7]; integer i; initial begin
("mem.dat", mem, 1, 6); // start_address=1, end_address=6
for (i= 0; i<8; i=i+1) ("mem[%0d] %b", i, mem[i]);
end endmodule
```



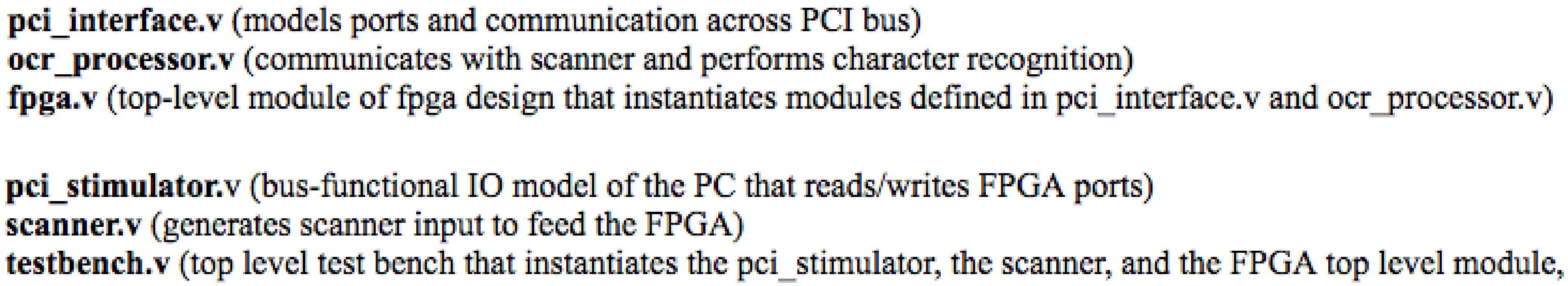
```
# ** Warning: (memory mem) file mem.dat line 2:
# More patterns than index range (hex 1:6)
# Time: 0 ns Iteration: 0 Instance:/
# mem[0] xxxxxxxx
# mem[1] xxxxxxxx
# mem[2] 10101111
# mem[3] xxxxxxxx
# mem[4] 01011111
# mem[5] 10101111
# mem[6] x1x1zzzz
# mem[7] xxxxxxxx
```

Useful Verilog Features

- Timing tasks
  - `` `timescale 10 ns / 1 fs ``
  - $printtimescale, $timeformat
  - $stop, $finish
  - timing-check tasks
- Simulation time functions
  - $time, $stime, $realtime
- Probability distribution functions
  - $random
  - $dist_uniform, $dist_normal, etc.

Putting It Together

- Verilog design flow (FPGA)
  - step 1: create RTL models and behavioral test bench



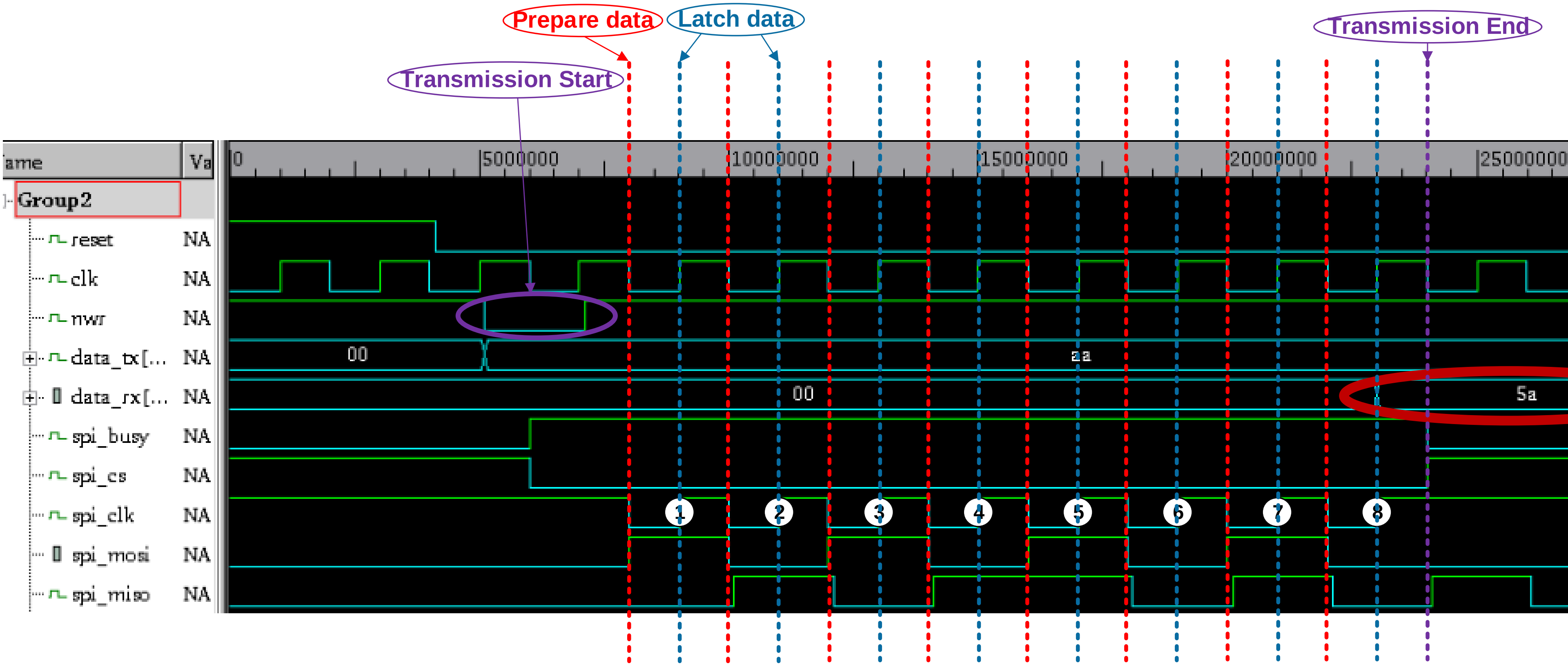
pci_interface.v (models ports and communication across PCI bus)
ocr_processor.v (communicates with scanner and performs character recognition)
fpga.v (top-level module of fpga design that instantiates modules defined in pci_interface.v and ocr_processor.v)

pci_stimulator.v (bus-functional IO model of the PC that reads/writes FPGA ports)
scanner.v (generates scanner input to feed the FPGA)
testbench.v (top level test bench that instantiates the pci_stimulator, the scanner, and the FPGA top level module,

  - step 2: functionally simulate the RTL design
  - step 3: convert RTL-level to gate-level with synthesizer
  - step 4: perform gate-level simulation
  - optional: perform gate-level simulation with SDF timing

Lab #2: Verilog Basics

- Due 10/3 (Monday)
- Serial Peripheral Interface (SPI)



Reference

- Chapter 11, ASIC Book
- http://www.asic-world.com/verilog/veritut.html
- http://www.verilogtutorial.info/
- "Verilog HDL" by Samir Plantikar
- https://docs.numato.com/kbcategory/getting-started-with-fpga/

Questions?

Comments?

Discussion?