Cloud FPGA

bit.ly/cloudfpga
This lecture is mostly based on contents of Chapter 7, from “The Verilog Hardware Description Language” book [1], 5th edition. Example figures and (modified) code are from the textbook unless otherwise specified.

Topics covered:

• Clock events
• Input and output sets of `always` blocks
• Reset function
• Mealy and Moore state machines
• Behavioral synthesis
Cycle-Accurate Specification

- When using cycle-accurate specification, also called scheduled behavior, the system is described using behavioral (always) statements.
- One of the most common ways to describe synchronous digital systems:
  - All the state (registers) get updated on a clock edge.
- Cycle-accurate specification specifies what happens in each clock cycle as the design runs:
  - Actual implementation of the logic is left to the synthesis tools when using behavioral description.
  - Or can be specified using logic-level description.
  - Typically mix both types.
- Cycle-accurate specification basically specifies that state machines that define the operation of the system and inputs and outputs, and computations done by the state machines.
Clock Events and Cycle-Accurate Specification

- Cycle-accurate specification is divided into clock events
  - Each clock event defines a start of a new state
  - Behavioral description after each clock event defines the behavior in that state

Sample state machine and state diagram from textbook:

```
module simpleTutorial
(input clock,
 output reg [7:0] x, y );

reg [7:0] i;

always begin
@ (posedge clock)
  x <= 0;
  i = 0;
  while (i <= 10)
  begin
    @ (posedge clock);
    x <= x + y;
    i = i + 1;
  end
end
@ (posedge clock);
if (x < 0)
y <= 0;
else
  x <= 0;
end
endmodule
```

A clock event

State A

Blocking assignment used for internal registers

State B

At end of always block, “got back” to start of always block

State C

Sample state machine and state diagram from textbook:
**Input and Output Sets of always Blocks**

- **Internal Register Set** – the set of all registers used on the left-hand side of statements in the `always` block, that are not used by other always blocks.

- **Input Set** – the set of all the signals on the right-hand side of statements in the `always` block, that are not in the internal register set.

- **Output Set** – the set of all signals on the left-hand side of statements in the `always` block, that are not in the input set.

```verilog
code
module inOutExample

  input [7:0] r, s, input clock,
  output reg [7:0] qout

endmodule

reg [7:0] q;
always
begin
  @ (posedge clock)
  q <= r + s;

  @ (posedge clock)
  qout <= q + qout;
end
endmodule
```
Cycle-Accurate Behavior

- When working with cycle-accurate specification, all the states and registers are updated on the clock edge
  - The `always` block specifies which edge: `posedge` or `negedge`
  - Can mix both, but typically just use one or the other

```
always begin
  @ (posedge clock)
  q <= r + s;
@ (posedge clock)
  qout <= q + qout;
end
```

Good practice to update each output set register on each clock, when no change is needed just do `qout <= qout`
Maintaining Cycle-Accurate Specification

- Different realizations of state machines are possible based on same cycle-accurate specification code
  - Synthesis tools can attempt to optimize the design while maintaining the same behavior
  - Level of possible optimization depends on the tools and the “effort” level for the tool

```
module sampleModule
(input [7:0] i, j, k,
    output reg [7:0] f, h );

reg [7:0] g, q, r, s;
always begin
    // ...
    @ (posedge clock);
    f <= i + j;
    g = j * k;
    @ (posedge clock);
    h <= f + k;
    @ (posedge clock);
    f <= f * g;
    q = r * s;
    // ...
end
endmodule
```

Possible to reduce design size by using only one multiplier in each cycle
Reset Function

- A reset function can be used to initialize the state into a known state
  - Can be achieved with reset logic in an `initial` statement, only used for simulation
  - Can be achieved with `posedge` or `negedge` of reset signal in `always` blocks

```verilog
module accumulate
(output reg [11:0] qout,
 input [11:0] r, s,
 input clock, reset)
);

reg [11:0] q;

initial forever
@negedge reset
begin
  disable main;
  qout <= 0;
end

always begin : main
  wait (reset);
  @posedge clock
  q <= r + s;
  @posedge clock
  qout <= q + qout;
end
endmodule
```
Mealy and Moore State Machines

- Mealy state machine – outputs depend on both current state and the inputs
- Moore state machine – outputs depend only on the current state

- Mealy vs. Moore:
  - For Moore machines, registers in output set are not updated with input set registers, but only depend on internal register set
  - For Mealy machines, there are usually fewer states, but logic may be more complex

Moore machine for a FIR filter

Mealy machine for a FIR filter
Mealy and Moore State Machines

Textbook examples of FIR filters as Moore and Mealy machines:

```
module firFiltMealy
    input clock, reset,
    input [7:0] x,
    output reg [7:0] y ;

    reg [7:0] coef_array [7:0];
    reg [7:0] x_array [7:0];
    reg [7:0] acc;
    reg [2:0] index, start_pos;

    initial forever
        @(posedge reset)
        begin
            disable firmain;
            start_pos = 0;
        end

    always begin firmain
        wait (reset);
        @(posedge clock); // State A;
        x_array[start_pos] = x;
        acc = x * coef_array[start_pos];
        index = start_pos + 1;
    begin loop1
        forever
            @(posedge clock); // State B;
            acc = acc + x_array[index] * coef_array[index];
            index = index + 1;
            if (index == start_pos) disable loop1;
        end
        // loop1
        y <= acc;
        start_pos = start_pos + 1;
    end
endmodule
```

```
• When designing for FPGAs, the behavioral code is synthesized into the hardware using vendor-specific tools
  • Tools have to determine how to generate hardware logic based on the behavioral description
  • Use different standard modules, e.g. multiplier, to implement user’s logic
  • Can have different optimizations and optimization effort levels
  • Target area, power, or performance
References