Digital Design & Computer Arch.
Lab 4 Supplement:
Finite-State Machines

(Presentation by Aaron Zeller)
Frank K. Gürkaynak
Seyyedmohammad Sadrosadati
ETH Zurich
Spring 2024
[09. April 2024]
What Will We Learn?

- In Lab 4, you will implement a finite-state machine using Verilog.
- Design and implement a simple circuit that emulates the blinking lights of a Ford Thunderbird.
- Understand how the clock signal is derived in the FPGA board.
- Write an FSM that implements the Ford Thunderbird blinking sequence.
Tail Lights of a 1965 Ford Thunderbird

In this lab, you will design a finite-state machine to control the tail lights of a 1965 Ford Thunderbird.
Tail Lights of a 1965 Ford Thunderbird

- There are three lights on each side that operate in sequence to indicate the direction of a turn.

Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e

Copyright from ClassicLEDs.com
Recall: Finite State Machines (FSMs)

- Each FSM consists of three parts:
  - next state logic
  - state register
  - output logic

At the beginning of the clock cycle, next state is latched into the state register.
Part 1: FSM Design

- An FSM must do three things:
  - **Next State Logic:** Determine the next state from the present state and the inputs.
  - **Output Logic:** Determine the output signals based on the present state and input signals.
  - **State Register:** keeps track of the present state; must be updated at every clock cycle.

The manual contains the details of this FSM specifications.
Part 1: FSM Design

- An FSM must do three things:
  - **Next State Logic**: Determine the next state from the present state and the inputs.
  - **Output Logic**: Determine the output signals based on the present state and input signals.
  - **State Register**: keeps track of the present state; must be updated at every clock cycle.

For more details, please refer to Lecture 7:
- Slides 26+: Finite State Machines
- Slides 28+: Moore vs. Mealy FSMs
always blocks are used in **sequential circuits** and are used to create circuits involving flipflops.

**Sensitivity List:** Whenever a signal in the list changes the body of the always block is executed.

```verilog
always @ (posedge clk, ...)
a <= b;
...
```
Part 2: Verilog Implementation

- **Do not mix** assign statements and always blocks.

- **assign** is used to connect **wires**.

- **<=** and **=** inside **always blocks** are meant to set **registers** to some value.
  - **=** (blocking) : assignments are executed in sequential order.
  - **<=** (non-blocking)
  - Do not mix **<=** and **=** in the same always block.
Part 2: Verilog Implementation

- Do **not** assign the same register in two different always blocks.

```verilog
always @(*)
a = b;
always @(*)
a = c;
```

This should be avoided
Part 2: Verilog Implementation

- Separate three parts of the code:
The **next state logic** determines how to transition from the **current state** to the **next state**.

- **Current state**: state_p
- **Next state**: state_n

```verilog
// FSM to determine next state depending on input
always @ (posedge inputs)
begin

    // Add next state logic here

end
```

This is the general idea and not taken from a solution.
Part 2: Verilog Implementation

- Separate three parts of the code:
Verilog Implementation: Flip-Flops

- We use a rising-clock-edge triggered flip-flop (with reset) as a state register.

- In Verilog we use an always block with the clock signal and the reset signal in the sensitivity list.

- The rising-clock-edge is given to us in Verilog by posedge.
Verilog Implementation: Flip-Flops

```
//State update
always @ (posedge CLK, posedge RESET)
begin
  if (RESET) state_p <= [ ];
  else state_p <= [ ];
end
```

This is the general idea and not taken from a solution.
Part 2: Verilog Implementation

- Separate three parts of the code:
This is the general idea and not taken from a solution
Part 3: Implementing the Clock (I)

The problem of using push-buttons as clock:

- Compared to the speed of the FPGA the change in a push button is very slow (~1 million times slower)

- During the slow transition, the FPGA will see many fast occurring transitions and would interpret each of them as a clock edge. (Bouncing)
Part 3: Implementing the Clock (II)

- **CLK100Mhz (W5):** Your board contains a 100Mhz crystal oscillator circuit.
- **Problem:** The clock is too fast.
- **Solution:** A clock divider:

```verilog
module clk_div(input clk, input rst, output clk_en);
    reg [24:0] clk_count;
    always @ (posedge clk)
        //posedge defines a rising edge (transition from 0 to 1)
        begin
            if (rst)
                clk_count <= 0;
            else
                clk_count <= clk_count + 1;
        end
    assign clk_en = &clk_count;
endmodule
```
Verilog Implementation: Clock Design

- `&clk_count` evaluates to 1 exactly when all bits of `clk_count` are 1 and 0 otherwise.

- This is the case when we counted from 0 to $2^{25} - 1$.
  - $2^{25} - 1 = 25b'11\ldots1$
  - Overflow handles the transition from $25b'1\ldots1$ to $25b'0\ldots0$.

- The slowed clock hence ticks every $\sim2^{25}$ clock ticks.

```verilog
module clk_div(input clk, input rst, output clk_en);
reg [24:0] clk_count;
always @ (posedge clk)
//posedge defines a rising edge (transition from 0 to 1)
begin
  if (rst)
    clk_count <= 0;
  else
    clk_count <= clk_count + 1;
end
assign clk_en = &clk_count;
endmodule
```
Part 4: Defining the Constraints

- We must specify constraints for:
  - Buttons for control
  - LEDs for output lights
  - Connections for clock

The manual contains more information about the constraints.
In Lab 4, you will implement a finite-state machine using Verilog.

Design and implement a simple circuit that emulates the blinking lights of a Ford Thunderbird.

Understand how the clock signal is derived in the FPGA board.

Write an FSM that implements the Ford Thunderbird blinking sequence.

In the report you will implement a dimming function, so that the lights are not only on and off, but can have intermediate levels.
[26. April 2024 23:59]
Digital Design & Computer Arch.
Lab 4 Supplement:
Finite-State Machines

Frank K. Gürkaynak
Seyyedmohammad Sadrosadati
ETH Zurich
Spring 2024
[09. April 2024]