HDL, RTL and FPGA: Lab 2

Review of your next steps in designing digital hardware

Yuri Panchul, Senior Hardware Design Engineer, MIPS
Lecture for Innopolis University - 2018-02-01
Refreshing knowledge from Lab 1
Hardware/software dualism - an informal example

Microcontroller (embedded chip, ASIC, SoC)

CPU
Designed in Verilog
Executes instructions

Memory
Contains a program, a sequence of instructions
Compiled from C or a similar language

18800005
00001025
00451021
0044182a
5460ffe
00451021
03e00008
00000000
Software: from C to processor instructions

C:

```c
int f (int a, int b)
{
    int s = 0;
    while (s < a)
        s += b;
    return s;
}
```

Assembly:

```assembly
.set noreorder
sum:
    blez    $4, exit
    move    $2, $0
    addu    $2, $2, $5
loop:
    slt     $3, $2, $4
    bnel    $3, $0, loop
    addu    $2, $2, $5
exit:
    jr      $31
    nop
```

Machine code

```
18800005
00001025
00451021
0044182a
5460ffe
00451021
03e00008
00000000
```
module counter
(
    input clock,
    input reset,
    output logic [1:0] n
);
always @(posedge clock)
begin
    if (reset)
        n <= 0;
    else
        n <= n + 1;
end
endmodule
What is an FPGA? A simplified explanation

A matrix of cells with changeable function

One cell can become AND, another OR, yet another - one bit of memory

An FPGA does not contain a fixed CPU, but can be configured to work as a CPU

A picture from http://jjmk.dk/MMMI/PLDs/FPGA/fpga.htm
From combinational to sequential logic
From Lab 1: Combinational logic

- The outputs of the group of components depend only on inputs
- You set inputs and get outputs after some time
- This group is called "a combinational cloud"
- Used to calculate logic and arithmetic functions

```
module top
(
    input a, b, c, d,
    output e, f
);

assign e = a & (b | c);
assign f = ~ (c ^ d);

endmodule
```
Computation in combinational logic is not instant

- Before the results are ready, the outputs may contain random values.
- How to find when the results are ready and can be used by the next step of computation?
- We can synchronize the computation with a special signal called clock.

The picture is from Digital Design and Computer Architecture, 2nd Edition by David Harris and Sarah Harris. Elsevier, 2012
Clock is a periodic signal with square waveform

- This period is long enough for any combinational computation to complete.
- Clock frequency = \( \frac{1}{\text{period}} \).
- Clock is usually generated by a crystal oscillator (find it on the board).
A circuit synchronized with a clock is called sequential

- The sequential circuit go through a sequence of states.
- The current state is stored in D-flip-flops.
- A new state is computed based on the previous state and the circuit’s inputs.
- A new state is recorded into D-flip-flops when clock goes from low to high.
- The outputs are computed based on the inputs and the current state
Huffman model of sequential circuits

Inputs → Combinational Cloud → Outputs

State Registers

Clock
What the sequential logic allows us to do

- Counting
- Memorizing the information
- Adding new data and repeating the computation
- Waiting for an event coming from outside the device
- In short: sequential logic is what makes a computer to do interesting things

A picture of John Searle’s Chinese room thought experiment is from http://deskarati.com/2014/07/01/john-searles-chinese-room-thought-experiment
The Exercises
Lab 2 exercises

1. The function of a D-flip-flop, the basic brick of a sequential design
2. Counter, a combination of combinational and sequential
3. Connecting flip-flops back-to-back to make a shift register
4. Finite State Machine (FSM) for sequence recognition
5. The application of FSM for interfacing sensors
6. The concept of pipelining
7. Looking forward schoolMIPS and MIPSfpga
D-flip-flop records the data at the end of clock cycle

- “Always at positive edge of clk store the value of signal d in a D-flip-flop inferred by variable q. q is also connected to the output”.

- `always` block is similar to the `initial` block, however it is evaluated on every event described after `@`.

- The assignment `<=` is called non-blocking, it will take effect after all current `always` blocks get evaluated

```verilog
classmodule d_flip_flop
(
    input clk,
    input d,
    output reg q
);

always @ (posedge clk)
    q <= d;
```

Reset signal guarantees the initial state

- This design uses reset active low (“negedge rst_n”), some others use reset active high.
- Reset is necessary for control signals, so the device does not act erratically on power-up.

```verilog
module dff_async_rst_n
(
    input clk,
    input rst_n,
    input d,
    output reg q
);

always @ (posedge clk or negedge rst_n)
    if (!rst_n)
        q <= 0;
    else
        q <= d;
```

Source: https://github.com/MIPSfpga/digital-design-lab-manual/blob/master/lab_02/src/06_dff_with_async_rst_n/dff_async_rst_n.v
A group of D-flip-flops is called a register.

- Do not confuse with Verilog `reg` or CPU registers in programming.

- We can parameterize the number of D-flip-flops in a register using Verilog `parameter` declaration.

```verilog
module dff_async_rst_n_param
#

    parameter WIDTH = 8,
    RESET = 8'd0

(

    input             clk,
    input             rst_n,
    input             [WIDTH - 1 : 0] d,
    output reg        [WIDTH - 1 : 0] q

);
```

Register keeps the value during the cycle
Two registers back-to-back delay for two cycles
Compare
Generating and using clock in a testbench

You can also use clock to schedule assigning stimuli, the values for the DUT ports (DUT = Design under Test).

```verilog
initial
begin
  clk_en <= 1'b1;
  arg_vld <= 1'b0;
  repeat (2) @ (posedge clk);
  rst_n <= 0;
  repeat (2) @ (posedge clk);
  rst_n <= 1;
end
```

Source: https://github.com/MIPSfpga/digital-design-lab-manual/blob/master/lab_10/src/lab_10_1_pow_5/01_sim_pow_5/testbench.v
Adder + Register = Counter

- A group of D-flip-flops is called a register.
- Do not confuse with Verilog `reg` or CPU registers in programming.
- In this code the result of addition is stored to use in the next clock cycle.

```verilog
module counter
(
    input    clock,
    input    reset_n,
    output   reg [31:0] count
);

always @(posedge clock or negedge reset_n)
begin
    if (!reset_n)
        count <= 32'b0;
    else
        count <= count + 32'b1;
end
```

Source: https://github.com/yuri-panchul/2017-tomsk-novosibirsk-astana/blob/master/pre_event_drafts/terasic_de10_lite/08_counter/top.v
The register contains 0, it gets propagated to the adder.
0 enters the adder. The adder's output is not stable yet.
The adder computed $0 + 1 = 1$. Register still contains 0.
Positive edge of the clock is coming. Register is still 0.
The aperture time. Register is about to store 1.
The register recorded 1 and is about to propagate it outside.
The current state is 1, it gets propagated to the adder.
1 enters the adder. The adder’s output is not stable yet.
The adder computed $1 + 1 = 2$. The register still contains 1.
A positive edge of the clock is coming. The register is still 1.
The aperture time. The register is about to store 2.
The register recorded 2 and is about to propagate it outside.
The current state is 2, it gets propagated to the adder.
Finite State Machine, the decision maker

- Let’s implement an example of FSM that recognizes sequences.
- We got this example from Digital Design and Computer Architecture by David Harris and Sarah Harris, 2012.
- “A snail crawls down a paper tape with 1’s and 0’s on it. The snail smiles whenever the last two digits it has crawled over are 01. Design a state machine of the snail’s brain.”
- An FSM is a special case of a Huffman sequential circuit.
- Mealy FSM uses inputs directly to compute outputs, Moore’s FSM does not.
FSM designers use state transition diagrams

- Circles designate states.
- Arcs designate transitions depending on inputs.
- For Mealy FSM arcs indicate inputs / outputs.
Coding FSMs in Verilog - State register

```verilog
module pattern_fsm_moore
(
    input clock,
    input reset_n,
    input enable,
    input a,
    output y
);

parameter [1:0] S0 = 0, S1 = 1, S2 = 2;
reg [1:0] state, next_state;

// State register
always @ (posedge clock or negedge reset_n)
    if (! reset_n)
        state <= S0;
    else if (enable)
        state <= next_state;
```
// Next state logic
always @*
  begin
    case (state)
      S0: if (a) next_state = S0; else next_state = S1;
      S1: if (a) next_state = S2; else next_state = S1;
      S2: if (a) next_state = S0; else next_state = S1;
      default: next_state = S0;
    endcase
  end

// Output logic based on current state
assign y = (state == S2);
Examples: interfaces to sensors

- Ultrasonic distance sensor
  - https://github.com/yuri-panchul/2017-year-end/blob/master/terasic_de10_lite/hc_sr04_receiver.v

- Digilent Ambient Light Sensor with SPI protocol
  - https://github.com/yuri-panchul/2017-tomsk-novosibirsk-astana/blob/master/parts_and_examples/pmod_als_spi_receiver/pmod_als_spi_receiver.v
Synthesis tools recognize FSMs and optimize them.
The concept of pipelining - 1

- Suppose we have a computation that consists of multiple steps.
- We can compute a new set of data each clock cycle.
- The clock has a period of $t_{\text{FGH}}$. 
The concept of pipelining - 2

- We add registers between the steps.
- It reduces the clock period from $t_F + t_G + t_H$ to $\max(t_F, t_G, t_H)$.
- Now we can put a new set of inputs before the results for the previous sets went all the way $F \rightarrow G \rightarrow H$.
- The throughput of the module increases.
The concept of pipelining - 3

- An application of the pipelining principle for computing $F(n) = n^5$.
- We put a new input each cycle and get the result 4 cycles later.

Source: https://github.com/MIPSfpga/digital-design-lab-manual/tree/master/lab_10/src/lab_10_1_pow_5
CPU pipeline, best-known example of the pipelining principle

The execution unit of MIPS M5150 CPU core processes the stream of instructions.
Learn about CPUs using schoolMIPS and MIPSfpga

- schoolMIPS is as simple RISC CPU as you can get, use it to learn the basics.
- MIPSfpga is to experiment with an industrial core, it uses a variant of MIPS M5150 from the previous slide.
Thank You!