FEUP
Universidade do Porto
Faculdade de Engenharia

Departamento de Engenharia Eletrotécnica e de Computadores

EEC0055 – Digital Systems Design

4º year - 1º semester
Exam – 14 January 2021

Duration: 1h30m, closed book.

[4 points]

1 - Consider the methodology of digital system design for integrated microelectronic technologies that was studied in PSDi. A fundamental task that must be carried out at different design stages is the verification of the models of the system, using logic simulation.

a) For each of the 3 main simulation steps (functional, post-synthesis and post-place & route) explain briefly what are the main differences between the models that are simulated and identify the types of design errors that are intended to be identified with each of the 3 simulation processes.

The functional simulation simulates the source HDL models (in our design environment, written in Verilog HDL) that specify the behavior of the digital system under design, at the register-transfer level (RTL). This verification stage intends to validate the RTL models and detect possible functional errors not compliant with the design specification [or when the HDL constructs written by the designer do not define the logic functionality].

The post-synthesis simulation verifies the netlist created by the RTL synthesis process, formed by an interconnection of the primitive logic blocks available in the target technology. This verification stage can detect design errors that can make the netlist not equivalent to the source RTL model. This may result from incorrect coding in Verilog, as for example using HDL constructs that can be synthesized to a logic circuit the is not functionally equivalent to the source HDL code (or statement that are just ignored by the synthesis tool, like explicit delay statements).

The post-place&route simulation verifies the netlist built after place&route (or physical synthesis). This netlist is now annotated with detailed timing information of the logic blocks and the interconnections among them. This simulation stage intends to detect design faults due to incorrect timing, as for example, violations of the setup or hold times of the flip-flops or multiple trigger of flip-flops or transparent latches by signals with glitching activity.

b) Consider the design of a synchronous digital system with a single clock signal. Explain under which conditions (based on the analysis of results provided by earlier design stages) the last verification process can be avoided without compromising the guarantee of the correctness of the design.

This simulation step can be skipped if three conditions are met simultaneously:
1. *All the previous verification stages have been successfully completed.*
2. *The design behavior does not depend on external input signals that are not synchronized with the clock signal and that have not been timing constrained for guiding the P&R process.*
3. *After place&route the static timing analysis reports a maximum clock frequency [for the unique clock signal] above the intended clock frequency.*

[2 points]

2 - Clock gating is a fundamental technique for reducing power consumption in digital CMOS circuits, by turning off the clock signal when it is not needed. Explain why this should not be implemented by synthesizing the following Verilog code:

```
assign clk_gate = clock & en_clock;
always @(posedge clk_gate)
    reg <= new_reg;
```

*Turning on and off the clock activity [or gating the clock] should not be done with the circuit specified by this code, because the insertion of combinational logic in the clock path [the "AND" gate) will delay the gated clock signal and, assuming that the clock enable signal **en_clock** is a control signal synchronized to the same clock, the gated clock **clk_gate** may exhibit glitching activity and thus provoke incorrect loads of the destination register.*

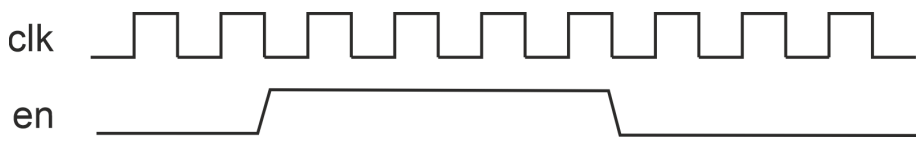
[4 points]

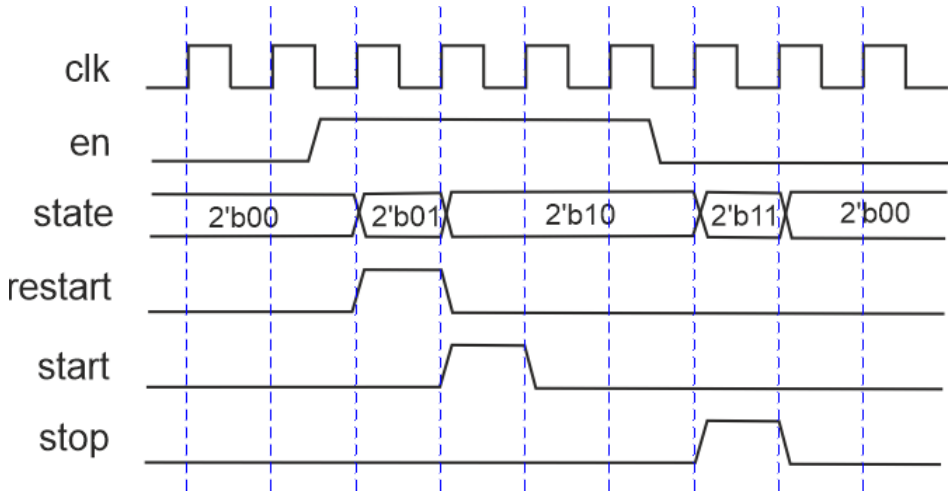
3 - The following Verilog code implements a finite state machine:

```
module fsm( input reset, input clk, input en,
            output reg restart, output reg start, output reg stop );
reg [1:0] state;

always @(posedge clk)
if (reset)
  ... // set all registers to zero
else
begin
    restart <= 1'b0; start <= 1'b0; stop  <= 1'b0;
    case ( state )
          2'b00: if ( en )
                 begin
                   restart <= 1; state <= 2'b01;
                 end
          2'b01: begin
                   start <= 1; state <= 2'b10;
                 end
         2'b10: if ( ~en )
                begin
                  stop <= 1; state <= 2'b11;
                end
         2'b11: begin
                  state <= 2'b00;
                end
    endcase
end
endmodule
```

a) Represent a timing diagram along 9 consecutive clock cycles showing the sequence of values of register **state** and the 3 outputs, when the input en is equal to zero in the first two clock transitions, is set to 1 for the next 4 transitions and returns to zero in the remaining clock transitions. Assume that in the first clock transition, all registers have a value of zero.





***Note**: this is the timing diagram that answers the question. Any other representation that do not show clearly the relationship of the signals transitions to the clock active edge (as for example a table view with the logic values in each clock cycle), is not acceptable.*

b) This state machine is the controller of a synchronous datapath that uses the same global clock signal. Although this circuit implements correctly the intended functionality for the state machine, this RTL model contains unnecessary logic and can thus be improved in order to reduce the complexity of the logic circuit. Explain how and illustrate your answer by presenting the Verilog code of the improved version of this finite state machine.

*This code implements all the outputs as registers, which is not necessary if, as said, these outputs serve as control signals for a datapath using the same clock signal. To reduce the logic complexity the outputs can be coded as combinational, saving flip-flops. From the timing diagram above, we can see that **restart** is 1 only when **state** is 2'b01, start is 1 only during the first clock cycle when **state** is 2'b10 and stop is 1 when **state** is 2'b11. Thus, outputs **restart** and **stop** only depend on the current state but **start** still needs to be implemented with an additional register.*

*A simpler RTL model only implements the **state** register and the **start** output in the clocked synchronous **always** process and the other two outputs are combinational, defined with **assign** statements.*

```
module fsm( input reset, input clk, input en,
            output restart, output reg start, output stop );
reg [1:0] state;
always @(posedge clk)
if (reset)
  begin  state <= 2'b00;start <= 1'b0;  end
else
begin
     start <= 1'b0;
     case ( state )
           2'b00: if ( en )
                  begin
                    state <= 2'b01;
                  end
           2'b01: begin
                    start <= 1'b1; state <= 2'b10;
                  end
          2'b10: if ( ~en )
                 begin
                   state <= 2'b11;
                 end
          2'b11: begin
                    state <= 2'b00;
                 end
     endcase
end
assign restart = (state == 2'b01);
assign stop = (state ==  2'b11);

endmodule
```

[6 points]

4 - The following Verilog module implements a finite impulse response (FIR) filter with 7 constant coefficients. The coefficients are signed numbers represented in two's complement as fixed point values with 1 bit for the integer part and 15 bits for the fractional part. The input and output data (**din** and **dout**) are 16-bit integers signed numbers in two's complement, received and produced at each clock signal period. It is known that for the target technology of this project, and for the dimensions of the data involved, a combinational multiplier has a maximum propagation delay of 8 ns and a combinational adder has a maximum propagation delay of 4ns.

```
1:  module psd2021( input reset,
2:                  input clk,
3:                  input signed  [15:0] din,
4:                  output signed [15:0] dout );
5:  reg signed [15:0] h[0:6];
6:  reg signed [15:0] x[0:6];
7:  reg signed [33:0] yr;
8:  integer i;
9:
10: initial
11: begin
12:    h[0] = 16'hEF6D; h[1] = 16'h0F5C; h[2] = 16'h2F4A; h[3] = 16'h3FC3;
13:    h[6] = 16'hEF6D; h[5] = 16'h0F5C; h[4] = 16'h2F4A;
14: end
15:
16: always @(posedge clk)
17:    if (reset)
18:       for(i=0; i<7; i=i+1) x[i] <= 16'd0;
19:    else
20:    begin
21:       for(i=0; i<6; i=i+1 ) x[i+1] <= x[i];
22:       x[0] <= din;
23:       yr <= h[0]*x[6] + h[1]*x[5] + h[2]*x[4] + h[3]*x[3] +
24:             h[4]*x[2] + h[5]*x[1] + h[6]*din;
25:    end
26: assign dout = yr[33:18];
27: endmodule
```

a) Present and justify an estimated value for the maximum frequency of the clock signal that can be applied to this circuit.

First of all, the minimum clock period is determined by the maximum propagation delay of the combinational logic between registers, but a correct answer must present and justify conveniently the value presented.

This module defines a clocked synchronous datapath where the critical path is the combinational circuit implemented by the expression in lines 23-24. This combinational block is formed by 7 multipliers (16 x 16 bits) and a 7-operand adder for 34-bit operands.

The multipliers introduce a maximum propagation delay of 8ns, but note that the maximum propagation delay is expected to be from the least significant bits of the operands to the most significant bit of the result.

The 7-operand adder may be implemented as a tree of adders or as a cascade of adders (see figure below). In either case, the maximum propagation delay of an adder is also from the least significant bits of the operands to the most significant bit of the result. Thus, the total propagation delay of the 7-operand adder will be significantly smaller than the sum of the propagation delays of each adder (6 x 4 = 24ns for the cascaded adder and 3 x 4 = 12 ns for the tree adder). If the adders are implemented as ripple-carry adders, and not considering the propagation delays of interconnects, each adder level will sum the propagation delay of a single full-adder, that can be estimated as 1/34 of the 4ns total propagation delay for a 34-bit adder. That said, a rough estimation of the propagation delay of the cascaded adder is 4ns + 5 x 1/34 x 4ns = 4.6 ns, and for the adder tree it is 4ns + 2 x 1/34 x 4ns = 4.3ns. Also note that this maximum propagation delay will happen from the least significant bit of the operands (the outputs of the multipliers) to the most significant bit of the result.

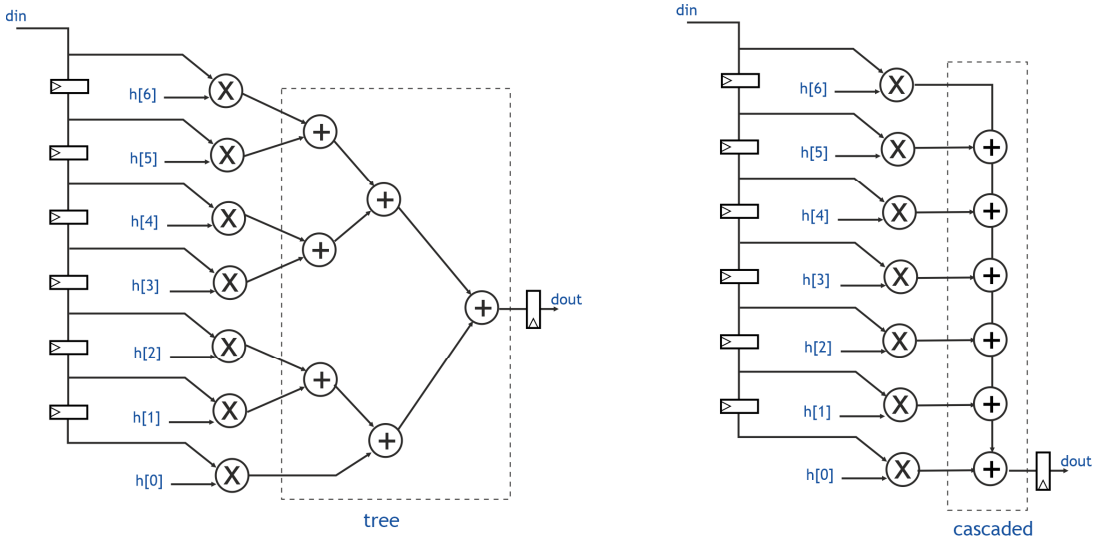
Before the adder circuit we have the multipliers, where the 8ns maximum propagation delay is also from the LSB to the MSB.

Examples of possible answers:

Best: because of the reasons explained above, we can expect a maximum propagation delay below 8ns + 4.6ns, 12.6ns cock period or 79 MHz (this is considering the slower cascaded adder tree implementation; it would be 12.3ns or 81 MHz for the faster adder tree implementation).

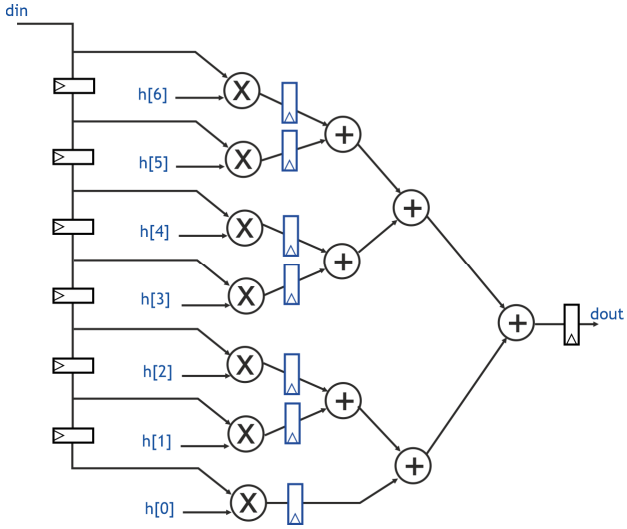
Acceptable, but not so good: the same answer above without the "below"

Wrong, too conservative: maximum delay is 8ns for the mult + 3 x 4ns for the adder tree: clock frequency 50 MHz (or 8ns for the mult + 6 x 4ns for the cascaded adder: clock frequency 31 MHz)



b) In order to increase the clock frequency, it was decided to implement this module as a pipelined circuit. Explain how, presenting the Verilog code that shows the changes to be made in the presented module and a block diagram that illustrates the solution proposed.

To transform the circuit to a pipelined version, the solution is to cut the combinational path that is constraining the clock period into logic slices balanced in propagation delay and insert registers between them. Without "cutting" through the arithmetic operators, and based on the discussion presented in a), the best solution will be using a single pipe stage after the multipliers, because the 7-operand adder (either tree or cascaded) will have a propagation delay expectably smaller than 8ns (the propagation delay of the multiplier). The block diagram and corresponding Verilog code are (new/modified Verilog code is in light blue):



```
1:  module psd2021_pipe( input reset,
2:                       input clk,
3:                       input signed  [15:0] din,
4:                       output signed [15:0] dout );
5:  reg signed [15:0] h[0:6];
6:  reg signed [15:0] x[0:6];
7:  reg signed [33:0] yr;
7.1:reg signed [31:0] m0, m1, m2, m3, m4, m5, m6;
8:  integer i;
9:
10: initial
11: begin
12:    h[0] = 16'hEF6D; h[1] = 16'h0F5C; h[2] = 16'h2F4A; h[3] = 16'h3FC3;
13:    h[6] = 16'hEF6D; h[5] = 16'h0F5C; h[4] = 16'h2F4A;
14: end
15:
16: always @(posedge clk)
17:    if (reset)
17.1:  begin
18:       for(i=0; i<7; i=i+1) x[i] <= 16'd0;
18.1:     m0 <= 0; m1 <= 0; m2 <= 0; m3 <= 0; m4 <= 0; m5 <= 0; m6 <= 0;
18.2:  end
19:    else
20:    begin
21:       for(i=0; i<6; i=i+1 ) x[i+1] <= x[i];
22:       x[0] <= din;
22.1:     m0 <= h[0]*x[6];  m1 <= h[1]*x[5];  m2 <= h[2]*x[4];
22.2:     m3 <= h[3]*x[3];  m4 <= h[4]*x[2];  m5 <= h[5]*x[1];
22.3:     m6 <= h[6]*din;
23:       yr <= m0 + m1 + m2 + m3 + m4 + m5 + m6;
24:
25:    end
26: assign dout = yr[33:18];
27: endmodule
```

c) For the solution proposed in b), present and justify an estimate for the maximum clock frequency supported by the pipelined version of the circuit.

The answer to this question is already included in the answers above: minimum clock period is 8ns (frequency 125 MHz) that is now the maximum propagation delay of the combinational logic between registers.

[4 points]

5 - Consider that you want to implement a digital system with the same functionality of the circuit in question 4, but where the input data is now applied at a rate 20 times lower than the clock frequency. Assume that this new circuit has an additional input signal to enable the arrival of the input data.

In the next questions it not required the presentation of a detailed and complete Verilog module. If you consider it would be convenient for clarifying the solution proposed, you may illustrate your answer with Verilog and/or block diagrams, although a clear textual description can be sufficient. Provide as many details as possible and imagine you are explaining your solution to a colleague that will be responsible for writing down the Verilog implementation.

a) Explain how you could implement this new version of the system in order to minimize the logic complexity of the resulting circuit.

With 20 clock cycles available between input samples the same calculation process can be done sequentially using only one multiplier and one adder. Assuming that the clock period is greater than the propagation delay of the multiplier, the multiplication and the addition required for each iteration can be done in two consecutive clock cycles, consuming a total of 13 clock cycles. The remaining 7 clocks are more than sufficient for the rest of the control operations (zeroing the accumulator, loading the output register). A simple controller would also be necessary.

b) Assume that the input data rate is now 200 times lower than the clock signal frequency. Explain how you could exploit this feature in order to further reduce the complexity of the logic circuit you proposed in the previous paragraph.

In this scenario we have, in average, 28 clock cycles for calculating each multiply-add. The obvious solution here is to use a single sequential multiplier that, in its simplest form (the basic shift-add multiplier) would require 16 clock cycles to complete each 16 x 16 multiplication. This would use 112 clock cycles and the rest of the time is comfortably enough for doing all the additions (with a fully combinational adder) and any additional housekeeping.
More sophisticated solutions could be considered, as reusing the adder of the sequential multiplier for doing also the additions but the potential reduction in the logic complexity can only be evaluated after synthesis (note this would require additional registers and multiplexers).

-Ω= The end =ω-