FEUP
Universidade do Porto
Faculdade de Engenharia

Departamento de Engenharia Eletrotécnica e de Computadores

EEC0055 – Digital Systems Design

4º year - 1º semester
Exam – 24 January 2020

Maximum duration: 2h 30 m (closed book)

[3 points]

1 - The RTL synthesis tools such as XST (*Xilinx Synthesis Technology*) from XILINX which was used in the laboratory projects, allow the designer to specify various parameters that guide the optimization processes performed by these tools. Two basic parameters supported by the XILINX tool allow defining the overall optimization goal (area or speed) and the level of effort that is applied to these processes.

Consider the design of a synchronous digital circuit with a clock signal and explain whether the following statements are true or false:

[1.5 points]

a) By synthesizing a Verilog module that encodes a finite state machine (for example a sequential multiplier), using the Verilog model templates expected by the synthesis tool for coding finite state machines, and selecting optimization for speed, is it possible to reduce the number of states implemented by the finite state machine and thereby making the circuit perform the intended operation in a smaller number of clock cycles.

False: the number of states, and thus the number of clock cycles needed to complete a certain task, is determined by the functional description encoded by the HDL models (Verilog). If the HDL models are built correctly, according to coding rules expected by the synthesis tool, the models specify exactly all the data transfers between registers that will happen in each clock cycle (this is why the HDL models are usually called "RTL models" and the synthesis process "RTL synthesis"). The optimization processes done RTL synthesis can optimize various design parameters but will never change the behavior represented by the source Verilog code that dictates the number of states of a FSM and number of clock cycles used by a digital synchronous circuit.

[1.5 points]

b) Defining the optimization goal to minimize the area, is it possible that the RTL synthesis of the Verilog statement `always @ ( posedge clock ) Y <= A * B;` lead to the implementation of a sequential synchronous multiplier, that requires much less area than a combinational multiplier.

False: the multiplier specified by the operator "*" will always be translated to a fully combinational multiplier (actually, *any* right-hand side expression is translated to a combinational circuit), as this code specifies that the register Y is loaded with the result of A * B in every clock cycle. If a sequential synchronous multiplier is required, it must be built from scratch as a custom digital circuit, to compute the multiplication sequentially along a certain number of clock cycles, using a proper datapath and sequential controller.

[4 points]

2 - Consider the design of a synchronous sequential digital circuit with a single clock signal, that must be able to work with a 200 MHz clock signal. After concluding the RTL synthesis process, a maximum clock frequency of 63 MHz was reported and after the physical synthesis (*place&route*) this value was recalculated to 78 MHz, both well below the intended 200 MHz.

[1 point]

a) Explain how the timing analysis tools estimate the value for the maximum frequency of the clock signal.

The minimum clock period is estimated as the maximum propagation delay of all the combinational paths between flip-flops. The maximum clock frequency is then the reciprocal of the minimum clock period.

[1.5 points]

b) Explain which of the two values indicated represent the best estimate for the maximum clock signal frequency.

The value reported after place&route (78 MHz) is the most accurate. The P&R process defines the physical location of the logic blocks implementing the digital circuit and builds the electrical interconnections between them. Only after this stage the tools can characterize the circuit elements (logic and wiring) using delay models that are used to calculate the safe minimum clock period. The value reported after synthesis is a rougher approximation as the propagation delays of the interconnections is unknown and is only

estimated based on the design complexity, logic density and cell fanout (number of inputs connected to an output).

[1.5 points]

c) Although those values are much lower than the target clock frequency, a more detailed analysis of the timing reports created by the design tools, and taking into consideration the knowledge of the operation of the circuit, allow the designer conclude that the circuit can effectively work safely with the 200 MHz clock. Explain how it might be possible to reach that conclusion.

The minimum clock period calculated as described in a) assumes, by default, that all registers are loaded with the result of the combinational circuit preceding it in every clock cycle. If the circuit section that is limiting the clock frequency is clock-enabled only at every N clock cycles, and the time between the enabled (or active) clock cycles (N x clock period) is enough to propagate data through the constraining combinational circuits, the circuit can effectively be clocked with a frequency higher than the reported by the tools.

The static timing analysis (STA) tool allows measuring the combinational propagation delay from every flip-flop to every flip-flop. By using this tool, the designer can identify the circuit sections that are limiting the clock period, and knowing that these sections are working at a lower enough rate, as explained above, he/she may be able to conclude that.

[3 points]

3 - The construction of Verilog models for RTL synthesis processes must follow a set of coding rules currently accepted by synthesis tools. One of these rules requires that the construction of a synchronous circuit with clock must implement all the assignment operators with the *non-blocking* assignment (**`Y <= <expression>`**).

Consider the implementation of a synchronous circuit to calculate in each clock transition the **Y** value defined by the following pseudo-code (**a** and **b** are signals coming from registers synchronous with the same clock):

```
if (s)
   Y = (a * b) / (a + b);
else
   Y = (a * b) * (a – b);
```

To implement this circuit the following Verilog module was built, using (wrongly) the *blocking* assignments instead of *non-blocking* assignments:

```
always @(posedge clock)
begin
  if ( reset )
    Y = 0;
  else
  begin
    Ym = a * b; Ys = a – b; Ya = a + b;
    if ( s )
      Y = Ym / Ya;
    else
      Y = Ym * Ys;
  end
end
```

[1.5 points]

a) Explain whether the digital circuit resulting from the RTL synthesis of this code would implement correctly the intended function.

The behavior described by the Verilog code is the same as the pseudo-code given above. As the registers **Ym**, **Ys** and **Ya** are assigned using blocking assign statements, the values of those registers considered in expressions **Y = Ym / Ya** and **Y = Ym / Ys** are the values after the previous assignments are completed or **a * b**, **a – b** and **a + b**, respectively. The use of non-blocking assignments in clocked synchronous processes is not a mandatory rule for writing synthesizable Verilog models, but this is a recommended practice to ensure coherency between the behavioral simulation and the synthesized circuit.

[1.5 points]

b) Having detected that error, the correction carried out consisted of simply replacing all the "=" assignments for "<=". Explain whether or not this is correct and, if not, show how it could be corrected.

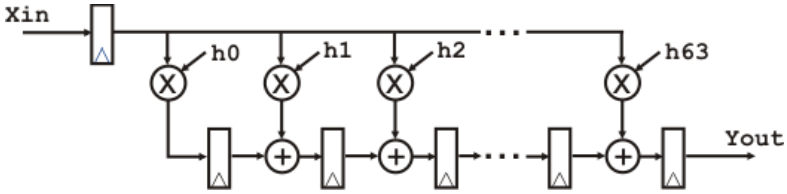
Just changing the blocking assignments to non-blocking assignments is not correct as that will change the function represented by the Verilog code. If non-blocking assignments are used, and keeping the rest of the code, register **Y** will be loaded with the values of **Ym**, **Ys** and **Ya** loaded in the previous clock cycle with results calculated with the values of **a** and **b** in the previous clock cycle.

One possible solution is re-writing the Verilog code with the same form given in the pseudo-code:

```
always @(posedge clock)
begin
  if ( reset )
    Y <= 0;
  else
  begin
    if (s)
       Y <= (a * b) / (a + b);
    else
       Y <= (a * b) * (a - b);
  end
end
```

[4.5 points]

4 - During the design of a digital signal processing application, it was necessary to implement a FIR filter similar to the **real2cpx** module, but requiring now a set of 64 coefficients (**h0** to **h63**, all different from zero and not necessarily symmetrical), instead of the 8 that were used in the laboratory project. The sampling frequency of the digital input signal is 30 MHz and it was decided to use a clock frequency which is an integer multiple of the sampling frequency. According to other design constraints, the options considered are 30 MHz, 60 MHz, 90 MHz or 120 MHz. The output may have a latency up to 1 µs. The architecture proposed for this circuit is represented in the following figure:

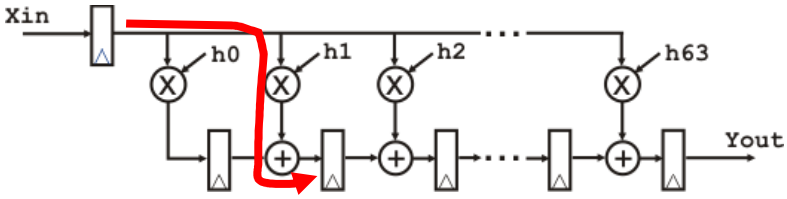


It is know that in the digital technology intended to implement this system, and for the data sizes involved in this design, the fastest multiplier and adder operators have a maximum propagation delay of 10 ns and 4 ns, respectively.

[1.5 points]

a) Considering the information available, explain which is the highest clock frequency (between the four possibilities shown) which will allow the implementation of the circuit as shown in the figure above.

The critical path in this circuit (along the combinational circuit between registers with the highest propagation delay) involves the circuit between the input register and all the other registers in the bottom section of the circuit except the leftmost:



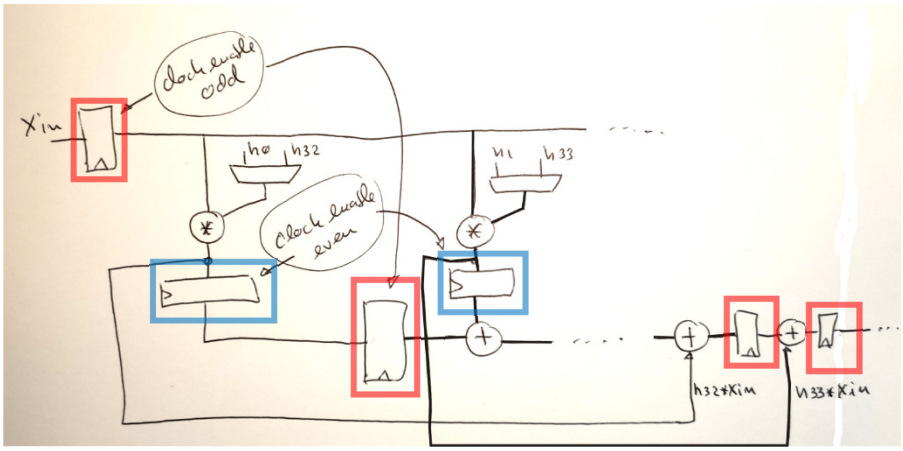
Given the information available, we can conclude that the propagation delay of the mult-add combinational sections of the circuit (the longest combinational path between registers) will be less than 10 + 4 ns (remember the highest propagation delay of the multiplier is to the MSB). The minimum guaranteed clock period is thus 14 ns and the highest clock frequency allowed by this circuit is 60 MHz (period 16.7 ns).

[1.5 points]

b) Consider that the objective of this project is to minimize the circuit size. In a first analysis this may be measured by the number of arithmetic operators required. Assuming the clock frequency you indicated in the previous paragraph is used, explain how the proposed circuit could be modified in order to reduce the number of arithmetic operators (note that the proposed circuit requires 64 multipliers and 63 adders).

The circuit given is able to calculate the output value Yout in a single clock cycle. With a 60 MHz clock we have two clock cycles between input samples. Making use of both clock cycles in each sampling period, we can use only 32 multipliers, registering the results of the products between the input sample and the coefficients h0 to h31 in the first clock cycle (when the input samples arrives) and computing the additions and the other multiplications in the second clock cycle.

Figure below shows a possible organization of the proposed circuit. In the clock cycle when a new input sample arrives, the "red" registers are enabled (odd clock cycle), with the muxs selected to the right data input (coefficients h32 to h63); in the next clock cycle, all the "blue" registers are enabled (clock enable even) and the mux is selected to the left data input (coefficients h0 to h31).



[1.5 points]

c) Now admit that it is necessary to implement a FIR filter similar to the one presented but intended to process a signal with a sampling frequency of only 100 kHz. Considering a 30 MHz clock signal, explain how to build an alternative circuit to the proposed one, in order to reduce significantly its size.

If the sampling frequency is 100 KHz and the clock frequency is 30 MHz we have now 300 clock cycles between two samples to compute 64 multiplications and 63 additions.
If using only one multiplier and one adder as a single cascaded combinational circuit, the operation can be completed in 64 clock cycles, as the combinational propagation delay of the multiplier and adder fits comfortably the 33 ns clock period. However, this solution will consume only 64 clock cycles (eventually one or two more for control purposes) and will not make use of the remaining 230+ clocks. This will also require a simple control unit to sequentially apply the different hk coefficients to the input of the multiplier (these may be stored into a ROM) and count the number of iterations.

To further reduce the logic complexity, a solution is to use sequential multipliers instead of combinational. However, depending on the number of bits of the operands it may not be possible to implement basic sequential shift-add multipliers processing one bit of the multiplier per clock. By dividing the 300 clock cycles by the 64 mult-add we conclude we have 4 clock cycles to calculate each mult-add operation (44 clock cycles will remain free for control and housekeeping). With 4 clock cycles available, the multiplier could be implemented as a sequential operator, by decomposing the N bits by M bits multiplication into four N/4 bits by M bits multiplications and additions, performed sequentially in 4 clock cycles.

[5.5 points]

5 - It is intended to develop a digital block that implements a real time clock. The development of the project must be conducted to optimize the circuit area (logical complexity) and energy consumption. The main functional requirements are:

i) Counting seconds, minutes, hours, day of month, month and year;
ii) Year count from 2020 to at least 2100, considering the compensation for leap years;
iii) Setting of the clock registers (writing into the counters) and reading through a synchronous serial interface (SPI).

The circuit to build should be synchronous with a single global clock signal. The circuits receives an external clock with frequency equal to 32768 Hz and this signal is divided internally to generate the 1 Hz clock signal that drives the entire circuit. The circuit is based on 6 counters with parallel loading capability via the serial interface. In time counting mode, the counters are enabled in cascade, each one being incremented when the previous counter reaches its maximum value (except the counter of seconds, which counts at each transition with the 1 Hz clock signal).
The 6 counters have the minimum number of bits required to represent the respective values (unsigned values):

Unit of time	Values to represent	Number of bits
Seconds	0 .. 59	6
Minutes	0 .. 59	6
Hours	0 .. 23	5
Day of month	1 .. 31	5
Month	1 .. 12 (1 = January, 12 = December)	4
Year	20 .. 100 (20 = 2020, 100 = 2100)	7

[1.5 points]

a) Assume that the counters of minutes and seconds produce at the outputs **maxmin** and **maxsec**, respectively, a signal set to 1 when the value of the respective counter is equal to 59. Write a synthesizable Verilog model that implements the counter of hours. The interface of this module should be:

```
module count_hours (
               input clk1hz,            // 1 Hz clock
               input reset,             // synchronous reset , active high
               input enable,            // global enable, active high
               input maxmin,
               input maxsec,
               input [4:0] hours_in,    // counter input
               input load_hour ,        // load hour_in to counter
               output [4:0] hours_out,  // counter output
               output maxhour           // set to 1 when current hour == 23
          );

 reg [4:0] hcount;            // The counter of hours, assigned below to hours_out

 always @(posedge clk1hz)
   if ( reset )
     hcount <= 5'd0;
   else
     if ( enable )
       if ( load_hour )
         hcount <= hours_in;          // Load input value to counter
       else
         if ( maxmin & maxsec ) // hour register iterates only at 59:59
           if ( maxhour )       // hour increments to 0 when it is 23
             hcount <= 5'd0;
           else
             hcount <= hcount + 5'd1;

 assign maxhour   = ( hcount == 23 );  // This is 1 when the current hour is 23
 assign hours_out = hcount;

 endmodule
```

[1.5 points]

b) Now, using the **maxhour** output of the module built in a), develop the circuit (Verilog synthesizable) that implements the counter of day of the month, implementing the leap year compensation for the days of the month of February. For the specified range of years, all years divisible by 4 are leap years, except the year 2100. Consider the inputs and outputs you deem necessary and explain your solution.

```
module count_days (
               input clk1hz,            // 1 Hz clock
               input reset,             // synchronous reset , active high
               input enable,            // global enable, active high
               input maxmin,
               input maxsec,
               input maxhour,           // need to increment the day only at 23:59:59
               input [3:0] month,       // we need to know the month and year
               input [7:0] year,        // to implement the leap year compensation
               input [4:0] day_in,      // counter input to set the day
               input load_day ,         // load day_in to counter
               output [4:0] day_out,    // counter output
               output maxday            // set to 1 when the last day of current month
          );

 reg [4:0] dcount;              // The counter of days, assigned below to day_out

 always @(posedge clk1hz)
   if ( reset )
     dcount <= 5'd1;            // reset the counter of days to 1 (not zero!)
   else
     if ( enable )
       if ( load_day )
         dcount <= day_in;      // Load input value to counter
       else
         if ( maxmin &
                 maxsec &
                    maxhour )   // day register iterates only at 23:59:59
           if ( maxday )        // day is set to 1 when it is the last day of the month
             dcount <= 5'd1;    // Reset the day counter to 1 (not zero!)
           else
             dcount <= dcount + 5'd1;
```

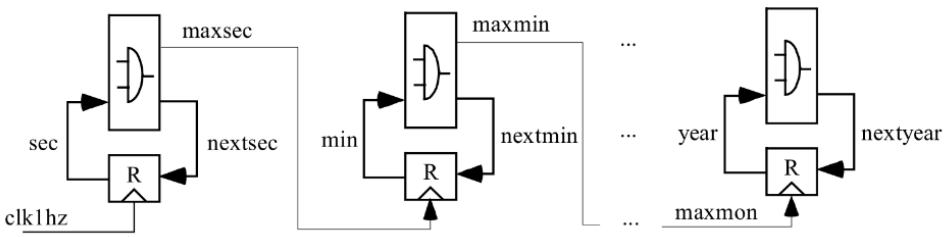
```
// Set the number of days of each month (combinational process)
// The condition for month 2 resolves to 28 or 29 depending
//   on the leap year rule: year divisible by 4 and different from 100
always @*
case ( month )
  4'd1:  maxday = 5'd31; // Jan
  4'd2:  maxday = ( ( year[1:0] == 2'd0 ) && year != 7'd100 ) ? 5'd29 : 5'd28; // Feb
  4'd3:  maxday = 5'd31;
  4'd4:  maxday = 5'd30;
  4'd5:  maxday = 5'd31;
  4'd6:  maxday = 5'd30;
  4'd7:  maxday = 5'd31;
  4'd8:  maxday = 5'd31;
  4'd9:  maxday = 5'd30;
  4'd10: maxday = 5'd31;
  4'd11: maxday = 5'd30;
  4'd12: maxday = 5'd31;  // Dec
  default: maxday = 5'dx; // set to don't care
endcase

assign day_out = dcount;

endmodule
```

[1.5 points]

c) Instead of designing the entire synchronous circuit with the only global 1 Hz clock signal, an alternative solution was proposed, which consists of using the signal indicating the end of counting of the previous counter as the counter clock:



Explain whether this solution has advantages or disadvantages with respect to the initial proposal (synchronous circuit), in what concerns to (i) occupied area, (ii) speed and (iii) energy consumption.

(iii) energy consumption: this implementation will consume less dynamic power as the clock signals of the counters above the seconds will be clocked with much slower clock signals than in the synchronous solution.
(ii) speed: the primary function of this circuit is to count time. The clock frequency applied to the synchronous circuit and the rate at each counter increments (in this last version) is dictated by the functional specification of the system. It does not make sense to compare both circuits in terms of speed!
(i) area: The main blocks of both circuits are the 6 counters that will have similar complexity. The synchronous version may require a bit more logic to generate the individual clock enable signals that will enable the counters running slower than 1 Hz (corresponding to the logic implementing the condition: `if ( maxmin & maxsec & maxhour )` in the code presented in the previous answer).

[1 point]

d) In the solution referred to in the previous question, the **max…** signals used as clock signals can be produced as a result of the combinational comparison of the counter value (for example, for the counter of seconds, using the statement: **`assign maxsec = ( sec == 59 )`**? Explain why and if your answer is negative, provide a solution that may ensure the correct operation of the circuit.

The statement **assign** will implement signal maxsec as the output of a combinational circuit. As the output of a combinational circuit may present surplus transitions (or glitching) when changing between logic levels, combinational outputs should never be used as clock signals to edge-triggered flop-flops as several active transitions may happen. In this example, the counter of minutes would probably increment several times when sec changes from 58 to 59 and from 59 to 0.

One solution is to synchronize this maxsec signal to create a glitch-free gated clock derived from the main 1 Hz clock and not drive the clock input of the counter of minutes directly with the maxsec signal.

-◖= the end =◗-