FEUP Universidade do Porto Faculdade de Engenharia

Departamento de Engenharia Electrotécnica e de Computadores

Digital Systems Design 2015/2016

4th year – 1st semester
Exam – January, 25th 2016

Duration: 2h30m, closed book.

[1.5 points]
1 - Explain the difference (objectives and what is simulated) between the post-route simulation (after executing the place&route or physical synthesis) and the post-synthesis simulation (after performing the RTL synthesis).

Both simulations simulate a netlist (logic circuit) created by the RTL synthesis process from the functional HDL specification. While the post-synthesis simulation uses simplified timing models (propagation delays) for the interconnections and for the logic blocks, the post-route simulation (or timing simulation) includes accurate timing models of logic and interconnections, extracted from the routing actually implemented on the target device. These two simulation stages have different purposes: the post-synthesis simulation verifies the functional correctness of the logic circuit that may differ from the behavior of the HDL models (mainly) due to incorrect modelling in HDL; the post-place&route simulation verifies the operation of the logic circuit taking into account the propagation delays, also verifying the compliance to the timing constraints of the memory elements (flip-flops, latches, memories).

[1.5 points]
2 - When designing a synchronous digital system with a single clock domain for a FPGA device, is it safe to avoid the post-route simulation? If yes, under which conditions? Explain.

Under the conditions stated, and considering the implementation for a FPGA reconfigurable device, the post route simulation can be avoided if the final timing reports indicate a maximum clock frequency allowed for the circuit clearly above the desired operating clock frequency [as a reference, XILINX used to recommend not running a circuit with a clock faster than 80% of the maximum frequency reported by the timing analysis tools]. However, if the circuit has some section that is not fully synchronous with the clock (as, for example, an external asynchronous input), the post-route simulation should be performed to evaluate the behavior of the circuit under variable timing of the asynchronous signals.

[1.5 points]
3 - During the RTL synthesis process, several optimizations are performed according to implementation goals the designer may set (for example, minimize the area). Is it possible that some synthesis optimizations may lead to a reduction of the number of clock cycles needed to complete a certain operation? Justify.

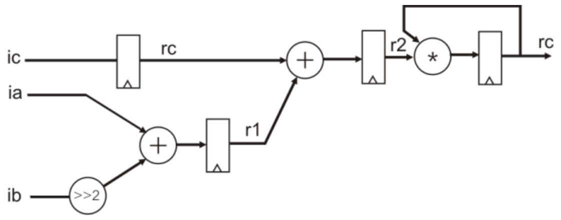
No. The number of clock cycles used by a digital system (assumed clocked synchronous) is determined only by the architecture of the circuit established by the HDL models, and this is not modified by any optimization done during the RTL synthesis process. These optimizations may reduce the propagation delays in the combinational circuits and thus reduce the clock period (or increase the clock frequency), but always keeping the number of clock cycles.

[4 points]
4 - The Verilog code below describes a synchronous datapath, where all the signals are 32 bit wide.

```
always @(posedge clock)
begin
  rc <= ic;
  r1 <= ia + (ib >> 2);
  r2 <= r1 + rc;
  r3 <= r3 * r2;
end
```

[2 points]

a) Draw a block diagram representing the digital circuit created by the synthesis of this code.



[2 points]

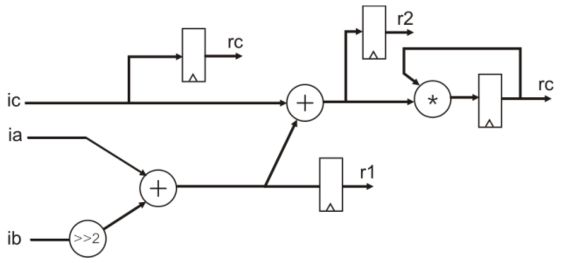
b) The designer in charge of this project has wrongly used blocking assignments (`rc = ic`…) instead of the correct non-blocking assignments (`rc <= ic`…). Draw the block diagram representing the digital circuit that would be synthesized from that code.

Note that in the sequence of blocking assignments:

r1 = ia + (ib >> 2);

r2 = r1 + rc;

the 'r1' in the second assignment represents the value that results from the previous assignment The seconds assignment is thus equivalent to r2 = ia + (ib >> 2) + rc. However, r1 is also translated to a register, although its value is not used in this block (the same happens for rc and r2).



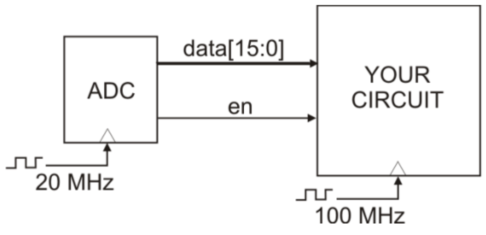
[1.5 points]

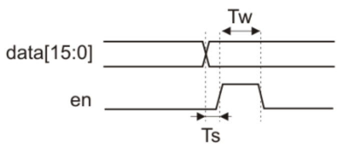
5 - In the last laboratory project the frequency of the clock signal has been imposed equal to 24.576 MHz. If you could choose freely the clock frequency for your design, what would you do? Consider a scenario where the same design goal also applies (minimize the circuit area).

To minimize the area the solution is to share hardware resources as most as possible, and that implies increasing the clock frequency. Regarding the last laboratory project, the utilization of the slowest and smallest version of the sequential multiplier would require a faster clock to accommodate all the operations within the available time (or number of clock cycles). An even higher clock frequency could allow using only one single multiplier, shared by the four filter blocks.

[2 points]

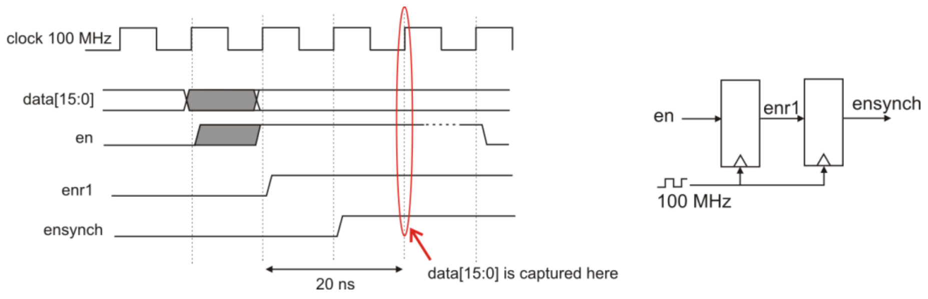
6 - You have been assigned for designing a digital system that will receive and process data from an analog to digital converter. The ADC is clocked by a local clock signal with frequency equal to 20 MHz and your digital system will be synchronous with a 100 MHz clock, generated by another clock source. The ADC outputs data samples at a rate of 1 Msamples/s. The ADC output is a 16 bit bus (signal `data[15:0]`) plus a one bit enable signal (signal `en`). When the ADC outputs a new sample, the enable signal is set during 50ns (one period of the 20 MHz clock), according to the timing diagram below. Explain how to implement the data synchronization in order to guarantee a correct data acquisition into the clock domain of your circuit. Illustrate your answer with a logic diagram or a section of Verilog code.



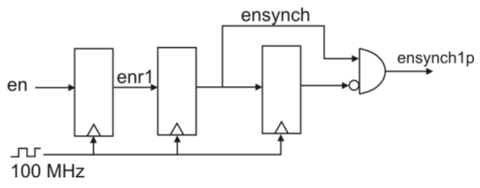


Ts min = 0 ns
Ts max = 2 ns
Tw = 1/20MHz = 50 ns

*The signal **en** can be synchronized with the 100 MHz clock using a two stage shift-register. The output of this synchronizer **ensynch** will be delayed by at least 20 ns (two clock periods of the 100 MHz clock), with respect to the **en** input. Thus, using this signal to enable loading a register with input **data[ ]** guarantees a correct acquisition of this data into the 100 MHz clock domain.*

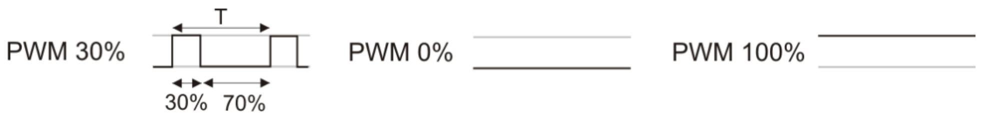


*This solution will create a synchronous enable signal (**ensynch**) with a duration of at least 4 clock cycles (40 ns). A single clock period enable pulse can be easily created by adding a third flip-flop to delay **ensynch** by one additional clock period and then and-ing the negation of this signal with **ensynch**:*



[3 points]

7 - A PWM modulator is a digital circuit that generates a square wave with a constant frequency and duty-cycle proportional to an input value. A duty-cycle equal to xx% should generate a square wave with a certain fixed frequency, being 1 for xx% of the period and 0 for the rest of the period; a duty-cycle of 0% means the output is constantly set to zero; a 100% duty-cycle means the output constantly equal to 1:



Build a synthezisable Verilog module implementing a PWM modulator as a clocked synchronous digital circuit, using a single clock domain and a synchronous global reset. The PWM modulator should have the following characteristics:

- Main clock signal: 10 MHz (period 100 ns)
- Output PWM frequency: 100 KHz (period 10 μs)
- Input: 7 bit unsigned, represents the duty-cycle of the output wave in percentage of the period (0% to 100%). Input values greater than 100 should be considered as 100.

```
module pwm( input clock,
            input reset,
            input [6:0] pwmin,
            output pwmout
          );

reg [6:0] count, pwminr;

always @(posedge clock)
if ( reset )
begin
  count <= 7'd0;
  pwminr <= 7'd0;
end
else
  if ( count == 7'd99 ) // counts 100 clock periods (0 .. 99 )
  begin
    count <= 7'd0;
    pwminr <= pwmin;     // this guarantees the input data
  end                    // is constant during one PWM period
  else
     count <= count + 1;

  // pwmout is 1 when the counter value is less than the required pwm %
  // values greater than 100 are naturally considered equal to 100:
  assign pwmout = ( count < pwminr );

endmodule
```

[5 points]

8 - Your team has been assigned to implement a module for accelerating the multiplication of 8x8 matrices. This system will be implemented in a small FPGA that will be attached to a microcontroller as a memory-mapped peripheral. The interface between the microcontroller and your circuit is implemented through a synchronous memory write/read port, similar to the interface used to access the coefficient memories in the 3rd laboratory project. Remember matrix multiplication: the element *M(i,j)* of the result of multiplication A.B is:

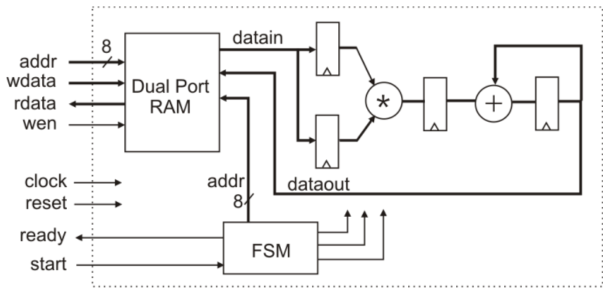
$$M(i,j) = \sum_{k=0}^{7} A(i,k) \times B(k,j)$$

The application running in the microcontroller will need to perform sequences of 32 multiplications of matrices A x B, where matrix B is the same for the whole sequence but using 32 different matrices A. To perform one set of multiplications, the microcontroller will execute the following pseudo-program:

```
Load matrix B to the address space (0..63)
For i=0 to 31
    Load matrix Ai to the address space (64..127)
    Activate signal START (start matrix multiplication)
    Wait for signal READY to be set (matrix mult. has ended)
    Read the result matrix (R = Ai x B) from address space (128..191)
```

Matrices are stored in memory row by row: the element **`(i,j)`** of a matrix is located in address **`baseAddr+i*8+j`**, where **`baseAddr`** is 0 for matrix B, 64 for matrix A and 128 for matrix R (the result).

The first proposal for this system uses a dual port memory and only one multiplier and accumulator:



[1.5 points]

a) Indicate an approximate number of clock cycles needed to perform one matrix multiplication using this datapath.

To complete one multiplication of the two matrices we need to repeat 64 times the row x column inner product (expression above), one for each element of the result matrix. To compute one row x column product we need to execute 8 multiplications and 8 additions (accumulation). As the proposed system only has one memory access port, we can only read or write one element at a time. Considering only the execution of one row x column inner product, the operations executed in each clock cycle are the following:

Cycle	*Operation*
0	*Load mult operand A0; Clear accumulator*
1	*Load mult operand B0*
2	*Load mult result A0xB0; Load mult operand A1*
3	*Load accumulator; Load mult operand B1*
4	*Load mult result A1xB1; Load mult operand A2*
5	*Load accumulator; Load mult operand B2*
	…
14	*Load mult result A6xB6; Load mult operand A7;*
15	*Load accumulator; Load mult operand B7;*
16	*Load mult result A7xB7;*
17	*Load accumulator;*
18	*Write result*

We thus need 16 clock cycles to read the 16 operands into the input registers, two clock cycles at the end to flush the pipeline and a final cycle to write the result into memory. The total number of clock cycles will be 64 x 19 = 1216.

[2 points]

b) To improve the performance, one suggestion is to use three independent memories with 64 locations for each of the three matrices, instead of a single memory. Explain how this solution can be exploited to improve the performance of the matrix multiplication and present an approximate value for the minimum number of clock cycles required for this implementation.

Using three memories it is now possible to read the two operands of the multiplier in the same clock cycle, this reducing the first 16 clock cycles of the previous solution to only 8. After reading the last operands (A7 and B7, in the 8th clock cycle), we need more two clock cycles to flush the pipeline (load the multiplier result and loading the accumulator) and one more clock cycle to write the result into memory. As we have an independent memory for the result, this last operation can be done in parallel with the read of the first operands for the next row x column product. The total number of clock cycles is (8+2+1) x 64 = 704 (58% of the previous solution), or (8+2) x 64 + 1 = 641 (53%), if overlapping the write operation with the reads of the next row x column cycle.

[1.5 points]

c) To further reduce the computation time, one teammate suggested using more multipliers and accumulators, working in parallel. Comment this proposal considering a datapath with 4 multipliers and accumulators and explain how to modify the circuit to exploit conveniently the parallel operation of the multipliers and accumulators.

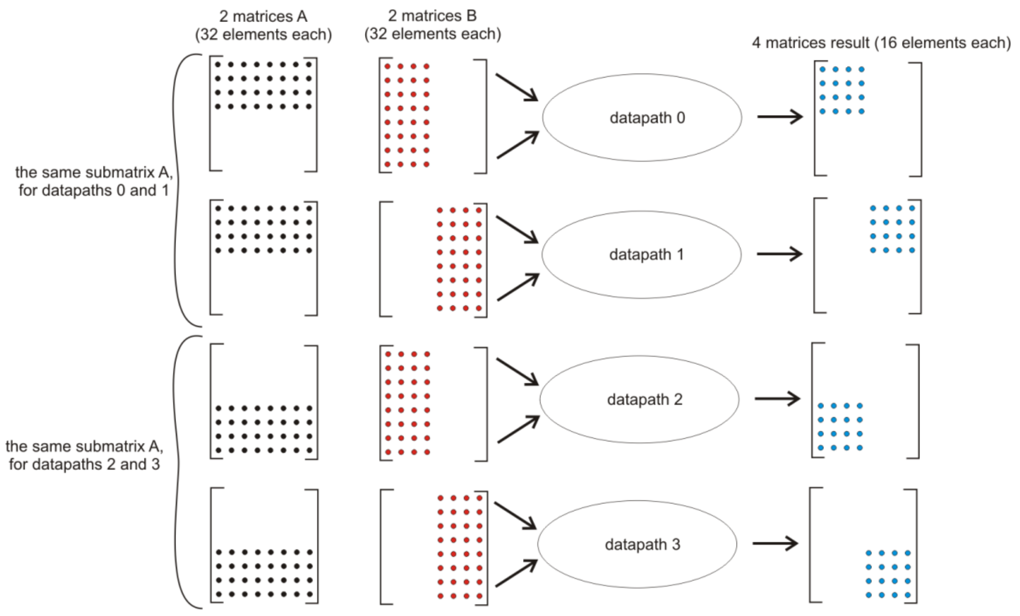
With 4 similar datapaths it is possible to paralelize the operations and divide the computation time by approximately 4, but only if two conditions are met:

i) The algorithm allows the paralelization into four independent computations
ii) The memory system is able to provide in parallel all the operands to the 4 datapaths, at the clock rate.

In our problem (matrix multiplication) the first condition is true: we can calculate independently each element of the result matrix. In the limit, using 64 of these datapaths it would be possible to calculate in parallel all the elements of the result matrix, but only if condition ii) is true! The second condition requires designing a different memory structure and data organization in the memories in order to supply the data to the datapaths, at clock rate.

Additional notes:
An answer addressing the topics referred above was considered correct. Nevertheless, additional details about the memory and data organization are included below to help understand a possible solution to this problem.

A simple solution uses four sets of 3 memories, to hold slices of rows of matrix A, columns of matrix B and sub-matrices of the matrix result, as depicted in the figure below (each sub-matrix only stores the data represented by the colored dots). However, the first two sub-matrices A store the same top 4 rows and can be addressed with the same sequence of addresses to feed datapaths 0 and 1. Thus, only one memory can be used instead of two (the same happens with the two bottom memories for matrix A, the first and thrid memories for matrix B and the second and fourth memories for matrix B). With this simplification, only 8 independent memories would suffice: 2 for matrix A (4 rows each), 2 for matrix B (4 columns each) and 4 for the matrix result (4x4 elements each).


2 matrices A (32 elements each)
2 matrices B (32 elements each)
4 matrices result (16 elements each)
the same submatrix A, for datapaths 0 and 1
the same submatrix A, for datapaths 2 and 3
datapath 0
datapath 1
datapath 2
datapath 3