FEUP
Universidade do Porto
Faculdade de Engenharia

Departamento de Engenharia Electrotécnica e de Computadores

Digital Systems Design 2016/2017

4th year, 1st semester
2nd exam - 1 February 2017

Maximum duration: 3h00m, closed book.

[1.5 points]

1 - The RTL synthesis tool integrated in the XILINX design environment (*XST - Xilinx Synthesis Technology*) provides an estimate of the maximum clock frequency supported by a digital system synthesized from a Verilog RTL model. Explain why the maximum clock frequency obtained after the place&route is, in general, different from the clock frequency estimated by the synthesis tool (consider a Verilog model that uses only one clock domain).

The RTL synthesis creates a logic netlist (a schematic) interconnecting the primitive blocks available in the target FPGA device (look-up tables, flip-flops, RAM blocks, multipliers, etc), but does not define the physical location of each block in the FPGA fabric. Although these blocks are characterized by their propagation delay, the propagation delay of the interconnections between the primitive blocks cannot be accurately determined at this stage. To provide a first approximation of the maximum clock frequency, the synthesis tool estimates the propagation delay of the critical path (the slowest combinational path from flip-flop to flip-flop) based on the accurate propagation delay models of the FPGA blocks, but using only an estimate for the propagation delay of the interconnections, as there is no information about their physical layout. After place&route all the blocks and interconnections are physically located and the timing analysis tool can compute the most accurate delays (and the minimum clock period) that will depend on the actual type and number or interconnection resources used (wire segments, configuration switches, and even look-up tables that may be used to route some signals). Because of this, the maximum clock frequency reported at these implementation stages is, in general different. Note that the clock frequency estimate after synthesis is not necessarily a pessimistic estimation!

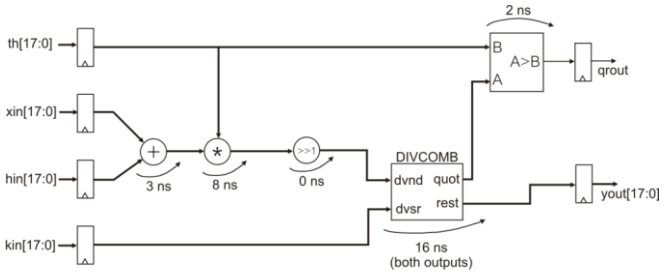
[1.5 points]

2 - After performing the place&route implementation stage for a FPGA device, the timing analysis tool reports a maximum clock frequency equal to 155 MHz. However, the timing simulation (or post-route simulation) shows this circuit can operate correctly with a 300 MHz clock (and this has also been verified with the system working in the FPGA device). Considering that the clock frequency estimate given by the timing analysis tool is correct, explain why such situation can happen.

In a clocked synchronous circuit, the maximum clock frequency reported after P&R is calculated as the inverse of the propagation delay of the slowest combinational path from flip-flop to flip-flop. This assumes that both registers (source and destination) are loaded with new data at every clock cycle. However, this may not be true if that section of the circuit is enabled at a lower rate than the clock signal. The situation described is justified if the registers at the input and output of the critical section, limiting the clock frequency to 155 MHz, are only enabled at each two clock cycles and there is no other section in the circuit limiting the clock frequency below 300 MHz.

[4 points]

3 - The diagram shown a digital synchronous circuit where the block **DIVCOMB** is a fully combinational divider whose interface is given below the diagram. This system will process a continuous stream of data applied at its inputs at the clock rate of 50 MHz (period 20 ns), generating the results **yout** and **qrout** at the same rate. All the internal buses are 18 bit wide and the propagation times of the functional blocks are indicated near each one.




```
module DIVCOMB( input [17:0]  dvnd, // Dividend
                input [17:0]  dvsr, // Divisor
                output [17:0] quot, // Quotient
                output [17:0] rest  // Rest
              );
```

[1.5 points]

a) Write a synthesisable Verilog model representing the circuit shown, using the interface below that does not need to be copied to your answer.

```
module Q3( input         clock,
           input         reset,
           input  [17:0] th,
           input  [17:0] xin,
           input  [17:0] hin,
           input  [17:0] kin,
           output reg [17:0] yout,
           output reg        qrout
         );

reg  [17:0] th_r, xin_r, hin_r, kin_r; // the 4 input registers
wire [17:0] multout;                   // the output of the multiplier
wire [17:0] multdiv2;                  // the multiplier output divided by 2
wire [17:0] quot, rest;                // the outputs of the divider
wire        AGTB;                      // output of comparator

// input registers:
always @(posedge clock)
if ( reset )
begin
  th_r  <= 18'd0;
  xin_r <= 18'd0;
  hin_r <= 18'd0;
  kin_r <= 18'd0;
end
else
begin
  th_r  <= th;
  xin_r <= xin;
  hin_r <= hin;
  kin_r <= kin;
end

// adder and multiplier:
assign multout = ( xin_r + hin_r ) * th_r;

// divider by 2 (assuming data is unsigned):
assign multdiv2 = multout >> 1;

// Instantiate the divider combinational module:
DIVCOMB divcomb1( .dvnd( multdiv2 ), .dvsr( kin_r ),
                  .quot( quot ),     .rest( rest )
                );

// Comparator:
assign AGTB = ( quot > th_r ) ? 1'b1 : 1'b0;

// Output registers:
always @(posedge clock)
if ( reset )
begin
  qrout  <= 1'd0;
  yout <= 18'd0;
end
else
begin
  qrout  <= AGTB;
  yout <= rest;
end

endmodule;
```

A more compact version is presented below. Note, however, that more compact HDL code does not mean necessarily better code! Packing long combinational expressions into a single statement (like **`assign multdiv2=...`** *) is not recommended as this removes the observability of the partial results within the expression (for example, in this case the output of the multiplier before the shift operation and the output of the adder before the multiplier cannot be monitored during a simulation).*

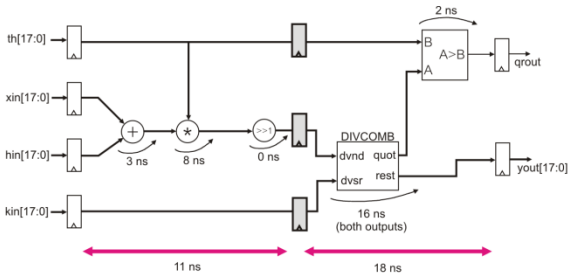
```
module Q3( input         clock,
           input         reset,
           input  [17:0] th,
           input  [17:0] xin,
           input  [17:0] hin,
```

```
            input  [17:0] kin,
            output reg [17:0] yout,
            output reg        qrout
          );

reg  [17:0] th_r, xin_r, hin_r, kin_r; // the 4 input registers
wire [17:0] multdiv2;                  // the multiplier output divided by 2
wire [17:0] quot, rest;                // the outputs of the divider
wire        AGTB;                      // output of comparator

// input and output registers:
always @(posedge clock)
if ( reset )
begin
  th_r  <= 18'd0;
  xin_r <= 18'd0;
  hin_r <= 18'd0;
  kin_r <= 18'd0;
  qrout  <= 1'd0;
  yout <= 18'd0;
end
else
begin
  th_r  <= th;
  xin_r <= xin;
  hin_r <= hin;
  kin_r <= kin;
  qrout  <= ( quot > th_r ) ? 1'b1 : 1'b0;
  yout <= rest;
end

// adder, multiplier and divider by 2:
assign multdiv2 = ( ( xin_r + hin_r ) * th_r ) >> 1;

// Instantiate the divider combinational module:
DIVCOMB divcomb1( .dvnd( multdiv2 ), .dvsr( kin_r ),
                  .quot( quot ),     .rest( rest )
                );

endmodule;
```

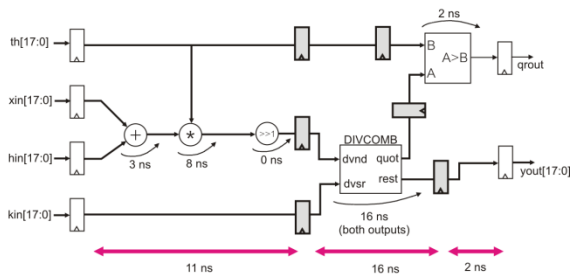
[1.5 points]

b) After completing the place&route the maximum clock frequency reported for this circuit is 30 MHz (33ns period), below the 50 MHz required. Show how to transform the circuit given in order to reach that clock frequency (period 20 ns). Represent you solution on the block diagram given or in the Verilog code proposed in a), and justify adequately the modifications proposed.

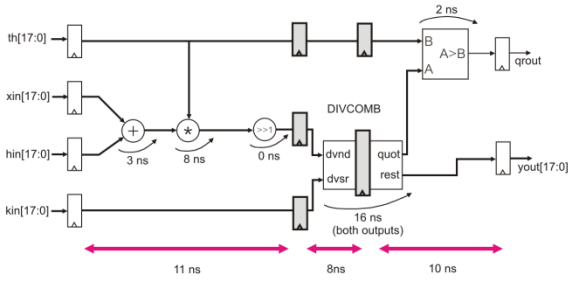
*The critical path of this circuit is from registers **xin_r** and **hin_r** to register **qrout**, passing through all the functional blocks. The propagation delays represented in the diagram totalize 29 ns, representing the nominal minimum clock period and not counting the delays introduced by the interconnections (the 4 ns different to the 33 ns clock period accounts for the interconnection delay). To increase the clock frequency to 50 MHz (20 ns period) the better (and simplest) solution is to build a pipelined circuit. To maintain the same **DIVCOMB** block, we can insert only one register stage at both inputs of the divisor and at the input B of the comparator. The second stage (divider and comparator) as a propagation delay of 18 ns and the first stage (adder and multiplier) only 11 ns. This solution may guarantee the 20 ns clock period, but has only 2 ns of time slack to the minimum clock period. Is this solution does not fit the required timing, a more aggressive pipelining would need a second pipeline stage at the output of the divider and both inputs of the comparator. If this solution does not yet meet the timing constraints, then it would be necessary to redesign the divider in order to move back the second pipeline register to within the divider logic. Figures i), ii) and iii) illustrate these three options.*



i)



ii)



iii)

[1 point]

c) Explain how the two solutions (original and your proposal in b)) compare in terms of power consumption.

As the original circuit can only run at 30 MHz and the new circuit is able to run with the required 50 MHz clock, then the second circuit will consume more dynamic power (proportional to the clock frequency). Besides, the new circuit has the additional 3 (or 6) pipeline registers, which contribute to the increase of the dynamic power consumption (higher load in the clock signal and more nodes with switching activity). Because the new circuit will be larger (more logic for the additional registers) the static power consumption will also increase.

[3 points]

4 - A critical situation that may occur during the RTL synthesis is the inference of transparent latches in a block intended to be combinational, represented by a process **`always @* ...`** .

[1.5 points]

a) Explain in which conditions this situation may occur and illustrate with an example in Verilog.

Transparent latches are inferred from HDL code intended to represent a combinational block when its outputs are not defined for all the conditions that are evaluated within that block. The simplest example is illustrated in the Verilog code:

```
always @*
if ( <condition> )
  output = <expression true>;
```

According to the semantic of the Verilog statement **`always`**, *when* **`<condition>`** *evaluates to false, the signal* **`output`** *will hold its current value. This behavior is implemented with a transparent latch whose enable signal will be driven by the result of* **`<condition>`**. *Similarly, this also happens when a* **`case`** *statement does not define all the output values of the block for all the possible conditions evaluated.*

Note that the correct answer to this question is not "using an **`if`** *without the* **`else`**" *or "using a* **`case`** *with missing cases or without the* **`default`** *(incomplete case)". For example, the code below does not have the else but implements correctly a combinational block:*

```
always @*
begin
  output = <expression false>;
  if ( <condition> )
    output = <expression true>;
end
```

[1.5 points]

b) Explain why this situation is, in general, fatal for the behavior of a circuit and describe in which verification stage this fault may be detected.

Assuming the block intended to be combinational is driven by registers clocked with the same clock signal, the value stored in the latch when the **<condition>** *evaluates to false will depend on the relative propagation delays of the signals involved in the computation of* **<condition>** *and* **<expression>**. *If the result of the expression is ready only after the condition is evaluated to false, then the behavior of the circuit will match the functional behavior of the Verilog code (and the circuit will work as expected!). If the expression is computed faster than the condition, then the data stored in the latch will be the result of the expression computed with the new data after the clock transition. And if several multibit signals are involved, the final result hold in the latch may have some bits from the previous state, some other from the next state and even some random. A complete mess!*

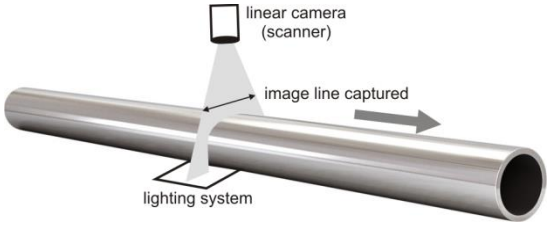
As this wrong behavior depends directly on the actual timing of the signals, this situation can only be detected during the timing simulation stage (after P&R). Note that the XILINX synthesis tool issues a warning message when it detects such HDL situations, but the question here refers to a verification stage.

Additional comment: of course the transparent latches are useful in certain situations and the code above can be correctly synthesized if the synthesis and/or P&R is guided by appropriate timing constraints to guarantee that the wrong behavior described above does not occur.

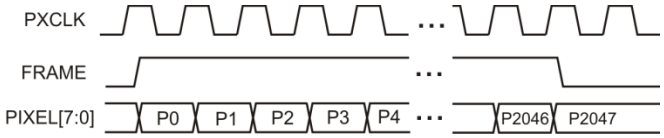
[4 points]

5 - In an industrial process it is necessary to design a digital system to measure with no physical contact and in real time the diameter of a metallic tube as it exits the machine producing it.

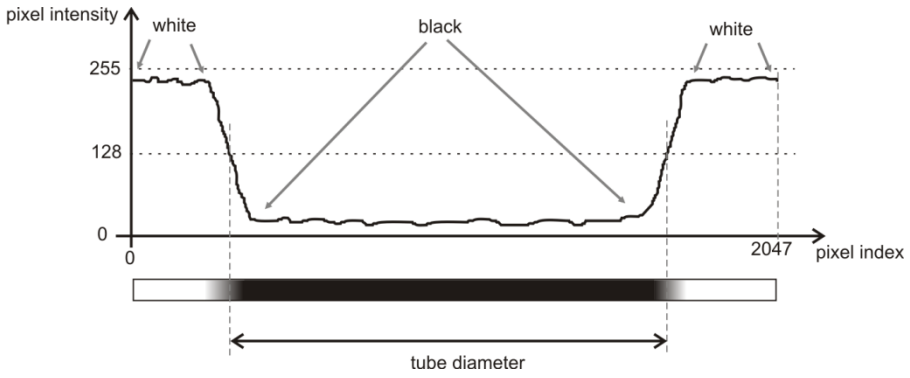
The system will be based on a linear camera that captures a single image line perpendicular to the tube axis with 2048 pixels with a frequency of 10.000 single-line images per second. The image captured by the camera is processed by a custom digital system to calculate in real time the diameter of the tube, for each image frame.



The image acquired by the camera is unidimensional (single line of pixels), monochrome with 8-bit per pixel. Each image is formed by 2048 pixels generated by the camera synchronously with a 24 MHz clock (**PXCLK**). The signal **FRAME** is active high while the camera outputs the 2048 pixels and the signal **PIXEL[7:0]** contains the 8-bit values that represent the pixel intensity (0=black, 255=white). The timing diagram below shows the operation of the camera serial interface.



With a convenient illumination system, the image captured by the camera will be formed by dark pixels (value below 30) in the region obstructed by the tube and white pixels (value above 200) in both sides of the tube. The tube diameter can be calculated as the distance (in number of pixels) between the white-black and black-white transitions, converted to metric units by multiplying by a scale factor (the whole image corresponds to approximately 40 mm and thus each pixel represents 19 μm)



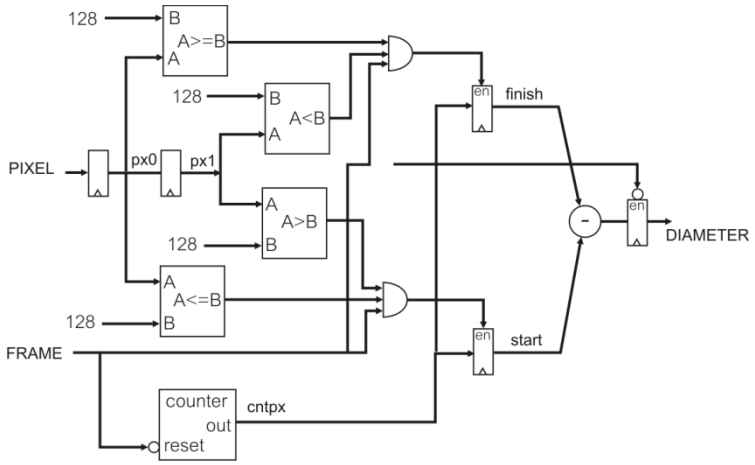
A simple process to measure the tube diameter consists in determining the index of the pixels passing by a certain threshold level, for example half of the dynamic range of 8 bit unsigned = 128. If *pixel(i) > 128* and *pixel(i+1) <= 128* (*i= 0.. 2047*), then *i* represents the index of a pixel in the transition white-black (left edge of the tube).When *pixel(j) < 128* and *pixel(j+1) >= 128* (*j= 0.. 2047*), the value *j* represents the index of the pixel in the black-white transition (right edge of the tube). The final measure will be *j-i*, with an accuracy below the 19 μm (1 pixel).

Build a block diagram and the corresponding Verilog synthesizable module that implements this circuit. The main clock signal is **PXCLK** (24 MHz, positive active edge) and the final result must be present at the output synchronously with the clock, when **FRAME** goes low. The global reset must be synchronous and active with the logic level high.

A first solution implements directly the procedure described above:

i. *counter **cntpx** increments from 0 to 2047 when **FRAME**==1 (its value represents the current pixel index);*
ii. *a two-stage shift-register holds the current pixel in **px0** and the previous pixel in **px1**; register is loaded with input **PIXEL** when **FRAME** is 1;*
iii. *register **start** is loaded with **cntpx** value when **px0** <= 128 and **px1** > 128;*
iv. *register **finish** is loaded with **cntpx** value when **px0** >= 128 and **px1** < 128;*
v. *when **FRAME**==0 the output **DIAMETER** is loaded with (**finish**-**start**);*

A block diagram that implements this system is (for clarity, the global clock and reset are not represented):



The Verilog code representing the block diagram is:

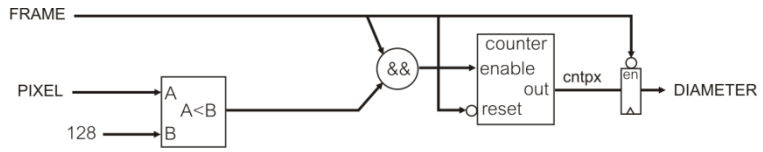
```
module Q5( input         PXCLK,
           input         reset,
           input  [7:0]  PIXEL,
           input         FRAME,
           output reg [11:0] DIAMETER
         );

reg  [7:0] px0, px1;       // the two-stage shift-register
reg [11:0] cntpx;          // the pixel index counter
reg [11:0] start, finish;  // pixel index of white-black and black-white edges

// all in a single process:
always @(posedge PXCLK)
if ( reset )
begin
  px0 <= 8'd0;
  px1 <= 8'd0;
  cntpx  <= 12'd0;
  start  <= 12'd0;
  finish <= 12'd0;
  DIAMETER <= 12'd0;
end
else
begin
  if ( FRAME )
  begin
    cntpx <= cntpx + 12'd1;  // pixel index count
    px0 <= PIXEL;            // current pixel
    px1 <= px0;              // previous pixel
    if ( px0 <= 8'd128 && px1 > 8'd 128 )   // white-black transition
      start <= cntpx;                       // store pixel index
    if ( px0 >= 8'd 128 && px1 < 8'd 128 )  // black-white transition
      finish <= cntpx;                      // store pixel index
  end
  else
  begin
```

```
      DIAMETER <= (finish - start);   // compute the tube diameter
      cntpx <= 12'd0;                 // restart the pixel index counter
    end

endmodule
```

A much simpler solution (proposed by a student) computes the tube diameter as just the number of "black" pixels:

*i. counter **cntpx** increments when **FRAME**==1 and **PIXEL** < 128;*
*ii. when **FRAME**==0 the output **DIAMETER** is loaded with the current value of **cntpx**;*

A block diagram that implements this system is:



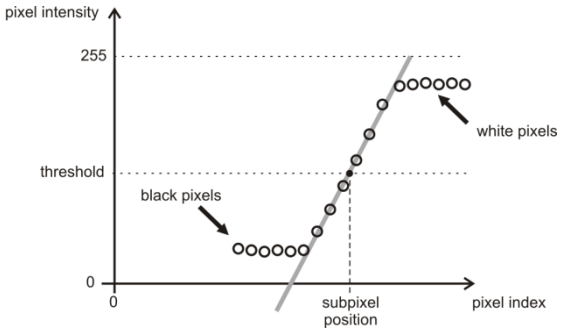
The Verilog code describing the block diagram is:

```
module Q5( input         PXCLK,
           input         reset,
           input  [7:0]  PIXEL,
           input         FRAME,
           output reg [11:0] DIAMETER
         );

reg [11:0] cntpx;          // the pixel index counter

// all in a single process:
always @(posedge PXCLK)
if ( reset )
begin
  cntpx  <= 12'd0;
  DIAMETER <= 12'd0;
end

else

begin
  if ( FRAME )
  begin
    if ( PIXEL < 128 )
      cntpx <= cntpx + 12'd1;  // pixel index count
  end
  else
  begin
    DIAMETER <= cntpx;         // compute the tube diameter
    cntpx <= 12'd0;            // restart the pixel index counter
  end
end

endmodule
```

[6 points]

6 - In the system described in question 5, it is required to implement an improved process to increase the accuracy of the diameter measurement. The solution is to determine the X-position of the transitions white-black and black-white with a sub-pixel resolution. The algorithm presented below can achieve more than 1/10 of pixel

i. select a set of N adjacent pixels around the threshold level (N = 6 in figure);
ii. use linear regression to calculate the best fit linear approximation to those points;
iii. calculate numerically the intersection between that line and the horizontal threshold line.



With the N selected points (xi,yi) (xi represents the index of the pixel, between 0 and 2047 and yi is the pixel value, between 0 – black - and 255 - white), the two parameter that describe the straight line m (slope) and b (Y-intersect), can be obtained by solving the two simultaneous equations:

$$\begin{cases} m\sum_{i=1}^{N} x_i & + & b\sum_{i=1}^{N} 1 & = & \sum_{i=1}^{N} y_i \\ m\sum_{i=1}^{N} {x_i}^2 & + & b\sum_{i=1}^{N} x_i & = & \sum_{i=1}^{N} x_i y_i \end{cases}$$

Is the x values are considered as relative to the position of the first selected pixel, and N=6, (obtained experimentally) the x values will always be equal to (0,1,2,3,4,5) and the previous equations will be simplified to:

$$\begin{cases} 15m & + & 6b & = & \sum_{i=1}^{N} y_i \\ 55m & + & 15b & = & \sum_{i=1}^{N} x_i y_i \end{cases}$$

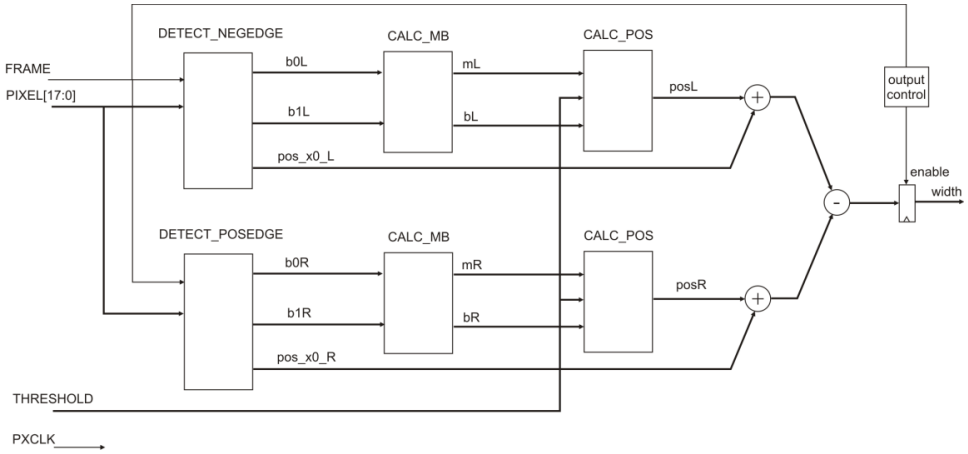
Using N = 6, $b_0 = \sum_{i=1}^{6} y_i$ e $b_1 = \sum_{i=1}^{6} x_i y_i$, the solution m e b can be calculated by the equations:

$$\begin{cases} m & = & \frac{6}{105} b_1 & - & \frac{15}{105} b_0 \\ b & = & \frac{-15}{105} b_1 & + & \frac{55}{105} b_0 \end{cases}$$

Finally, the fractional pixel position of the intersection between the best fit line and the threshold line, relative to the position of the first pixel in the set is given by:

$$pos = \frac{(\,threshold - b\,)}{m}$$

The system to implement can be represented by the simplified block diagram below. The blocks **DETECT_NEGEDGE** and **DETECT_POSEDGE** receive the pixels from the camera, select the 6 pixels around the white-black and black-white transitions and calculate the summations *b0* and *b1* in the outputs **b0*** and **b1***. These blocks also output the signal **pos_x0_*** representing the pixel index of the first pixel in the set of 6 selected for calculating **b0*** and **b1*** (this will be needed later to calculate the absolute position of the sub-pixel edge position). The 2 blocks **CALC_MB** receive **b0*** and **b1*** and compute the parameters **m*** and **b*** that define the best fit lines (for both edges). Finally, the blocks **CALC_POS** compute the fractional pixel position that must be added to the origin index for each set before subtracting them to calculate the tube diameter.



It is known that the image of interest (containing both tube edges) is always within 5mm of each side of the frame (5 mm = 255 pixel) and the tube has a nominal diameter equal to 25 mm (1316 pixel). Thus, between the white-black transition and the black-white transition there will be approximately 1316 clock cycles and between the detection of the black-white transition and the reception of the last pixel at least 255 clock cycles will occur.

[1.5 points]

a) Consider that the signals **m*** e **b*** are represented by 32-bit two's complement numbers, with 16 fractional bits. The block **CALC_POS** can be implemented with three different architectures for the divisor: combinational (created by the synthesis of the operator "/"), pipelined with 4 pipe-stages or sequential, producing the result in 34 clock cycles. Justify which is the best solution for the divider to use in this system.

The best solution is using a sequential divider. The system has to perform only two divisions to process a frame, for computing the sub-pixel position of each edge. We know that the second edge (transition black-white) will happen approximately 1300 clock cycles after the first edge and the end of the image will occur not before 255 clock cycles after the detection of that second edge. As we need 34 clock cycles to perform the division, the number of clock cycles available is significantly greater than the required to execute the division sequentially.

A combinational divider would be significantly larger than the sequential divider and will have a long propagation delay, possibly requiring some clock cycles (of the 24 MHz clock) to produce the result. Although a combinational divider may certainly complete the division in less time than the sequential divider, this does not represent any benefit in any terms.

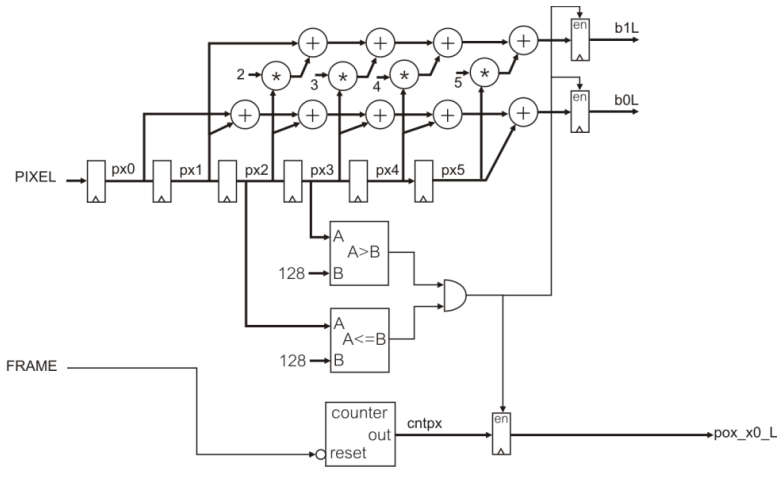
The same applies to a pipelined divider that is basically based on a combinational divider with internal register stages. Besides, as the divider has to compute only two divisions during a time period greater than 2048 clock cycles, there is no advantage in using pipelining, because pipelining is only advantageous when processing (long) sets of input data arriving at the clock rate.

[1.5 points]

b) Present a solution for the block **DETECT_NEGEDGE** and explain the modifications necessary to build the block **DETECT_POSEDGE**. The signals **b0L** and **b1L** correspond to the variables $b_0 = \sum_{i=1}^{6} y_i$ and $b_1 = \sum_{i=1}^{6} x_i y_i$ defined before (recall that the values xi are always equal to 0, 1, 2, 3, 4 e 5)

*Figure below shows a possible solution for this circuit. The 4 multipliers by the constants 2, 3, 4, 5 should be implemented as multipliers by constants and not generic combinational multipliers. The multiplications by 2 and 4 are just shifts (zero logic) and the multiplications by 3 and 5 only require one adder (x*3 = x + (x<<1) and x*5 = x + (x<<2)). This solution assumes that the propagation delay of the adder tree fits the clock period.*

To create the module DETECT_POSEDGE, only the A inputs of the comparators need to be swapped (A input of the (A>B) connects to px2 and A input of the (A<=B) connects to px3)



[1.5 points]

c) Considering that the design goal is to minimize the circuit area, present a block diagram or Verilog code for block **CALC_MB**.

*First, the constants shown in the equations as rational numbers must be represented as fixed-point fractional numbers, for example computing (x * 6/105) as (x * floor(6/105*2^K)) / 2^K (where K is the number of fractional bits used in the fixed point representation). Using K=16, the expression (x* 6/105) can be*

*computed as (x * 3744), where 3744 is 6/105*2^{16}, the 16 rightmost bits representing the fractional part. As the CALC_MB block will need to perform only one computation for each edge detected, the four multiplications could be implemented using a single sequential multiplier (each multiplier will need approximately 32 clock cycles, thus 128 clock cycles to execute the four multiplications). A 6-state FSM would be needed to apply the operands to the multiplier and compute the final results (state 0: wait for a start signal coming from the previous module; states 1 to 4: initiate and wait for completion of the four sequential multiplications; state 5: do the final addition and subtraction to compute the outputs m and b.*

Another possibility could be designing custom designed multipliers for each of the constants, implementing them as a sum of the non-zero partial products. For example, 3744 is represented in binary as 111010100000 and thus only 5 partial products are non-zero (only 4 adders needed - if using Booth recoding, only 2 adders and one subtractor would be needed: 111010100000 = 100$\underline{1}$01010000, where $\underline{1}$ represents a negative weighted bit). Note that this solution will possibly be more expensive than the previous proposal, based on a single sequential multiplier.

[1.5 points]

d) Still taking into consideration the area minimization design goal, suggest and justify a better (smaller) implementation for the circuit given above.

It is clear that the blocks CALC_MB and CALC_POS are duplicated as they will be required for computing the edges position at times separated by approximately 1300 clock cycles, equivalent to the nominal pixel distance between the two edges. Using a simple controller, multiplexers and registers it would be possible to use only two instances of those blocks. Also, the two edge detector modules (DEPECT_POSEDGE and DETECT_NEGEDGE) could be easily merged into a single module, as only the threshold crossing condition will be different for detecting each edge. Thus, a single datapath (DETECT_XEDGE, CALC_MB and CALC_POS) would be sufficient to compute the sub-pixel position for both edges. With this proposal, the circuit area will be reduced to approximately 50% (actually a bit more than...) of the original.

..:: the end ::..