Test 2: Lectures 17 to 22

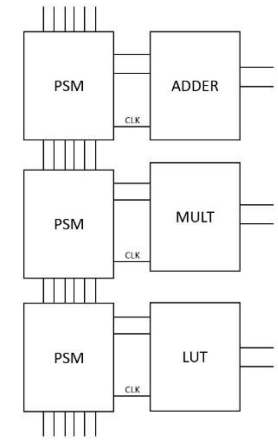
EEE4084F 2017-06-08

SOLUTIONS!!

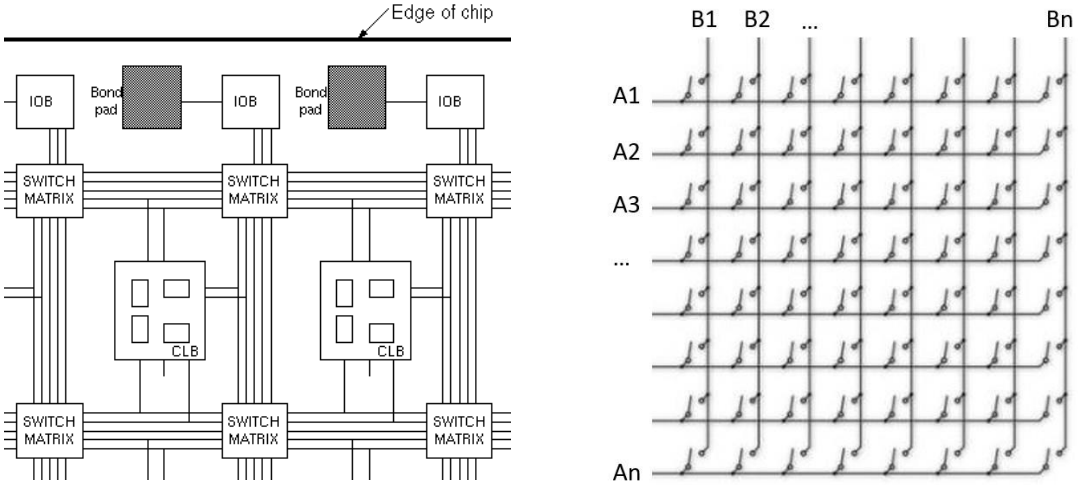
Question 1: Thoughts about programmable logic devices and FPGAs [10 Total]

#1 The register X configures the multiplexer to either connect the output of the LUT directly to the output wire of the CLB (this is unsynchronized behaviour) or to connect the LUT output via the FF to the output. Since the FF only stores the input value on a clock pulse, this makes the output of the CLB synchronized to the clock. Thus the combination of the register X and multiplexer is used to configure the CLB to be synchronized, to the clock, or unsynchronized. **[3]**

#2 Indeed it is a simplification, although not necessarily so for earlier (ancient in computing terms) FPGA devices. Nowadays FPGAs tend to be designed around supporting both DSP / more complex components and more general logic (e.g. gate structures and LUTs). The more complex components, such as 'hardcore' adders, multipliers and other processing components are provided as they are more efficient (in surface area) and may operate significantly faster than attempting to perform equivalent operations using only LUTs or more basic components. The diagram below illustrates the structure…



#3 A PSM (programmable switch matrix) is essentially constructed using multiplexers and flipflops (or registers) to store the configuration of the multiplexers which connect one signal line, of a set of possibilities lines, to the input or output of a CLB or to another set of lines which are routed further along by another PSM. The diagram on the left below shows how each switch matrix connectors from one bank of lines to another bank of lines, inside each switch matrix there are registers that store the configuration of how the banks of lines are connected. The illustration on the right below shows how the switch matrix connects lines in the one bank of lines (say bank A) to another bank (say bank B) – this is a simplified view as these connections would actually be implemented using multiplexers (or a more optimized switch element). (The switch matrix structure is obviously instrumental to the flexibility of the FPGA but also to a large extent responsible performance limitations). **[3]**



Question 2: Thoughts about computing devices [10 Total]

This question concerns Lecture 17 and Chapter 13.

#1 Arithmetic operations typically carry out the main computation needs, such as performing calculations; often these computations are more complex and could require more clock cycles than is the case or more general operations, such as swapping values between registers. It is thus more accurate to base the performance of a DSP processor, which is likely going to carry out mainly arithmetic operations, based on the processor's performance of executing arithmetic operations. It is expected that branch operations and load/store operations (which could also take a couple of clock cycles) would occur less frequently than arithmetic operations, e.g. there is likely to be a chain of arithmetic operations for each input that must be completed within a particular time. **[4]**

#2

Battery provides 4W for 1h (i.e. 4Wh capacity). This is for the processor operating at its maximum clock speed of 100MHz at 3.3V. We may likely need to know what the gate capacitance (C) of the processor is, which we can work out from the formula $P=C \cdot f \cdot V^2$ as follows:

$P=C \cdot f \cdot V^2$

$4 = C \times 100\,000\,000 \times 3.3 \times 3.3$

$C = 4 / (100\,000\,000 \times 10.89)$

$C = 3.673 \times 10^{-9} = 3.673$ nF

The proposed strategy to cut down the power consumption is to reduce the clock rate to the minimum (that would be setting f to 20MHz) and lowering the volatage also to the minimum that the processor supports, which would be down to 1.2V. With this configuration the total power usage would be

$P = C \cdot f \cdot V^2$

$P = 3.673 \times 10^{-9} \times 20 \times 10^{6} \times (1.2)^2$

$P = 5.2891 \times 10^{-3} \times 20$

$P = 0.105$ W

Wow!! That is a major saving for running at 20MHz at only 1.2V (assuming I got the calculation right).

Then to calculate how long it would run for…

4Wh / 0.105 W = 38h

Which is hugely longer than the case of operating at full speed.

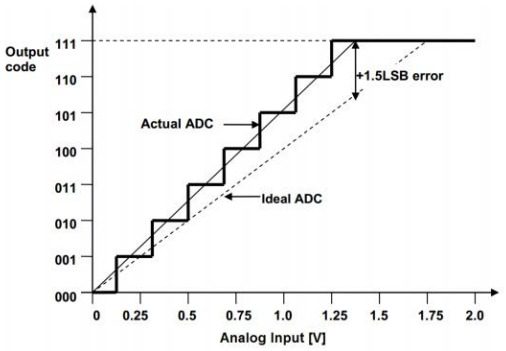
[6]

Question 3: Thoughts about computing devices [10 marks]

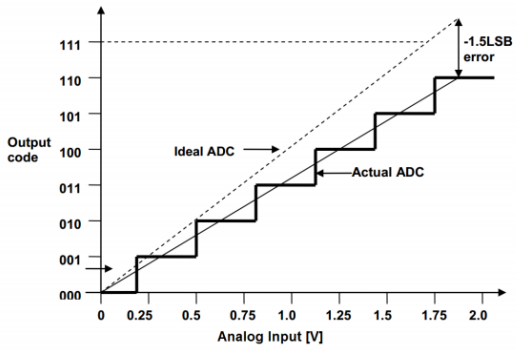
This question relates to ADCs covered in Seminar 6 / Chapter 7.

#1 Explain the difference between Positive Gain Error (PGE) and Negative Gain Error (NGE). You can use diagram(s) to illustrate your answer.

PGE and NGE are occur when the actual transfer function slope deviates from the ideal slope. In the case of PGE, the actual transfer slope is successively at a level above the ideal transfer slope, i.e. giving a digitized output that keeps overshooting what the value should be. NGE is the reverse, the actual transfer slope is successively at a level below the ideal transfer slope, i.e. giving a digitized output that keeps undershooting what the value should be. This is illustrated below (using the relevant figures form the textbook).



Positive Gain Error (PGE)



Negative Gain Error (NGE)

[5]

#2 For a 10-bit ADC for a voltage range -2V to 2V, or 4V range, the voltage differential corresponding to a change of the LSB is 4 / 1024 = 0.0039 V. So I can safely conclude that:

YES, the ADC should deliver the required accuracy.

[5]

Question 4: Of interfacing and HDL [20 marks]

Example available at https://www.edaplayground.com/x/5LaK

PacketRecv.v

```
// Implementation of the PacketRCV module
module PacketRCV (
  input [7:0]DIN, input RDY, input SOP, input EOP, input RESET, input CLK,
  output reg CSYNC, output reg PTO, output reg AHP
  );
  reg startr;
  reg [7:0]PIDX;
  reg [7:0]PKT[1023:0];
  // Handle any reset
  always@(posedge CLK or posedge RESET)
   begin
    if (RESET == 1'b1)
       begin
        $display("RESET!");
        // Clear CSYNC and PTO
        CSYNC = 1'b0;
        PTO   = 1'b0;
        AHP   = 1'b0;
        startr= 1'b0;
        PIDX  = 8'd0;
        AHP   = 1'b0;
       end
   end
  // Check for other signals
  always@(posedge SOP)
  begin
   if (startr == 1) AHP = 1'b1; // exception due to
   startr = 1'b1;
   PIDX   = 8'd0;
   CSYNC  = 1'b1;
   PTO    = 1'b0;
  end

  always@(posedge RDY)
  begin
```

```
   if (startr == 0)
      begin
       // unexpected RDY, packet not started
       $display("-- unexpected RDY");
       AHP    = 1;
       startr = 1;
       PKT[0] = DIN;
       PIDX   = 1;
      end
      else
      begin
       PKT[PIDX] = DIN;
       PIDX = PIDX + 1;
       CSYNC = 0;
      end
  end

  always@(posedge EOP)
  begin
   if (startr == 0)
      begin
       AHP = 1;  // unexpected EOP, packet not started
      end
      else
      begin
       startr = 0;
       PTO    = 1; // indicate successful end of packet
      end
  end

endmodule
```

Testbench for PacketRecv

```
// Testbench for PacketRCV
module PacketRCV_tb;
  // declare registers for the data and control lines
  reg [7:0]DIN;
  reg RDY, SOP, EOP, RESET, CLK;
  wire CSYNC, PTO, AHP;

initial
  begin
    $display("PacketRCV Test!");

    // monitor some of the signals
    $monitor("CLK=%b  RESET=%b  RDY=%b  SOP=%b  EOP=%b  CSYNC=%b  PTO=%b  AHP=%b
DIN=%d",CLK,RESET,RDY,SOP,EOP,CSYNC,PTO,AHP,DIN);
    // finish after 100 clocks
    CLK = 0;
    RESET = 1;
    RDY   = 0;
    #5  CLK = ~CLK;
    RESET = 0;
    #5  CLK = ~CLK;
    SOP = 1;
    #5  CLK = ~CLK;
    SOP = 0;
    #5  CLK = ~CLK;
    DIN = 100;
    #5  CLK = ~CLK;
    RDY = 1;
    #5  CLK = ~CLK;
    RDY = 0;
    #5  CLK = ~CLK;
    DIN = 101;
    #5  CLK = ~CLK;
    RDY = 1;
    #5  CLK = ~CLK;
    RDY = 0;
```

```
    #5  CLK = ~CLK;
    EOP = 1;
    #5  CLK = ~CLK;
    EOP = 0;
    #10

    $display("Test fault..");
    DIN = 99;
    #5  CLK = ~CLK;
    RDY = 1;
    #5  CLK = ~CLK;
    RDY = 0;
    EOP = 1;
    #5  CLK = ~CLK;
    EOP = 0;


    #50 $finish;
  end

  // instantiate the module
  PacketRCV rcv(DIN,RDY,SOP,EOP,RESET,CLK,CSYNC,PTO,AHP);

endmodule
```