Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 – Introductory Digital Systems Laboratory

**Problem Set 3**

**Comments:** This problem set is intended to prepare you for Laboratory 3 which is a Finite Impulse Response (FIR) filter. Problem 1 is intended to familiarize you with the concepts of convolution and filtering. While we do not expect you to have a background in signals (6.003 is not a prerequisite) we would like for you to have an appreciation for how digital filtering works.

**Problem 1: Convolution**
An FIR filter uses convolution to generate the data points for the filtered output. Convolutions are a weighted accumulation of sampled points from a signal. Given that h[n] is the Finite Impulse response for a specific filter and x[n-k] is the $k^{th}$ recent sample, the equation that best describes the convolution we are going to implement is

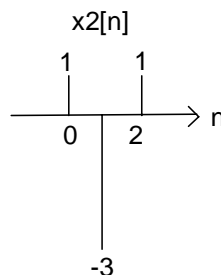$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k]$$

where $N$ is the number of points in the convolution. Given the filter shown below compute the convolution with the following inputs
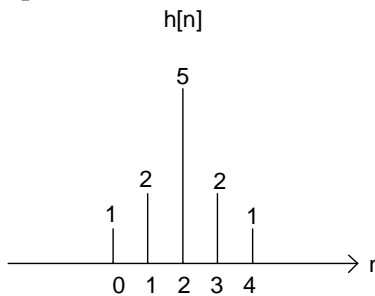


**Part (a)**
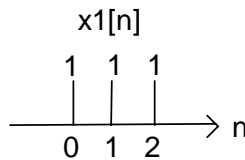


**Part (b)**

**Part (c)** Compute and plot the convolution of x[n] and h[n] where

$$x[n] = \delta[n] + 2\delta[n-1] - \delta[n-3] \text{ and } h[n] = 2\delta[n+1] + 2\delta[n-1].$$

**Part (d)** Given a 10-tap (delay) filter where each of the filter coefficients is represented using 8 bits and the input x[n] is represented using 8 bits. Assume that a single accumulator is used to add the product terms from this filter.

How many bits (i.e., the width of the adder) should the adder output be?

## Problem 2: Twos Complement Multiplier

In Lecture #8 a 4x4 Twos Complement Multiplier was presented. This problem asks you to design an 8x8 Twos Complement Multiplier in Verilog and validate your design using a testbench.

Design an 8x8 multiplier which takes as input the two's complement numbers X [7:0] and Y [7:0] then multiplies them together to produce a twos complement output Z.

Code the multiplier you designed above in Verilog and validate it using a testbench.

For this problem turn in Verilog code for your multiplier and testbench. We also ask that you submit a screen capture of your simulation.

## Problem 3: Critical Path Timing Analysis

The Figure below is the 16-bit Carry-Bypass Adder from Lecture 8 (see notes for a clear block diagram).
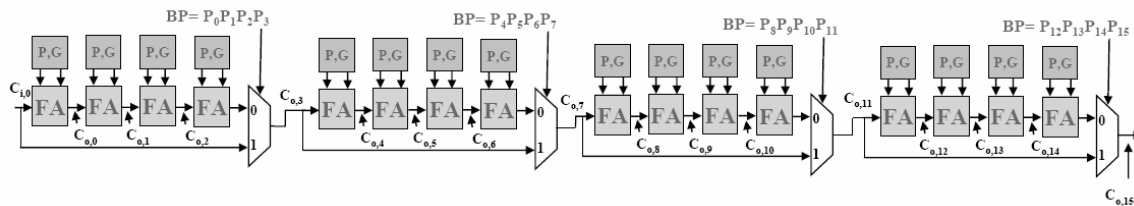


**Figure 1: Carry-bypass adder**

Assume the following delay for each gate:

> Producing $P_i$, $G_i$ from $A_i$, $B_i$: 1 unit
> $P_i$, $G_i$, $C_i$ to $C_o$ or Sum for a FA: 1 unit
> 2:1 mux delay: 1 delay unit
> BP: It takes 1 delay unit to generate BP from the propagate signals.

What is the worst case propagation delay for the 16-bit adder?

**Problem 4: FIFO Design and Major/Minor FSMs**
In this problem you will create a RAM and FIFO controller using Altera's parameterized modules and simulate them using Max+PlusII.

**Part (a)** Create a RAM module in Max+PlusII using the Megawizard Plugin Manager. (You may want to refer to the online help at http://web.mit.edu/6.111/www/s2004/software.html). Instantiate an "LPM_RAM_DQ" (separate input and output bus) with an 8 bit data bus and 4 bit address bus. Let the data and address inputs be registered.

The Megawizard will create a few files. One of them will be a .v file that contains the module definition. Open the file and examine its contents. Notice that there is no output enable and that write enable uses positive logic.

**Part (b)** Set the device settings for the FPGA on your lab kit. Compile the RAM module. Simulate this file in MAX+PlusII using the waveform editor and the simulator.

Lastly, we will create a memory controller with first in first out (FIFO) behavior. The system block diagram looks as follows:
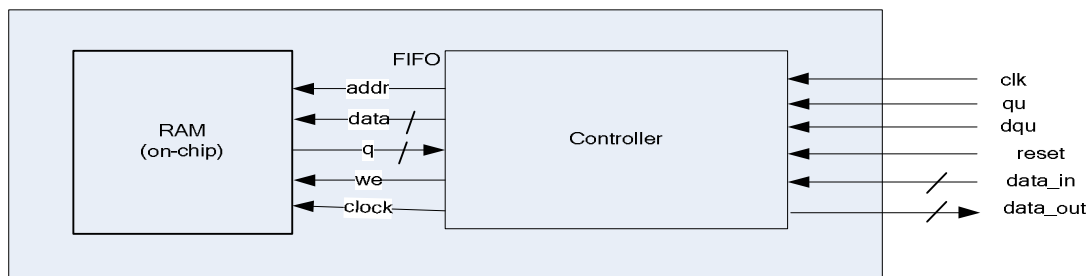


**Figure 2: System block diagram of FIFO memory unit.**

The RAM is the parameterized module you created earlier. You will design the controller. When qu = 1, the value of data_in is put at the tail of the FIFO. When dqu = 1, the value at the head is removed and placed on data_out. Overflow and boundary cases will be the responsibility of the user (i.e. you don't have to worry about removing data from an empty FIFO or adding a new entry to a full FIFO).

**Part (c)** First create an FSM diagram showing your states, outputs and transitions. Remember that the RAM has separate input and output buses and its output is constantly enabled. Your FSM should be substantially simpler than the one for the 6264 SRAM controller.

**Part (d)** Implement the FSM in Verilog and simulate in MAX+PlusII to test its functionality. The following code should start you off.

```
module FIFO_controller (clk, reset, qu, dqu, data, out);
  input clk, qu, dqu, reset;
  input [7:0] data;
  output [7:0] out;
  reg [7:0] out_ram, out;
  reg [3:0] addr;
  reg we;
```

```
    /* Your parameters and internal variables here*/

    ram_from_part1 myram (
        .address(addr),
        .inclock(clk),
        .we(we),
        .data(data),
        .q(out_ram)
    );
    /* Your FSM code here */
endmodule
```

Submit all code and waveforms.