Prelab 9
Joaquin Salas

1. With dataflow Verilog, describe the Generate/Propagate Unit, the Carry-Lookahead Unit, and the Summation Unit in Figure 1 as separate modules. Do not include delays in your models. We will add them later in the lab experiments. Use the module interfaces below as a guide. Gate-level schematics can be hard to read so you may find expressions (3) through (11) easier to follow. Note: If you are unsure of what dataflow Verilog looks like, consult lab 5!

```
`timescale 1ns/1ps
`default_nettype

/*This simple module will demonstrate the concept of clock frequency*
*division using a simple counter. We will use behavioral Verilog    *
*for our circuit description.                                       */

module clock_divider(ClkOut, ClkIn);

    output wire [3:0] ClkOut;
    input wire ClkIn; //wires can drive regs

    /*-this is a keyword we have not seen yet! *
    * as the name implies, it is a parameter   *
    * that can be changed at compile time...   */
    parameter n = 5; //make count 6-bits for now...

    reg [n:0] Count; //count bit width is based on n! how cool is that!

    /* simple behavioral construct to describe a counter...    */
    always@(posedge ClkIn) //should look familiar...
        Count <= Count + 1; //worlds shortest always block

    assign ClkOut[3:0] = Count[n:n-3];

endmodule
```

2. Now, use structural Verilog along with the modules you have just created to wire up a 4-bit Carry-Lookahead adder. The module interface you should use is provided below.
Note: If you are unsure of what structural Verilog looks like, consult lab 5!

```
`timescale 1ns/1ps
`default_nettype

module top_level(LEDs, SWs, BTNs, FastClk);
```

```
  //all ports will be wires
  output wire [3:0] LEDs;
  input wire [1:0] SWs, BTNs;
  input wire FastClk;
  //Fast Clk is 1 bit wire
  //intermediate nets
  wire [3:0] Clocks;
  reg SlowClk; //will use an always block for mux
  //behavioral 4 way MUX
  always@(*) //combinational logic
    case(SWs)
      2'b00: SlowClk = Clocks [0];
      2'b01: SlowClk = Clocks [1];
      2'b10: SlowClk = Clocks [2];
      2'b11: SlowClk = Clocks [3];
    endcase
  //use up counter to connect to LEDs
  up_counter COUNTER0(LEDs[2:0], LEDs[3], BTNs[0], SlowClk, BTNs[1]);
  //use clock divider
  clock_divider clk_div0(
    .ClkOut(Clocks),
    .ClkIn(FastClk)
  );
endmodule
```

. The previous problems were concerned with a single-level 4-bit carry-lookahead adder. In one of the
lab experiments, we will construct a 16-bit, 2-level carry-lookahead adder. The following questions
will prepare you for this exercise. What is the propagation delay of the 16-bit, 2-level carry-lookahead
adder in Figure 2? Likewise, what is the gate-count?
`timescale 1ns / 1ps //specify 1 ns for each delay unit

```
`default_nettype none

// this module describes a simple 4-bit up counter using
//  half-adder modules built in the previous step

module up_counter(Count, Carry3, En, Clk, Rst);

    output reg [3:0] Count;
    output wire Carry3;
    input wire En, Rst, Clk;

    //intermediate nets
```

```
    wire [3:0] Carry, Sum;

    half_adder halfadder0(Sum[0], Carry[0], En, Count[0]);
    half_adder halfadder0(Sum[1], Carry[1], Carry[0], Count[1]);
    half_adder halfadder0(Sum[2], Carry[2], Carry[1], Count[2]);
    half_adder halfadder0(Sum[3], Carry[3], Carry[2], Count[3]);

    assign Carry3 = Carry[3]; //wire up carry3

    //positive edge triggered flip-flops for count
    //posedge Rst implies asynchronous reset
    always@(posedge Clk or posedge Rst)
        if(Rst)
            Count<=0; //reset the count
        else
            Count<=Sum; //otherwise, latch sum
endmodule
```