*CS/EE 120A*
*(Section: 021)*

# Lab #5:
# Datapath Components - Adders

**Dan Murphy; SID: 862xxxxxx**
**dmurp006@ucr.edu**

Partner:
Jesse Garcia

## Subject Overview

In order to efficiently learn and implement adders via data-path components, the lab was separated into two parts: A and B.

**Part A:**

Within this section, our goal was to create a schematic of a timer system. This particular timer system included a controller and a comparator. An input of 1 is taken when the button is pressed then outputs to the comparator. When the input from the controller is identical to the [reset], (which is initialized to 1), an output will be present in addition to the counter being reset. The schematic simulated the appropriate, respective behaviors based on input into the controller.

**Part B:**

Within this section, we created the transitions for the Finite State Machine: START, ON, & OFF. In addition to this implementation, a UCF file was created to map inputs onto the FPGA board. With the Boolean transitions in place, START will turn ON the machine until the timer is set to 1. The machine is OFF until a button is pressed.

## New Concepts

1. Describe how this lab built upon previous ones:

*This lab built upon previous labs as we have to understand how implement adders into Verilog and further grasp the concept of utilizing clocks.*

2. Describe the most difficult part of this lab for you:

*By far, the most difficult part was decrypting the errors given to us through the IDE. This was my partner's first time leading himself through a lab within Verilog. Due to this being his first time, it took a lot of explanation to show him what to do and allow him to understand not only what he was doing but also why he was doing it.*

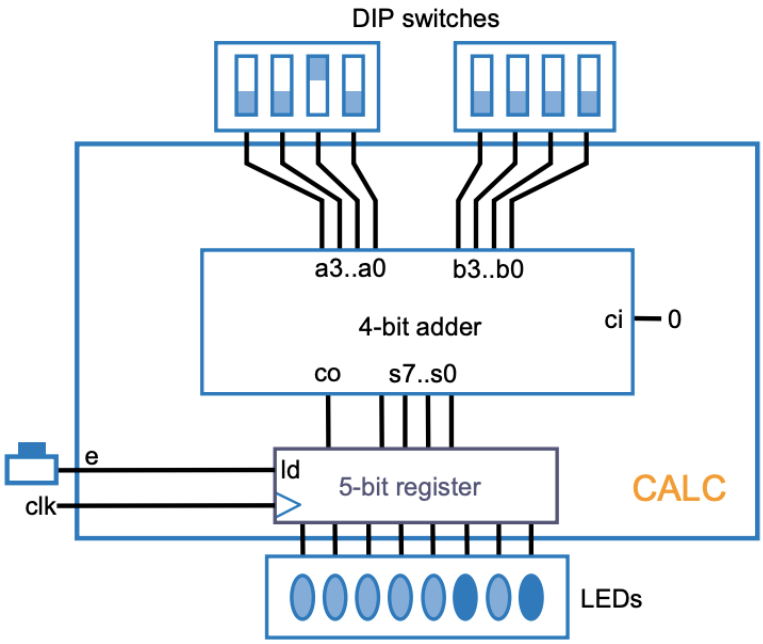
3. Describe problems you faced and how you solved them:

*Fortunately, this lab was straight-forward enough to follow the directions and not have any issues. However, my partner, Jesse Garcia, has not been actively performing the labs – so I forced him to learn how to program in Verilog and demo the results after walking him through the process step-by-step. I have completed the lab myself prior to showing Jesse how to do the lab during our lab period, which helped me efficiently communicate directions [better] than if I had not done the lab beforehand.*

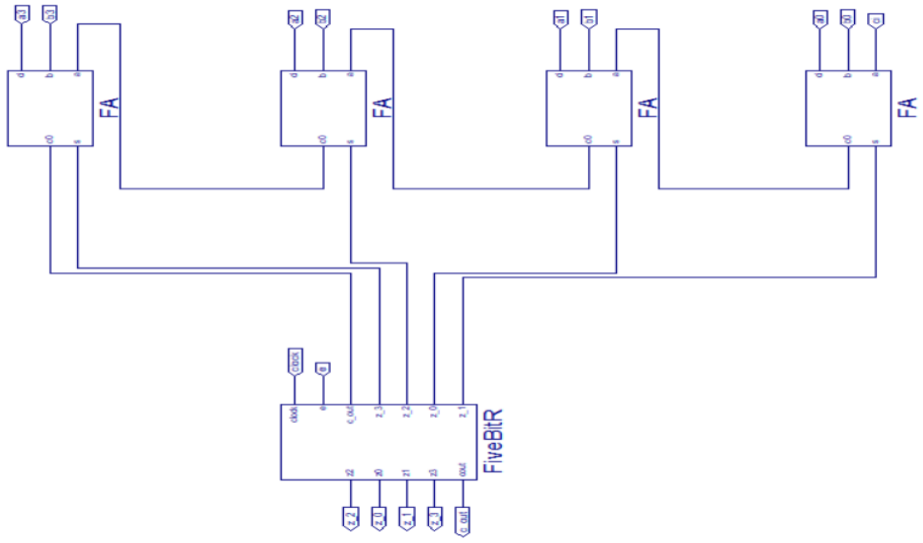
4. Do you verify that the code included with this report is yours and your partner's original work (yes/no)?

*Yes - our code is inherently original.*

## Specification




## Implemented Schematic



## Verilog Code Implemented

```
// Module Instantiation ---------
module carrylookahead_st(
        input clock ,
        input enable ,
        input cin,
        input [3:0] x,
        input [3:0] y,
        output cout,
        output [3:0] r
 );
// Wires ------------------------
        wire [3:0] c;
        wire [3:0] ir1 ;
        wire [4:0] ir2 ;

// Compute Carries ---------------
carrylogic cx5 ( c, cin, x, y) ;

// Compute R ---------------------
falogic cx6 ( ir1[0], x[0], y[0], cin ) ;
falogic cx7 ( ir1[1], x[1], y[1], c[0] ) ;
falogic cx8 ( ir1[2], x[2], y[2], c[1] ) ;
falogic cx9 ( ir1[3], x[3], y[3], c[2]) ;

// Register logic ----------------
register_logic cx10 ( clock, enable, {c[3],ir1}, ir2 ) ; // 1'b1
// Results
assign r[0] = ir2[0];
assign r[1] = ir2[1];
assign r[2] = ir2[2];
assign r[3] = ir2[3];
assign cout = ir2[4];

// carry logic --------------------
module carrylogic(
output [3:0] cout ,
input cin,
input [3:0] x,
input [3:0] y
 );
```

```
// Computing all gx ---------------
wire g0, g1, g2, g3 ;
assign g0 = x[0] & y[0] ;
assign g1 = x[1] & y[1] ;
assign g2 = x[2] & y[2] ;
assign g3 = x[3] & y[3] ;

// Computing all px --------------
wire p0, p1, p2, p3 ;
assign p0 = x[0] | y[0] ;
assign p1 = x[1] | y[1] ;
assign p2 = x[2] | y[2] ;
assign p3 = x[3] | y[3] ;

// Computing all bit carries ----------------------------------
assign cout[0] = g0 | ( p0 & cin) ;
assign cout[1] = g1 | ( p1 & ( g0 | (p0 & cin) ) ) ;
assign cout[2] = g2 | (p2 & (g1 | ( p1 & ( g0 | (p0 & cin) ) )));
assign cout[3] = g3 | (p3 & (g2 | (p2 & (g1 | ( p1 & ( g0 | (p0 &
cin) ) )))));
endmodule // END -----------------


// module register ---------------

module register_logic(
// Inputs ---
 input clk,
 input enable ,
 input [4:0] Data ,
// Output ---
 output reg [4:0] Q ) ;

 always @(posedge clk )
 begin
 if ( enable) begin
 Q = Data;
 end // End of local scope
 end // End of local scope
endmodule // END -----------------
```

```
// fa logic ----------------------
module falogic(
output r,
input x,
input y,
input cin
 );

// Boolean logic ------
xor cx1 ( t1, x,y );
xor cx2 ( r, t1, cin );
endmodule // END -----------------
```

## UCF File

```
// Inputs ------
NET “clk” LOC = “B8” ;
NET “b”   LOC = “G12” ;

// Output ------
NET “light” LOC = “M5” ;
```

## Conclusion

In this lab, we learned how to implement a full adder into Verilog. This was performed by writing the logic out in Verilog through the carry unit and the four-bit carry look ahead module. In essence, the adder adds to four-bit numbers together while carrying leftover bits.

As a whole, the circuit design and Boolean logic allowed us to see the implementation of the adder in Verilog and thus onto our FPGA BASYS board. Once the results expected were yielded, our component system was successfully completed.