Remi de Ferrieres
X664611
CS 120A Section 021
Lab 5 – Datapath components – Adder
Yann Abou Jaoude
X663581

OVERVIEW

In this lab we created a Full adder using structural Verilog and tested it. For this in the Verilog we had to write down components. One component to implement the logic of a full adder and a N-bit register and another to implement the logic of the carry unit. From there we also created a 4-bit carrylookahead module using the previous components created.

ANALYSIS

Source code:

```
module falogic(
   output r,
   input x,
   input y,
   input cin
    );
   xor cx1 ( t1, x,y );
   xor cx2 ( r, t1, cin );
endmodule

module register_logic(
    input clk,
    input reset,
    input enable ,
    input [4:0] Data ,
    output reg [4:0] Q ) ;
    always @(posedge clk )
    begin
    if (reset) begin
    Q = 5'd0;
    end
    if (enable) begin
    Q = Data;
    end
 end
endmodule

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

//module carrylookahead_st(

//endmodule

module Lab_5_2_VM(
    input clk ,
    input reset,
    input enable ,
    input cin,
    input [3:0] x,
    input [3:0] y,
    output cout,
    output [3:0] r
    );
   wire [3:0] c;
   wire [3:0] ir1 ;
   wire [4:0] ir2 ;

   // Compute Carries
   carrylogic c1(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
   register_logic cx10 ( clk, reset, enable, {c[3],ir1}, ir2 ) ;
   // Results
   assign cout = c[3];
   assign r[0] = ir1[0];
   assign r[1] = ir1[1];
   assign r[2] = ir1[2];
   assign r[3] = ir1[3];


endmodule
```

UCF file :

```
//Inputs
NET "clk" LOC = "B8" ;
NET "cin" LOC = "A7" ;
// Xs and Ys
NET "x[0]" LOC = "N3" ;
NET "x[1]" LOC = "E2" ;
NET "x[2]" LOC = "F3" ;
NET "x[3]" LOC = "G3" ;
NET "y[0]" LOC = "B4" ;
NET "y[1]" LOC = "K3" ;
NET "y[2]" LOC = "L3" ;
NET "y[3]" LOC = "P11" ;
// Outputs
NET "r[0]" LOC = "G1" ;
NET "r[1]" LOC = "P4" ;
NET "r[2]" LOC = "N4" ;
NET "r[3]" LOC = "N5" ;
NET "cout" LOC = "P6" ;
```

DISCUSSION:

The works according to provided specifications. The main difficulty was to understand what was given to us. The statement gave us a lot of code already written. Sometimes we used parts of code that we did not understand at the moment. To make our system more useful we can change our adder so that it adds Two's complement. In this case, we cant use a carry-in.

CONCLUSION:

Today we learned to design an adder. More than that we have consolidated our knowledge. We can now manipulate variables, create systems.

QUESTIONS:

Is it possible for two 4-bit numbers and a carry-in to result in a number too big to represent using 4 sum bits and a carry-out bit? The bigger number you can have by adding two 4-bit numbers and a carry-in is 1 1111. 1 1111 in 4 sum bits and a carry-out bit. So, there is no number too big to represent using 4 sum bits and a carry-out bit.