We learned how to design a calculator that’s able to add 2 4 bit numbers. We created a look-ahead adder that used a separate module for calculating the carry bits which was fed into the full adder.

We created our own structural model to test the full adder. We implemented 4 components; 1 for the logic of a full adder, 1 for an n-bit register, one for the logic of the carry unit, and one for a 4 bit carry adder look-ahead structure model that uses the other components. We also implemented a 4 bit adder system on the Basys Board. We used 2 4 bit position DIP switches for the 2 4 bit inputs, a single switch for carry-in inputs and 5 LEDs for the given outputs.

Equations

Ci+1

Gi + pici

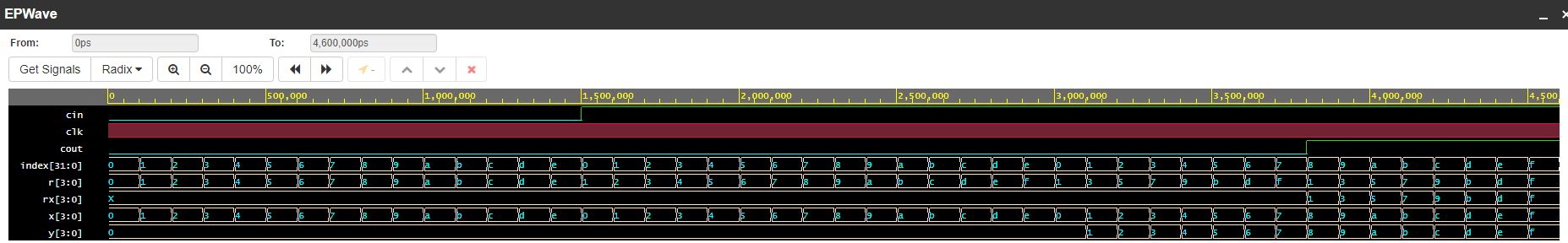
Gi = (xi & yi)

Pi = (xi |yi)

Purpose of this lab was to implement a calculator as an application of FPGA whilst using a different design to avoid the delays from carrying ripple adders.

Is it possible for 2 4-bit numbers and a carry in to result in a number too large to represent using 4-sum bits and a carry out bit?

Yes because overflow occurs when an arithmetic operation is too large or small.



| ///////////////////////////////////////////////////////////////////////////  /////  ///////////////////////////////////////////////////////////////////////////  /////  `timescale 1ns / 1ps  module carrylookahead\_tb;  // Inputs  reg cin;  reg [3:0] x;  reg [3:0] y;  reg clk;  // Outputs  wire cout;  wire [3:0] r;  reg [3:0] rx;  integer index ;  //always @(\*) // no sensitivity list, so it always executes  // begin  // clk = 1; #100; clk = 0; #100; // 10ns period  // end  // Instantiate the Unit Under Test (UUT)  carrylookahead\_st uut (  .clk(clk),  .cin(cin),  .x(x),  .y(y),  .cout(cout),  .r(r)  );  initial begin  $dumpfile("dump.vcd"); $dumpvars;  // Initialize Inputs  cin = 'd0;  y = 'd0;  // r = x + 0 ; cout = 0;  $display("TC11 ");  for (index=0; index < 15; index = index + 1) begin  x = index ;  #100;  if ( r != x ) $display ("Result is wrong");  if ( cout != 1'b0 ) $display ("Result is wrong - Carryout ");  end  // r = x + 1 ;  cin = 1'b1;  y = 4'b0;  $display("TC12 ");  for (index=0; index < 15; index = index + 1) begin  x = index ;  #100;  if ( r != (x + 'd1) ) $display ("Result is wrong %b %b" , r, (x+1) );  if ( cout != 1'b0 ) $display ("Result is wrong - Carryout ");  end  // r = x + y + 1 ;  cin = 1'b1;  $display("TC13 ");  for (index=0; index < 8; index = index + 1) begin  x = index ;  y = index ;  #100;  if ( r != (x + y +1 ) ) $display ("Result is wrong %b %b" , r, (x+y) );  if ( cout != 1'b0 ) $display ("Result is wrong - Carryout ");  end  // r = x + y + 1 ;  cin = 1'b1;  $display("TC14 ");  for (index=8; index < 16; index = index + 1) begin  x = index ;  y = index ;  rx = x + y + cin ;  #100;  if ( r != rx ) $display ("Result is wrong %b %b" , r, rx );  if ( cout != 1'b1 ) $display ("Result is wrong - Carryout ");  end  end  endmodule |
| --- |
| module falogic(  output r, // We label our output as r instead of z  input x,  input y,  input cin  );    wire t1;  xor cx1 ( t1, x,y );  xor cx2 ( r, t1, cin );    endmodule  module register\_logic(  input clk,  input enable,  input [4:0] Data,  output reg [4:0] Q );      //on real FPGA board which has clk signal, we use the following always statement:  // always @(posedge clk )  // begin  // if ( enable) begin  // Q = Data;  // end  // end  // endmodule  // for simulation, we force the statement to execute without clk signal:    always @(\*) begin  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 & (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    module carrylookahead\_st(  input clk ,  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 cx1 ( c, cin, x, y ) ;    // Compute R  falogic cx6 ( ir1[0], x[0], y[0], cin) ;  // Your code (3 more full adders)  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, 1'b1, {c[3],ir1}, ir2);    // Results  assign r = ir2[3:0] ;  assign cout = ir2[4] ;    endmodule |