**EE 310 – Lab 2 Report**

**NAU, 14 February 2020**

**By Alenn Wright**

**Problem Description**

In this lab, we have been asked to *design an add/subtract accumulator. Accumulators are very commonly used state machines in all fields of electrical and computer engineering. Accumulator has two main inputs (in addition to the usual rst and clk):*

* 1-bit input S: To select the accumulation direction – S=0 for “add”, S=1 for “subtract”,*

* N-bit input V: Value to be accumulated.*

*It also has an internal accumulation register (acc), which is “zeroed” upon reset. Then at every clock, the state machine checks if S=0 and adds value of V onto the current value of accumulation register, otherwise (if S=1) it subtracts V from the current value of accumulation egister.*

|  |
| --- |
|  |

Figure 1. Expected behavior of the circuit

**Solution Plan**

In order to solve the problem explained above, I have *decided to make 2 methods. One method will do the state math whenever the clock reaches a positive edge. Then If Acc changes I will have a seperate method with a switch case built in to decide the the proper code for the 7-segment displays.*

|  |
| --- |
| same as before |

Figure 2. State diagram for the proposed solution

**Implementation and Test Plan**

I have implemented the solution plan explained above, by *the code is exactly what I set out to do the only difference was that I had an issue over getting the states to work. When initializing the variables I found that I initialized them incorrectly and that they did not work. After they were resolved the code worked flawlessly.*

|  |
| --- |
| //--------------------------------------------------------------------------//  // Title: baseline\_pinout.v //  // Rev: Rev 1.0 //  // Last Revised: 10/13/2015 by Geraldine Baniqued //  //--------------------------------------------------------------------------//  // Description: Baseline design file contains Cyclone V GX Starter Kit //  // Board pins and I/O Standards. //  //--------------------------------------------------------------------------//  //Copyright 2012 Altera Corporation. All rights reserved. Altera products  //are protected under numerous U.S. and foreign patents, maskwork rights,  //copyrights and other intellectual property laws.  //  //This reference design file, and your use thereof, is subject to and  //governed by the terms and conditions of the applicable Altera Reference  //Design License Agreement. By using this reference design file, you  //indicate your acceptance of such terms and conditions between you and  //Altera Corporation. In the event that you do not agree with such terms and  //conditions, you may not use the reference design file. Please promptly  //destroy any copies you have made.  //  //This reference design file being provided on an "as-is" basis and as an  //accommodation and therefore all warranties, representations or guarantees  //of any kind (whether express, implied or statutory) including, without  //limitation, warranties of merchantability, non-infringement, or fitness for  //a particular purpose, are specifically disclaimed. By making this  //reference design file available, Altera expressly does not recommend,  //suggest or require that this reference design file be used in combination  //with any other product not provided by Altera  //----------------------------------------------------------------------------  //`define ENABLE\_DDR2LP  //`define ENABLE\_HSMC\_XCVR  //`define ENABLE\_SMA  //`define ENABLE\_REFCLK  //`define ENABLE\_GPIO  module baseline\_c5gx(  ///////// ADC ///////// 1.2 V ///////  output ADC\_CONVST,  output ADC\_SCK,  output ADC\_SDI,  input ADC\_SDO,  ///////// AUD ///////// 2.5 V ///////  input AUD\_ADCDAT,  inout AUD\_ADCLRCK,  inout AUD\_BCLK,  output AUD\_DACDAT,  inout AUD\_DACLRCK,  output AUD\_XCK,  ///////// CLOCK /////////  input CLOCK\_125\_p, ///LVDS  input CLOCK\_50\_B5B, ///3.3-V LVTTL  input CLOCK\_50\_B6A,  input CLOCK\_50\_B7A, ///2.5 V  input CLOCK\_50\_B8A,  ///////// CPU /////////  input CPU\_RESET\_n, ///3.3V LVTTL  `ifdef ENABLE\_DDR2LP  ///////// DDR2LP ///////// 1.2-V HSUL ///////  output [9:0] DDR2LP\_CA,  output [1:0] DDR2LP\_CKE,  output DDR2LP\_CK\_n, ///DIFFERENTIAL 1.2-V HSUL  output DDR2LP\_CK\_p, ///DIFFERENTIAL 1.2-V HSUL  output [1:0] DDR2LP\_CS\_n,  output [3:0] DDR2LP\_DM,  inout [31:0] DDR2LP\_DQ,  inout [3:0] DDR2LP\_DQS\_n, ///DIFFERENTIAL 1.2-V HSUL  inout [3:0] DDR2LP\_DQS\_p, ///DIFFERENTIAL 1.2-V HSUL  input DDR2LP\_OCT\_RZQ, ///1.2 V  `endif /\*ENABLE\_DDR2LP\*/  `ifdef ENABLE\_GPIO  ///////// GPIO ///////// 3.3-V LVTTL ///////  inout [35:0] GPIO,  `else  ///////// HEX2 ///////// 1.2 V ///////  output [6:0] HEX2,  ///////// HEX3 ///////// 1.2 V ///////  output [6:0] HEX3,  `endif /\*ENABLE\_GPIO\*/  ///////// HDMI /////////  output HDMI\_TX\_CLK,  output [23:0] HDMI\_TX\_D,  output HDMI\_TX\_DE,  output HDMI\_TX\_HS,  input HDMI\_TX\_INT,  output HDMI\_TX\_VS,  ///////// HEX0 /////////  output [6:0] HEX0,  ///////// HEX1 /////////  output [6:0] HEX1,  ///////// HSMC ///////// 2.5 V ///////  input HSMC\_CLKIN0,  input [2:1] HSMC\_CLKIN\_n,  input [2:1] HSMC\_CLKIN\_p,  output HSMC\_CLKOUT0,  output [2:1] HSMC\_CLKOUT\_n,  output [2:1] HSMC\_CLKOUT\_p,  inout [3:0] HSMC\_D,  `ifdef ENABLE\_HSMC\_XCVR  input [3:0] HSMC\_GXB\_RX\_p, /// 1.5-V PCML  output [3:0] HSMC\_GXB\_TX\_p, /// 1.5-V PCML  `endif /\*ENABLE\_HSMC\_XCVR\*/  inout [16:0] HSMC\_RX\_n,  inout [16:0] HSMC\_RX\_p,  inout [16:0] HSMC\_TX\_n,  inout [16:0] HSMC\_TX\_p,  ///////// I2C ///////// 2.5 V ///////  output I2C\_SCL,  inout I2C\_SDA,  ///////// KEY ///////// 1.2 V ///////  input [3:0] KEY,  ///////// LEDG ///////// 2.5 V ///////  output [7:0] LEDG,  ///////// LEDR ///////// 2.5 V ///////  output [9:0] LEDR,  `ifdef ENABLE\_REFCLK  ///////// REFCLK ///////// 1.5-V PCML ///////  input REFCLK\_p0,  input REFCLK\_p1,  `endif /\*ENABLE\_REFCLK\*/  ///////// SD ///////// 3.3-V LVTTL ///////  output SD\_CLK,  inout SD\_CMD,  inout [3:0] SD\_DAT,  `ifdef ENABLE\_SMA  ///////// SMA ///////// 1.5-V PCML ///////  input SMA\_GXB\_RX\_p,  output SMA\_GXB\_TX\_p,  `endif /\*ENABLE\_SMA\*/  ///////// SRAM ///////// 3.3-V LVTTL ///////  output [17:0] SRAM\_A,  output SRAM\_CE\_n,  inout [15:0] SRAM\_D,  output SRAM\_LB\_n,  output SRAM\_OE\_n,  output SRAM\_UB\_n,  output SRAM\_WE\_n,  ///////// SW ///////// 1.2 V ///////  input [9:0] SW,  ///////// UART ///////// 2.5 V ///////  input UART\_RX,  output UART\_TX  );  lab2 dut( SW[6] , KEY[0] , SW[4] , SW[3:0] , HEX1 , HEX0 ) ;  endmodule  // Lab2 module  module lab2(rst, clk, S, V, ss1, ss0);  input clk , rst , S;  input [3:0] V ;  output reg [6:0] ss1 , ss0 ;  reg [7:0] acc;  always @ ( posedge clk ) begin // posedge clk  if ( rst ) begin // if  acc = 0 ;  end  else begin  case(S)  1:begin  acc = acc - V;  end  0:begin  acc = acc+V;  end  endcase  end  end  always @( acc ) begin  case ( acc[3:0] )  4'b0000 : ss0 = 7'b1000000 ;  4'b0001 : ss0 = 7'b1111001 ;  4'b0010 : ss0 = 7'b0100100 ;  4'b0011 : ss0 = 7'b0110000 ;  4'b0100 : ss0 = 7'b0011001 ;  4'b0101 : ss0 = 7'b0010010 ;  4'b0110 : ss0 = 7'b0000010 ;  4'b0111 : ss0 = 7'b1111000 ;  4'b1000 : ss0 = 7'b0000000 ;  4'b1001 : ss0 = 7'b0010000 ;  4'b1010 : ss0 = 7'b0001000 ;  4'b1011 : ss0 = 7'b0000011 ;  4'b1100 : ss0 = 7'b1000110 ;  4'b1101 : ss0 = 7'b0100001 ;  4'b1110 : ss0 = 7'b0000110 ;  4'b1111 : ss0 = 7'b0001110 ;  endcase  case ( acc[7:4] )  4'b0000 : ss1 = 7'b1000000 ;  4'b0001 : ss1 = 7'b1111001 ;  4'b0010 : ss1 = 7'b0100100 ;  4'b0011 : ss1 = 7'b0110000 ;  4'b0100 : ss1 = 7'b0011001 ;  4'b0101 : ss1 = 7'b0010010 ;  4'b0110 : ss1 = 7'b0000010 ;  4'b0111 : ss1 = 7'b1111000 ;  4'b1000 : ss1 = 7'b0000000 ;  4'b1001 : ss1 = 7'b0010000 ;  4'b1010 : ss1 = 7'b0001000 ;  4'b1011 : ss1 = 7'b0000011 ;  4'b1100 : ss1 = 7'b1000110 ;  4'b1101 : ss1 = 7'b0100001 ;  4'b1110 : ss1 = 7'b0000110 ;  4'b1111 : ss1 = 7'b0001110 ;  endcase  end  endmodule  module lab2\_tb;  reg clk\_tb, rst\_tb, S\_tb;  reg [7:0] V\_tb, acc\_tb;  reg [7:4] msb\_tb;  reg [3:0] lsb\_tb;  wire [6:0] ss1\_tb, ss0\_tb;  integer i;  lab2 dut(rst\_tb, clk\_tb, S\_tb, V\_tb, ss1\_tb, ss0\_tb);  // Generate clock  always begin  clk = 0;  #10 ;  clk = 1;  #10 ;  end  // Other stimulus  initial begin  rst = 1 ;  #20 ;  rst = 0 ;  #20 ;  for(i=0; i<= 10; i = i + 1)begin  S\_tb = 0;  V\_tb = 4'b1110;  acc\_tb = 4'b0000;  #20;  $stop;  end  end  endmodule |

Figure 3. Verilog code for the proposed solution

|  |
| --- |
|  |

Figure 5. Lab pictures of the running solution