Class:	**CPE300L**	Semester:	**Fall 2021**
Points		Document author:	**Jerrod Batu**
		Author's email:	**batuj1@unlv.nevada.edu**
		Document topic:	**Postlab 7**
Instructor's comments:			

1. Introduction / Theory of Operation

Throughout this lab, I will continue my knowledge of the general data path implementation. Specifically, I will design a control unit for the given summation algorithm, test on a testbench, and debug the circuit.

a. The **control unit** is designed to decipher when and where the control words are to be implemented into the general data path. First, the control signal from the datapath would decide on the proper inputs to be inserted into the control unit. Next, the control unit would utilize a state transition diagram to determine where the control words are to be initiated into the proper inputs of the components within the data path. Lastly, the outputs of the state diagram would be the inputs into the components within the datapath.

b. The **control signals** that would be encountered are based from the datapath and state transition diagram. Typically, the input control signal from the datapath would consist of gate logic that would determine how the inputs of the components of the datapath would be initiated from the outputs of the control unit. The control words are input signals into the datapath. The control words consist of IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, and OE. IE is usually initiated when the input is enabled into the datapath; therefore, one initialization per clock cycle. Then, WE, WA, RAE, RAA, RBE, and RBA are when the inputs are written or read from the Register File. By determining if the inputs are read or written, this allows the algorithm to be properly implemented on how the inputs to the ALU are initiated. Next, the ALU inputs are to be determined on how the state transition diagram is implemented on where the ALU would be needed. For example, if the algorithm were to require an increment, then this is where the ALU would be signaled. The same were to apply to a SHifter as the inputs are determined on where the state transition diagram would require it. OE is always initiated at the end of the datapath; therefore, utilizing the buffer to figure out the final output of any algorithm.

2. Prelab

https://docs.google.com/document/d/1STWC6v6QR5H1JHVg2bfFpIqNAMuaD1xW/edit?usp=sharing&ouid=102808507017671072128&rtpof=true&sd=true

This is the link to my prelab 7.

3. Results of Experiments

Experiment 1

```
module CU (start, clk, rst, neq0, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE);
	input start, clk, rst;
	output IE, WE, RAE, RBE, OE;
	output [1:0] WA, RAA, RBA, SH;
	output [2:0] ALU;

	input wire neq0;
	reg [2:0] state;
	reg [2:0] nextstate;

	parameter S0 = 3'b000;
	parameter S1 = 3'b001;
	parameter S2 = 3'b010;
	parameter S3 = 3'b011;
	parameter S4 = 4'b100;

	initial
		state = S0;

	// State register
	always @ (posedge clk)
	begin
		state <= nextstate;
	end

	// Next State Logic
	always @ (*)
		case (state)
			S0: if (start)
					nextstate = S1;
				else
					nextstate = S0;
			S1: if (neq0)
					nextstate = S2;
				else
					nextstate = S4;
			S2: nextstate = S3;
			S3: if (neq0)
					nextstate = S2;
				else
```

```
                                        nextstate = S4;
                            S4: if (rst) nextstate = 0;
                                    else nextstate = S4;
                            default: nextstate  = S0;
                    endcase

            // Output Logic
            assign IE = state == S1;

            assign WE = (state == S0) || (state == S1) || (state == S2) || (state == S3);
            assign WA[1] = 0;
            assign WA[0] = (state == S1) || (state == S3);
            assign RAE = (state == S0) || (state == S2) || (state == S3) || (state == S4);
            assign RAA[1] = 0;
            assign RAA[0] = (state == S3);

            assign RBE = (state == S0) || (state == S2);
            assign RBA[1] = 0;
            assign RBA[0] = 0;

            assign ALU[2] = (state == S0) || (state == S2) || (state == S3);
            assign ALU[1] = state == S3;
            assign ALU[0] = (state == S0) || (state == S3);

            assign SH[1] = 0;
            assign SH[0] = 0;
            assign OE = state == S4;
endmodule
```

This is my Verilog code for the control unit implemented with the help of figure 4 and figure 6.

EXPERIMENT 2

```
`timescale 1ns / 100ps

module topLevel (sumGDP, startGDP, rstGDP, clkGDP, nGDP, complete);
            input startGDP, clkGDP, rstGDP;
            input [7:0] nGDP;
            output complete;
            output [7:0] sumGDP;

            wire WE, RAE, RBE, OE, IE, neq0;
            wire [1:0] WA, RAA, RBA, SH;
```

```
	wire [2:0] ALU;

	CU control (~startGDP, clkGDP, ~rstGDP, neq0, IE, WE, WA, RAE, RAA, RBE, RBA,
ALU, SH, OE);
	DP datapath (nGDP, clkGDP, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE,
neq0, sumGDP);

	assign complete = OE;
endmodule

module CU (start, clk, rst, neq0, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE);
	input start, clk, rst;
	output IE, WE, RAE, RBE, OE;
	output [1:0] WA, RAA, RBA, SH;
	output [2:0] ALU;

	input wire neq0;
	reg [2:0] state;
	reg [2:0] nextstate;

	parameter S0 = 3'b000;
	parameter S1 = 3'b001;
	parameter S2 = 3'b010;
	parameter S3 = 3'b011;
	parameter S4 = 4'b100;

	initial
		state = S0;

	// State register
	always @ (posedge clk)
	begin
		state <= nextstate;
	end

	// Next State Logic
	always @ (*)
		case (state)
			S0: if (start)
					nextstate = S1;
				else
					nextstate = S0;
			S1: if (neq0)
					nextstate = S2;
```

```
                                        else
                                                nextstate = S4;
                                S2: nextstate = S3;
                                S3: if (neq0)
                                                nextstate = S2;
                                        else
                                                nextstate = S4;
                                S4: if (rst) nextstate = 0;
                                        else nextstate = S4;
                                default: nextstate  = S0;
                        endcase

                // Output Logic
                assign IE = state == S1;

                assign WE = (state == S0) || (state == S1) || (state == S2) || (state == S3);
                assign WA[1] = 0;
                assign WA[0] = (state == S1) || (state == S3);
                assign RAE = (state == S0) || (state == S2) || (state == S3) || (state == S4);
                assign RAA[1] = 0;
                assign RAA[0] = state == S3;

                assign RBE = (state == S0) || (state == S2);
                assign RBA[1] = 0;
                assign RBA[0] = state == S2;

                assign ALU[2] = (state == S0) || (state == S2) || (state == S3);
                assign ALU[1] = state == S3;
                assign ALU[0] = (state == S0) || (state == S3);

                assign SH[1] = 0;
                assign SH[0] = 0;
                assign OE = state == S4;
endmodule

//Data Path
module DP (nIn, clk, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE, ne0, out);
                input clk, IE, WE, RAE, RBE, OE;
                input [1:0] WA, RAA, RBA, SH;
                input [2:0] ALU;
                input [7:0] nIn;
                output ne0;
                output wire [7:0] out;
```

```
        reg [7:0] rfIn;
        wire [7:0] rfA, rfB, aluOut, shOut, n;

        initial
        rfIn = 0;

        always @ (*)
        rfIn = n;

        gdpMux mux (shOut, nIn, IE, n);
        gdpRF RF (clk, WE, RAE, RBE, RAA, RBA, WA, rfIn, rfA, rfB);
        gdpALU theALU (rfA, rfB, ALU, aluOut);
        gdpShift SHIFT (aluOut, SH, shOut);
        gdpBuffer buff (shOut, OE, out);

        assign ne0 = n != 0;  //note: checks the false //condition

endmodule

// 2-to-1 mux
module gdpMux (a, b, sel, out);
        input[7:0] a;
        input[7:0] b;
        input sel;
        output reg [7:0] out;

        always @(*)
                if(sel == 0)
                        out = a;
                else
                        out = b;
endmodule

// register file
module gdpRF(clk, WE, RAE, RBE, RAA, RBA, WA, inD, ReadA, ReadB);
        input clk, WE, RAE, RBE;
        input [1:0] RAA, RBA, WA;
        input [7:0] inD;
        output [7:0] ReadA, ReadB;
        reg [7:0] regF [0:3];

        // Write when WE asserted
        always @(posedge clk)
        if (WE == 1)  regF[WA] <= inD;
```

```
          //reading to Port A and B, combinational
          assign ReadA = (RAE)? regF [RAA]:0;
          assign ReadB = (RBE)? regF [RBA]:0;
endmodule

// arithmetic logic unit
module gdpALU (a,b,sel, out);
   input [7:0] a,b;
   input [2:0] sel;
   output reg [7:0] out;

   always @ (*)
   begin
      case (sel)
         3'b000: out = a;
         3'b001: out = a & b;
         3'b010: out = a | b;
         3'b011: out = !a;
         3'b100: out = a + b;
         3'b101: out = a - b;
         3'b110: out = a + 1;
         3'b111: out = a - 1;
      endcase
   end
endmodule

// Shifter
module gdpShift (a,sh,out);
   input [7:0] a;
   input [1:0] sh;
   output reg [7:0] out;

   always @ (*)
   begin
      case(sh)
         3'b00: out = a;
         3'b01: out = a << 1;
         3'b10: out = a >> 1;
         3'b11: out= { a[6],a[5],a[4],a[3],a[2],a[1],a[0], a[7] } ;
      endcase
   end
endmodule
```

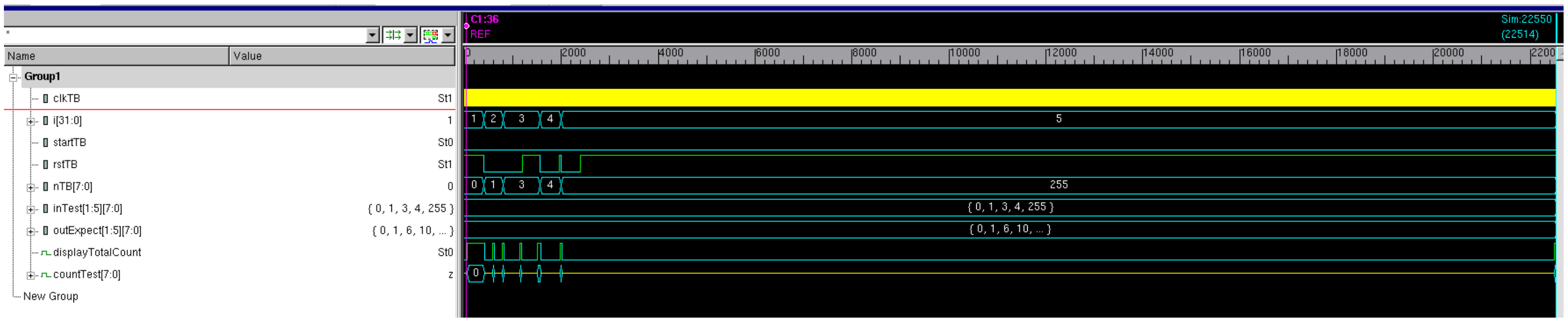
```
//buffer
module gdpBuffer (a, buff, out);
        input [7:0] a;
        input buff;
        output reg [7:0] out;

        always @(*)
                if(buff == 1)
                        out = a;
                else
                        out = 8'bzzzz_zzzz;
endmodule
```

This is my top-level module utilizing the datapath from lab 6 and the control unit code from experiment 1 to create a complete general datapath code.

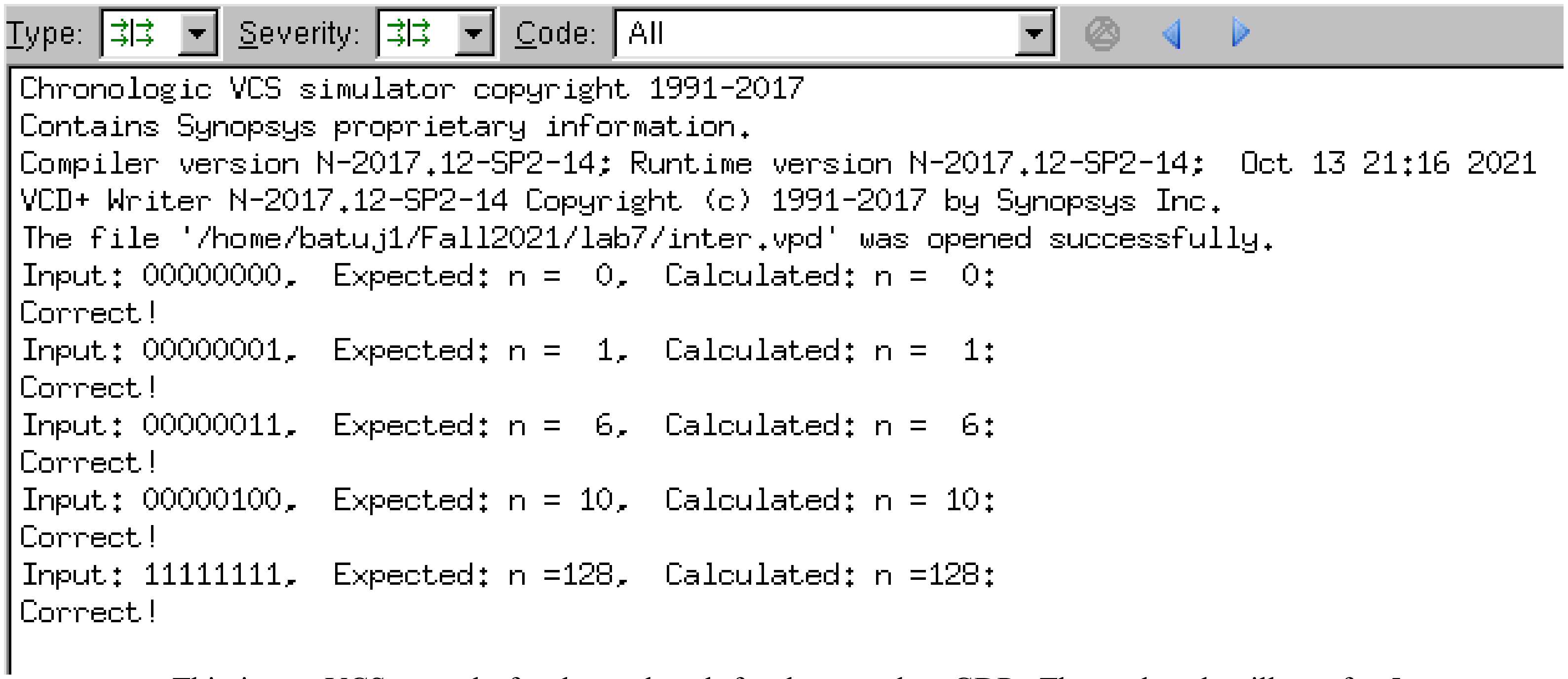
EXPERIMENT 3

```
`timescale 1ns / 100ps

module GDPTB;
        reg startTB, rstTB, clkTB;
        reg [7:0] nTB;
        wire displayTotalCount;
        wire [7:0] countTest;

        // Test Vectors for desired output
        reg [7:0] inTest [1:5];
        reg [7:0] outExpect [1:5];

        integer i;

        initial
                begin
                        // initialize inputs to test and their expected outputs
                        inTest[1]    = 8'b00000000;
                        outExpect[1] = 8'b00000000;

                        inTest[2]    = 8'b00000001;
                        outExpect[2] = 8'b00000001;

        inTest[3]    = 8'b00000011;
        outExpect[3] = 8'b00000110;
        inTest[4]    = 8'b00000100;
        outExpect[4] = 8'b00001010;

        inTest[5]    = 8'b11111111;
        outExpect[5] = 8'b10000000;

                        startTB = 1;
                        rstTB = 1;
                        clkTB = 0;

                        // generate clock
                        forever
                                #2 clkTB = ~clkTB;
                end

        always
                begin
                        for(i = 1; i <= 5; i = i + 1) // test for the 5 inputs
                                begin
                                        nTB <= inTest[i];
                                        startTB = 0;
                                        #40 rstTB = 1;

                                // Waiting for code to finish then displayResults when done
                                        while (displayTotalCount != 1)
                                                #5 begin end

                $write ("Input: %b,  Expected: n =%d,  Calculated: n =%d: ",
                    nTB, outExpect[i], countTest);
                                        if (countTest == outExpect[i])
                                                $display("Correct!");
                                        else
                                                $display("Incorrect!");
                                        rstTB = 0;
                                        startTB = 1;
                                end
                        $stop;
                end

        //Instantiate GDP
        topLevel mainGDP (countTest, startTB, rstTB, clkTB, nTB, displayTotalCount);
endmodule
```

This is my testbench for the complete GDP that tests for correctness within my GDP design.



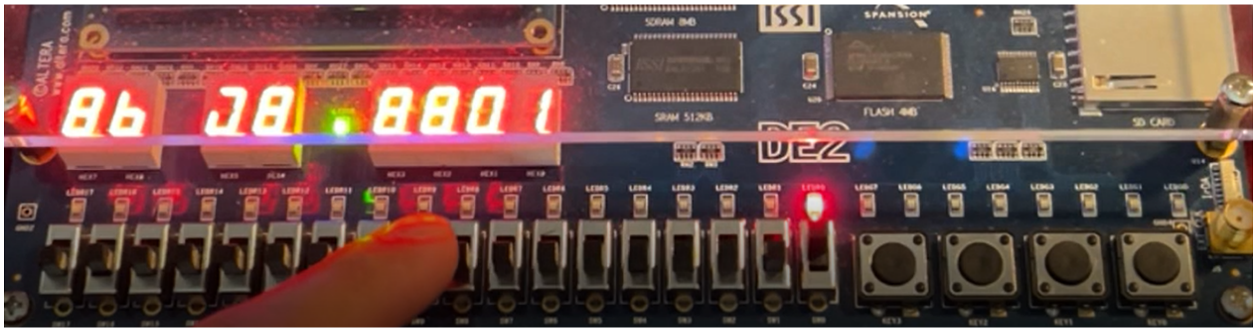
These are my VCS waveforms for the testbench for the complete GDP. The clock here is shown as a yellow bar because there are several positive clock edges that would be needed for a 11111111 (255) input.



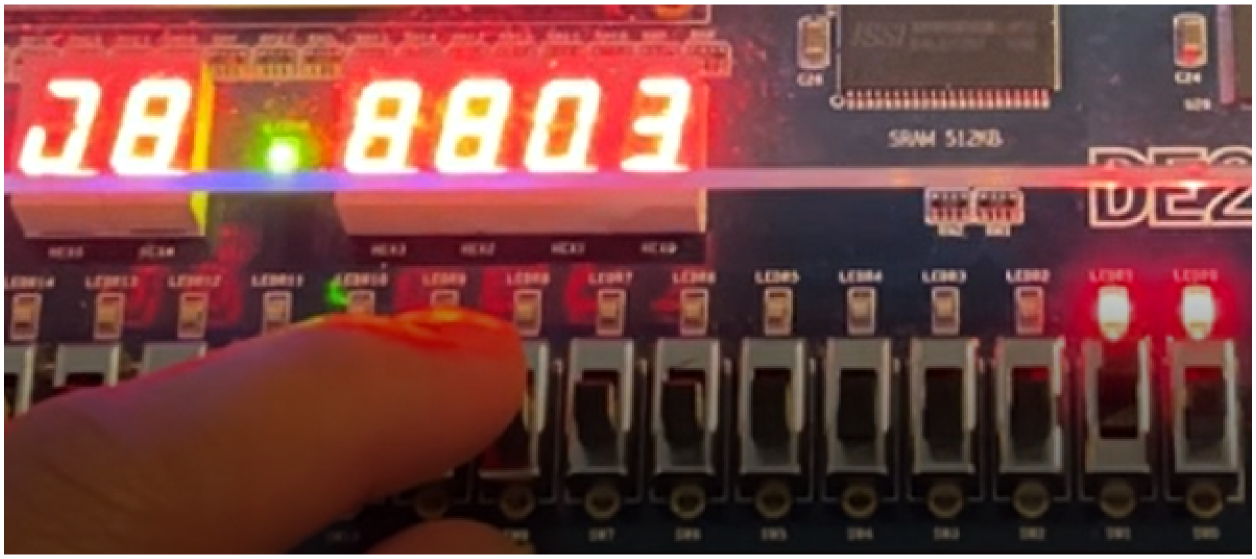
```
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Oct 13 21:16 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab7/inter.vpd' was opened successfully.
Input: 00000000,  Expected: n =  0,  Calculated: n =  0:
Correct!
Input: 00000001,  Expected: n =  1,  Calculated: n =  1:
Correct!
Input: 00000011,  Expected: n =  6,  Calculated: n =  6:
Correct!
Input: 00000100,  Expected: n = 10,  Calculated: n = 10:
Correct!
Input: 11111111,  Expected: n =128,  Calculated: n =128:
Correct!
```

This is my VCS console for the testbench for the complete GDP. The testbench will test for 5 different values: 0, 1, 3, 4, and 255. The expected result will be the output to be compared with the calculated value as the output that is actually found from the GDP Verilog code.

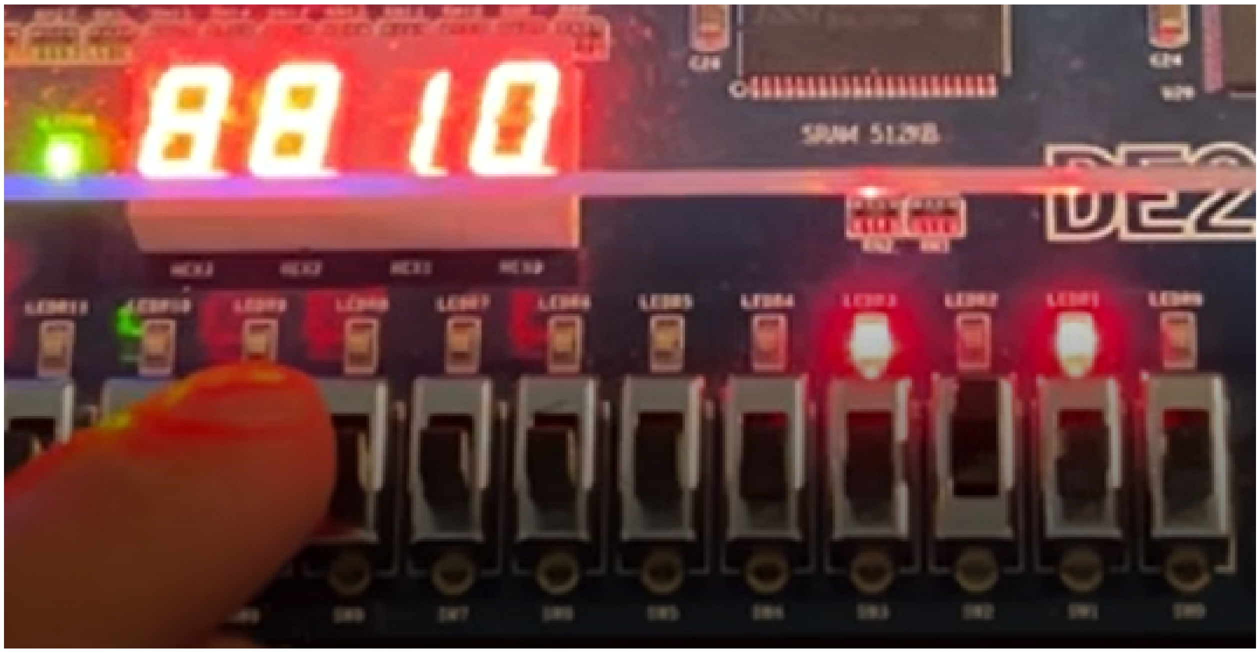
EXPERIMENT 4

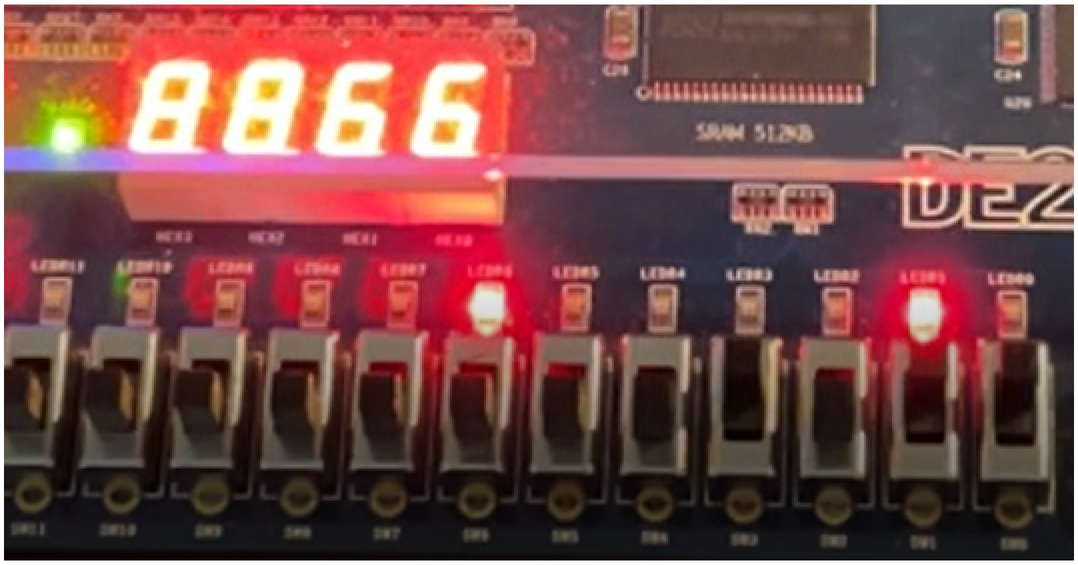


This is my Verilog code for the GDP that is implemented onto the DE2 board along with the 7 segment display. Here, the input is 1 and the output is 1.



This is my Verilog code for the GDP that is implemented onto the DE2 board along with the 7 segment display. Here, the input is 2 and the output is 3.


This is my Verilog code for the GDP that is implemented onto the DE2 board along with the 7 segment display. Here, the input is 4 and the output is 10.


This is my Verilog code for the GDP that is implemented onto the DE2 board along with the 7 segment display. Here, the input is 11 and the output is 66.

https://drive.google.com/file/d/1h29ut75ivcXWp8rHNUDc7__GKVpwOAIl/view?usp=sharing
This is the link to my video explaining the operation of the summation algorithm implemented onto the GDP using the DE2 board.

4. Answers to questions

Question 1: The control unit follows the positive clock edge and will start upon initialization from the top-level module. The control unit also utilizes finite state machines that would dictate how the control words of each state would be inputted as the control signals to the general data path. In addition, the control unit also determines the next state logic based on the condition given from the data path module. When implementing the control unit onto the summation algorithm, its role is to properly loop when the condition is not equal to 0, store the input N into the output every time N is not equal to 0, and decrement N until N is equal to 0. The final output is determined by the final state of the summation algorithm. In addition, the control signals, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, and OE, are to be utilized based on the state's current condition from the control word table.

Question 2: The main disadvantage of using a general datapath and dedicated datapath would lie where tests are being performed from the testbench. By using the dedicated datapath from the previous lab, it is clear to see how the inputs are being stored within the register file after they are added to one another. When using the general datapath, the user is only able to see the final output, and input to be calculated. The advantage that the dedicated datapath has is its flexibility when troubleshooting each component to figure out where the code was not properly performing. By using the general datapath, it is rather hard to tell where the errors occur as the control unit and datapath are being used at the same time.

5. Conclusions & Summary

This lab was fairly easy, and I was able to complete it within one day. We did a similar assignment like this in the CPE 300 lecture; therefore, this lab expanded my thinking process and knowledge on how the control unit is properly implemented onto the datapath; however, I did prefer last lab as the datapath were easily to be troubleshooted if something went wrong within my code. In contrast, I do like learning about the control unit, and how the states affect the overall code and performance. I am becoming very familiar with the general datapath, and I am looking forward to what is in store for the next lab.