UNIVERSITY OF NEVADA LAS VEGAS. DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING LABORATORIES.

Class:	**CPE300L Digital Systems Architecture and Design 1001**		Semester:	**Fall 2024**
Points		Document author:	**Narek Kalikian**	
		Author's email:	**kalikn1@unlv.nevada.edu**	
		Document topic:	**Postlab 4**	
Instructor's comments:				

1. Introduction / Theory of Operation

In this lab, we worked with the general datapath and designed a control unit for a summation algorithm. Using a finite state machine to show state transitions, we were able to write the output logic code for this algorithm. The datapath is responsible for performing the arithmetic and shifting operations. The general datapath combines the control unit and datapath to integrate the functionality of the system. The control unit generates signals for the datapath based on the inputs and state of the FSM. The testbench is used to simulate and test the functionality of the system.

- The control unit handles the state transitions and the general behavior of the datapath. The CU cycles through the states of the FSM based on the value of input nIn.
- Input Enable: Controls whether nIn is used in the datapath
- Write Enable: Enables writing to the register
- Read Enable A & B: Controls whether the contents of these registers are read and sent to ALU
- Output Enable: Controls whether the result found in the datapath is sent to Sum output
- Write Address: Specifies the destination address for writing
- Read Address A & B: Specifies the source register for the first and second operands
- Arithmetic Logic Unit: Determines which operation needs to be performed
- Shifter: Determines whether data is shifted left, right, or not shifted
- N Equals Zero: Determines when input is equal to zero

2. Description of Experiments

Experiment 1 - Verilog Code for Datapath Module:

```
146  module DP (nEQZero, sum, nIn, clk, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE);
147
148    input clk, IE, WE, RAE, RBE, OE;
149    input [1:0] WA, RAA, RBA, SH;
150    input [2:0] ALU;
151    input [7:0] nIn;
152
153    output nEQZero;
154    output wire [7:0] sum;
155
156    reg [7:0] rfIn;
157    wire [7:0] RFa, RFb, aluOut, shOut, n;
158
159     initial
160     rfIn = 0;
161
162    always @ (*)
163     rfIn = n;
164
165    MUX muxs (n, shOut, nIn, IE);
166    Regfile RF (clk, RAA, RFa, RBA, RFb, WE, WA, rfIn, RAE, RBE);
167    ALU theALU (aluOut, RFa, RFb, ALU);
168    Shifter SHIFT (shOut, aluOut, SH);
169    Buffer buffer1 (sum, shOut, OE);
170
171    assign nEQZero = n == 0;
172
173  endmodule
```

Experiment 2 - Verilog Code for Control Unit Module:

```
175  module CU(IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE, start, clk,
176    restart, nEqZero);
177  input start, clk, restart;
178  output IE, WE, RAE, RBE, OE;
179  output [1:0] WA, RAA, RBA, SH;
180  output [2:0] ALU;
181  input wire nEqZero;
182  reg [2:0] current_state;
183  reg [2:0] next_state;
184    parameter S0 = 3'b000;
185    parameter S1 = 3'b001;
186    parameter S2 = 3'b010;
187    parameter S3 = 3'b011;
188    parameter S4 = 3'b100;
189    initial
190    current_state = S0;
191
192    always @ (posedge clk)
193     begin
194      current_state <= next_state;
195     end
196
197    always @ (*)
198      case(current_state)
199      S0:  if(start)
200             next_state = S1;
201           else
202             next_state = S0;
203
204      S1:  if(nEqZero)
205             next_state = S4;
206           else
207             next_state = S2;
208
209      S2:    next_state = S3;
210             //sum = sum + n;
211
212      S3:  if(nEqZero)
213              next_state = S4;
214           else
215              next_state = S2;
216
217      S4:  if(restart)
218              next_state = S0;
219           else
220              next_state = S4;
221
222      default: next state = S0;
```

```
223       endcase
224     assign IE = current_state == S1;
225     assign WE = ~current_state[2];
226     assign WA[1] = 0;
227     assign WA[0] = ~current_state[2] && current_state[0];
228     assign RAE = ~current_state[2] && current_state[1] || ~current_state[1] && ~current_state[0];
229     assign RAA[1] = 0;
230     assign RAA[0] = current_state == S3;
231     assign RBE = ~current_state[2] && ~current_state[0];
232     assign RBA[1] = 0;
233     assign RBA[0] = current_state == S2;
234     assign ALU[2] = ~current_state[2] && ~current_state[0] || ~current_state[2] && current_state[1];
235     assign ALU[1] = current_state == S3;
236     assign ALU[0] = current_state == S0 || current_state == S3;
237     assign SH[1] = 0;
238     assign SH[0] = 0;
239     assign OE = current_state == S4;
240   endmodule
```

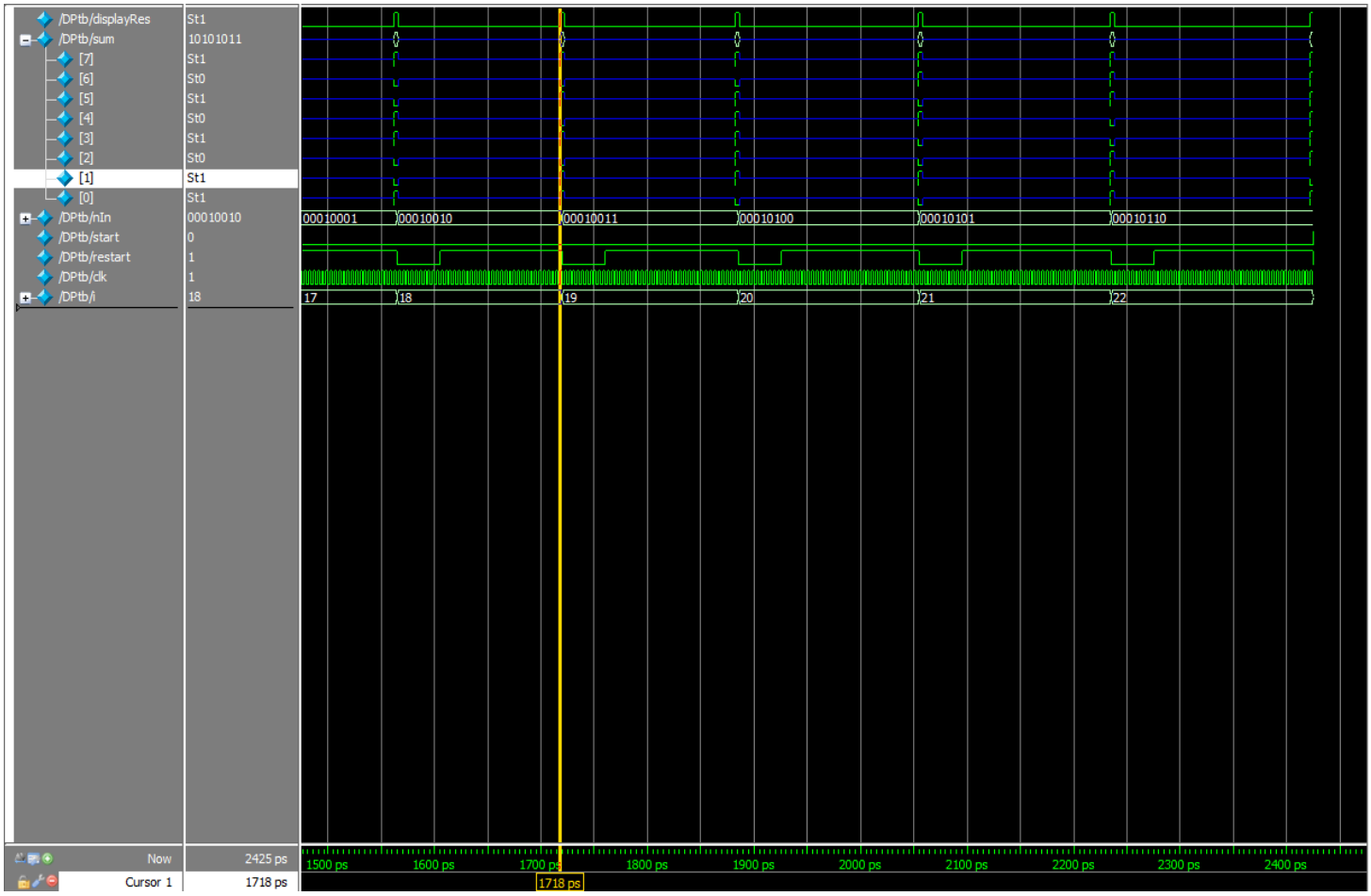
Experiment 3 - Verilog Code for Top-Level General Datapath Module:

```
47  module GDP(sum, start, restart, clk, n, done);
48  input start, clk, restart;
49  input [7:0] n;
50  output [7:0] sum;
51  output done;
52  wire WE, RAE, RBE, OE, nEqZero, IE;
53  wire [1:0] WA, RAA, RBA, SH;
54  wire [2:0] ALU;
55  CU control (IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE, ~start, clk, ~restart,
56  nEqZero);
57  DP datapath (nEqZero, sum, n, clk, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE);
58  assign done = OE;
59  endmodule
```

Experiment 4 - Summation Algorithm Testbench Code:

```
1   module DPtb;
2
3   wire displayRes;
4   wire [7:0] sum;
5   reg [7:0] nIn;
6   reg start, restart, clk;
7   integer i;
8
9   initial
10  begin
11    clk = 0;
12    start = 1;
13    restart = 1;
14    forever
15      #2 clk = ~clk;
16  end
17
18  always
19  begin
20    for (i = 0; i < 23; i = i + 1)
21    begin
22      nIn <= i;
23      start = 0;
24      #40
25      restart = 1;
26
27      while(displayRes != 1)#5
28      begin
29           end
30        $write ("Input: %2d, Result: %3d, Expected: %3d: ", nIn, sum, i*(i+1)/2);
31
32        if (sum == (i*(i+1)/2))
33        $display ("Sum is Correct.");
34        else
35        $display ("Sum is Incorrect.");
36
37        restart = 0;
38        start = 1;
39      end
40      $stop;
41  end
42
43  GDP main (sum, start, restart, clk, nIn, displayRes);
44
45  endmodule
```

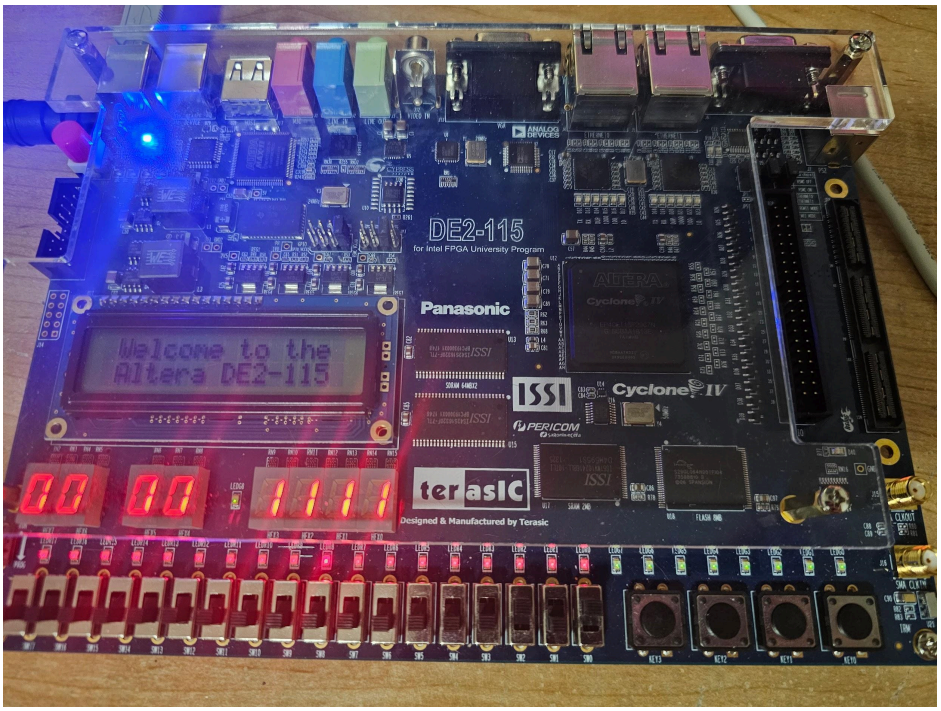
Experiment 4 - Summation Algorithm Waveform Simulation:



Experiment 4 - Summation Algorithm Simulation Console:

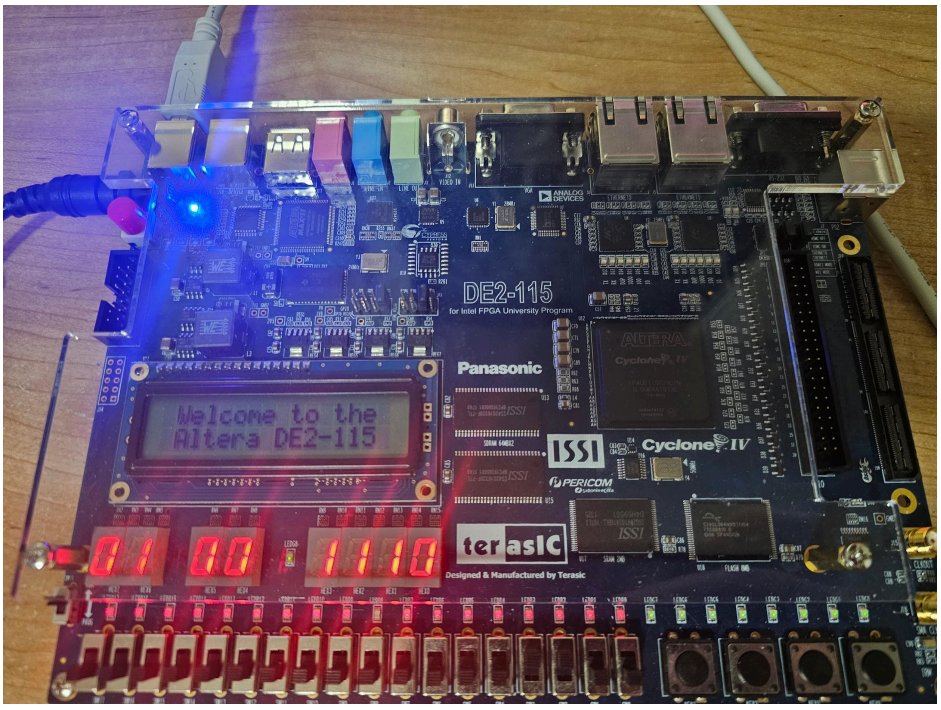
```
VSIM 5> run -all
# Input:  0, Result:   0, Expected:   0: Sum is Correct.
# Input:  1, Result:   1, Expected:   1: Sum is Correct.
# Input:  2, Result:   3, Expected:   3: Sum is Correct.
# Input:  3, Result:   6, Expected:   6: Sum is Correct.
# Input:  4, Result:  10, Expected:  10: Sum is Correct.
# Input:  5, Result:  15, Expected:  15: Sum is Correct.
# Input:  6, Result:  21, Expected:  21: Sum is Correct.
# Input:  7, Result:  28, Expected:  28: Sum is Correct.
# Input:  8, Result:  36, Expected:  36: Sum is Correct.
# Input:  9, Result:  45, Expected:  45: Sum is Correct.
# Input: 10, Result:  55, Expected:  55: Sum is Correct.
# Input: 11, Result:  66, Expected:  66: Sum is Correct.
# Input: 12, Result:  78, Expected:  78: Sum is Correct.
# Input: 13, Result:  91, Expected:  91: Sum is Correct.
# Input: 14, Result: 105, Expected: 105: Sum is Correct.
# Input: 15, Result: 120, Expected: 120: Sum is Correct.
# Input: 16, Result: 136, Expected: 136: Sum is Correct.
# Input: 17, Result: 153, Expected: 153: Sum is Correct.
# Input: 18, Result: 171, Expected: 171: Sum is Correct.
# Input: 19, Result: 190, Expected: 190: Sum is Correct.
# Input: 20, Result: 210, Expected: 210: Sum is Correct.
# Input: 21, Result: 231, Expected: 231: Sum is Correct.
# Input: 22, Result: 253, Expected: 253: Sum is Correct.
# Break in Module DPtb at C:/altera/13.0spl/datapath.v line 40
```

Experiment 5 - DE2 Board Implementation n < 10 (n = 5):



n = 00000101 (5) → 1 + 2 + 3 + 4 + 5 = 15. Displayed Sum = 00001111 (15)

Experiment 5 - DE2 Board Implementation n > 10 (n = 12):



n = 00001100 (12) → 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 = 78. Displayed Sum = 01001110 (78)

Experiment 5 - DE2 Board Implementation Video: *Attached to Canvas Submission*

Experiment 5 - DE2 Board Implementation Video Explanation:
Tested values of n from n = 1 to n = 15. Switches 0 through 7 are mapped to the respective bits of the 8-bit binary input value n. The system takes the value of n, and sums all of the numbers up to n. This sum is displayed in binary on the seven-segment displays on the DE2 board. 8-bit signals were used so this means n can only be tested to up to 23, after which the sum value will be greater than 256 which will overflow due to sum being an 8-bit output signal.

3. Questions and Answers

Control Unit - A control unit (CU) is part of a digital system. It generates control signals based on the system's inputs and current state and manages the operations of the system. In the context of this lab, the control unit dictates how the components of the datapath work together for the function of the summation algorithm and when certain operations need to be performed to accomplish this.
Disadvantages of GDP: The disadvantage of using the general datapath over a dedicated datapath is reduced efficiency. As the name implies, the GDP is designed to handle multiple tasks, even ones you might not need. It has more flexibility but at the cost increased power and resource consumption. The dedicated datapath is designed for one specific task.

4. Conclusions

We were able to learn about the general datapath and its various components. We designed each of these components and learned how they work with each other to implement a simple summation algorithm. The datapath and its components are critical to computer architecture and important to understand for logic design.