Class:	CPE-300L	Semester:	Spring 2020

Points		Document author:	Brysen Kokubun
		Author's email:	Brysenkokubun96@gmail.com
		Document topic:	Postlab 7

Instructor's comments:

1. Introduction / Theory of Operation

- For this lab experiment we created the control unit for the GDP that we constructed and verified in lab experiment 6.

a. Describe how the control unit is designed.

- The control unit is designed by implementing an algorithm to send certain control signals/words to a general data path to perform different operations.

b. Explain all the control signal that you encountered.

- The control signals that were encountered were as followed.

Control Word	Instruction	IE 15	WE 14	$WA_{1,0}$ 13–12	RAE 11	$RAA_{1,0}$ 10–9	RBE 8	$RBA_{1,0}$ 7–6	$ALU_{2,1,0}$ 5–3	$SH_{1,0}$ 2–1	OE 0
1	$sum = 0$	0	1	00	1	00	1	00	101 (subtract)	00	0
2	INPUT n	1	1	01	0	××	0	××	× × ×	××	0
3	$sum = sum + n$	0	1	00	1	00	1	01	100 (add)	00	0
4	$n = n - 1$	0	1	01	1	01	0	××	111 (decrement)	00	0
5	OUTPUT sum	×	0	××	1	00	0	××	000 (pass)	00	1

Each of the control words/signals created different instructions to be performed on the GDP. Each control word was composed of 16-bits and every bit has a particular meaning which will cause the GDP to perform different operations.

2. Prelab report

- Attached Below

3. Experiment Results

Experiment #1: Implementation of control unit for GDP

```
module CU(IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE, start, clk, restart, nEqZero);

    input start, clk, restart;
    output IE, WE, RAE, RBE, OE;
    output [1:0] WA, RAA, RBA, SH;
    output [2:0] ALU;

    input wire nEqZero;
    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 = 3'b100;

    initial
        state = S0;

    // State register
    always @ (posedge clk)
    begin
        state <= nextstate;
    end
    // NS logic
    always @ (*)
        case(state)
            S0:
                if(start) nextstate = S1;
                else nextstate = S0;
            S1:
                if(nEqZero) nextstate = S4;
                else nextstate = S2;
            S2:
                nextstate = S3;
            S3:
                if(nEqZero) nextstate = S4;
                else nextstate = S2;
            S4:
                if(restart) nextstate = S0;
                else nextstate = S4;

            default: nextstate = S0;
        endcase
    // output logic
    assign IE = (state == S1);

    assign WE = ~(state == S4);
    assign WA[1] = 0;
    assign WA[0] = (state == S1) || (state == S3);

    assign RAE = ~(state == S1);
    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 == S4);
    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
```

Experiment #2: Top-level module

Verilog code with top-level GDP module with Control Unit instantiation

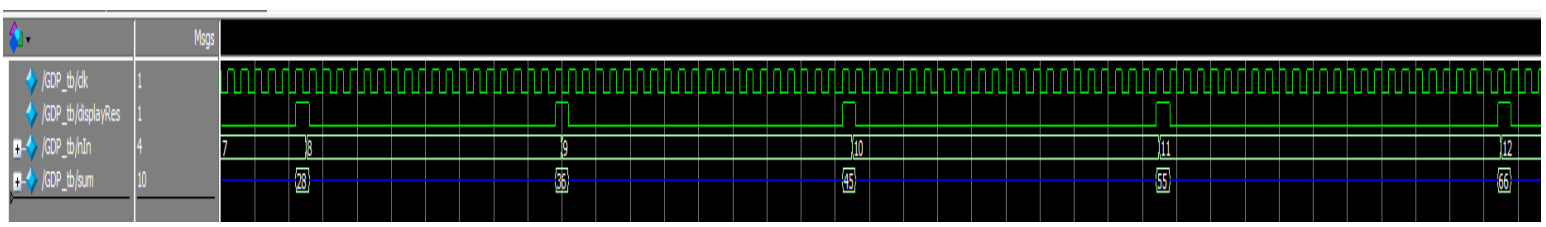
```
module GDP(Sum, start, restart, clk, n, done);

input start, clk, restart;
    input [7:0] n;
    output [7:0] Sum;
    output done;

    wire WE, RAE, RBE, OE, nEqZero, IE;
    wire [1:0] WA, RAA, RBA, SH;
    wire [2:0] ALU;

    CU control (IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE, ~start, clk, ~restart, nEqZero);
    DP datapath (nEqZero, Sum, n, clk, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE);

    assign done = OE;

endmodule
```

Experiment #3: Testbench

Verilog code for GDP test bench for waveform verification

```
module GDP_tb;
    wire displayRes;
    reg start, restart, clk;
    wire [7:0] sum;
    reg [7:0] nIn;

    integer i;

    initial
     begin
        start = 1;
        restart = 1;
        clk = 0;

        forever
          #2 clk = ~clk;
    end

    always
    begin
        for(i = 0; i < 23; i = i + 1)  // 22 is max to not cause overflow
        begin
            nIn <= i;
            start = 0;
            #50 restart = 1;

            while(displayRes != 1) // Wait for completion signal
                #5 begin end
$write("Input: %2d,  Result: %3d,  Expected: %3d:  ", nIn, i*(i+1)/2, sum);

             if(sum == (i*(i+1)/2))          //calculates the sum of 1,2,3,4,5...n
                 $display("Correct!");
             else
                 $display("Incorrect!");

             restart = 0;
             start = 1;
         end

         $stop;
     end

//Initialize GDP
GDP main (sum, start, restart, clk, nIn, displayRes);

endmodule
```

Waveform Verification



Terminal Verification

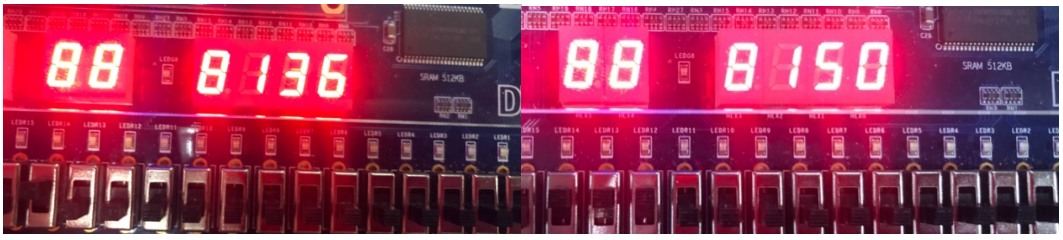
```
VSIM 17> run -all
# Input:  0, Result:   0, Expected:   0: Correct!
# Input:  1, Result:   1, Expected:   1: Correct!
# Input:  2, Result:   3, Expected:   3: Correct!
# Input:  3, Result:   6, Expected:   6: Correct!
# Input:  4, Result:  10, Expected:  10: Correct!
# Input:  5, Result:  15, Expected:  15: Correct!
# Input:  6, Result:  21, Expected:  21: Correct!
# Input:  7, Result:  28, Expected:  28: Correct!
# Input:  8, Result:  36, Expected:  36: Correct!
# Input:  9, Result:  45, Expected:  45: Correct!
# Input: 10, Result:  55, Expected:  55: Correct!
# Input: 11, Result:  66, Expected:  66: Correct!
# Input: 12, Result:  78, Expected:  78: Correct!
# Input: 13, Result:  91, Expected:  91: Correct!
# Input: 14, Result: 105, Expected: 105: Correct!
# Input: 15, Result: 120, Expected: 120: Correct!
# Input: 16, Result: 136, Expected: 136: Correct!
# Input: 17, Result: 153, Expected: 153: Correct!
# Input: 18, Result: 171, Expected: 171: Correct!
# Input: 19, Result: 190, Expected: 190: Correct!
# Input: 20, Result: 210, Expected: 210: Correct!
# Input: 21, Result: 231, Expected: 231: Correct!
# Input: 22, Result: 253, Expected: 253: Correct!
# Break in Module GDP_tb at C:/Users/oit/Desktop/Lab_7/GDP_tb.v line 42
```

Experiment #4: Datapath on DE2

Verilog Code for DE2 Board Implementation

```
module GDP_7seg(clk, rst_n, start, result, nIn, hout, tout, oout);
  input    clk;
  input    rst_n; // Active-low reset
  input    start;
  input  [7:0]nIn;
  output [7:0] result;
  output [6:0] hout;
  output [6:0] tout;
  output [6:0] oout;
  wire done;

  GDP u0(.Sum(result), .start(start), .restart(rst_n), .clk(clk), .n(nIn), .done(done));
  threeSevenSeg seg2 ((result / 100), hout);
  threeSevenSeg seg1 (((result % 100) /10), tout);
  threeSevenSeg seg0 (((result % 100) % 10), oout);

endmodule
module threeSevenSeg(a, out);
 input [7:0]a;
 output reg [6:0]out;

 always@(*)
 case (a)
 8'b00000000: out = 7'b1000000; //0
 8'b00000001: out = 7'b1111001; //1
 8'b00000010: out = 7'b0100100; //2
 8'b00000011: out = 7'b0110000; //3
 8'b00000100: out = 7'b0011001; //4
 8'b00000101: out = 7'b0010010; //5
 8'b00000110: out = 7'b0000010; //6
 8'b00000111: out = 7'b1111000; //7
 8'b00001000: out = 7'b0000000; //8
 8'b00001001: out = 7'b0011000; //9
 default: out = 7'b1111111; // turn off
 endcase

 endmodule
```

DE2 Board Implementation



4. Question

1. What is a control unit and what role does it play in the implementation of the summation algorithm?

- Control unit is the component that takes input information and sends its control signals to the GDP/processor to respond to certain program instructions. The control unit for the summation algorithm takes input n and sends control words to the GDP depending on the value of n. The value of sum will be calculated as long as n != 0. Once n = 0 then the summation algorithm is completed, and the control unit will send the control signal to the GDP in order to output the value of the sum.

5. Conclusions

- To conclude this lab experiment, we created the control unit for the GDP that was designed in lab experiment 6. The algorithm that was implemented in the control unit was the summation algorithm. The only encountered problem that we faced was trying to display the GDP/control unit on the DE2 board. But after further inspecting the needed inputs/outputs for the DE2 implementation we were able to overcome this problem. Overall for this lab experiment, we created the control unit, the top-level module with all instantiations, testbench for waveform verification and DE2 board implementation.

Class:	CPE-300L	Semester:	Spring 2020

Points		Document author:	Brysen Kokubun
		Author's email:	Brysenkokubun96@gmail.com
		Document topic:	Prelab 7

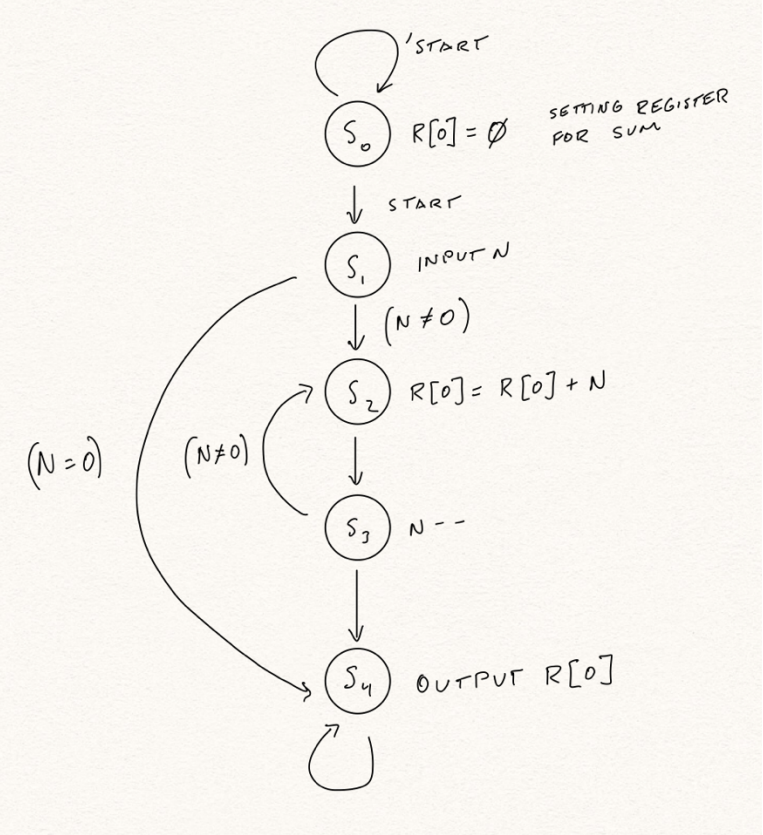
Instructor's comments:

Introduction / Theory of operation

- For this lab experiment, we will be designing the control unit for the GDP that was designed in lab experiment #6. We will be creating the CU module to implement the summation algorithm, instantiate it within the top-level GDP module, verify it with a testbench and waveform and display it on the DE2 board.

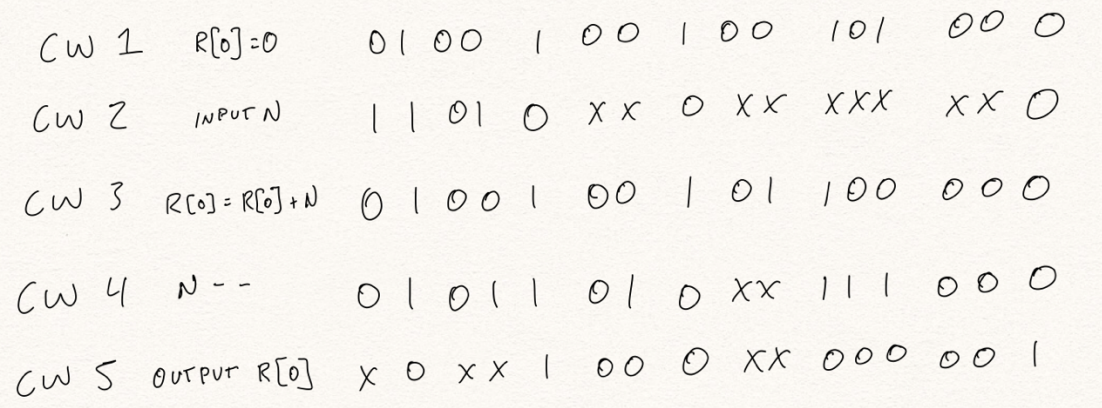
Prelab main content

1. Implement a state graph for the summation algorithm and verify it with fig 4. Is it possible to have a different state graph than the one mentioned in fig 4?



It is not possible to have a different state graph than the one mentioned in fig 4. This is because the state graph follows specific steps, states and next states in order to implement the algorithm.

2. Implement the control words for the state graph that you created in 1 and verify it with fig 6. Is it possible to have a different control words than the one mentioned in fig 6?



It is possible to have different control words because someone may use different registers to store the value of sum and input n. Changing the register will cause a change in the control words. Instead of 00, one may use 01, 10, or 11 this will cause the control words to be different than the control words that I created.