Name: Joseph Hany **ID:900182870**

Architecture Lab 3 Report

2) A technical summary of experiments conducted in the lab (steps, results, components, code functionality, etc.)

- **Experiment 1:**
  This experiment aims to implement an N-bit register with reset and load control
- In order to implement such a circuit, we should do the following:
  1- Create three module: DFlipFlop, MUX_2_to_1, n_bit_register
  2- DFlipFlop module acts as a single bit register that saves its value until D changes from Q in any positive clock edge
  3- MUX_2_to_1 module selects wich value will enter the flipflop from D depending on the selector of the MUX.
  4- n_bit_register module assembles all the muxes and flipflops to acts as one block that can store n bits and have a reset input which if was high, all the flipflops will reset to zero.

  For example:
  In this block of code, we generate n 2 by1 muxes and n DFipFlops that all together act as an n bit register

```
  generate
     for(i=0;i<n;i=i+1)
     begin:RegUnit
        MUX_2_to_1 mux(.A(Q[i]),.B(D[i]),.select(load),.C(C[i]));
        DFlipFlop flipflop(.clk(clk),.rst(reset),.D(C[i]),.Q(Q[i]));
     end
  endgenerate
```

- **Experiment 2:**
  This experiment aims to implement an n-bit 2x1 multiplexer which chooses between two n-bit inputs according to the input selector
- In order to implement such a circuit, we should do the following:
  1- Create two module: TwoByOneMux, nbitMux
  2- TwoByOneMux module is a normal 2 by 1 mux that selects one of two inputs depending on the input selector.
  3- nbitMux module which instantiates TwoByOneMux module n times in order to choose between the two n-input bits and will output either one of them depending on the selector.

For example:

```
generate for(i=0;i<N;i=i+1)
TwoByOneMux m(a[i],b[i],s,c[i]); // we just generate n two by one muxes in order to
                                                  //choose only one input from the two n-bits input
endgenerate
```

- **Experiment 3:**
  This experiment aims to implement a circuit which shifts the bits to the lift by 1 place
- In order to implement such a circuit, we should do the following:
  1- Create one module: A n-bit shift left 1 module.
  2- This n-bit shift left 1 module is nothing but a rewiring for the n-bits input, where we connect the least significant bit (i.e. number 0) to bit number 1 (the wire on the left), and so on until we leave one disconnected wire which is the left-most (most significant) bit. Besides, we connect the least significant bit of the other bus to the ground.
  For example:

```
always @(*)
   begin
   for(i=1;i<=31;i=i+1)
   begin
    ShiftOut2[i]=In_Shift[i-1]; //here, we rewire each input to the wire right to the left to it
   end
   ShiftOut2[0]=1'b0; // then we connect the right most output wire to the ground since it
                             //has no input
   end
```

- **Experiment 4:**
  This experiment aims to implement a circuit which generates and reassembles the immediate bits that are found in the types of different instructions and then extends the sign on the most significant bit in the immediate bus
- In order to implement such a circuit, we should do the following:
  1- Create one module: an ImmGen module.
  2- This module assembles the scattered immediate bits in the different instructions depending on the instruction itself.
  For example:

```
if(inst[6]) begin                    // in case of BEQ instruction
   gen_out[11]=inst[31];
   gen_out[10]=inst[7];            // a concatenation of bits 31, 7, 30:25, and 11:8 for BEQ
```

```
        gen_out[9:4]=inst[30:25];
        gen_out[3:0]=inst[11:8];
        gen_out[31:12]={20{gen_out[11]}};
        end
    else if(inst[5]) begin             // in case of  SW instruction
        gen_out[11:5]=inst[31:25];
        gen_out[4:0]=inst[11:7];       // a concatenation of bits 31:25 and 11:7 for SW
        gen_out[31:12]={20{gen_out[11]}};
        end
    else begin                         // in case of  LW instruction
        gen_out[11:0]=inst[31:20]; //  bits 31:20 for LW
        gen_out[31:12]={20{gen_out[11]}};
        end
```

3) Verilog code for all modules of each experiment.

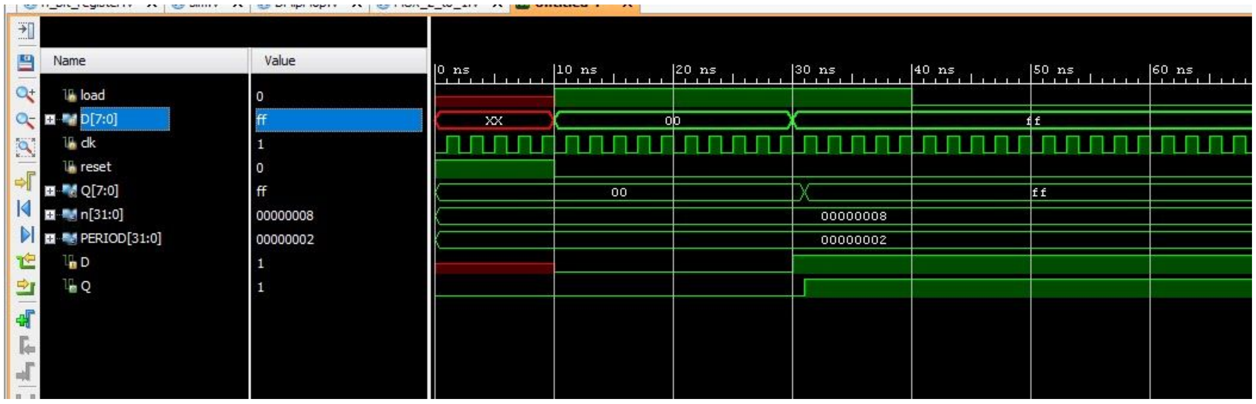
- **Verilog code of each experiment is attached in the zip file**

4) Verilog code for the testbench module of each experiment's top-level module.

- **Verilog code for the testbench is attached in the zip file**

5) Snapshot of simulation output:

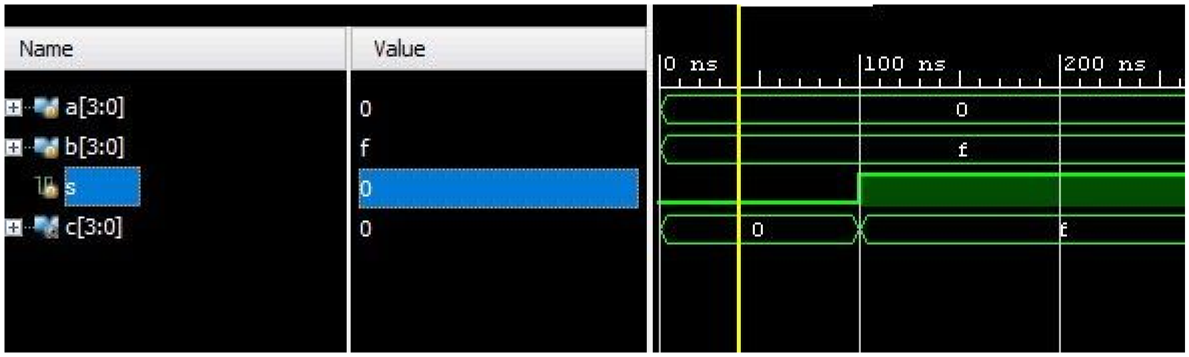
- **EXP1 Simulation Snapshot:**



Here, we can see that when:

- Reset = 0 → Q=8'b00000000
- reset=0 & D=8'b00000000 & load=1 → Q=8'b00000000
- D=8'b00000000 & load=1 → Q=8'b00000000
- D=8'b11111111 & load=1 → Q=8'b11111111 (after the first positive clock edge)
- D=8'b11111111 & load=0 → Q=(last Q)= 8'b11111111

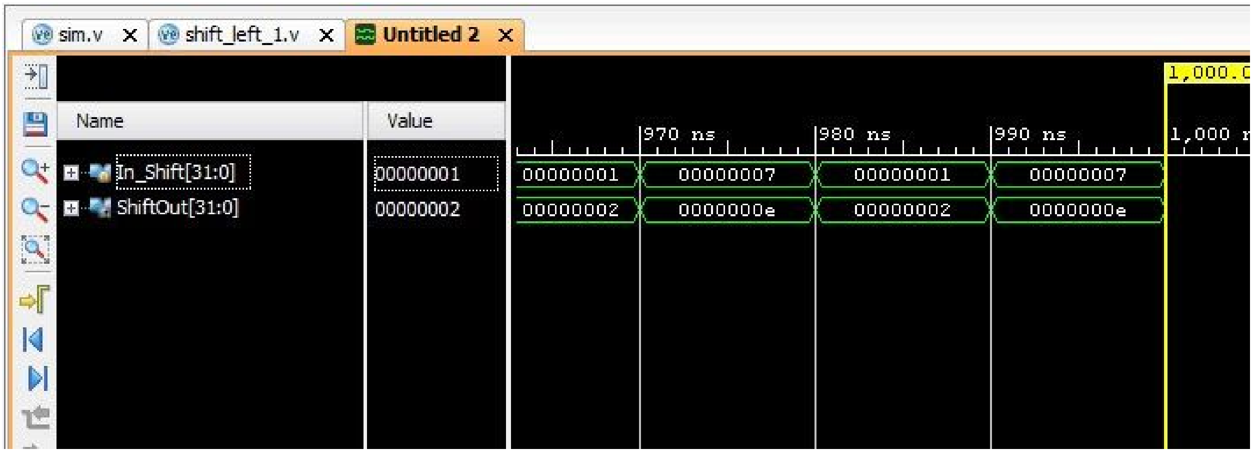
- **EXP2 Simulation Snapshot:**



Here, we can see that when:

- a=4'b0000 & b=4'b1111 & s=1'b0 → since selection =0 so the output c=4'b0000
- s=1'b1 → since selection =1 so the output equal to the second input c=4'b1111

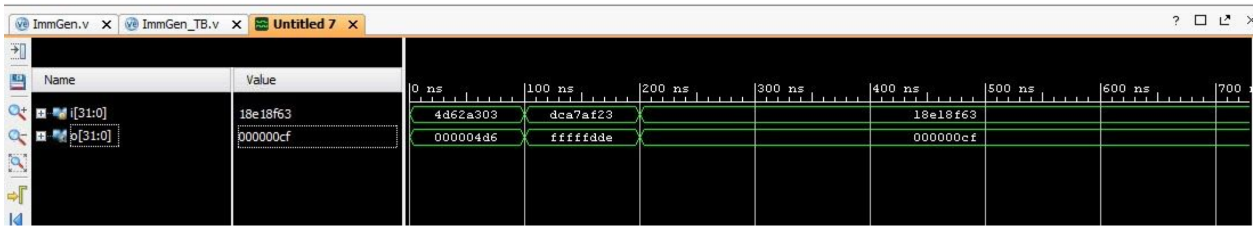
- **EXP3 Simulation Snapshot:**



Here, we can see that when:

- In_Shift=32'b00000000000000000000000000000001 (1 in decimal)→ ShiftOut=32'b00000000000000000000000000000010 (2 in decimal)
- In_Shift=32'b00000000000000000000000000000111 (7 in decimal)→ ShiftOut=32'b00000000000000000000000000001110 (14 in decimal= e in hexa)

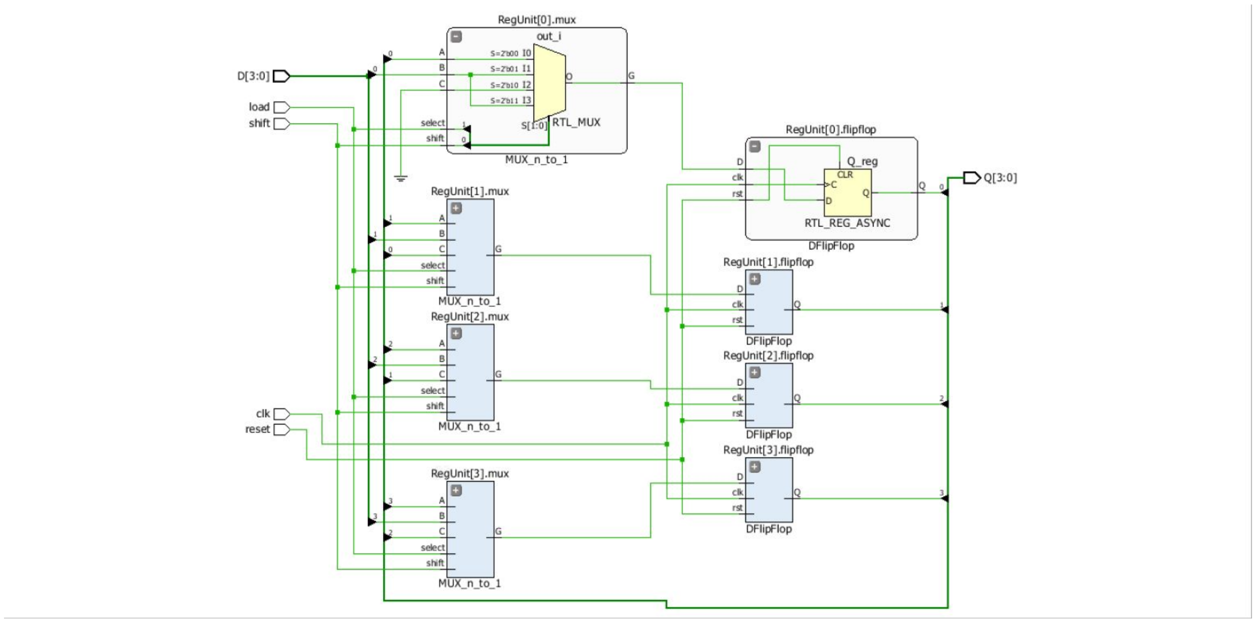
- **EXP4 Simulation Snapshot:**



Here, we can see that when:

- i=32'b010011010110_00101_010_00110_0000011 (LW) → 1238
- i=32'b1101110_01010_01111_010_11110_0100011 (SW) → -546
- i=32'b0001100_01110_00011_000_11110_1100011 (BEQ) → 207

6) Schematic design to our N-Bit register to allow extra input Shift:
- If (shift control==1 && load==0) shifts the data in the register 1 bit to the left.
- If (shift control==1 && load==1) load data ignore shifting


RegUnit[0].mux
out_i
RTL_MUX
MUX_n_to_1
RegUnit[0].flipflop
Q_reg
RTL_REG_ASYNC
DFlipFlop
D[3:0]
load
shift
clk
reset
Q[3:0]
RegUnit[1].mux
RegUnit[2].mux
RegUnit[3].mux
RegUnit[1].flipflop
RegUnit[2].flipflop
RegUnit[3].flipflop