HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY – VNU HCMC
OFFICE FOR INTERNATIONAL STUDY PROGRAM
FACULTY OF ELECTRICAL AND ELECTRONIC ENGINEERING

*

DIGITAL SYSTEMS (LAB)
EXPERIMENTAL REPORT (Lab 5)

Lecturer : **Mr. Nguyễn Tuấn Hùng**
Subject : **Digital Systems**
Class : **TT06**
Name : **Lương Triển Thắng**
Student ID : **2051194**

Ho Chi Minh City, 14th June, 2022

Contents

V Laboratory 5

A simple processor

1. Design and implement a simple processor

a. Code

DFF.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY DFFn IS
        PORT (
                D, clk, rst, en : IN STD_LOGIC;
                Q : OUT STD_LOGIC);
END DFFn;
ARCHITECTURE Behavior OF DFFn IS
BEGIN
        PROCESS (rst, clk, en)
        BEGIN
                IF rst = '1' THEN
                        Q <= '0';
                ELSIF rising_edge(clk) AND en = '1' THEN
                        Q <= D;
                END IF;
        END PROCESS;
END Behavior;
```

myReg.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY myReg IS
        PORT (
                clk, rst, en : IN STD_LOGIC;
                D : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                Q : OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
        );
END myReg;

ARCHITECTURE arch OF myReg IS
        COMPONENT DFFn IS
                PORT (
                        D, clk, rst, en : IN STD_LOGIC;
                        Q : OUT STD_LOGIC);
        END COMPONENT;
BEGIN
        gen : FOR i IN 8 DOWNTO 0 GENERATE
                DFFs : DFFn PORT MAP(D(i), clk, rst, en, Q(i));
        END GENERATE;
END ARCHITECTURE;
```

RegMUX.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
```

```
ENTITY regMUX IS
        PORT (
                SIGNAL R0, R1, R2, R3, R4, R5, R6, R7 : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                SIGNAL g, DIN : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                SIGNAL sel : IN INTEGER RANGE 0 TO 10;
                SIGNAL MUXout : OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
        );
END regMUX;

ARCHITECTURE arch OF regMUX IS
BEGIN
        WITH sel SELECT
                MUXout <=
                R0 WHEN 0,
                R1 WHEN 1,
                R2 WHEN 2,
                R3 WHEN 3,
                R4 WHEN 4,
                R5 WHEN 5,
                R6 WHEN 6,
                R7 WHEN 7,
                g WHEN 8,
                DIN WHEN 9,
                "000000000" WHEN 10;
END ARCHITECTURE;
```

AddSub.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY AddSub IS
    PORT (
        SIGNAL A, B : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
        SIGNAL sel : IN STD_LOGIC;
        SIGNAL C : OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
    );
END AddSub;

ARCHITECTURE arch OF AddSub IS
BEGIN
C <= STD_LOGIC_VECTOR(UNSIGNED(A) + UNSIGNED(B)) WHEN sel = '0' ELSE
        STD_LOGIC_VECTOR(UNSIGNED(A) - UNSIGNED(B));
END ARCHITECTURE;
```

ctrlunitFSM.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY ctrlunitFSM IS
        PORT (
                SIGNAL rin : OUT STD_LOGIC_VECTOR(0 TO 7);
                SIGNAL ain, gin, IRin, addsub : OUT STD_LOGIC;
                SIGNAL IR : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                SIGNAL outsel : OUT INTEGER RANGE 0 TO 10;
                SIGNAL run, rst, clk : IN STD_LOGIC;
                SIGNAL done : BUFFER STD_LOGIC
        );
END ctrlunitFSM;
```

```
ARCHITECTURE arch OF ctrlunitFSM IS
        TYPE State_type IS (T0, T1, T2, T3);
        SIGNAL Tstep_Q, Tstep_D : State_type;

        SIGNAL opcode : STD_LOGIC_VECTOR(2 DOWNTO 0);
        SIGNAL RX, RY : NATURAL RANGE 0 TO 7;
BEGIN
        opcode <= IR(8 DOWNTO 6);
        RX <= TO_INTEGER(UNSIGNED(IR(5 DOWNTO 3)));
        RY <= TO_INTEGER(UNSIGNED(IR(2 DOWNTO 0)));
        statetable :
        PROCESS (Tstep_D, run)
        BEGIN
                CASE Tstep_D IS
                        WHEN T0 =>
                                IF run = '0' THEN
                                        Tstep_Q <= T0;
                                ELSE
                                        Tstep_Q <= T1;
                                END IF;
                                -- data IS loaded into IR IN this TIME step
                        WHEN T1 =>
                                IF done = '1' THEN
                                        TStep_Q <= T0;
                                ELSE
                                        Tstep_Q <= T2;
                                END IF;
                        WHEN T2 =>
                                Tstep_Q <= T3;
                        WHEN T3 =>
                                IF done = '1' THEN
                                        TStep_Q <= T0;
                                ELSE
                                        -- ERROR
                                END IF;
                        WHEN OTHERS => NULL;
                END CASE;
        END PROCESS;

        controlsignals :
        PROCESS (Tstep_D, IR, opcode, RX, RY)
        BEGIN
                rin <= (OTHERS => '0');
                ain <= '0';
                gin <= '0';
                outsel <= 10;
                addsub <= '0';
                CASE Tstep_D IS
                        WHEN T0 => -- store DIN IN IR as long as Tstep_Q = 0
                                IRin <= '1';
                                done <= '0';
                                outsel <= 9;
                        WHEN T1 =>
                                CASE opcode IS
                                        WHEN "000" =>
                                                outsel <= RY;
                                                done <= '1';
                                                rin(RX) <= '1';
                                        WHEN "001" =>
                                                outsel <= 9;
                                                done <= '1';
                                                rin(RX) <= '1';
```

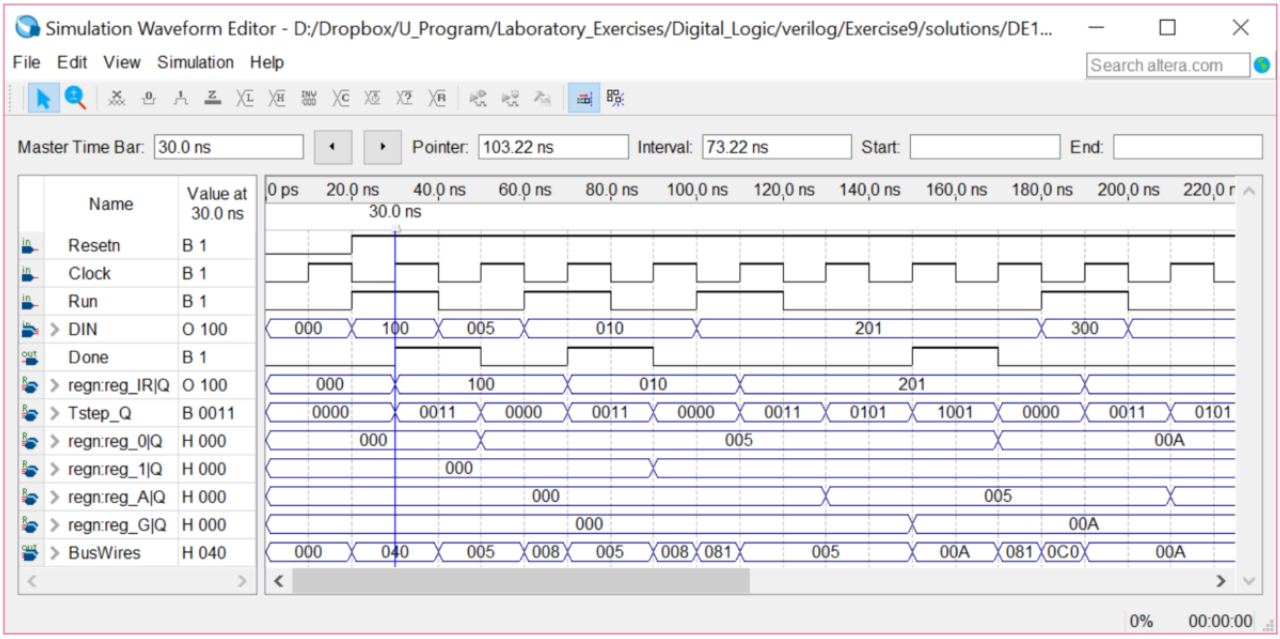
```
                                        WHEN "010" | "011" =>
                                                outsel <= RX;
                                                ain <= '1';
                                        WHEN OTHERS => NULL;
                                END CASE;
                        WHEN T2 =>
                                CASE opcode IS
                                        WHEN "010" =>
                                                outsel <= RY;
                                                gin <= '1';
                                        WHEN "011" =>
                                                outsel <= RY;
                                                gin <= '1';
                                                addsub <= '1';
                                        WHEN OTHERS => NULL;
                                END CASE;
                        WHEN T3 =>
                                CASE opcode IS
                                        WHEN "010" | "011" =>
                                                outsel <= 8;
                                                rin(RX) <= '1';
                                                done <= '1';
                                        WHEN OTHERS => NULL;
                                END CASE;
                        WHEN OTHERS => NULL;
                END CASE;
        END PROCESS;
        flipflops : PROCESS (clk, rst, Tstep_Q)
        BEGIN
                IF rst = '1' THEN
                        Tstep_D <= T0;
                ELSIF rising_edge(clk) THEN
                        Tstep_D <= Tstep_Q;
                END IF;
        END PROCESS;
END ARCHITECTURE;
```

Exc1.vhd

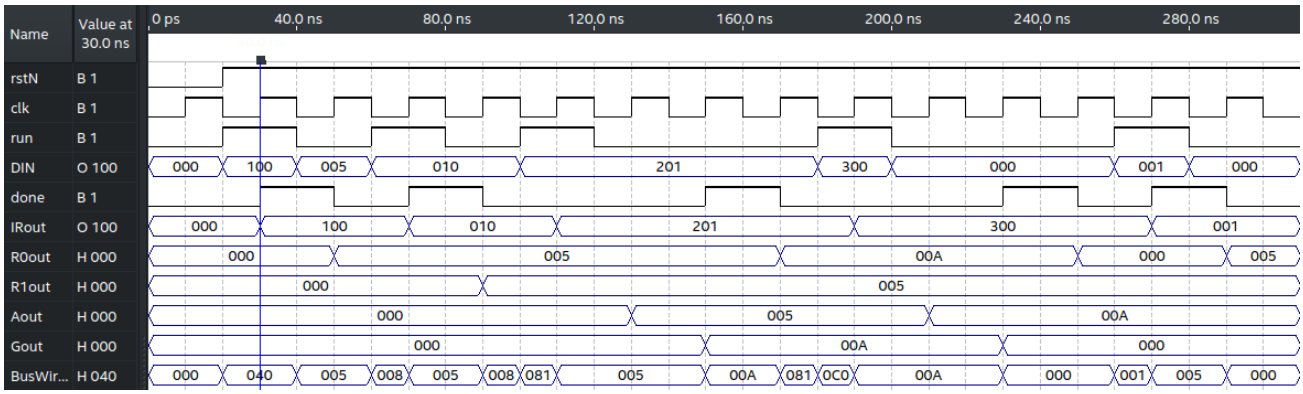
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Exc1 IS
        PORT (
                R0out, R1out, Aout, Gout, IRout : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
                DIN : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                rstN, clk, run : IN STD_LOGIC;
                done : BUFFER STD_LOGIC;
                BusWires : BUFFER STD_LOGIC_VECTOR(8 DOWNTO 0)
        );
END Exc1;

ARCHITECTURE Behavior OF Exc1 IS
        -- MISC
        SIGNAL addOrSub : STD_LOGIC;

        -- REGISTERS, A, G, IR
        SIGNAL regin : STD_LOGIC_VECTOR(0 TO 7);
        SIGNAL A, G, Gt : STD_LOGIC_VECTOR(8 DOWNTO 0);
        SIGNAL ain, gin, IRin, rst : STD_LOGIC;
        SIGNAL R0, R1, R2, R3, R4, R5, R6, R7, IR : STD_LOGIC_VECTOR(8 DOWNTO 0);
        SIGNAL sel : INTEGER RANGE 0 TO 10;
```

```
        COMPONENT myReg IS
                PORT (
                        clk, rst, en : IN STD_LOGIC;
                        D : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                        Q : OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
                );
        END COMPONENT;

        COMPONENT AddSub IS
                PORT (
                        SIGNAL A, B : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                        SIGNAL sel : IN STD_LOGIC;
                        SIGNAL C : OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
                );
        END COMPONENT;

BEGIN
        R0out <= R0;
        R1out <= R1;
        Aout <= A;
        Gout <= G;
        IRout <= IR;

        rst <= NOT(rstN);
        -- ADDSUB
        as : AddSub PORT MAP(A, BusWires, addOrSub, Gt);

        -- REGS
        reg0 : myReg PORT MAP(clk, rst, regin(0), BusWires, R0);
        reg1 : myReg PORT MAP(clk, rst, regin(1), BusWires, R1);
        reg2 : myReg PORT MAP(clk, rst, regin(2), BusWires, R2);
        reg3 : myReg PORT MAP(clk, rst, regin(3), BusWires, R3);
        reg4 : myReg PORT MAP(clk, rst, regin(4), BusWires, R4);
        reg5 : myReg PORT MAP(clk, rst, regin(5), BusWires, R5);
        reg6 : myReg PORT MAP(clk, rst, regin(6), BusWires, R6);
        reg7 : myReg PORT MAP(clk, rst, regin(7), BusWires, R7);
        Areg : myReg PORT MAP(clk, rst, ain, BusWires, A);
        Greg : myReg PORT MAP(clk, rst, Gin, Gt, G);
        IRreg : myReg PORT MAP(clk, rst, IRin, DIN, IR);

        -- MUX
        regMUX0 : ENTITY work.RegMUX PORT MAP(R0, R1, R2, R3, R4, R5, R6, R7, G, DIN, sel, BusWires);

        -- FSM
        ControlUnit : ENTITY work.ctrlUnitFSM PORT MAP(
                regin, ain, gin, IRin, addorsub, IR, sel, run, rst, clk, done
        );

END ARCHITECTURE;
```

b. Waveform

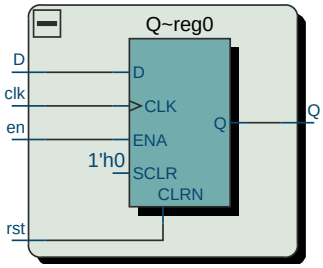


Reference waveform

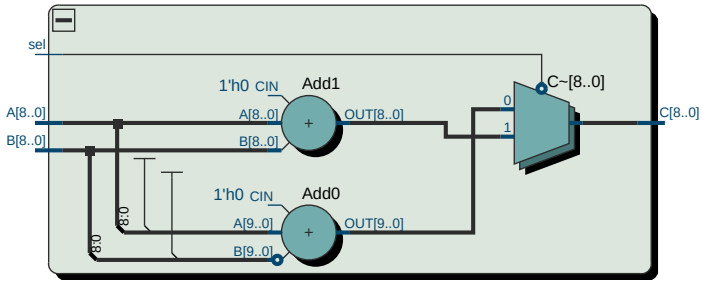


Simulation result

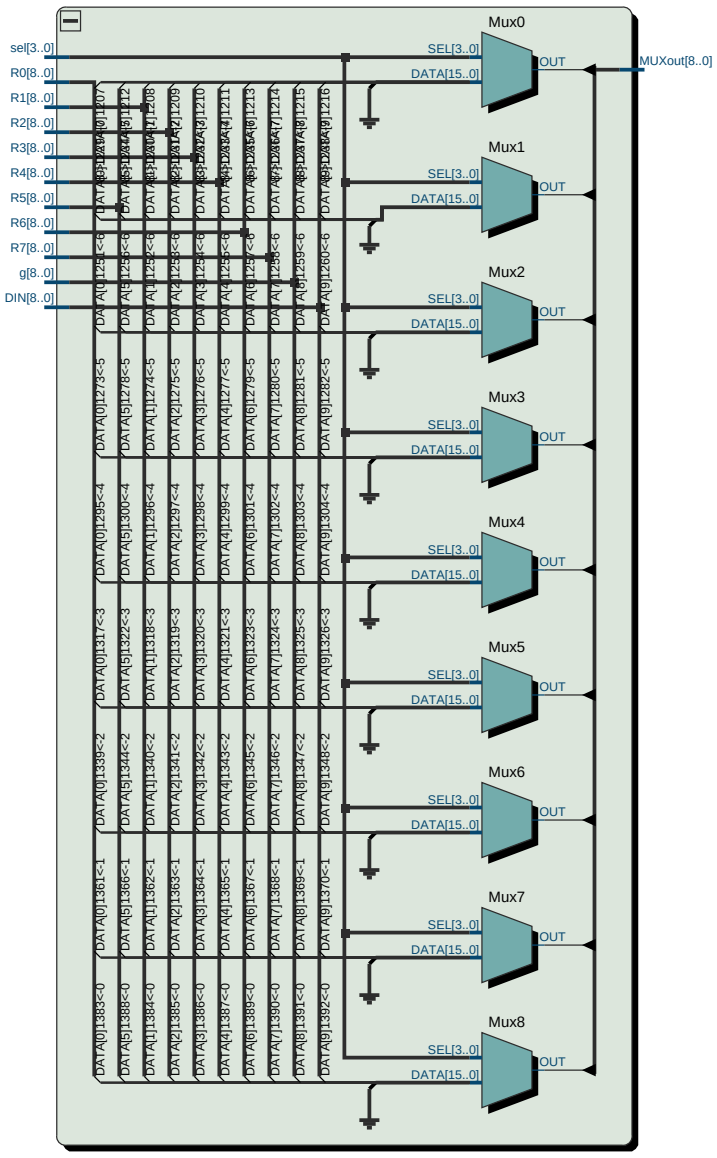
c. Result of RTL viewer



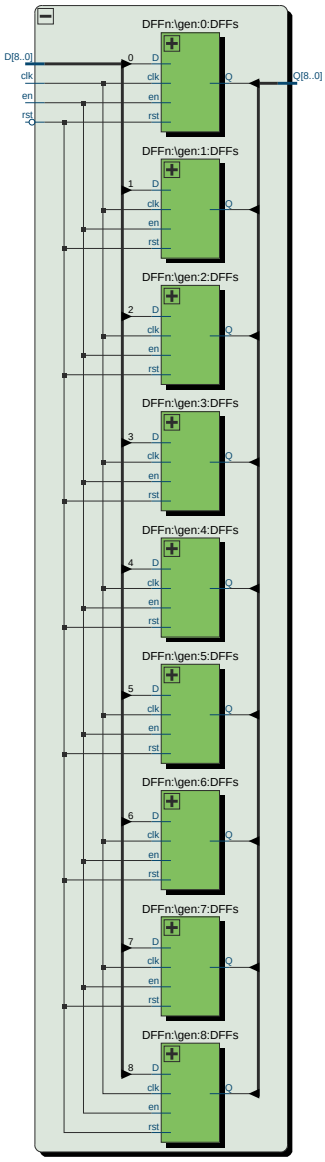
D flip-flop



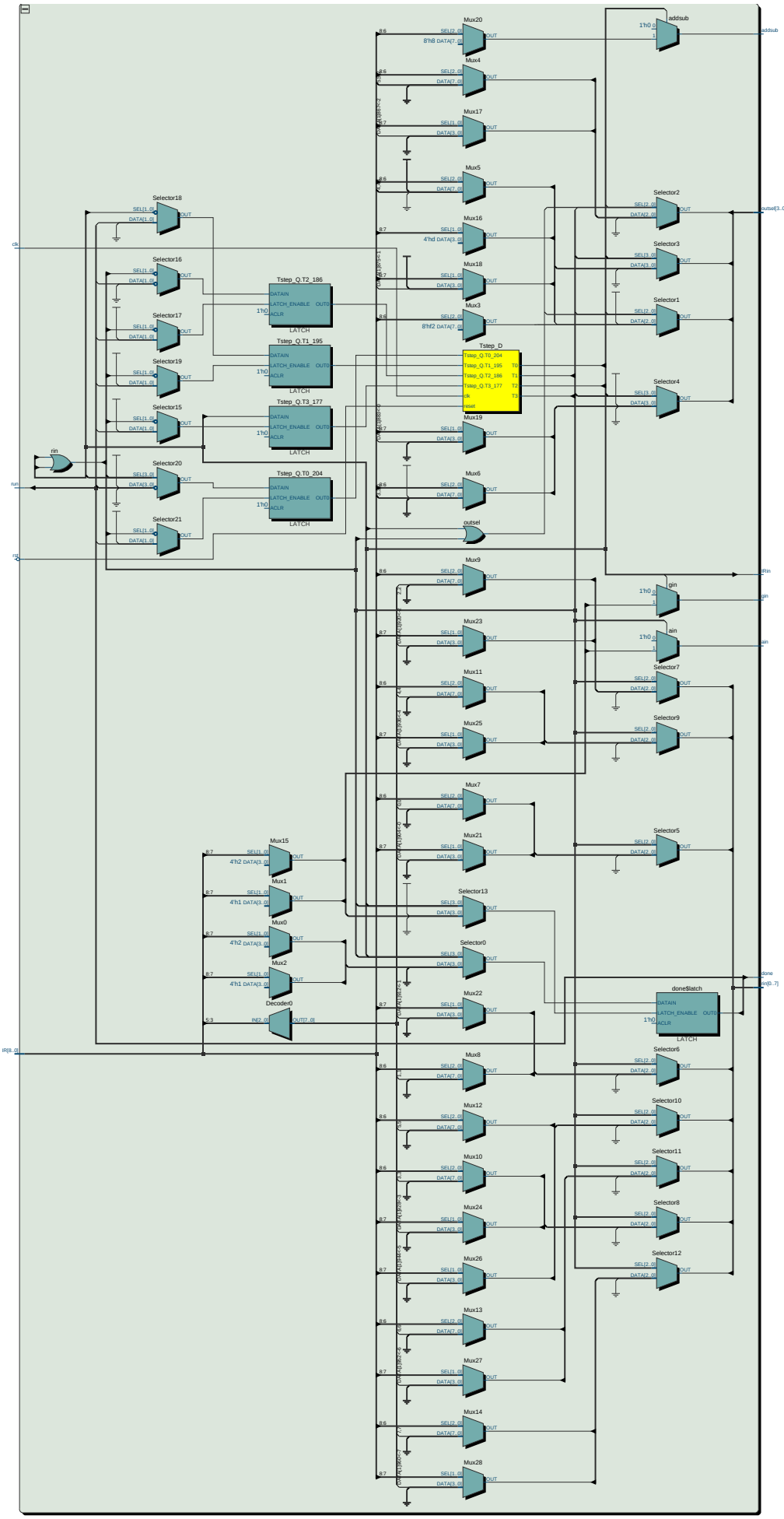
Add-Sub



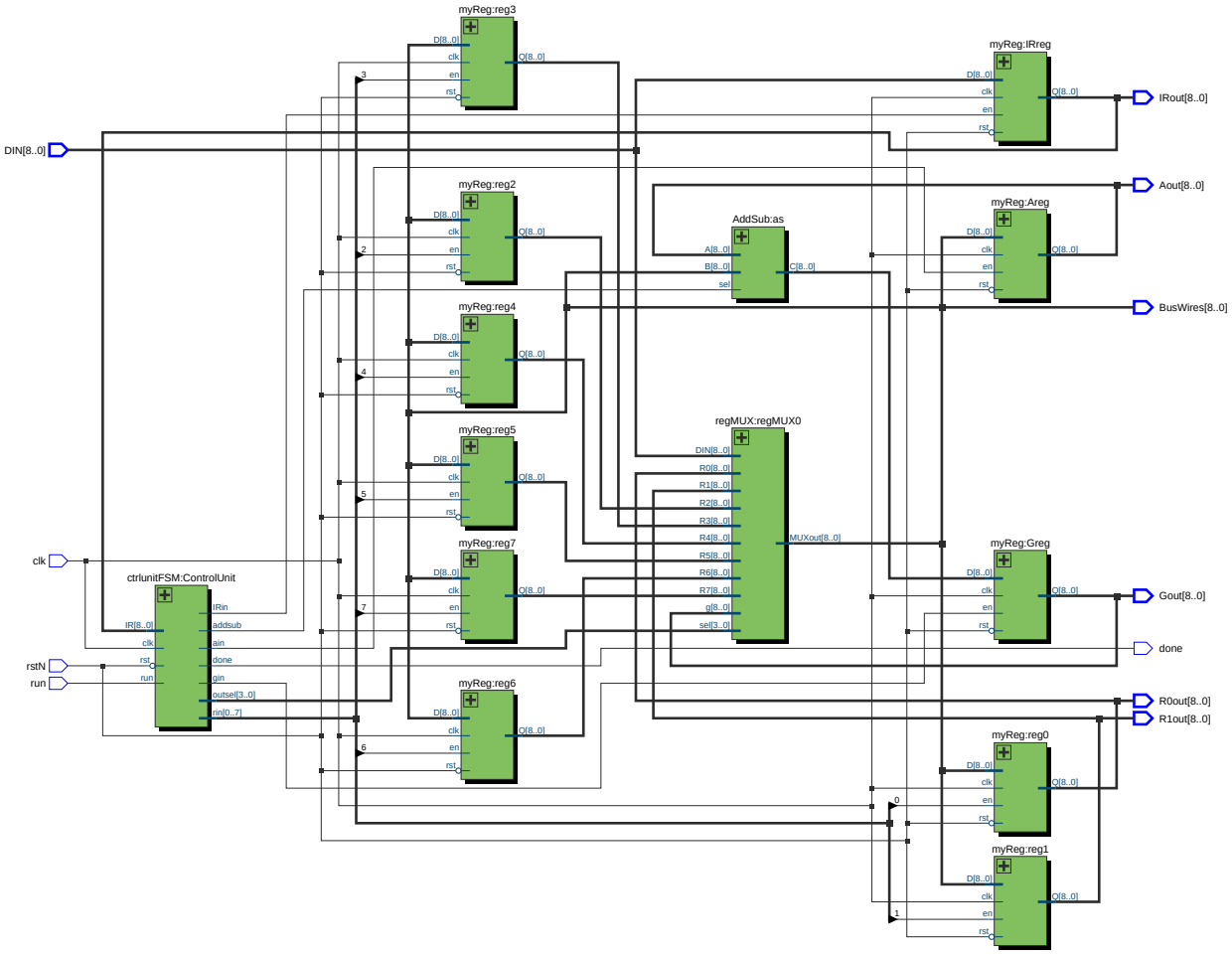
Registers MUX



9-bit register



Control Unit (zoom in for better viewing)



Top level (zoom in for better viewing)

2. Design and implement a simple processor with memory

This exercise is just an extenstion of the previous, so I just propose the additional parts.

I will change the circuit, so that '`run`' will be not required to set manually in waveform, but in the ROM itself instead. A ROM data now contains 10 bits with MSB is '`run`'. See the code section for the ROM and waveform to learn more.

A better version of this processor will be introduced in the report of Lab6.

a. Code

myROM.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY myROM IS
        GENERIC (
                addr_width : INTEGER := 32; -- store 32 elements
                addr_bits : INTEGER := 5; -- required bits to store 32 elements
                data_width : INTEGER := 10 -- each element has 10-bits
        );
        PORT (
                addr : IN STD_LOGIC_VECTOR(addr_bits - 1 DOWNTO 0);
                data, nextData : OUT STD_LOGIC_VECTOR(data_width - 1 DOWNTO 0)
        );
END myROM;

ARCHITECTURE arch OF myROM IS
        TYPE rom_type IS ARRAY (0 TO addr_width - 1) OF STD_LOGIC_VECTOR(data_width - 1 DOWNTO 0);
        SIGNAL user_ROM : rom_type;
        SIGNAL address : INTEGER RANGE 0 TO addr_width - 1;

        ATTRIBUTE ram_init_file : STRING;
        ATTRIBUTE ram_init_file OF user_ROM : SIGNAL IS "rom_data.mif";

BEGIN
        address <= to_integer(unsigned(addr));
        data <= user_ROM (address);
        nextData <= user_ROM (address + 1);
END arch;
```

rom_data.mif

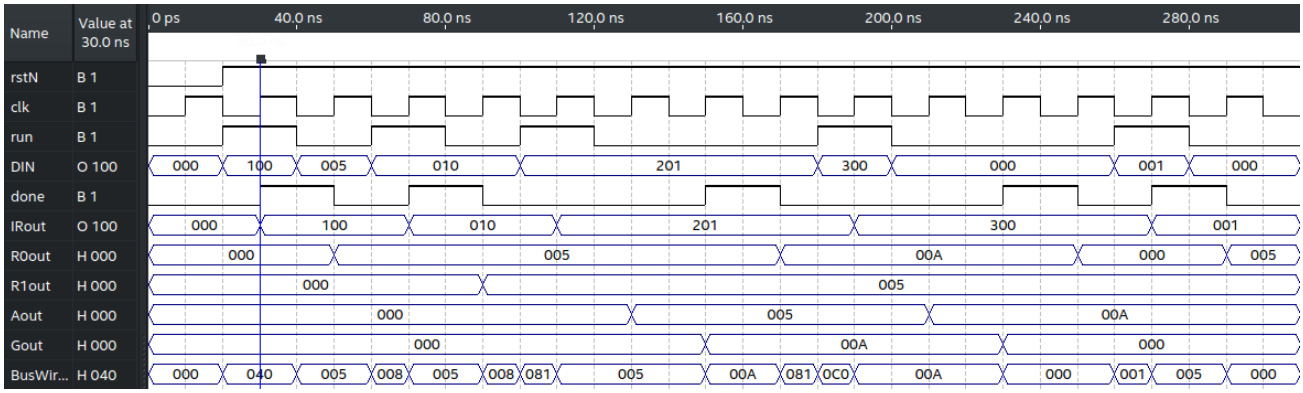
```
WIDTH=10;
DEPTH=32;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
        [0..31]  :   0000000000;
        0 : 0000000000;
        1 : 1001000000;
        2 : 0000000101;
        3 : 1000001000;
        4 : 1010000001;
        5 : 1011000000;
        6 : 1000000001;
END;
```

Exc2.vhd

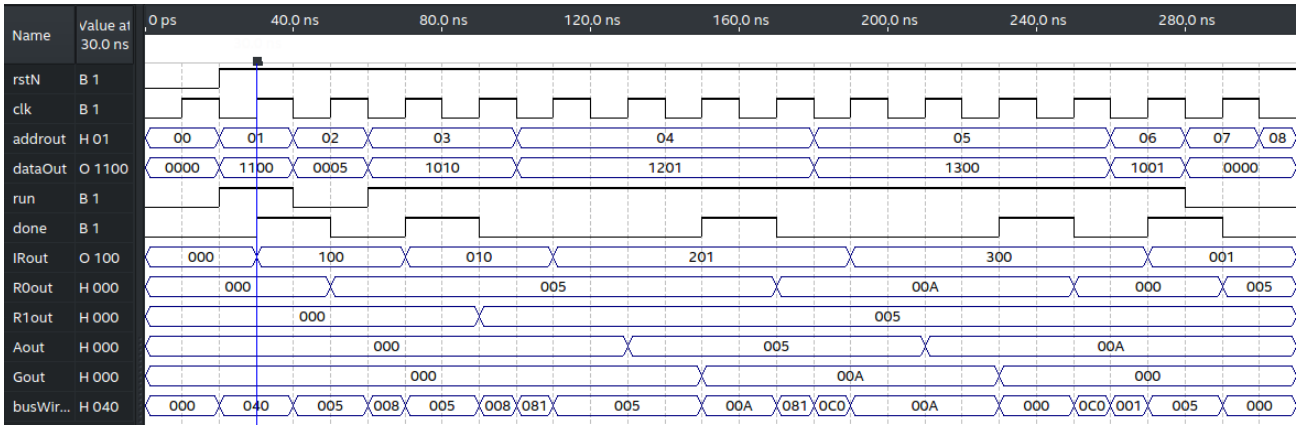
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Exc2 IS
  PORT (
    clk, rstN : IN STD_LOGIC;
    run, done : BUFFER STD_LOGIC;
    addrout : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
    dataOut : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
    ROout, R1out, Aout, Gout, IRout : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
    busWires : OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
  );
END Exc2;

ARCHITECTURE arch OF Exc2 IS
  -- signal to store received data
  SIGNAL data, nextData : STD_LOGIC_VECTOR (9 DOWNTO 0);
  SIGNAL IR : STD_LOGIC_VECTOR(8 DOWNTO 0);
  SIGNAL addr : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00000";
  SIGNAL mclk, isIR : STD_LOGIC;

  TYPE states IS (T0, T1, T2);
  SIGNAL state : states;

  COMPONENT Exc1 IS
    PORT (
      ROout, R1out, Aout, Gout, IRout : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
      DIN : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
      rstN, clk, run : IN STD_LOGIC;
      done : BUFFER STD_LOGIC;
      BusWires : BUFFER STD_LOGIC_VECTOR(8 DOWNTO 0)
    );
  END COMPONENT;
BEGIN
  user_ROM : ENTITY work.myROM PORT MAP (addr, data, nextData);

  processor : Exc1 PORT MAP(ROout, R1out, Aout, Gout, IRout, IR, rstN, clk, run, done, busWires);

  run <= data(9);
  IR <= data(8 DOWNTO 0);

  addrout <= addr;
  dataOut <= data;

  PROCESS (clk, rstN, done)
  BEGIN
    IF rstN = '0' THEN
      state <= T0;
      addr <= "00000";
    ELSIF falling_edge(clk) THEN
      CASE state IS
        WHEN T0 =>
          addr <= STD_LOGIC_VECTOR(UNSIGNED(addr) + 1);
          state <= T1;
        WHEN T1 =>
          IF done = '1' THEN
            IF nextData(9) = '0' THEN
              addr <= STD_LOGIC_VECTOR(UNSIGNED(addr) + 1);
            END IF;
            state <= T0;
```

```
        ELSE
          state <= T1;
        END IF;
      WHEN T2 => NULL;
    END CASE;
  END IF;
END PROCESS;
END arch;
```

b. Waveform

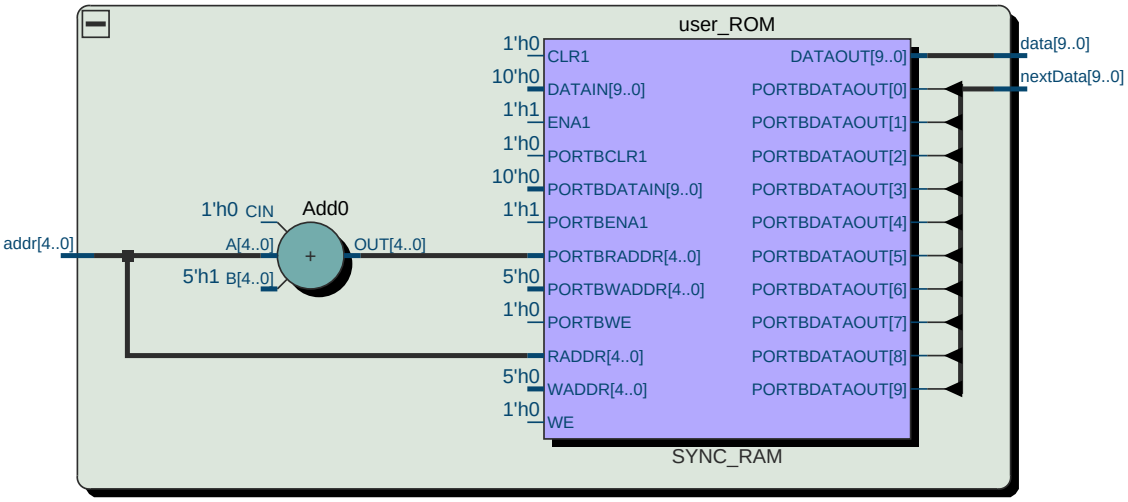


Simulation result of Exc1

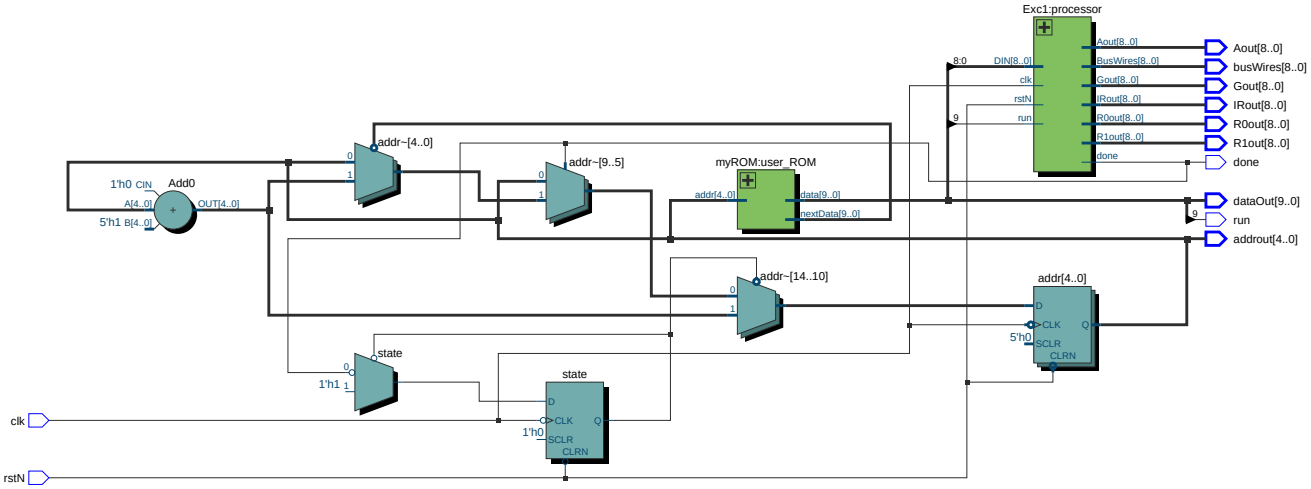


Simulation result

c. Result of RTL viewer



Read-only memory (ROM)



Top level (zoom in for better viewing)