In The Name of God
COMPUTER ENGINEERING DEPARTMENT OF AMIRKABIR UNIVERSITY OF TECHNOLOGY

Design Automation Homework - 5

Iman Tabrizian (9331032)

May 12, 2017

1 QUESTION 6

(a) Every part of a system has an execution time and every of them have a number of exection. Also every of part of the system has subsystems and also these parts have hierachy themselves. You can find this statistic using **Profiling** tools.

For codesign we divide the program into two parts. One part is given to the software to execute and the other is given to the hardware. This act is called **Partioning**.

(b) Implementation using software has the benefit of sequential execution and also has the benefit of reusing the existing hardware components, however hardware implementation has the benefit of concurrency and faster execution time but it consumes more resources. It is recommended that for the parts that take a lot of time, use the hardware implementation while for the parts that don't consume much time use the software implementation.

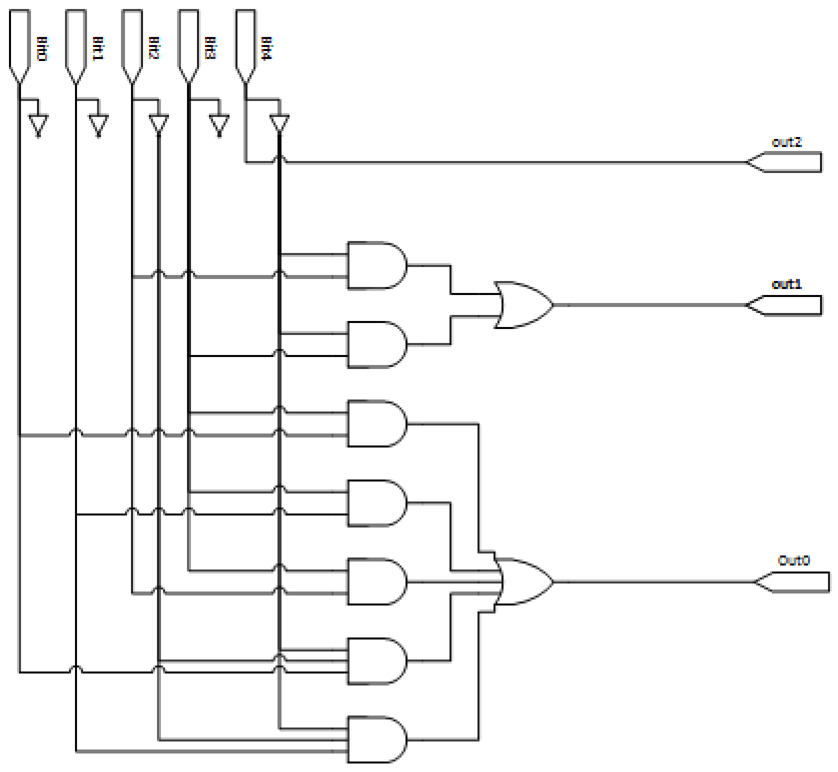
(c) **Hard Core** is an actual integrated circuit that does the job of central processing unit, while **Soft Core** is implemented in FPGA using the LUTs and there is no actual circuit.

If we implement a core by ourselves it has the overhead of not being optimized. Because both of the above cores are fully optimized. The hard core is optimized using electrical elements and the soft core is optimized by the vendor that provides it.

(d)
  a) Point to Point: In this mode the perphial device is directly connected to microprocessor. It uses FIFO and transmits data in bytestreams.
  b) Bus: In this mode there are multiple perphial device on the same bus. There is a bus arbiter whose job is to determine the periphial device to be connected.

2 QUESTION 7

The circuit that I have found by my self is like below:



In the above circuit the MSB is *bit4* and in the output also the MSB is *out2*.

```
library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity square_root is
    port(
            input: in std_logic_vector(4 downto 0);
            output: out std_logic_vector(2 downto 0)
        );
end entity;

architecture rtl of square_root is
begin
    output(2) <= input(4);
    output(1) <= ((not input(4)) and input(2)) or ((not input(4)) and input(3));
    output(0) <= (input(3) and input(0)) or (input(3) and input(1)) or
                 ((input(3) and input(2))) or ((not input(4)) and
                 (not input(2)) and input(0)) or ((not input(4)) and
                 (not input(2)) and input(1));
end rtl;

architecture memory of square_root is

    type ram_type is array(31 downto 0) of std_logic_vector(2 downto 0);
    signal ram: ram_type := ("101", "101", "101", "101", "101", "101", "101",
                            "100", "100", "100", "100", "100", "100", "100",
                            "100", "100", "011", "011", "011", "011", "011",
                            "011", "011", "010", "010", "010", "010" , "010",
                            "001", "001", "001", "000");
begin

    output <= ram(to_integer(unsigned(input)));

end memory;
```

And below is the result of delay and area reports:
Memory based method:
+—Adders :
2 Input 32 Bit Adders := 1
+—Registers :
64 Bit Registers := 3
32 Bit Registers := 1
4 Bit Registers := 1
+—Muxes :
2 Input 64 Bit Muxes := 2
4 Input 36 Bit Muxes := 1
3 Input 3 Bit Muxes := 1
3 Input 2 Bit Muxes := 1
3 Input 1 Bit Muxes := 4
2 Input 1 Bit Muxes := 3
5 Input 1 Bit Muxes := 1
Module fsm
Detailed RTL Component Info :
+—XORs :
2 Input 1 Bit XORs := 1
Module fsm
Detailed RTL Component Info :
+—Muxes :
2 Input 4 Bit Muxes := 4
Module fsm
Detailed RTL Component Info :
+—Muxes :
2 Input 4 Bit Muxes := 2
2 Input 1 Bit Muxes := 1
The rtl based method:
+—Adders :
2 Input 32 Bit Adders := 2
+—Registers :
64 Bit Registers := 4
32 Bit Registers := 2
4 Bit Registers := 2
+—Muxes :
2 Input 64 Bit Muxes := 3
4 Input 36 Bit Muxes := 2
3 Input 3 Bit Muxes := 2
3 Input 2 Bit Muxes := 2
3 Input 1 Bit Muxes := 5
2 Input 1 Bit Muxes := 4
5 Input 1 Bit Muxes := 2

Module fsm
Detailed RTL Component Info :
+—XORs :
2 Input 1 Bit XORs := 2
Module fsm
Detailed RTL Component Info :
+—Muxes :
2 Input 4 Bit Muxes := 5
Module fsm
Detailed RTL Component Info :
+—Muxes :
2 Input 4 Bit Muxes := 3
2 Input 1 Bit Muxes := 2

3 Question 8

We should implement this circuit using the memory method:

State bit 2	State bit 1	State bit 0	Input bit 1	Input bit 0	Next State bit 2	Next State bit 1	Next State bit 0	Output
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0
0	0	0	1	0	1	0	0	0
0	0	0	1	1	0	0	0	0
0	0	1	0	0	0	0	1	1
0	0	1	0	1	0	1	1	1
0	0	1	1	0	1	0	1	1
0	0	1	1	1	0	0	1	1
0	1	0	0	0	0	0	0	0
0	1	0	0	1	0	1	0	0
0	1	0	1	0	1	0	0	0
0	1	0	1	1	0	0	0	0
0	1	1	0	0	0	0	1	1
0	1	1	0	1	0	1	1	1
0	1	1	1	0	0	0	0	0
0	1	1	1	1	0	1	0	0
1	0	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	1	1
1	0	0	1	1	0	1	1	1
1	0	1	0	0	1	0	1	1
1	0	1	0	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	0	1	1	1	0	1	0	0

As you know the default encoding method is sequential and also we have used sequential encoding explicitly in here.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity fsm is
    port(
            input: in std_logic_vector(1 downto 0);
            clk: in std_logic;
            output: out std_logic
        );
end entity;

architecture behavorial of fsm is
    signal state: std_logic_vector(2 downto 0) := "000";
    type ram_type is array(19 downto 0) of std_logic_vector(3 downto 0);
    signal ram: ram_type := ("0101", "1000", "1000", "1000", "1000", "1000",
                             "1000", "1000", "1000", "1000", "1000", "1000",
                             "1000", "1000", "1000", "1000", "1000", "1000",
                             "1000", "1000");
   signal address: std_logic_vector(4 downto 0);
begin
    address <= state & input;
    process(clk)
    begin
        if(rising_edge(clk)) then
            state <= ram(to_integer(unsigned(address)))(3 downto 1);
            output <= ram(to_integer(unsigned(address)))(0);
        end if;
    end process;
end behavorial;
```

And the testbench for this module is like below:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm_tb is
end entity;

architecture tb of fsm_tb is
    component fsm is
        port(
                input: in std_logic_vector(1 downto 0);
                clk: in std_logic;
                output: out std_logic
            );
    end component;

    signal input: std_logic_vector(1 downto 0);
    signal clk: std_logic;
    signal output: std_logic;
begin
    mapping: fsm port map(input, clk, output);

    input <= "00" after 10 ns,
             "10" after 20 ns,
             "11" after 30 ns;
    clk <= not clk after 5 ns;
end tb;
```

4 Question 9

The source code is like below:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity arbiter is
    port(
            data_bus0: in std_logic_vector(7 downto 0);
            request0: in std_logic;
            ack0: buffer std_logic;
            data_bus1: in std_logic_vector(7 downto 0);
            request1: in std_logic;
            ack1: buffer std_logic;
            data_bus2: in std_logic_vector(7 downto 0);
            request2: in std_logic;
            ack2: buffer std_logic;
            data_bus3: in std_logic_vector(7 downto 0);
            request3: in std_logic;
            ack3: buffer std_logic;
            output: out std_logic_vector(7 downto 0);
            clk: in std_logic
        );
end entity;

architecture rtl of arbiter is

begin
    process(clk)
    begin
        if(rising_edge(clk)) then
            if(request0 = '0' and request1 = '0' and request2 = '0' and request3 = '0') then
                ack0 <= '0';
                ack1 <= '0';
                ack2 <= '0';
                ack3 <= '0';
            elsif(request0 = '0' and request1 = '0' and request2 = '0' and request3 = '1') then
                ack0 <= '0';
                ack1 <= '0';
                ack2 <= '0';
                ack3 <= '1';
                output <= data_bus3;
            elsif(request0 = '0' and request1 = '0' and request2 = '1' and request3 = '0') then
                ack0 <= '0';
                ack1 <= '0';
                ack2 <= '1';
                output <= data_bus2;
                ack3 <= '0';
            elsif(request0 = '0' and request1 = '0' and request2 = '1' and request3 = '1') then
                ack0 <= '0';
                ack1 <= '0';
                if(ack2 = '1' and ack3 = '0') then
                    ack2 <= '0';
                elsif(ack2 = '0' and ack3 = '1') then
                    ack3 <= '0';
                else
                    output <= data_bus2;
                    ack2 <= '1';
                end if;
            elsif(request0 = '0' and request1 = '1' and request2 = '0' and request3 = '0') then
                ack0 <= '0';
                ack1 <= '1';
                output <= data_bus1;
                ack2 <= '0';
                ack3 <= '0';
            elsif(request0 = '0' and request1 = '1' and request2 = '0' and request3 = '1') then
                ack0 <= '0';
                ack2 <= '0';
                if(ack1 = '1' and ack3 = '0') then
                    ack1 <= '0';
                elsif(ack1 = '0' and ack3 = '1') then
                    ack3 <= '0';
                    output <= data_bus3;
                else
                    ack1 <= '1';
                end if;
            elsif(request0 = '0' and request1 = '1' and request2 = '1' and request3 = '0') then
                ack0 <= '0';
                ack3 <= '0';
                if(ack1 = '1' and ack2 = '0') then
                    ack1 <= '0';
                elsif(ack1 = '0' and ack2 = '1') then
                    ack2 <= '0';
                else
                    ack1 <= '1';
                    output <= data_bus1;
                end if;
            elsif(request0 = '0' and request1 = '1' and request2 = '1' and request3 = '1') then
                ack0 <= '0';
                ack3 <= '0';
                if(ack1 = '1' and ack2 = '0') then
                    ack1 <= '0';
                elsif(ack1 = '0' and ack2 = '1') then
                    ack2 <= '0';
                else
                    ack1 <= '1';
                    output <= data_bus1;
                end if;
            elsif(request0 = '1' and request1 = '0' and request2 = '0' and request3 = '0') then
```

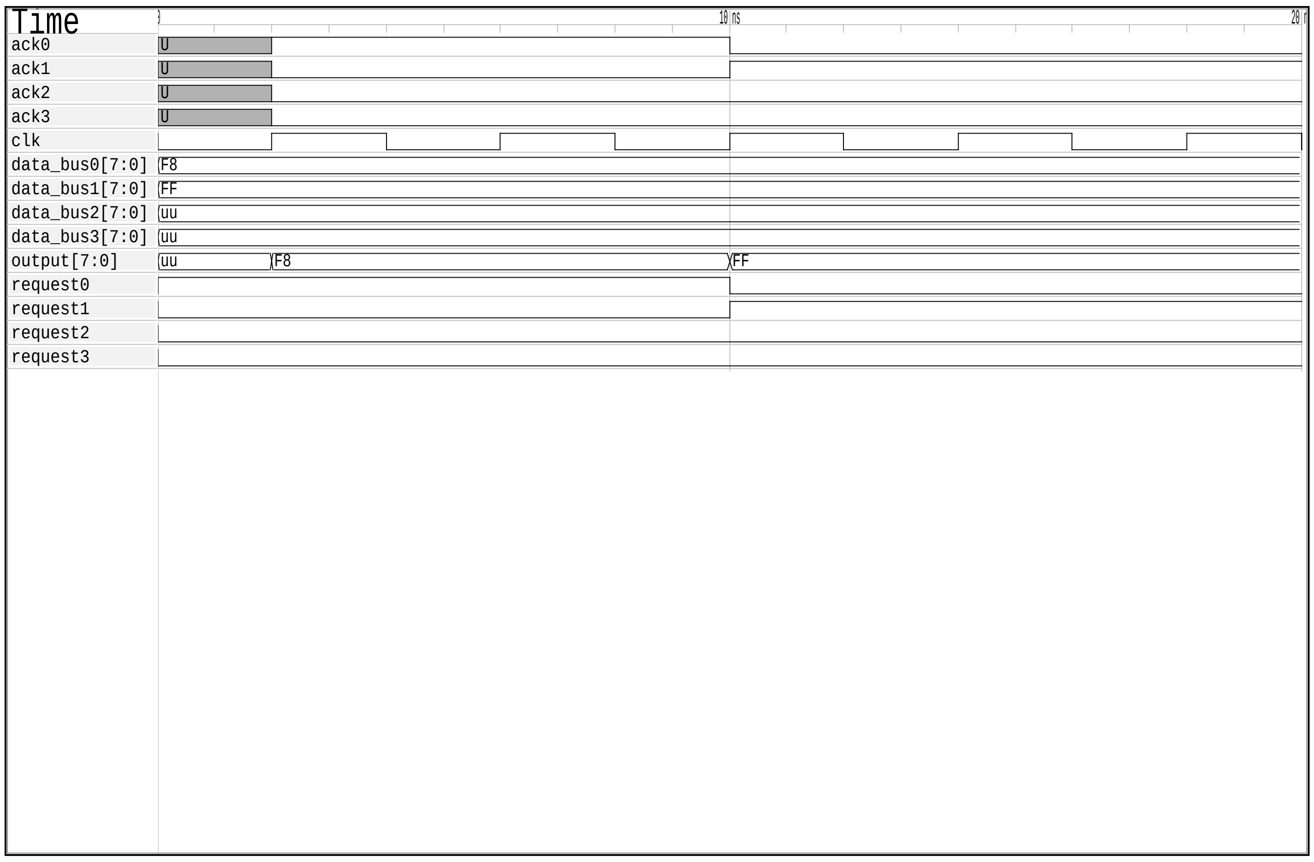
```
    ack0 <= '1';
    output <= data_bus0;
    ack1 <= '0';
    ack2 <= '0';
    ack3 <= '0';
elsif(request0 = '1' and request1 = '0' and request2 = '0' and request3 = '1') then
    ack0 <= '0';
    ack3 <= '0';
    if(ack1 = '1' and ack2 = '0') then
        ack1 <= '0';
    elsif(ack1 = '0' and ack2 = '1') then
        ack2 <= '0';
    else
        ack1 <= '1';
        output <= data_bus1;
    end if;
elsif(request0 = '1' and request1 = '0' and request2 = '1' and request3 = '0') then
    ack0 <= '0';
    ack3 <= '0';
    if(ack1 = '1' and ack2 = '0') then
        ack1 <= '0';
    elsif(ack1 = '0' and ack2 = '1') then
        ack2 <= '0';
    else
        ack1 <= '1';
        output <= data_bus1;
    end if;
elsif(request0 = '1' and request1 = '0' and request2 = '1' and request3 = '1') then
    ack0 <= '0';
    ack3 <= '0';
    if(ack1 = '1' and ack2 = '0') then
        ack1 <= '0';
    elsif(ack1 = '0' and ack2 = '1') then
        ack2 <= '0';
    else
        ack1 <= '1';
        output <= data_bus1;
    end if;
elsif(request0 = '1' and request1 = '1' and request2 = '0' and request3 = '0') then
    ack0 <= '0';
    ack3 <= '0';
    if(ack1 = '1' and ack2 = '0') then
        ack1 <= '0';
    elsif(ack1 = '0' and ack2 = '1') then
        ack2 <= '0';
    else
        ack1 <= '1';
        output <= data_bus1;
    end if;
elsif(request0 = '1' and request1 = '1' and request2 = '0' and request3 = '1') then
    ack0 <= '0';
    ack3 <= '0';
```

```
                if(ack1 = '1' and ack2 = '0') then
                    ack1 <= '0';
                elsif(ack1 = '0' and ack2 = '1') then
                    ack2 <= '0';
                else
                    ack1 <= '1';
                    output <= data_bus1;
                end if;
            elsif(request0 = '1' and request1 = '1' and request2 = '1' and request3 = '0') then
                ack0 <= '0';
                ack3 <= '0';
                if(ack1 = '1' and ack2 = '0') then
                    ack1 <= '0';
                elsif(ack1 = '0' and ack2 = '1') then
                    ack2 <= '0';
                else
                    ack1 <= '1';
                    output <= data_bus1;
                end if;
            elsif(request0 = '1' and request1 = '1' and request2 = '1' and request3 = '1') then
                ack0 <= '0';
                ack3 <= '0';
                if(ack1 = '1' and ack2 = '0') then
                    ack1 <= '0';
                elsif(ack1 = '0' and ack2 = '1') then
                    ack2 <= '0';
                else
                    ack1 <= '1';
                    output <= data_bus1;
                end if;
            end if;
        end if;
    end process;
end rtl;
```

Here is the testbench:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity arbiter_tb is
end entity;

architecture tb of arbiter_tb is
    component arbiter is
        port(
                data_bus0: in std_logic_vector(7 downto 0);
                request0: in std_logic;
                ack0: buffer std_logic;
                data_bus1: in std_logic_vector(7 downto 0);
                request1: in std_logic;
                ack1: buffer std_logic;
                data_bus2: in std_logic_vector(7 downto 0);
                request2: in std_logic;
                ack2: buffer std_logic;
                data_bus3: in std_logic_vector(7 downto 0);
                request3: in std_logic;
                ack3: buffer std_logic;
                output: out std_logic_vector(7 downto 0);
                clk: in std_logic
            );
    end component;

    signal data_bus0: std_logic_vector(7 downto 0);
    signal request0: std_logic := '0';
    signal ack0: std_logic := '0';
    signal data_bus1: std_logic_vector(7 downto 0);
    signal request1: std_logic := '0';
    signal ack1: std_logic := '0';
    signal data_bus2: std_logic_vector(7 downto 0);
    signal request2: std_logic := '0';
    signal ack2: std_logic := '0';
    signal data_bus3: std_logic_vector(7 downto 0);
    signal request3: std_logic := '0';
    signal ack3: std_logic := '0';
    signal output: std_logic_vector(7 downto 0);
    signal clk: std_logic := '0';

begin
    mapping: arbiter port map(data_bus0, request0, ack0, data_bus1, request1, ack1, data_bus2, request2, ac
    clk <= not clk after 2 ns;
    data_bus0 <= "11111000";
    request0 <= '1' after 0 ns,
                '0' after 10 ns;
    data_bus1 <= "11111111";
    request1 <= '0' after 0 ns,
                '1' after 10 ns;

end tb;
```

And here is the simulation result:


Time
0
10 ns
20 n
ack0
ack1
ack2
ack3
clk
data_bus0[7:0]
data_bus1[7:0]
data_bus2[7:0]
data_bus3[7:0]
output[7:0]
request0
request1
request2
request3
U
F8
FF
uu
F8
FF