University of the West of England
BRISTOL

NORTHSHORE
COLLEGE OF BUSINESS AND TECHNOLOGY

CPU Design

Supervised by
Mr. Sandun Rajapakshe

Module Code: UFMFE8-30-2

Submitted to the Faculty of Environment and Technology, University of the West of England, UK in partial fulfillment of the requirements for the MEng (Hons) degree in Electrical and Electronic Engineering

16TH AUGUST 2018

Abstract— This coursework's objective is to determine the design of the VS CPU, ROM and RAM. The assignment consists of four main sections; Control Unit (CU), Register section, Arithmetic and Logic Unit (ALU), Memory unit (RAM & ROM). After completion of these four parts the design was implemented using VHDL codes. In this course work, there were four different options to design and we decided on following the first option of designing the VS-CPU using hardwired control or VHDL state machine control unit to include registers and ALU.

I. INTRODUCTION

This is begun with examining the design content of a CPU and identifying how the registers are organized within and how data is routed to and from registers. Mainly how these registers are controlled by the CPU. Next, looking at the ALU process how the data is received and stored to registers, then how the internal process and design is happened can be learnt. 8-bit wide instructions are used with 3-bit wide opcode [7 to 5] and 5-bit wide address [4 to 0].

II. CPU

CPU is a sequential circuit which repeatedly reads and executes instructions from its memory. Each instruction has two parts; the opcode and the address. CPU is a finite state machine with micro operations of,

- fetch - Fetch the instruction from memory, then go to the next cycle decode
- Decode – Decode that instruction which has been fetched and then go to execute cycle
- Execute – Execute the instructions and again go to fetch cycle and fetch the next instruction

Decode operation has multiple ways from the end of the fetch which leads to each individual execute routine. Diagram below is the structure of a very simple CPU,

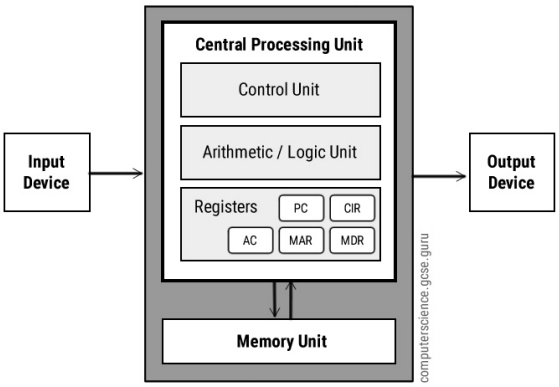


Fig 1. Architecture design of a basic CPU

The CPU has three main parts,

A. Control Unit

This controls the action of the other computer parts so that instructions are executed in the correct sequence, in the means of operating internal and external signals in sequence to process instruction / data, inputs the op-code and flags.

B. Arihmetic and Logic Unit (ALU)

This does the arithmetic operations and logical operations on the register section and combinational logic operation. In this course work the main operations are ADD and SUB. First receives the operand from register section then do the operation and again store the result in register section

C. Registers

This is the temporary storage part inside the CPU, it can be read and written at high speed and register hold a computer instruction, a storage address, or any kind of data. These are connected via busses.

Table below shows the instructions used for this design implementation,

INSTRUCTION	REMARKS
ADD	Add the number in register A to a number stored in a given memory location and store the result back in A.
SUB	Subtract a number stored in a given memory location from the number in register A and store the result back in A.
STA	Store the value in A to a given memory location
LDA	Read the value from a given memory location to Register A
JMP	Jump to the instruction at the address given
CMP	Compare the number stored in the given memory location with the number in register A and; i. Skip 1 instruction if A<MEM(address) ii. Skip 2 instructions if A>MEM(address)
DIO	Read data from the input_port to A if IR[3]=0 Write data from A to output_port if IR[3]=1
HLT	Halt the program execution

Fig 2. Instructions table

III. REGISTERS

A register is a small place where a set of data is held. This may hold a computer instruction, a storage address, or any kind of data such as a bit sequence or a character. Depending on the processor design registers may be numbered or named differently. However, this course work's design will be using an 8-bit data bus, ALU and flag registers.

A. Registers

Some of the registers are used to store data while some are used for a special task. Such as the program counter (PC).

1) Address Register(AR): this is used for storing the next address of instruction to be executed. Which meeans, AR supplies an address to the memory subsystem RAM via a 8-bit data bus by using the 5-bits [4 to 0].

2) Program counter(PC): This contains the address of the next instruction to be executed.

3) Instruction Register(IC): This stores the opcode.

4) Incrimenter (INC): This increments the instruction accordingly either by 1 or 2. It adds to the input from the PC.

5) Register A, B: This are general purpose registers that provides the facility of storing data.

B. Data bus

Buses are described as a set of signals. Even though control signals and ports normally are unidirectional, buses are bi-directional and it will need to be described with a function

to resolve the bus access. Which means, when a signal is driven by more than one source there must be a way of resolving what to happen if multiple drivers are active at the same time. In this VS-CPU design tri-state buffers (using enable pin) have been used to control the data bus.

C. ALU

Arithmetic and logic unit is a digital circuit used to perform arithmetic and logic operations. This represents the major structural block of the CPU and nowadays modern CPU's use very powerful and complex ALUs.

The control unit tells the ALU what operation to perform on the data loaded and then it is stored in a register. As mentioned before, this can perform arithmetic operations such as addition and subtraction with logic operations of AND, OR, XOR and NOT. In this design, we will be using addition and subtraction as arithmetic operations and comparator as a logic operation.

IV. CONTROL UNIT

Control unit generates the signals to cause the operations to happen in proper desired sequence. A control unit of a very simple CPU has three main units,

- counter – containing the current state signal
- decoder – generate new signals individually for each state using the current state
- logic – take state signals and generate control signals for each and controls the counter

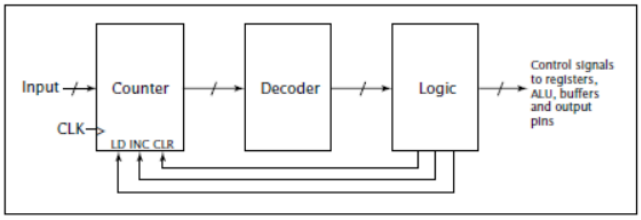


Fig 3. VS-CPU control unit

As mentioned before CPU's one of the main operation is fetching. Once the instruction is fetched it is executed by assigning states to the output of the decoder according to the current signal the counter has.

The diagram below is on the states of VS-CPU describing the outputs according to the inputs.

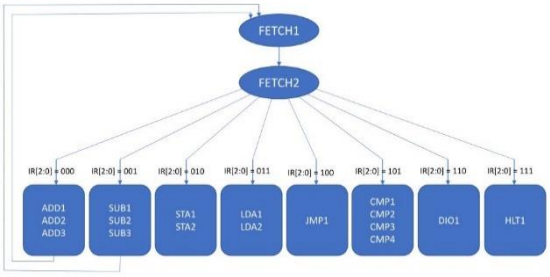


Fig 4. State diagram of VS-CPU

V. MEMORY

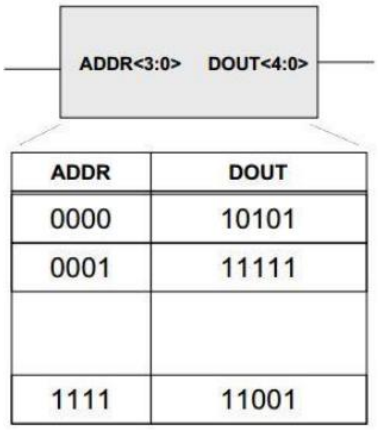
Memory is the part that holds data and instructions for processing. The control unit will be sending data and instructions from memory to ALU so that an arithmetic or/and logic operation can be achieved. After the process, also information is being sent to memory to hold till it is ready to output.

Program instructions must be positioned into memory from an input device or storage device before an instruction to be executed (data will make a temporary stop in a register). Once the necessary data and instructions are in the memory,

- Control unit fetches the instruction from memory
- It decodes the instruction (decides what should be done) and directs the necessary data to be moved from memory to the ALU
- ALU performs the actual operation on data
- Result will be stored in a memory or register by the ALU

A. Read-only-memory (ROM)

Memory structure is a combination of the data bus and the address bus while, the output is controlled by control signals. For memories, a lookup table is used such that each address corresponds to a data output.



ADDR	DOUT
0000	10101
0001	11111
1111	11001

Fig 5. Structure of addressing modes

In the case of memory, it should be able to read and write data. But, ROM have to consider the values in the lookup table. Within a process every logic has to follow different steps to have fulfill the desired output. These instructions which come in middle will be stored in the ROM but the final output in the RAM. There are two main types if ROM

a) PROM

uses bipolar transistors
one time programmable
cheap
rarely FPGA used

b) EEPROM / EPROM

Erasable
Programmable
ROM
Embedded OS

B. Random Access Memory (RAM)

Purpose is to provide quick read and write access to a storage device. The data we actively using is temporarily saved in RAM. In this design, from the address register the address will be sent to the RAM to get the stored data from

the mentioned address. Typically, this has two pins to either read or write.

VI. Implementation of the Very Simple CPU

The 8-bit CPU with eight instructions was designed with ModelSim-Altera software. This was a challenging task and by implementing this, we gained a lot of knowledge in VHDL. The VHDL codes and the schematic are as follows:

A. *Incrimenter(INC)*

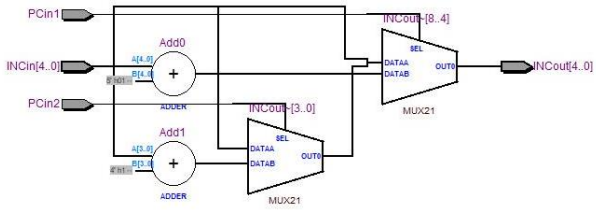


Figure 6: INC schematic

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity INC is
6    port(
7      INCin            : in signed(4 downto 0);
8      PCin1, PCin2     : in std_logic;
9      INCout           : out signed(4 downto 0)
10   );
11 end entity INC;
12
13 architecture behave of INC is
14
15 begin
16
17   process(INCin,PCin1,PCin2)
18   begin
19
20     if(PCin1='1') then
21       INCout <= INCin + 1;
22
23     elsif(PCin2='1') then
24       INCout <= INCin + 2;
25
26     else
27       INCout <= INCin;
28
29     end if;
30   end process;
31 end architecture behave;
```

Figure 7: VHDL code for INC

B. *Program Counter(PC)*

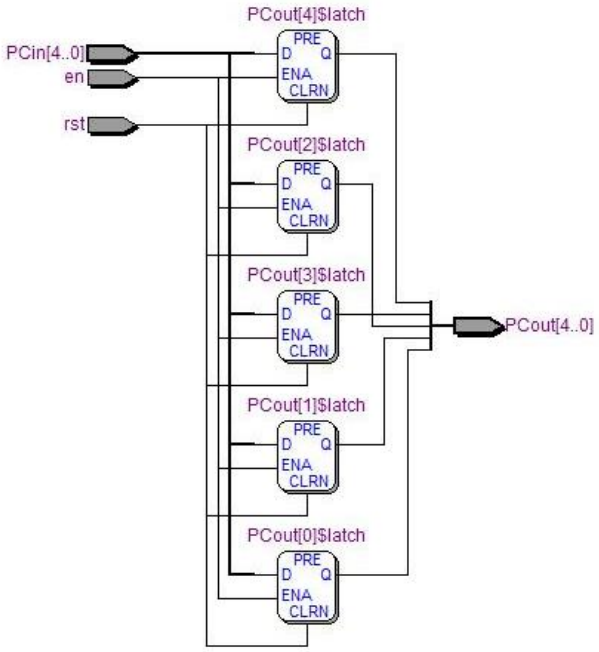


Figure 8: PC schematic

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity PC is
5    port(
6      PCin       : in std_logic_vector(4 downto 0);
7      en         : in std_logic;
8      rst        : in std_logic;
9      PCout      : out std_logic_vector(4 downto 0)
10   );
11 end entity PC;
12
13 architecture behave of PC is
14
15 begin
16   process(en,rst)
17   begin
18
19     if(rst='1') then
20       PCout <= (others => '0');
21
22     elsif(en='1') then
23       PCout <= PCin;
24
25     end if;
26   end process;
27 end architecture behave;
```

Figure 9: VHDL code for PC

C. Memory Address Register(MAR)

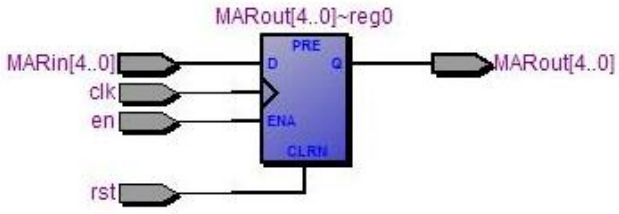


Figure 10: MAR schematic

```
library ieee;
use ieee.std_logic_1164.all;

entity MAR is
  generic (
    N: integer := 5
    );
  port (
    MARin              :in  std_logic_vector(N-1 downto 0);
    clk,rst,en         :in  std_logic;
    MARout             :out std_logic_vector(N-1 downto 0)
  );
end entity MAR;

architecture behave of MAR is

begin
  process(clk,rst) begin

    if(rst='1') then
      Marout <= (others => '0');

    elsif(rising_edge(clk)) then

      if(en='1') then
        MARout <= MARin;

      end if;
    end if;
  end process;
end architecture behave;
```

Figure 11: VHDL code for MAR

D. Random Access Memory(RAM)

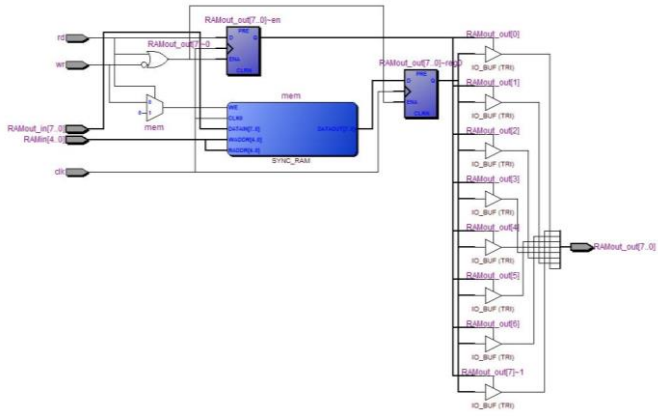


Figure 12: RAM schematic

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

entity RAM is
  generic (
    A  : integer := 5;
    N  : integer := 8
  );
  port(
    RAMin          : in std_logic_vector(A-1 downto 0);
    clk,rd,wr      : in std_logic;
    RAMout_in      : in std_logic_vector(N-1 downto 0);
    RAMout_out     : out std_logic_vector(N-1 downto 0)
  );
end entity RAM;

architecture behave of RAM is
  type temp_memory is array (0 to (2**A)-1) of std_logic_vector(N-1 downto 0);
  signal mem  : temp_memory  :=(
```

Figure 13: VHDL code for RAM - part 1

```
    0   => "11000110",
    1   => "01011111",
    2   => "11001111",
    3   => "00111111",
    4   => "01001100",
    5   => "11100000",
    6   => "00000000",
    7   => "00000000",
    8   => "00000000",
    9   => "00000000",
    10  => "00000000",
    11  => "00000000",
    12  => "00000000",
    13  => "00000000",
    14  => "00000000",
    15  => "00000000",
    16  => "00000000",
    17  => "00000000",
    18  => "00000000",
    19  => "00000000",
    20  => "00000000",
    21  => "00000000",
    22  => "00000000",
    23  => "00000000",
    24  => "00000000",
    25  => "00000000",
    26  => "00000000",
    27  => "00000000",
    28  => "00000000",
    29  => "00000000",
    30  => "00000000",
    31  => "00000000"
```

Figure 14: VHDL code for RAM - part 2

```
begin

process(clk)
 begin
   if(rising_edge(clk)) then

     if(rd='1') then
       RAMout_out <= mem(to_integer(unsigned(RAMin)));

     elsif(wr='1') then
       mem(to_integer(unsigned(RAMin))) <= RAMout_in;

     else
       RAMout_out <= (others => 'Z');

     end if;
   end if;
end process;

end architecture behave;
```

Figure 15: VHDL code for RAM - part 3

E. Instruction Register(IR)

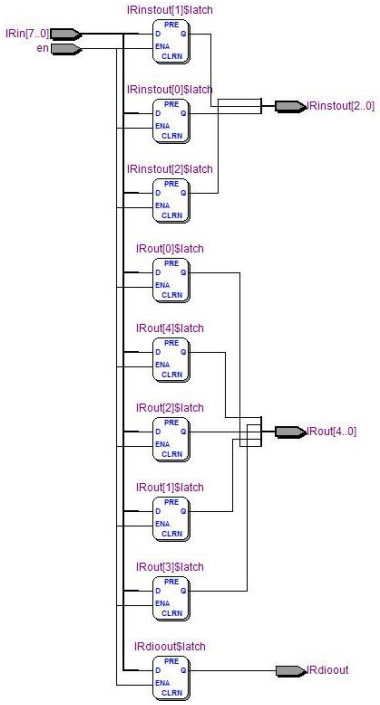


Figure 16: IR schematic

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity IR is
  generic (
    N: integer := 8;
    A: integer := 5
    );
  port (
    IRin        :in  std_logic_vector(N-1 downto 0);
    en          :in  std_logic;
    IRout       :out std_logic_vector(A-1 downto 0);
    IRinstout   :out std_logic_vector(2 downto 0);
    IRdioout    :out std_logic
  );
end entity IR;

architecture behave of IR is

begin
  process(IRin,en) begin

    if(en='1') then
        IRout <= IRin(A-1 downto 0);
        IRinstout <= IRin(N-1 downto A);
        IRdioout <= IRin(A-1);

    end if;
  end process;
end architecture behave;
```

Figure 17: VHDL code for IR

F. Register A

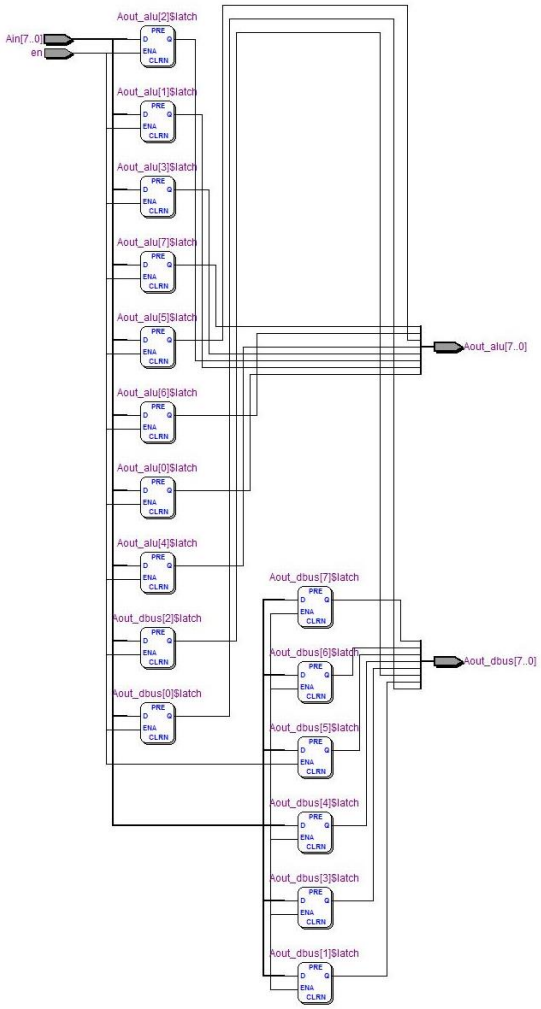


Figure 18: Register A schematic

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity A is
  generic (
    N: integer := 8
    );
  port (
    Ain             :in  std_logic_vector(N-1 downto 0);
    en              :in  std_logic;
    Aout_alu        :out std_logic_vector(N-1 downto 0);
    Aout_dbus       :out std_logic_vector(N-1 downto 0)
  );
end entity A;

architecture behave of A is

begin
  process(Ain,en) begin

    if(en='1') then

      Aout_alu <= Ain;
      Aout_dbus <= Ain;

    end if;
  end process;
end architecture behave;
```

Figure 19: VHDL code for Register A

G. Register B

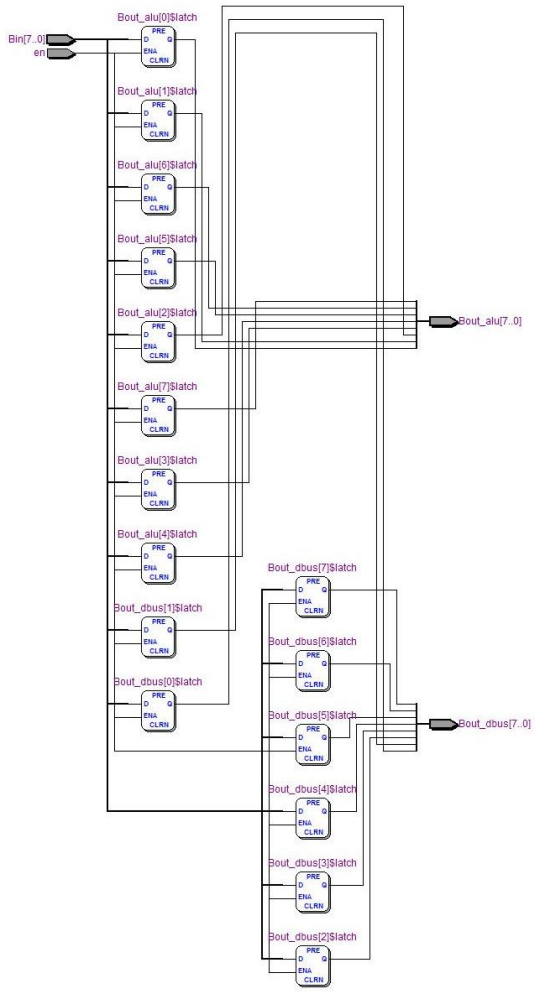


Figure 20: Register B schematic

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity B is
  generic (
    N: integer := 8
    );
  port (
    Bin             :in  std_logic_vector(N-1 downto 0);
    en              :in  std_logic;
    Bout_alu        :out std_logic_vector(N-1 downto 0);
    Bout_dbus       :out std_logic_vector(N-1 downto 0)
  );
end entity B;

architecture behave of B is

begin
  process(Bin,en) begin

    if(en='1') then

      Bout_alu <= Bin;
      Bout_dbus <= Bin;

    end if;
  end process;
end architecture behave;
```

Figure 21: VHDL code for Register B

H. Arithmetic and Logic Unit(ALU)

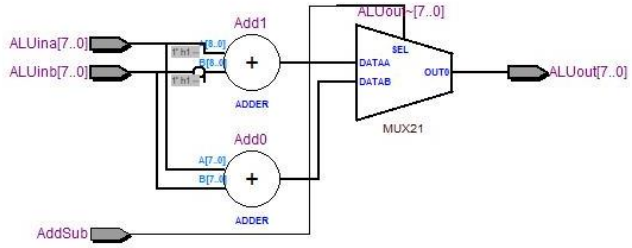


Figure 22: ALU schematic

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ALU is
  generic (
    N : integer := 8
  );
  port(
    ALUina, ALUinb    : in signed(N-1 downto 0);
    AddSub            : in std_logic;
    ALUout            : out signed(N-1 downto 0)
  );
end entity ALU;

architecture behave of ALU is

begin

  process(AddSub)
  begin

    case AddSub is

      when '1' =>
        ALUout <= ALUina + ALUinb;

      when '0' =>
        ALUout <= ALUina - ALUinb;

      when others =>
        null;

    end case;
  end process;
end architecture behave;
```

Figure 23: VHDL code for ALU

VII. Problems and Solutions

A. Latchings

When a bus is connected to many ports, there can be latches. When giving an input to this bus, it will not always accept the exact input. This may be due to other active ports connected to the bus. The solution is to first deactivate the other ports connected to the bus and activate only the required port.

B. Clock Process

Not every components used are sequential. So some does not depend on clock. Some of these depended and independent components need to work at the same state, so it is not possible to drive them at the same time. The solution is to make a new state and separate these component instructions while keeping the previous conditions unchanged.

C. Useless buffer type ports

In RAM, it should be able to read from the data bus and write to data bus, so there has to be a buffer port, which can act as an input and output. Sometimes these buffer ports will not work at some states; this can be due to Latching. To prevent such case we can introduce 2 new ports, an Input and an Output in replacement of the buffer port. These two ports can be connected to the data bus through signals. Latch cannot happen as two of these ports are not connected together internaly.

Acknowledgment

We wish to express our sincere gratitude to our module leader Mr. Sandun Rajapakshe for providing us vital support and guidance to complete this coursework. We would like to thank our program leaders, seniors and friends who assisted us in completing this coursework successfully with their support and willingness to spend some time for us.

References

[1] J. D. Carpinelli, "CPU design- chapter 6," [Online].

[2] "The Central Processing Unit," [Online]. Available: http://www.electronics.dit.ie/staff/tscarff/DT089_Physical_Computing_1/central_.

[3] "How Computers Work: The CPU and Memory," [Online]. Available: https://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading04.htm.

[4] "Random Access Memory (RAM)," [Online]. Available: https://searchstorage.techtarget.com/definition/RAM-random-access-memory.