Chapter 03: RTL and HDL

Hari K.C.

Department of Software Engineering

Gandaki College of Engineering and Science

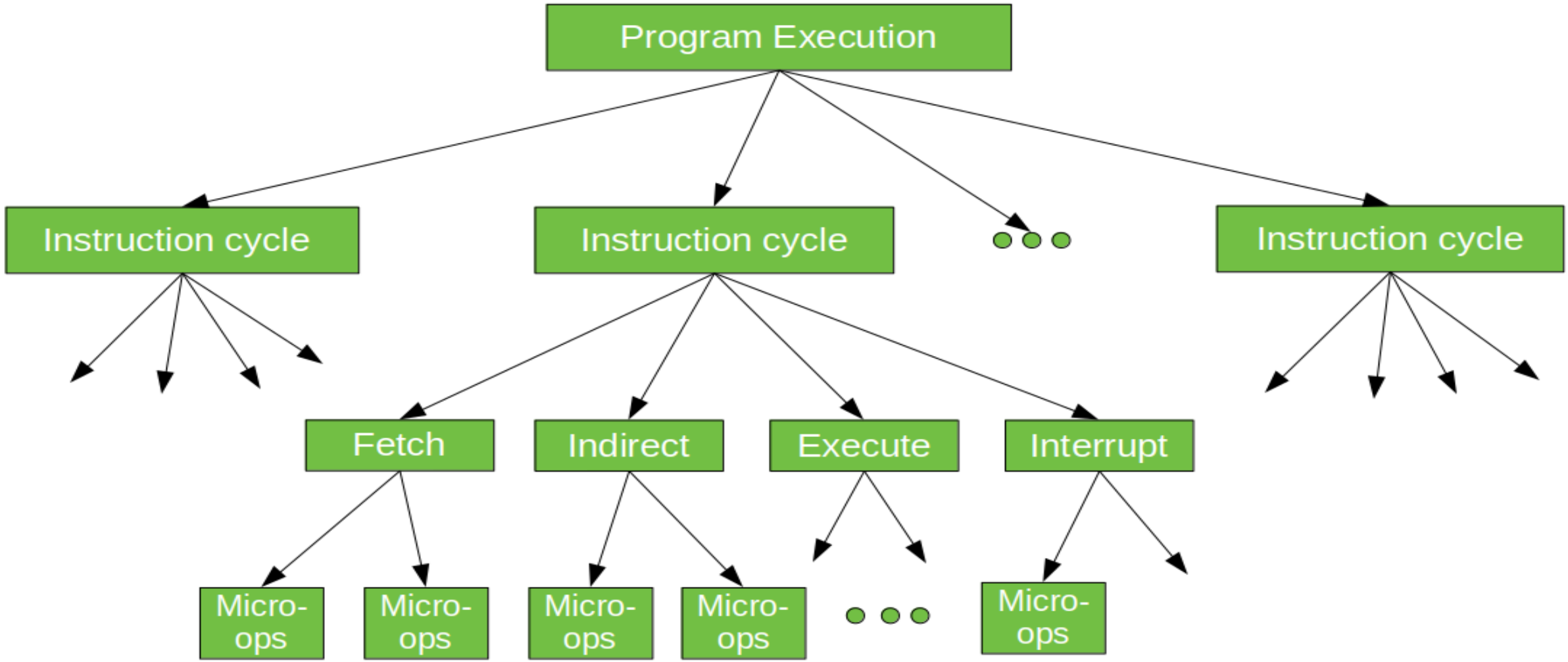
Microoperations

- The operations to be performed at one unit time(One T state) is microoperation.
- It is a set of 1 or 2 operations that can be accomplished with a single clock pulse.

Register Transfer Language:

It is a symbolic notation for microoperation.

It is a HDL(hardware description language) in which the microoperation define literally is expressed in a language that can be used to transfer the data among the registers is called RTL.


Program Execution
Instruction cycle
Instruction cycle
Instruction cycle
Fetch
Indirect
Execute
Interrupt
Micro-ops
Micro-ops
Micro-ops
Micro-ops
Micro-ops

HDL:

It is used to specify the conditions and transfer of the operation that corresponds to the system configuration.

It is used to specify logic of the system.

VHDL :

VHDL can specify both the logic and implementation of the system.

VHDL : VHSIC hardware description language

(Very high speed integrated circuit)

```
USE TEXTIO.all, mypackage.all;
-----------------------------------
ENTITY module IS
   PORT (X, Y: IN BIT; Z: out BIT_VECTOR(3 DOWNTO
   0);
END module;
-----------------------------------
ARCHITECTURE behavior OF module IS
   SIGNAL A, B: BIT_VECTOR(3 DOWNTO 0);
BEGIN
   A(0) <= X AFTER 20 ns; A(1) <= Y AFTER 40 ns;
   PROCESS (A)
       VARIABLE P, Q: BIT_VECTOR(3 DOWNTO 0);
   BEGIN
       P := fft(A);
       B <= P AFTER 10 ns;
   END PROCESS;
   Z <= B;
END behavior;
```

Data transfer through microoperation

- Let us consider the digital system with two 1 bit registers X and Y.
- X← Y , copying the contents of register Y to X.

- Microoperation just specify the transfer to be made not on how to copy data.
- There are two types of Connection:

a) Direct Connection

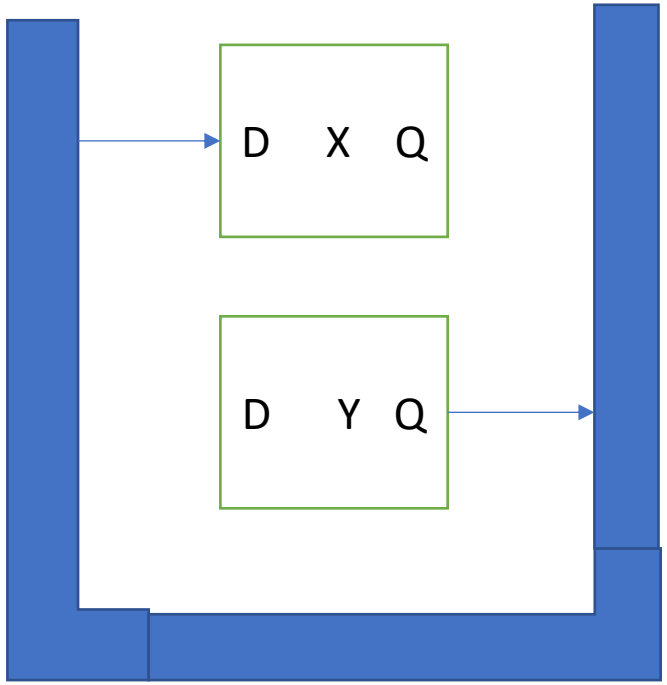
b) Creating a Bus

Direct Connection

- The set of microoperation is sufficient for designing its data path, the connection between components used for data transfer.
- Direct connection is also called as Direct path.

Creating a Bus

- Using a Bus , data are transferred from one register to other registers.

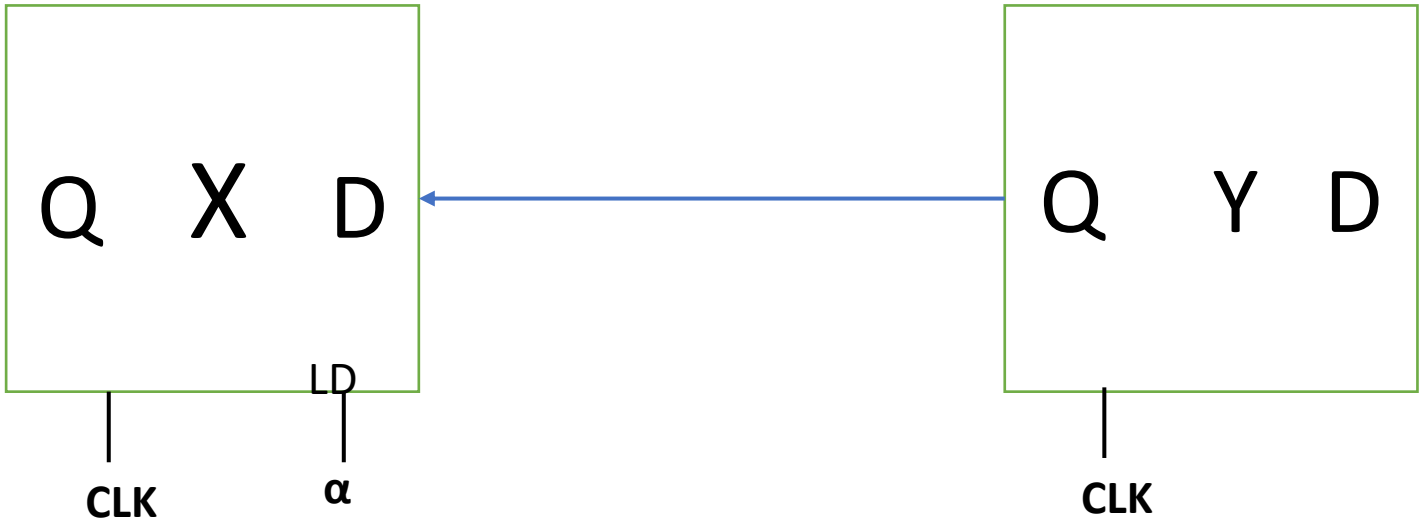


- RTL uses a compact notation

Condition : Microoperation

so, the transfer can be written as:-

α : X ← Y



Use direct connection to show the data transfer:

1) **α : X ← Y ← Z**
2) **α : X← Y , Z ←Y**

Arithmetic and logical microoperations

- ADDITION : X← X+ Y
- SUBTRACTION : X ← X – Y OR X ← X + Y’ + 1
- INCREMENT: X ←X +1
- DECREMENT : X ← X -1
- AND’ING : X ← X ^ Y
- OR’ING : X ← X $\vee Y$
- XOR : X ← X $\oplus$ Y
- NOT : X ← X’

SHIFT MICROOPERATION

4 TYPES OF SHIFT

1) LINEAR SHIFT
2) CIRCULAR SHIFT
3) ARITHMETIC SHIFT
4) DECIMAL SHIFT

LINEAR SHIFT

- Linear shift left:

if X = 1011, then performing linear shift left.

shl(X) = 0110

- Linear shift right:

if X = 1011, then performing linear shift right

shr(X) = 0101

Circular shift

Circular shift left:

if X = 1011

then, Cil(X) = 0111

Circular shift right

if X = 1011

then, Cir(X) = 1101

Arithmetic shift

- **Arithmetic shift left**

if X = 1011

then

ashl(X) = 1110

Arithmetic shift right

if X= 1011

then ashr(X) = 1101

Decimal shift

- It act like linear shift but it works for block of bits rather than single bits.

Decimal shift left:

if X = 1001 0111

then dshl(X) = 0111 0000

Decimal shift right:

if X = 1001 0111

then dshr(X) = 0000 1001

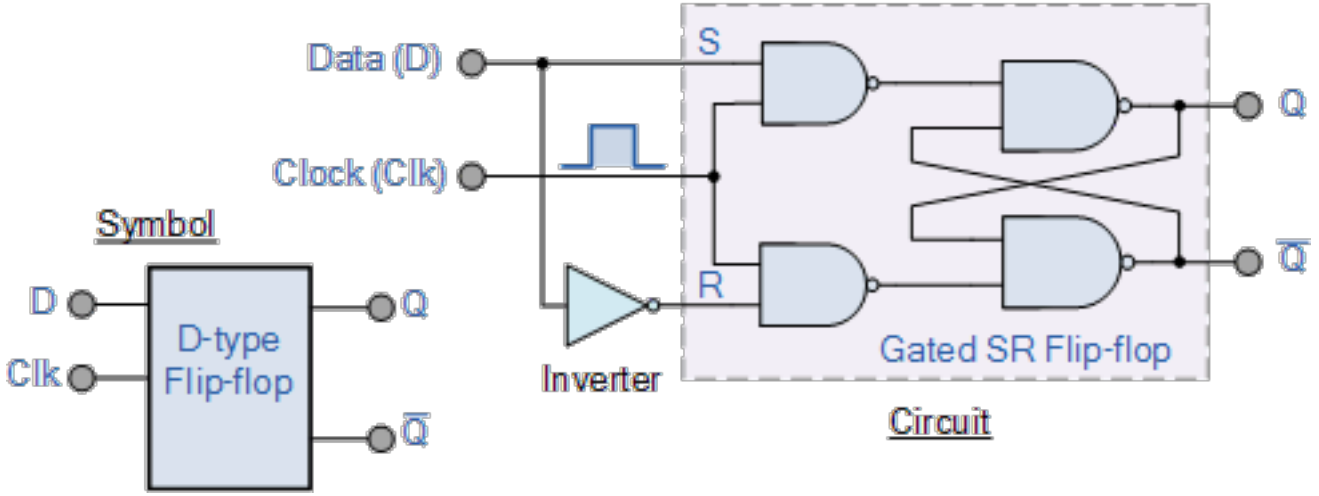
RTL for Digital system

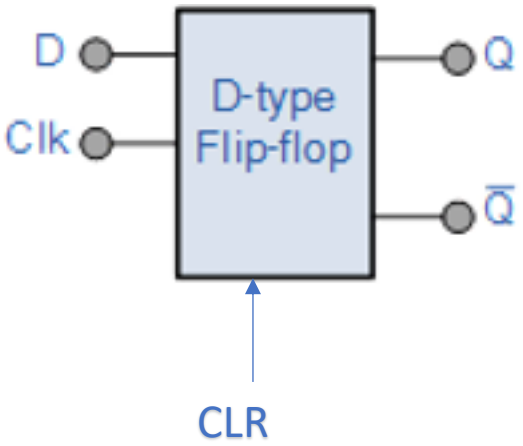
- RTL can be used to specify the behavior of the sequential digital system.

Example 1: Let us consider a flipflop. Its function is given by single RTL statement.

LD : Q←D

When LD input is high, the value on D input is loaded and made available on Q output. This happens on rising edge of clock pulse.





When CLR = 1, the flipflop is set to 0 (zero).

LD :Q← D
CLR: Q←0

When D, LD and CLR are equal to 1, the above system fails.

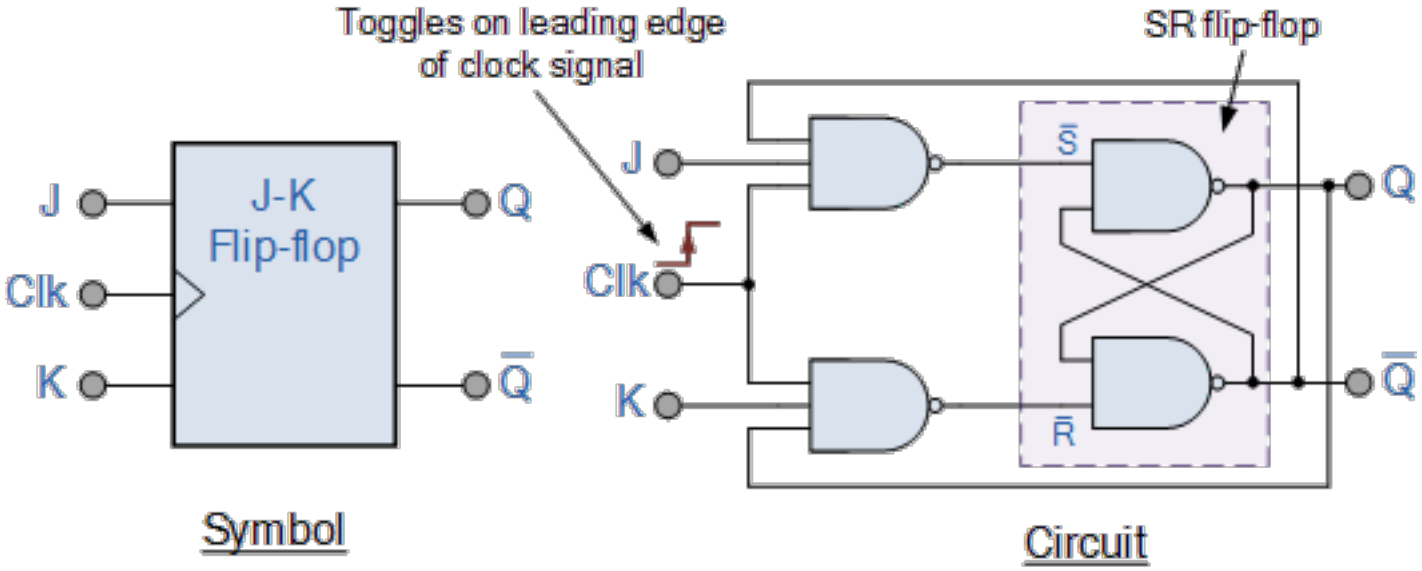
The RTL code is:

1) CLR' LD : Q ← D
2) LD: Q ← D
3) CLR : Q ←0
4) LD' CLR : Q←0

RTL code for JK flipflop

Truth Table

J	K	CLK	Q
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q}_0$ (toggles)



The RTL code for jk flipflop with clock pulse (CLK) high

1) J' K : Q ← 0
2) J K' : Q←1
3) J K: Q ←Q'

When J=K=0, no condition is met and flipflop retains its previous value. so, no RTL statement is needed.

Specification and implementation of digital systems

- Consider the system with four, 1 bit flipflops.
- Given the RTL statements

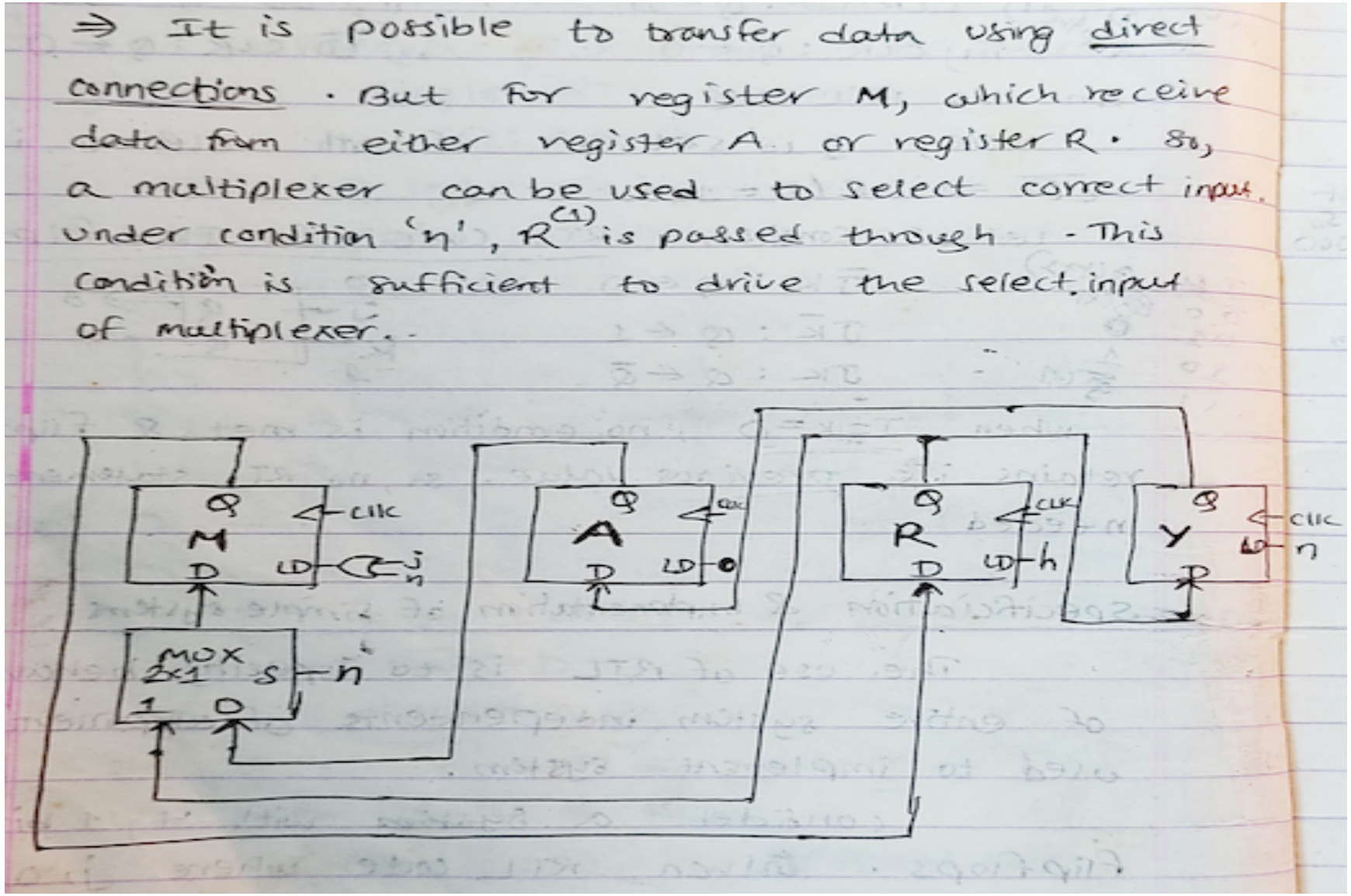
j : M←A

o: A←Y

h: R←M

n: Y←R , M←R

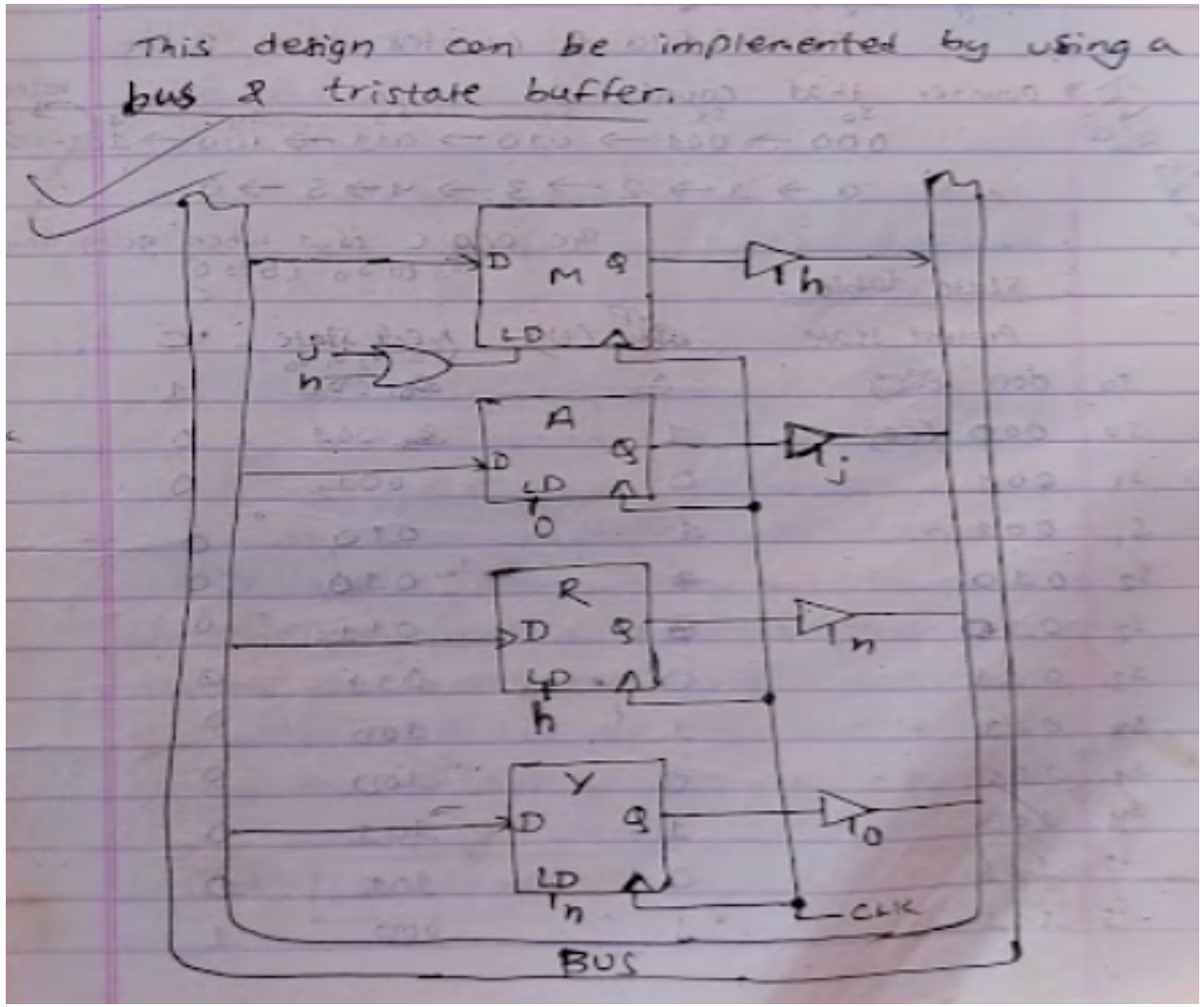
Direct connection for the given RTL



4 flipflops
1 multiplexer(2 x 1)

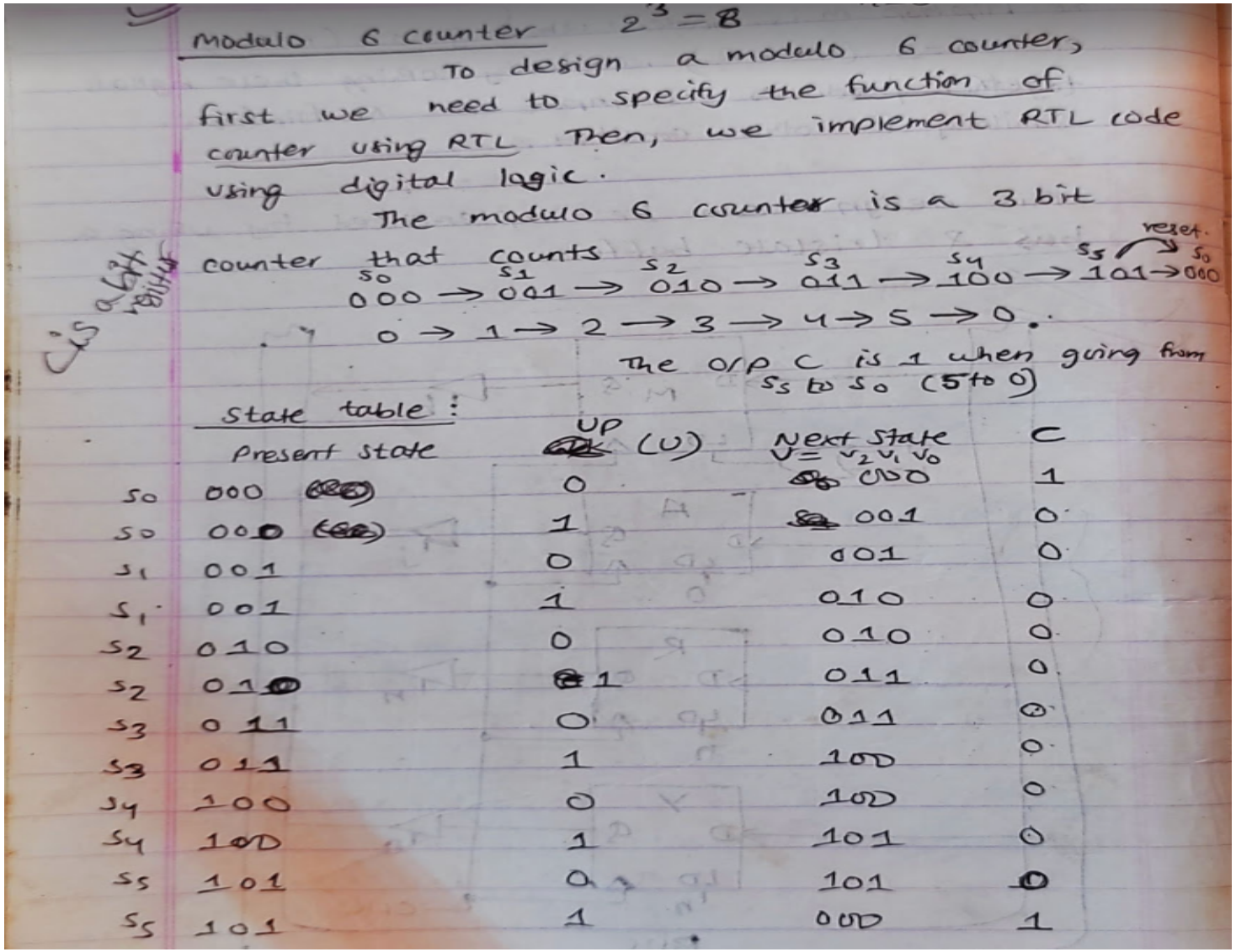
The flipflop is loaded when j =1 or n= 1

Bus connection for given RTL



j : M←A
o: A←Y
h: R←M
n: Y←R , M←R

Modulo – 6 counter



Modulo 6 counter $2^3 = 8$

To design a modulo 6 counter, first we need to specify the function of counter using RTL. Then, we implement RTL code using digital logic.

The modulo 6 counter is a 3 bit counter that counts

S_0 000 → S_1 001 → S_2 010 → S_3 011 → S_4 100 → S_5 101 → reset S_0 000

0 → 1 → 2 → 3 → 4 → 5 → 0.

The o/p C is 1 when going from S_5 to S_0 (5 to 0)

State table :

	Present state	UP (U)	Next state $V = V_2 V_1 V_0$	C
S_0	000	0	000	1
S_0	000	1	001	0
S_1	001	0	001	0
S_1	001	1	010	0
S_2	010	0	010	0
S_2	010	1	011	0
S_3	011	0	011	0
S_3	011	1	100	0
S_4	100	0	100	0
S_4	100	1	101	0
S_5	101	0	101	0
S_5	101	1	000	1

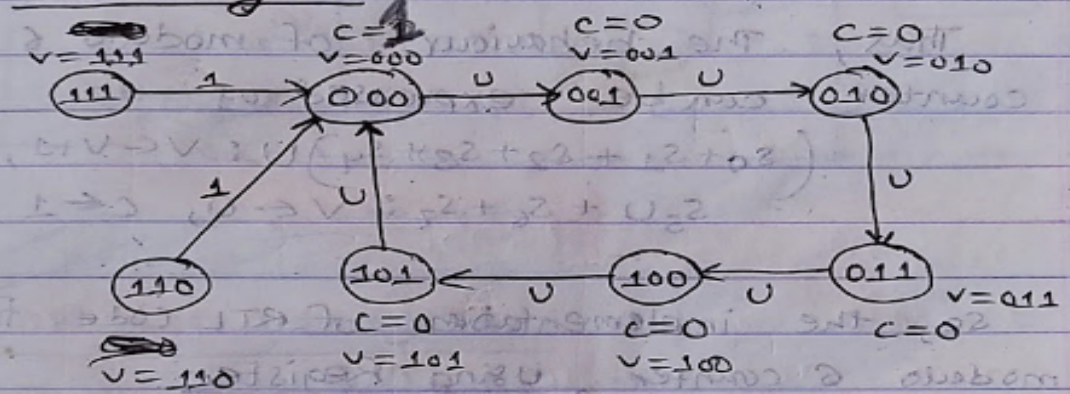
000→ 001→ 010→011→100→101→000

So
S1
S2
S3
S4
S5
S0

110	S_6	X	000	1
111	S_7	X	000	1

The additional states S_6 & S_7 to handle the case when mod – 6 counter powers up in an invalid state.

State diagram:



⇒ when counter value is 000 to 100 & its U (up) (U=1) signal is asserted, the output of counter is incremented. i.e

$(S_0+S_1+S_2+S_3+S_4)U$

under this condition, C is also set to 0.

S_0, $V \leftarrow V+1$, $C \leftarrow 0$

Thus, $(S_0+S_1+S_2+S_3+S_4)U: V \leftarrow V+1, C \leftarrow 0$

Additional states
S6
S7
for invalid state and errors

A state diagram is a diagram to describe the behavior of a system considering all the possible states of an object when an event occurs. This behavior is represented and analyzed in a series of events that occur in one or more possible states. Each diagram represents objects and tracks the various states of these objects throughout the system.

when counter is in state S_5 (V=101) & U=1, the counter must be reset to 000 & C must be 1 i-e,

$S_5U : V \leftarrow 0, C \leftarrow 1$

$V \leftarrow V+1$ 101 → 000

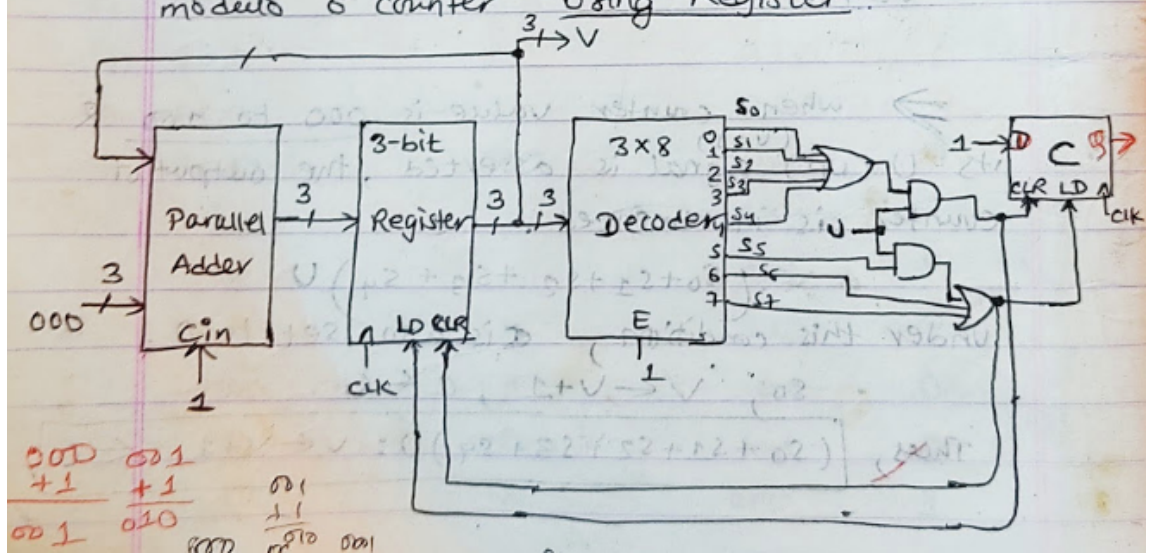
For invalid state, regardless of value of U,

$(S_6 + S_7) : V \leftarrow V+1, C \leftarrow 0$ or 1.
$V \leftarrow 0, C \leftarrow 1$

Thus, the behaviour of modulo 6 counter can be expressed as

$(S_0 + S_1 + S_2 + S_3 + S_4)U : V \leftarrow V+1, C \leftarrow 0$
$S_5U + S_6 + S_7 : V \leftarrow 0, C \leftarrow 1$

So, the implementation of RTL code for modulo 6 counter using Register



Register , decoder and adder used for the implementation of modulo 6 counter.

Using counter

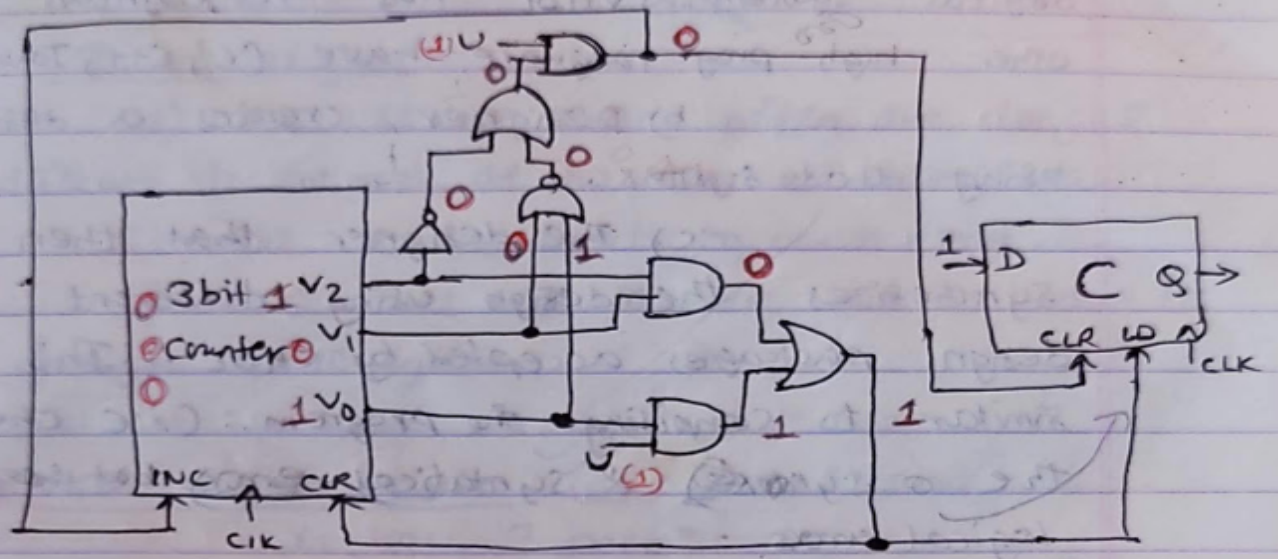


fig. (b)

from 101 - 000

Counter must be clear at some time

Clr load i.e L=1

Introduction to VHDL

VHDL ⇒ VHSIC Hardware description language

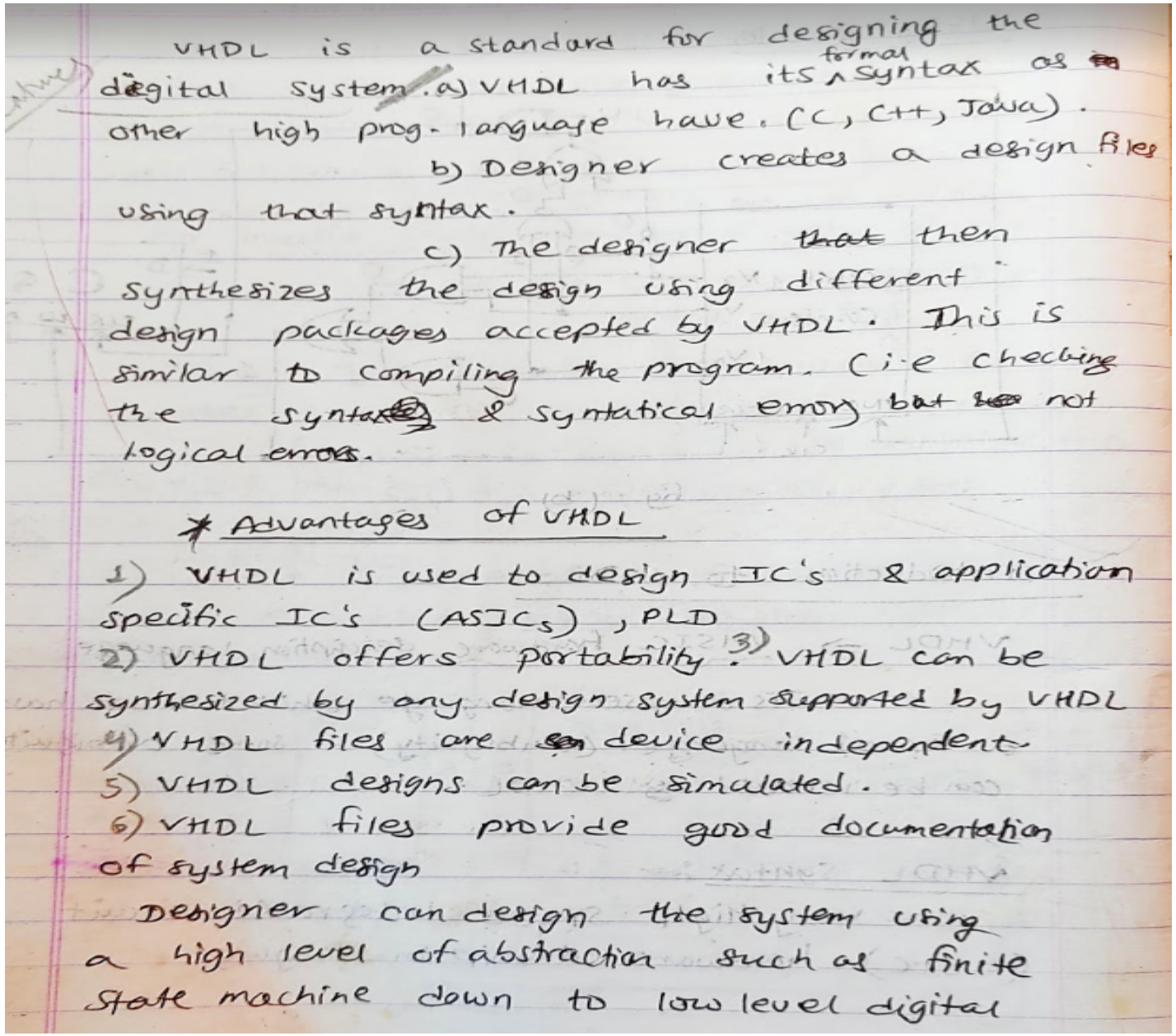
RTL is a pseudo language which does not have a formal language. (ambiguity). some ambiguity can be removed by VHDL.

VHDL Syntax :-

Very High Speed Integrated circuit (VHSIC) hardware description language.

Using counter to implement the modulo-6 counter

VHDL



VHDL is a standard for designing the digital system. a) VHDL has its formal syntax as other high prog. language have. (C, C++, Java).

b) Designer creates a design files using that syntax.

c) The designer then synthesizes the design using different design packages accepted by VHDL. This is similar to compiling the program. (i.e checking the syntax & syntatical error) but not logical errors.

* Advantages of VHDL

1) VHDL is used to design IC's & application specific IC's (ASICs), PLD
2) VHDL offers portability. 3) VHDL can be synthesized by any design system supported by VHDL
4) VHDL files are device independent
5) VHDL designs can be simulated.
6) VHDL files provide good documentation of system design

Designer can design the system using a high level of abstraction such as finite state machine down to low level digital

VHSIC Hardware Description Language

logic implementation.

Disadvantages:

1) VHDL source code often are long & difficult to follow at low level of abstraction.

2) VHDL are difficult to understand.

3) VHDL does not provide detail about its design implementation.

* VHDL design code

It has 3 parts:—

a) library declarations
b) entity section
c) architecture section.

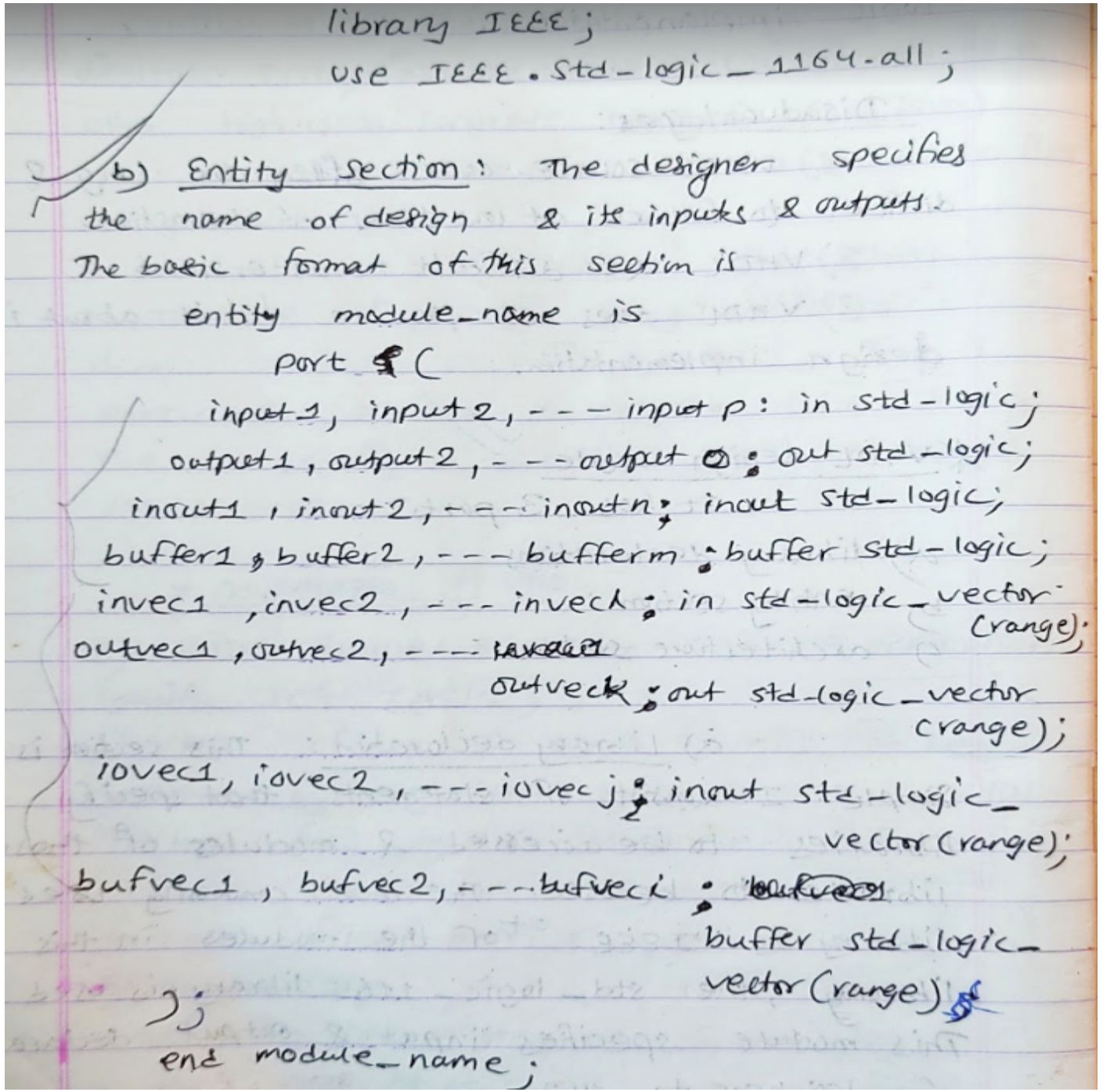
a) Library declaration: This section is simple. It consist of statements that specify libraries to be accessed & modules of those libraries to be used. The most commonly used library is IEEE. out of the modules in this library, the std_logic_1164 library is used This module specifies input & output declaration for designer to use.

Sections of VHDL code

Library section

Entity section

Architecture section



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

b) Entity section: The designer specifies
the name of design & its inputs & outputs.
The basic format of this section is
    entity module_name is
        port (
    input1, input2, --- inputp : in std_logic;
    output1, output2, --- outputo : out std_logic;
    inout1, inout2, --- inoutn : inout std_logic;
    buffer1, buffer2, --- bufferm : buffer std_logic;
    invec1, invec2, --- inveck : in std_logic_vector
                                 (range);
    outvec1, outvec2, --- outveck : out std_logic_vector
                                 (range);
    iovec1, iovec2, --- iovecj : inout std_logic_
                                 vector(range);
    bufvec1, bufvec2, --- bufveci : buffer std_logic_
                                 vector(range)
        );
    end module_name;
```

Entity section of VHDL

input port
output port
buffer
vector

The first two lines specify the module & initiate the declaration of its signals.
The signals input 1, input 2, --- input p are of type std-logic. This declares them as inputs to design.
if design is to be synthesized as a chip (IC, PLD) these inputs will be assigned to pins of chip during synthesis.

Type out std-logic define outputs of design

Type inout std-logic defines bidirectional signals

Type buffer std-logic is used for internal signal within design that are neither input nor output.

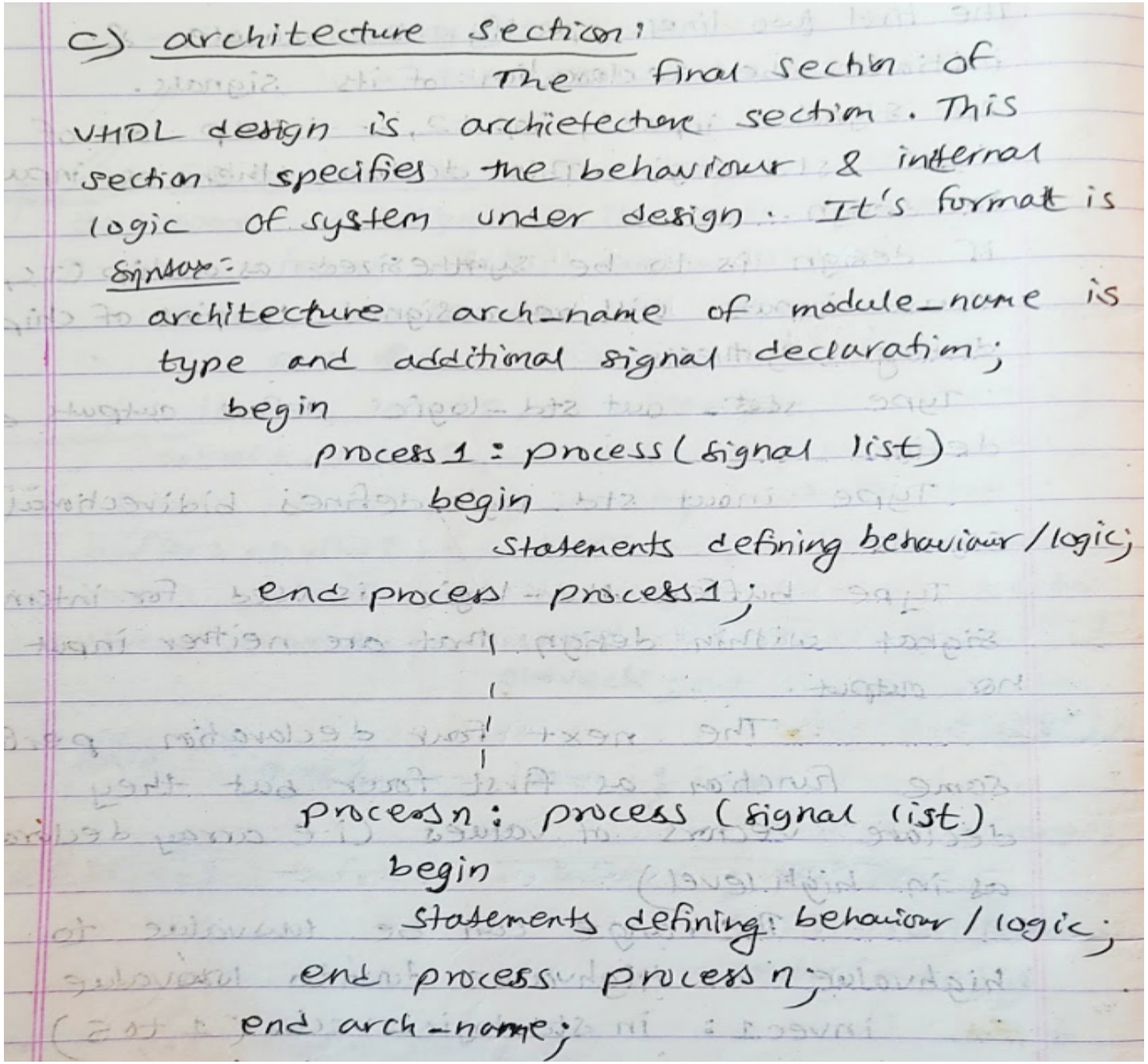
The next four declaration perform some function as first four but they declare vectors of values (i.e array declaration as in high level)

The range can be lowvalue to highvalue or highvalue down to lowvalue.

invec1 : in std-logic-vector (1 to 5)

invec2 : in std-logic-vector (6 downto 0)

Explantion of entity section



c) architecture section:

The final section of VHDL design is architecture section. This section specifies the behaviour & internal logic of system under design. It's format is

Syntax:

```
architecture arch_name of module_name is
    type and additional signal declaration;
begin
        process 1 : process (signal list)
            begin
                statements defining behaviour / logic;
    end process process 1;
        .
        .
        .
        process n : process (signal list)
            begin
                statements defining behaviour / logic;
    end process process n;
end arch_name;
```

Architecture section description

defining architecture and process

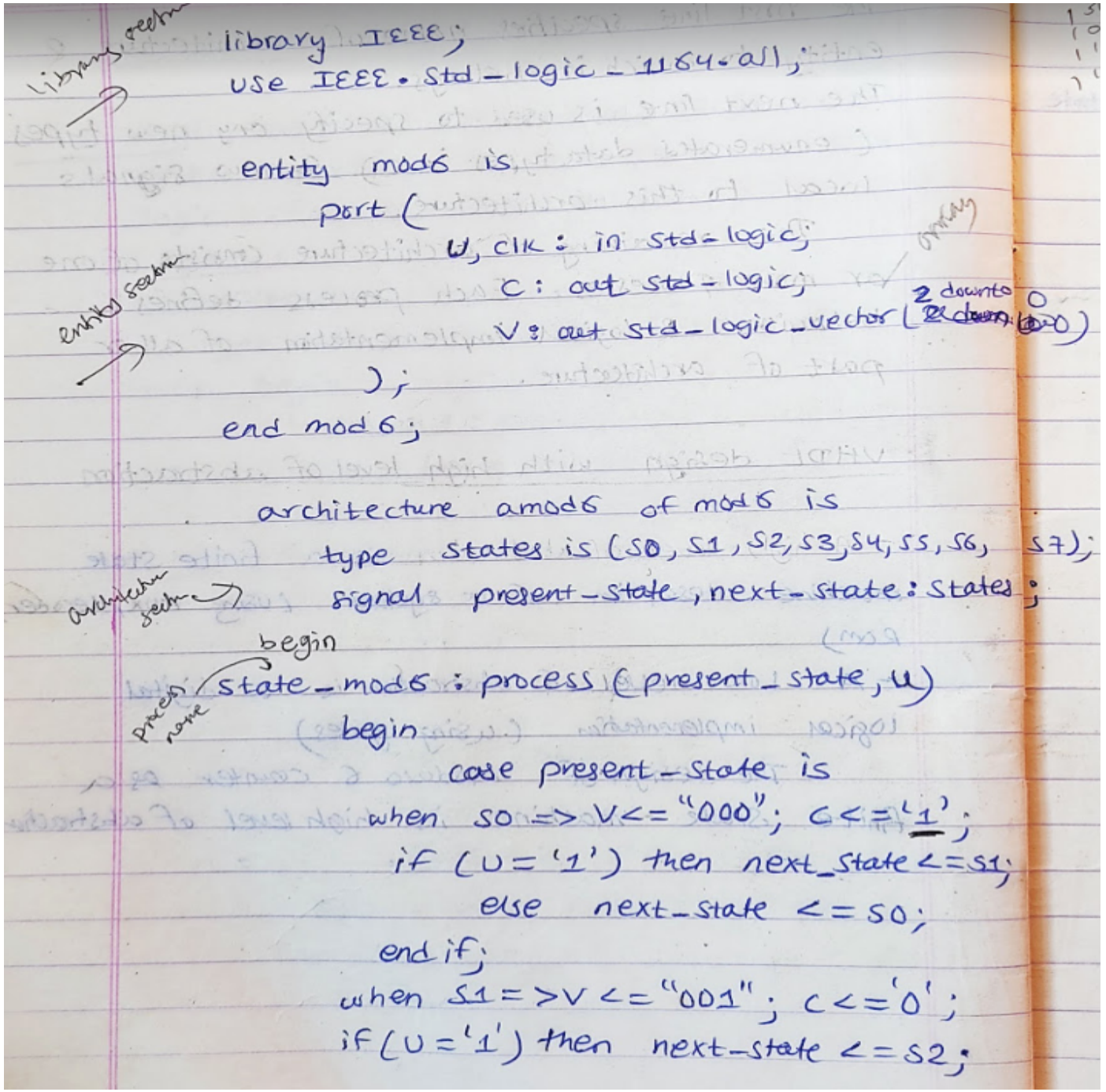
The first line specifies name of architecture & entity to which it belongs.
The next line is used to specify any new types (enumerated data types in C) & new signals local to this architecture.

The remaining of architecture consists of one or more processes. Each process defines behavioural & logical implementation of all or part of architecture.

VHDL design with high level of abstraction

- A high level of abstraction is a finite state machine description of a system (using MUX, decoder, ROM)
- A Low level of abstraction is a digital logical implementation (using gates)

The design of modulo 6 counter as a finite state machine is high level of abstraction.



library section

entity section

architecture section

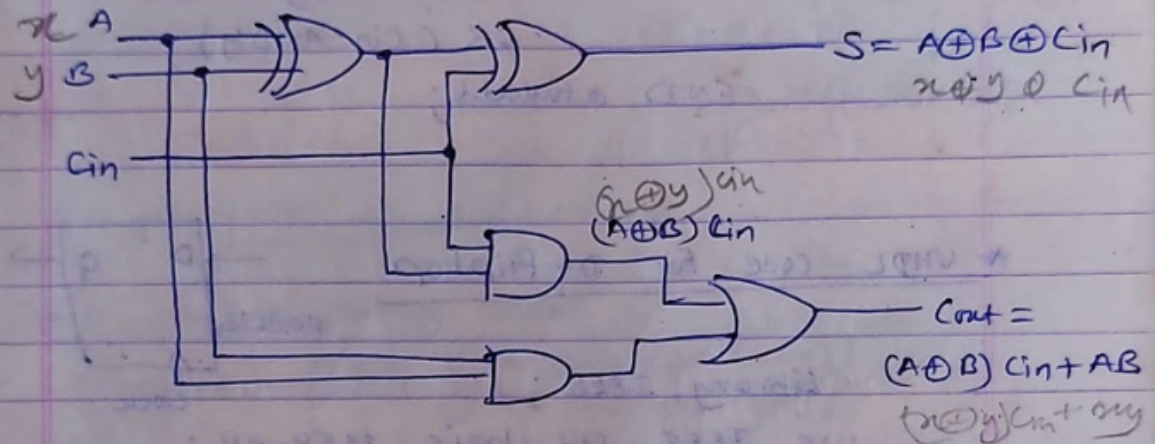
process

```
    else next_state <= S1;
    end if;
when S2 => V <= "010"; C <= '0';
    if (U='1') then next_state <= S3;
        else next_state <= S2;
    end if;
when S3 => V <= "011"; C <= '0';
    if (U='1') then next_state <= S4;
        else next_state <= S3; end if;
when S4 => V <= "100"; C <= '0';
    if (U='1') then next_state <= S5;
        else next_state <= S4;
    end if;
when S5 => V <= "101"; C <= '0';
    if (U='1') then next_state <= S0;
        else next_state <= S5;
    end if;
when S6 => V <= "110"; C <= '0';
        next_state <= S0;
    when S7 => V <= "111"; C <= '0';
            next_state <= S0;
    end case;
end process state_mod6;
```

```
state_transition : process (clk)
  begin
    if rising_edge (clk) then present_state <= next_state;
    end if;
  end process state_transition;
end amod6;
```

* VHDL code for full adder.

logic design (low level of abstraction)

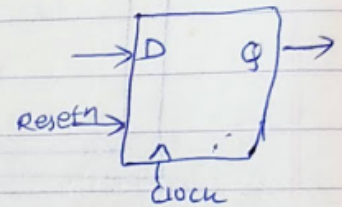



```
Library IEEE;
use IEEE.std_logic_1164.all;

Entity fulladd is
 port (
        Cin, x, y  : in std_logic;
        s, Cout    : out std_logic
 );
end fulladd;
```

```
architecture afulladd of fulladd is
begin
        s <= x XOR y XOR Cin;
        Cout <= (x AND y) OR (Cin AND x)
                OR (Cin AND y);
END afulladd;
```

* VHDL code for D-flipflop



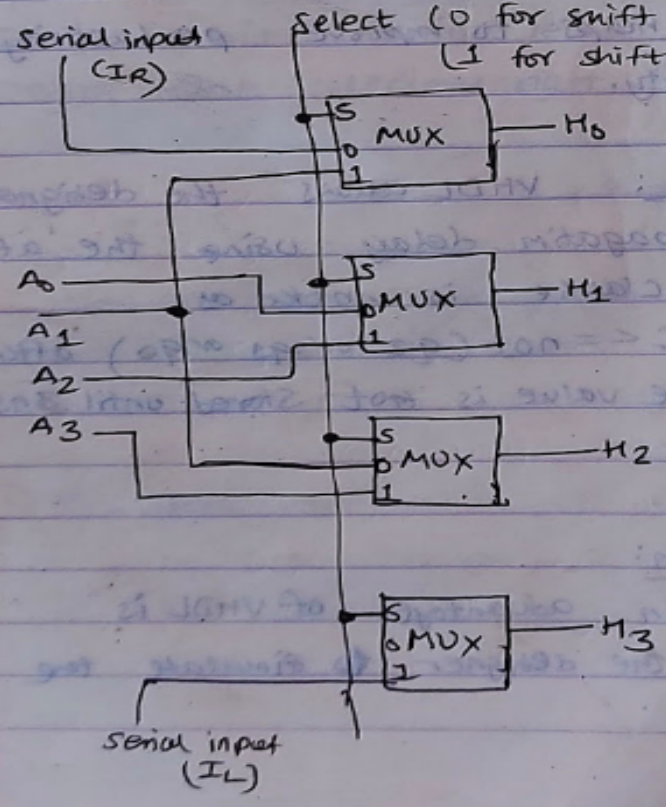

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity flipflop is
 port (
        D, Resetn, clock : in std_logic;
        Q                : out std_logic);
end flipflop;

architecture behaviour of flipflop is
begin
process
begin
```

```
wait until clock'event and clock = '1';
  if Resetn = '0' then
     Q <= '0';
     else
     Q <= D;
  end if
  end process;
end behaviour;
```

Q* Design a 4 bit shifting circuit.



	A0	A1	A2	A3
select S	H0	H1	H2	H3
0	IR	A0	A1	A2
1	A1	A2	A3	IL

O/P

Multiplexer:

A multiplexers is a circuit having number of data inputs, one or many select inputs and one output. It passes the single value on one of the data inputs to the output. The data inputs are selected by the values of the select inputs.

2 to 1 mux:

D_0	D_1	S	Y
1	0	0	1
0	1	1	1

$Y = S'W_0 + SW_1$

CODE:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux 2 to 1 IS
PORT(D0, D1, S: IN STD_LOGIC;
        f       : OUT STD_LOGIC);
END mux 2 to 1;

ARCHITECTURE Behavior OF mux 2 to 1 IS
BEGIN
WITH S SELECT
F<= D0 WHEN '0',
    D1 WHEN OTHERS;
END Behavior;
```

Four bit shift register:

Here the line code are numbered for ease of reference. Instead of using subcircuit, the shift register is described using a sequential statement. Due to the WAIT UNTIL statement in line 13, any signal that is assigned a value inside the process has to be implemented as the output of flip flop. Line 14 and 15 specifies the parallel loading of the shift register when L = 1. The ELSE clause in the line 16 to 20 specifies the shifting operation. Line 17 shifts the value of Q_1 in to the flip flop with the output Q_0. Line 18 and 19 shifts the value of Q_2 and Q_3 in to the flip flop with the output Q_1 and Q_2 respectively. Finally line 20 shifts the value of w in to the left most flip flop, which has the output Q_3. Hence all four flip flop changes their value at the same time, as required in the shift register.

CODE:

```
1.  LIBRARY ieee;
2.  USE ieee.std_logic_1164.all;

3.  ENTITY shift4 IS
4.  PORT (R          : IN     STD_LOGIC_VECTOR(3 DOWNTO O);
5.          Clock    : IN     STD_LOGIC ;
6.           L, w    : IN     STD_LOGIC ;
7.           Q       : BUFFER STD_LOGIC_VECTOR(3 DOWNTO O));
8.  END shift4;

9.  ARCHITECTURE Behavior OF shift4 IS
10. BEGIN
11. PROCESS
12. BEGIN
13.  WAIT UNTIL Clock'EVENT AND Clock = '1'
14. IF L= '1' THEN
15. Q<=R;
16. ELSE
17. Q(0)<= Q(1);
18. Q(1)<= Q(2);
19. Q(2)<= Q(3);
20. Q(3)<= w;
21. END IF;
22. END PROCESS;
23. END Behavior;
```

End of chapter 03