Module : III

MSI Logic (Medium scale Integrated Circuit)

Decodes

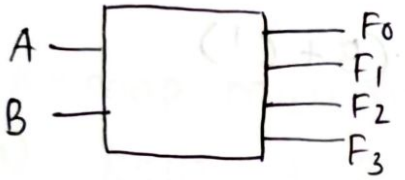
Circuit that takes in a binary code and has o/p s assaulted for specific values of i/p codes ie each output is specific for one-and-only one i/p code.

Normally combinational logic circuits only produce a single output but in case of decodes each o/p requires a seperate combinational circuit.

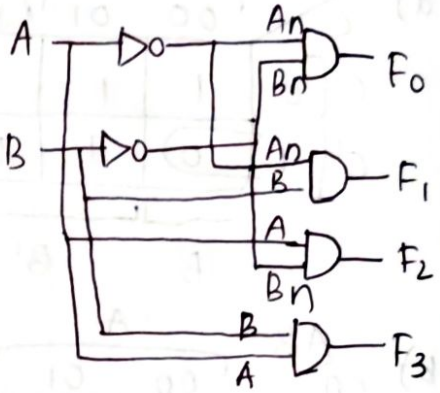
In decodes if there are n inputs the outputs will be 2^n.

One hot decoder

1)



A	B	F_3	F_2	F_1	F_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



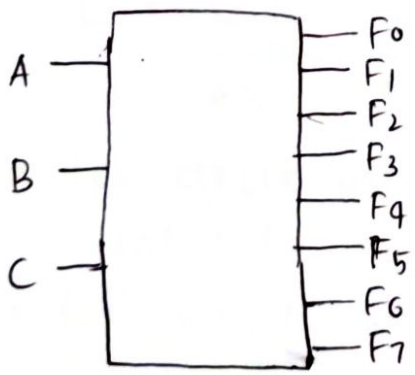
$$F_0 = \sum_{(A,B)}(0) = A'B'$$ (0 = row no.)

$$F_1 = \sum_{(A,B)}(1) = A'B$$

$$F_2 = \sum_{(A,B)}(2) = AB'$$

$$F_3 = \sum_{(A,B)}(3) = AB$$

2)



A	B	C	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$$F_0 = \sum_{(A,B,C)}(0) = A'B'C'$$

$$F_1 = \sum_{(A,B,C)}(1) = A'B'C$$

$F_2 = \sum_{(A,B,C)} (2) = A'BC'$

$F_3 = \sum_{(A,B,C)} (3) = A'BC$

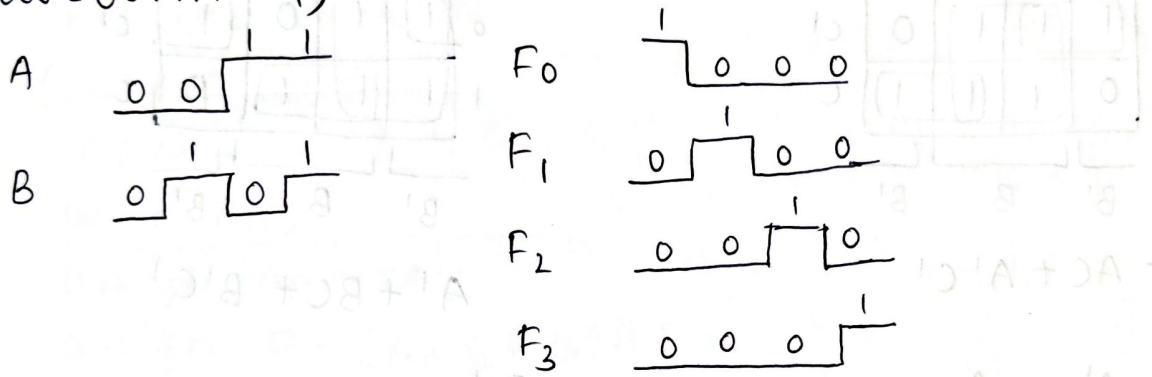
$F_4 = \sum_{(A,B,C)} (4) = AB'C'$

$F_5 = \sum_{(A,B,C)} (5) = AB'C$

$F_6 = \sum_{(A,B,C)} (6) = ABC'$

$F_7 = \sum_{(A,B,C)} (7) = ABC$

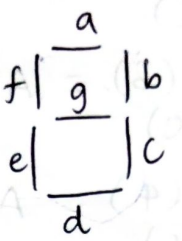
Waveform : 1)



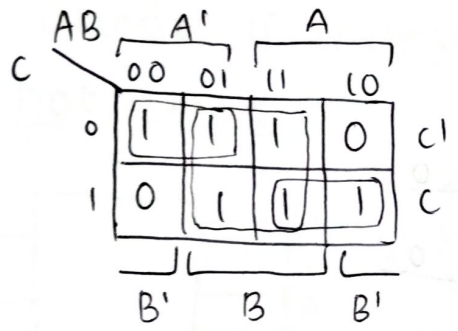
7 Segment display decoder

* Commonly used in applications such as digital clocks and other household appliances.
* It has upto 7 individual LED's labelled a-g
* The i/p to decoder is the binary equivalent of decimal or hex character that is to be displayed.
* Decoders with 2 inputs can drive characters 0-3
* Decoders with 3 inputs can drive characters 0-7
* Decordels with 4 inputs can drive characters 0-F

A	B	C	F^a	F^b	F^c	F^d	F^e	F^f	F^g
0	0	0	1	1	1	1	1	1	0
0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	0	0	1
1	0	0	0	1	1	0	0	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0

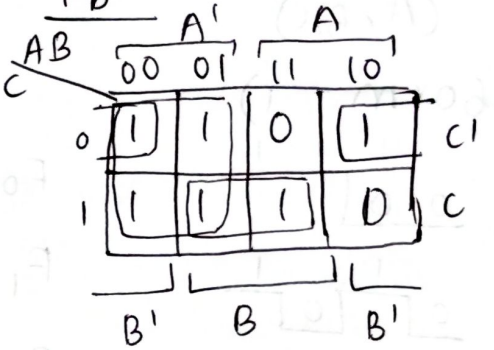


Fa



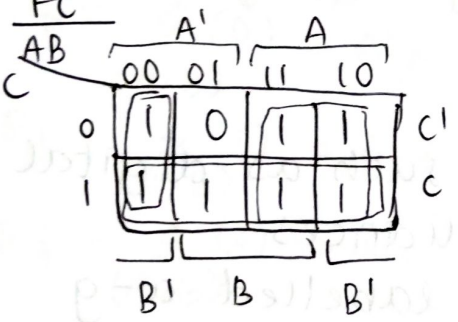
$B + AC + A'C'$

Fb



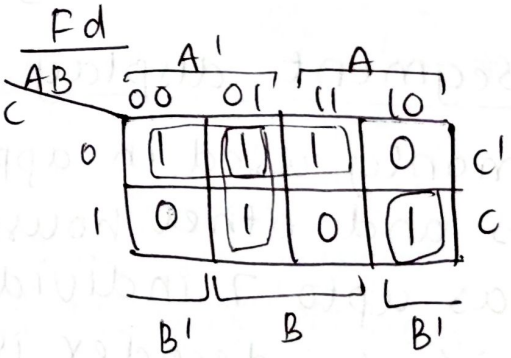
$A' + BC + B'C'$

Fc



$A + B' + C$

Fd



$AB'C + A'C' + BC' + A'B$

9/9/25

Verilog Concepts — Module I (continuation)

- A verilog describes a single system in a single file.
- The file has the sufix "*.v"
- within the file, the system description is contained in a module.
- A module includes the interface of the system, ie inputs and outputs and the description of system behaviour.

Eg:- Example.v

```
module example
(output wire F;
input wire A,B,C);
wire An;
assign An = !A;
assign F = (An && B)|| C
end module
```

Rules in Verilog

1. Verilog is case sensitive
2. Verilog assignment declaration and debinition is terminated with a semicolon
3. Comments in verilog are supported in two ways
   a) Line comment denoted by "//" : Any sentence abter // will be considered as a comment till the end of the line.
   b) Block comment denoted by /* ... */ Any sentence coming in blw /* and */ is considered the comment
4. Value Set

   verilog supports four values ie, that a signal can take:

a) Zero : logic 0 or false condition
b) One : logic 1 or true condition
c) x or X : unknown or uninitialised
d) z or Z : high impedance, tristated or floating

5. Net Data Types : A net data type models an interconnection blw components and can take the values 0, 1, x or z.

Types

a) Wire : a simple connection blw components
2. wor (wired OR) : the values of multiple drivers are ORed together.
3. wand (wired AND)
4. Supply zero : To model VSS, GND powersupply
5. Supply one : To model Vcc power supply
6. Tri : For readability of net driven by multiple sources.
7. Trior : similar to wor
8. Triand : similar to wand
9. Trione : logic one when tristated
10. Trizero : logic zero when tristated
11. Trireg (Tri register) : holds last value when tristated.

Variable data Types

1. Reg : • Variable used for logic storage
• It can hold the values 0, 1, x and z.
2. Integer : A 32 bit 2's complement representing whole numbers
3. Real : A 64 bit floating point variable

representing real numbers.

4. Time : unsigned 64 bit variable taking on values from 0 to $+(9.2 \times 10^{18})$

5. Real time :

10/9/25

Vector

- In verilog, a vector is a one dimensional array. The syntax is as follows

<type> [MSB_index : LSB_index] vector_name
(vector_name: user defined)

eg:- wire [7:0] sum;

This is an 8 bit vector of type wire and name sum.

- reg [15:0] Q;

This is a 16 bit vector of type reg, name Q.

- Individual bits within the vector can be addressed using
- Groups of bits can be accessed using an index range.

eg: individual : sum[0];
group : Q[15:8];

Arrays

- An array is a multidimensional array of elements. Also known as vector of vectors.

The syntax is :

<element_type> [MSB_index : LSB_index] <Array-name> [array-startindex> : <array_endindex>]

eg:- reg[7:0] Mem [0 : 4095];

This is an array named Mem containing

4096 elements which is also an type 8 bit vector of type reg.

- integer A[1:100];
  This is array named A having 100 elements and elements are of type integer.

Expressing Numbers using Different Bases

Syntax: <size_in_bits>'<base><value>

1. a) 'b : unsigned binary
   2) 'o : unsigned octal
   3) 'd : unsigned decimal
   4) 'h : " hexadecimal
   5) 'sb : signed binary
   6) 'so : " octal
   7) 'sd : " decimal
   8) 'sh : " hexadecimal
   9) 4'b1111 : four bit binary no. with value 1111 to the base 2.

Module

Pre-verilog 2001 syntax:

```
module module-name [port_list];
// port definitions
// module_items
end module
```

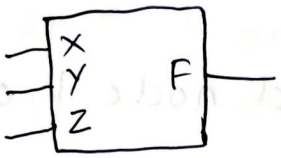
Verilog 2001 and after format syntax

```
module module-item [port_list, port-definitions];
//module-items
end module
```

Port Definitions

- The first items to be debined inside a module is collectively known as ports.
- Each port has a user defined name, a direction and type.
- The port directions are three types: input, output and inout

⇒ System I



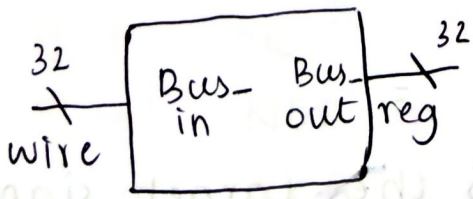
a) Preverilog 2001:

```
module System_1 (F,X,Y,Z);
output F;
input X,Y,Z; // port directions
wire F;
wire X,Y,Z; // port types
//module items
end module.
```

b) verilog:

```
module System_1 (output wire F, input wire
                                X,Y,Z);
// module items
end module.
```

⇒ System II



Verilog:

```
module system2 (output reg Bus-out [31:0],
                input wire Bus-in [31:0]);
// module items
end module
```

11/9

Signal Declaration

- A signal ~~declarat~~ is used for internal connections within a system and is declared within the module ~~with~~ before its first use.

Syntax : <type>name

```
wire node I;  // signal called node I of
                 type wire
reg Q2, Q1, Q0;
wire [63:0] bus I; // a 64 bit vector named
                      bus I with all of its bits
                      having type wire.
```

Parameter Declaration

- A parameter or constant is used for representing a quantity that will be used multiple times in the architecture.

Syntax:

parameter <type> constan-name = <value>;

eg: parameter Bus_width = 64; (no type specified, hence would be Integer).

Verilog Operators

1. Assignment operator
   - Denoted using the symbol =
   - The LHS of assignment is the target signal

and RHS of assignment contains input arguments, signals, constants and operators.

eg: $F_1 = A$

2. Bitwise logical operators

- Bitwise operators perform logic functions on individual bits.

Types:

1) ~ : negation
   eg: ~X : invert each bit in X
2) & : and
3) | : or
4) ^ : XOR
   eg: X^Y
5) ~^ : XNOR eg: X~^Y
6) << : logical left shift
   Fills empty lsb locations with zero
7) >> : logical right shift
   Fills empty msb locations with zero.

3. Reduction logic operators

- It is an operator that uses each bit of a vector as individual inputs into a logic operatn and produces a single bit output

1) & : AND all bits of a vector together (one bit result)
2) ~& : NAND all bits of a vector together.
3) | : OR
4) ~| : NOR
5) ^ : XOR

6) ~^ : XNOR

4. Boolean logic operators

- Boolean logic operator is the one that returns a value of True (1) or False (0) based on logic operations of inputs

Type:

1) ! : Negation
2) && : AND
3) || : OR

eg: X&&Y : True if the bitwise AND of X and Y results in all ones, false otherwise

5. Relational Operator

- A relational operator is the one that returns a value true or false based on comparison of two inputs.
- symbols : == , != , < , > , <= , >=

6. Conditional Operator

Syntax : <target_net> = <boolean_condition> ?
<true-assignment> : <false_assignment>;

- The keyword for conditional operator is '?'
- This operator specifies a Boolean condition which if evaluated true, the true assignment will be assigned to the target and if evaluated false, false assignment will be assigned to the target.

eg: • F = (A == 1'b0) ? 1'b1 : 1'b0;

If A is zero, F=1 otherwise F=0

- F = (sel == 1'b0) ? A : B;

If sel is zero, F = A otherwise F = B.

7. Concatenation Operator

- In verilog {} are used to concatenate multiple signals

eg: Bus1 [7:0] = { Bus2[7:4], Bus3[3.0]}

The upper four bits of Bus2 is concatenated with the lower four bits of Bus 3 and is assigned to the 8 bit combination of Bus1

8. Replication Operator

- Verilog provides the ability to concatenate a vector with itself through replication operator denoted by {{ }}

Syntax:

{no_of_replications{{vector_to_be_replicated}};

eg: BusX = {4{Bus1}};

This is equivalent to:

Bus X = {Bus1, Bus1, Bus1, Bus1}

- Bus Z = {Bus1, {2{Bus1}}};

ie, Bus Z = {Bus1, Bus2, Bus2}

9. Numerical operators

1) + : Addition

2) - : . subtraction (when placed blw arguments)

. Two's complement & negation (when placed in front of an argument.

3) * : Multiplication

4) / : Division
5) % : Modulus
6) ** : Raised to power [x**y = x^y]
7) <<< : shift to left, fill with zeroes
8) >>> : shift to right, fill with signed bits

Module: 2

Modelling Concurrent Functionality in Verilog

- In normal programming languages the lines of codes are executed sequentially as they appear in the source file.
- In verilog, the lines of codes represent behaviour of real hardware. Therefore assignments are executed concurrently.

Continuous Assignment

Verilog uses the keyword 'assign' to denote continuous assignment. After the keyword 'assign' an assignment is made using '=' symbol.

- The LHS of the assignment is the target signal and must be a net type and RHS of the assignment contains i/p arguments which can contain regs, nets, constants or operators.
- Continuous assignment models combinational logic.

Eg:-
```
assign F1 = A;
assign F2 = 1'b0;
assign F3 = 1'b1;
```

F_1 is updated any time A changes where A is a signal.

- assign X = A;
  assign Y = B;
  assign Z = C;

~~assig~~ The signal assignment in the above code happens at the same time ie, assign concurrently

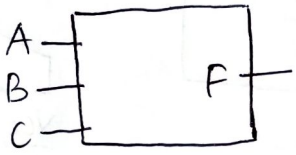
- assign A = B;
  assign B = C;

In verilog, the signal assignments are C to B and B to A will take place at the same time, ie during synthesis the signal B will be eliminated from the design.

Continuous Assignment with Logical operator

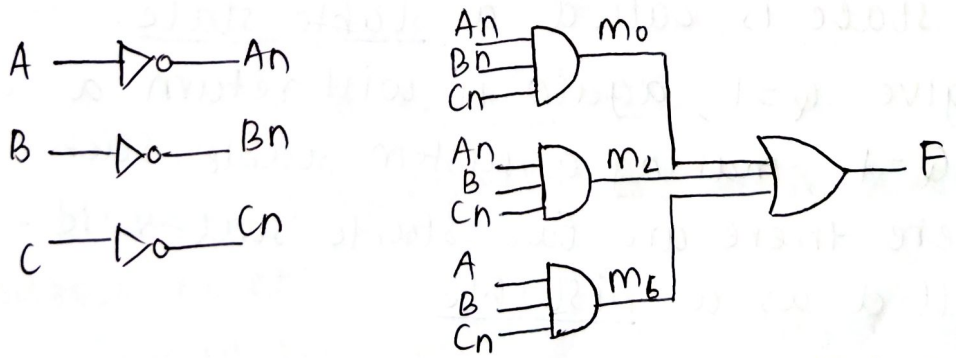
A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

SystemX.v



$F = \sum_{A,B,C} (0,2,6)$

$= A'B'C' + A'BC' + ABC'$



module SystemX (output wire F, input wire A,B,C);

wire An, Bn, Cn;

wire m0, m2, m6;

```
assign An = ~A;
assign Bn = ~B;
assign Cn = ~C;
assign m0 = An & Bn & Cn;
assign m2 = An & B & Cn;
assign m6 = A & B & Cn;
assign F = m0 | m2 | m6;
end module
```