



BIRZEIT UNIVERSITY

ANSWER BOOKLET

Student: <u>HP Digital</u> Number <u>7</u>
Course: Department: _____ Number: _____
Division: _____ Instructor: _____
Date: _____ Day Month Year

For Instructor's Use

Question	Grade
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
Total	

- The limitation of this encoder is: only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination.

e.g. if D_3 and D_6 are 1 simultaneously $\Rightarrow$ output will be 111 (but this doesn't represent either binary 3 or binary 6).

$\Rightarrow$ to solve this problem, we can establish a priority encoder.

- if we establish a higher priority for inputs with higher subscript numbers, now if both D_3 and D_6 are 1 at the same time $\Rightarrow$ the output will be 110 because D_6 has higher priority than D_3 .

⊗ Another problem

$xyz = 000$ when $D_0 = 0 \dots D_7 = 0$

and $xyz = 000$ when $D_0 = 1, D_1 = 0, D_2 = 0 \dots D_7 = 0$

$\Rightarrow$ this problem can be solved by providing one more output to indicate that at least one input is equal to 1.

Example 4-bit Priority Encoder

- if 2 or more inputs are equal to 1 at the same time $\Rightarrow$ the input having the highest priority will take precedence

inputs				outputs		
D_0	D_1	D_2	D_3	x	y	$V \leftarrow \text{valid}$
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

instead of listing 16 terms
(condensed form truth table)

eg $\begin{matrix} X & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$ represent

$D_2 D_3$ \ $D_0 D_1$	00	01	11	10
00	X	1	1	1
01		1	1	1
11		1	1	1
10		1	1	1

$D_2 D_3$ \ $D_0 D_1$	00	01	11	10
00	X	1	1	
01		1	1	
11		1	1	
10		1	1	

$$x = D_2 + D_3$$

$$y = D_3 + D_1 D_2'$$

$$V = D_0 + D_1 + D_2 + D_3$$

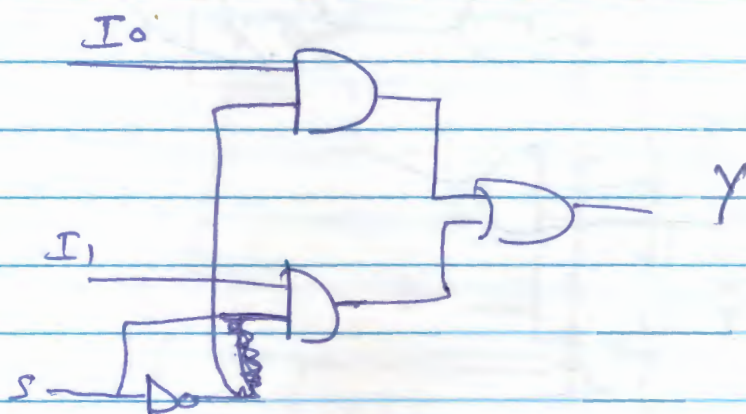
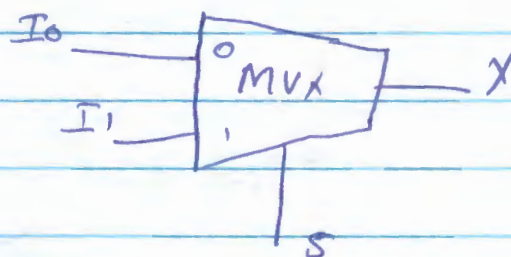
⊕ Multiplexers :-

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.

- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

Ex: 2-to-1 line multiplexer

S	X
0	I_0
1	I_1



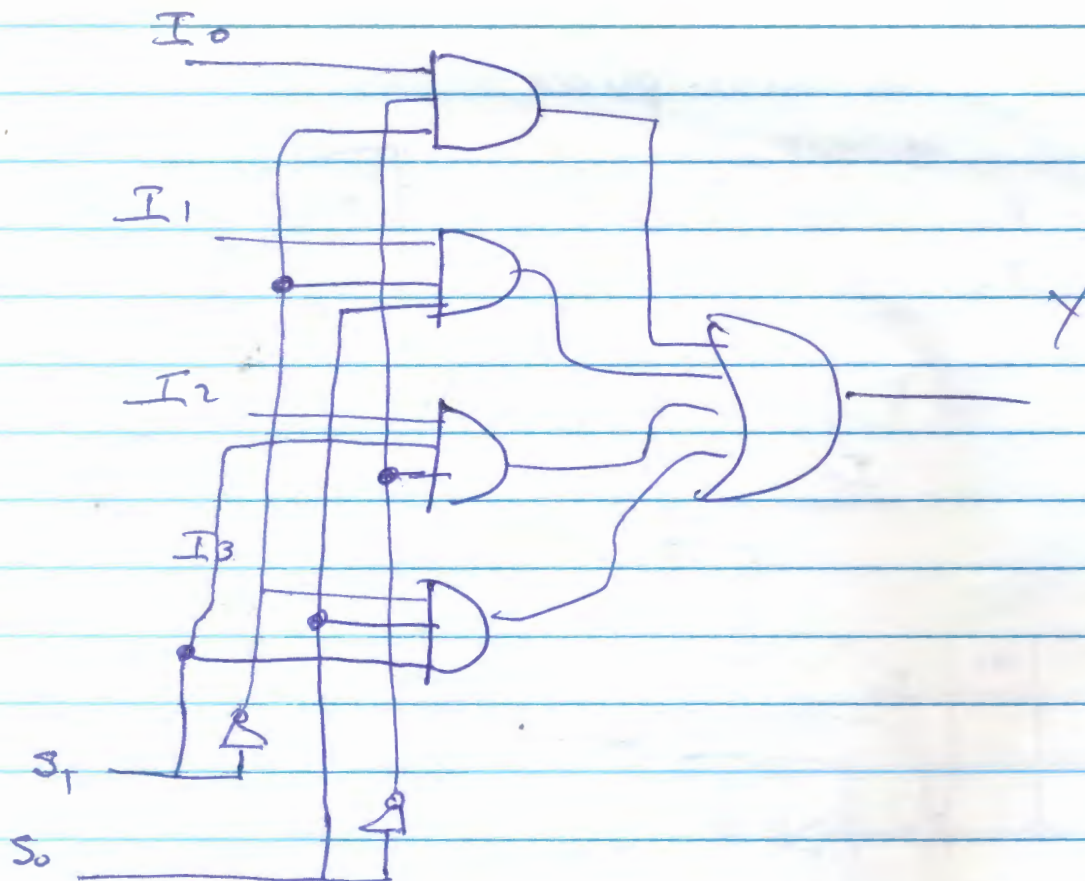
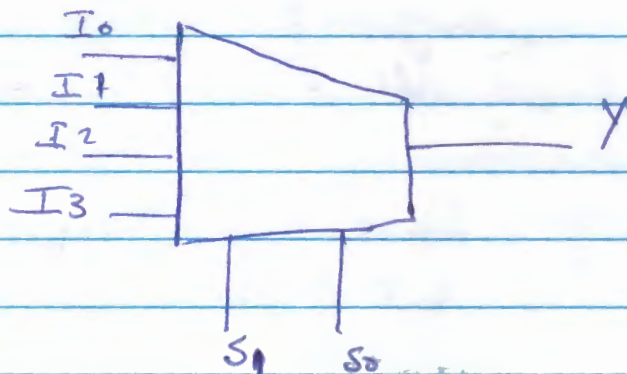
S	I_0	I_1	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$I_0 I_1$	00	01	11	10
0			1	1
1		1	1	

$$Y = S'I_0 + SI_1$$

Q Ex: 4-to-1 line MUX

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

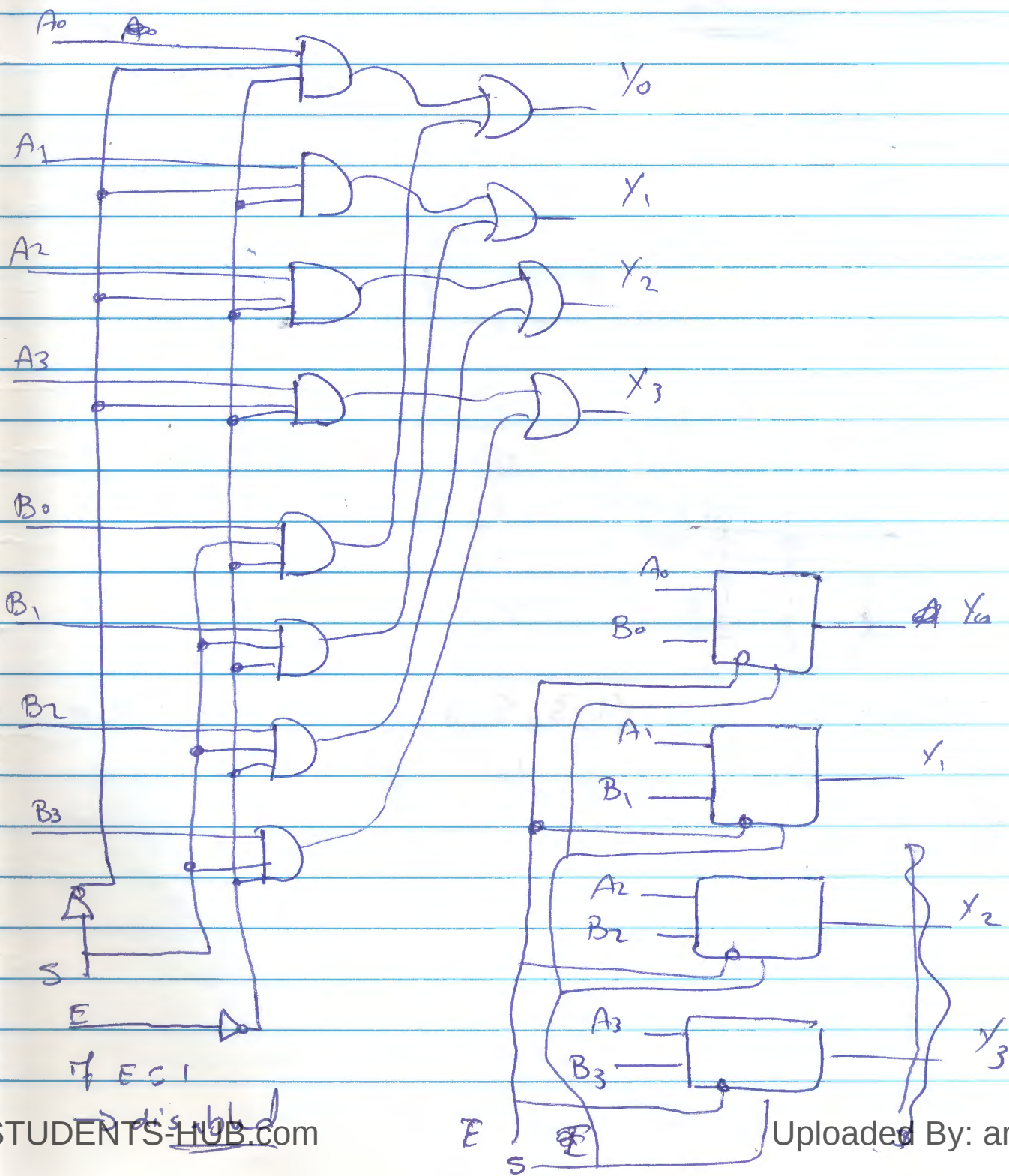


- Mux is decoder that decodes the selections (and we add 2^n input lines, one for each and gate, and all and gates are combined with OR gate).

- Mux may have enable input

- Multiplexer circuit can be combined with common selection inputs to provide multiple bit selection logic.

e.g quaduple 2-to-1 line mux

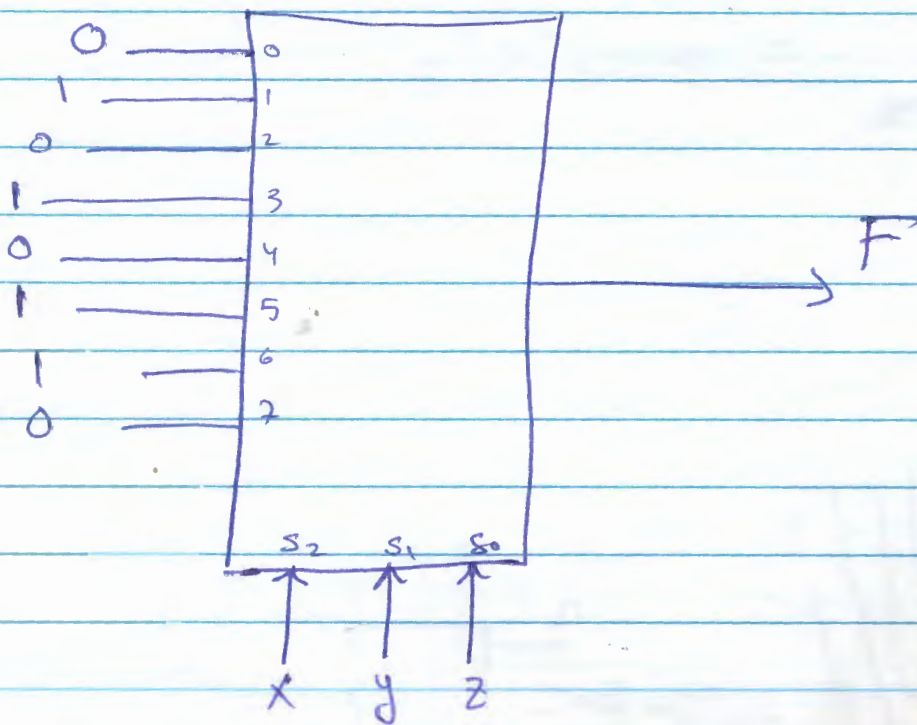


⊕ Boolean Function Implementation

Ex. $F(x, y, z) = \sum(1, 3, 5, 6)$

use 8-to-1 mux to implement this function

Solution : trivial



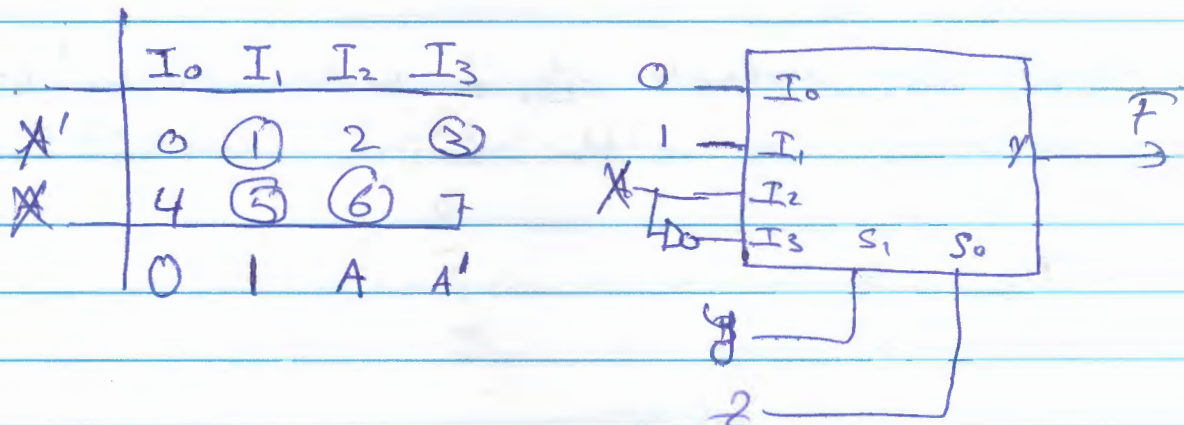
Ex. $F(x, y, z) = \sum(1, 3, 5, 6)$

use 4-to-1 mux to implement this function:

first solution

minterm	x	y	z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

- Chose to connect y, z to selection lines
and A to input lines



2nd way

1 2nd way

~~min-term F x y z~~
~~0 0 0~~

min-term	F	x	y	z	
0	0	0	0	0	} I_0
4	0	1	0	0	
1	1	0	0	1	} I_1
5	1	1	0	1	
2	0	0	1	0	} I_2
6	1	0	1	0	
3	1	0	1	1	} I_3
7	0	1	1	1	

- chose to connect y, z with selection inputs, x with input lines

- $I_0 = 0$

- $I_1 = 1$

- $I_2 = x$

- $I_3 = x'$

Ex: $F(A, B, C) = \sum(1, 2, 4, 5)$

using 4x1 mux

- connect A, B with S_1, S_0 .

m.b	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

$F = C$

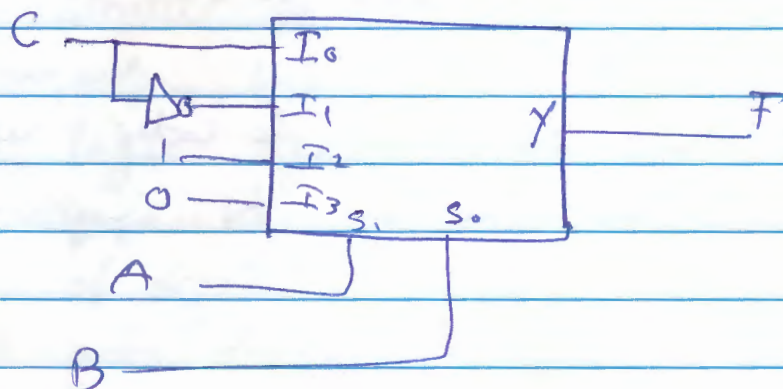
$F = C'$

$F = 1$

$F = 0$

or

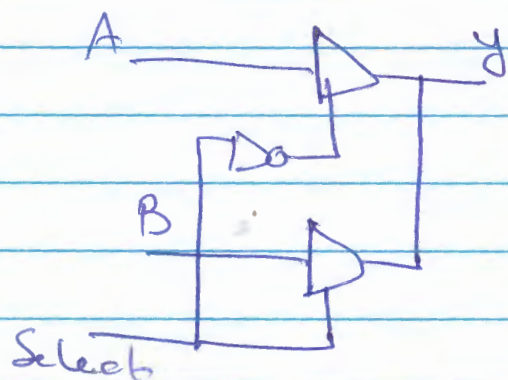
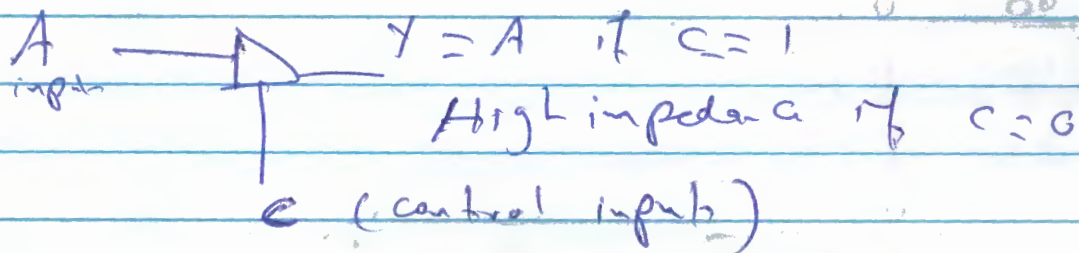
	I_0	I_1	I_2	I_3
C'	0	(2)	(4)	6
C	(1)	3	(5)	7
	C	C'	1	0



Ex $F(x, y, z) = \sum (1, 2, 6, 7)$

$F(A, B, C, D) = \sum (1, 3, 4, 11, 12, 13, 14, 15)$

④ three state gate



(using wim to connect outputs to
(advantage of 3-state buffer)

if select = 0

→ $y = A$

if select = 1 → $y = B$

④ how to use decoder with three state buffer to implement mux??

* VERILOG HDL

- HDL (Hardware Description Language) is a language that describes the hardware of digital systems in a textual form.

- two applications of HDL processing

① Logic Simulation: representation of the structure and behavior of a digital logic system through the use of a computer.

(readable output, timing diagrams) that predicts how the hardware will behave before it is actually fabricated.

Simulation allows the detection of functional errors in a design without having the physical circuit.

② Logic Synthesis: is the process of deriving a list of components and their interconnections (called netlist) from the model of a digital system described in HDL.

Logic synthesis is similar to compiling a program in high level language, but instead of producing an object code, logic synthesis produces a database with instructions on how to fabricate a physical piece of digital hardware that implements ~~digital~~ the statements described by the HDL code.

- VHDL vs Verilog.

⊛ Verilog HDL

- 100 keyword, lower case, case sensitive
- use // for comments
- A module is the building block in verilog. It is declared by the keyword module and is always terminated by endmodule
- module and endmodule
- Semicolon
- input and output for ports
- wire for internal connection
- name of gates are optional

⊛ Example 3-1 HDL

⊛ Example 3-2 HDL (Circuit with delay)

- test bench: An HDL description that provides the stimulus to a design is called a test bench.

⊛ Example 3-3

- two modules are included: a stimulus module and circuit description module.
- The stimulus module has no ports.
- the inputs to the circuit are declared with reg and the outputs with a wire keyword.
- in test bench, the second module will be instantiated.

Stimulus module

```
module testcircuit  
  reg TA, TB;  
  wire TC;  
  circuit c1 (TA, TB, TC)
```

Design module

```
module circuit  
  (A, B, C);  
  input A, B;  
  output C;
```

- The initial statement specifies the inputs between the keywords begin and end.
- (1'b0 : one binary digit with a value of 0)
- \$finish : system task to finish timing in simulation
- ⊛ keywords: reg, initial, begin, \$finish, end.

⊛ HDL example 3-4

- Assign keyword
- symbols (&), (|) and (~) for and, or, and not.

User-Defined Primitive (UDP)

- logic gates like (and, or, ...) are defined by the system and are referred to as system primitives.

- The user can create additional primitives by defining them in a tabular form (eg truth table).

- no module keyword, but primitive keyword.

④ HDL Example 3-5

- declared with the keyword primitive followed by a name and port list
- only one output, listed first, declared with output key word
- any number of inputs, order is important between input declaration and table.
- truth table between keywords table and endtable
- inputs ending with : , output with ;
- ends with endprimitive

⇒ this ~~is~~ user primitive will be like system primitive and can be used by other circuits.

⊕ Section 4-11 "HDL for Combinational Circuits"

⊕ Gate Level modelling

- four valued Logic : 0, 1, x (unknown),
Z (high impedance)

and gate

	0	1	x	Z
0	0	0	0	0
1	0	1	x	x

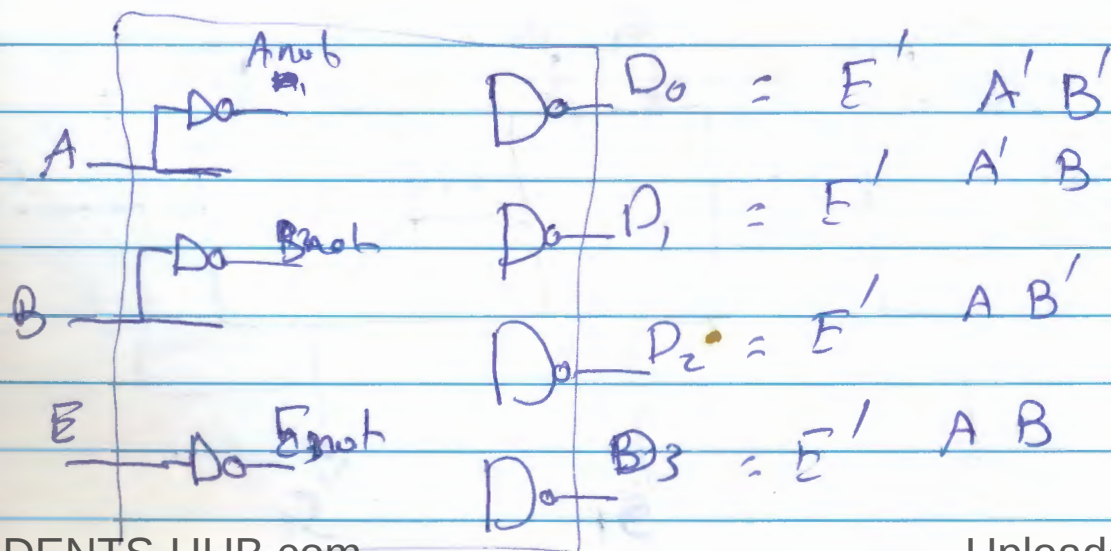
or gate

	0	1	x	Z
0	0	1	x	x
1	1	1	1	1

not

	0	1	x	Z
0	1	0	x	x

⊕ 2-to-4 line decoder



⇒ gate level description (EX HDL 4.1)

& data flow modelling (EX HDL 4.3).

④ Binary Adder

Example 4-bit adder

- top-down design.
- bottom-up design.

Gate level of bottom up design

- half adder

