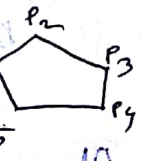
Right

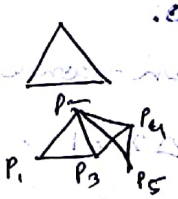
CSE-458 — Graphics

Quiz + Viva

* OpenGL — Open Graphics Library
* Vector graphics — images represented as mathematical function
  Raster — " " " as grid / pixels.
* 3 types of geometric pattern — (i) Points (ii) Lines, (iii) Polygon.
* glut32.dll → system folder
  " .lib → codeblocks/MinGW/lib
  glut.h → " " / include/GL
* glVertex3f(~)
  - belong to GL library
  - function name
  - no. of components
  - data type
  - parameter
* GL_POINTS — draw individual point
  " _LINES — draw a line
  " _LINE_STRIP — series of connected line
  " _LINE_LOOP — segment added between first and last
  " _POLYGON — simple, convex polygon



" _TRIANGLES — draw triangle
" _ " _STRIP — linked triangle
" _ " _FAN — linked fan of "
" _QUADS — quadruplet of vertices



glutInit() — initialize the GLUT library and negotiate a session with window system.

glBegin() - glEnd() — geometric primitives (points lines, triangles) can be specified by a list of vertex. data between these two.

glFlush() — empties all command in these buffers and forces all pending commands to be executed.

Event	OpenGL
Keyboard	glutKeyboardFunc
mouse	" MouseFunc

glMatrixMode — specifies which matrix is the target

glLoadIdentity — Replaces the current matrix with Identity matrix

gluOrtho2D — define a 2D orthographic projection matrix

glClearColor — specifies red, green, blue used by glClear to clear the buffer

glClear — sets the bit area selected by glClearColor.

glutInitDisplayMode — create top level window.

GLUT_SINGLE — single buffered window

GLUT_RGB — RGBA window mode.

DDA Line Algo.

incremental scan-conversion method

if $|m| < 1$, $\Delta x = 1$, $y_{i+1} = y_i + m$; setPixel(x, round(y)), x++;

else $|m| > 1$, $\Delta y = 1$, $x_{i+1} = x_i + 1/m$; setPixel(round(x), y);

Code — $dx = x2 - x1$, $dy = x - 1$

abs(dx) > abs(dy) → inc = dx else inc = dy.

$xin = \frac{dx}{inc}$, $yin = \frac{dy}{inc}$

for(i=1, i<=inc; i++)

$x \pm xin$; $y \pm yin$

glVertex2i(x,y);

Bresenham's Line Algo

highly efficient incremental method

$dx = x2 - x1$, $dy = y2 - y1$; $P = 2dy - dx$; plot(x1, y1)

while (not equal) (GL_POINTS

if (p < 0)

plot (x1+1, y1) ; $p \pm 2dy$; x1++ / $2(dx-dy)$; y1++

else

plot (x1+1, y1+1) ; $p \pm 2(dy-dx)$; x1++, y1++ / $2dy - 2dx$; y1++ ; x1++

Constraint — it only works when slope, m → $0 < m < 1$

Apply B.L. at out of $0 < m < 1$ by mirroring them either horizontally / vertically / diagonally.

(#) for $-1 < m < 0$, has a hor. counterpart at $(x_1, -y_1)$ $(x_2, -y_2)$ with $0 < m < 1$

so scan convert this counterpart at but negate y co-ordinate at each end of iteration.

(#) if slope between 45° to 90° ($m > 1$) exchange x and y co-ordinate of endpoint

The scan-convert, but must change x and y in the call to setPixel.

Midpoint Circle

$x = 0$, $y = r$, $P = \frac{5}{4} - r$. plot(x,y)

while (y > x)

if P < 0 ; x++ ; $P \pm 2*x + 1$ (GL_POINTS)

else, y--, x++, $P \pm 2(x-y) + 1$;

plot(x,y), ... (8-way symmetry points)

Midpoint Ellipse

plot( ) {

glVertex2D (x,y) . (-x,y) . . (-x,-y) . . (x,-y) }

(GL_Points)

x = 0, y = b ; aa = a * a , bb = b * b , aa2 = aa * 2, bb2 = bb * 2

fx = 0 , fy = aa2 * b , p = bb − aa * b + 0.25 aa ,

while (fx < fy) {

plot (x,y) , x++ , fx ± bb2

if (p < 0) → p += fx + bb

else y-- , fy -= aa2 ; p += fx − fy + bb

}

plot (x,y)

$p = bb * (x + \frac{1}{2})^2 + aa * (y-1)^2 - aa * bb$

while (y > 0) {

y-- ; fy -= aa2

if (p > 0) → p -= fy + aa

else x++ , fx += bb2 ; p += fx − fy + aa

plot (x,y)

}

S-Gasket Algo

primitive is filled triangle , task is to take out the area defined by lines connecting the midpoint of the edge of a filled triangle.

S_0 S_1 S_2

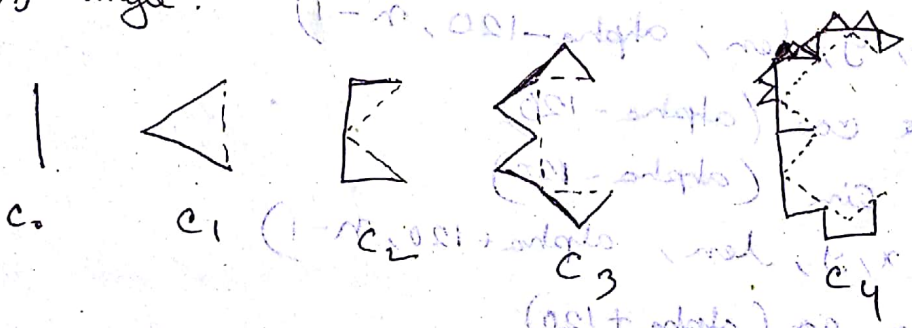
```
triangle( )
    glVertex2f(x1, y1), glVertex2f(x2) . . . . ;

gasket ( x1,y1, x2,y2,x3, ~~y2~~, y3 , n)
    float x12, x13, x23 , y12 , y13 , y23 ;
if (n>0) {
    x12 = (x1+x2)/2,  x13 = (x1+x3)/2 ,  x23 = (x2+x3)/2
    y   "      y   "      y   "
    gasket (x1, y1,  x12 , y12 , x13 , y13 , n-1) ;
    gasket ( x12, y12 , x2, y2, x23 , y23 , n-1);
    gasket ( x13 , y13 ,  x23 ; y23 ,  x3, y3 , n-1);
}
else { triangle (x1 , y1 , x2, y2, x3, y3) ;}
```

```
GL_Begin(GL_Tri
    angles)
gasket( . . . )
glEnd(), glFlush()
```

C-Curve

primitive is line . replace a line by two shorter, equal length line at 90° angle.

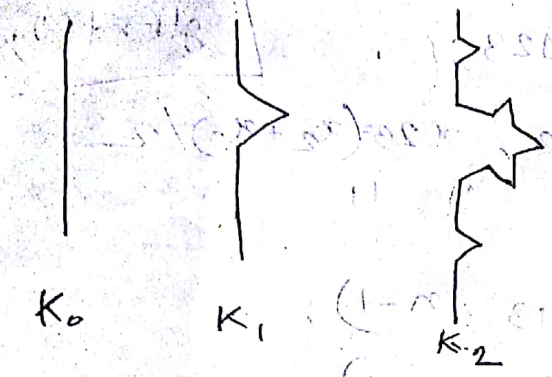



```
line()
    glVertex2f( ) . . .
c-curve(x, y, len , alpa , n)
    if (n>0)
        len = len/√2
        c-curve (x, y, len , alpha+45 , n-1)
        x = x + len * cos (alpa+45) ;
        y2 = y + len * Sin (          ) ;
        c-curve ( x, y , len , alpha-45, n-1)
else { line( x, y, x+ len * cos(alpha) , y + len * sin(alpha));
```

```
GL_Linea
c-curve(x, y, len,
    alpa, n)
```

Koch Curve

primitive is line, divide a line into three equal segments and replace middle segment with two lines of same length.



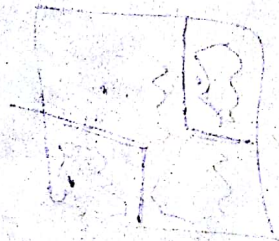
gl - Lines

```
line( )
Koch curve(. .)
  if (n>0)
      len = len/3
      K-cv ( x, y, len, alpha, n-1)
      x ± len * cos(alpha)
      y ±  len * sin( " )
      K-curve(x, y, len, alpha-120, n-1)
      x ± len * cos (alpha - 120).
      y ± len * sin (alpha - 120)
      K-curve(x, y, len, alpha+120, n-1)
      x ± len * cos(alpha+120)
      y ± len * sin(alpha+120)
      K-curve( x, y, len, alpha, n-1)

else { line(x, y, x+len*cos(alpha), y+len*sin(alpha));
```

In Boundary fill, it fills pixels with a color if it is not boundary. If it doesn't reach to boundary, it recursively fill with color every pixel.

In flood fill, a seed checks if a pixel has its regions color. If no; it fills with color, otherwise it simply skips.



First

CSE-433 – SE

C.T.-2 – Ch – 14

* Basic –

Testing, Verification, Validation, Debugging

* Testing definition

Module/Unit test

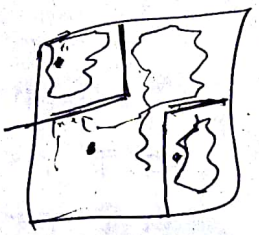
Integration test

Function test

System test

Acceptance test

Installation test



* Testability

Operability – operates cleanly

Observability – results of each test case are readily observed.

Controllability – degree to which testing can be optimized..

Decomposability – testing can be targeted.

Simplicity – reduce complex architecture and logic to simplify test

Stability – few changes are requested

Understandability

* 'Good Test'

(I) has a high probability of finding an error

(II) is not redundant

(III) should be 'best of breed'

(IV) should be neither too simple nor too complex.

Rizvi

CSE-429- DSP
Quiz + Viva

* VHDL — VHSIC Hardware Description Language

VHSIC — Very High Speed Integrated Circuits

* VHDL is coding model for —
  - requirement specification
  - documentation
  - testing using simulation
  - formal verification
  - synthesis
  - class assignments

* VHDL is a programming language — allows to model & develop complex digital systems — allows to designate in/out ports and specify behaviour

* 3 ways —

① Data flow — statements that defines actual flow of data

such as x <= y

entity declaration — describes the input/output ports of a module

Ex:- entity name is
port (d0 : in bit
d1 : out bit);
end entity name;

② Behavioral — architecture body → describes an implementation of an entity

Behavioral architecture → describes algorithm performed by module, contains — process statements

Ex:- architecture behav of reg4 is
begin
process ( )
begin
end process ; end behav.

(iii) Structural — implements module as composition of subsystem

contains —

signal declaration

component instances

port mapping

* Test Bench — testing a design by simulation, apply test sequences to input

* VHDL design begins with ENTITY block — describes the interface for the design.

ARCHITECTURE block — describes internal operation of the design.

SIMULATION — type of test to see if the basic logic works

SYNTESIS — allows timing factors to influence.

* Verilog vs VHDL

for modeling logic gates	for modeling High level hardware

(i) Verilog has simple data types and all are predefined.

(ii) " does not have library management

(iii) VHDL is strongly typed language → written with matched and defined data type

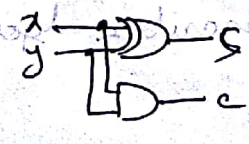
Verilog " loosely " "

(iv) Verilog is case-sensitive, VHDL is not.

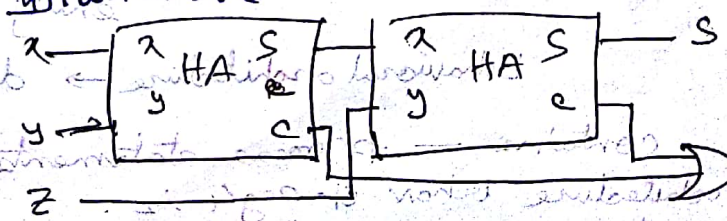
* Half adder

S ⇐ x xor y

C ⇐ x and y



* Full adder



S ⇐ x xor y xor z ; C ⇐ (x & y) or (y & z) or (z & x)

Flip-Flop

J	K	Q(t+1)
0	0	Q(t) – no change
0	1	0 – Reset
1	0	1 – Set
1	1	$\overline{Q}(t)$ – toggle

D	Q(t+1)
0	0 Reset
1	1 Set

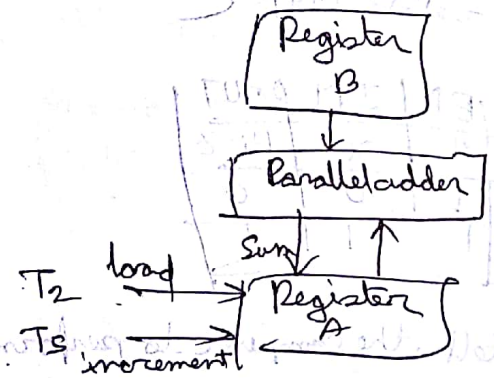
T	Q(t+1)
0	Q(t) – no change
1	$\overline{Q}(t)$ – toggle

Flip flop design — Behavioral → easy that way

process define — how algo. works, input CLK, RESET

Why temp — if output is given inside process, value will be changed frequently. That's why it is declared outside of process using temp.

Add & increment op.



ALU Design

$x_i = A_i + S_2 S_1' S_0' B_i + S_2 S_1 S_0 B_i'$

$y_i = S_0 B_i + S_1 B_i'$

$z_i = S_2' C_i$

Status register

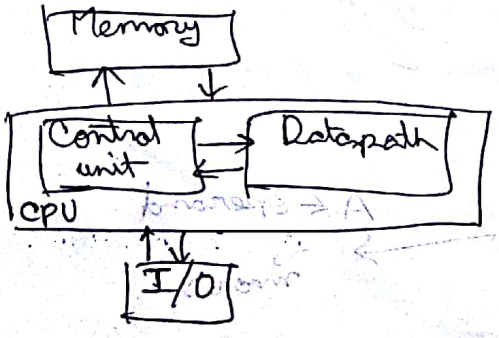
overflow – V = C_8 xor C_9

carry – C = C_9

zero – Z = $(C_1 \text{ or } C_2 \ldots C_8)'$

sign – S = C_8

Digital Computer



Modular approach

system is partitioned into modular subsystem, each performing specific task

3 basic component of R.T.

(i) set of registers

(ii) operations to be performed

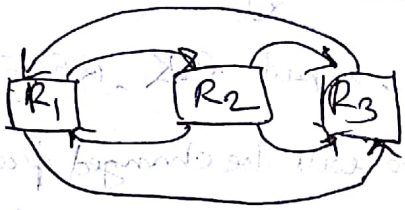
(iii) control to supervise sequence of operation

RTL/HDL contains statements, control functions, list of micro-op.

Commonly used Micro-op.

① Inter-register transfer
② arithmetic
③ Logic
④ Shift

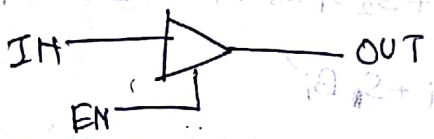
Bus transfer



Memory transfer

R: MBR ← M W: M ← MBR

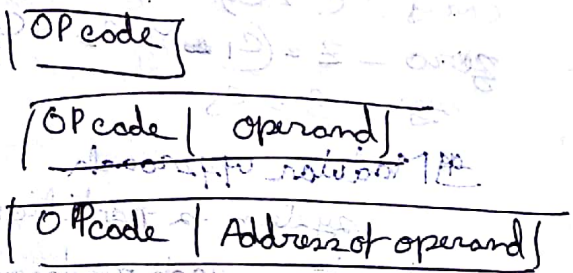
Method for constructing bus → three-state buffer



EN	IN	OUT
0	X	Hi-Z
1	0	0
1	1	1

Instruction code – group of bits that tell the computer to perform a specific operation.

format —



Micro op vs Macro op

hardware. requires only one control function

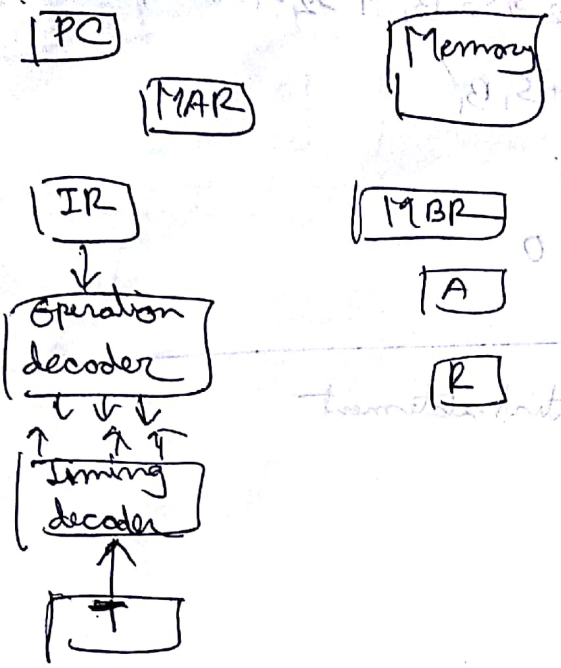
requires more than one

A ← R

A ← operand

macro

Simple computer



- MOV — 00 000001
- LDI OPRD — 00000010
- LDA ADRS — 00000011

Timing variables used are control function

Process — datapath

control unit — specifies the operation to be performed.

CPU — combination of processor and control unit

ALU — combinational circuit able to perform arithmetic + logic op.

Scratchpad memory

register in a processor is enclosed within a small memory unit, called S.M.

Adv. Bus system is cheaper.

Difference between S.M and BUS system

memory is selected by means of address

~~Info~~ transfer is selected by multiplexers.

Accumulator Register — used to perform serial addition

Arithmetic unit

$X_i = A_i$, $Y_i = S_0 B_i + S_1 B_i'$

operation	
S_1 S_0 C_{in}	Transfer
	Increment A
	A + B
	A + B + 1
	A + $\bar{B}$
	A + $\bar{B}$ + 1
	Decrement
	Transfer

Logic unit

S_1	S_0	
0	0	OR
0	1	XOR
1	0	AND
1	1	NOT

Finally,

$X_i = A_i + S_2 S_1' S_0' B_i + S_2 S_1 S_0 B_i'$

$Y_i = S_0 B_i + S_1 B_i'$

$Z_i = S_2' C_i$

for arithmetic, $S_2 = 0$

Process — function containing sequential statement

Syntax –

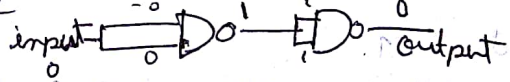
```
process (CLK, RESET)
  begin
  end process.
```

Possible to design computer without ALU ?

Yes, possible. But no arithmetic operation will be available.
Operation like – MOV, Shift, LDI, LDA will be done

Control unit — selects the microop. to be executed.

Buffer construction from ① NAND gate



XOR gate can act as ① Buffer → $X\overset{(A)}{OR} = A\bar{B} + \bar{A}B$, now, if $B=0$, $X\overset{(A)}{OR} = A$

② Inverter → $X\overset{(A)}{OR} = A\bar{B} + \bar{A}B$, now, if $B=1$, $X\overset{(A)}{OR} = \bar{A}$

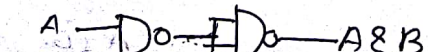
Universal gate – NAND, NOR

can be used to get any other gate

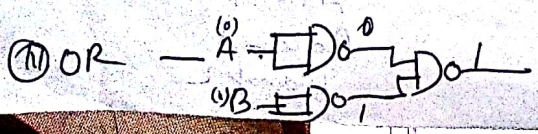
NAND

① inverter

② AND –



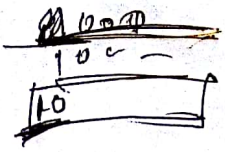
③ OR –



$A\bar{B}+\bar{A}B$

1 0 0 1

00 — 0
11 — 0
01 — 1
10 — 1



Overflow, V — signed
Carry, C — unsigned

$$\begin{array}{r} 0\ 0010100 \\ 0\ 0010100 \\ \hline 0\ 0101000 \end{array}$$

35
40

* Five framework –

① communication
② Planning
③ Modeling
④ Construction
⑤ Deployment

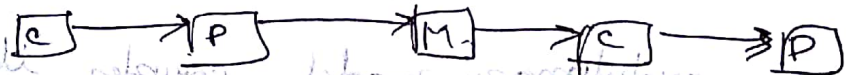
* Process is a collection of work activities, actions and tasks
* Process flow – describe how the framework activities and the actions, tasks occur within each framework.
* liner process flow – executes each of five in sequence

iterative " " – repeats one or more before proceeding to next

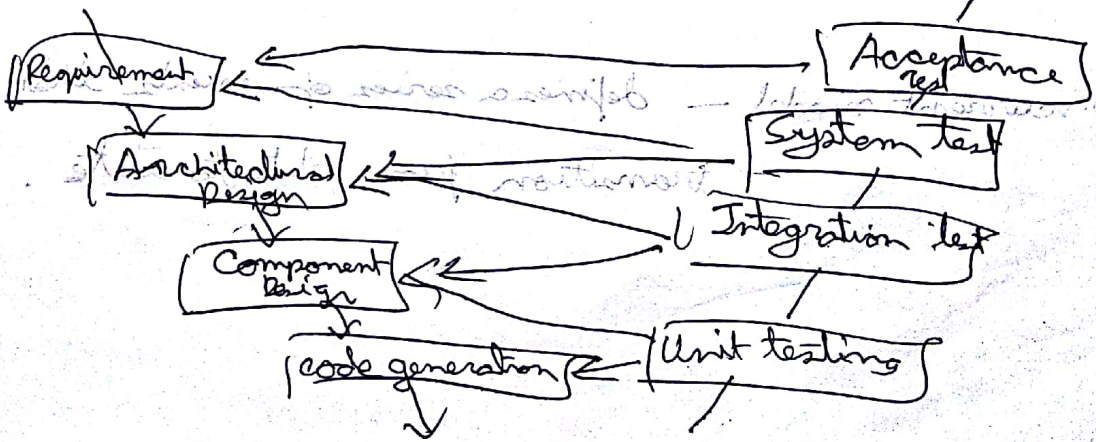
evolutionary " " – executes in circular manner.

Parallel " " – executes one or more activities in parallel.

* Waterfall model –



Variation of waterfall model is V model :– provides how verification validation actions are applied.



Incremental process model – provide a set of functionality to users quickly.

combines linear & parallel process.

First increment → often a core product.

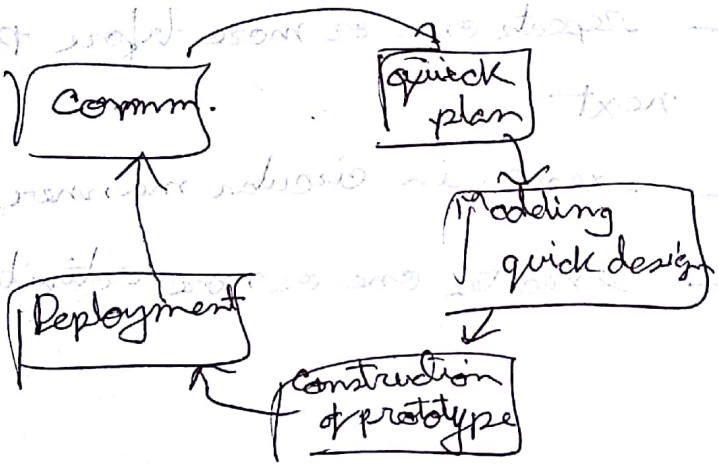
* Evolutionary process model.

accomodate a product that grows and changes.

iterative.

Prototyping paradigm.

Prototype serves as a mechanism for identifying requirements.



Spiral model — evolutionary model, couples iterative nature, rapid development.

During early iterations the release might be a model/prototype.

* Concurrent model — defines a series of events that trigger transition from state to state.

1304045

1304003

Exp-1: Learning VHDL through Combinational Circuit Design

Objectives:

The objectives of this laboratory are:

1. Learn the basic programming style of VHDL module and VHDL TestBench
2. Design a Full-Adder using three types of Architecture of VHDL
3. Design a 4-bit Parallel Adder.

Required circuits diagrams

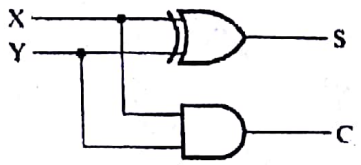


Fig 1. Logic Diagram of Half-adder

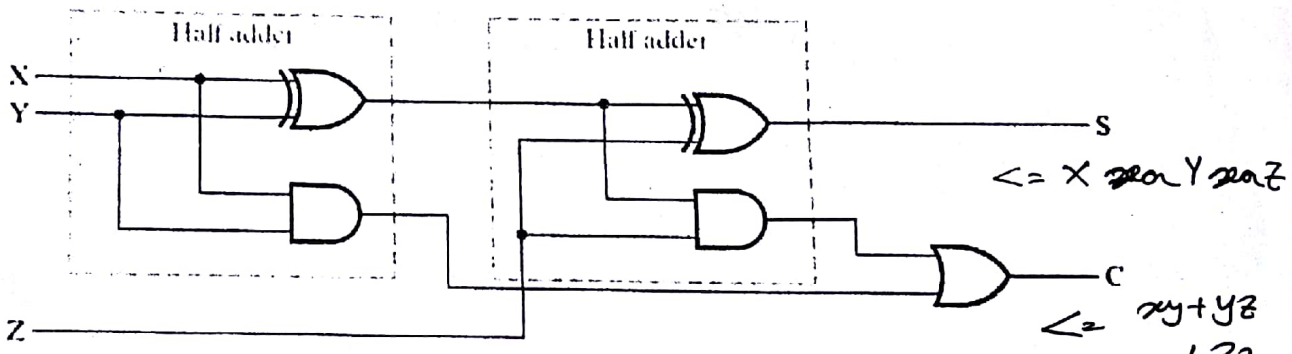


Fig 2. Logic Diagram of Full-adder

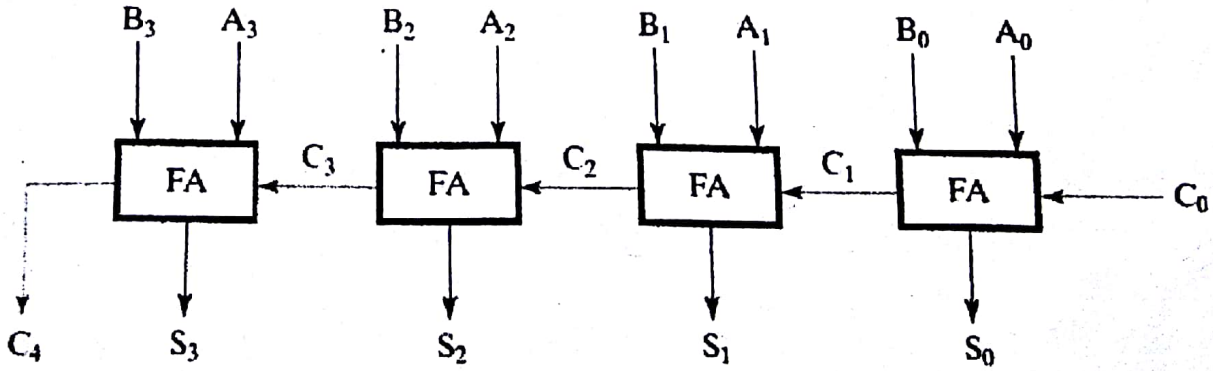


Fig 3. Logic Diagram of 4-bit parallel-adder

Scanned by CamScanner

$xy + yz + zx$

$xy + (xyz + x'yz + xyz + xy'z$

$xy + xyz + x'yz + xy'z$

$xy + z(xy + x'y + xy')$

$(xy + x \oplus y)$

xyz

130405

Exp-2: Learning VHDL through Sequential Circuit Design

Objectives:

The objectives of this laboratory are:

1. Design a JK Flip-flop T Flip-flop and D Flip-flop in VHDL
2. Design a 4-bit register and a 4-bit counter in VHDL
3. Design a 4-bit shift-register in VHDL

Required diagrams

JK Flip-Flop

J	*K*	*Q*(*t* + 1)	
0	0	*Q*(*t*)	No change
0	1	0	Reset
1	0	1	Set
1	1	*Q*′(*t*)	Complement

D Flip-Flop

D	*Q*(*t* + 1)	
0	0	Reset
1	1	Set

T Flip-Flop

T	*Q*(*t* + 1)	
0	*Q*(*t*)	No change
1	*Q*′(*t*)	Complement

Fig. 1 Characteristic Tables of FFs

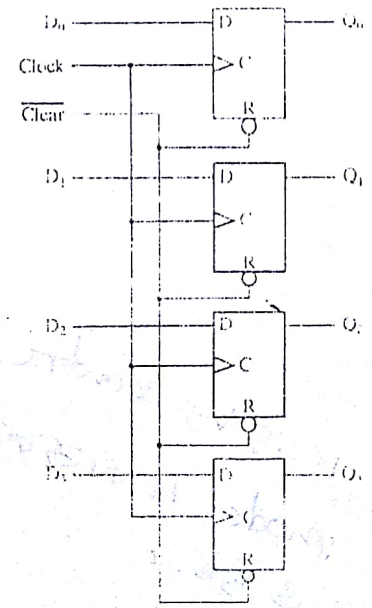


Fig 2. Logic Diagram of a 4-bit register

Reg Vs contr

model কি কাজ করেছে

S √sb.

:=

Scanned by CamScanner

Exp-5

Design and Implementation of an 4-bit Arithmetic Circuit

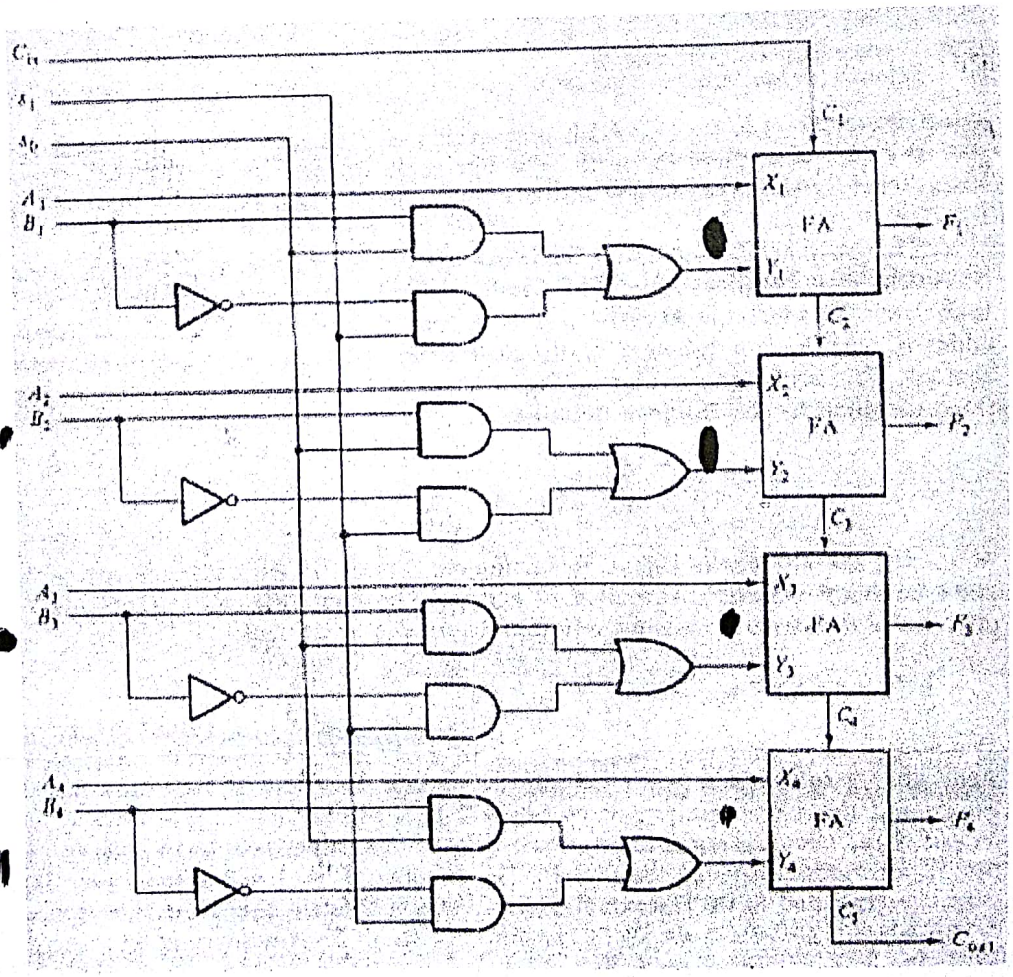
Equipments

- DC-Power supply,
- Bread-board,
- LEDs,
- Switches
- AND gates IC
- OR gates IC
- Not gates IC
- Full adder IC (74LS83),

Function Table

Function select			Y equals	Output equals	Function
s_1	s_0	C_{in}			
0	0	0	0	$F = A$	Transfer A
0	0	1	0	$F = A + 1$	Increment A
0	1	0	B	$F = A + B$	Add B to A
0	1	1	B	$F = A + B + 1$	Add B to A plus 1
1	0	0	$\bar{B}$	$F = A + \bar{B}$	Add 1's complement of B to A
1	0	1	$\bar{B}$	$F = A + \bar{B} + 1$	Add 2's complement of B to A
1	1	0	All 1's	$F = A - 1$	Decrement A
1	1	1	All 1's	$F = A$	Transfer A

Circuit Diagram



Exp-6.

Design and Implementation of an Arithmetic Logic Unit (ALU) in VHDL

Objectives.

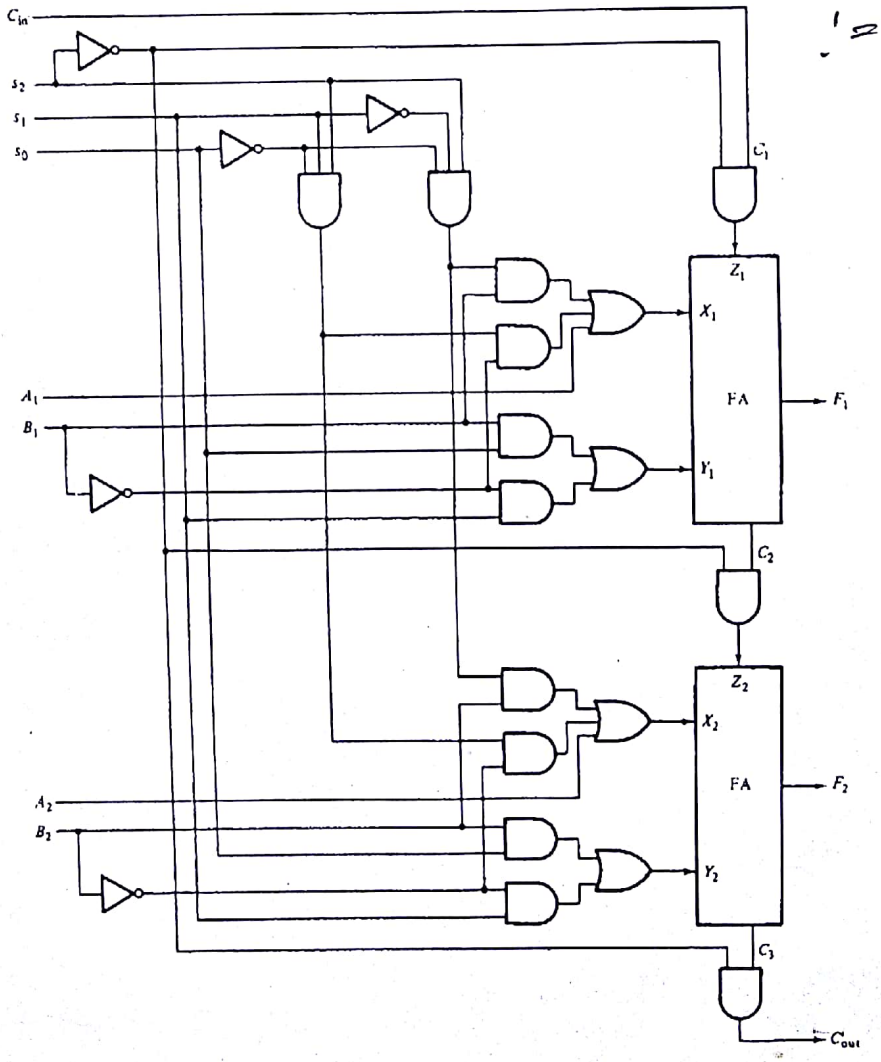
1. Design an Arithmetic circuit and then modify it to design an efficient 4-bit ALU.
2. Implement the ALU in VHDL and observe various outputs and inputs
3. Incorporate a status register to the ALU.

portmap'
=> case
<= assignment operator.
↳ signal.
! = variable.

The ALU design

$$X_i = A_i + s_2 s_1' s_0' B_i + s_2 s_1 s_0' B_i'$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$Z_i = s_2' C_i$$

Circuit Diagram (2-bit ALU)



Exp-7

Design and Implementation of a 4-bit Combinational Shift Unit

Equipments

- DC-Power supply,
- Bread-board,
- LEDs,
- Switches
- 4x1 MUX

Circuit Diagram