Unit 1

Introduction to Embedded System

Embedded System:

- An embedded system is one that has computer hardware with software embedded in it as one of its most important component.
- An embedded system has three main components: Hardware, RTOS (Real time operating System) and Application software

An embedded system is a microprocessor or microcontroller based system that is built to control a function or range of functions and is not designed to be programmed by the end user in the same way that a PC is.

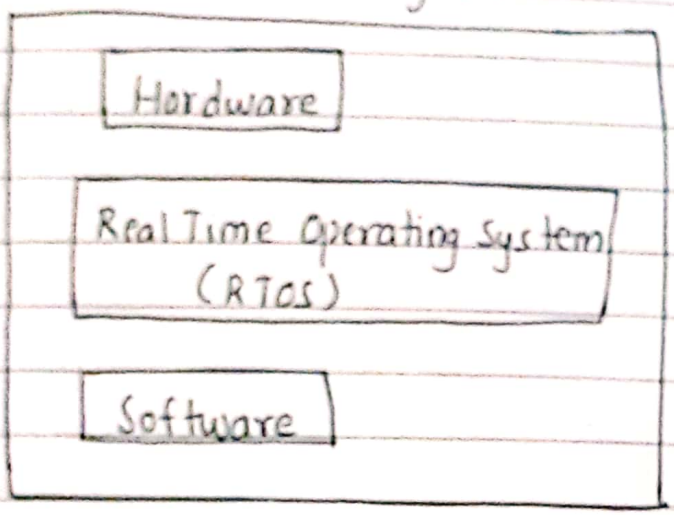
The various definitions of an embedded system can be expressed as a device which is:

i) designed to perform a dedicated function
ii) incorporated or ~~embed~~ implanted within a larger system
iii) A computing system with tightly coupled hardware/software
iv) Used to control, monitor or assist in the operation of a physical system
v) Microprocessor or microcontroller based, software driven, reliable, real time, autonomous or network or human controlled entity

Microprocessor	Microcontroller
1. The functional blocks are ALU, Registers, Timing and Control Unit.	1. It has functional blocks of microprocessor and additionally timer, RAM, paraller I/O, EPROM, ADC and DAC
2. Bit handling instruction is less, one or two types.	2- Many types of bit handling instructions are available
3. Rapid movement of code and data between external memory and microprocessor.	3. Rapid movement of code and data with microprocessor
4. Generally used for designing general purpose digital computer systems.	4. Used for designing application specific dedicated systems.

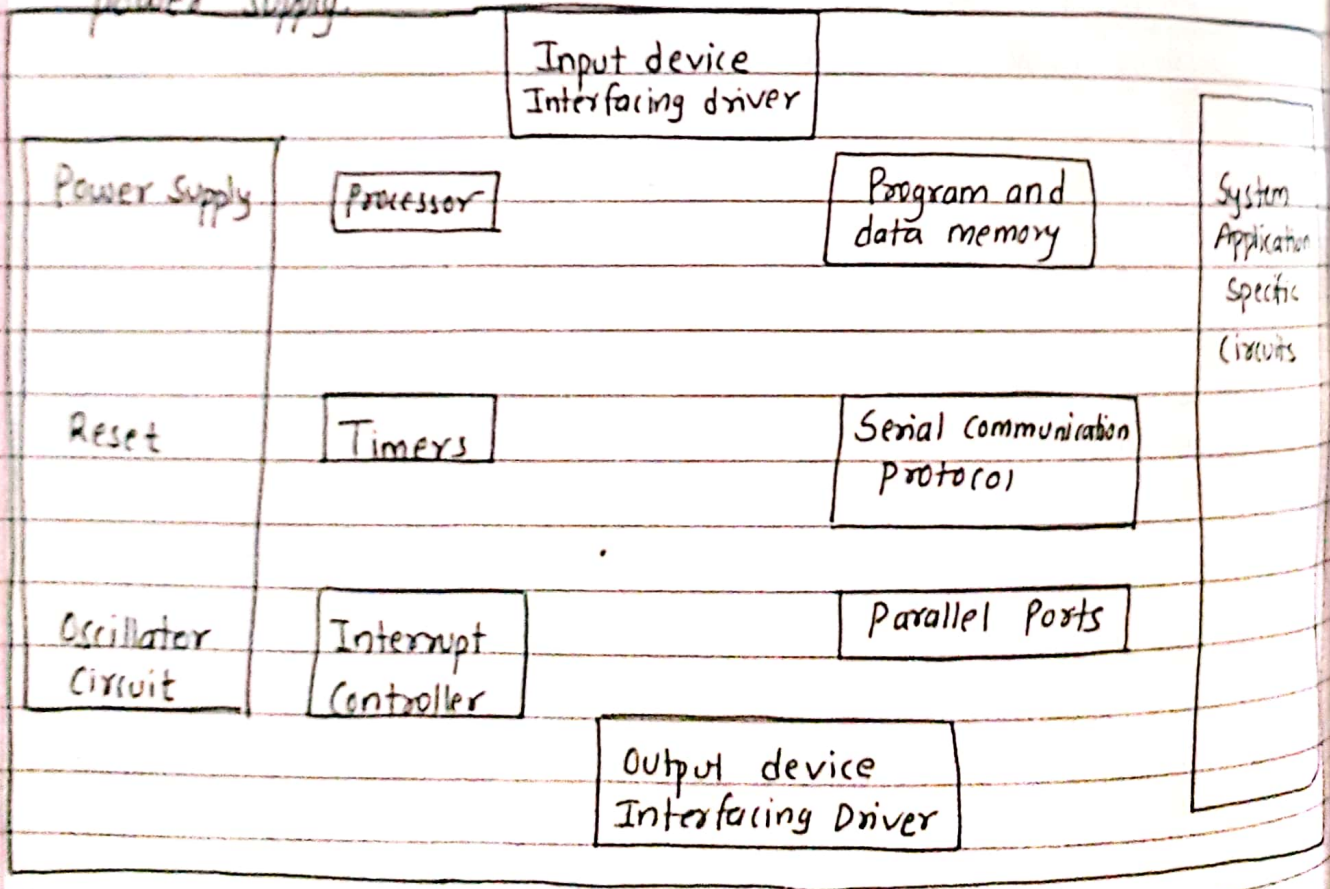
Application of Embedded system:

Components of Embedded System



Hardware:

It consist of microprocessor or microcontroller, timers, interrupt controller, program and data memory, serial ports, parallel ports, input devices, interfaces, output device and power supply.



RTOS

- It supervise the application software and provides a mechanism for the processor to run a scheduled process and does the context switch bet^n various processes.
- The RTOS defines the way in which the Embedded System work. It organizes access to a resource consisting of a series of ~~the~~ task in sequence and schedules their execution by following a plan to control the latencies and to meet the deadline.

Application Software

- It can perform a single task or concurrently perform a series of task or multiple task.
- Software is designed to keep in view 3 constraints:
  i) Available system memory
  ii) Available processor speed
  iii) Need to limit power dissipation when running a system in continuous cycles of events, run, stop and wakeup

Img Characteristics of Embedded System

Embedded Computing is much more demanding than the general computing system. Functionality is important in both the system but embedded computing systems must meet a number of constraints, more than that for general computing systems. Embedded computing systems also have to provide the ~~short~~ sophisticated functionalities such as :-

i) Complex algorithm

The operations performed by the microprocessor may be very sophisticated. Eg: the microprocessor that controls an automobile engine must perform complicated filtering functions to optimize the performance of the car while minimizing pollution and fuel utilization.

ii) User Interface

Microprocessors are frequently used to control complex user interfaces that may ~~input~~ include multiple menu and many options. The moving map in GPS (Global Positioning System), navigation are good examples of sophisticated User Interfaces

iii) Real Time

Many embedded computing system have to operate in real time. If the data is not ready by the certain deadline, the system breaks. Faliure to meet deadline is ~~so~~ unsafe in some cases. (like Antilock braking system (ABS), pacemaker) or can cause dissatisfaction to customers & Eg: missed deadlines in printers can result in scrambled pages.

iv) Multi Rate

Many embedded computing system have several real time activities going on at the same time. The may simultaneously control some operation that run at slow rates and others that run at high rates. Eg: Multimedia application constituting audio and video portions are prime examples of multirate behaviour. They may remain closely synchronized. Faliure to meet a deadline ~~if~~ on either

the audio or video portion spoils the perception of the entire presentation.

There are a number of parameters that affect the design and development of embedded system. The unique characteristics of ES (Embedded system) that distinguish such systems from other computing system can be given as:-

i) Single functioned / specific
An ES usually executes a specific program repeatedly.
Eg: A pager is always a pager, and a pacemaker is always used to regulate the heart rate.
In contrast, a PC executes a variety of programs in the form of various application software programs.

ii) Tightly Constrained:
All computing systems have constraints on design ~~matrix~~ metric, but those on ES can be specially tight. ~~Desired matrix~~. Design metric is the measure of implementation feature such as cost, size, performance and power.

iii) Reactive and real time
Many ES must continually react to changes in the systems environment and must compute certain results in real time without delay. Eg: a pace maker fitted within the human body near chest must send the electrical pulses to the heart at the correct time. It continually monitors and react to changes in the systems environment. (contraction and dilation of heart muscles) and must compute

certain results in real time without delay.

A Digital Camera

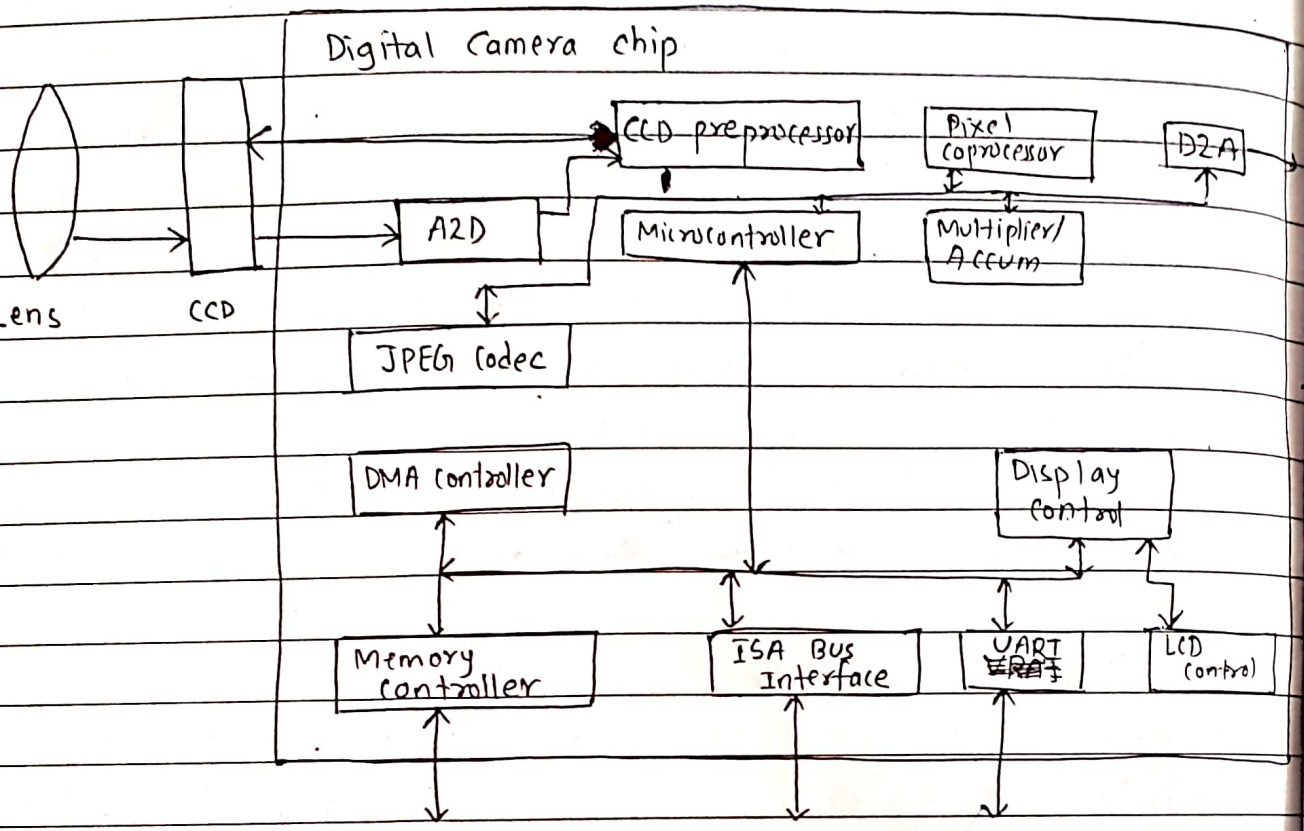


fig: Internal structure of digital camera

The description of the internal structure of a digital camera is presented in the following points:

i) CCD (Charged Coupled Device) contains an array of light sensitive photocells that capture an image

ii) The A2D and D2A circuits convert analog images to digital and digital to analog respectively.

iii) CCD preprocessor provides commands to the CCD to read the image.

iv) The JPEG codec compresses and decompresses an image using the JPEG compression standard, enabling compact storage of images in the limited memory of the camera.

v) The pixel coprocessor aids in rapidly displaying images.

vi) The memory controller controls access to a memory chip, also found in the camera.

vii) The DMA controller enables direct memory access by other device while the microcontroller is performing other functions.

viii) UART enables common with a PC's (personal cmp) serial port for uploading video frames

ix) ISA Bus Interface enables a faster connection with a PC's ISA. bus.
→ (Industrial Standard Architecture)

x) The LCD control and display control circuits control the display of images on the camera's Liquid Crystal Display clear.

xi) The multiplier/accumulator circuits performs a particular frequently executed multiply/accumulate computation faster than the microcontroller itself, would.

xii) Microcontroller: Heart of the System
It is a programmable processor that controls the activities of all the other circuits.

Classification of ES: (Assignment)

A. Based on Complexity and Performance Requirement
(Small scale, Medium, Large ..)

B. Based on Generation
(first gen, second, third)

C. Based on deterministic behaviour

D. Based on triggering
(Level and edge triggering)

Design Metric and Design Challanges

Design Metric

The complexity of embedded system is in the increasing trend to meet the market requirements. Also the technological advancement and growth aids to the development of complex embedded systems.

	Metric	Description
1.	NRE Cost (Non Recurring cost)	Non Recurring Engineering Cost: The initial cost of the designing the system. (one time cost)
2.	Unit Cost	Manufacturing cost of each copy of the system
3.	Size	Physical space Required

4.	Performance	Instruction execution time or throughput of the system
5.	Power	Amount of power consumed by the system
6.	Flexibility	Ability to change the functionality of the system.
7.	Time to prototype	Time needed to build a working version of the system.
8.	Time to market	Time required to develop a system before releasing to the customers.
9.	Maintainability	Ability to modify the system after its initial release.
10.	Correctness	Correct implementation of system functionality.
11.	Safety	Probability that the system will not cause harm.

Design Challanges:

Optimization is defined as the task of making the design metrics values best possible. Optimization of these design metric is a major design challange because often the design metric compete with one another. This means that improving one may affect the other.

For Eg: If an implementation size is reduced, performance of the system may suffer. To meet the optimization challange, the designer must be able to switch to the appropriate hardware and software technologies in order to achieve the best implementation for a given constraints and applications.

Processor Technology

i) General Purpose Computing System (software)

The designer of a general purpose processor or microprocessor builds a programmable device that is suitable for variety of applications to maximize the number of devices ~~sold.~~ sold.

Features:

i) Program Memory

The designer of such a processor doesnot know what program will run on the processor, so the program cannot be built into the digital circuit.

ii) General Data Path

The datapath must be general enough to handle a variety of computations, so, such a datapath typically has a larger register file and one or more general purpose ALU's.

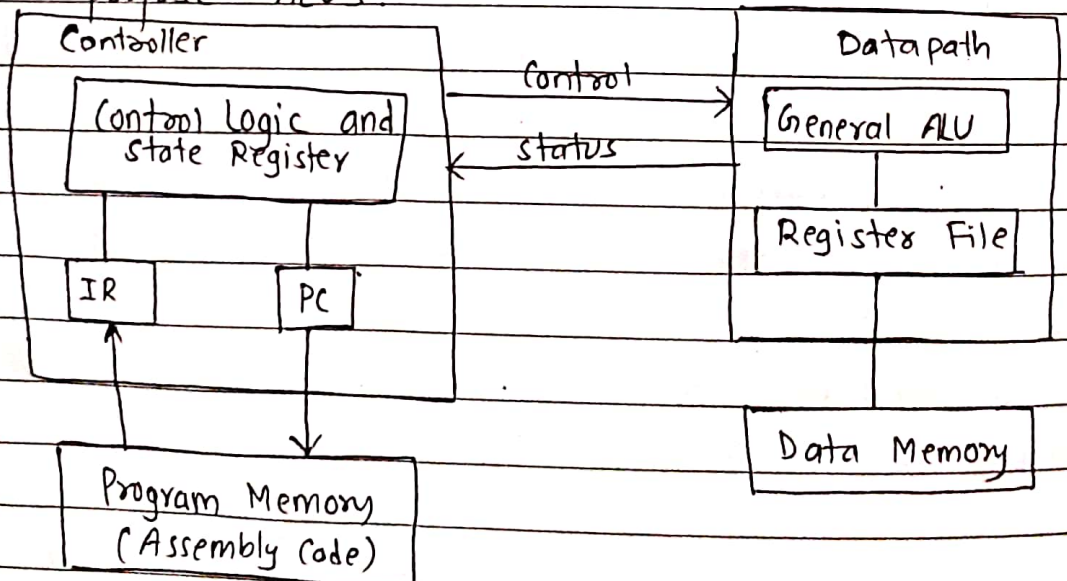
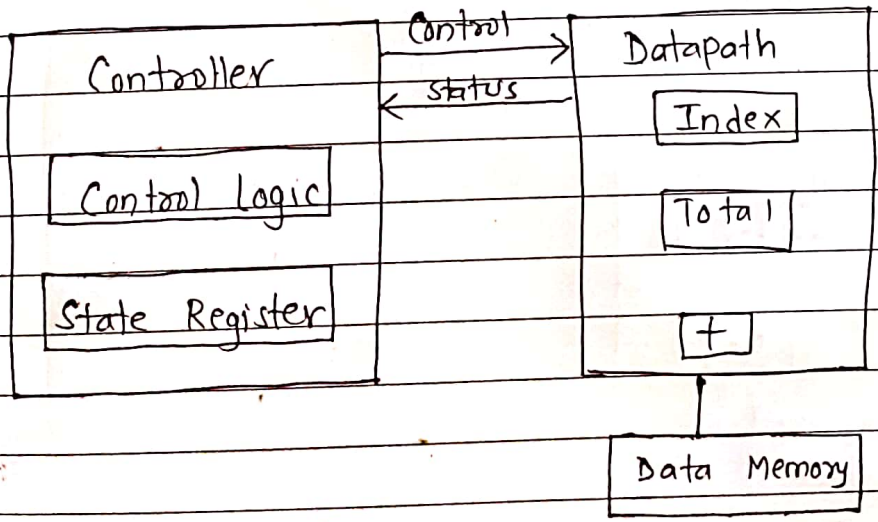


fig: General Purpose Processor

- The functionality is stored in a program memory.
- The controller fetches the current instruction as indicated by the PC into the IR.
- It then configures the data path according to this instruction and executes the instruction.
- It then determines the next instruction address, sets PC to this address and fetches again.

ii) Single Purpose Processor (Hardware)

A single purpose processor is a digital circuit designed to execute exactly one program. All the components other than the micro controller are single purpose processor. An embedded system designer may create a single purpose processor by designing a custom digital circuit.



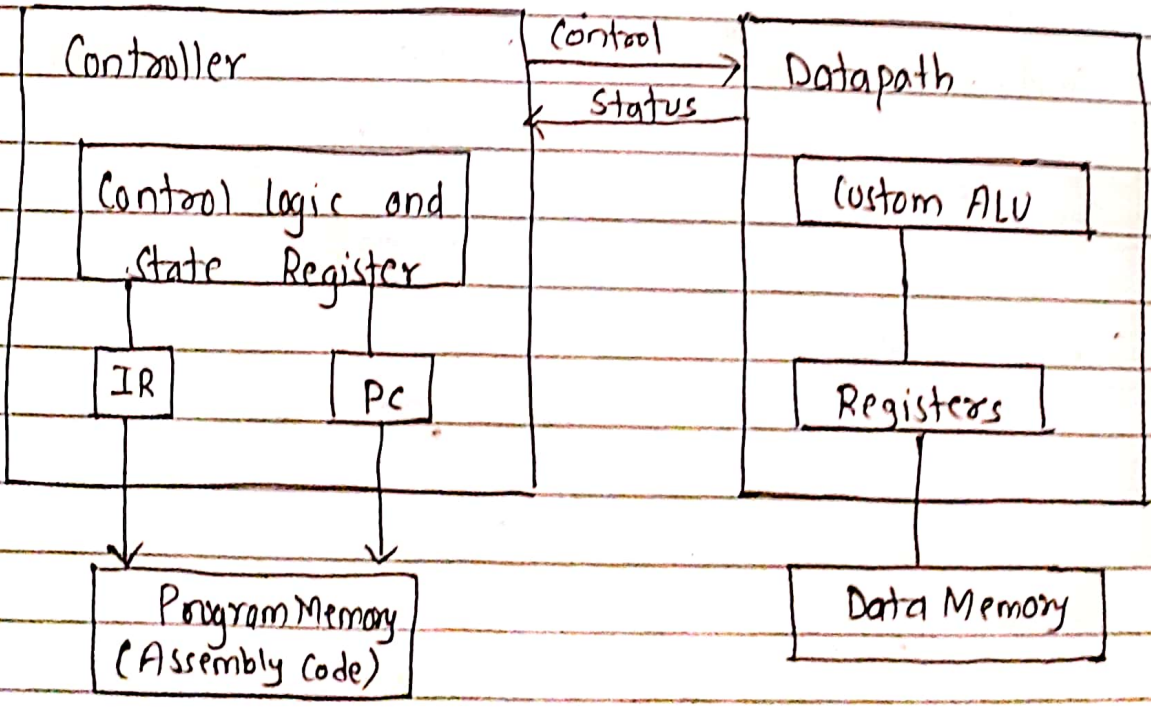
- The datapath may be customized, it may have an auto increasing register, a path that allows us to add a register with a memory location in one instructor fewer register

and the simple controller.

- Represents an exact fit of the desired functionality, no-thing more, nothing less
- Contains only the essential components for this program: two registers and one adder.
- We hardwired the program instructions directly into the control logic and use a state register to step through these instructions, so no program memory is necessary.

iii) Application specific Processor

This processor can solve as a compromize between the other processor options such as SPP and GPP. Application specific processor optimized for a particular class of applications having common characteristics such as embedded control, digital signal processing or tele communication



Chapter 2

NOTE:

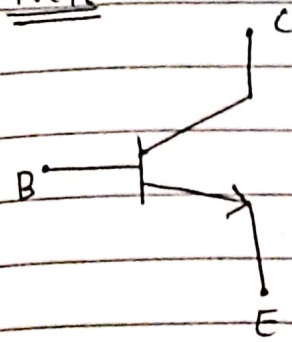


fig: NPN BJT

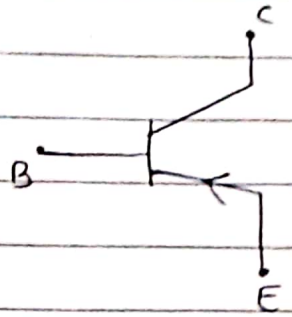


fig: PNP BJT

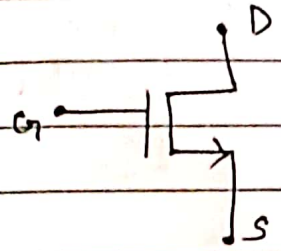


fig: NMOS

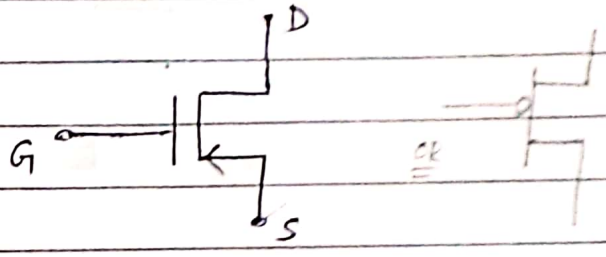


fig: PMOS

Combinational Circuit

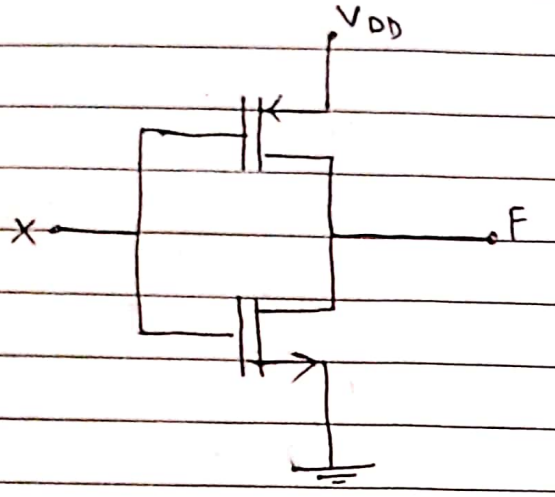
A combinational circuit is a digital circuit whose output is purely a function of its present inputs. Such type of circuit has no memory of past inputs and can be designed using basic logic gates

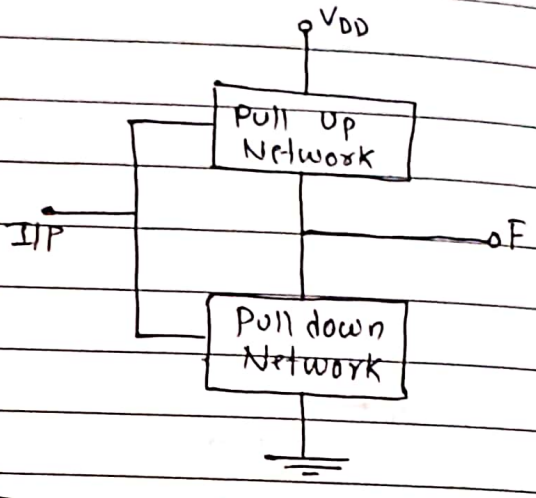
BiMOS / BiCMOS:

These devices have the advantages of both the MOSFET and Bipolar transistor. The high input impedance and low power operation of MOSFETs and very high frequency operation and current driving capability of bipolar transistor.

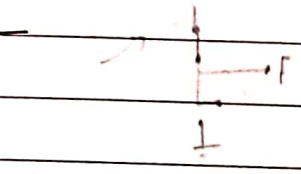
CMOS logic:

i) As inverter:

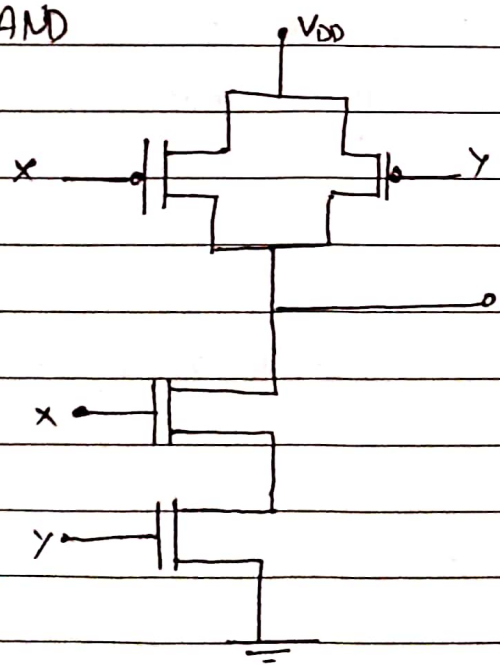




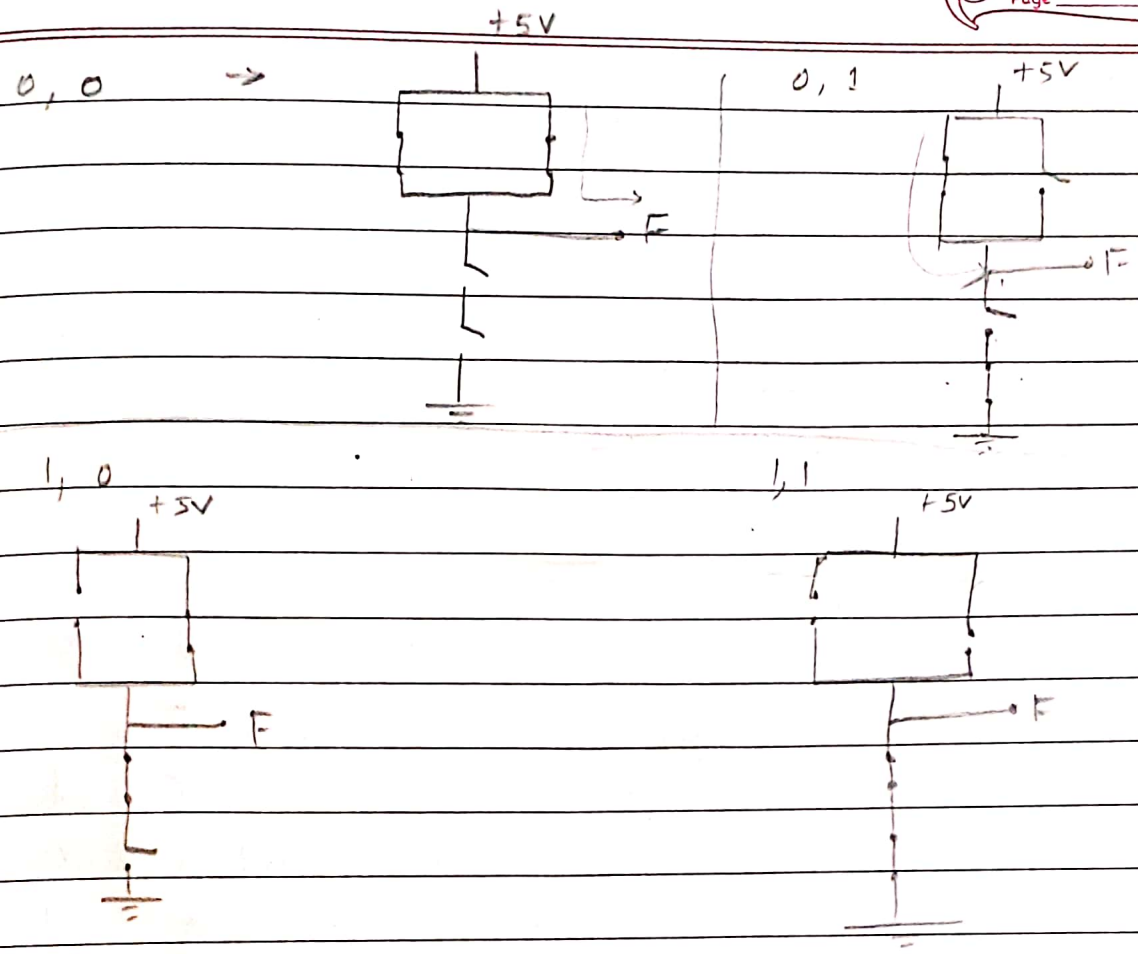
X	F
0	1
1	0



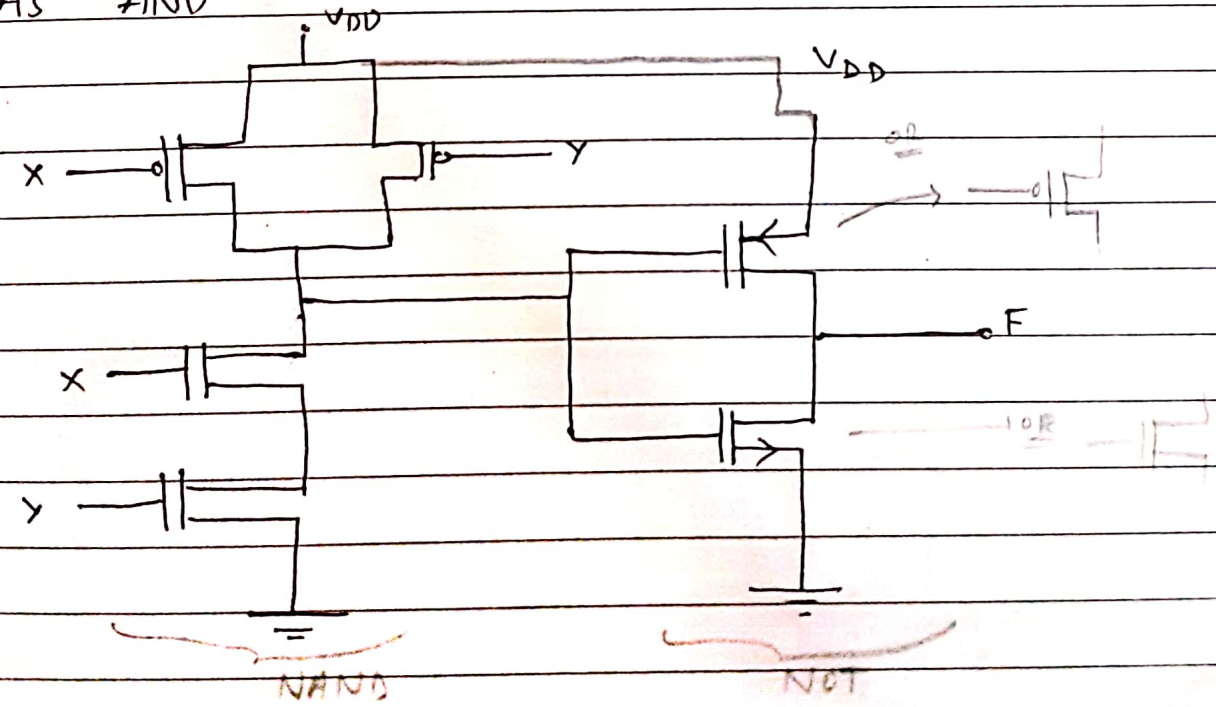
ii) As NAND



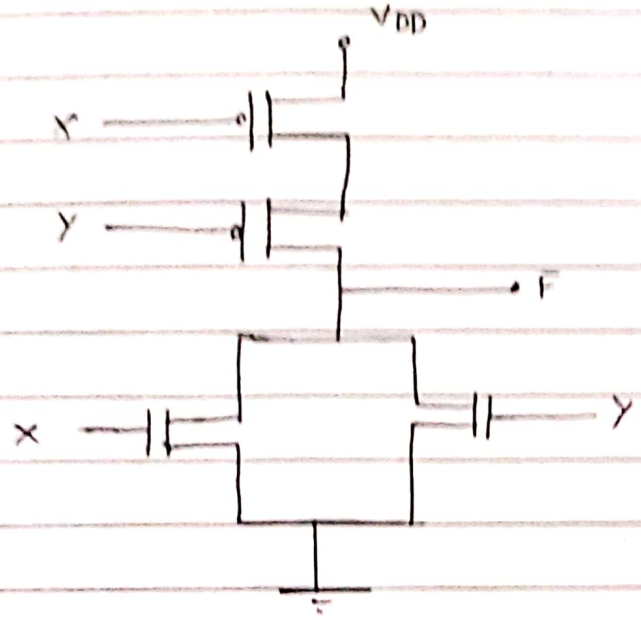
X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0



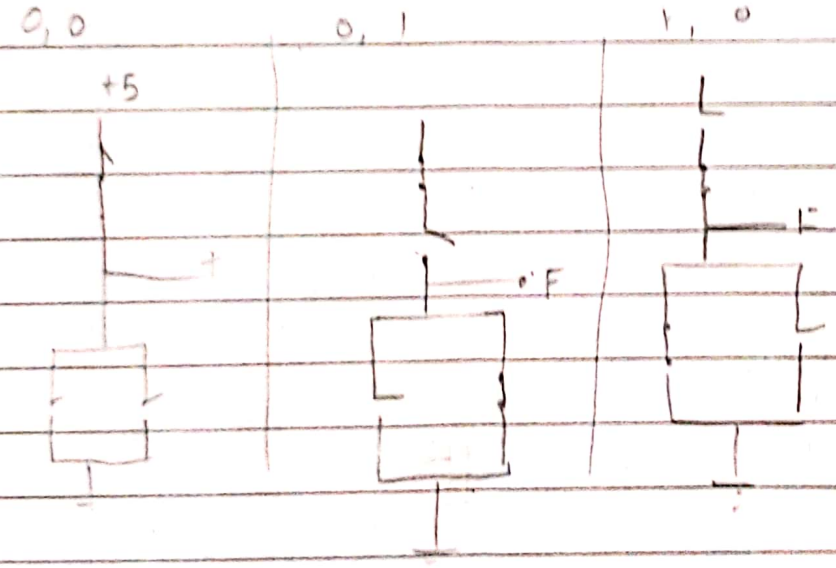
iii) As AND



iv) As NOR



X	Y	F
0	0	1
0	1	0
1	0	0
1	1	0



Combinational Logic Design:

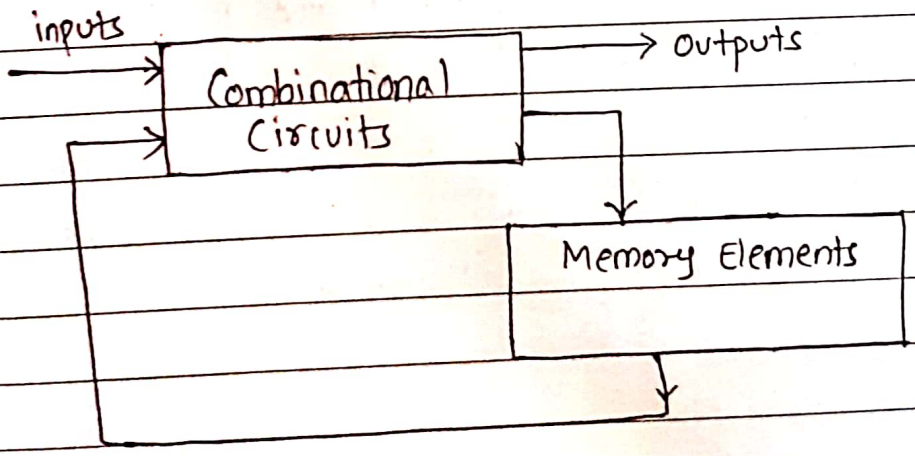
(RT → Register transfer)

RT level components:

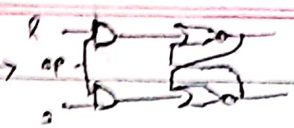
Logic circuit could be very complex. For eg: a circuit with 16 inputs would have 2^{16} or 64K rows in the truth table. To reduce such complexity, combinational components often called RT level components are used. Some of these RTlevel components are: MUX, Encoder, Decoder etc.

Sequential Circuit

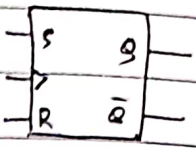
A logic circuit whose present output depends upon present input and also on previous output and input is known as sequential circuit. It is the circuit with feedback. They have the memory storage elements to store the previous output values.

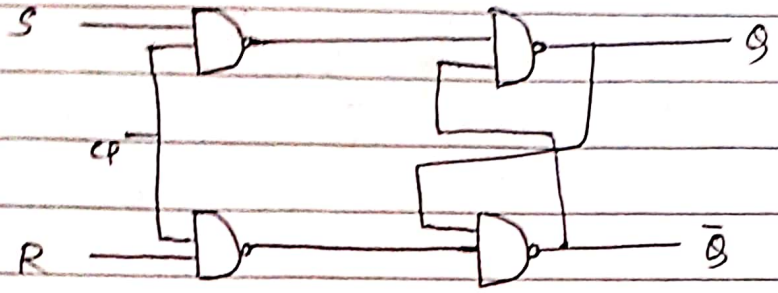


Flipflop

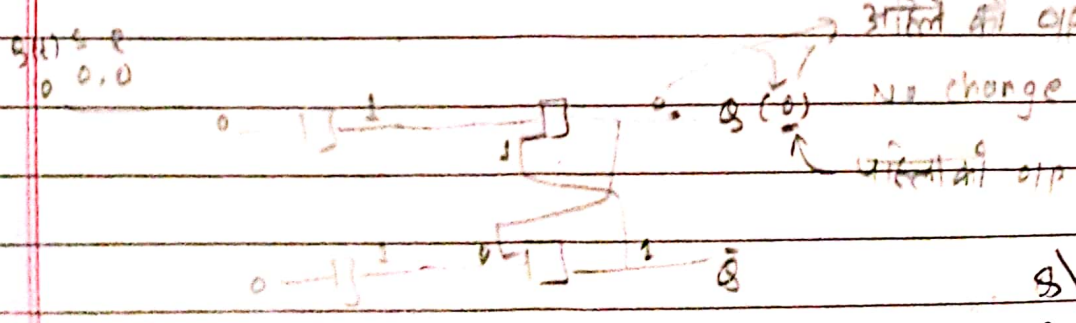


1. SR flipflop





$Q(t)$	S	R	$Q(t+1)$	Remarks
0	0	0	0	No change
0	0	1	0	Reset
0	1	0	1	Set
0	1	1	(x) Invalid	
1	0	0	1	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	(x) Invalid	

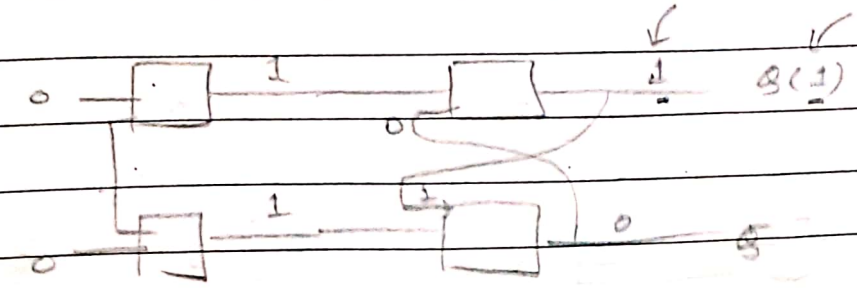


S \ SR	00	01	11	10
0	0	0	x	1
1	1	0	x	0

$$Q(t+1) = S + Q\bar{R}$$

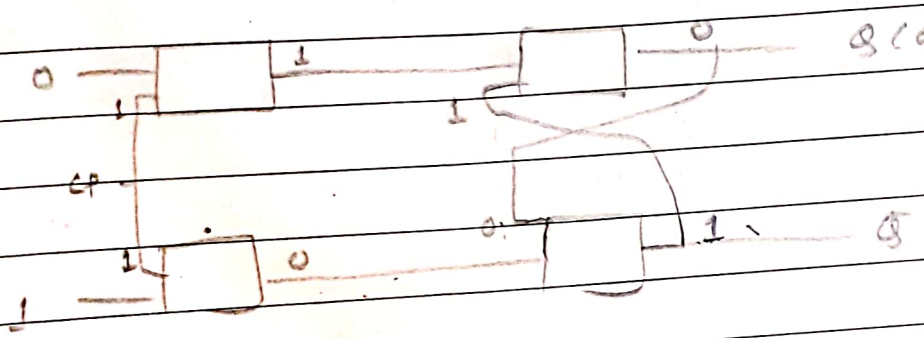
Q(t) S R
1 0 0

अगलेको o/p
पहिलाको o/p



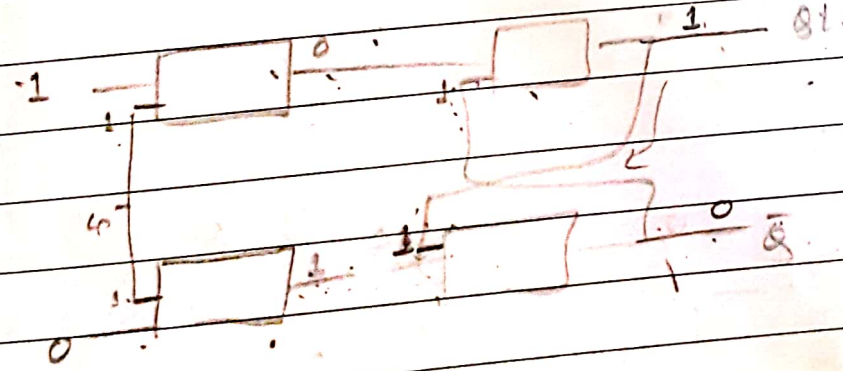
Q(t) S R
0 0 1

Reset



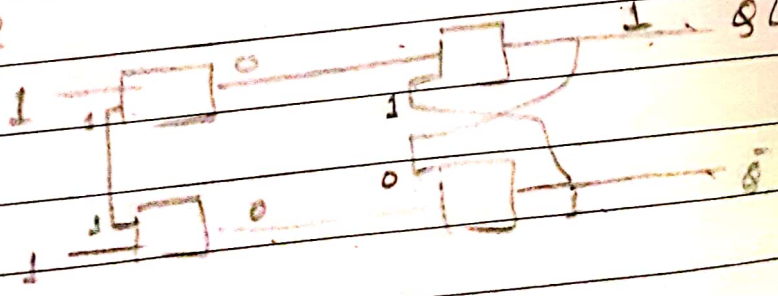
Q(t) S R
0 1 0

(set)



(Invalid)

Q(t) S R
0 1 1



(Delay)

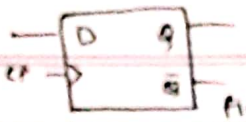


Fig: Graphical symbol

2) D flipflop

It is the modification of the clocked SR flipflop.

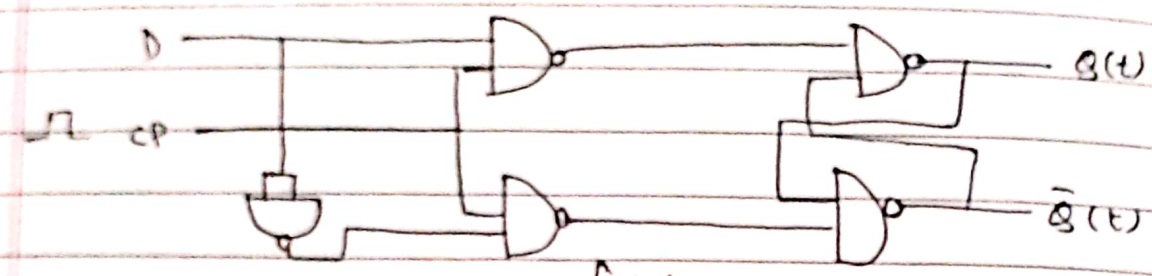


Fig: Logic diagram

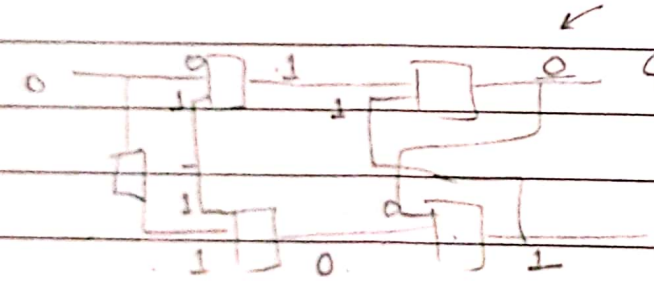
Characteristics Table

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

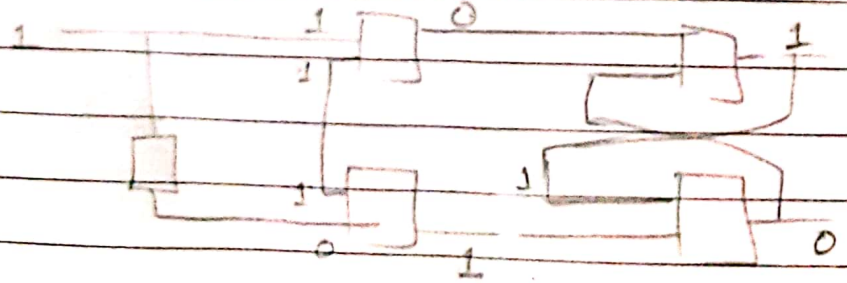
Q(t) \ D	0	1
0	0	1
1	0	1

Q(t+1) = D

D = 0



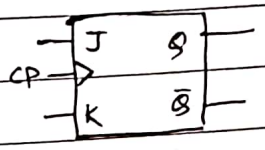
D = 1

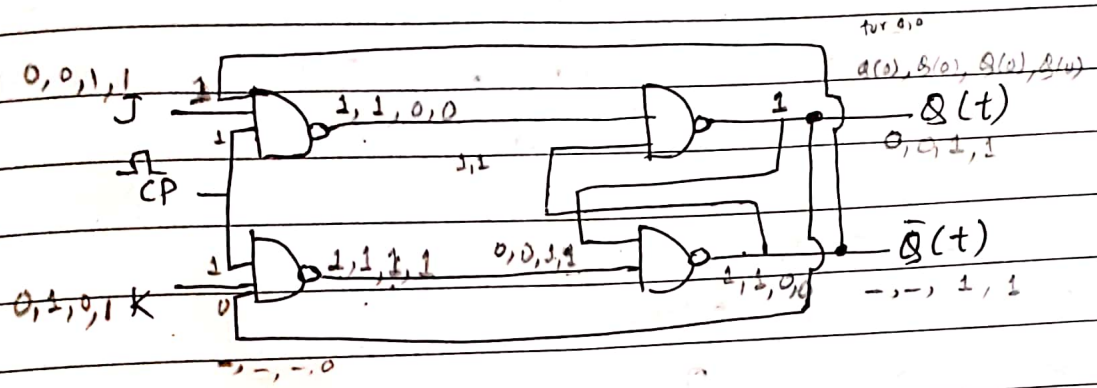


JK flipflop

- A JK flipflop is a refinement of the SR flipflop to solve the problem of intermediate state when both inputs are 1.

- In the JK flipflop inputs J and K behave like inputs S and R to set and reset the flip flop





Characteristic table

Q(t)	J	K	Q(t+1)	
0	0	0	0	(No change)
0	0	1	0	(Reset)
0	1	0	1	(set)
0	1	1	1	(toggle)
1	0	0	1	
1	0	1	0	
1	1	0	1	
1	1	1	0	

(Race around condition arises here)

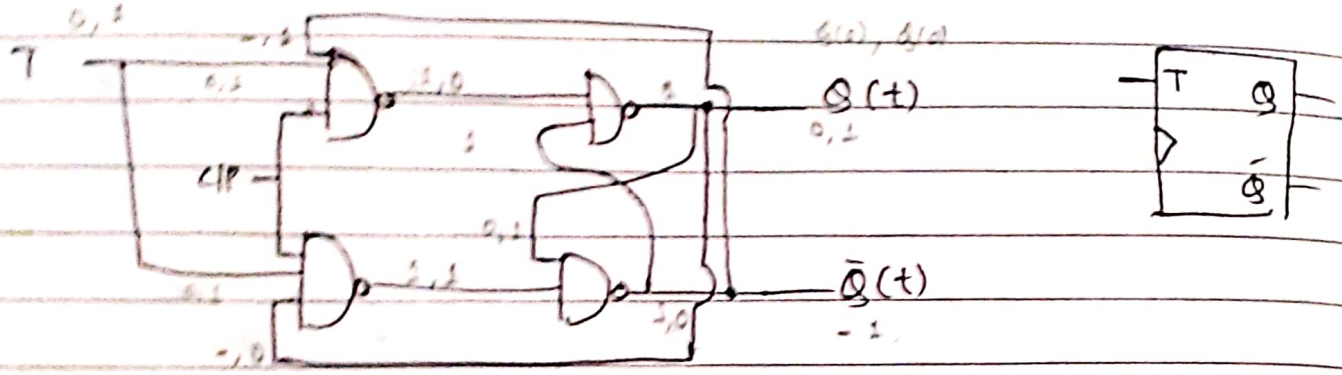
$Q = Q(t)$

Q(t) \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$Q(t+1) = \bar{Q}J + Q\bar{K}$$

T flipflop

The T flipflop is a single input version of JK flipflop as shown in figure.



Characteristics Table

Q(t)	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

When $\bar{T}=0$, $Q(t+1) \rightarrow$ No change
$T=1$, $Q(t+1) \rightarrow$ Complement of $Q(t)$

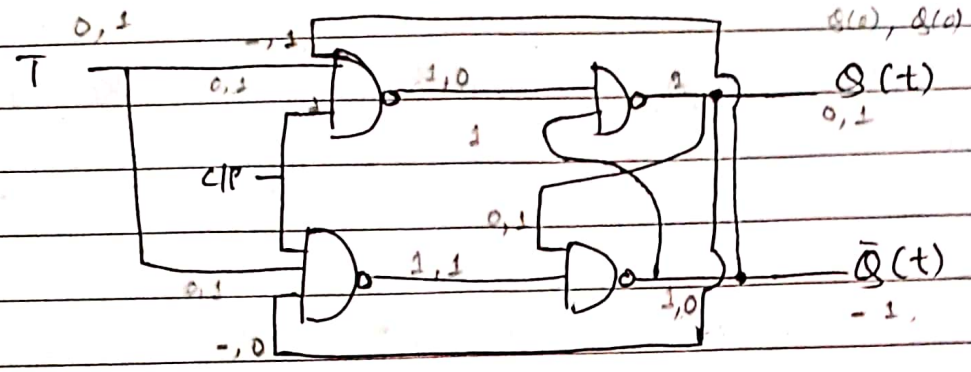
Q(t) \ T	0	1
0	0	(1)
1	(1)	0

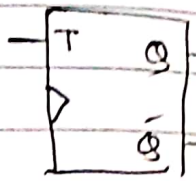
$$Q(t+1) = Q\bar{T} + \bar{Q}T$$

Characteristic equation.

T flipflop

The T flipflop is a single input version of JK flipflop as shown in figure.





Characteristics Table

Q(t)	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

When $\bar{T}=0$, Q(t+1) → No change
$T=1$, Q(t+1) → Complement of Q(t)

Q(t) \ T	0	1
0	0	(1)
1	(1)	0

$$Q(t+1) = Q\bar{T} + \bar{Q}T$$

characteristic equation.

Design of Synchronous Sequential Circuit

There are two distinct model by which we can design synchronous sequential circuit.

Moore Model	Melay Model
1. The output depends only on the present state and not on the input	1. The output depends on the present state aswell as input.
2. It requires more number of states and thereby more hardware.	2. It requires less number of states and thereby less hardware.
3. Output is generated one clock cycle later than Melay.	3. Output is generated one clock cycle earlier than Moore.
4. The output remains stable over entire clock period time and changes only when a state change occurs.	4. The output doesnot remain stable over entire clock period and changes over the same state.
5.	

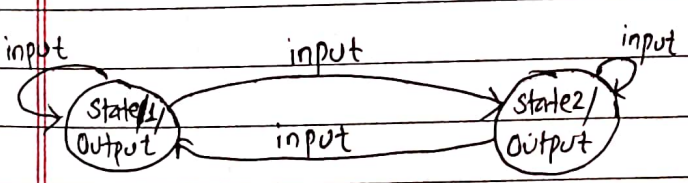


fig: Moore (state diagram)

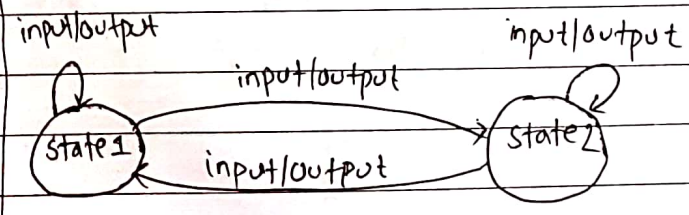


fig: Melay

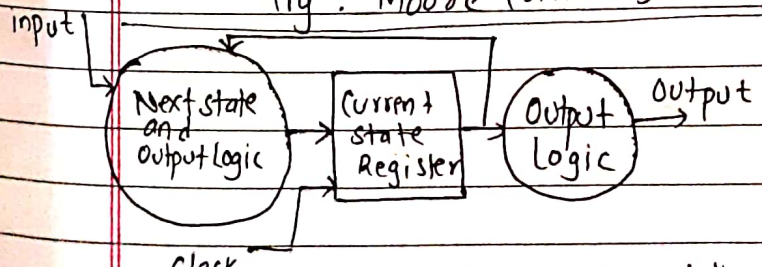
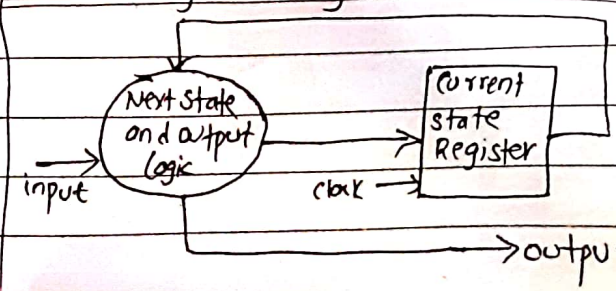


fig: Block diagram representation of moore

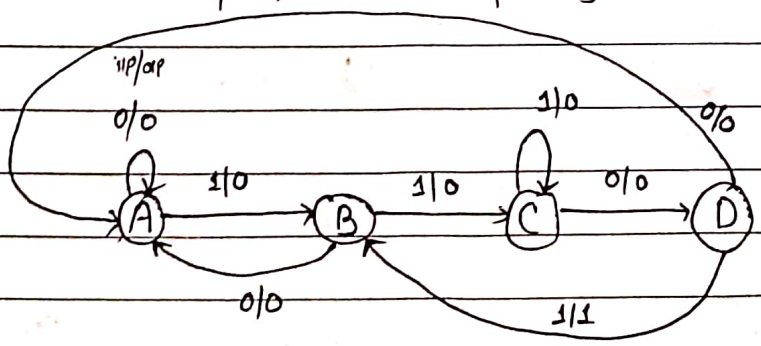


Sequential Machine design steps:

1. Select the model to be implemented either Moore or Melay.
2. Obtain the state diagram from the statement of the problem.
3. Obtain the next state table from the given information.
4. Reduce the number of states using state reduction method if necessary.
5. Assign binary codes to the states.
6. Choose the type of flipflop used. Determine the number of flipflop to be used and derive the ~~exic~~ excitation table and synthesis table.
7. Derive the simplified flipflop input equation and output equation from the statetable using KMAP.
8. Draw the logic diagram with the flip flop and combinational gates according to those equations.

Q. Design a sequential detector that receives binary data stream at its input x and when a combination 1101 arrives at the input, its output y is set high.

Soln:



↓ 4bit so, 4 state required at least in Melay.

1101

1101

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
00	00	01	0	0
01	00	10	0	0
10	11	10	0	0
11	00	01	0	1

↳ 2 bit so 2 flipflop needed.

Present state		Input	Next state		D_1	D_2	Y
Q_1	Q_0	X	I_1	I_0			
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	0
0	1	1	1	0	1	0	0
1	0	0	1	1	1	1	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	1	0	1	0	1	1

Q_1 and I_1 are D_1 nikalne.
I_1 and D_1 same.

For D1:

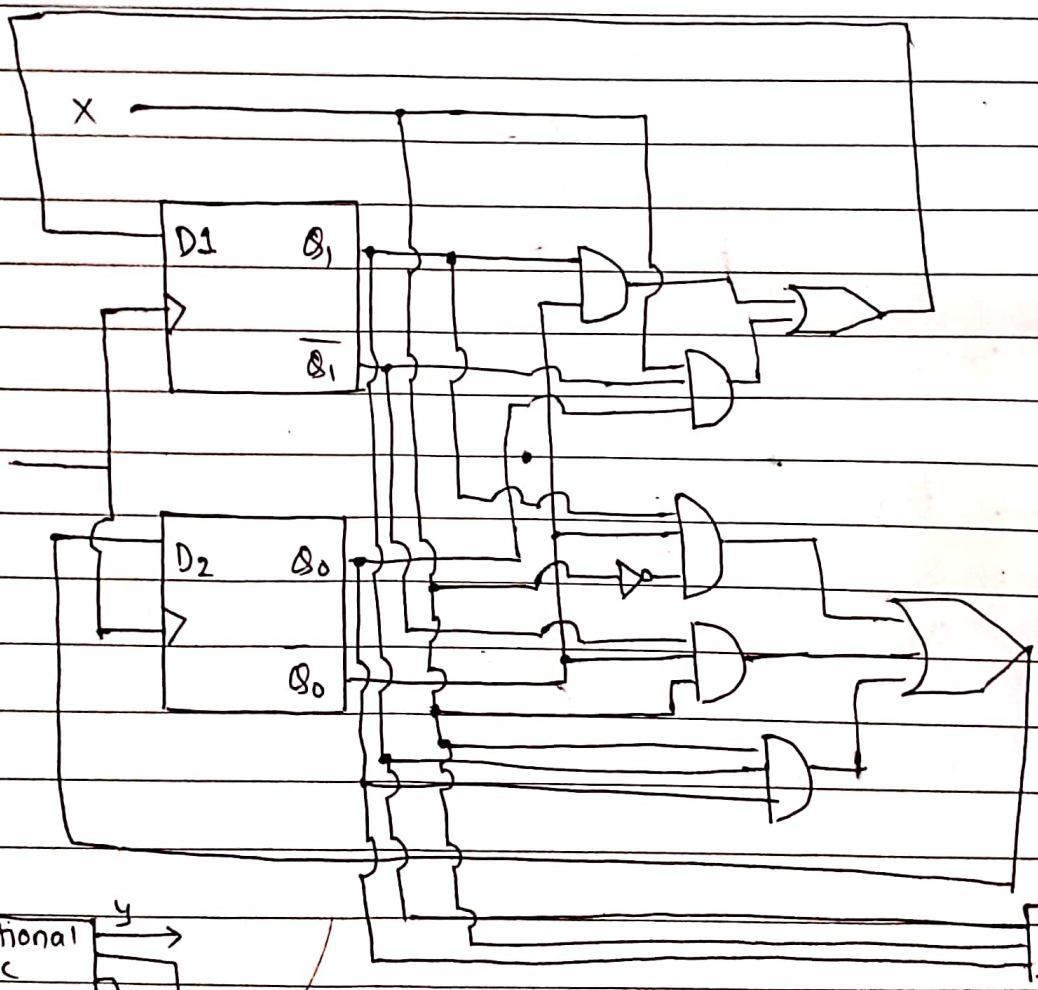
Q_1 \ Q_0X	00	01	11	10
0	0	0	(1)	0
1	[1]	[1]	0	0

$D_1 = \bar{Q}_0 Q_1 + Q_0 X \bar{Q}_1$

For D2:

Q_1 \ Q_0X	00	01	11	10
0	0	(1)	0	0
1	(1)	0	(1)	0

$D_2 = Q_1 \bar{Q}_0 \bar{X} + \bar{Q}_1 \bar{Q}_0 X + Q_1 Q_0 X$



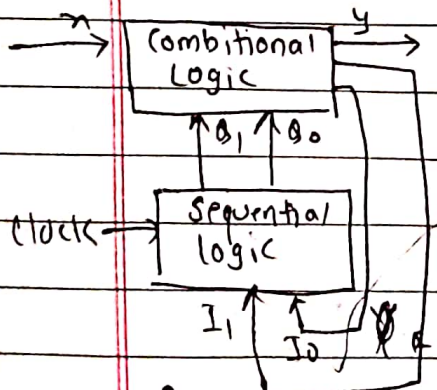


Fig: Block diagram

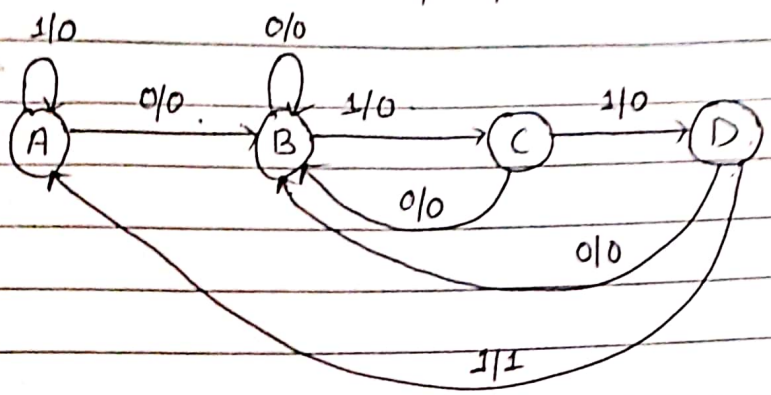
$Y = Q_1 Q_0 \bar{X}$

For X

Q_1 \ Q_0X	00	01	11	10
0	0	0	0	0
1	0	0	0	(1)

(Sn same as previous)

8. .... 0111 Using T flipflop



Present state	Next State		Output	
	X=0	X=1	X=0	X=1
A	B	A	0	0
B	B	C	0	0
C	B	D	0	0
D	B	A	0	1

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
00	01	00	0	0
01	01	10	0	0
10	01	11	0	0
11	01	00	0	1

Present state Q_1	Q_0	Input x	Next state J_1	J_0	T_1	T_2	y
0	0	0	0	1	0	1	0
0	0	1	0	0	0	0	0
0	1	0	0	1	0	0	0
0	1	1	1	0	1	1	0
1	0	0	0	1	1	1	0
1	0	1	1	1	0	1	0
1	1	0	0	1	1	0	0
1	1	1	0	0	1	1	1

For T_1

Q_1 \ Q_0x	00	01	11	10
0	0	0	1	0
1	1	0	1	1

$$T_1 = Q_0X + Q_1Q_0 + Q_1\bar{X}$$

For T_2

Q_1 \ Q_0x	00	01	11	10
0	1	0	1	0
1	1	1	1	0

$$T_2 = \bar{Q}_0\bar{X} + Q_1\bar{Q}_0 + Q_0X$$

For Y

Q_1 \ Q_0x	00	01	11	10
0	0	0	0	0
1	0	0	0	1

$$Y = Q_1Q_0\bar{X}$$

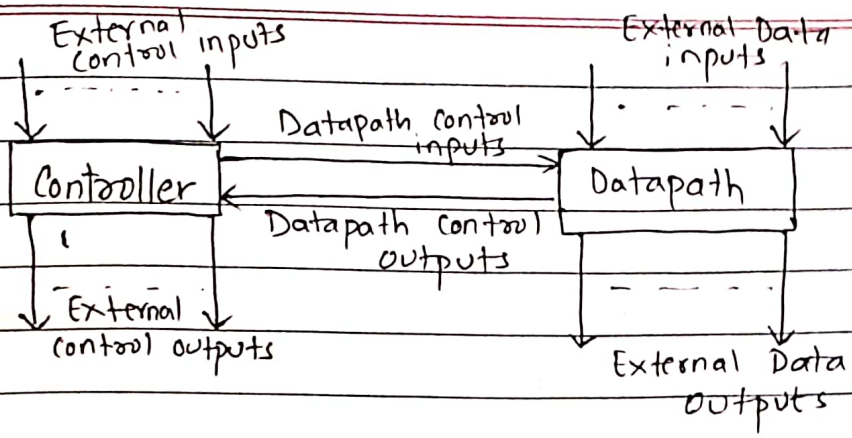
Custom Single Purpose Processor design

Data Path

- Stores and manipulates a system data of different forms.
- Data may include binary numbers representing external conditions like temperature, pressure or speed, characters to be displayed on screen or a digitized image to be stored and compressed etc.
- Contains register units, functional units and connection units like wires, MUX.
- Can be configured to read data from particular register, fit that data through functional units, configure to carry out particular operations like add or shift and store the operation results back into particular registers.

Controller

- Carries out the above stated configuration of datapath.
- Set the datapath control includes like register load and MUX select signals of the register units, functional units and connection unit to obtain the desired configuration at a particular time.
- monitors external control inputs aswell as datapath control output known as status signals, coming from functional units and sets external control output aswell.



Technique for building a processor

We begin with a sequential program describing the computation task we wish to implement. We provide an example based on computing the factorial of a number. The systems have n_i as input data d_o as output data and f_i as a control input. The system functionality should be implemented to output the factorial of the number n_i.

To begin building our single purpose processor implementing the factorial program, we first convert our program into complex state diagram in which states and arcs may include arithmetic expression and these expression may use external inputs and outputs as well as variables.

Converting a program into FSMD : (Final state machine)

```
#include <stdio.h>
int fact (int n);
int main (){
{
    int n, res;
```

```
printf("Enter a number");
  res = fact(n
scanf("%d", &n);
  res = fact(n);

printf("Factorial = %d", res)
  return 0;
}
int fact( int n ){
if n== 1
      return 1
else
      return   n * fact(n-1)
}
```

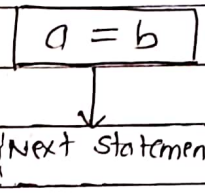
Recursion

```
int n,d
while(1)
{
  while (!f_i);
    {
      d = 1;
      n = n-1;
    }
  while (n!=0)
   {
      d= d *n;
      n= n - 1;
   }
  d_0 = d;
}
```

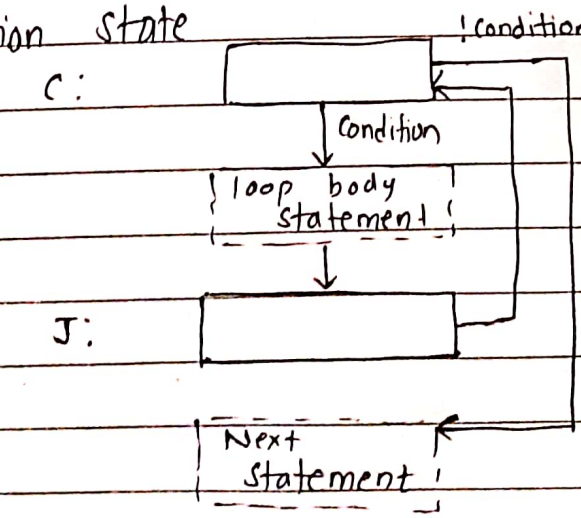
Templates for the various types of statements can be used for converting program to an FSMD.

- First we classify each statement as an assignment statement, loop statement or branch statement.

- For an assignment statement, we create a state with that statement as its action. We add an arc from this state to the state further next statement, whatever type it may be

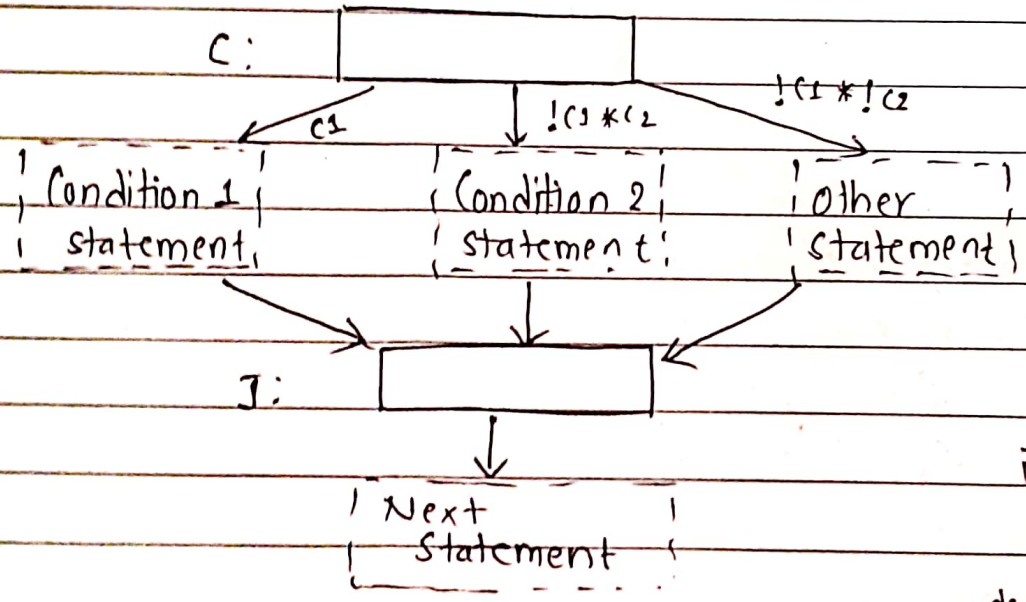


- For a loop statement, we create a condition state C and a join state J, both with no action. We add an arc with the loops condition from the condition state to the first statement in the loop body. We add a second arc with the complement of the loop condition from the condition state to the next statement after the loop body. We also add an arc from the join state back to the condition state




```
while (condition)
{
    loop body statement
}
next statement
```

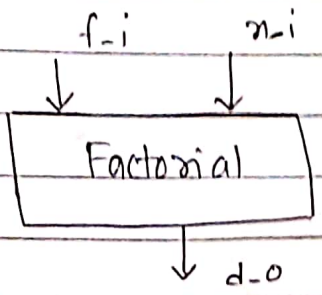
- For a branch statement, we create a condition state 'C' and a join state 'J', both with no action. We add an arc with the first branch condition & from the condition state to the branch first statement. We add another arch with a complement of the first branch condition anded with second branch condition from the condition State to the branch first statement. We repeat these for each branch. Finally we connect arc leaving the last statement of each branch to the join state and we add an arc from this state to the next statement's state




```
if (c1)
    cond1 statement
else if (c2)
    cond1 statment
else
    other statement

Next statement
```

CSPP computing factorial of a number:

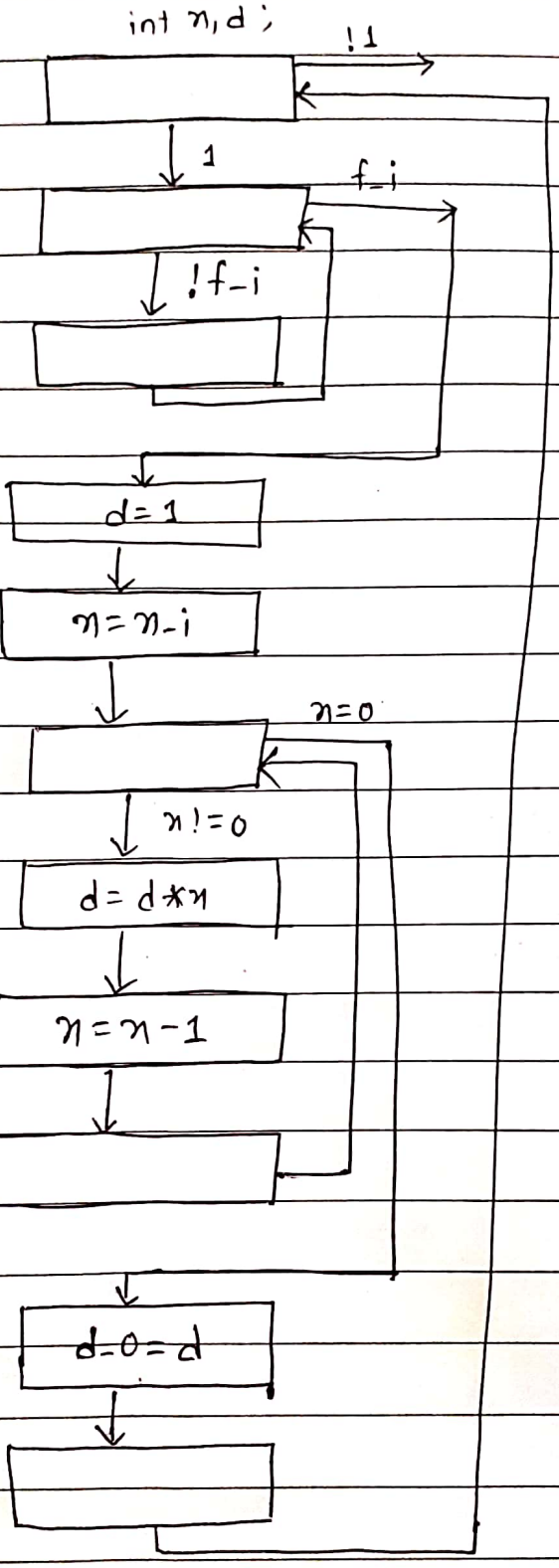
1) Black Box



2) Structured Algorithm (Program)

```
int n, d;
while (1)
{
    while (!f_i);
    d = 1
    n = n_i;
    while (n!=0)
    {
        d = d * n;
        n = n - 1;
    }
    d_o = d;
}
```

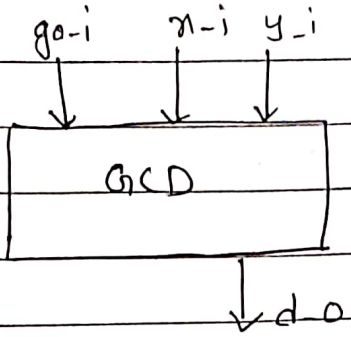
3. Final state Machine.



==0)

Design a CSPP that calculates the Greatest Common Divider (GCD) or HCF of two numbers. Start with the algorithm and design the FSMD, daptaperth and controller (FSM)

1. Blackbox

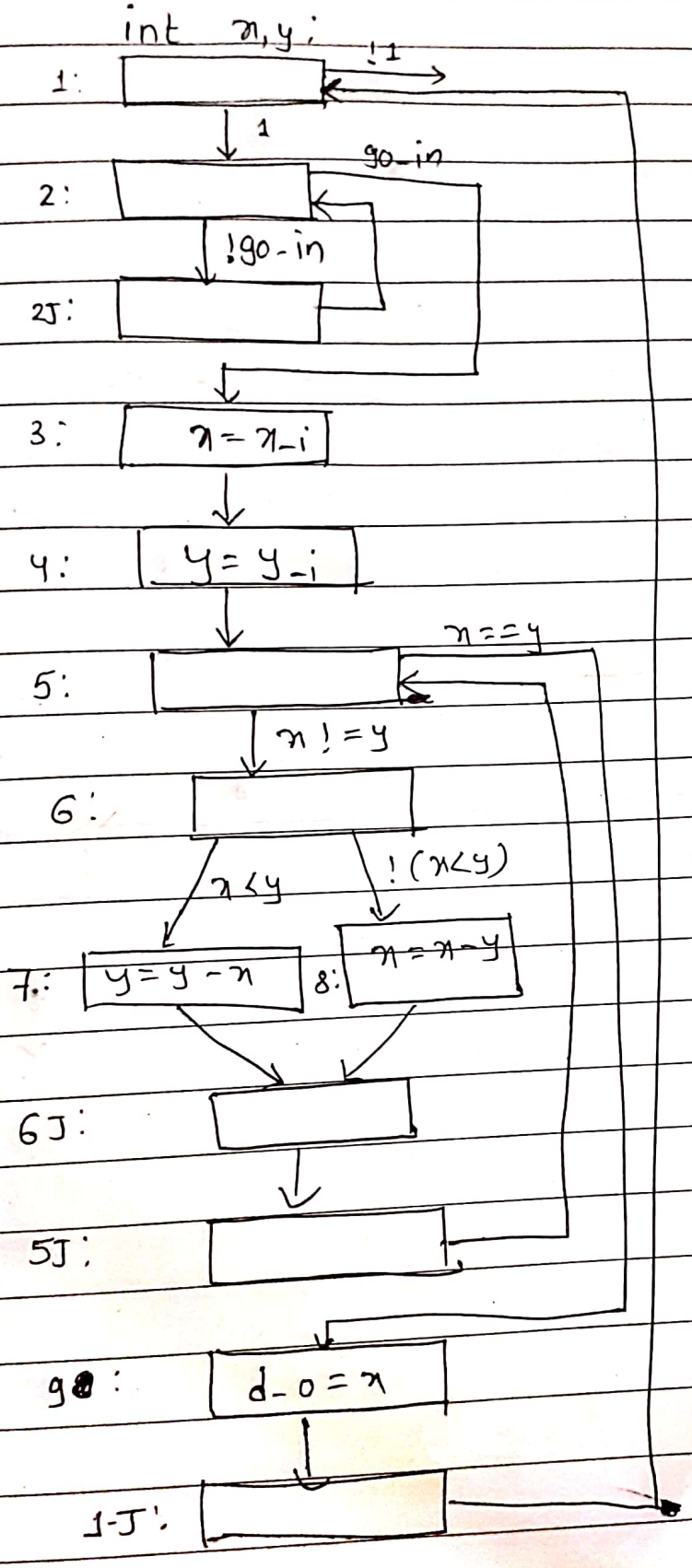


2)

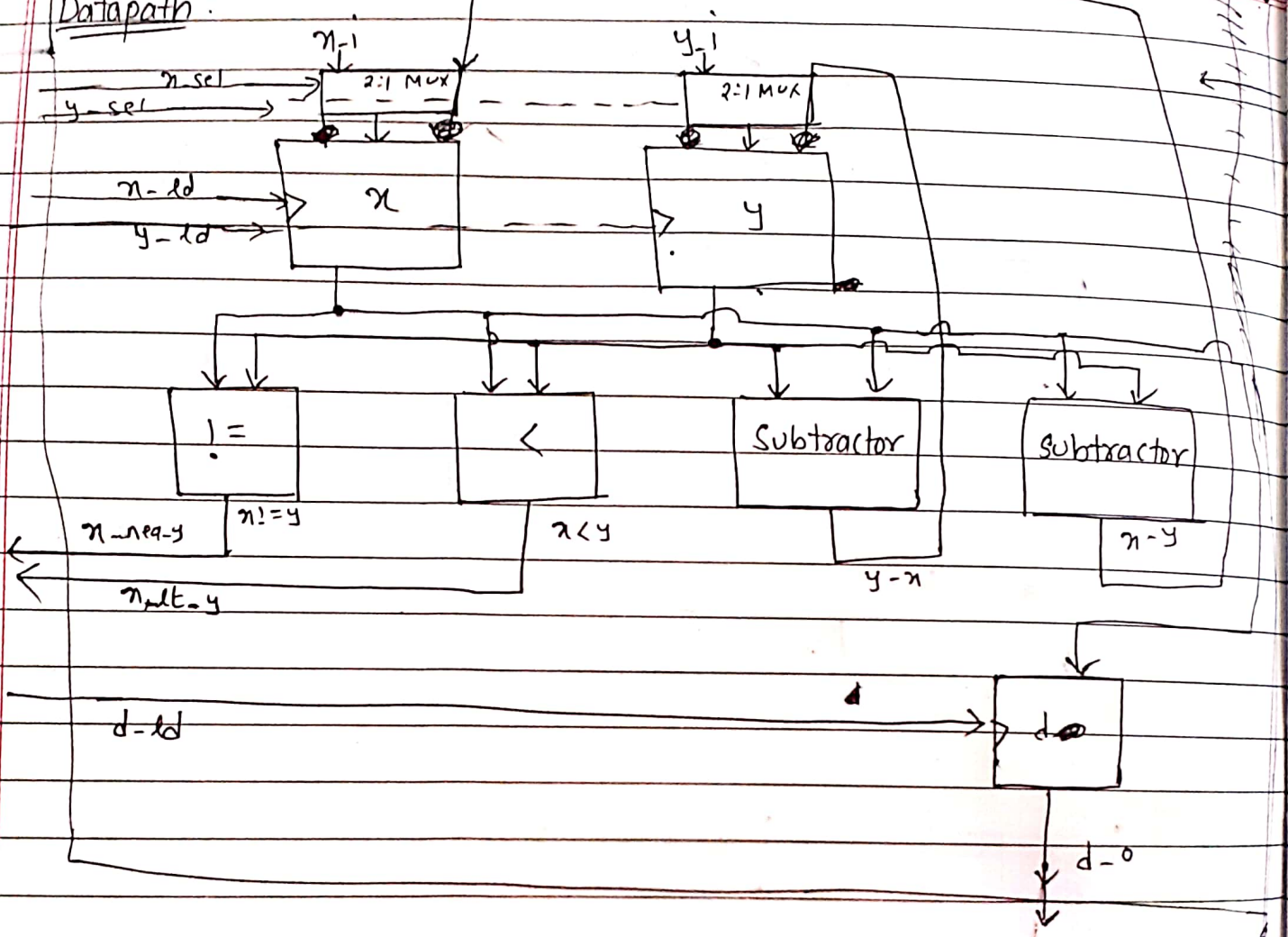
```
int i, x, y, min, hcf;
while(1){
printf("Enter two num");
scanf("%d %d", &x, &y);

min = (x < y) ? x : y;

for (i=1, i<=min, i++)
{
    if (x%i==0 && y%i==0)
    {
        hcf = i;
    }
}
d_o = hcf;
}
```

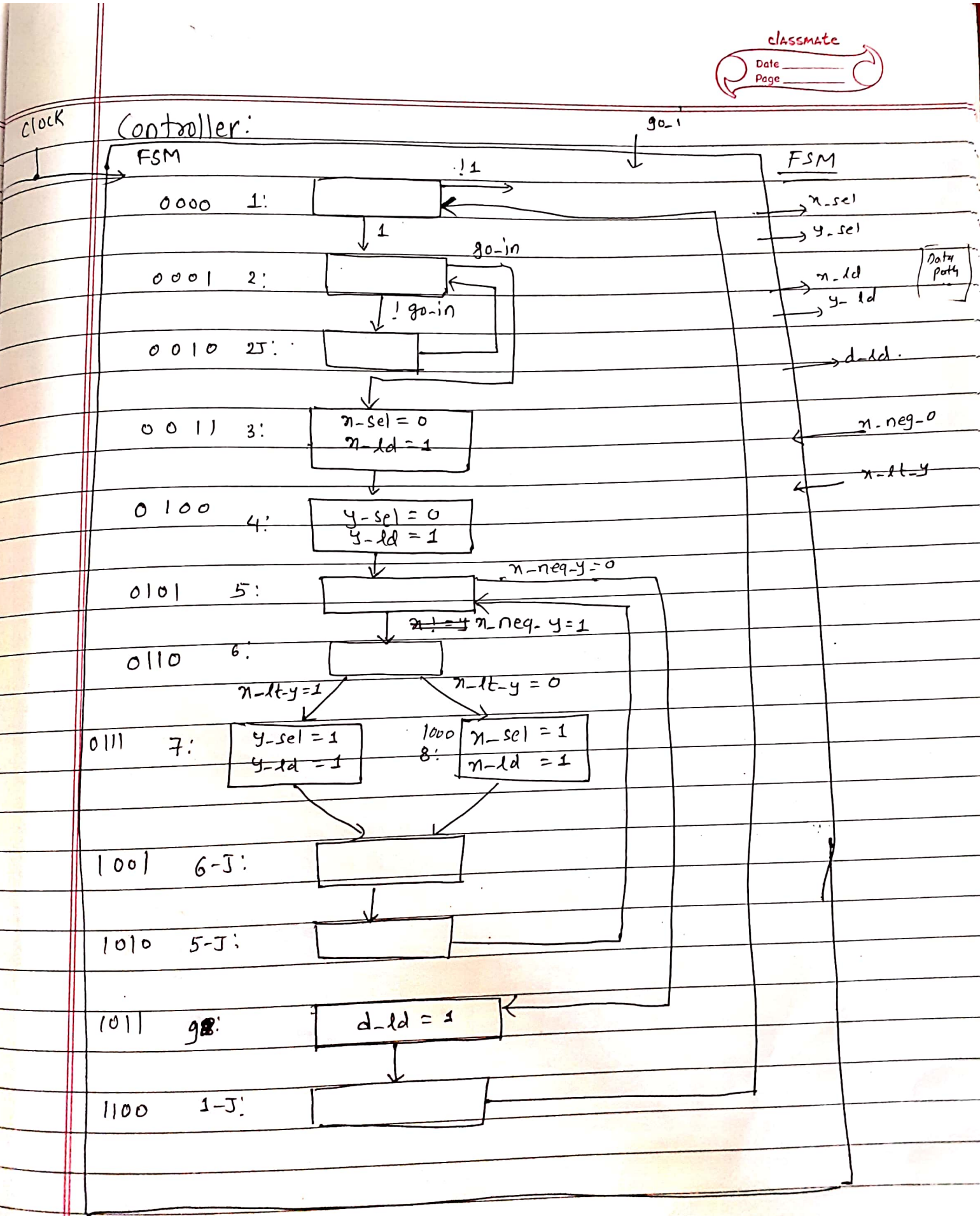
```
int x, y;
while (1)
{
    while (!go_in);
    x = x_i
    y = y_i
    while (x != y)
    {
        if (x < y)
            y = y - x;
        else
            x = x - y;
    }
    d_o = x;
}
```

FSMD


int x,y;
1:
!1
1
go_in
2:
!go_in
2J:
3:
x = x_i
4:
y = y_i
x==y
5:
x != y
6:
x<y
!(x<y)
7:
y = y - x
8:
x = x - y
6J:
5J:
9:
d_o = x
1-J:

Datapath:


x_in
y_in
x_sel
y_sel
2:1 MUX
2:1 MUX
x_ld
y_ld
x
y
!=
<
Subtractor
Subtractor
x!=y
x<y
y-x
x-y
x_neq_y
x_lt_y
d_ld
d
d_o

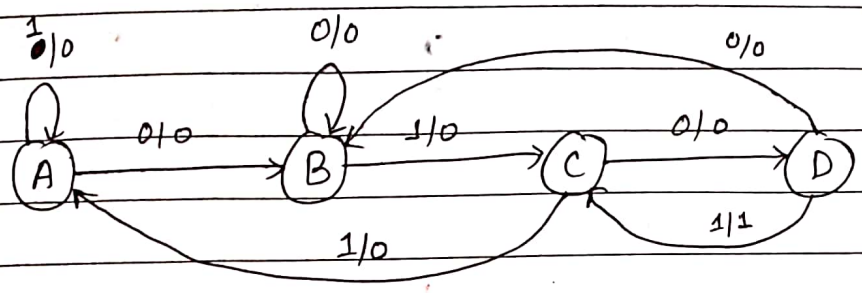

Controller:
clock
FSM
go_i
!1
0000 1:
1
go_in
0001 2:
! go_in
0010 2J:
0011 3:
x_sel = 0
x_ld = 1
0100 4:
y_sel = 0
y_ld = 1
0101 5:
x_neq_y = 0
x = y
x_neq_y = 1
0110 6:
x_lt_y = 1
x_lt_y = 0
0111 7:
y_sel = 1
y_ld = 1
1000 8:
x_sel = 1
x_ld = 1
1001 6-J:
1010 5-J:
1011 9:
d_ld = 1
1100 1-J:
FSM
x_sel
y_sel
x_ld
y_ld
Data path
d_ld
x_neq_0
x_lt_y

Roll: 15

Assignment

Q Design a sequence detector that receives binary data stream at its input 'x' and when a combination of '0101' arrives at the input, its output 'y' is set to high.

Soln:



State Table:

Present State	Next State		Output	
	X = 0	X = 1	X = 0	X = 1
A	B	A	0	0
B	B	C	0	0
C	D	A	0	0
D	B	C	0	1

Present state	Next State		Output	
	X = 0	X = 1	X = 0	X = 1
A 00	01	00	0	0
B 01	01	10	0	0
C 10	11	00	0	0
D 11	01	10	0	1

To design a circuit:

Present state		Input	Next state		D_1	D_2	Y
Q_1	Q_0	X	I_1	I_0			
0	0	0	0	1	0	1	0
0	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	1	1	0	1	0	0
1	0	0	1	1	1	1	0
1	0	1	0	0	0	0	0
1	1	0	0	1	0	1	0
1	1	1	1	0	1	0	1

Input Equation for D1:

Q_1 \ Q_0X	00	01	11	10
0	0	0	1	0
1	1	0	1	0

$D1 = Q_0X + Q_1\bar{Q}_0\bar{X}$

Input Equation for D2:

Q_1 \ Q_0X	00	01	11	10
0	1	0	0	1
1	1	0	0	1

$D2 = \bar{X}$

output Equation

Q_1 \ Q_0X	00	01	11	10
0	0	0	0	0
1	0	0	1	0

$Y = Q_1Q_0X$

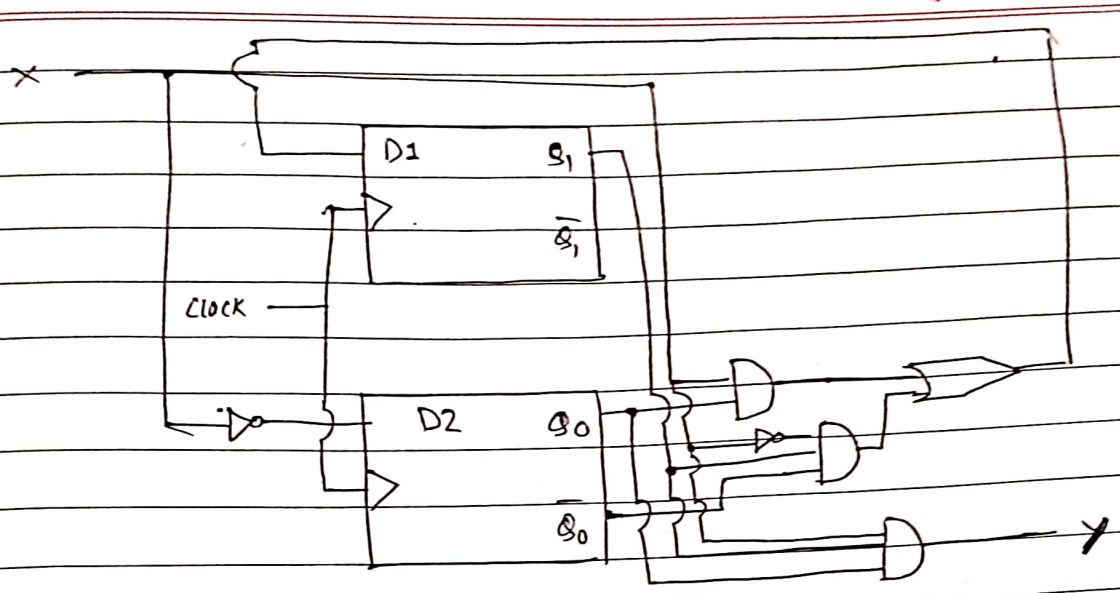


fig: Circuit Diagram

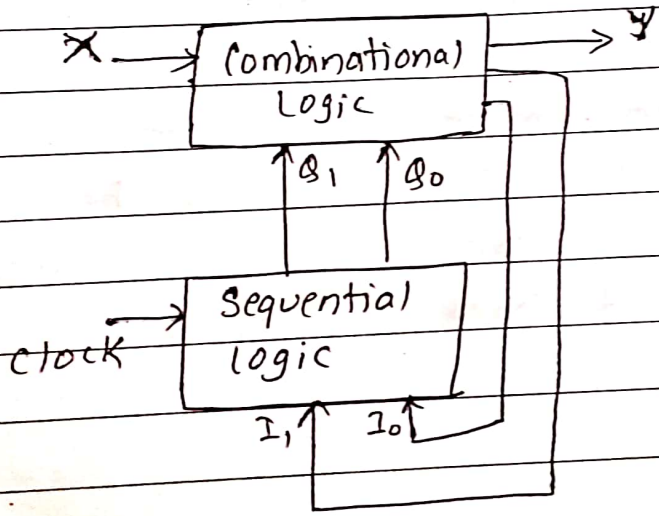


fig: Block Diagram

Optimizing CSPP:

Optimization is the task of making the design metric values the best possible. The FSMD and FSM have several states that do nothing and could have been removed. Basically there are four areas in which we have the optimization opportunities:

i) Optimizing the original program
ii) Optimizing the FSMD
iii) Optimizing the Datapath
iv) Optimizing the FSM

i) Optimizing the Original program

Let us consider the GCD program stated earlier. We can analyze the number of computations and size of variables that are required by the algorithm. In other words, we can analyze the algorithm interms of time Complexity and space Complexity.

```
int x,y;
While(1)
{
    while (!go_in);
    if (x_i >= y_i)
    {
        x = x_i ;
        y = y_i ;
    }
    else {
        x = y_i;
        y = x_i ;
    }
```

```
while (y != 0)
{
    r = x % y;
    x = y;
    y = r;
}
d_o = x;
}
```

Let us compare, this second algorithm with the earlier one. Computing the GCD of 8 and 48 with the earlier algorithm would step through its inner loop yielding the x and y values as follows:

(8, 48), (8, 40), (8, 32), (8, 24), (8, 16), (8, 8)

Thus outputing 8.

The second algorithm would step through as follows:

(8, 48), (8, 0)

Thus outputing 8.

The second algorithm would only require two steps instead of six steps for the first program. Therefore the second algorithm is far more efficient interms of time.

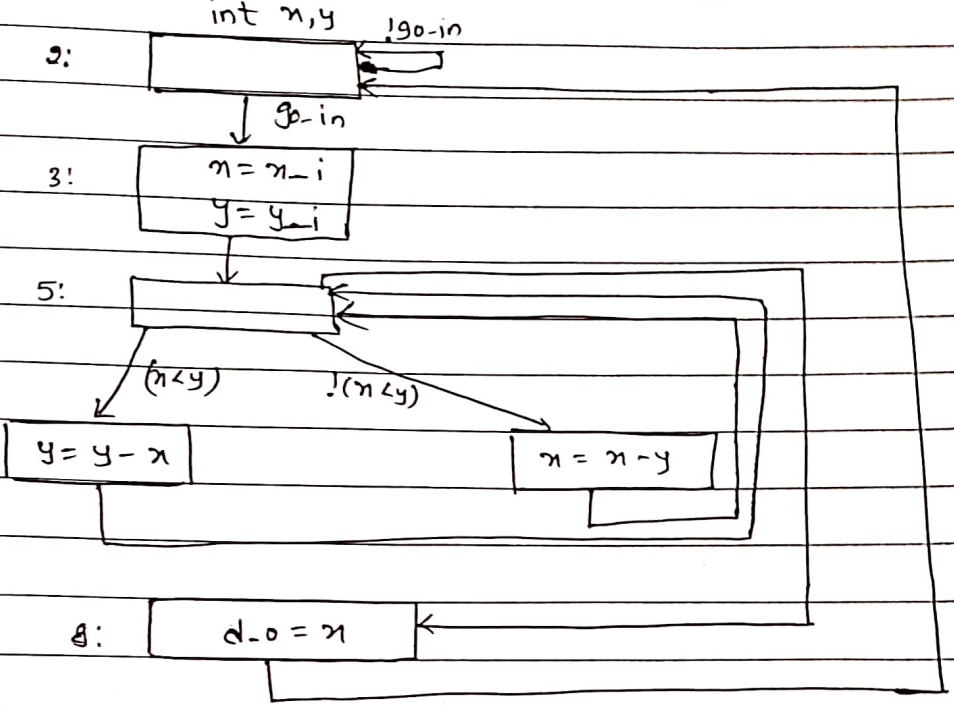
ii) Optimizing the FSMD

The task of assigning operations from the original program to states in an FSMD is refered as scheduling. Thus obtained scheduling using the template based method can be improved.

Consider the original FSMD for the GCD processor.

- State 1 is clearly not necessary since its outgoing transition have constant values.
- States 2 and 2-J can be merged into a single state since there are no loop operation in betn them.
- State 3 and 4 can be merged since these perform assignment operations that are independent of one another
- States 5 and 6 can be merged
- States 5-J and 6-J can be eliminated with the transition from state 7 and 8 pointing directly to state 5.
- Similarly state 1-J can be eliminated.

The resulting reduced FSMD is shown in figure below:
The FSMD can be reduced from 13 states to only 6 states.



iii) Optimizing the Datapath

We created a unique functional unit for each arithmetic operation in the FSMD. Such one to one mapping is often not necessary. Many arithmetic operation can share the same functional unit if that functional unit supports those operation and those operations occured in different states. In the GCD example, states 7 and 8 perform the subtraction and in datapath each subtraction got its own subtractor. Instead we could use a single subtractor and use multiplexer to chose whether the subtraction is x-y or y-x.

iv) Optimizing the controller FSM

Optimization in the design of FSM could be achieved in two areas.

a) State encoding

The task of assigning the unique bit pattern to each state in FSM is called state encoding. For an FSM with n states where n is a power of 2, there are n! possible encodings. Even more encodings are possible since we can use more than $\log_2(n)$ bits to encode n states upto n bits. to achieve one-hot encoding.

b) State minimization

The task of merging equivalent states into a single state. Two states are equivalent if for all possible input combinations, those two states generates the same output and transition to the same next state. Such states are clearly equivalent, since merging them will yield exactly the same output behaviour. unlike optimizing FSMD.