# DIGITAL SYSTEM DESIGN

HDL

Verilog VHDL

c = a + b → Tool

↓

instr (compiler)

↓

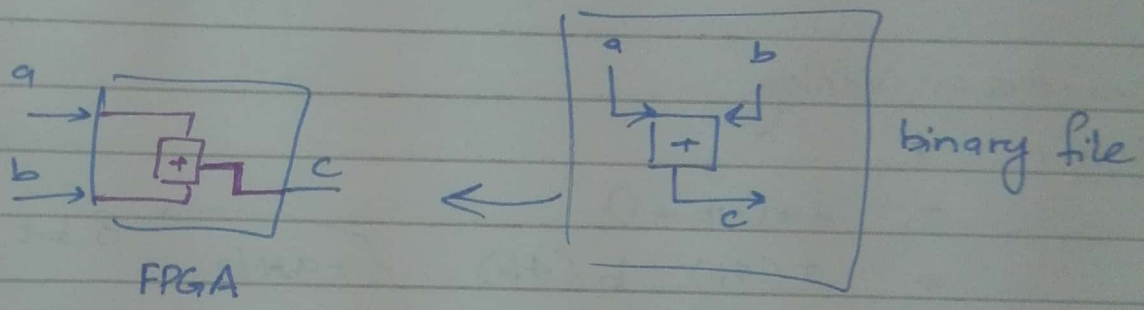
decode

parse operati

( + )

Translation

S/W to H/W



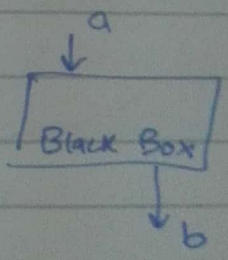

## Verilog

- is an HDL
- C like syntax
- change of thought process

basic building block → Module

port list

```
module user_defined_name (a, b);
  —
  —
  —
endmodule
```



# Dataflow Modelling

- LHS must be declared as type wire
- always use keyword assign before expression

'assign' c = a+b → output is wire by default

```
module test(a, b);
input a;
output b;    // by default a wire
assign b = a + 1;
endmodule
```

# Operators in Verilog

Arth:

+

\- a+(~b+1)

* a (4b) b (4b) c = a*b → 8 bit (4+4) add not double

/ $2^{power}$ Dem: 1, 2, 4, 8...

% same constraint as divide

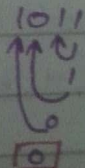
** 

# Unary Reduction

always 1 bit answer

```
wire [3:0] a;
wire       c;
assign c = &a;
```

a = 4'b 1011




# Logical

always 1 bit answer

assign c = a && b

if (a>0)
1

a = 2'b 1x

b = 2'b 1x

assign c = (a == b) → c = x

assign c = (a === b) → c = 1   === treats x and z as value not garbage

## Concatenation

wire [1,0] a,b;   a = 2'b 11   b = 2'b 01

wire [3,0] c ;

c = {b, a} ;   4'b 0111

~~{sum, c} = a + b ;~~

{c, sum} = a + b

## Replication

assign c = {4{a}}   // a = 2'b 10

↳ 10101010

c = {{2{a[1]}}, a} // copy a's MSB bit (sign extension)

## Shift Operators

| | left | right |
|---|---|---|
| Logical | << | >> |
| Arth | <<< | >>> |

left shift → xply by $2^{shift}$

right shift → divide by $2^{shift}$

# Behavioral Modelling

- LHS type 'reg'
- procedural block

initial
begin
—
—
end

runs once at start of execution

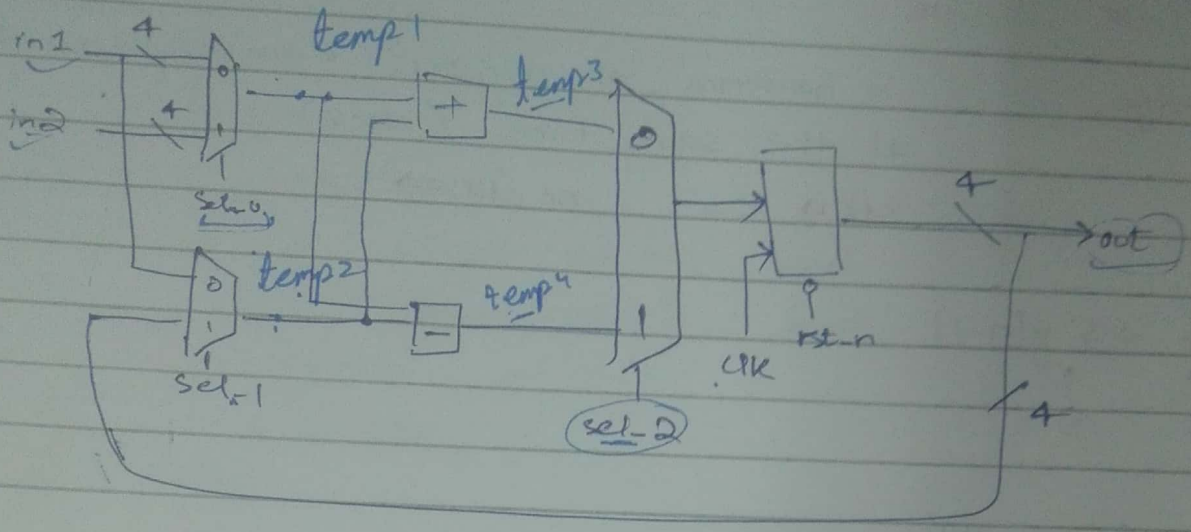
always
begin
—
—
end

loops infinitely

used for generating clock

```
always @ (b,c,e,f)
begin
  a = b+c
  d = e+f
end
```

runs if any one of (b,c,e,f) changes




Behav Modelling (initial) dont use, doesnt map to hardware

- always @ (*)
- always @ (sensitivity list)
- always @ (posedge clk) → physical register only

sequential logic → works on clock

always @(posedge clk)

a <= c+d ] DON'T WRITE CODE HERE

e <= f+g

registers (physical)

b=3 c=2 f=1 a=0

Procedural Assignment

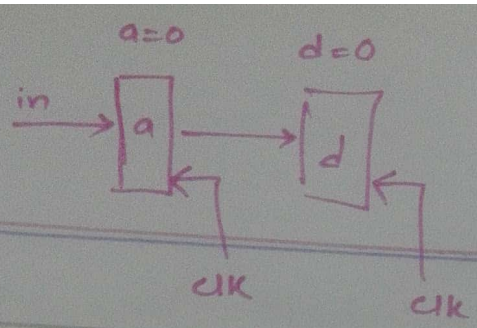
'=' a = b+c Blocking Assignment

'<=' a <= b+c Non Blocking Assignment

a <= b+c does not assign value to a, stores in temp var at =5

d <= a+f → dt = 1 value assignment when prog reached 'end'

run in parallel



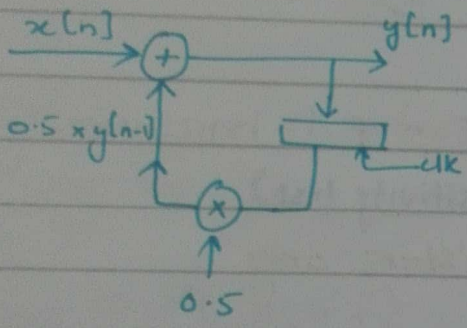

at 1st pos edge,
a = 5
d = 0

in hardware, all registers are updated at the same time (clock)

clk ⇒ <=  no clk ⇒ =

$$y[n] = x[n] + 0.5\, y[n-1]$$

① ②




If else, case, for, while, repeat, forever
Specific to behav modelling. only in blocks (initial or always)
if / else → multiplexer

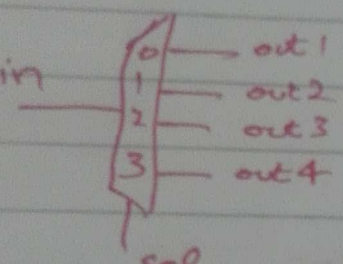
Syntax 1 assign out = (sel == 0) ? a : b

Syntax 2

```
always @ *
  begin
    if (sel == 0)
      out = a
    else if (sel == b)
      out = b
  end
```

## Case.

```
always @ *
  begin
    case (sel):
    0 : begin  out = a;  end
    1 : begin  out = b;  end
    default : out = 0
    endcase
  end
```




```
case (sel)
  0 : out1 = in ;
  1 : out2 = in ;
  2 : out3 = in ;
  3 : out4 = in ;
default : begin out1=0; out2=0; out3=0;
          out4=0; end
endcase;
```

## Repeat

```
always @ (posedge clk)
begin
  repeat (10)   // loop
    a = a+1;  ——→ repeat 10 times
end
```

Generates copies of hardware. Problem with loops.

**for loop**

```
always @ ( )
  begin
    for (i=0; i<N; i=i+1)
      begin

      end
  end
```

**while loop**

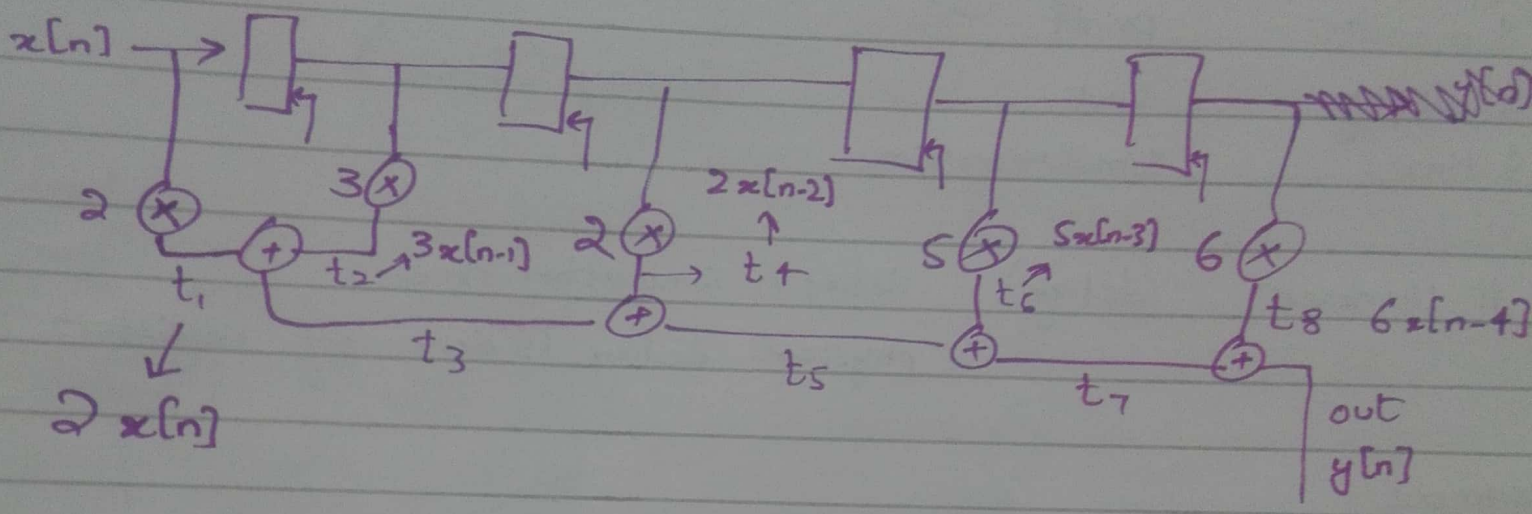
```
while (i<10)
  begin

  end
```

**forever** wont work on hardware, Just simulation

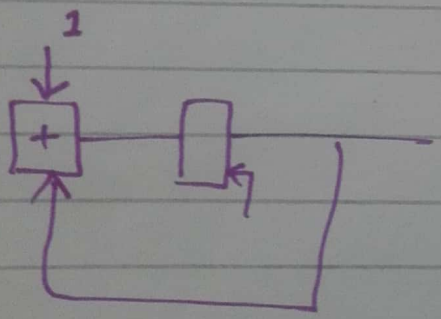
```
initial
  forever
    begin
    end
end
```

# Lab #3

## Task 2.




## Task 3.




4 bit register

# Simulation

System Tasks

$monitor printf — runs infinitely (whenever inputs change)

$display cout display ($time)

$time simulation time

# Number (delay) delay in simulation time

physical delay is done by registers

example

```
initial
    clk = 0

initial
  begin
    forever
      begin
        #10          delay
        clk = ~ clk  toggle
      end
  end
```

alternatively just use

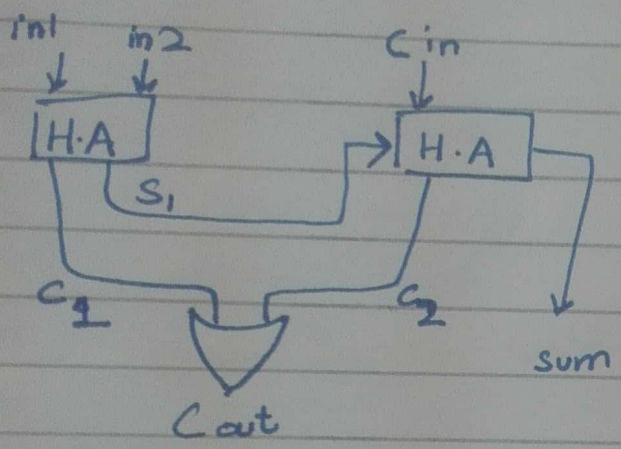
```
always          inf loop
  # 10
  clk = ~ clk;
```

# Module Calling (Instantiation)

```
module half_adder (in1, in2, s, c);
 input in1, in2;
 output s, c ;
 assign {c, s} = in1 + in2

end module
```

alternatively,

```
reg c, s
always @ *
{c, s} = in1 + in2
```




```
module full_adder (in1, in2, cin, cout, s)
 input in1, in2, cin
 output cout, s
 // calling module
  wire c1, s1, c2;          outputs will always be of type wire
  half_adder obj (in1, in2, s1, c1);
  half_adder obj2 (s1, cin, s, c2);
  assign cout = c1 | c2
  endmodule
```

now simulation

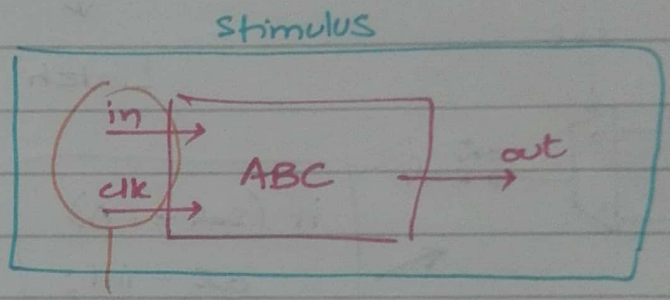
```
module stimulus();
reg in1,in2,cin;   wire s,cout;
full_adder obj (in1,in2,cin,cout,s);
 initial
 begin
     in1=0; in2=0; cin=0;
     #10
     in1 = 1;
     #10
     in2= 1;
     #10
     cin=1;
 end

 initial
 $monitor(     )

endmodule
```

input port is defined as wire or reg but output port is always of type wire in module calling.

# Simulation (Revision)

Stimulus or test bench

- verification of main logic
- call diff modules
- i/p's test vectors (inputs)
- main in C



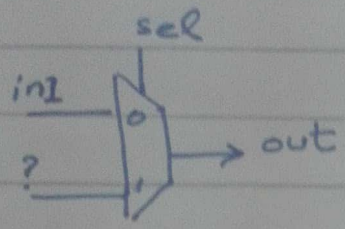

RTL : register transfer level  can be behavioral or hybrid

## Parameters

```
module adder (in1,in2,out)
 parameter N=1
 input [N-1:0] in1, in2 ;
 output [N-1:0] out ;

endmodule
```

```
adder #(,4) obj (in1, in2, out)      calling

     N=4
adder #(.N(4),.M(3)) obj (in1, in2, out)
```

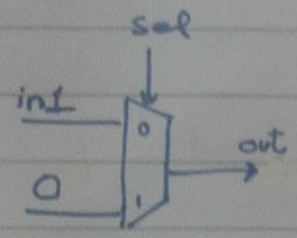
# Guidelines




```
if (sel==0)
  out = in1;
```

Latch

avoid latches at all costs

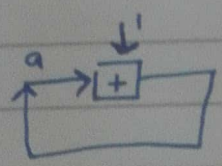



```
if (sel == 0)
    out = in1;
else
    out = 0;
```

```
out = 0;
if (sel == 0)
  out = in1;
```

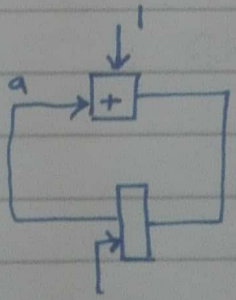
always @ *

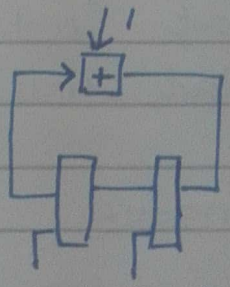
a = a + 1




combinational logic (without clock)

→ infinite loop

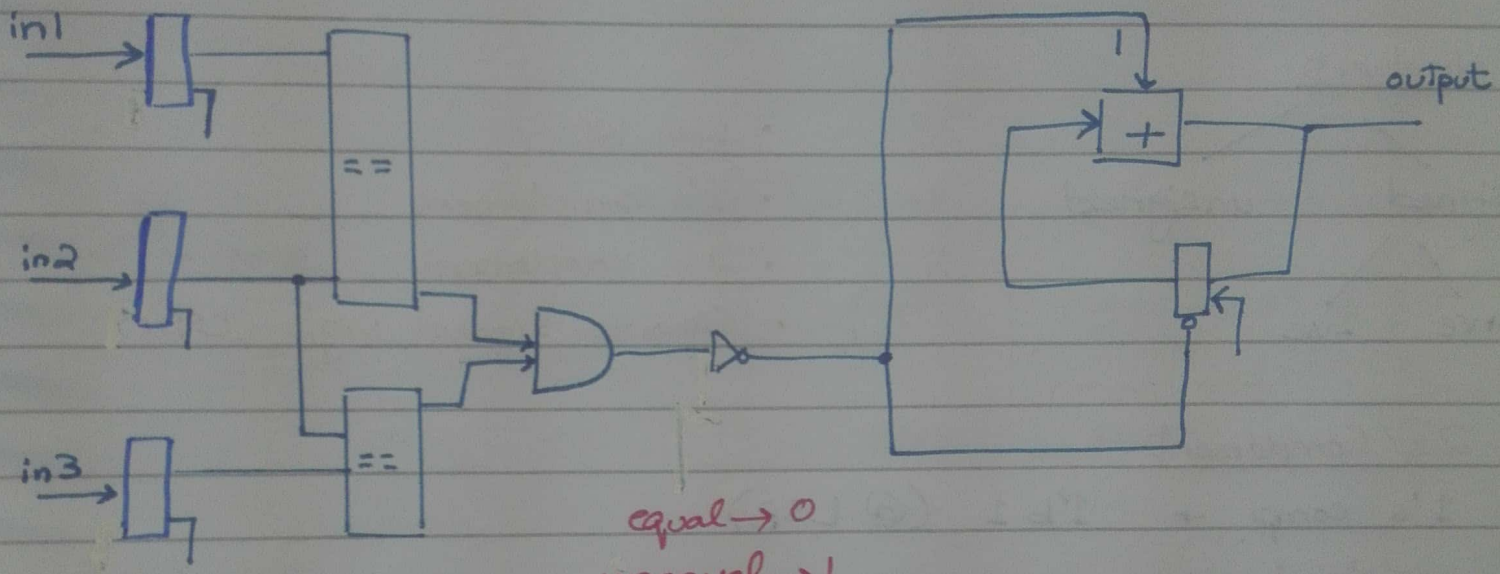


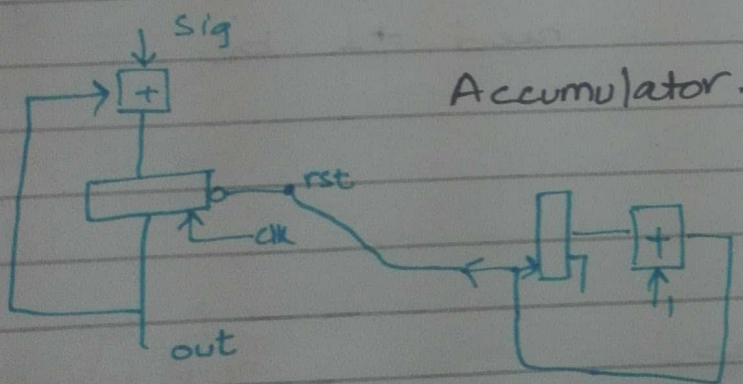

delay

# System Design

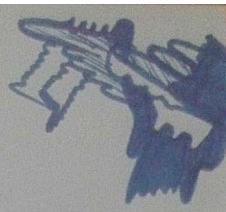
3 inputs N bits
1 output M bits




running sum (integrator)
add 4 samples, if > T Led On.




Accumulator. $\sum_{i=0}^{N-1} x_i$



# Chapter 3 Number representation in H/W and arithmetic

In hardware, real #s do not exist
decimal pt

decimal number formats
- fixed pt
- floating pt

## Numbers

- signed
  - +ve
  - -ve
- unsigned

- sign bit
- 1's complement
- 2's complement
- Canonic Signed Digit (CSD)

## 2's Complement.

1's comp + 1'b1 (@ LSB)

$-2^3\ 2^2\ 2^1\ 2^0$

0101 (5)
1011 (-5)
-8+2+1

101 (-3)
011 (3)

| | US | S (2's C) |
|---|---|---|
| 0 0 | 0 | 0 |
| 0 1 | 1 | 1 |
| 1 0 | 2 | -2 |
| 1 1 | 3 | -1 |

to achieve same range, need +1 bit

**US**
min → 0
max → $2^N - 1$

**Signed.**
min → $-2^{N-1}$
max → $2^{N-1} - 1$

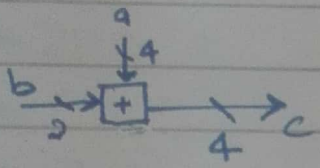
no. of bits = 3
-4 → 3

2's Complement $-2^2 2^1 2^0$

5 → 0101 101 → −3 not 5

if MSB 1, -ve
if MSB 0, +ve
] in 2's Complement

## Sign Extension




assign c = a+b OK for unsigned

for signed → assign c = a + { {2{b[1]}}, b }

replicate MSB and concatenate

compare
1 0 1 1 (-5)
1 0 1 1 (-5)
1 0 1 1 0 (-10)
6

to identify, compare MSB of inputs and output

underflow

compare
0 1 1 1 7
0 0 0 1 1
(-8) 1 0 0 0 8

overflow

{+, +} → − assign max (7)

{−, −} → + assign min (−8)

] error reduced

## Redundant Bits.

| | |
|---|---|
| 1 | -1 |
| 1 1 | -1 |
| 1 1 1 | -1 |

cant drop

0 1 0

drop 1

0 0 1 0

0 0 0 1 0

drop 2

## Floating Point Format

decimal pt ( real #'s)

IEEE format

| 1 | 8 | 23 |
|---|---|---|
| S | E | M |

$(-1)^S \times 1\_m \times 2^{e-b}$

0.75

x 2

(1).5

x 2

(1).0

-4.75

-100.11

Keep xplying till 0

(i) Binary

(ii) Scientific Notation $-1.0011 \times 2^2$

S = 1

m = 0011

e - b = 2

e = 2 + b

= 2 + 128 = 130

b = bias = $2^{N-1} - 1$

N = # of bits for exponent

bias added so exp always +ve

| 1 | 10000010 | 00110000 |
|---|---|---|

$(-1)^1 \times 1.0011 \times 2^{130-128}$

$= 1.0011 \times 2^2$

$2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}$

$= 100.11$

$-4.75$

$-4.75 \times 0.5$

$(-1)^1 \times 1.0011 \times 2^2$

$1.0 \times 2^{-1}$

sign — mantissa xplied — powers added

10 bit format.

| | | | |
|---|---|---|---|
| a = | 1 | 0111 | 10100 |
| b = | 0 | 1001 | 01000 |

**Step 1.**

$S_{ans} = S_a \otimes S_b$

$= 1 \otimes 0 = 1$

**Step 2.**

$2^{e-b} \rightarrow e = \# + b$

$e_{ans} = e_a + e_b - bias$

$= 7 + 9 - 7$

**Step 3.**

$a_m$ 1_101

$b_m$ 1_01

```
   1 101
  00 00X
 1 1 0 1XX
1 0.0 0 01
```

## Step 4.

normalize

$1.000001 \times 2^1$

$e_{ans} = 9 + 1 = 10$

## Step 5.

$m_{ans} = 000001$

truncate

→ 00000

| 1 | 1010 | 00000 |
|---|---|---|

if exponent overflow

| | | | |
|---|---|---|---|
| inf | 0 | 1111 | 00000 |
| -inf | 1 | 1111 | 00000 |

## Floating Point Addition

**Step 1** $a_m$ 1_1001 append 1 with mantissa

$b_m$ 1_01

**Step 2.**

identify smaller # $e_a = 7$

$e_b = 9$

shift smaller # to greater exponent diff = 2

$a_m >> 2$ $a_{new} = 0\_011001$

## Step 3

b - a
b + (-a)
b + (~a + 1)
2's comp.

Take 2's complement of -ve number

$a_{new} = 0\_011001$
$a_{new}2 = 1\_100111$

## Step 4

$b_m + a_{new}2$

1_100111
1_01

align point
Sign extension

```
 1 1_100111
 0 1_010000
10_1101111
```

ignore carry

100_110111

## Step 5

Take 2's comp if ans is -ve

## Step 6

normalize $1\_10111 \times 2^{-1}$

$e_{ans} = 9 + (-1) = 8$

## Step 7.

truncate

# floating pt addition

a  1 1011 00100
b  0 1001 01010

## Step 1

$a_m$  1_00100
$b_m$  1_01010

## Step 2

$e_a = 11$
$e_b = 9$
$e_a - e_b = 2$
$S_{ans} = 1$

$b_m \rightarrow 2$  $b_{new} = 0\_010101$

## Step 3

$a_m$ 01_001  if MSB is 1, append 0  take 2's comp of -ve so we can add
$a_{new}$ 10_111 → signed (2's c)

## Step 4

$a_{new} + b_{new}$

0 0_01 01 01
10_111
11_00110 1

if extension of signed number, replicate MSB
if extension of unsigned, replicate 0

## Step 5.

2's c of answer  00_110011

Step 6

$1\_10011 \times 2^{-1}$

$e_{ans} = 11 - 1 = 10$

Step 7

$m_{ans}$ 10011

1 1010 10011

# Fixed Point

- location of decimal is fixed
- decimal pt is virtual

$Q_{n.m}$ format

n = # of bits for integer

m = # of bits for fraction

x_xx $Q_{1.2}$

Conversion. (# into fixed pt format $Q_{n.m}$)

$\#_{fixed} = round(real\ \# \times 2^m)$

$\#_{real} = 2.35$

save as $Q_{2.4}$

n=2, m=4

$round(2.35 \times 2^4)$

$round(37.6) = 38$

real # fixed pt

2.35 38 $Q_{2.4}$ $38/2^4 = 2.375$

10_0110

$2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$

$\approx 2.375$

Conversion (fixed pt → real #)

fixed pt / $2^m$

## format Selection

process using 6 bits

$n+m=6$

① Choose 'n' 2.35 integer ⇒ 2.

$n=2$ $m=6-2=4$

$x = [-2.35 \quad 3.15 \quad 5.1]$

max : 5.1 min : -2.35

n = 3 + (1) to accomodate 2's complement

max (abs (x)) + 1 if any number -ve.

Real # → Fixed Pt

$\#_{fixed\ pt} = round(\#_r * 2^m)$

Fixed pt → real #

$\#_2 = \#_{fixpt} / 2^m$

Addition. $a + b$

$Q_{1.3} + Q_{3.1}$

$a = 1011$ $b = 0101$

1 1 1 _ 0 1 1

0 1 0 _ 1 0 0

1 0 0 1 _ 1 1 1

↑ ignore

if signed number, check MSB. if MSB 1, -ve

1 1 1 _ 0 0 1

1 0 0 _ 1 0 0

1 0 0 0 1 0 1

assign c1 = a+b

if (OF)

c = 2 ** (N-1) - 1

else if (UF)

c = -2 ** (N-1)

else

c = c1

OF / UF = carry into MSB ⊗ carry out of MSB

if carry in != carry out then OF / UF

$\{C_{in}, C_1\} = a_1[4:0] + b_1[4:0]$

$\{C_{out}, C_2\} = a_1[5] + b_1[5] + C_{in}$

$x = [5, 6, -4]$

$acc = x + acc$

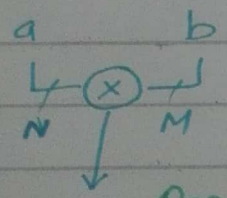
① $acc = 5 + 0 : 5$

② $acc = 6 + 5 : (11)$

```
  0 1 0 1
  0 1 1 0
  -------
  1 0 1 1
  1 1 0 0
  -------
[1] 0 1 1 1
 X
```

if final is within range, ignore intermediate OF's

## Fixed Point Multiplier




$Q_{n1.m1}$ $Q_{n2.m2}$

U/S U/S

| | | | Verilog | Real |
|---|---|---|---|---|
| U x U | 1_011 | $Q_{1.3}$ | 11 | 11/8 |
| | 1_101 | $Q_{3.1}$ | 13 | 13/2 |
| | 0 0 0 0 1 1 0 1 | | | |
| | 0 0 0 0 0 0 0 0 X | | partial products | |
| | 0 0 1 0 1 1 X X | | | |
| | 0 1 0 1 1 X X X | | | |
| | 1 0 0 0 1 1 1 1 | $Q_{4.4}$ | | |

## S x U

```
S   1_011      Q1.3      -5
U   1101       Q3.1      13

1 1 1 1 1 0 1 1     MSB replication
0 0 0 0 0 0 0 x
1 1 1 0 1 1 x x
1 1 0 1 1 x x x
-----------------
1 0 1 1 1 1 1 1
```

10
X

## U x S

```
U   1101        13
S   1011        -5

0 0 0 0 1 1 0 1     0 replication because PPs are +ve
0 0 0 1 1 0 1 x
0 0 0 0 0 0 x x
1 0 0 1 1 x x x
-----------------
1 0 1 1 1 1 1 1
```

0 1 1 0 1
1 0 0 1 1
Take 2's comp for last P.P (if MSB 1 only)
Add guard bit

## S x S

```
S   1101        -3
S   1011        -5

Φ Φ Φ Φ 1 1 0 1     MSB replication
Φ Φ Φ 1 1 0 1 x
0 0 0 0 0 0 x x
0 0 0 1 1 x x x
-----------------
0 0 0 0 1 1 1 1
0 0 0 1 1 1 1 0
```

Take 2's c without guard bit

drop

15 $15/2^4$

30 $30/2^5$

### Corner Case

for SxS left shift by 1

$Q_{4.4} << 1$

$Q_{3.5}$

# Corner Case.

5 x 5

min x min

| | | |
|---|---|---|
| 1.000 | -8 | -1 |
| 1 00.0 | -8 | -4 |
| 0100.0000 | 64 | 4 |
| 100.000000 | after shift number becomes -ve which is wrong | |

dont shift if both min

round before truncate

add 1 in bit to be truncated, then truncate

# Representation of Algorithms

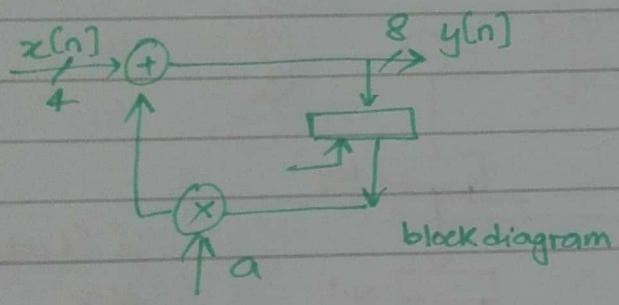
Block Diagram → [op] [k]

Signal Flow Graphs → → $z^{-1}$ gain →

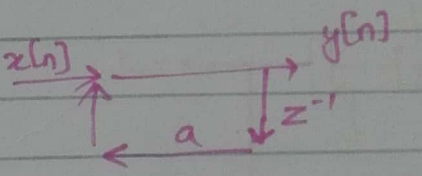
Data Flow Graphs  nodes & edges

(A) →

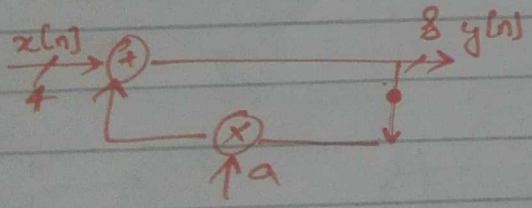
$y[n] = x[n] + y[n-1] * a$




block diagram




signal flow diagram




speed (f system)

f system = = fs  one to one mapping
f sys > fs  Time shared architecture / folded architecture
f sys < fs → pipelining
resource optimization
parallelism

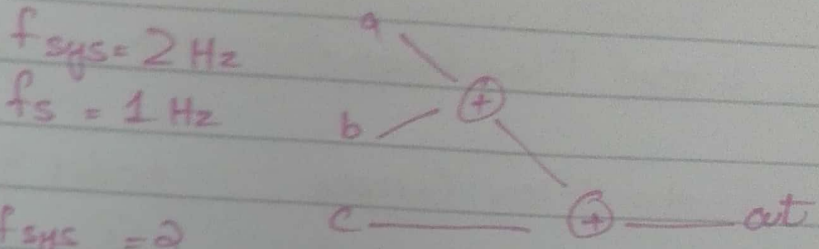
# Time Shared Architecture
(fsys > fs)

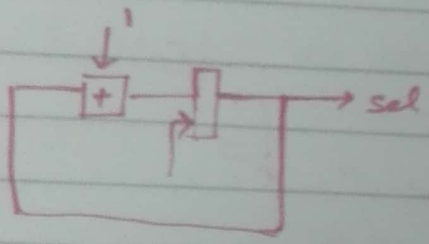
- Sequential
- Bit Serial
- Byte Serial
- Systolic Arch

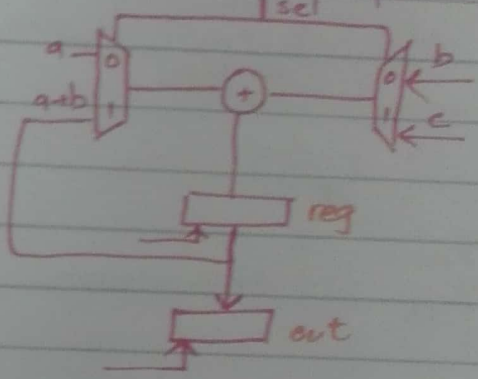
## Sequential Arch.

fsys = 2 Hz
fs = 1 Hz

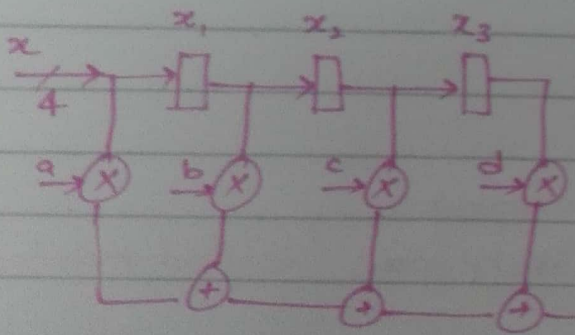
$\frac{f_{sys}}{f_s} = 2$ (# of cycles available per sample)



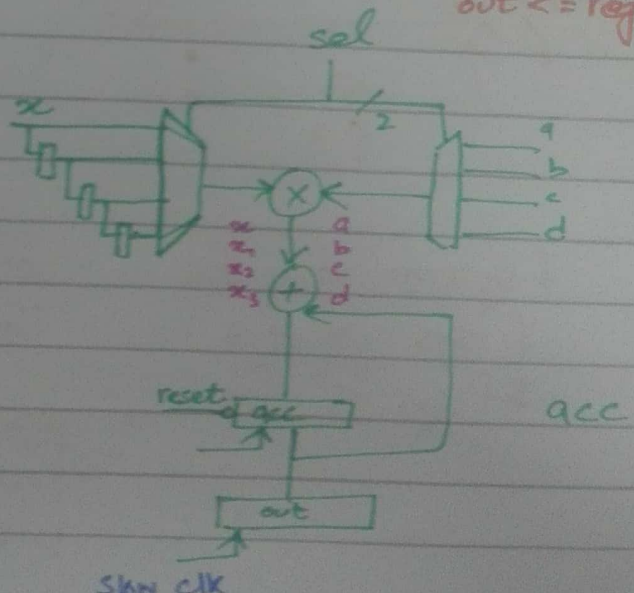




```
if (!rst)
else
  if (sel==0)
    out <= reg
```



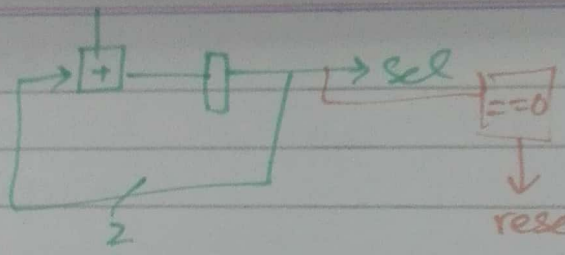

$\frac{f_{sys}}{f_s} = \frac{4}{1} = 4$

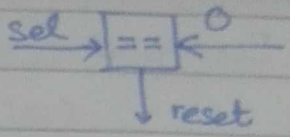
$x * a + x_1 * b + x_2 * c + x_3 * d$




acc

4 times slower than clk




reset = ~(sel==0) reset acc on 5th cycle

if (sel==0)

out <= ~~reg~~ acc

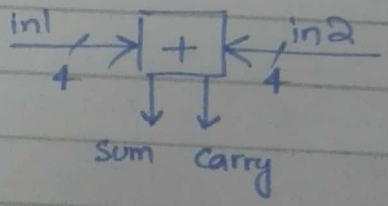
| sel |
|---|
| 0 0 |
| 0 1 |
| 1 0 |
| 1 1 |

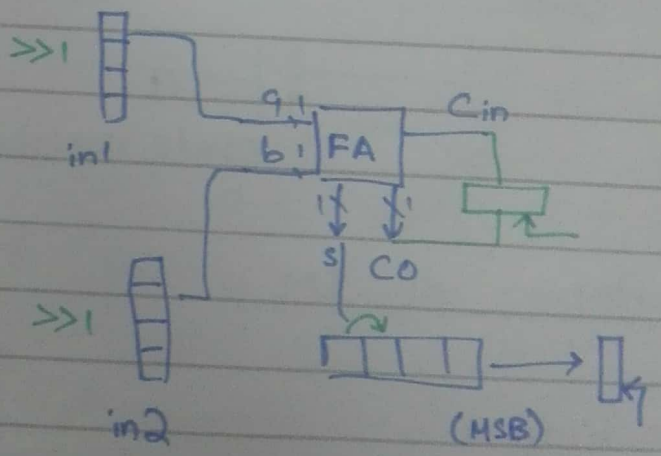
slow clk = ~ sel[1]

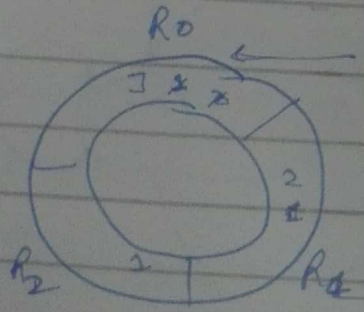
or

slow clk = ~ reset

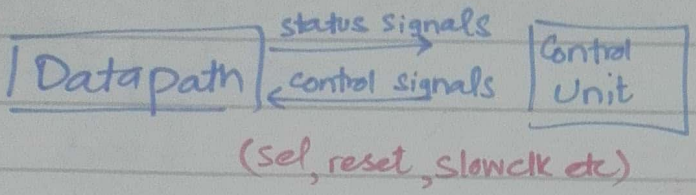
## Bit Serial




A = -100



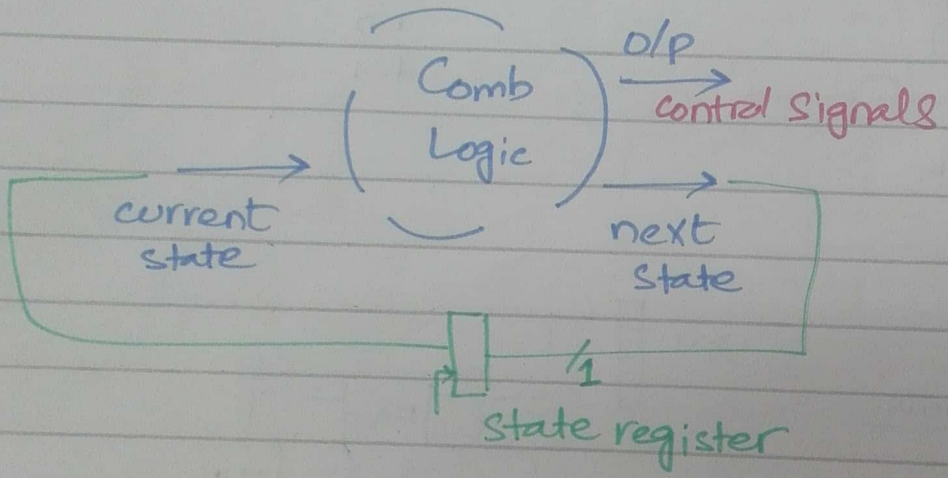

→ State machines.

an alternate implementation for control unit

considers systems as states

0 $S_0 = a+b$ sel=0

1 $S_1 = a+b+c$ sel=1



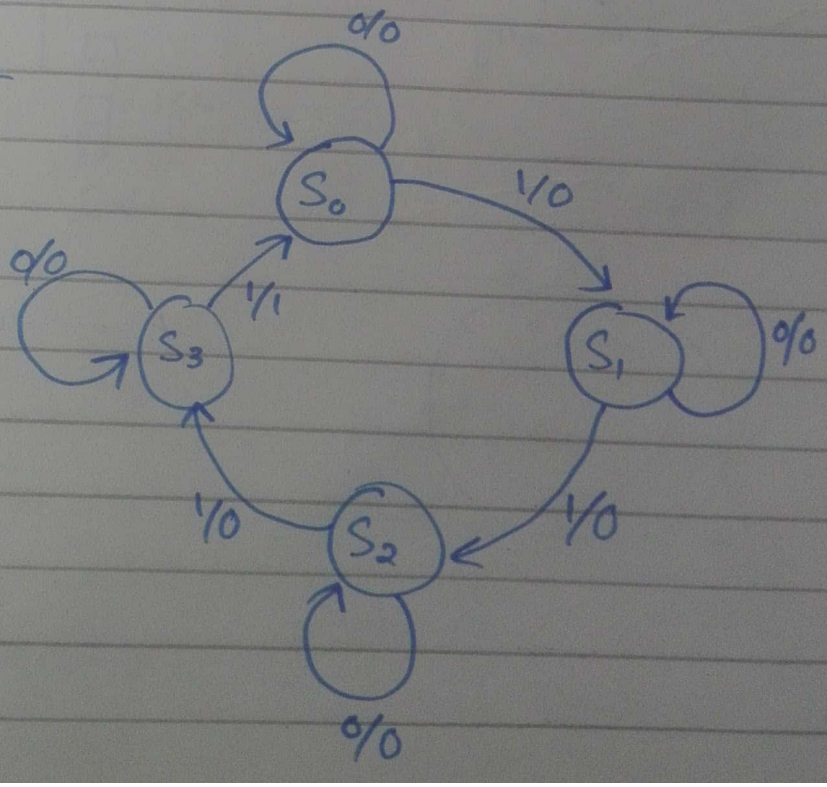

Representation of SM

Bubble Diagram

○ states

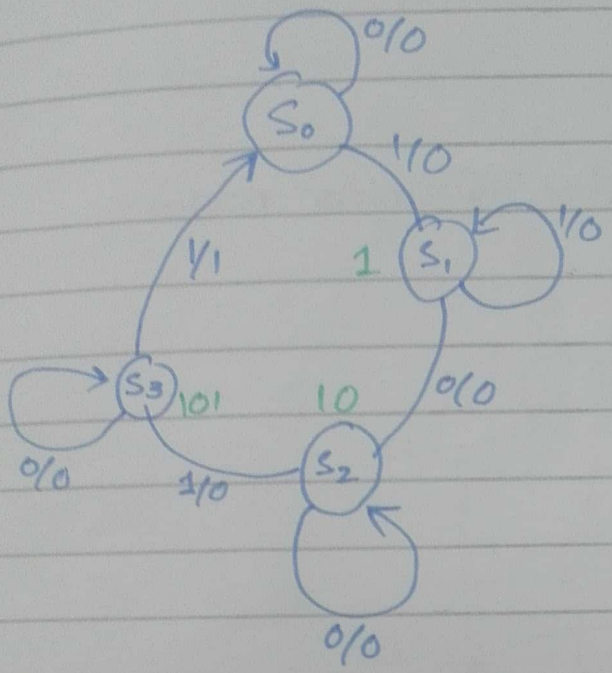
→ transition

in/out



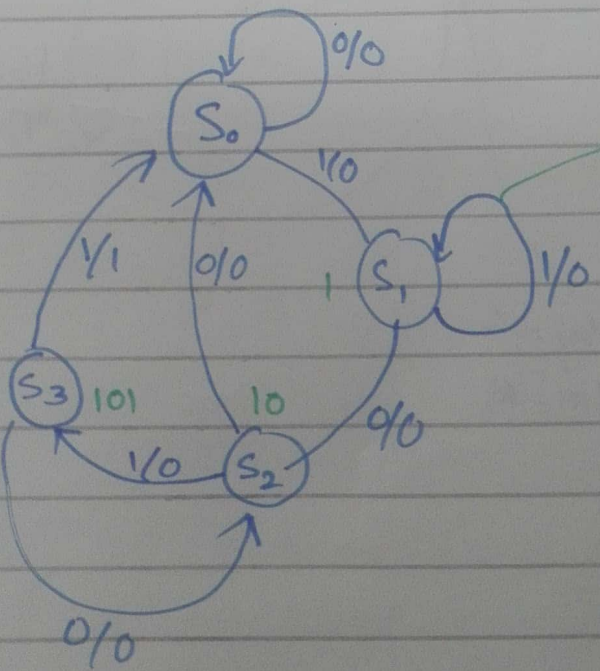
# FSM.

1011 1 bit input O/P 1 when above pattern is recognized.

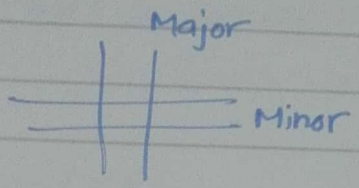



Sliding window

For consecutive 1011



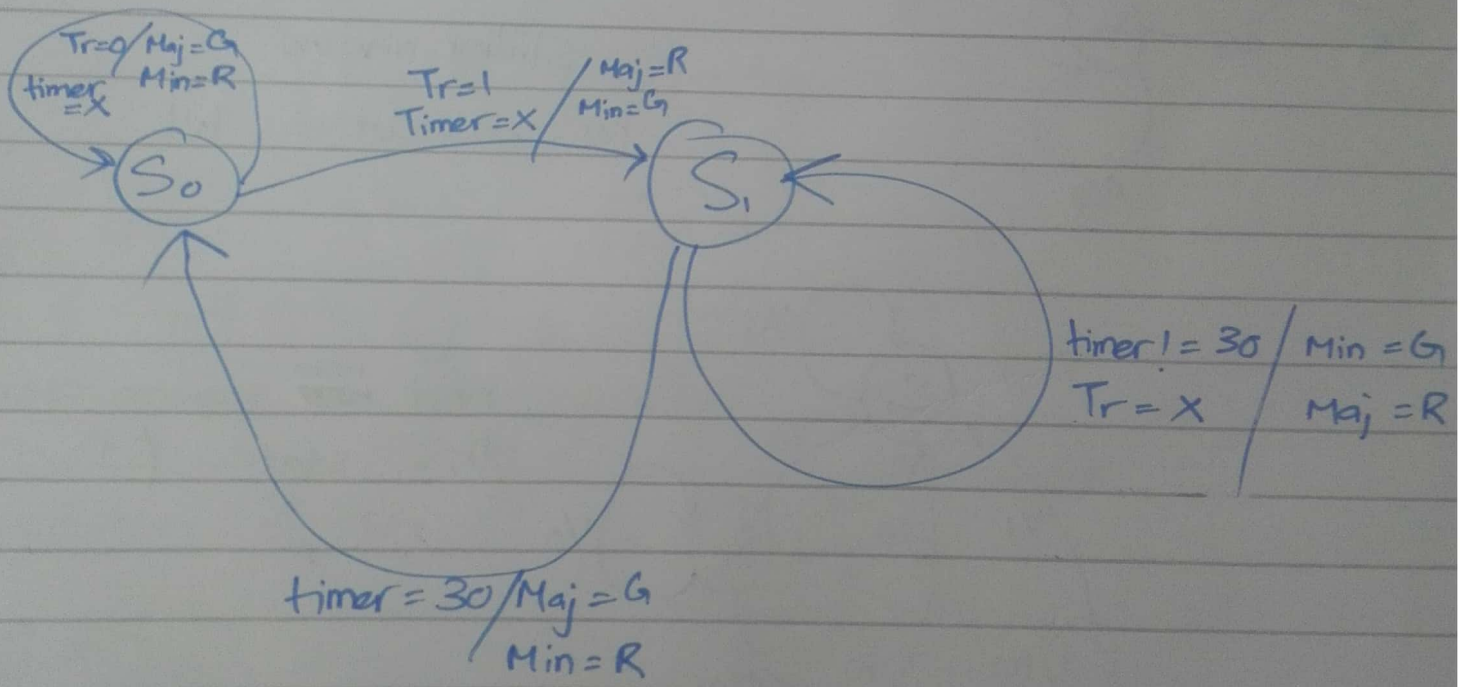
# Traffic Controller

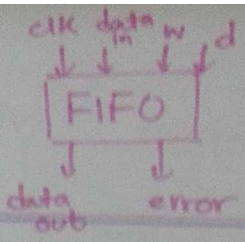



i) at all times major road green
ii) 30 sec minor green when traffic @ minor road

**Inputs**
Tr, timer

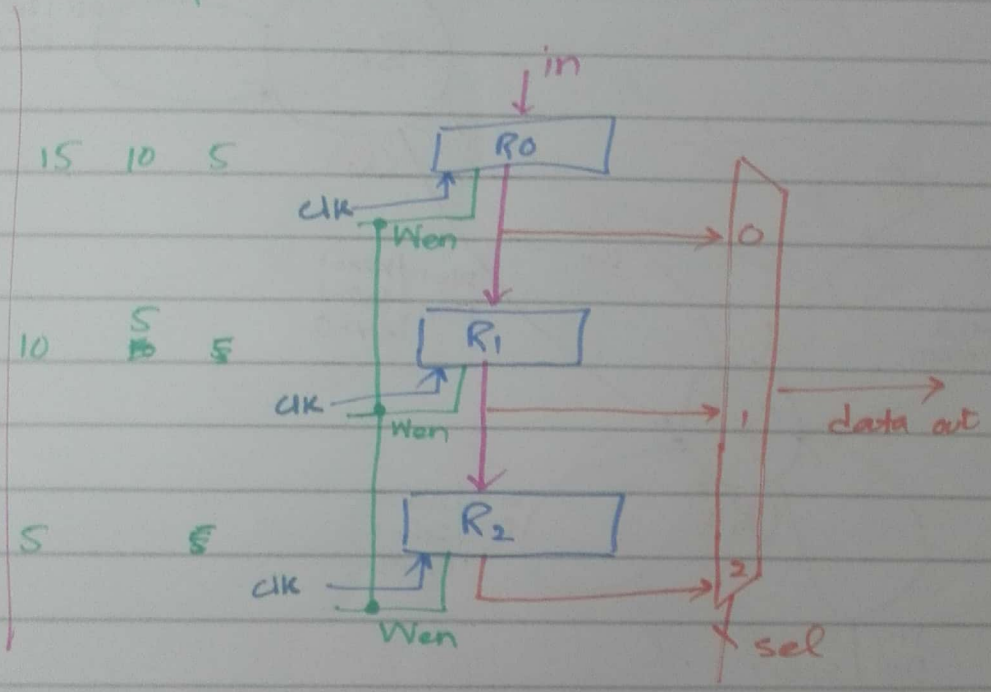
**Outputs**
Maj, Minor





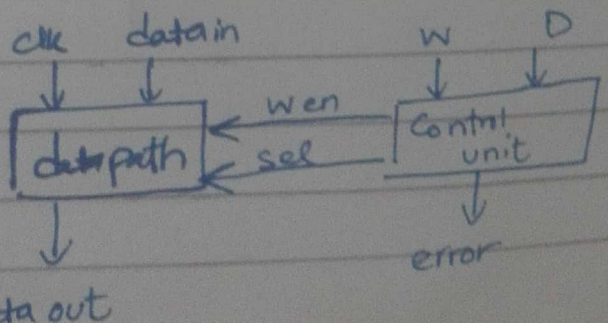

Design datapath & FSM based controller for 3 entry FIFO with following traits

- FIFO has 2 i/p signals (W & D) for write and delete
- FIFO gives error @ following
  - delete from empty
  - write in full fifo
  - write & delete @ same time

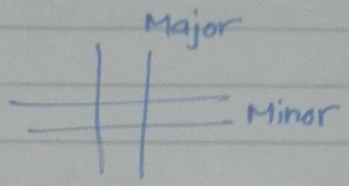



| | | | | | | |
|---|---|---|---|---|---|---|
| D | | | 1 | 1 | 0 | |
| w | 1 | 1 | 1 | 0 | 0 | 1 |
| s | 0 | 1 | 2 | 1 | 0 | 1 |

on w, s++
on d, s--



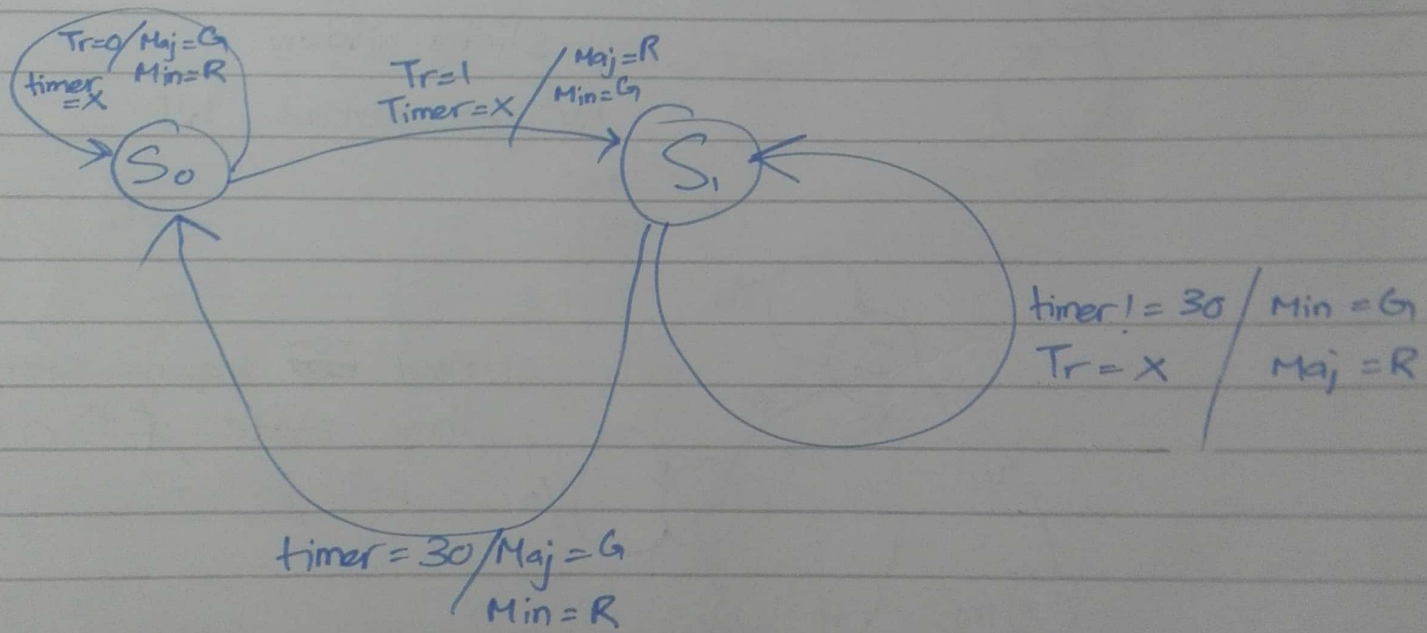
# Traffic Controller

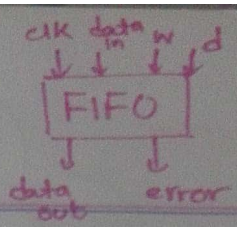



i) at all times major road green
ii) 30 sec minor green when traffic @ minor road

**Inputs**
Tr, timer

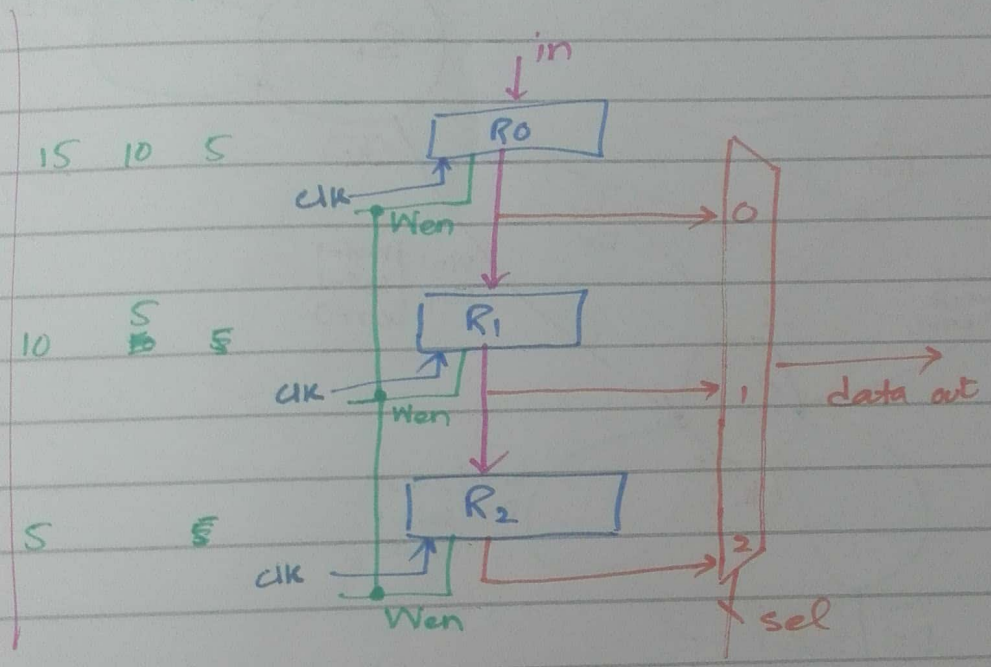
**Outputs**
Maj, Minor





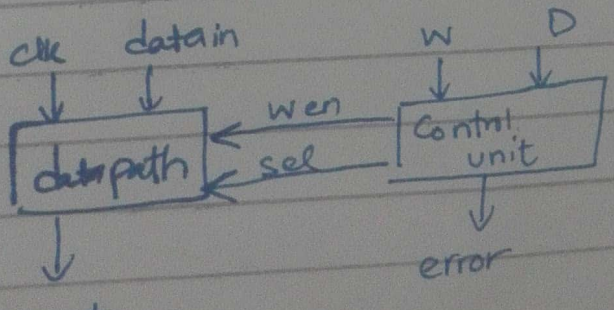

Design datapath & FSM based controller for 3 entry FIFO with following traits (1 bit)

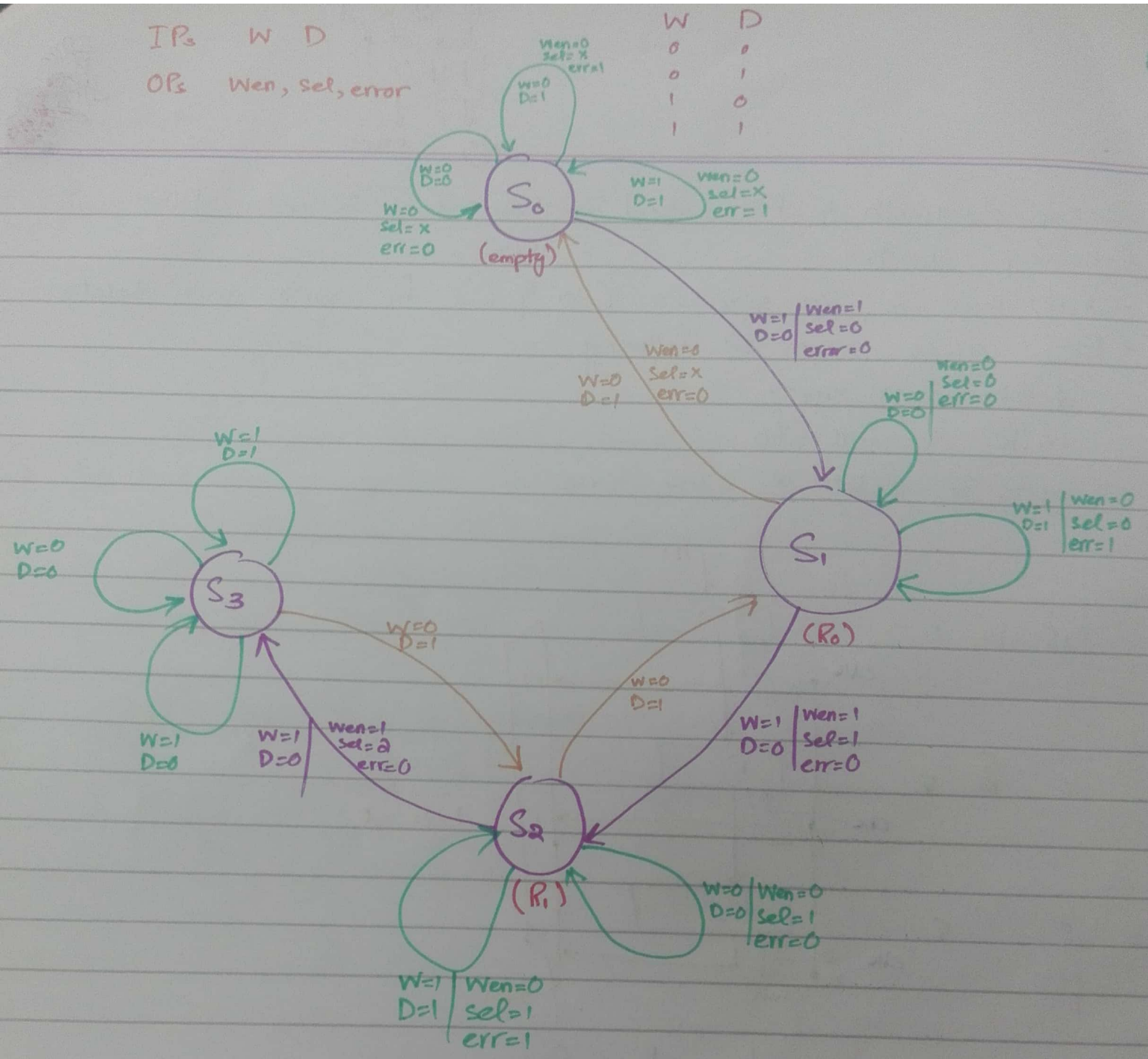
- FIFO has 2 i/p signals (W & D) for write and delete
- FIFO gives error @ following
  - delete from empty
  - write in full fifo
  - write & delete @ same time




| | | | | | | |
|---|---|---|---|---|---|---|
| D | | | 1 | 1 | 0 | |
| W | 1 | 1 | 1 | 0 | 0 | 1 |
| S | 0 | 1 | 2 | 1 | 0 | 1 |

on w, S++
on d, S--



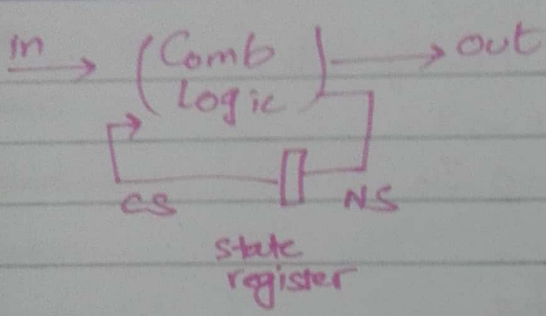



## State Machines.

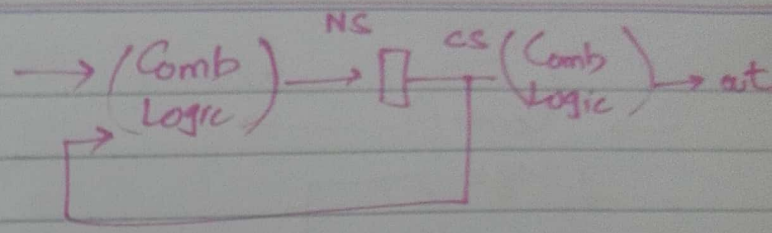
Types:

- Mealy Machine : O/P depends on I/P & CS
- Moore Machine : O/P depends on CS only

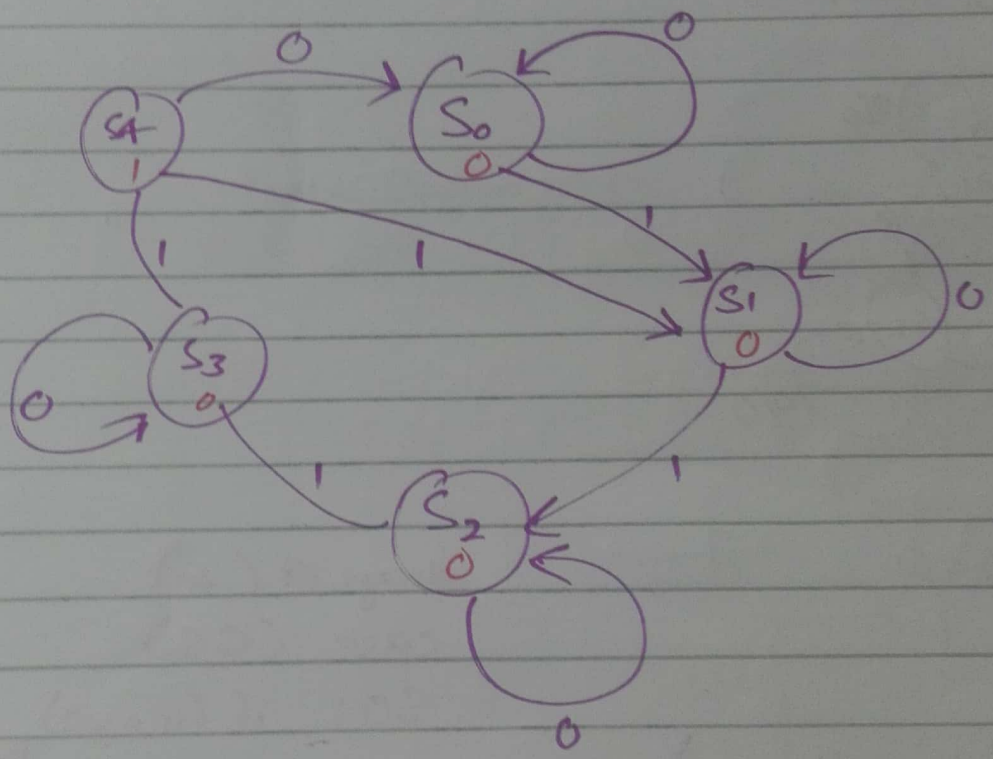
mealy



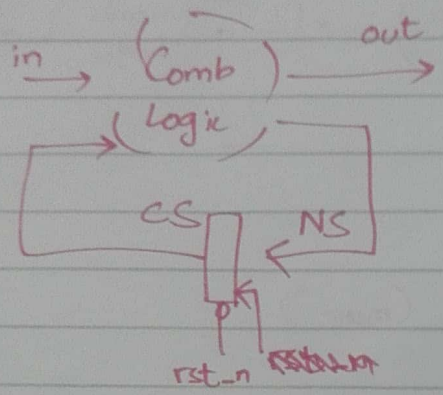

moore

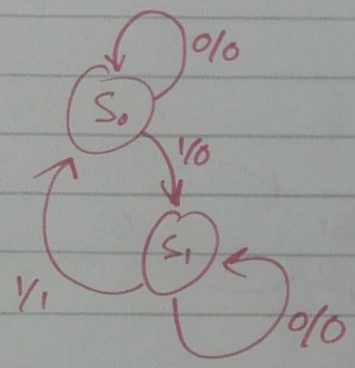



1111 identification (moore)



## Coding of SM.



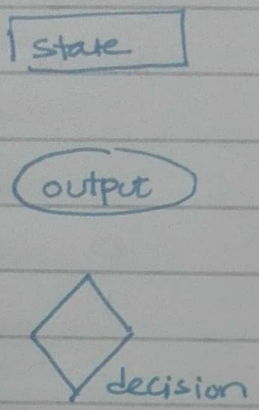



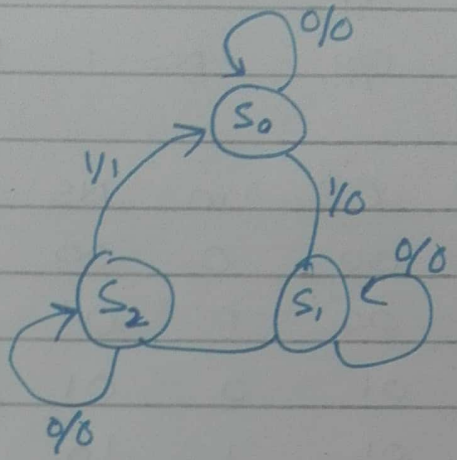

```
parameter S0 = 0 ;
          S1 = 1 ;
          S2 = 2 ;
          S3 = 3 ;
reg [1:0] NS, CS ;
always@(posedge clk
if (!rst_n)
  CS <= 0 S0
else
  cs <= ns
```

```
Parameter S0 = 0
          S1 = 1
reg NS, CS
always@(posedge clk or..)
if (! rst_n)
 cs <= S0;
else
cs <= nS;
```

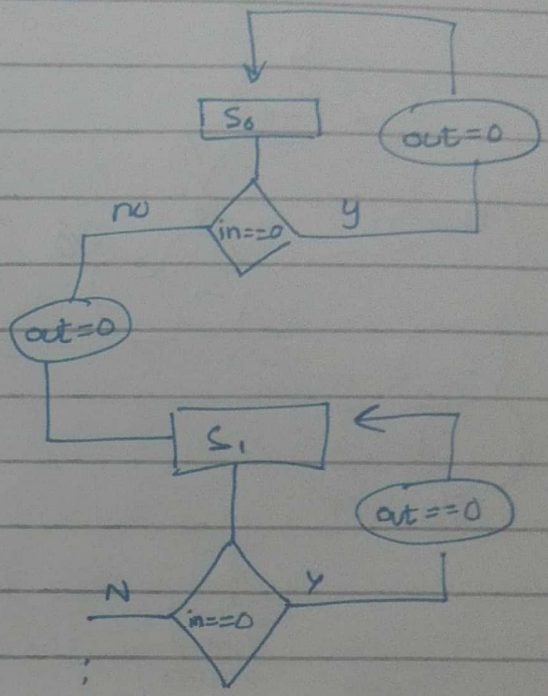
```
always @ (*)
 case (CS)
  S0: if (in==0)
        NS = S0      begin
        out=0        end
      else
        NS = S1      begin
        out = 0      end

  S1: if (in==0)
        NS = S1      begin
        out = 0      end
      else
        NS = S0      begin
        out = 1      end
```

# Algorithmic State Machine (ASM chart)

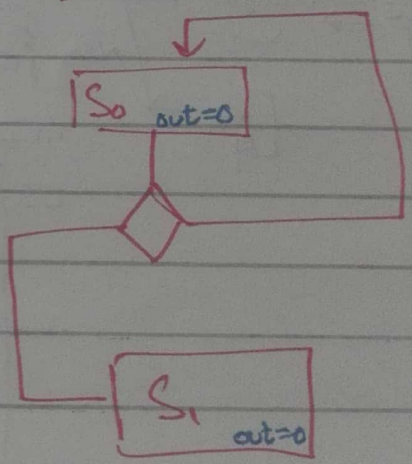


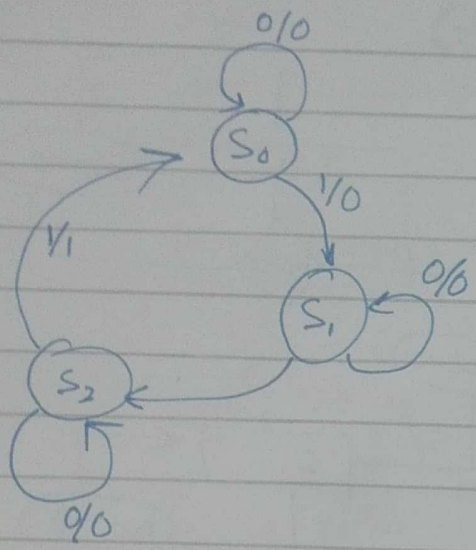

Bubble Diagram




for moore;



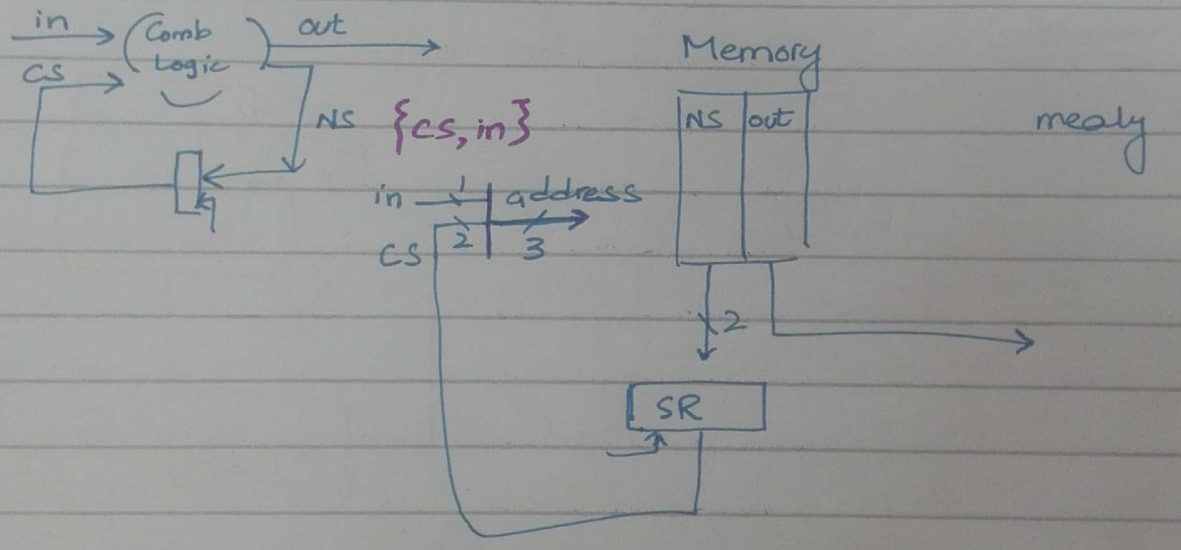
# Chap 10. Microcoded/Microprogrammed SM

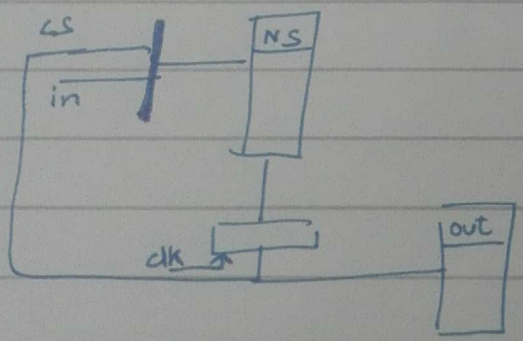



i/p CS : 1 bit in, 2 bit CS

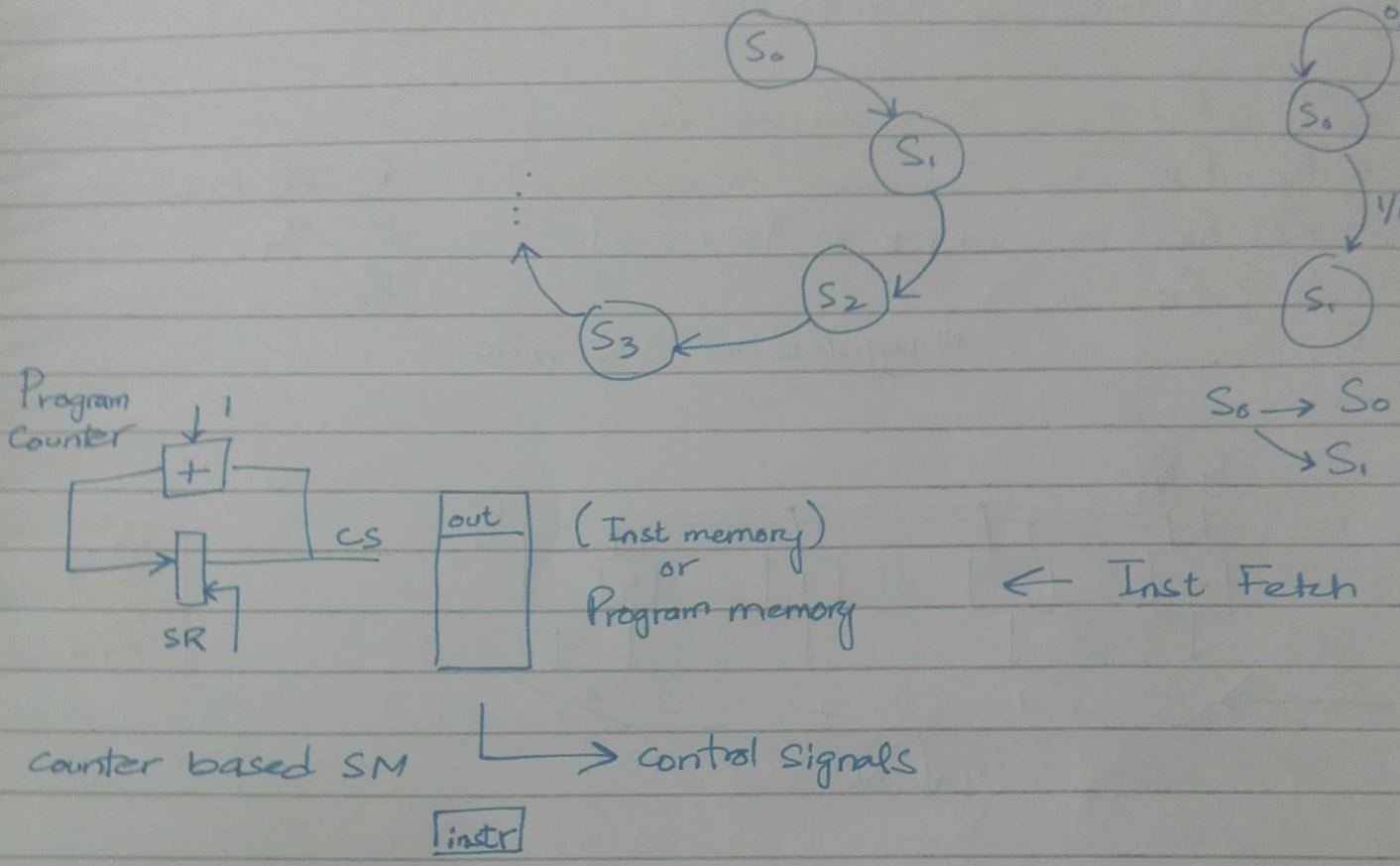
o/p NS : 1 bit at, 2 bit NS

| CS | in | NS | out | |
|---|---|---|---|---|
| 00 | 0 | 00 | 0 | 0 |
| 00 | 1 | 01 | 0 | 1 |
| 01 | 0 | 01 | 0 | 2 |
| 01 | 1 | 10 | 0 | 3 |
| 10 | 0 | 10 | 0 | |
| 10 | 1 | 00 | 1 | |




moore




## Custom Processor Design

ISA → inst set architecture

2 types. ALU Based (add, sub, shift etc) and Control (Branch, Loop etc)