A SIMPLE PROCESSOR

20.05.2010

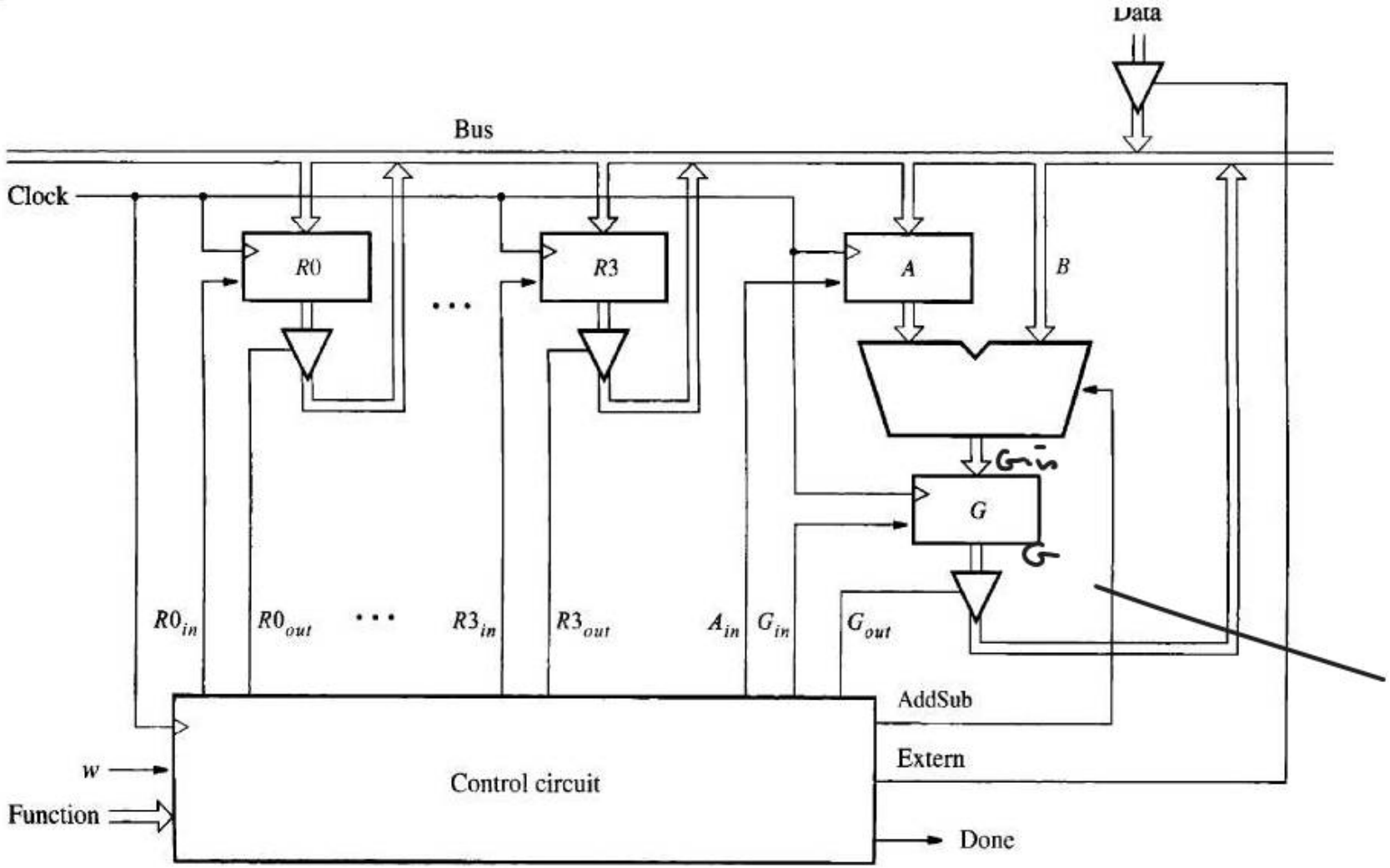


Figure 7.69 A digital system that implements a simple processor.

When w=1 operation starts.

W=0 no operation.

Chapter 6 in the book:

1, 2, 3, 4, 5, 6, 7, 8, 11, 18, 19, 21, 22, 23, 24, 25, 28, 31

Chapter 7 in the book:

9, 10, 13, 14, 16, 17, 18, 34

	Operation	Function Performed
1	Load *Rx*, *Data*	$Rx \leftarrow Data$
2	Move *Rx*, *Ry*	$Rx \leftarrow [Ry]$
3	Add *Rx*, *Ry*	$Rx \leftarrow [Rx] + [Ry]$
4	Sub *Rx*, *Ry*	$Rx \leftarrow [Rx] - [Ry]$

3 clock cycles

1 2 3

$R_X = R_0, R_1, R_2, R_3 \cdots$

$R_Y = R_0, R_1, R_2, R_3 \cdots$

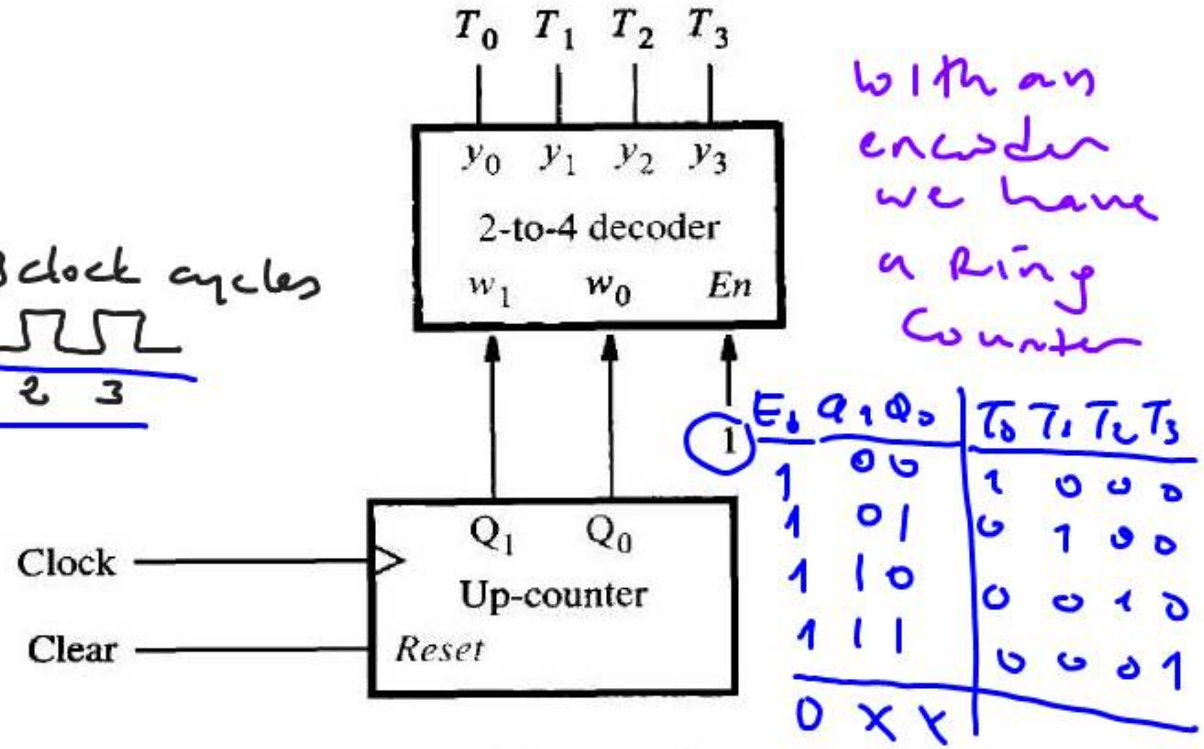
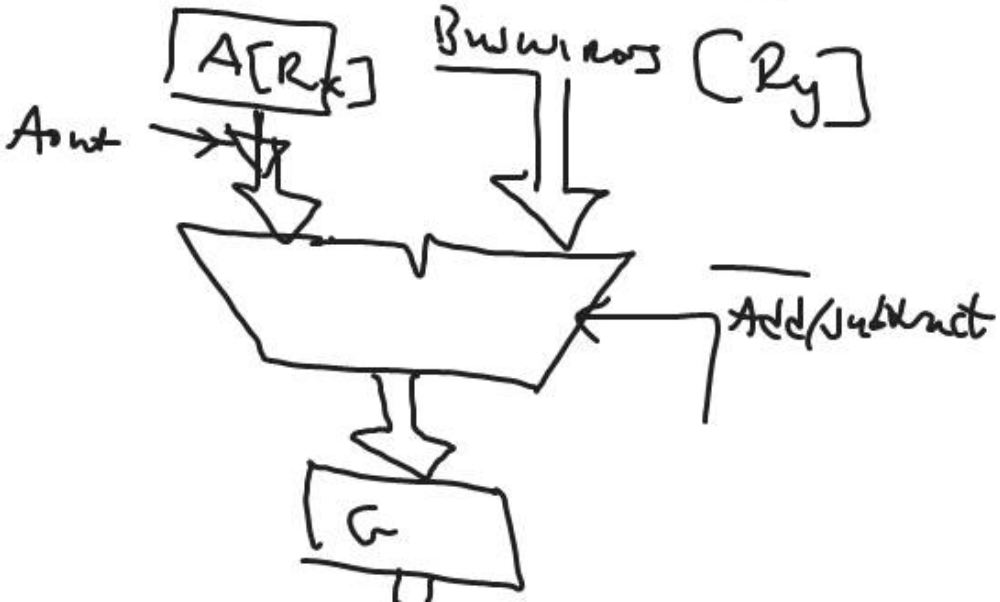


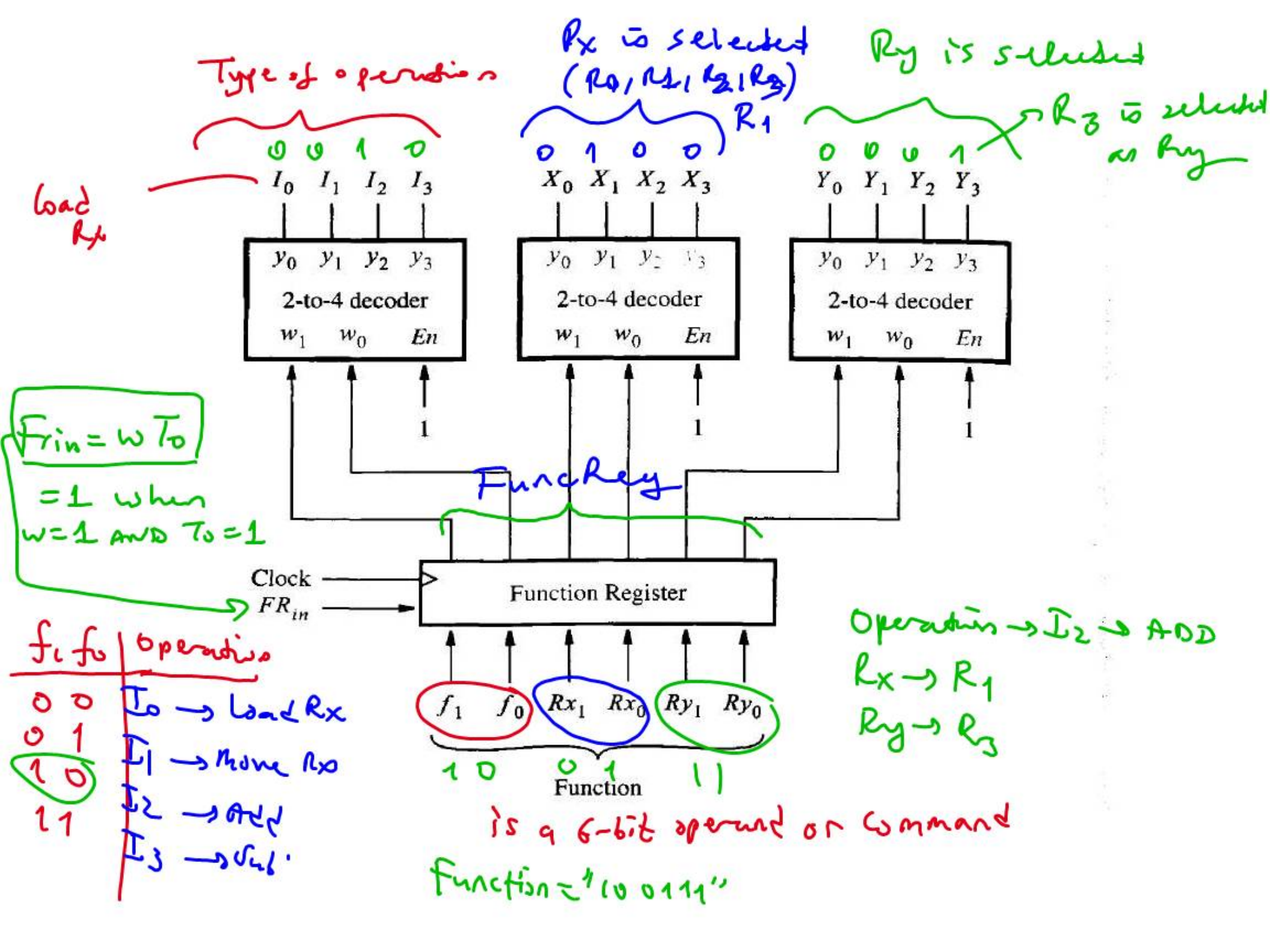
Figure 7.70 A part of the control circuit for the processor.

Sub Rx, Ry ⇒

- 1st step : T0 : [Rx] are transferred to register A.
- 2nd step T1 : [Ry] is placed onto the bus (BUS WIRES). The adder/subtractor module performs the subtraction and the result is stored in register G
- 3rd step T2 : [G] are transferred into Rx.



- Function → Specifies the operation to be performed at any given time
- When w=1 this operation is initiated
- The control circuit asserts the **Done** output when the operation is completed.


Type of operation
Rx is selected (R0, R1, R2, R3)
R1
Ry is selected
R3 is selected as Ry
0 0 1 0
0 1 0 0
0 0 0 1
Load Rx
I0 I1 I2 I3
X0 X1 X2 X3
Y0 Y1 Y2 Y3
y0 y1 y2 y3
2-to-4 decoder
w1 w0 En
1
Frin = w T0
=1 when w=1 AND T0=1
FuncReg
Clock
FRin
Function Register
f1 f0 Rx1 Rx0 Ry1 Ry0
1 0 0 1 1 1
Function
f1 f0 | operation
0 0 | I0 → Load Rx
0 1 | I1 → Move Rx
1 0 | I2 → Add
1 1 | I3 → Sub
Operation → I2 → ADD
Rx → R1
Ry → R3
is a 6-bit operand or command
Function = "10 0111"

$$R_0in = (I_0+I_1)T_1X_0 + (I_2+I_3)T_3X_0$$

$$R_0out = (I_2+I_3)T_1X_0 + I_1T_1y_0 + (I_2+I_3)T_2y_0$$

$$R_out = (I_2+I_3)(T_1x_0+T_2y_0) + I_1T_1y_0$$

$$Extern = I_0T_1$$

$T_0 = 1$ no operation

$$Done = I_0T_1 + I_1T_1 + I_2T_3 + I_3T_3$$

$$Done = (I_0+I_1)T_1 + (I_2+I_3)T_3$$

Table 7.3 Control signals asserted in each operation/time step.

T_1 (1st Step)	T_2 (2nd Step)	T_3 (3rd Step)
Extern, $R_{in} = X$, *Done*		
$R_{in} = X$, $R_{out} = Y$, *Done*		
$R_{out} = X$, A_{in}	$R_{out} = Y$, G_{in}, $AddSub = 0$	G_{out}, $R_{in} = X$, *Done*
$R_{out} = X$, A_{in}	$R_{out} = Y$, G_{in}, $AddSub = 1$	G_{out}, $R_{in} = X$, *Done*

f_1 f_0

0 0 → (Load): I_0

0 1 → (Move): I_1

1 0 → (Add): I_2

1 1 → (Sub): I_3

$$A_{in} = (I_2+I_3)T_1$$

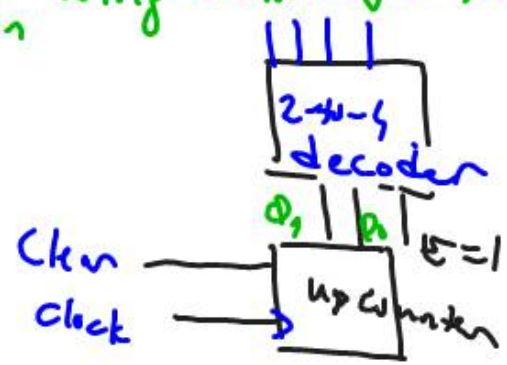
$$G_{in} = (I_2+I_3)T_2$$

$$G_{out} = (I_2+I_3)T_3$$

$$AddSub = I_3T_2$$

E_1	Q_1	Q_0	T_0	T_1	T_2	T_3	
1	0	0	1	0	0	0	No operation
1	0	1	0	1	0	0	1st step
1	1	0	0	0	1	0	2nd step
1	1	1	0	0	0	1	3rd step

← Ring Counter for timing $T_0T_1T_2T_3$



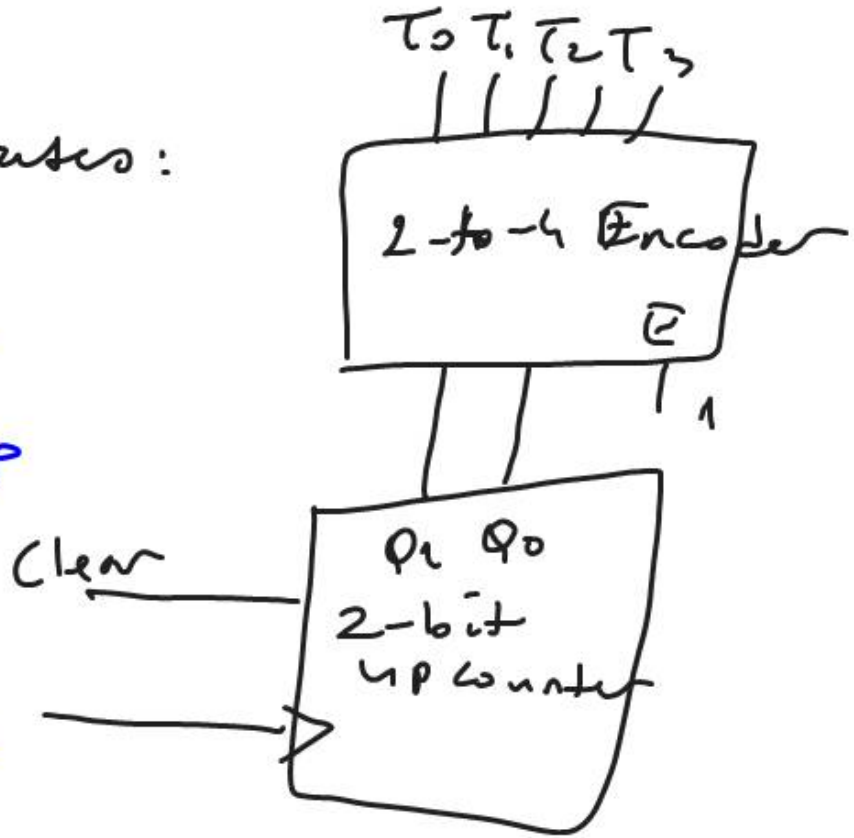
The Control circuit generates:

- Extern
- Done
- Ain
- Gin
- Gout
- Add Sub

FRin = 1

FRin = W T0 *

at the 0'th step the contents of the Function is loaded into the Control register.



- R0in, R1in, R2in, R3in
- R0out, R1out, R2out, R3out
- Clear ⟶ when Clear = '1' Count remains at "00"
- FRin

when W = 1

FRin = W T0

* as long as W = 0, no operation is executed.

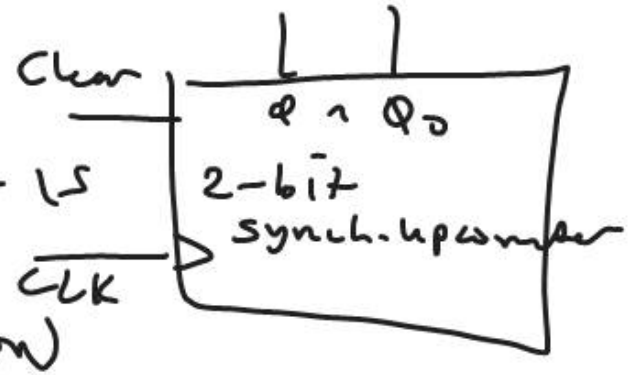
Clear = $\overline{W}$ T0 + Done

VHDL Code for the Processor:

2 bit up-counter for synchronous Reset:

```
USE ieee. Std_logic - unsigned. all
ENTITY upcount is
PORT( Clear, Clock : IN  STD_LOGIC ;
      Q : BUFFER STD_LOGIC_VECTOR( 1 DOWN TO 0));
END upcount;

ARCHITECTURE Behavior OF upcount IS
BEGIN
upcount: PROCESS (Clock)
BEGIN
IF ( Clock'EVENT AND Clock = '1' ) THEN
IF Clear = '1' THEN
  Q <= "00" ;
ELSE
  Q <= Q + 1 ;
END IF ;
END IF ; END Behavior
```

Clear is synchronous



regn
tris
upcount
dec2to4

} to be used as components in the main code:
= subccts

is the work file where these components reside

MAIN CODE for the PROCESSOR:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
USE work.subccts.all;
ENTITY proc IS
PORT( Data: IN STD_LOGIC_VECTOR(7 DOWN TO 0);
      Reset, W: IN STD_LOGIC;
      Clock : IN ;
      I, Rx, Ry : IN STD_LOGIC_VECTOR(1 DOWN TO 0);
```

2bit 2bit 4bit
Function = I & Rx & Ry
6-bit

8-bit data bus

```
Done : BUFFER STD_LOGIC;
BUSWIRES : INOUT STD_LOGIC_VECTOR( 7 DOWN TO 0));
END Proc;
ARCHITECTURE Behavior OF Proc IS
SIGNAL Rin, Rout : STD_LOGIC_VECTOR( 0 TO 3);
SIGNAL Clear, High, AddSub : STD_LOGIC ;
SIGNAL Count, Zero : STD_LOGIC_VECTOR( 1 DOWN TO 0);
SIGNAL T, I, X, Y : STD_LOGIC_VECTOR( 0 TO 3);
SIGNAL R0, R1, R2, R3 : STD_LOGIC_VECTOR( 7 DOWN TO 0);
SIGNAL A, SUM, G : STD_LOGIC_VECTOR( 7 DOWN TO 0);
SIGNAL Func, FuncReg : STD_LOGIC_VECTOR( 1 TO 6 );
BEGIN
Zero <= '00', High <= '1'
```

4 control signals for 4 registers

Clear ⇐ Reset OR Done OR (NOT W AND T(0)) ;

Counter : upcount PORTMAP (Clear, Clock, Count); (Clear = W To + Done

decT : dec2to4 PORTMAP (Count, High, T); (High = Enable
2bit 2bit 2bit = 6bit

Func ⇐ F & Rx & Ry ; (concatenation)

function register : reg GENMAP (N ⇒ 6)
PORTMAP (Func, FRin, Clock, FuncReg);

decI : dec2to4 PORTMAP (FuncReg (1 to 2), High, I); (1 to 2)
decX : dec2to4 PORTMAP (FuncReg (3 to 4), High, X);
decY : dec2to4 PORTMAP (FuncReg (5 to 6), High, X);

Extern ⇐ I(0) AND T(1);
Gin ⇐ (I(2) OR I(3)) AND T(2);
Ain ⇐ (I(2) OR I(3)) AND T(1);

```
Gout <= (I(2) or I(3)) AND T(3);
AddSub <= I(3);
RegControl:
 FOR K IN 0 TO 3 GENERATE
Rin(k) <= (I(0) OR I(1)) AND T(1) AND X(K) OR
          ((I(2) OR I(3)) AND T(3) AND X(K);
Rout(K) <= (I(1) AND T(1) AND Y(K)) OR ((I(2) OR I(3))
          AND ((T(1) AND X(K)) OR (T(2) AND Y(K))));
END GENERATE RegControl;
tri_extern: tri PORTMAP (Dcutx, Extern, BUSWIRES);
 reg0: regn PORTMAP (BUSWIRES, Rin(0), Clock, R0);
 reg1: regn "  (BUSWIRES, Rin(1), Clock, R1);
```

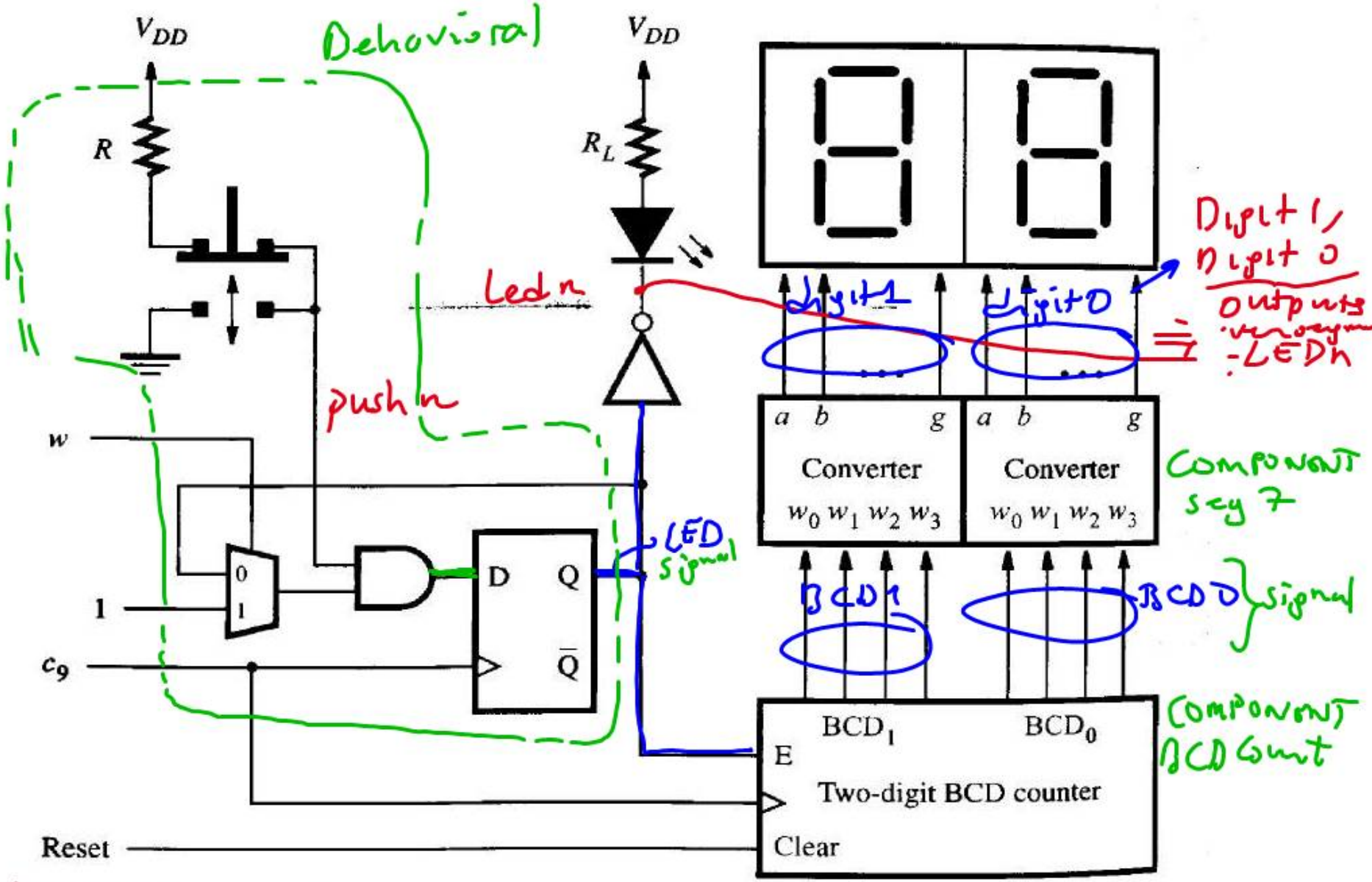
reg2 : regn PORTMAP (BUSWIRES, Rin2, Clock, R2);
reg3 : " " (" , Rin3, " , R3);

tri0 : trin PORTMAP (R0, Rout(0), BUSWIRES);
tri1 : " " (R1, Rout(1), ");
tri2 : " " (R2, Rout(2), ");
tri3 : " " (R3, Rout(3), ");

---

alu :
WITH Addsub SELECT
Sum ∈ A + B WHEN '0',
∈ A - B WHEN OTHERS;
END Behaviour;
regG : regn PORTMAP (Sum, Gin, Clock, G);
triG : trin PORTMAP (G, Gout, BUSWIRES);

REACTION TIMER:



Inputs: w, Clock (c9), Reset,

CODE: first BCD Counter (2-Digit)

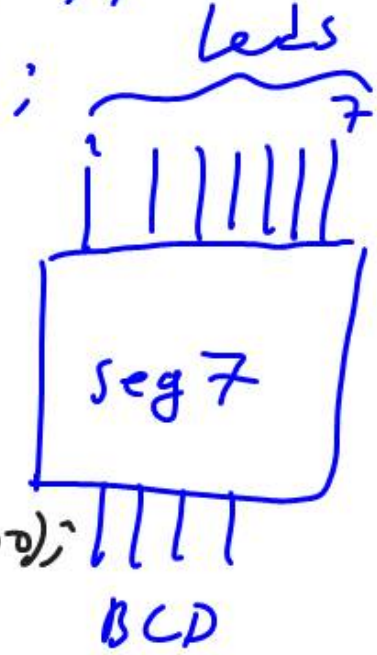
```
USE ieee.std_logic_unsigned.all;
ENTITY BCD_Count IS
PORT( Clock: IN STD_LOGIC;
      Clear_E: IN " ;

      BCD1, BCD0: BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0));

END BCD_Count;

ARCHITECTURE Behavior OF BCD_Count IS
BEGIN
PROCESS (clock)
BEGIN
```

```
IF CLOCK'EVENT AND CLOCK = '1' THEN
IF CLEAR = '1' THEN
BCD1 <= "0000"; BCD0 <= "0000";
ELSIF E = '1' THEN
IF BCD0 = "1001" THEN
    BCD0 <= "0000";
IF BCD1 = "1001" THEN
    BCD1 <= "0000";
ELSE
    BCD1 <= BCD1 + '1';
END IF;
ELSE
    BCD0 <= BCD0 + '1';
END IF;
END IF;
END IF;
END PROCESS;
END behavior;
```

CODE - Reaction Timer :

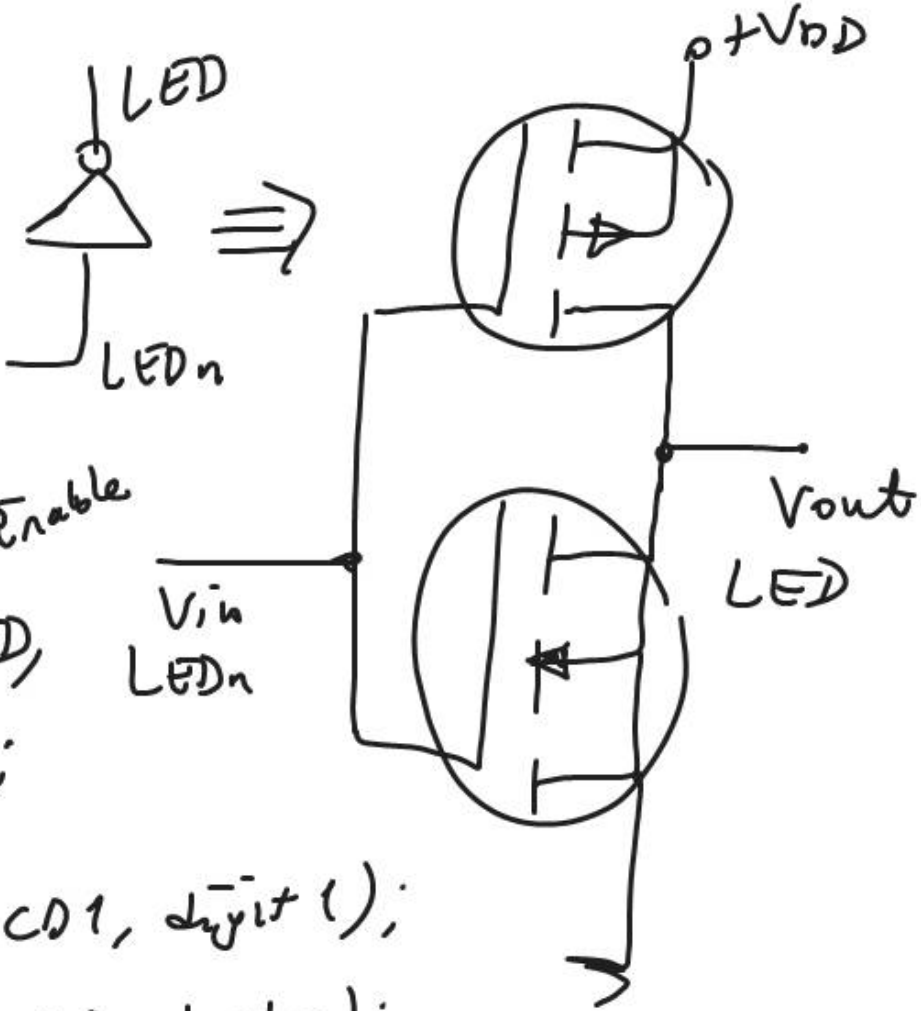
```
;
ENTITY Reaction IS
PORT ( Clk, Reset: IN STD_LOGIC;
       W, Push_n : IN  "  ;
       LED_n     : OUT STD_LOGIC;
   Digit1, Digit0 : BUFFER STD_LOGIC_VECTOR(3 DOWN TO 0));
End Reaction;
ARCHITECTURE Behavior of Reaction IS
COMPONENT BCD_count
PORT( Clock : IN STD_LOGIC;
      Clear, E : "  "  ;
   BCD1, BCD0 : BUFFER STD_LOGIC_VECTOR(3 DOWN TO 0));
END Component;
```

```
Component Seg 7
  PORT (bcd: IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
        leds: OUT ' ' ' ' ' ' (1 TO 7);
END Component;
```




```
SIGNAL LED : STD_LOGIC;

SIGNAL BCD0, BCD1 : STD_LOGIC_VECTOR(3 DOWNTO 0);

BEGIN

  flipflop: PROCESS ( Cy, pushn, w)

  BEGIN
  WAIT UNTIL Cy'EVENT AND Cy = '1';
  IF pushn = '0' THEN
   LED <= '0';
```

```
ELSIF  w = '1' THEN
  LED <= '1';
END IF
END PROCESS;
LEDn <= NOT LED;
```




```
Counter : BCD Count PORTMAP (Cg, Reset, LED,
                             BCD1, BCD0);
```

(Clock, clear, Enable)

```
seg1: seg7 PORTMAP ( BCD1, digit1);
seg2: seg7 "       ( BCD0, digit0);
END Behavior;
```

FINITE STATE MACHINES (FSM)

05.06.2010

©

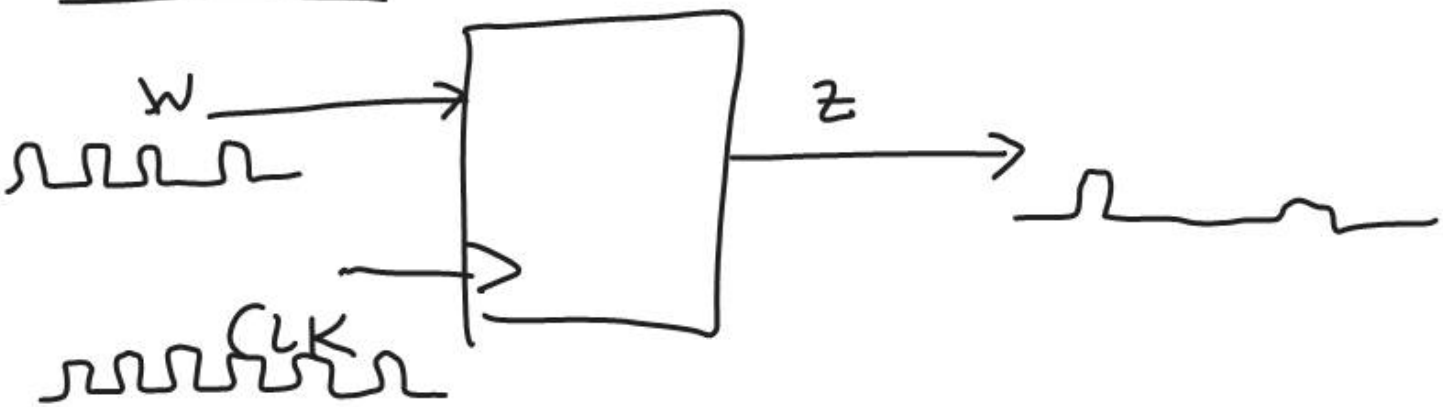
# of states ⇒ # of memory combinations

for example if we have 12 states

$2^4 = 16$ 4 FF can represent 16 different states (combinations)

We need at least 4 FFs.

Example



Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

MOORE TYPE MACHINE

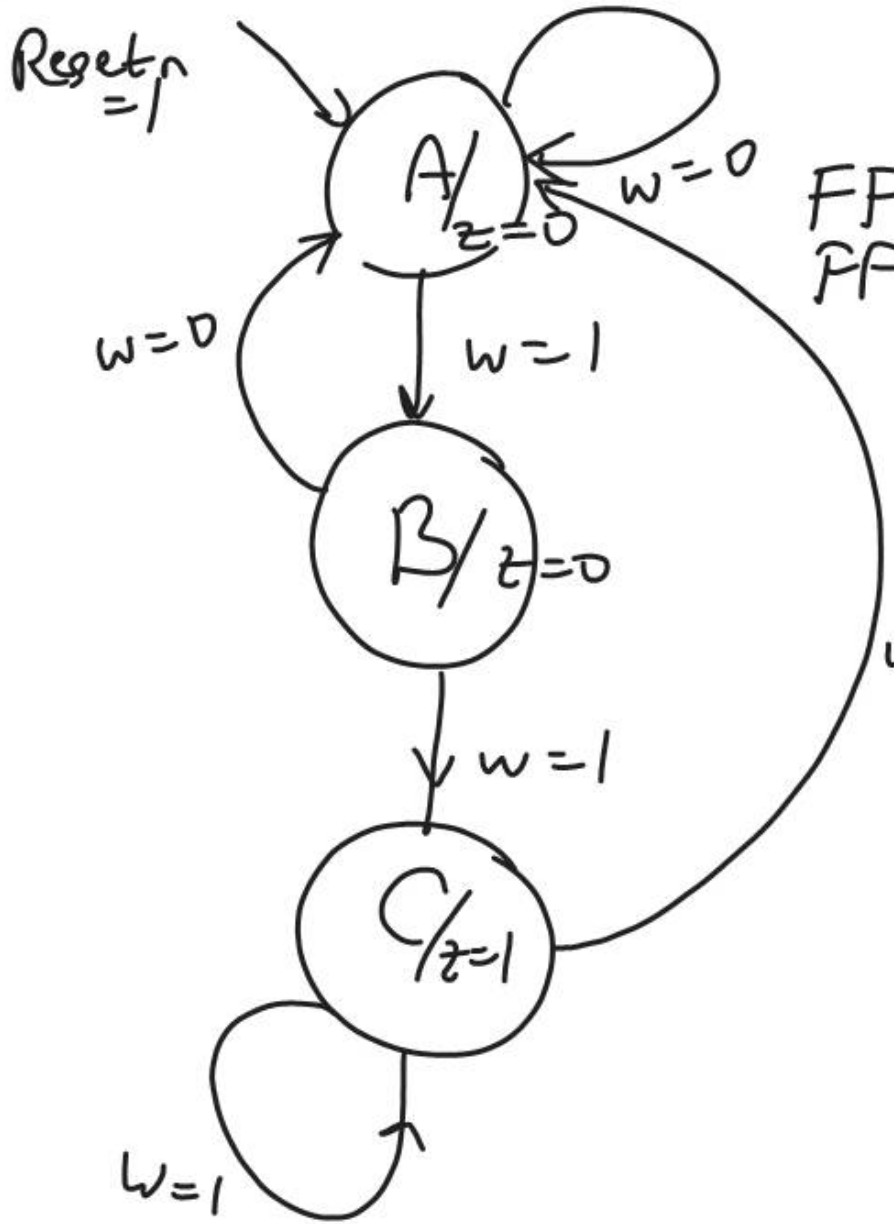
Output (z) is not the function of the input
but the function of the STATE

SEQUENTIAL CIRCUIT DESIGN STEPS

1. STATE DIAGRAM ⟶ Optimization (# of States, Moore vs Mealy State Design)
2. STATE TABLE
3. STATE ASSIGNED TABLE ⟶ Optimization (different combinations, one-hot encoding etc.)
4. FF TYPE SELECTION, DERIVATION OF NEXT STATES ⟶ OPTIMIZATION
   - different FFs
   - Reduction by Karnaugh maps & other methods
5. IMPLEMENTATION of the CIRCUIT

CAD Tools.
Intelligent Property (IP)

1st STEP : STATE DIAGRAM



3 states ⇒ 2 FFs

FF outputs : present states : y_1 y_0

FF inputs : next states : Y_1 Y_2

2. STEP : STATE TABLE

Present states y_1 y_0	Next States w=0 Y_1Y_0	Next States w=1 Y_1Y_0	Output z
A	A	B	0
B	A	C	0
C	A	C	1

MOORE MACHINE

3-STEP : STATE ASSIGNED TABLE:

2 FF

0 0	A
0 1	B
1 0	C
1 1	d

	Present		Next				Output
			w=0		w=1		
	y_1	y_0	Y_1	Y_0	Y_1	Y_0	z
A	0	0	0	0	0	1	0
B	0	1	0	0	1	0	0
C	1	0	0	0	1	0	1
	1	1	d	d	d	d	d

4-STEP

The easiest is to use D FF.

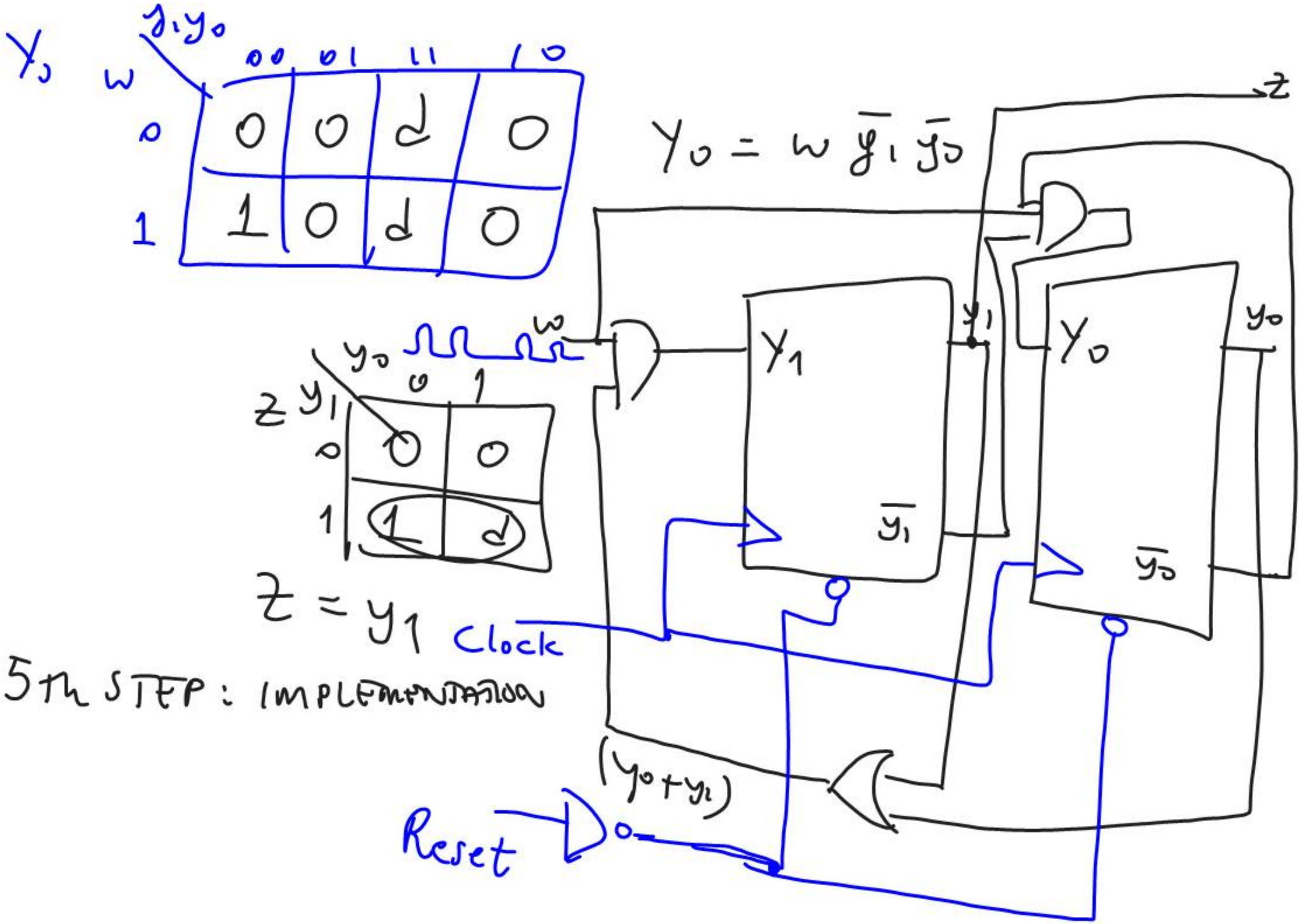
y_1y_0: 00, 01, 11, 10

Reduction →

Y_1 \ w	0	0	d	0
1	0	1	d	1

$Y_1 = w y_0 + w y_1$

$Y_1 = w(y_0 + y_1)$

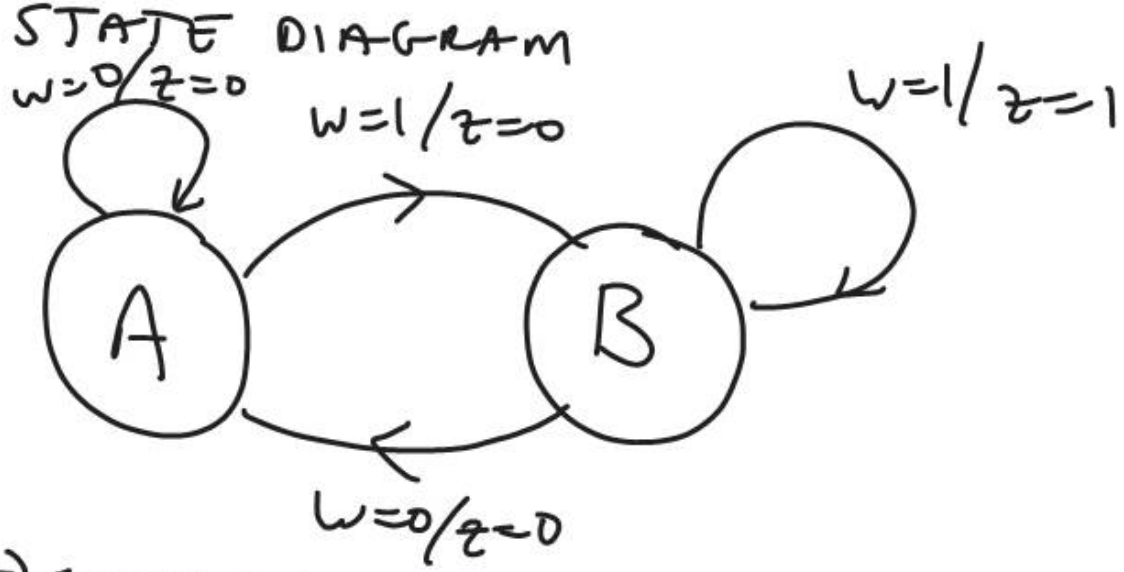


5th STEP : IMPLEMENTATION

Ex: Implement the same machine as a Mealy type FSM

Clock:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

STEP 1



2 States ⇒ 1 FF in DK

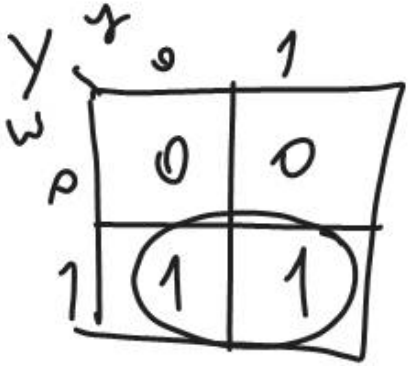
STEP 2: STATE TABLE

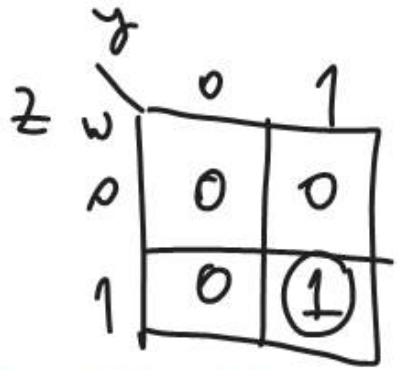
Present	Next w=0	Next w=1	Output z w=0	Output z w=1
A	A	B	0	0
B	A	B	0	1

Mealy type

z is function of w as well.

STEP 4:





STEP: 3 STATE ASSIGNED TABLE

y	Next y w=0	Next y w=1	Output z w=0	Output z w=1
0	0	1	0	0
1	0	1	0	1

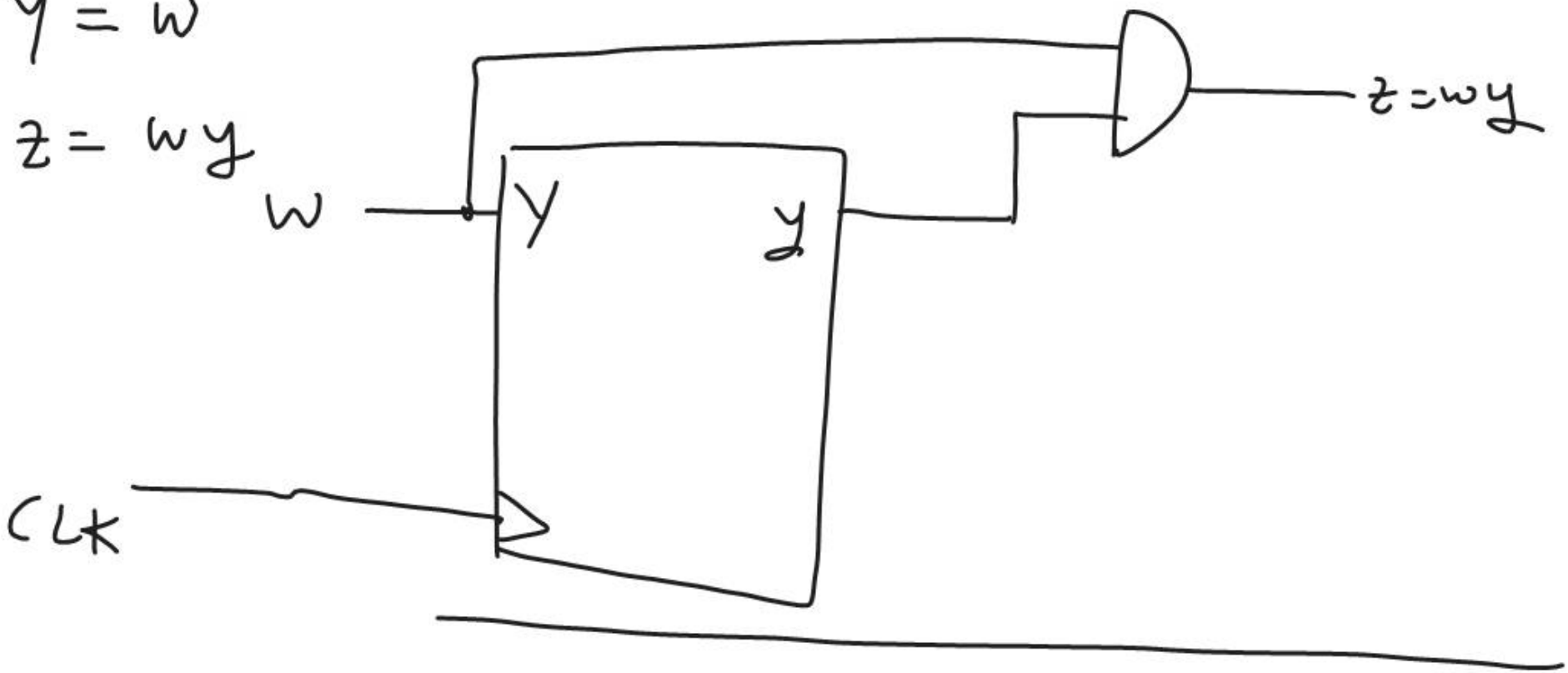
$Y = w$

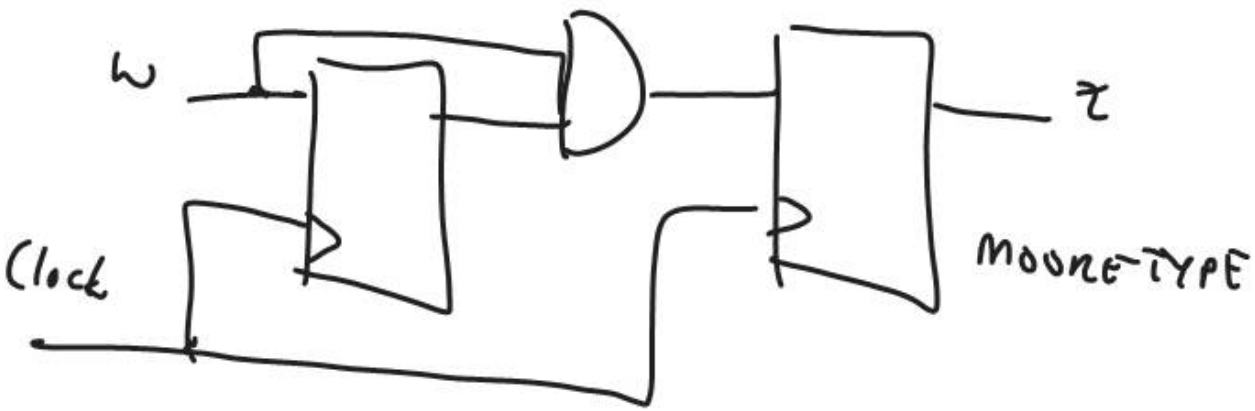
$z = wy$

STEP 5. IMPLEMENTATION

$Y = w$

$z = wy$

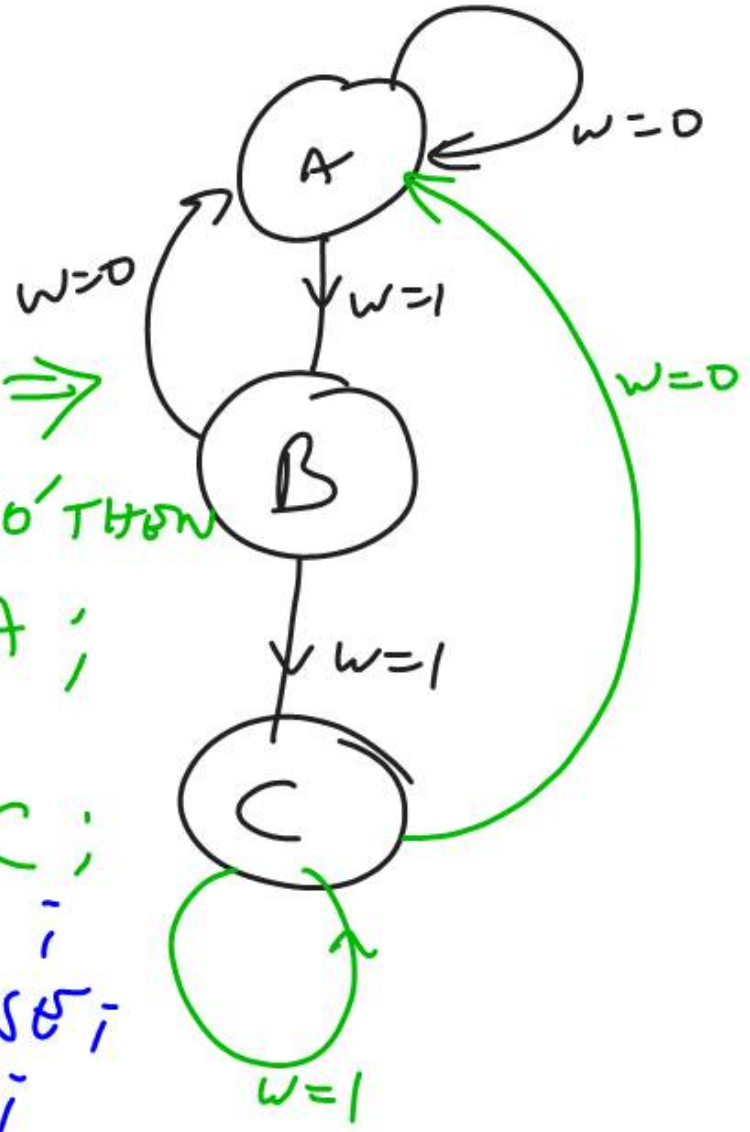




VHDL CODES for the MOORE TYPE FSM:

```
ENTITY SIMPLE IS
PORT ( Clock, Resetn, w : IN  STD_LOGIC;
                         : OUT STD_LOGIC);
       Z
END SIMPLE

ARCHITECTURE Behavior OF SIMPLE IS
TYPE State_type IS (A, B, C);
  SIGNAL y : State_type;
  BEGIN
  PROCESS ( Resetn, Clock)
  BEGIN
  IF Resetn = '0' THEN          } asynchronous Reset
        y <= A;
```

```
ELSIF (clock'EVENT AND Clock='1') THEN
CASE y IS
  WHEN A =>
    IF w='0' THEN
      y <= A;
    ELSE
      y <= B;
    END IF
  WHEN B =>
    IF w='0' THEN
      y <= A
    ELSE
      y <= C
    END IF
  WHEN C =>
    IF w='0' THEN
      y <= A;
    ELSE
      y <= C;
    END IF;
  END CASE;
  END IF;
END PROCESS;
z <= '1' WHEN y=C ELSE '0';
END Behavior;
```



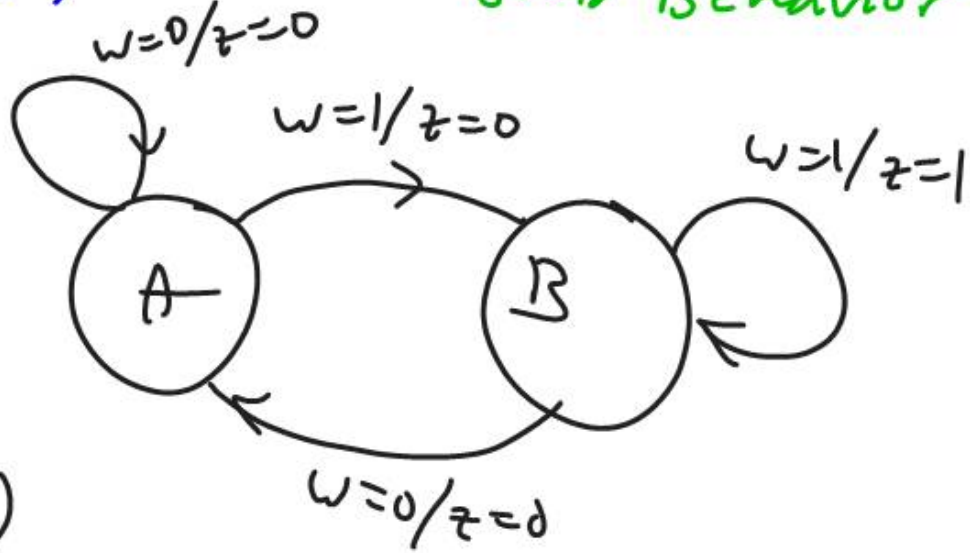
The VHDL Code of the MEALY TYPE FSM:

```
ARCHITECTURE Behavior OF simple LS
TYPE STATE_TYPE LS (A, B);
SIGNAL y : state_type;
BEGIN
PROCESS (Resetn, Clock)
BEGIN
 IF Resetn = '0' THEN
   y <= A;
 ELSIF (Clock'EVENT AND clock = '1') THEN
   CASE y IS
```

```
WHEN A =>
 IF W='0' THEN Y <= A;
 ELSE Y <= B;
 ENDIF;

 WHEN B =>
  IF W='0' THEN Y <= A;
  ELSE Y <= B;
  ENDIF;
 END CASE;
END IF;
END PROCESS;
```

2nd Process Statement

```
PROCESS (Y, W)
 CASE Y IS
  WHEN A =>
   z <= '0';
  WHEN B =>
   z <= W;
 END CASE;
END PROCESS;
END Behavior;
```

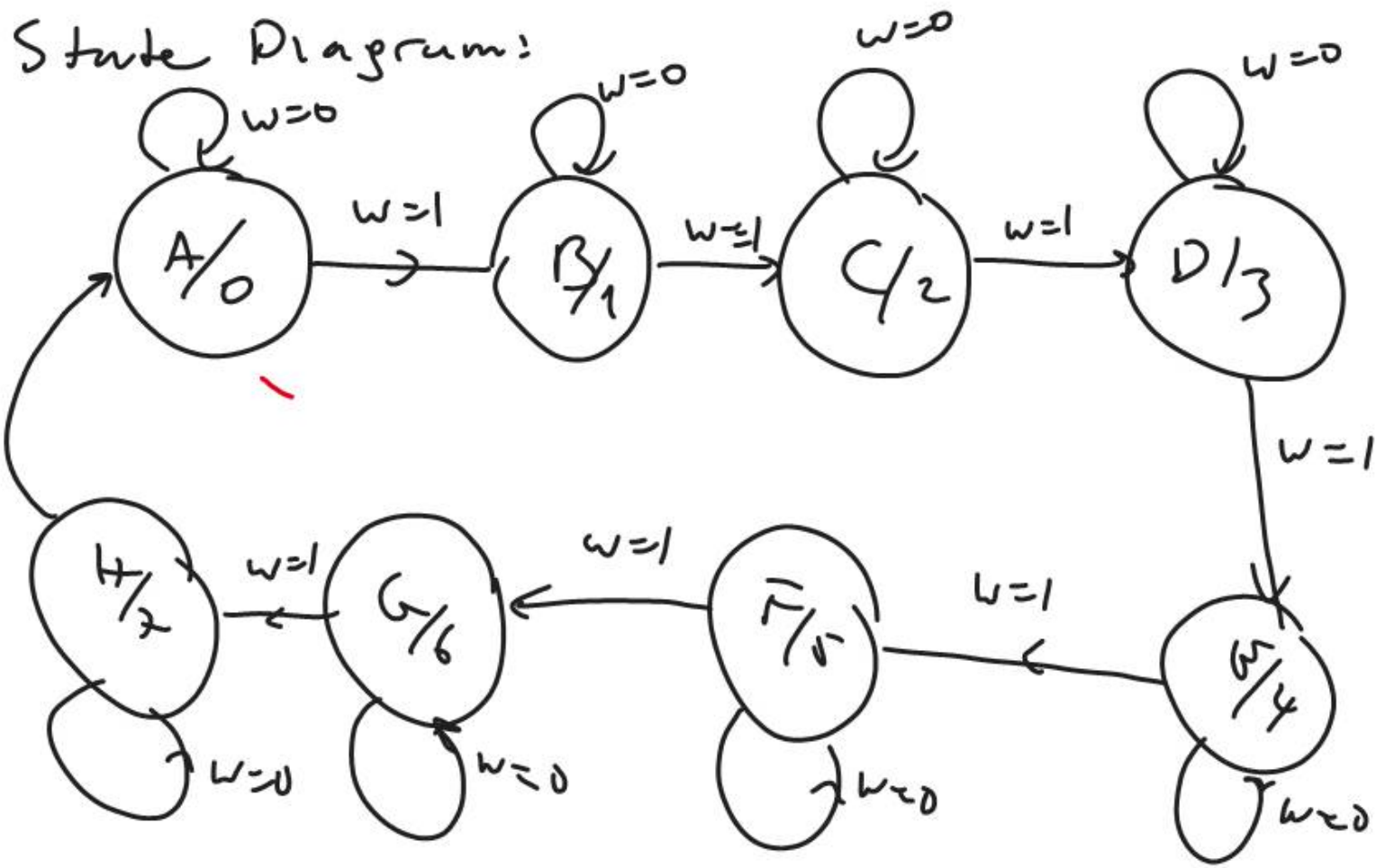


COUNTER DESIGN (as an FSM)

ex: Design a counter that counts the sequence

0, 1, 2, 3, 4, ---6, 7; 0, 1--

1. State Diagram:



2. State Table (MOORE)

Present State	Next State w=0	Next State w=1	Count Z
A	A	B	0
B	B	C	1
C	C	D	2
D	D	E	3
E	E	F	4
F	F	G	5
G	G	H	6
H	H	A	7

0 - 1 - - - 7

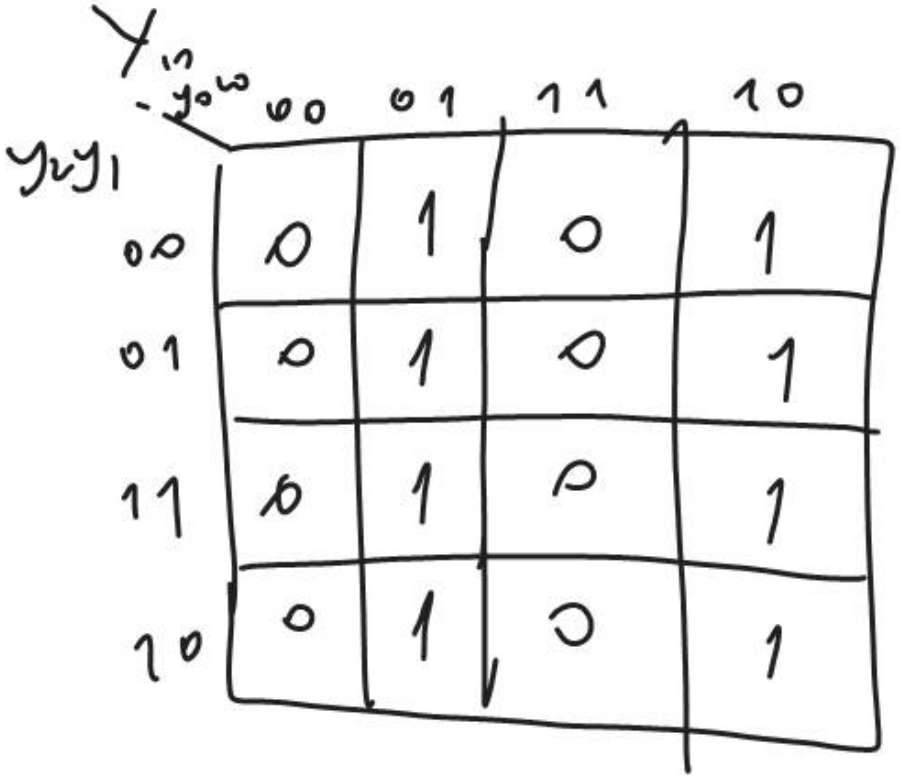
8 states = 2^3

3 FF

use D FF

1. State Assigned Table

Present State $y_2\ y_1\ y_0$	Next State $w=0$ $Y_2\ Y_1\ Y_0$	Next State $w=1$ $Y_2\ Y_1\ Y_0$	Output $z_2\ z_1\ z_0$
0 0 0	0 0 0	0 0 1	0 0 0
0 0 1	0 0 1	0 1 0	0 0 1
0 1 0	0 1 0	0 1 1	0 1 0
0 1 1	0 1 1	1 0 0	0 1 1
1 0 0	1 0 0	1 0 1	1 0 0
1 0 1	1 0 1	1 1 0	1 0 1
1 1 0	1 1 0	1 1 1	1 1 0
1 1 1	1 1 1	0 0 0	1 1 1



$$D_0 = Y_0 = \bar{y}_0 w + y_0 \bar{w} = w \oplus y_0$$

$$D_1 = Y_1 = \bar{w} y_1 + y_1 \bar{y}_0 + w y_0 \bar{y}_1 = w y_0 \oplus y_1$$

$$D_2 = Y_2 = \bar{w} y_2 + \bar{y}_0 y_2 + \bar{y}_1 y_2 + w y_0 y_1 \bar{y}_2 = w y_0 y_1 \oplus y_2$$