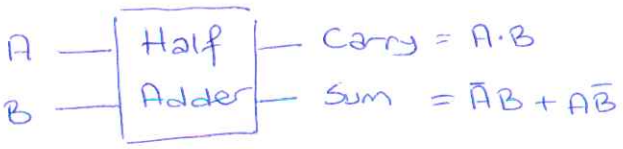
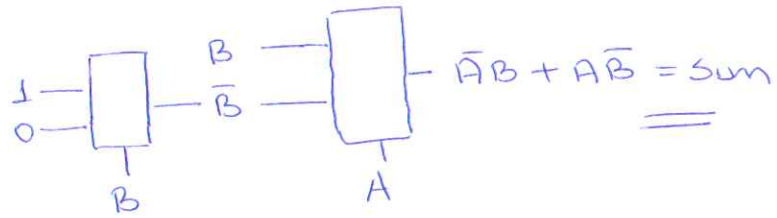
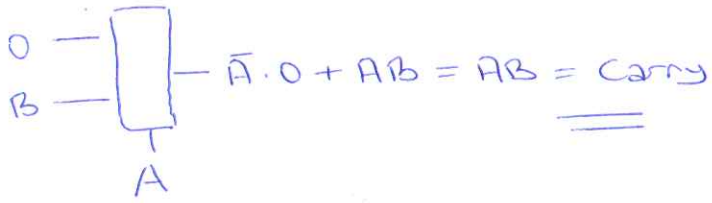
Ex1: Design a half adder with a minimum number of 2-1 multiplexers. The complements of the variables are NOT available. However, you can use logic 1 and 0. No other gates are available.

Ex2: Implement $F(X,Y,Z) = XY'Z + XYZ'$ with a single 2-to-4 decoder and a simple 2-to-1 multiplexer.

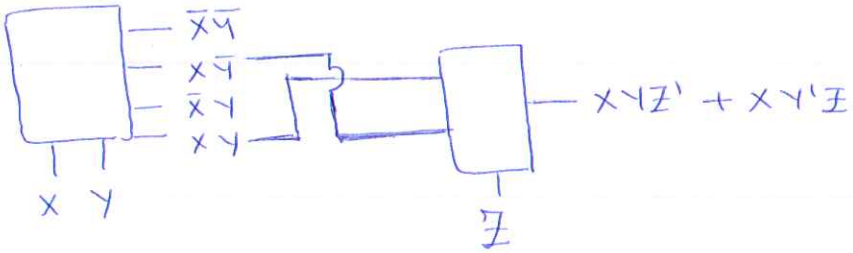
Sol1:




Sol2:



Ex3: Design a sequential circuit that outputs 1 when the total number of 1's on the input (including the current input) is a multiple of 3. Otherwise the circuit outputs 0.

As an example,

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| x | 0 | 1 | 1 | 0 | 1 | 0 | 1 | ... |
| y | 1 | 0 | 0 | 0 | 1 | 1 | 0 | ... |

where x is the input and y is the output.

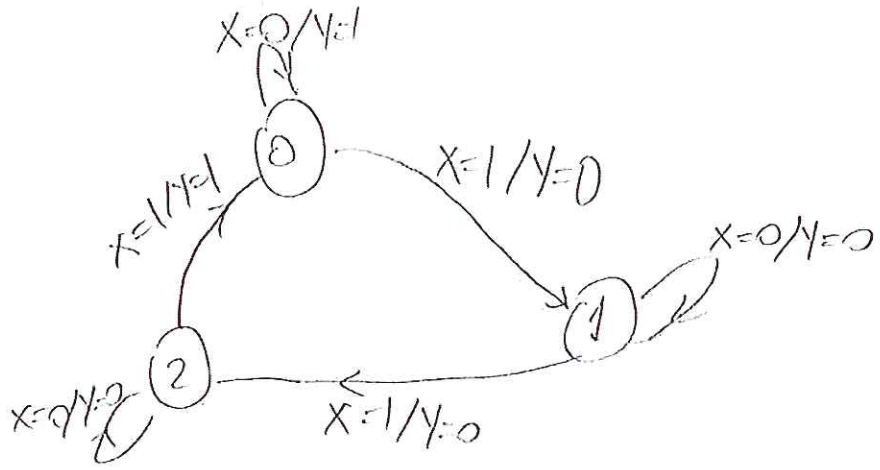
You can only use MB-type flip-flop, where an MB type flip-flop is defined as follows:

| M | B | Q(t+1) |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | Q(t) |

and a minimum number of NAND gates.

## Solution3:

X: input
Y: output




| $Q_1$ | $Q_0$ | X | $Q_1^+$ | $Q_0^+$ | Y |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |

| $M_1$ | $B_1$ |
|---|---|
| 1 | X |
| 1 | X |
| 1 | X |
| 0 | X |
| 0 | X |
| 1 | 0 |

| $M_0$ | $B_0$ |
|---|---|
| 1 | X |
| 0 | X |
| 0 | X |
| 1 | 0 |
| 1 | X |
| 1 | X |

| M | B |
|---|---|
| 1 | X |

also could be X 1
but I chose 0 X for this design.

For $M_1$

| X \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 1 | X | 0 |
| 1 | 1 | 0 | X | 1 |

$M_1 = \bar{Q}_1 \bar{X} + \bar{Q}_0 X$

For $B_1$

| X \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | X | X | X |
| 1 | X | X | X | 0 |

$B_1 = 0$

For $M_0$

| X \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | X | 1 |
| 1 | 0 | 1 | X | 1 |

$M_0 = \bar{Q}_0 \bar{X} + Q_0 X + Q_1$

For $B_0$

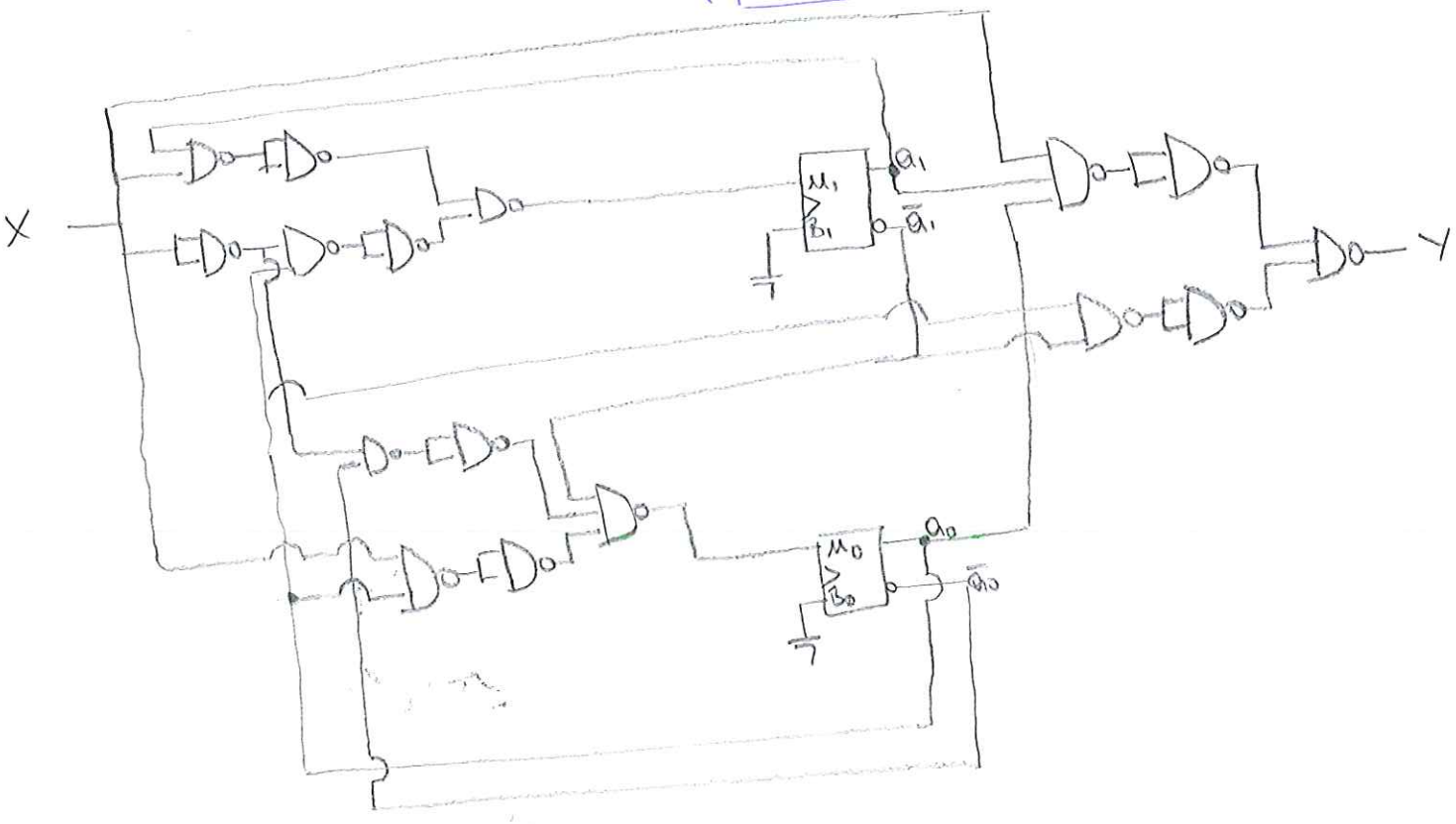
| X \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | X | X | 0 |
| 1 | X | X | X | X |

$B_0 = 0$

For Y

| X \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | X | 0 |
| 1 | 0 | 0 | X | 1 |

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



**EEE102** **1-12-2008**

# How to describe FSMs using VHD

## EXAMPLE 1:

**A Moore FSM has one input X and one output Y. This FSM looks at the values of X at clock ticks and if two successive 1s are detected then the output becomes 1 and remains there forever. Thus initially the output is 0 and becomes 1 when a "11" sequence is detected. Therefore it is a code detector. It detects the code "11" (one may think of code detectors which detect longer sequences of the input bits).**

*(i)*

**We can write the VHDL code just starting from the word description without having to make any design:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--If two successive ones come the output becomes 1 and remains there

entity E_11_detector is
Port ( CLOCK : in std_logic;
X : in std_logic;
Y: out std_logic);
end E_11_detector;

architecture Behavioral of E_11_detector is
signal lastX:std_logic :='0';
signal flag:std_logic :='0';begin

process(CLOCK)
begin
if CLOCK'event and CLOCK = '1' then
        lastX<=X;
        if flag ='0' and lastX='1' and X='1' then  flag<='1';end if;
end if;
end process;
Y<=flag;
end Behavioral;
```

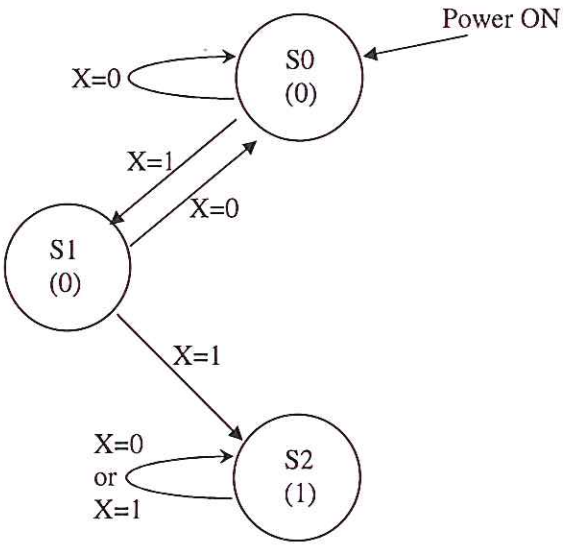
**--we need the signal flag because we cannot initialize Y which is an output.**

*(ii)*

**Another approach would be to proceed with the design manually until we obtain the state/output diagram and then switch to VHDL.**

**Let us define the states:**
**S0: No ticks yet or the last X (value of X at last tick) is 0.**
**S1: Last X (value of X at last tick) is 1.**

**S2: Last two X values are both 1.**
**The state output/diagram is then**




**Since we have 3 states we need 2 flip flops. Say the flip flop outputs are Q1 and Q0.**
**State encoding can be taken as**

| State name | Q1 | Q0 |
|---|---|---|
| S0 | 0 | 0 |
| S1 | 0 | 1 |
| S2 | 1 | 0 |

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity example2 is
Port ( CLOCK : in std_logic;
X : in std_logic;
Y : out std_logic);
end example2;

architecture Behavioral of example2 is
constant S0:std_logic_vector (1 downto 0) :="00";
constant S1:std_logic_vector (1 downto 0) :="01";
constant S2:std_logic_vector (1 downto 0) :="10";
signal S:std_logic_vector (1 downto 0) :=S0;
signal nextS:std_logic_vector (1 downto 0);
begin
--code for state memory block
```

```
L1: process(CLOCK)
begin
if CLOCK'event and CLOCK = '1' then S<=nextS;end if;
end process;
--code for next state logic circuit
L2: process(X,S)
begin
case S is
when S0 => if X = '0' then nextS <=S0;else nextS <= S1;end if;
when S1 => if X = '0' then nextS <=S0;else nextS <= S2;end if;
when S2 => if X = '0' then nextS <=S2;else nextS <= S2;end if;
when others => null;
end case;
end process;
--code for output circuit
L3: process (S)
begin
case S is
when S0=> Y<='0';
when S1=> Y<='0';
when S2=> Y<='1';
when others=> Y<='0';
end case;
end process;
end Behavioral;
```

*(iii)*

**Yet another approach is to proceed with the design manually till the end and write the VHDL code of the final circuit. In this case the purpose of writing VHDL is to make simulation only.**

**The next state table:**

| Q1 | Q0 | X | | Q1* | Q0* |
|---|---|---|---|---|---|
| 0 | 0 | 0 | | 0 | 0 |
| 0 | 0 | 1 | | 0 | 1 |
| 0 | 1 | 0 | | 0 | 0 |
| 0 | 1 | 1 | | 1 | 0 |
| 1 | 0 | 0 | | 1 | 0 |
| 1 | 0 | 1 | | 1 | 0 |
| 1 | 1 | 0 | | d | d |
| 1 | 1 | 1 | | d | d |

**In this table X is the value of input at the clock tick.**

**Excitation table is the same as above except Q1* is replaced by D1 and Q0* is replaced by D0.**

**The excitation table is**

| Q1 | Q0 | X | | D1 | D0 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | | 0 | 1 |
| 0 | 1 | 0 | | 0 | 0 |
| 0 | 1 | 1 | | 1 | 0 |
| 1 | 0 | 0 | | 1 | 0 |
| 1 | 0 | 1 | | 1 | 0 |
| 1 | 1 | 0 | | d | d |
| 1 | 1 | 1 | | d | d |

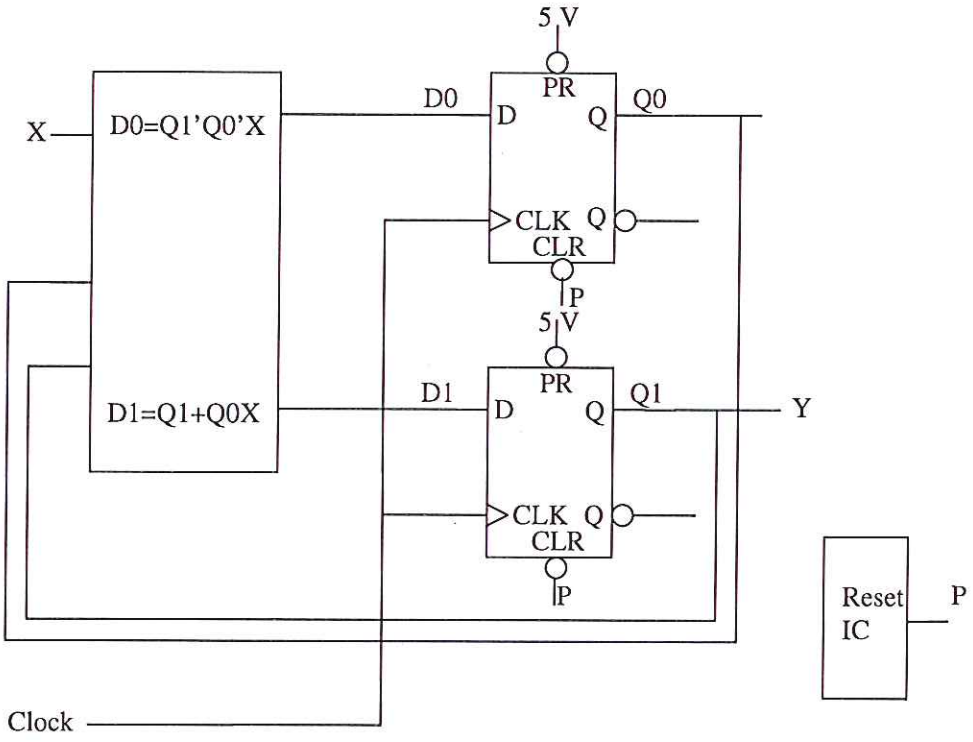
**Using Karnaugh maps we obtain D0=Q1'Q0'X and D1=Q1+Q0X.**

**The output table:**

| Q1 | Q0 | | Y |
|---|---|---|---|
| 0 | 0 | | 0 |
| 0 | 1 | | 0 |
| 1 | 0 | | 1 |
| 1 | 1 | | d |

**and Y = Q1.**

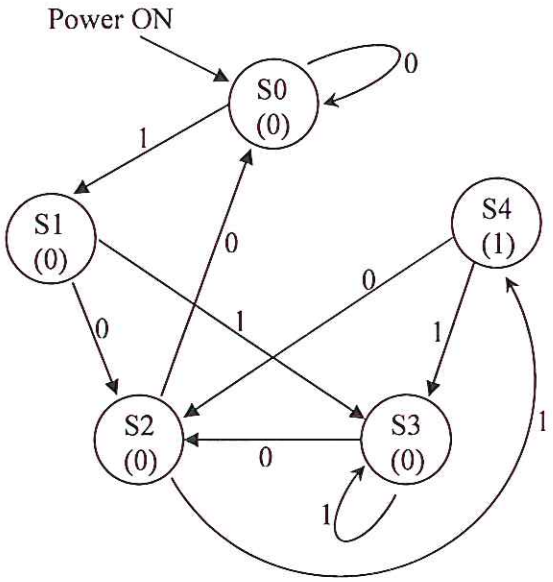
**We may now draw the circuit**




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--If two successive ones come the output becomes 1 and remains there

entity ones_counter is
Port ( CLOCK : in std_logic;
X : in std_logic;
Y : out std_logic);
end ones_counter;

architecture Behavioral of ones_counter is
signal S:std_logic_vector (1 downto 0) :="00"; --state variable
signal nextS:std_logic_vector (1 downto 0);--inputs of the D flip flops
begin
--code for state memory block
process(CLOCK)
begin
if CLOCK'event and CLOCK = '1' then S<=nextS;end if;
end process;

--code for next state logic circuit
nextS(0)<=not S(1) and not S(0) and X;
nextS(1)<=S(1) or (S(0) and X);

--code for output circuit
Y<=S(1);
end Behavioral;
```

**Q5. Design and draw the FSM which has the state/output diagram shown below. Show all your design steps. What does this FSM do?**




**Considering that S1 and S3 are equivalent states,**

**State Encoding:**

| State | Q1 | Q0 |
|---|---|---|
| S0 | 0 | 0 |
| S1 | 0 | 1 |
| S2 | 1 | 0 |
| S4 | 1 | 1 |

**Next State Table:**

| Q1 | Q0 | X | Q1* | Q0* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |

**Excitation Table:** Same as NS table.

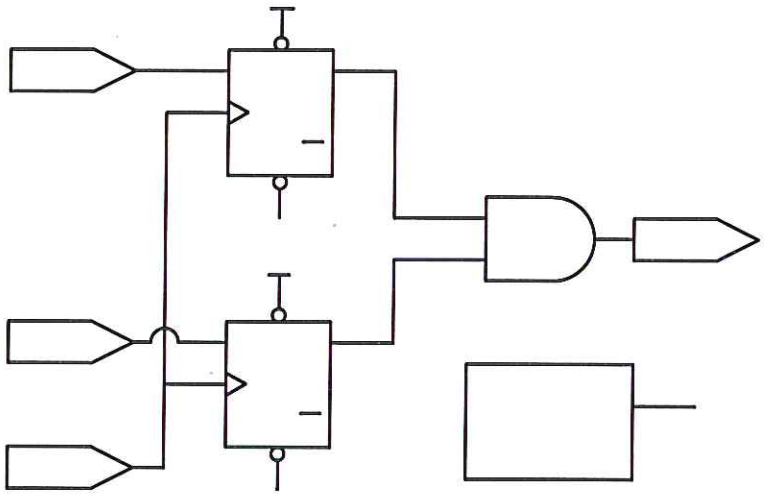
**Next State Logic:**
**Q1*=D1=Q0X'+Q1$\overline{Q0}$X**
**Q0*=D0=X**

**Output Table:**

| Q1 | Q0 | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Y=Q1.Q0**

**Circuit:**



**Function:**
Detects "101" sequence from the input.

+5

D1=Q0X'+Q1$\overline{Q0}$X

D SET Q

CLR Q

P
+5

D0=X

D SET Q

Q3.

Implementing functions using ROMs or RAMs

Use two 16x1 and one 8x1 ROM (or RAM) to design a combinational circuit with 5 inputs and one output.

Solution:

Let us assume the circuit has the following truth table:

| I4 | I3 | I2 | I1 | I0 | F |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | F0 |
| 0 | 0 | 0 | 0 | 1 | F1 |
| | . | | | | . |
| | . | | | | . |
| | . | | | | . |
| 0 | 1 | 1 | 1 | 1 | F15 |
| 1 | 0 | 0 | 0 | 0 | F16 |
| | . | | | | . |
| | . | | | | . |
| | . | | | | . |
| 1 | 1 | 1 | 1 | 1 | F31 |
| | . | | | | |

In Figure 1 below the upper 16 rows and the lower 16 rows of the TT are implemented using two 16x1 RAMs. The outputs of the RAMs are multiplexed using a 2-to-1 multiplexer with select I4.

In Figure 2 the 2-to-1 mux is also implemented using a RAM, in this case an 8x1 RAM.

Figure 3 explains the logic behind implementing a 2-to-1 mux with an 8x1 RAM.

Note that instead of RAMs we could have used ROMs. Control inputs of the RAMs are not shown in the figures. The RAMs must be configured to be in reading mode.

The boxes in the RAM blocks show the contents of the memory locations with ascending order.

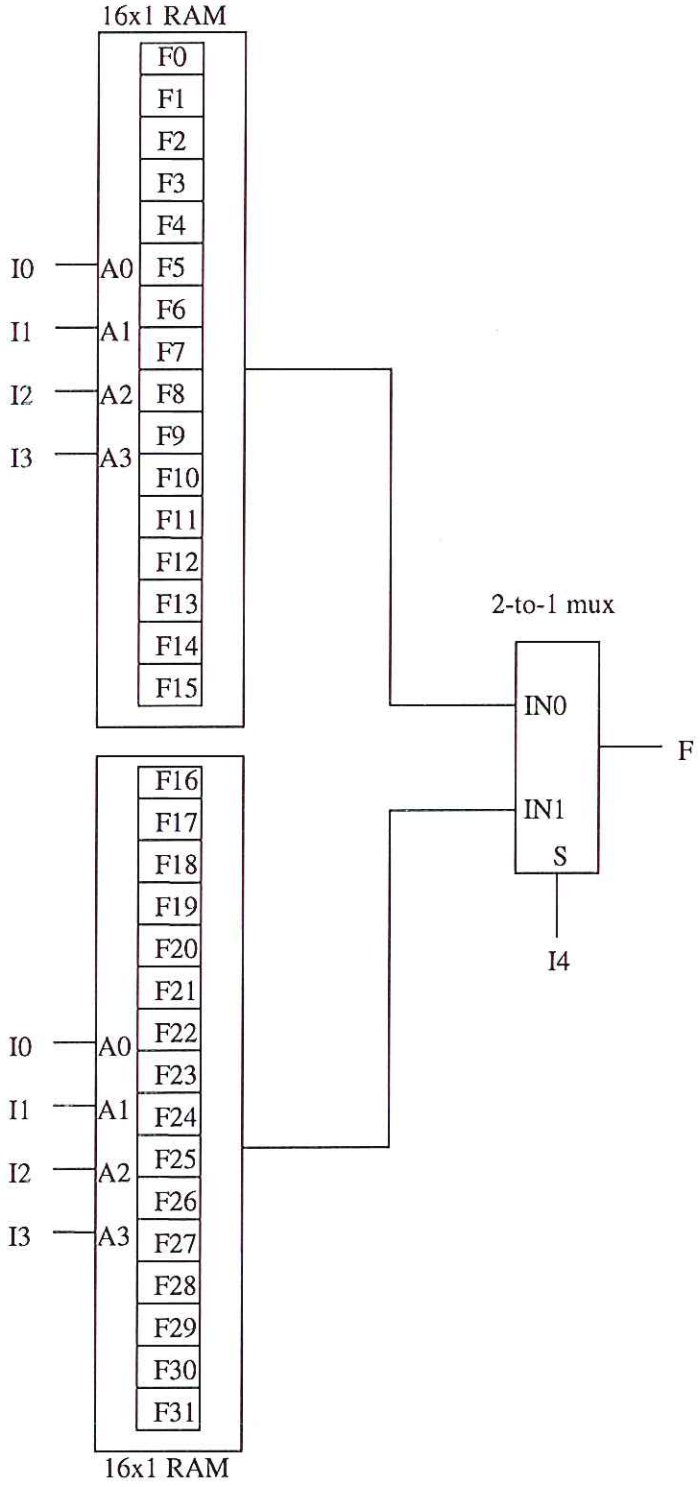

16x1 RAM
F0
F1
F2
F3
F4
I0
A0
F5
F6
I1
A1
F7
I2
A2
F8
F9
I3
A3
F10
F11
F12
F13
F14
F15
2-to-1 mux
IN0
F
IN1
S
I4
F16
F17
F18
F19
F20
F21
I0
A0
F22
F23
I1
A1
F24
I2
A2
F25
F26
I3
A3
F27
F28
F29
F30
F31
16x1 RAM

**Figure 1**

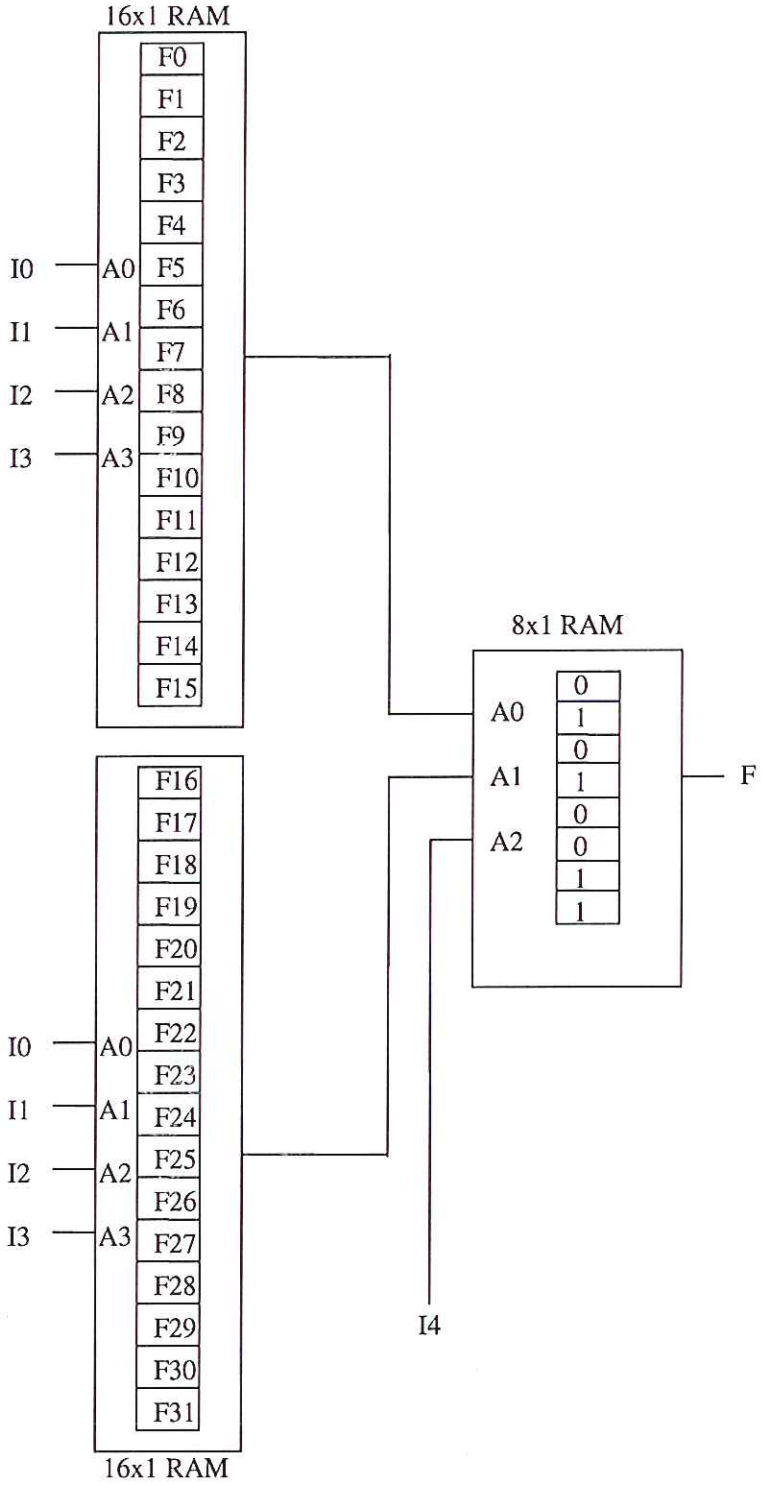

16x1 RAM
F0
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
I0
A0
I1
A1
I2
A2
I3
A3
8x1 RAM
A0
A1
A2
0
1
0
1
0
0
1
1
F
F16
F17
F18
F19
F20
F21
F22
F23
F24
F25
F26
F27
F28
F29
F30
F31
I0
A0
I1
A1
I2
A2
I3
A3
I4
16x1 RAM

**Figure 2**

**How to implement a 2-to-1 multiplexer using an 8x1 RAM.**

**Truth table of a 2-to-1 mux is**

| S | IN1 | IN0 | OUT |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**To implement this using an 8x1 RAM**

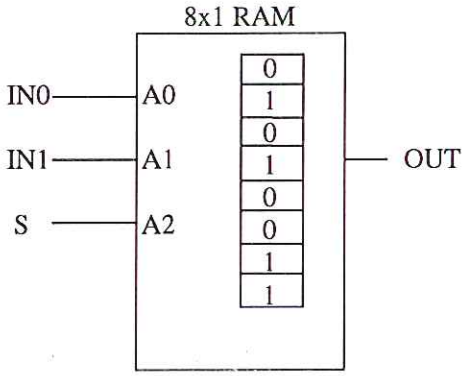



**Figure 3**