

State machines can be represented in several ways.

- As a two dimensional array
- as a set of excitation equations (dataflow)

and most commonly

- as a 'case statement'

which is probably the easiest.

```
architecture ....
```

```
    type state_type is (s0, s1 )  
    signal current, next_ : state_type
```

```
begin
```

```
    states : process( sensitivity list)
```

```
    begin
```

```
        case current is
```

```
            when s0 =>      next_ <= s1;
```

```
        end case;
```

```
    end process;
```

```
    ticker : process(clk)
```

```
    begin
```

```
        if (rising_edge(clk)) then
```

```
            current <= next_;
```

```
        end if;
```

```
    end process;
```

```
end ;
```

```
entity controller is

    port (
        car           : in  bit;
        timed         : in  bit;
        clk           : in  bit;
        reset         : in  bit;
        timer_run     : out bit;
        minor_green   : out bit;
        major_green   : out bit);

end controller;
```

architecture state_machine of controller is

```
type state_type is (green, red);
```

```
-- pragma CLOCK clk
```

```
-- pragma CURRENT_STATE current_state
```

```
-- pragma NEXT_STATE next_state
```

```
signal current_state : state_type;
```

```
signal next_state    : state_type;
```

```
begin  -- state_machine

state_comb : process(current_state,
                    car, timed)
begin
  if (reset = '0') then

    next_state      <= green;
    minor_green     <= '0';
    major_green     <= '1';

  else
```

```
case current_state is

when green =>
    major_green    <= '1';
    minor_green    <= '0';
    if (car = '1') then
        next_state <= red;
        timer_run  <= '1';
    else
        next_state <= green;
    end if;
```

```
when red      =>
    major_green    <= '0';
    minor_green    <= '1';
    if (timed = '1') then
        next_state <= green;
    else
        next_state <= red;
    end if;
when others   => null
end case;
end if;
end process state_comb;
```

```
process (clk)

begin

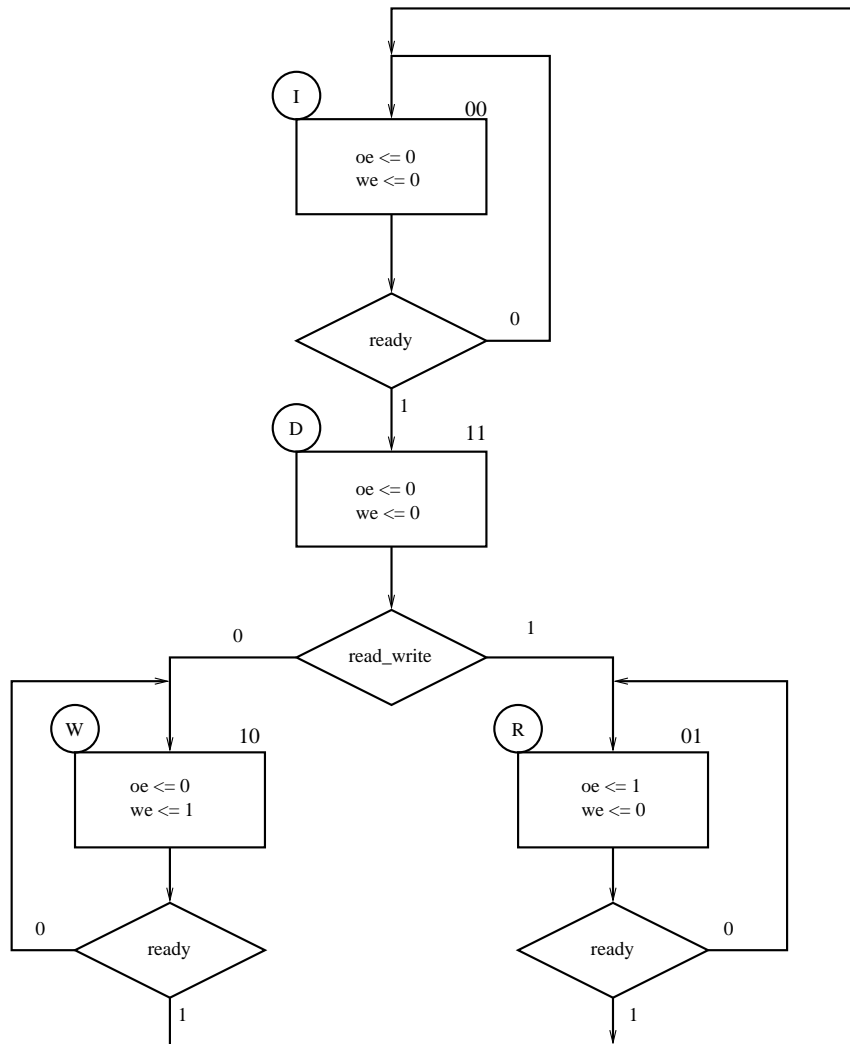
    if (clk = '0' and not clk'stable) then
        current_state <= next_state;
    end if;

end process;

end state_machine;
```

Algorithmic State Machines :

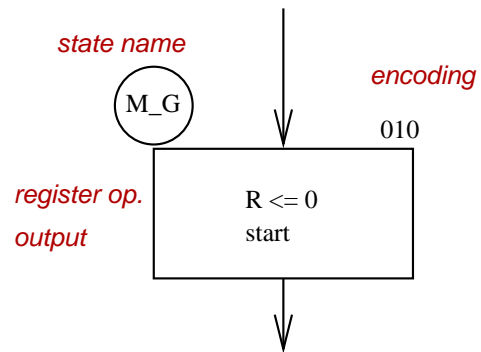
0



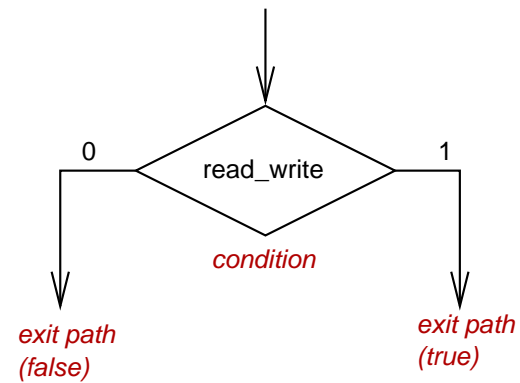
A typical ASM diagram.

Algorithmic State Machines : symbols

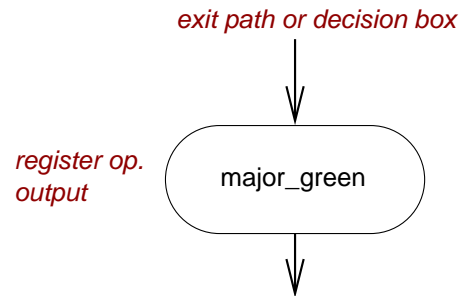
Basic state symbol.



Decision symbol



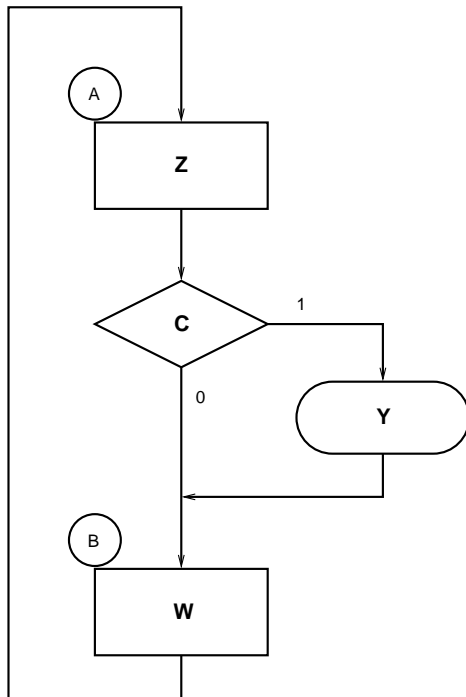
Conditional symbol



Algorithmic State Machines :

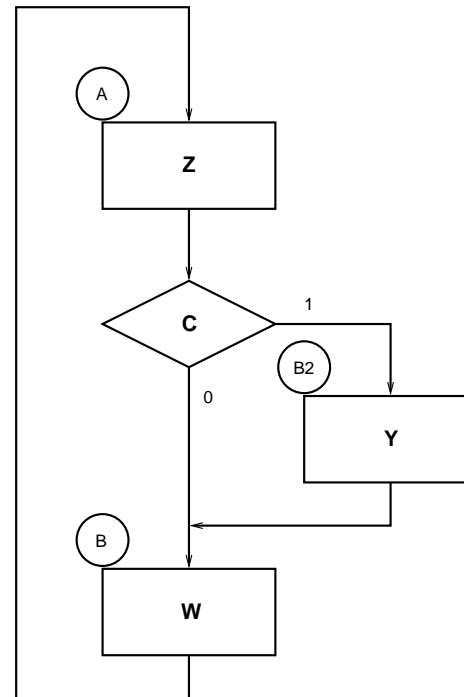
outputs

conditional output



Y is dependent on C.
Asserted if C is
true or becomes true.

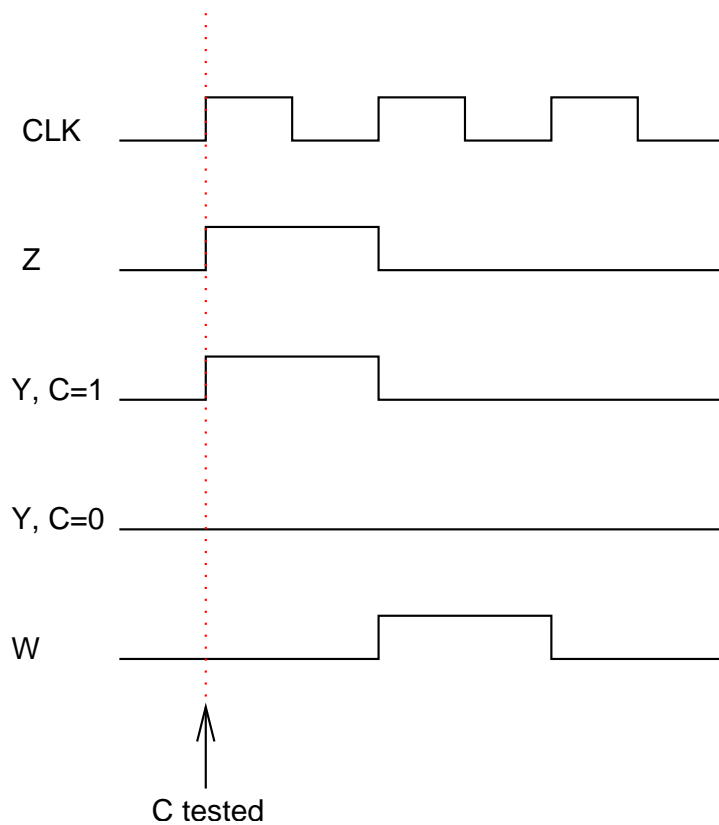
unconditional output



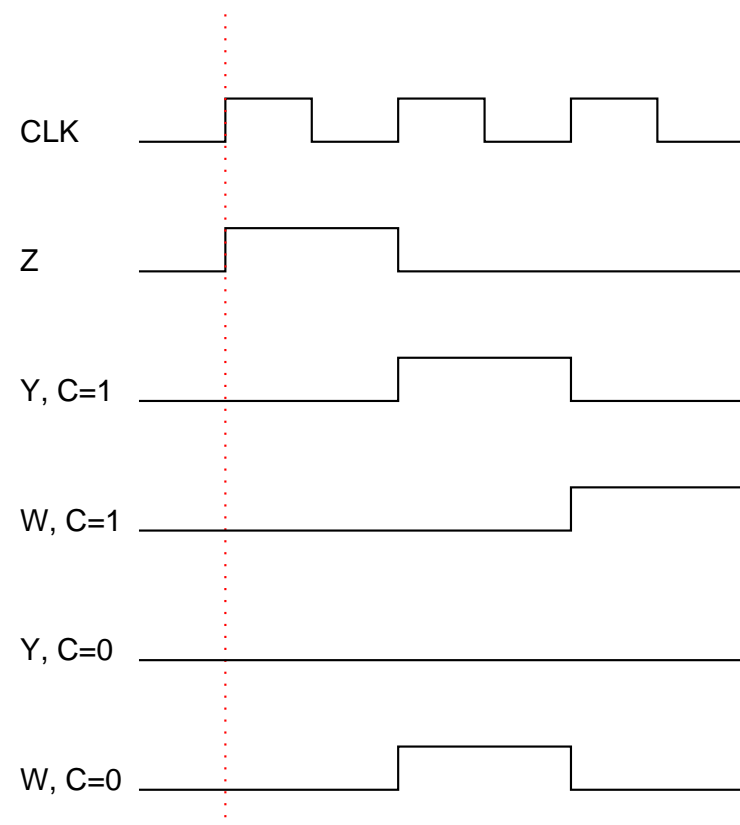
Y asserted if state
B2 entered

Algorithmic State Machines :

outputs 2



Timing diagram for the conditional output Y.



State dependent output Y

VHDL:

monitoring state.

```
in vdd B;
in vss B;;
in reset B;;
in clk B;;
in read_write B;
in ready B;;
out oe B;;
out we B;
register rw_buf_x.controller_current_state(1 downto 0) B;

begin

< 0 ns >  init  : 10 0 0 0 0 * * **;
< +2ns >      : 10 0 0 0 0 * * **;
< +2ns >      : 10 0 1 0 0 * * **;
```

Algorithmic State Machines :

outputs

```
syf -aEV rw_buf_x    rw_buf_x
```

ARCHITECTURE VBE OF rw_buf_x IS

```
SIGNAL controller_current_state : REG_VECTOR(1 DOWNT0 0) REGISTER;
```

```
SIGNAL controller_current_state_wrote : BIT    -- controller_current_state_wrote
```

```
SIGNAL controller_next_state_wrote : BIT;    -- controller_next_state_wrote
```

```
SIGNAL controller_current_state_rd : BIT;    -- controller_current_state_rd
```

```
# Encoding figure "controller"
```

```
-controller a 2
```

```
wrote 2
```

```
rd 1
```

```
decision 3
```

```
idle 0
```