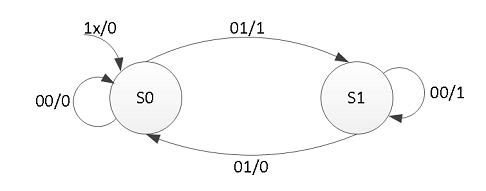
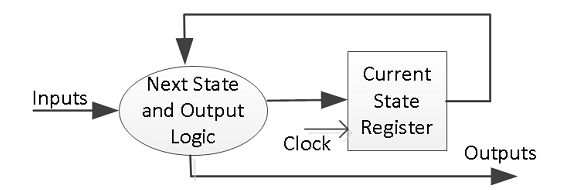
# Mealy

A general model of a Mealy sequential machine consists of a combinatorial network, which **generates the outputs and the next state**, and a **state register** which holds the present state as shown below.

The state register is normally modeled as **D flip-flops**. The state register must be sensitive to a clock edge. The other block(s) can be modeled either using the always procedural block or a mixture of the always  
procedural block and dataflow modeling statements; the always procedural block will have to be  
sensitive to all inputs being read into the block and must have all output defined for every branch in order  
to model it as a combinatorial block. The two blocks Mealy machine can be viewed as:



parity **<=** '0'**;**

**case** **(** state **)** **is**

**when**

S0 **=>**

**if** **(** x **)** **then**

parity **<=** 1 **;**

nextstate **<=** S1**;**

**else**

nextstate **<=** S0**;**

**end** **if;**

**when**

S1 **=>**

**if** **(** x **)** **then**

nextstate **<=** S0**;**

**else**

parity **<=** 1 **;**

nextstate **<=** S1**;**

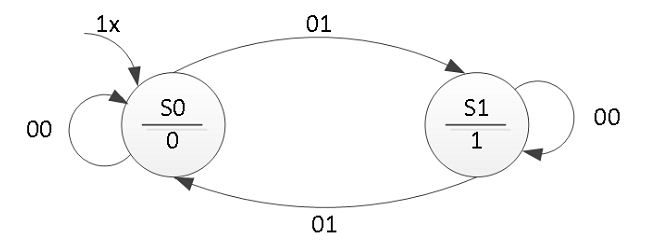
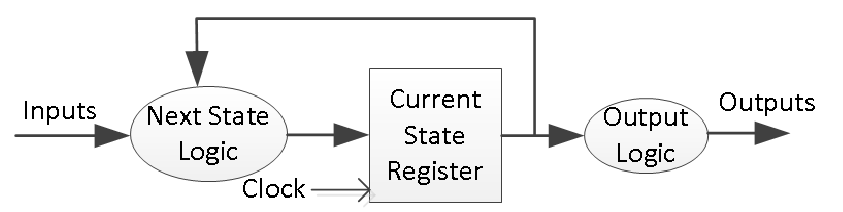
**end** **if;**

**when** **others** **=>** nextstate **<=** S0**;**

**end** **case;**

# Moore

A general model of a Moore sequential machine is shown below. Its output is generated from the state  
register block. The next state is determined using the present (current) input and the present (current)  
state. Here the state register is also modeled using D flip-flops. Normally Moore machines are described  
using three blocks, one of which must be a sequential and the other two can be modeled using always  
blocks or a combination of always and dataflow modeling constructs.



nextstate **<=** S0**;**

**case** **(** state **)** **is**

**when**

S0 **=>**

**if** **(** x **)** **then**

nextstate **<=** S1**;**

**end** **if;**

**when**

S1 **=>**

**if** **(** **(** not x **)** **)** **then**

nextstate **<=** S1**;**

**end** **if;**

**end** **case;**

# CAM

A CONTENT-ADDRESSABLE MEMORY (CAM) compares input search data against a table of stored data, and returns the address of the matching data [1]–[5]. CAMs have a single clock cycle throughput making them faster than other hardware- and software-based search systems.

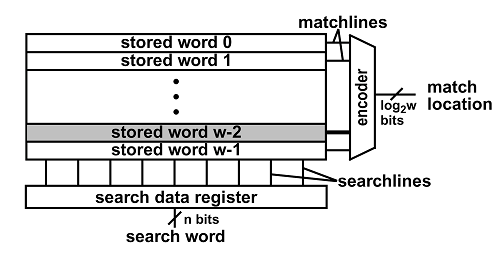


Figure 1Conceptual view of a content-addressable memory containing

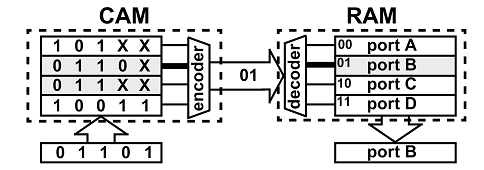


Figure 2CAM-based implementation

# ROM

**entity** ROM **is**

**port** **(**

ADDR **:** **in** std\_logic\_vector **(**3 **downto** 0**);** -- address input

DOUT **:** **out** std\_logic\_vector **(**4 **downto** 0**));** -- data output

**end** ROM**;**

**architecture** BEHAVIOR **of** ROM **is**

**type** ROMTABLE **is** **array** **(**0 **to** 15**)** **of** std\_logic\_vector **(**4 **downto** 0**);**

-- internal table

**constant** romdata **:** romtable **:=** **(**

"10101"**,** -- data for address 0

"11111"**,** -- data for address 1

"10101"**,**

"11110"**,**

"10110"**,**

"10101"**,**

"11010"**,**

"10010"**,**

"10110"**,**

"11101"**,**

"10110"**,**

"10111"**,**

"00110"**,**

"11101"**,**

"10010"**,**

"10110" -- data for address 15

**);**

**begin** -- BEHAVIOR

**process** **(**ADDR**)**

**begin** -- process

DOUT **<=** romdata**(to\_integer(**unsigned**(**ADDR**)));**

**end** **process;**

**end** BEHAVIOR**;**