09-FSM-State_Machines

August 22, 2019

1 09 - FSM - State Machines

1.1 General

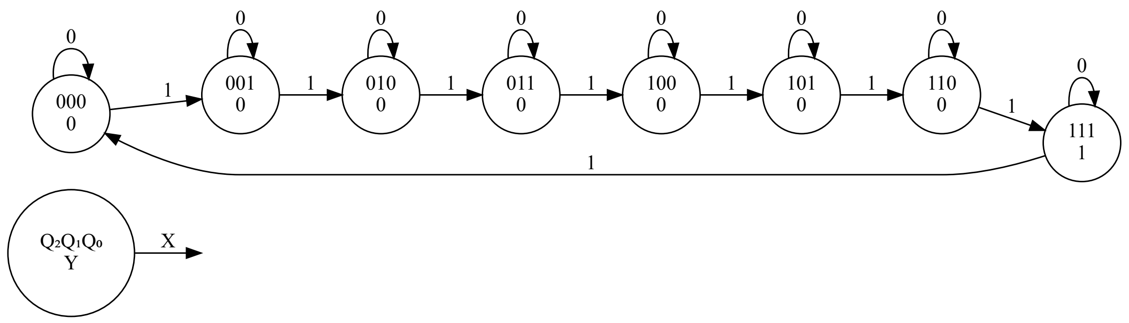
Al synchronous systems need to comply with the following conditions:

- All Flipflops use the same clock signal source
- No Flipflop has asynchronous inputs

1.2 Moore Machines

- Flipflops save the internal state
- combinatorial logic bloc determines the future inner state depending to the current inner state and the current inputs
- combinatorial logic bloc determines the output signals depending on the current inner state

1.2.1 Example counter

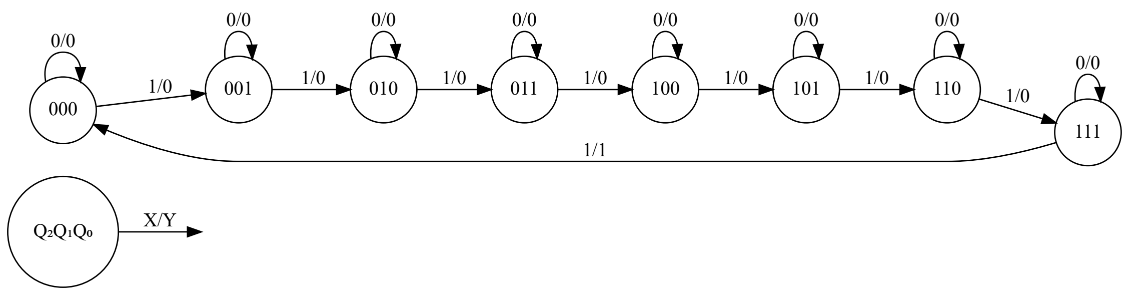


1.3 Mealy Machines

In contrast to Moore machines, Mealy machines have outputs that can also depend on the inputs.

```
<IPython.core.display.HTML object>
```

1.3.1 Example Counter



1.4 Creation of a Finite State Machine

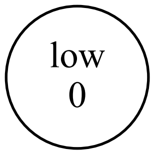
The state machines can be designed as follows: * Expressing the specification by means of a state graph, * Fill in the status table, * Reduction of the number of states, * Coding of the states, * Realization of the logical circuit. The most delicate thing is to develop a state graph based on the system's specifications. For this purpose 3 methods are listed below.

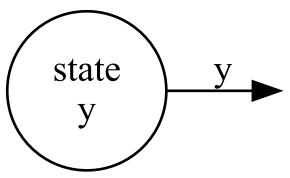
1.4.1 Development from a random state

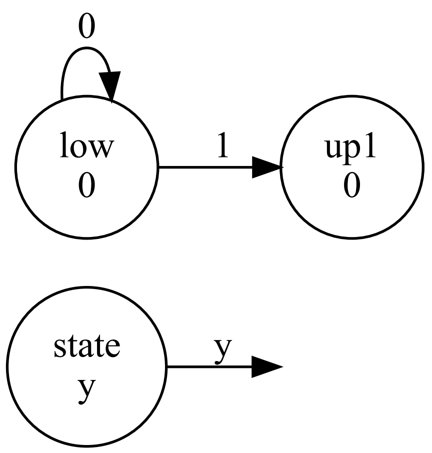
One way to create a state graph is to proceed from state to state. To do this - a certain state of the machine can be defined. - For this state, all possible input conditions (2^n cases for n inputs) must be analyzed and the corresponding arrows drawn. Depending on the case, new states are created. - Gradually carry out all not yet treated states, analyze all possible input conditions (2^n cases for n inputs) and draw the corresponding arrows. Each time a new state is created, it must be checked whether it already exists in the developing graph. Repeat this step until the graph is complete.

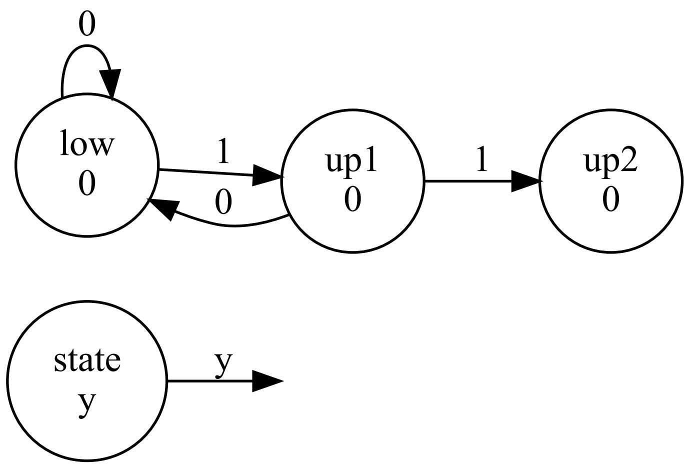
Remark There may be situations in which certain input conditions cannot occur. Significantly, this is the case when several input signals cannot change simultaneously due to the structure. In this case less than 2^n arrows lead away from each state.

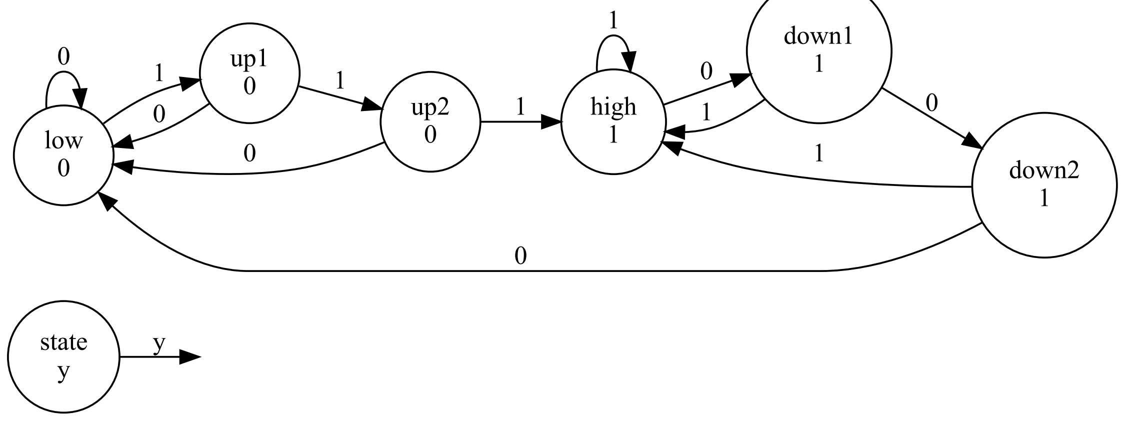
Example Debouncing Circuit See figures below






0
low 0
1
up1 0
state y
y

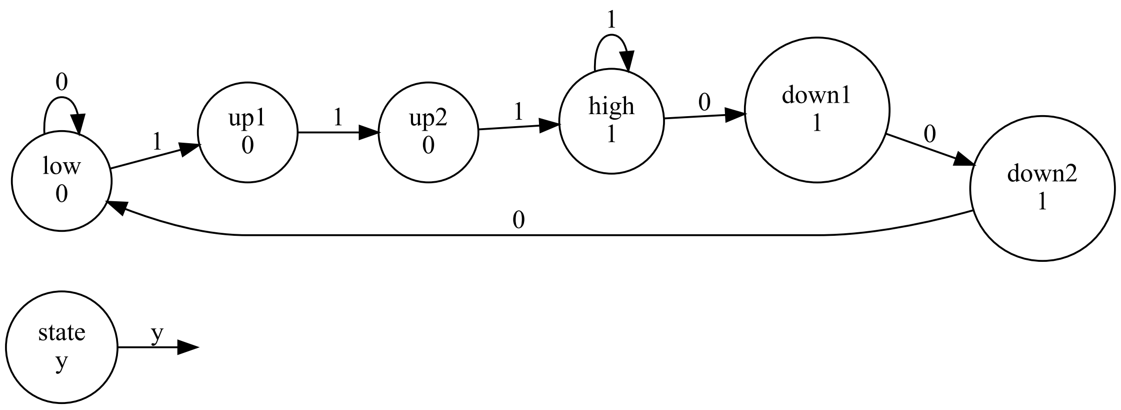

0
low 0
1
0
up1 0
1
up2 0
state y
y


0
low 0
1
0
up1 0
1
0
up2 0
1
1
high 1
0
1
down1 1
0
1
down2 1
0
state y
y

1.4.2 Development from a scenario

The second way to develop a state graph is to define a scenario. To do this * a scenario can be defined that covers a large part of the requirements specification. * The scenario must be developed in the form of a graph, with the states corresponding to the stages of the scenario and the arrows corresponding to the actions. * For each state defined in this way, all possible input conditions (2^n cases for n inputs) must be analysed and the arrows not yet present drawn. Sometimes new states arise, which in this case have to be analyzed completely.

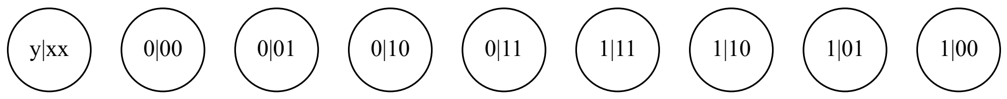
Example Debouncing Circuit See figure below

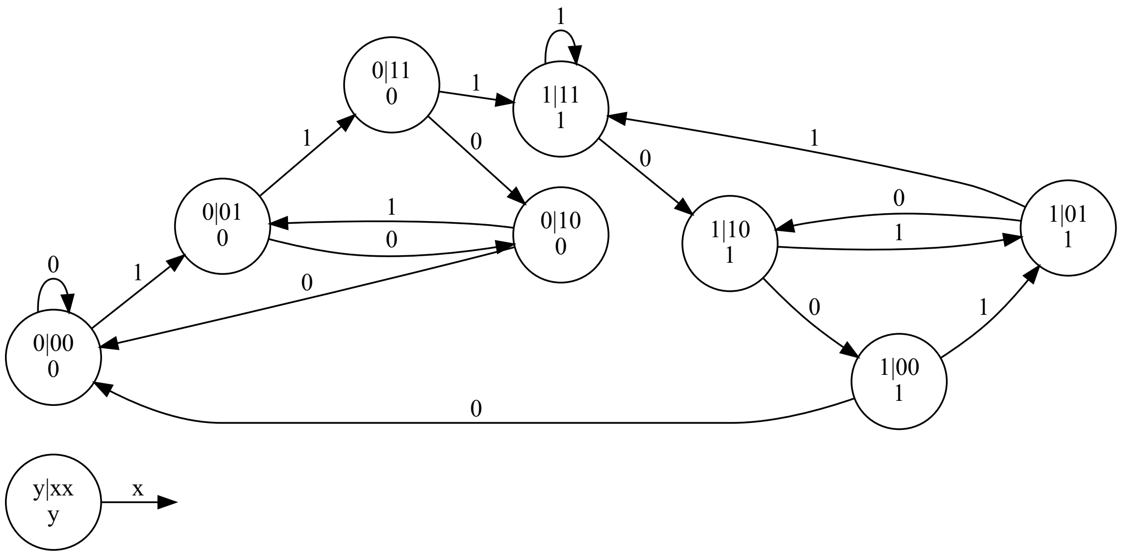


1.4.3 Development from a State list

In order to develop a state graph in the third way, its states must be known in advance. For this * the events or states to be stored are determined. * A state is drawn for each event to be stored. * For each defined state, all possible input conditions (2^n cases for n inputs) are analyzed and the corresponding arrows are drawn.

Example Debouncing Circuit See figure below





0|00
0
0|01
0
0|11
0
0|10
0
1|11
1
1|10
1
1|01
1
1|00
1
y|xx
y
x