Lecture 9

Finite State machine representation of sequential circuits

The story so far:

- All circuits can be represented by Boolean equations and vice versa.
- Sequential circuits, in which the outputs are fed back into the inputs have variables that appear on both the right and left hand side of the variables: eg

$$Q = (R \cdot P)' \text{ and } P = (S \cdot Q)'$$

- For one case R=S=1 there is no unique solution

The story so far:

- For sequential circuits an interpretation of the Boolean equations cannot be made without some model of time.
- A simple time delay model allows us to analyse the behaviour of the R-S flip flop with sufficient accuracy.
- However, we would prefer not to worry about time if possible.

The story so far:

- We can avoid worrying about time by using synchronous digital circuits.
- In a synchronous circuit a bank of D-type flip flops acts as a barrier between the inputs and the outputs.
- The flip flops are all triggered at the same time using a common clock.

The story so far:

Using synchronous design techniques the equation pair:

$$R = A \cdot B + P \text{ and } P = (A + B)'$$

is interpreted as:

$$R_t = A_{t-1} \cdot B_{t-1} + P_{t-1} \text{ and } P_t = (A_{t-1} + B_{t-1})'$$

The time interval is measured by the system clock which triggers the filp-flops.

Given the initial conditions we can always solve the equations to determine the circuit behaviour.

The story so far:

Using synchronous design we can stick to Boolean algebra. The only think we need to ensure is that the clock doesn't run too fast!

The outputs of each block of combinatorial logic must reach their final correct values before the clock pulse arives to change the circuit state.

Now read on!

Finite State Machines

Any synchronous circuit can be modelled by a finite state machine, and vice versa.

A finite state machine in its most general form can be represented by a pair of equations:

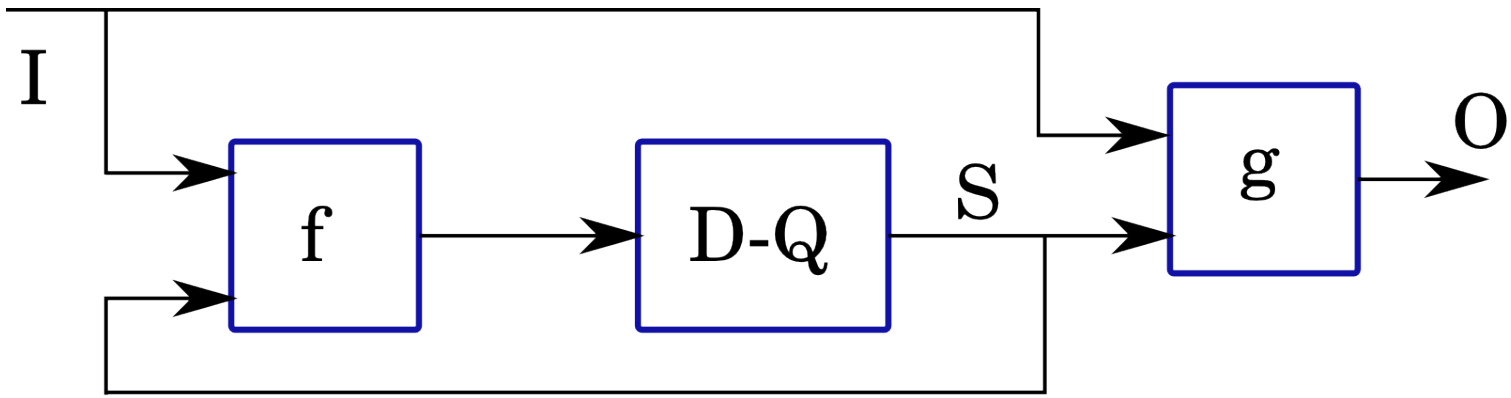
$$S_{t+1} = f(S_t, I_t)$$

$$O_{t+1} = g(S_t, I_t)$$

Where S_t is the state at time t and I_t the input at time t, and O_t is the output at time t, and f and g are functions implemented in combinatorial logic.

The Mealey Machine

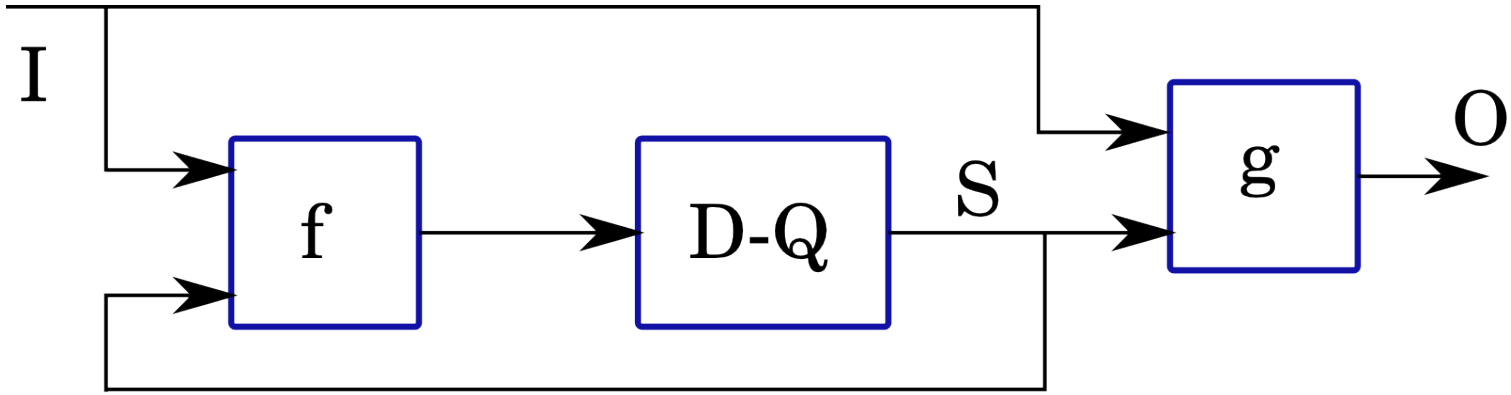
The most general form of the finite state machine is called the Mealey machine.



The boxes f and g are combinatorial circuits, and $D-Q$ is a set of flip flops which store the current state of the circuit as a binary number.

The Mealey Machine

The most general form of the finite state machine is called the Mealey machine.

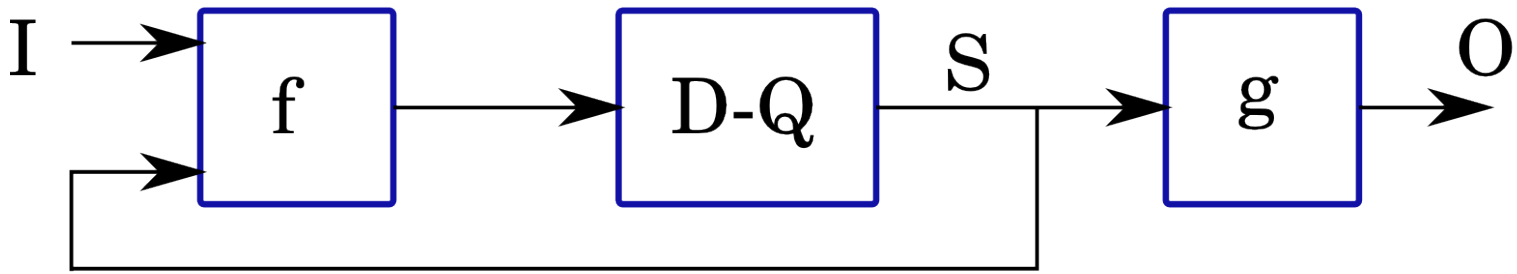


Box *f* determines the sequence that the machine goes through, and is called the state sequencing or input logic.

Box *g* determines its outputs, and is called the output logic.

The Moore Machine

A simpler model - called the Moore machine - without the connection between the input (I) and the output logic (*g*) is often preferred as it provides safer (more robust) design:



Spikes present in I are blocked from reaching *g*.

Why use finite state machines

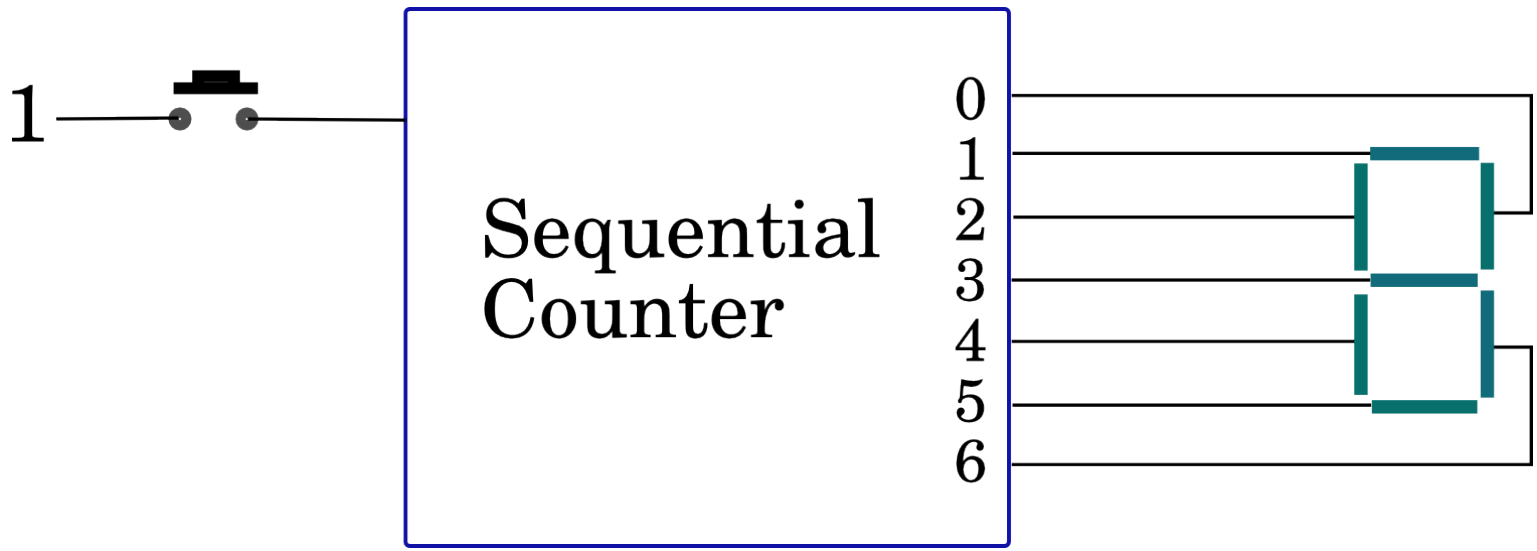
- The finite state machines give us a way specifying exactly what a circuit should do.
- Having designed the finite state machine we can apply a design methodology to create the actual circuit.
- FSMs play a crucial role in computer science as a modelling tool.

Sequential Circuit Design Methodology

1. Determine the number of states required
2. Determine the state transitions
3. Choose the way in which the states will be represented
4. Express the state sequencing logic as Boolean equations and minimise using Karnaugh Maps
5. Express the output logic as Boolean Equations and minimise

A Design Example

A manual counter with a 7 segment display:



Determining the Number of States

We determine the number of states required by examining the specification.

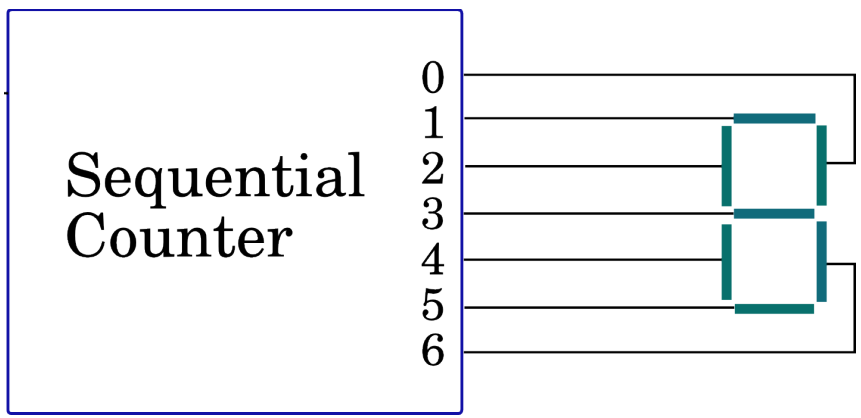
Let's suppose that our specification is of a counter that counts in order from 0 to 5 - such as might be found in a digital watch.

Clearly we will need six states which we will name $0 \cdots 5$, meaning:

- 0 - Zero displayed
- 1 - One displayed
- 2 - Two displayed
- etc

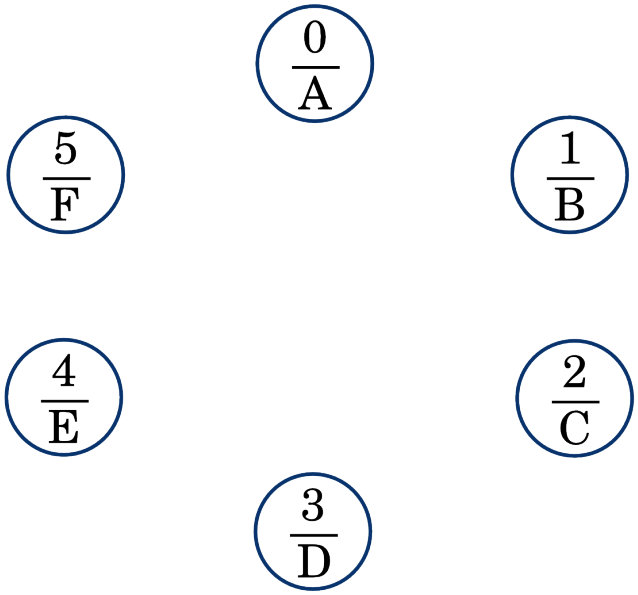
Specifying the states fully

We can now specify more fully what output we want from the circuit. Given that a one bit output of logic 1 will illuminate a segment, then the output [1,1,1,0,1,1,1] will display the digit “0”. We can fill up a table to complete this part of the specification.



State	Output
0	A = [1,1,1,0,1,1,1]
1	B = [1,0,0,0,0,0,1]
	etc

The Finite State Machine Diagram

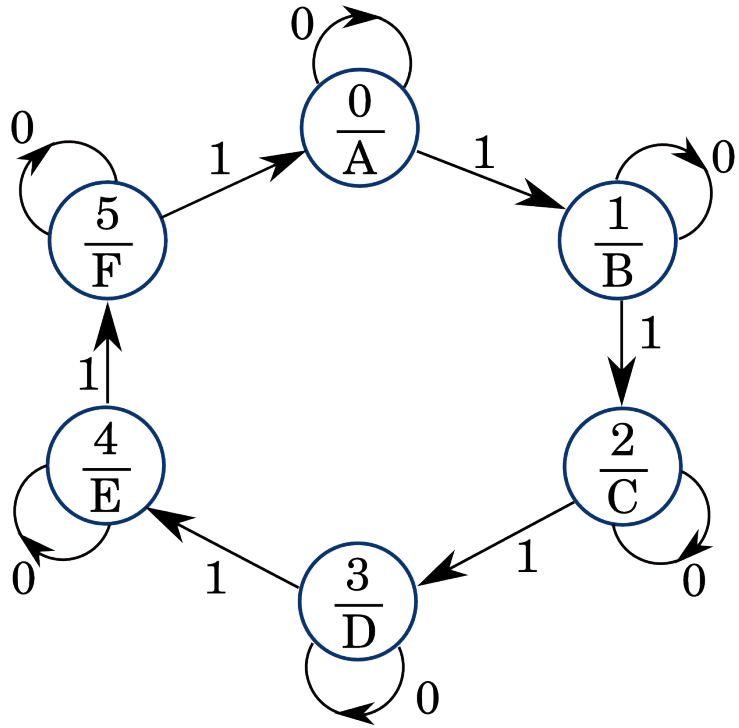


Output Vectors

A={1,1,1,0,1,1,1}
B={1,0,0,0,0,0,1}
C={1,1,0,1,1,1,0}
D={1,1,0,1,0,1,1}
E={0,0,1,1,0,0,1}
F={0,1,1,1,0,1,1}

By convention, in a Moore machine we write the state name (upper) and output (lower) in the finite state machine diagram circles.

The Finite State Machine Diagram



Output Vectors

A={1,1,1,0,1,1,1}
B={1,0,0,0,0,0,1}
C={1,1,0,1,1,1,0}
D={1,1,0,1,0,1,1}
E={0,0,1,1,0,0,1}
F={0,1,1,1,0,1,1}

By using the specification we can now add the transitions and their associated inputs.

The Specification is now Completely formalised

Designing the finite state machine diagram is a process of refining an informal specification into a formal specification of what will be built.

From now on there are some design choices to be made but the task could be done automatically.

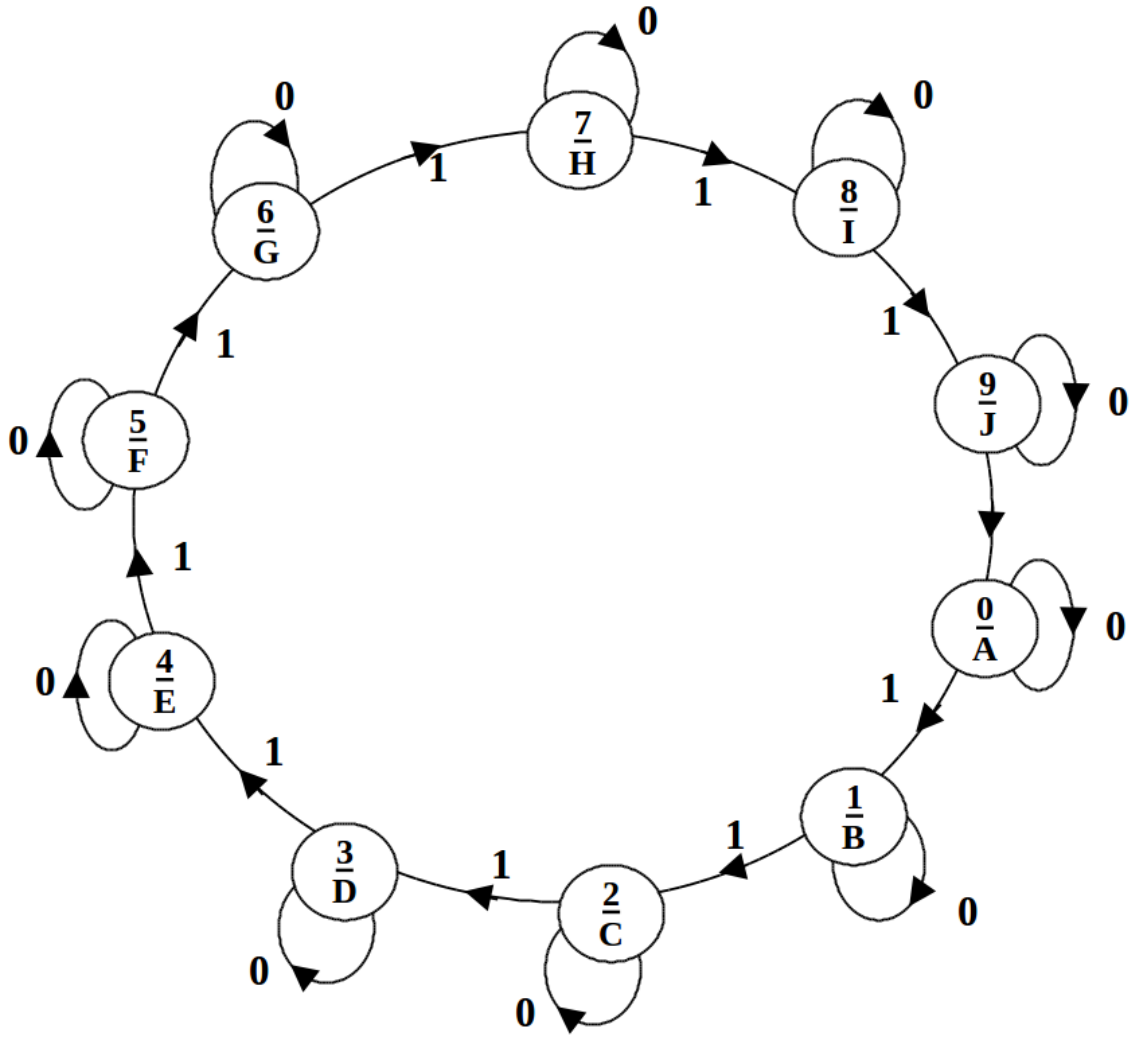
Problem Break

- If we wanted our counter to count from 0 to 9 rather than from 0 to 6 how many states would it need?
- What would the finite state machine look like?
- How many flip flops would we need to represent the states?

Problem Break

10 States

4 flip flops



State representation

- We now have to decide how we are to represent each state.
- One flip flop per state would be an expensive possibility.
- Three flip flops can be though of as a three bit memory, and therefore could represent eight states.
- The values Q_2,Q_1 and Q_0 define the state.

State Assignment

One design choice is how we represent the states as binary numbers. For example we could choose:

$Q_2 Q_1 Q_0$	State
0 0 0	0
0 0 1	5
0 1 0	Unused
0 1 1	4
1 0 0	3
1 0 1	Unused
1 1 0	1
1 1 1	2

State Assignment

For simplicity we will make the obvious choice of state assignments:

$Q_2Q_1Q_0$	State	Minterm
0 0 0	0	$Q_2' \cdot Q_1' \cdot Q_0'$
0 0 1	1	$Q_2' \cdot Q_1' \cdot Q_0$
0 1 0	2	$Q_2' \cdot Q_1 \cdot Q_0'$
0 1 1	3	$Q_2' \cdot Q_1 \cdot Q_0$
1 0 0	4	$Q_2 \cdot Q_1' \cdot Q_0'$
1 0 1	5	$Q_2 \cdot Q_1' \cdot Q_0$
1 1 0	unused	
1 1 1	unused	

(A good idea when answering exam questions!)

States and Minterms

Each state is now defined by a Boolean equation.

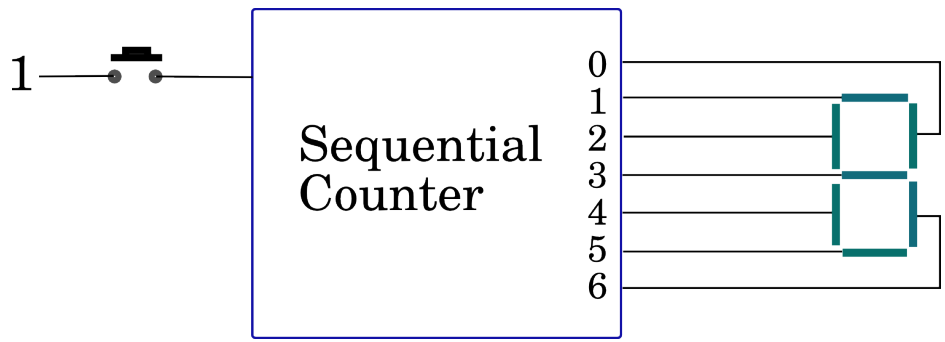
Let the variable S_i be equal to 1 when the circuit is in state i, and zero otherwise, then:

- $S_0 = Q_2' \cdot Q_1' \cdot Q_0'$
- $S_1 = Q_2' \cdot Q_1' \cdot Q_0$
- $S_2 = Q_2' \cdot Q_1 \cdot Q_0'$
- etc.

Output Logic Equations

Since the output depends only on the state we can directly write down its Boolean equations.

For example look at segment 1 (the top segment). It is illuminated when displaying digits 0, 2, 3 and 5:



Hence:

- $O_1 = S_0 + S_2 + S_3 + S_5$
- $O_1 = Q_2' \cdot Q_1' \cdot Q_0' + Q_2' \cdot Q_1' \cdot Q_0 + Q_2' \cdot Q_1 \cdot Q_0 + Q_2 \cdot Q_1' \cdot Q_0$

State Sequencing Logic Equations

The state sequencing logic equations can be read directly from the finite state machine model.

We use Boolean variables:

$N_i = 1$ meaning "The next state is S_i"

We can write an equation for each state change as follows:

- $N_0 = S_5 \cdot I + S_0 \cdot I'$
- $N_1 = S_0 \cdot I + S_1 \cdot I'$
- etc

Equations for the D inputs

For any given next state, we can find Boolean equations for the D inputs by looking at the state allocation table.

For example $Q_2 = 1$ when the state is 4 or 5

- $D_2 = N_4 + N_5$
- $D_1 = N_2 + N_3$
- $D_0 = N_1 + N_3 + N_5$

$Q_2 Q_1 Q_0$	State
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5

Equations for the D inputs

Substituting the N_i variables:

$$D_1 = N_2 + N_3 = S_1 \cdot I + S2 \cdot I' + S2 \cdot I + S3 \cdot I'$$

Substituting for the S_i variables

$$D_1 = Q_2' \cdot Q_1' \cdot Q_0 \cdot I + Q_2' \cdot Q_1 \cdot Q_0' \cdot I' + Q_2' \cdot Q_1 \cdot Q_0' \cdot I + Q_2' \cdot Q_1 \cdot Q_0 \cdot I'$$

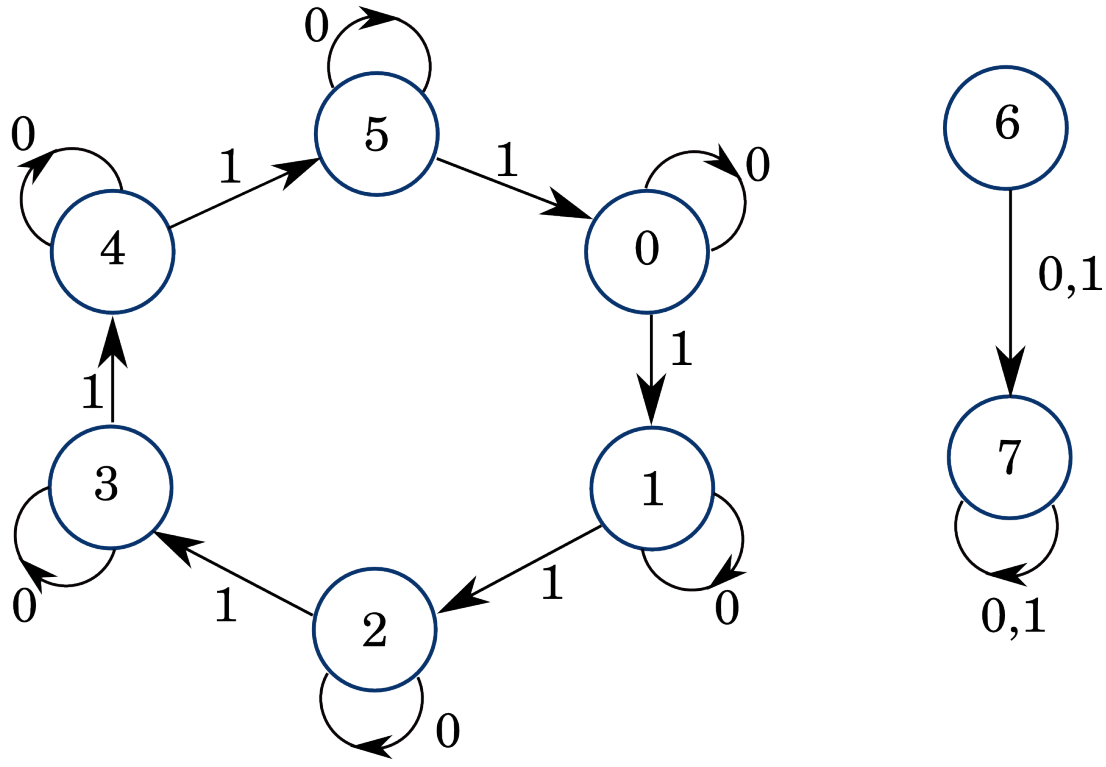
This is the boolean equation for D_1 in canonical form.

D_2 and D_0 can be found similarly

Unused States

- The unused states were designated don't care states. If we implement the equations directly using the canonical form they will have no effect.
- However, to get the minimum representation we will need to include them as don't cares in our Karnaugh maps.
- This can cause problems since the don't cares will have defined values in the final circuit.

A possible outcome of exploiting don't cares:



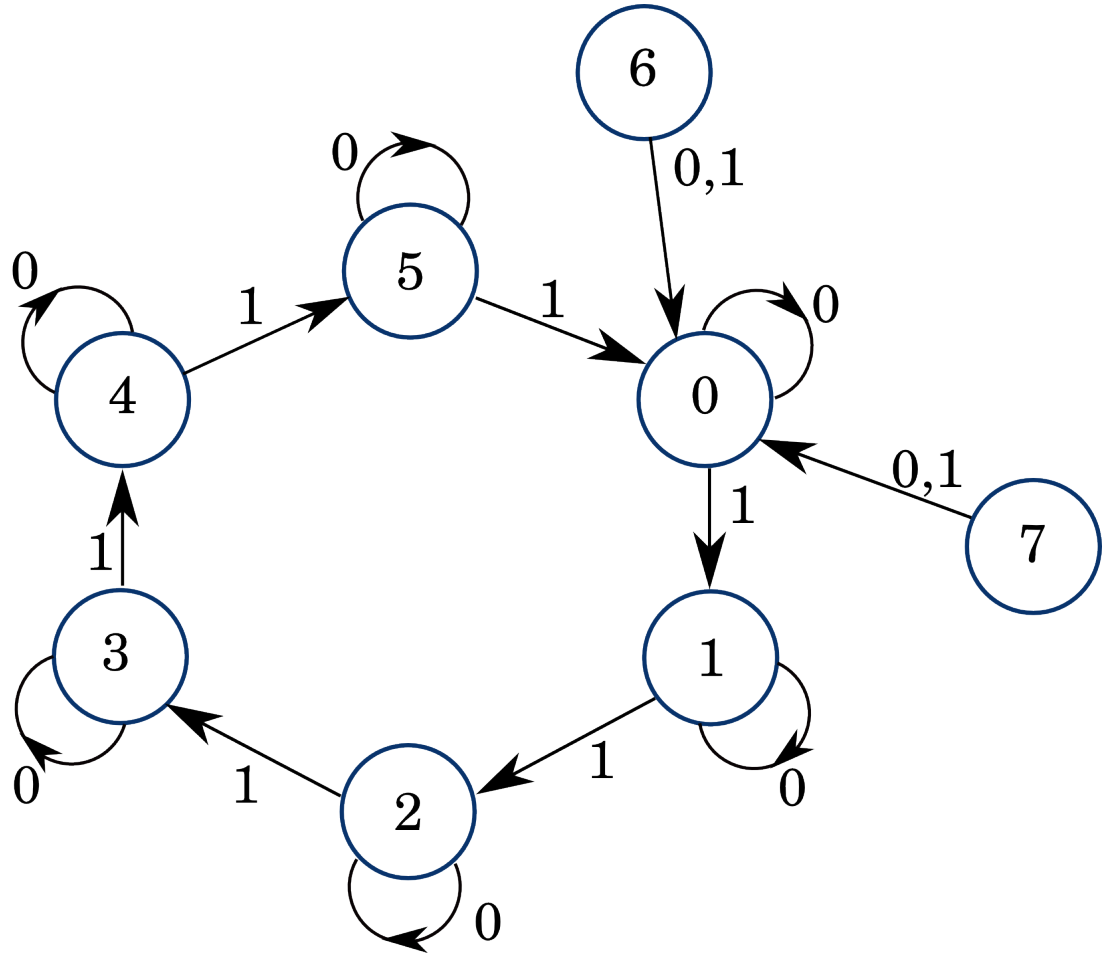
If the circuit gets into an unused state when switched on it will fail to operate.

Dealing with Unused States

If our unused states form an isolated maching we can try two possible approaches:

- See if a simple modification will fix the problem without increasing the size too much. (hacking!)
- Explicitly include the unused states in our finite state machine, and re-do the design.

Explicit allocation of Unused States:



Choosing the Best State Assignment

- The choice of state allocation can have a big impact on the size of the final circuit.
- If the number of states is small, then we can try all possible allocations
- The number of ways we could allocate the eight possibilities to the six states is 56
- Fortunately we can eliminate many of these to avoid tediously repeating the design problem.

Isomorphic Assignments

Since flip-flops usually always have complementary outputs (Q and Q') The same circuit will result from complementing each column of our assignment.

State	$Q_2Q_1Q_0$	Isomorphs
0	0 0 0	100 010 110
1	0 0 1	101 011 111
2	0 1 0	110 000 100
3	0 1 1	111 001 101
4	1 0 0	000 110 010
5	1 0 1	001 111 011

The isomorphs are formed by:

Inverting Column 1

Inverting Column 2

Inverting Columns 1 & 2

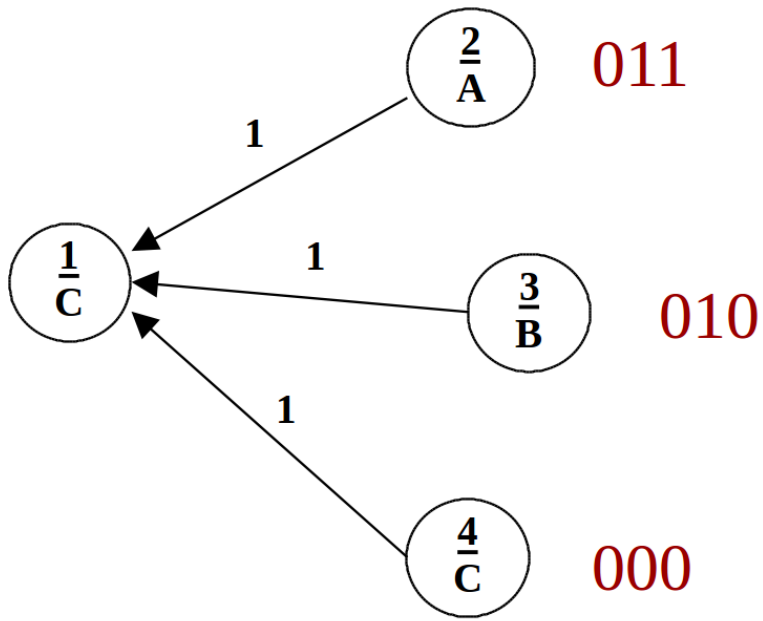
Larger Problems

- As the number of states increases the use of exhaustive search becomes less attractive.
- For a problem with 12 states, we need 4 flip flops, and the number of possible allocations becomes 1820.
- For a problem with 19 states, we need 5 flip flops, and the number of possible assignments is greater than 8 million.
- Any larger problem is infeasible even for automatic methods.

Heuristic Rules for State Allocation

There are two rules that should give good Karnaugh map minimisations for the sequencing logic.

First, all those states that have the same next state for the same input should be given adjacent state assignments



Heuristic Rules for State Allocation

Second, the next states of a state produced by applying adjacent input conditions should be given adjacent state assignments.