Digital Electronics: Sequential Logic

Synchronous State Machines 2

State Assignment

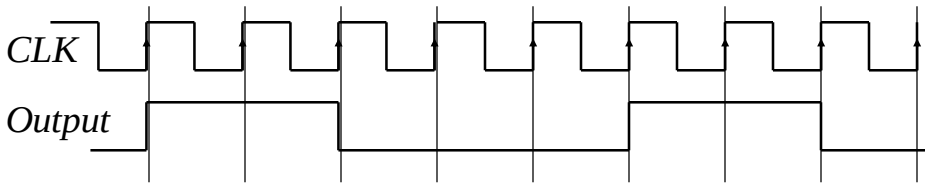
- As we have mentioned previously, state assignment is not necessarily obvious or straightforward
  - Depends what we are trying to optimise, e.g.,
    - Complexity (which also depends on the implementation technology, e.g., FPGA, 74 series logic chips).
      - FF implementation may take less chip area than you may think given their gate level representation
      - Wiring complexity can be as big an issue as gate complexity
    - Speed
  - Algorithms do exist for selecting the 'optimising' state assignment, but are not suitable for manual execution

State Assignment

- If we have m states, we need at least $\log_2 m$ FFs (or more informally, bits) to encode the states, e.g., for 8 states we need a min of 3 FFs
- We will now present an example giving various potential state assignments, some using more FFs than the minimum

Example Problem

- We wish to investigate some state assignment options to implement a divide by 5 counter which gives a 1 output for 2 clock edges and is 0 for 3 clock edges



Sequential State Assignment

- Here we simply assign the states in an increasing natural binary count
- As usual we need to write down the state transition table. In this case we need 5 states, i.e., a minimum of 3 FFs (or state bits). We will designate the 3 FF outputs as c, b, and a
- We can then determine the necessary next state logic and any output logic.

Sequential State Assignment

Current state			Next state		
c	b	a	c'	b'	a'
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0

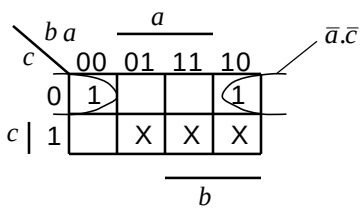
Unused states, 101, 110 and 111.

By inspection we can see:

The required output is from FF b

Plot k-maps to determine the next state logic:

For FF a:



$$D_a = \bar{a}.\bar{c}$$

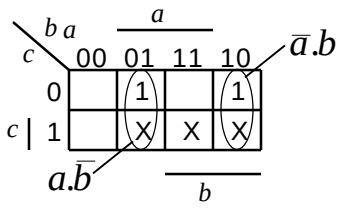
Sequential State Assignment

Current state / Next state

c	b	a	c'	b'	a'
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0

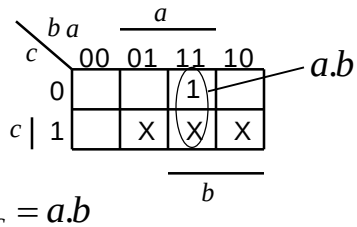
Unused states, 101, 110 and 111.

For FF b:



$$D_b = \bar{a}.b + a.\bar{b} = a \oplus b$$

For FF c:



$$D_c = a.b$$

Sliding State Assignment

Current state / Next state

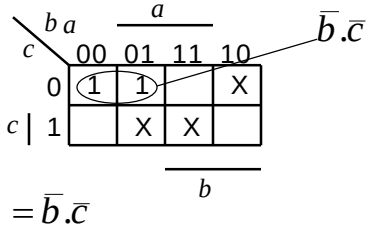
c	b	a	c'	b'	a'
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	1	1	0
1	1	0	1	0	0
1	0	0	0	0	0

Unused states, 010, 101, and 111.

By inspection we can see that we can use any of the FF outputs as the wanted output

Plot k-maps to determine the next state logic:

For FF a:



$$D_a = \bar{b}.\bar{c}$$

Sliding State Assignment

Current state			Next state		
c	b	a	c'	b'	a'
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	1	1	0
1	1	0	1	0	0
1	0	0	0	0	0

Unused states, 010, 101, and 111.

By inspection we can see that:

For FF b:

$$D_b = a$$

For FF c:

$$D_c = b$$

Shift Register Assignment

- As the name implies, the FFs are connected together to form a shift register. In addition, the output from the final shift register in the chain is connected to the input of the first FF:
  - Consequently the data continuously cycles through the register

Shift Register Assignment

Current state					Next state				
e	d	c	b	a	e'	d'	c'	b'	a'
0	0	0	1	1	0	0	1	1	0
0	0	1	1	0	0	1	1	0	0
0	1	1	0	0	1	1	0	0	0
1	1	0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	0	1	1

Unused states. Lots!

Because of the shift register configuration and also from the state table we can see that:

$$D_a = e$$
$$D_b = a$$
$$D_c = b$$
$$D_d = c$$
$$D_e = d$$

By inspection we can see that we can use any of the FF outputs as the wanted output

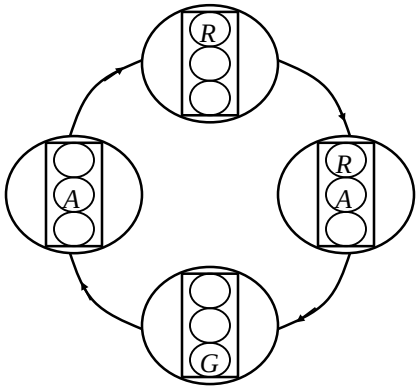
See needs 2 more FFs, but no logic and simple wiring

One Hot State Encoding

- This is a shift register design style where only one FF at a time holds a 1
- Consequently we have 1 FF per state, compared with $\log_2 m$ for sequential assignment
- However, can result in simple fast state machines
- Outputs are generated by ORing together appropriate FF outputs

One Hot - Example

- We will return to the traffic signal example, which recall has 4 states

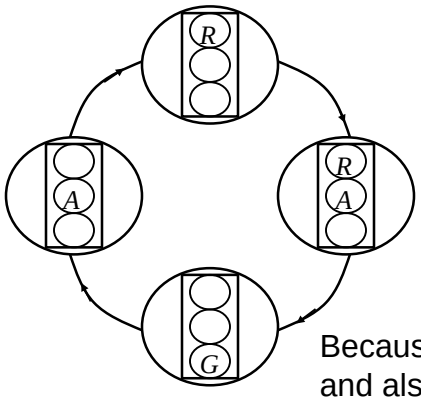


For 1 hot, we need 1 FF for each state, i.e., 4 in this case

The FFs are connected to form a shift register as in the previous shift register example, however in 1 hot, only 1 FF holds a 1 at any time

We can write down the state transition table as follows

One Hot - Example



Current state				Next state			
r	ra	g	a	r'	ra'	g'	a'
1	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	0	0	1
0	0	0	1	1	0	0	0

Unused states. Lots!

Because of the shift register configuration and also from the state table we can see that: $D_a = g \quad D_g = ra \quad D_{ra} = r \quad D_r = a$

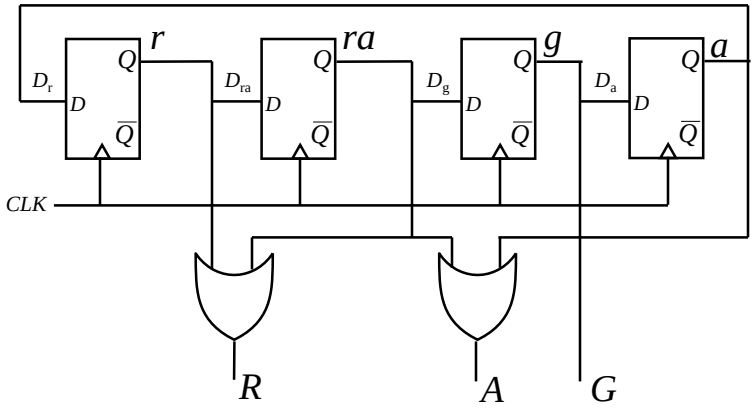
To generate the R, A and G outputs we do the following ORing:

$$R = r + ra \qquad A = ra + a \qquad G = g$$

One Hot - Example

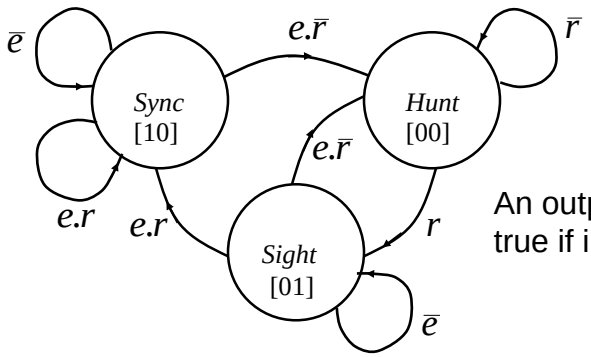
$D_a = g \quad D_g = ra \quad D_{ra} = r \quad D_r = a$

$R = r + ra \qquad A = ra + a \qquad G = g$



Tripos Example

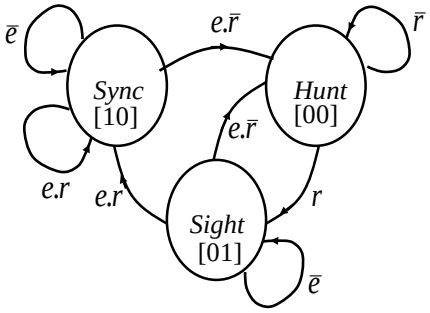
- The state diagram for a synchroniser is shown. It has 3 states and 2 inputs, namely e and r. The states are mapped using sequential assignment as shown.



FF labels
$[s_1\ s_0]$

An output, s should be true if in *Sync* state

Tripos Example



Unused state 11

From inspection, $s = s_1$

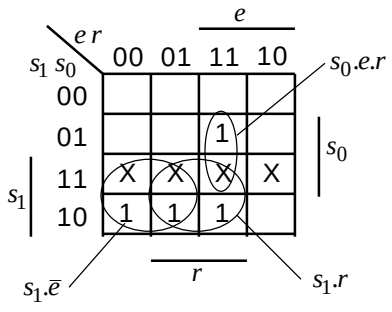
Current state		Input		Next state	
s_1	s_0	e	r	s_1'	s_0'
0	0	X	0	0	0
0	0	X	1	0	1
0	1	0	X	0	1
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	X	1	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	X	X	X	X

Tripos Example

Current state		Input		Next state	
s_1	s_0	e	r	s_1'	s_0'
0	0	X	0	0	0
0	0	X	1	0	1
0	1	0	X	0	1
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	X	1	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	X	X	X	X

Plot k-maps to determine the next state logic

For FF 1:



$$D_1 = s_1.\bar{e} + s_1.r + s_0.e.r$$

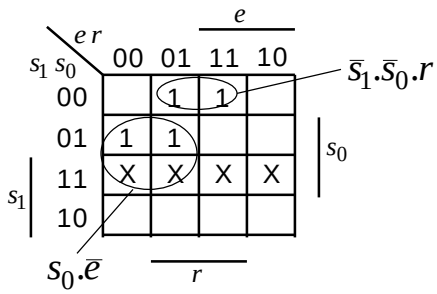
Tripos Example

Current state | Input | Next state

s_1	s_0	e	r	s_1'	s_0'
0	0	X	0	0	0
0	0	X	1	0	1
0	1	0	X	0	1
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	X	1	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	X	X	X	X

Plot k-maps to determine the next state logic

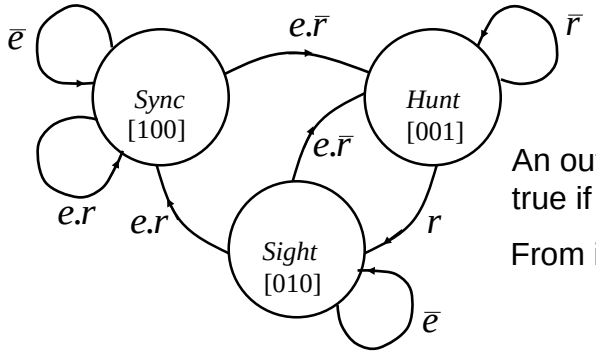
For FF 0:



$$D_0 = s_0.\bar{e} + \bar{s}_1.\bar{s}_0.r$$

Tripos Example

- We will now re-implement the synchroniser using a 1 hot approach
- In this case we will need 3 FFs

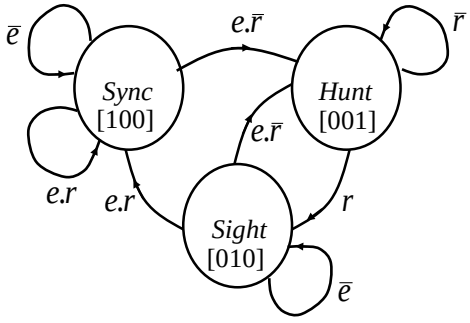


FF labels
$[s_2\ s_1\ s_0]$

An output, s should be true if in *Sync* state

From inspection, $s = s_2$

Tripos Example



Current state			Input		Next state		
s_2	s_1	s_0	e	r	s_2'	s_1'	s_0'
0	0	1	X	0	0	0	1
0	0	1	X	1	0	1	0
0	1	0	0	X	0	1	0
0	1	0	1	0	0	0	1
0	1	0	1	1	1	0	0
1	0	0	0	X	1	0	0
1	0	0	1	0	0	0	1
1	0	0	1	1	1	0	0

Remember when interpreting this table, because of the 1-hot shift structure, only 1 FF is 1 at a time, consequently it is straightforward to write down the next state equations

Tripos Example

Current state			Input		Next state		
s_2	s_1	s_0	e	r	s_2'	s_1'	s_0'
0	0	1	X	0	0	0	1
0	0	1	X	1	0	1	0
0	1	0	0	X	0	1	0
0	1	0	1	0	0	0	1
0	1	0	1	1	1	0	0
1	0	0	0	X	1	0	0
1	0	0	1	0	0	0	1
1	0	0	1	1	1	0	0

For FF 2:

$$D_2 = s_1.e.r + s_2.\bar{e} + s_2.e.r$$

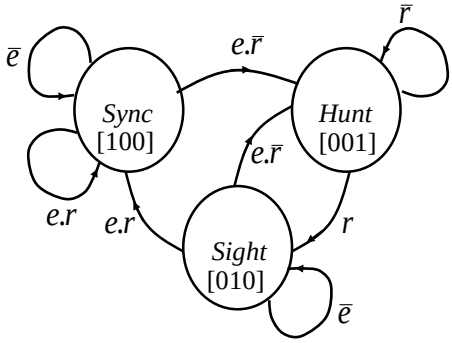
For FF 1:

$$D_1 = s_0.r + s_1.\bar{e}$$

For FF 0:

$$D_0 = s_0.\bar{r} + s_1.e.\bar{r} + s_2.e.\bar{r}$$

Tripos Example



Note that it is not strictly necessary to write down the state table, since the next state equations can be obtained from the state diagram

It can be seen that for each state variable, the required equation is given by terms representing the incoming arcs on the graph

For example, for FF 2: $D_2 = s_1.e.r + s_2.\bar{e} + s_2.e.r$

Also note some simplification is possible by noting that:

$s_2 + s_1 + s_0 = 1$ (which is equivalent to e.g., $s_2 = \overline{s_1 + s_0}$)

Tripos Example

- So in this example, the 1 hot is easier to design, but it results in more hardware compared with the sequential state assignment design