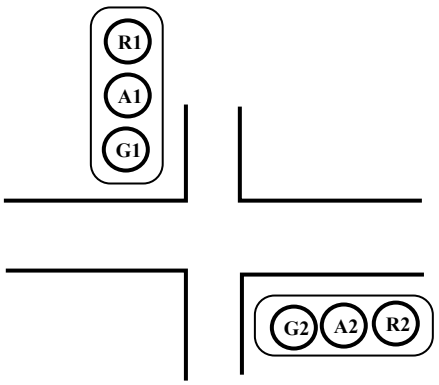
Department of Computing
Course DOC 112 -- Hardware

Lecture 10: Traffic Lights -- A Design Example

In this lecture we will work through a design example from problem statement to digital circuits.

The Problem: The traffic department is trying out a new system of traffic lights. We have to design a synchronous digital circuit, a Moore Machine, which operates this new type of traffic light at a simple road crossing.

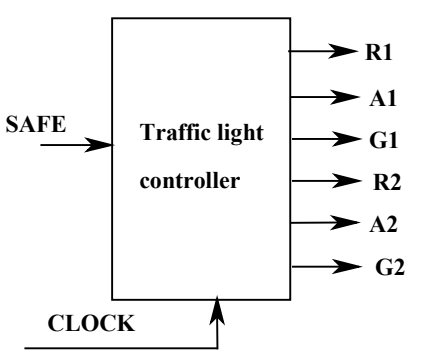


N-S	E-W
(SAFE)	
RED	GREEN
RED	AMBER
GREEN	RED
AMBER	RED
RED	GREEN
(UNSAFE)	
RED	GREEN
RED	AMBER
RED	RED
GREEN	RED
AMBER	RED
RED	RED
RED	GREEN

There are six lights to operate. The **Red**, **Amber**, and **Green** lights in the North-South direction will be designated as **R1, A1, G1**. Similarly, the lights in the East-West direction will be called **R2, A2,** and **G2**. When the digital signals are in the **Logic-1** state they turn their respective lights on, otherwise the lights are off. A digital clock signal will be supplied and at each clock pulse the lights should change according the schedule given above. There are two types of road crossing: safe ones that require one sequence, and dangerous ones that require another (delayed green) sequence. One digital input signal called **SAFE** will indicate whether the road crossing is safe. Thus, we have a one-input, six-output synchronous system to design.

Step 1: Understand the problem and decide how many states you need.

Here, "understanding the problem" refers to the understanding of the verbally described problem and its translation into digital circuit terms. Usually, the determination of the number of required states is not a trivial problem; and the determination of the minimum number of states may be very difficult. Probably, a reasonable approach is to find a number of states for which a state transition diagram can be constructed and then look at the problem again because, possibly, we can discover that some states are duplicated and thus can be eliminated. Our problem is simple enough, so this will not happen here.



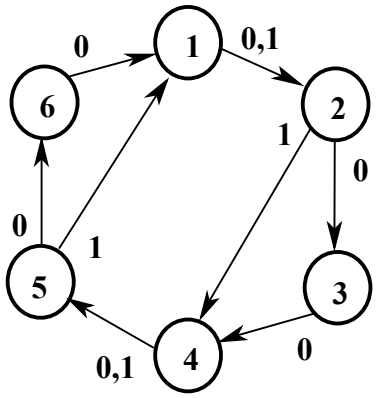
SAFE=0		SAFE=1	
<-1-> R A G	<-2-> R A G	<-1-> R A G	<-2-> R A G
1 0 0	0 0 1	1 0 0	0 0 1
1 0 0	0 1 0	1 0 0	0 1 0
1 0 0	1 0 0	0 0 1	1 0 0
0 0 1	1 0 0	0 1 0	1 0 0
0 1 0	1 0 0		
1 0 0	1 0 0		
1 0 0	0 0 1	1 0 0	0 0 1

Looking at the transition table, we see that there are six states in the first column (dangerous intersection) and four states in the second. However we do not need **ten** states because all four states

in the second column (the same six outputs) are included in the six states of the first column. Hooray! We need only six states. Let us number them **1** to **6** in the order they are shown in the state transition table.

Step 2. Construct the state transition diagram (ignore outputs)

STATE	N-S <-1-> R A G	E-W <-2-> R A G	
1	1 0 0	0 0 1	
2	1 0 0	0 1 0	
3	1 0 0	1 0 0	(A)
4	0 0 1	1 0 0	
5	0 1 0	1 0 0	
6	1 0 0	1 0 0	(B)



Two states (3 and 6), labelled A and B, have exactly the same traffic light outputs. Could they be merged as one state? The answer is no, unfortunately, because the state after 3 is 4 while the state after 6 is 1.

Step 3: Select the type and number of flip-flops for the circuit.

Since the number of states is equal to six, the minimum number of flip-flops, which can support six states, is three. The maximum number of flip-flops one may use is six (one flip-flop per state). For this design example we will use three D-type flip-flops. There will be two unused states.

Step 4: Assign state numbers to flip-flop outputs and construct the transition table.

While there are some heuristic rules for assigning states to flip-flop outputs, they are difficult to apply and do not guarantee a minimum circuit. We will minimise the K-maps only for **1**. Therefore, we will not use the two states **000** and **001**, which will be the two unused states 7 and 8. The idea behind this choice is that a large number of **1**s may provide easier minimisation so we use states with few **1**s for the unused states. The rest of the flip-flop outputs are assigned in order while constructing the transition table.

SAFE	State(t_n)	--> State(t_{n+1})	SAFE	$Q_i(t_n)$				Flip-Flop $Q_i(t_{n+1})$
0	1	2	0	000	(7)			XXX
0	2	3	0	001	(8)			XXX
0	3	4	0	010	(1)	-->	(2)	0 1 1
0	4	5	0	011	(2)	-->	(3)	1 0 0
0	5	6	0	100	(3)	-->	(4)	1 0 1
0	6	1	0	101	(4)	-->	(5)	1 1 0
0	7	U	0	110	(5)	-->	(6)	1 1 1
0	8	U	0	111	(6)	-->	(1)	0 1 0
1	1	2	1	000	(7)			XXX
1	2	4	1	001	(8)			XXX
1	3	U	1	010	(1)	-->	(2)	0 1 1
1	4	5	1	011	(2)	-->	(4)	1 0 1
1	5	1	1	100	(3)			XXX
1	6	U	1	101	(4)	-->	(5)	1 1 0
1	7	U	1	110	(5)	-->	(1)	0 1 0
1	8	U	1	111	(6)			XXX

State Transitions Using State Numbers **Assignment of Flip-flop Outputs**

We may go now to the next step directly (fill out K-maps and minimise). But in order to avoid errors in transferring the data from the transition table to the K-maps, a rearranged transition table is constructed first.

The order of the signals **S** (**SAFE**) and $\mathbf{Q_i}$ (the flip-flop outputs) are rearranged according to the sequence they are entered into the table (instead of **00->01->10->11** we use **00->01->11->10**). Also, since we are using D-type flip-flops, the terms $\mathbf{Q_i(t_{n+1})}$ become simply $\mathbf{D_i}$

S	Q1	Q2	Q3	D1	D2	D3
0	0	0	0	X	X	X
0	0	0	1	X	X	X
0	0	1	1	1	0	0
0	0	1	0	0	1	1
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	1	0	1	0
0	1	1	0	1	1	1
1	1	0	0	X	X	X
1	1	0	1	1	1	0
1	1	1	1	X	X	X
1	1	1	0	0	1	0
1	0	0	0	X	X	X
1	0	0	1	X	X	X
1	0	1	1	1	0	1
1	0	1	0	0	1	1

D1

S,Q1 \ Q2,Q3	00	01	11	10
00	X	X	1	0
01	1	1	0	1
11	X	1	X	0
10	X	X	1	0

D2

S,Q1 \ Q2,Q3	00	01	11	10
00	X	X	0	1
01	0	1	1	1
11	X	1	X	1
10	X	X	0	1

D3

S,Q1 \ Q2,Q3	00	01	11	10
00	X	X	0	1
01	1	0	0	1
11	X	0	X	0
10	X	X	1	1

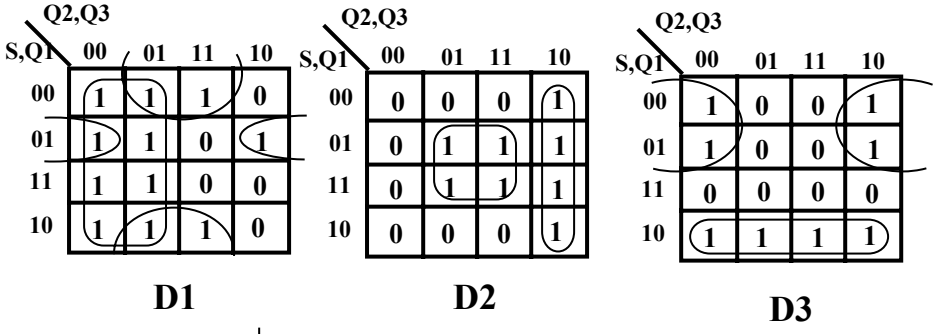
Step 5: Fill in K-Maps and determine the minimised expressions (see above).

The next step is to determine the required logic expressions for the three flip-flop inputs $\mathbf{D_1}$, $\mathbf{D_2}$, and $\mathbf{D_3}$. We use graphical method, i.e. the K-maps for the minimisation; however, any minimising algorithm can be used. One piece of caution should be mentioned here: we have to construct nine (!) combinational logic circuits. In addition to the three circuits for the flip-flop inputs, six simpler (only three-input) output control circuits must be built for the six traffic light signals **R1** to **G2**.

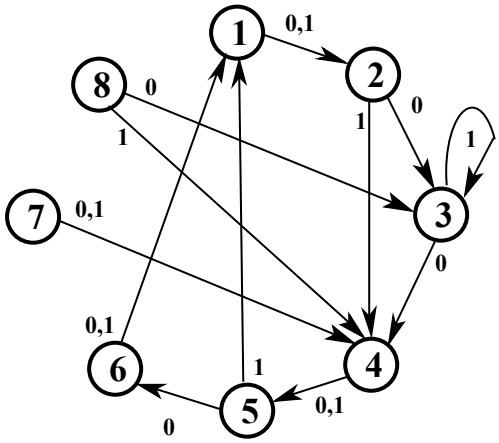
The K-Maps minimise each circuit individually; however, when multiple outputs are required, minimisation can arise by reusing expressions. For example, if for one circuit the term **S'•Q1•Q3'** appears, which appears for another circuit as well, this part of the circuit has to be built only once and the signal used as many times as needed. For this reason, we will not try to factorise the expressions until we have all nine expressions. Also, we have to check whether the circuit works.

Step 6: Construct the Diagram for all States (including don't cares).

Once minimisation of the K-Maps is determined and the indicated grouping of **1**s and **0**s are shown, we can replace the "don't care" outputs with the actual outputs (since this is exactly what the grouping show). Thus, we have now a completely defined sequential circuit and before we determine the Boolean expressions for the flip-flop inputs we should check whether the system behaves correctly even if it starts from one of the unused state. A convenient way of checking this is by constructing the complete transition diagram in which the unused states **000** and **001** by states **7** and **8** respectively.



	S	Q1	Q2	Q3	D1	D2	D3	
(7)	0	0	0	0	1	0	1	(4)
(8)	0	0	0	1	1	0	0	(3)
(2)	0	0	1	1	1	0	0	(3)
(1)	0	0	1	0	0	1	1	(2)
(3)	0	1	0	0	1	0	1	(4)
(4)	0	1	0	1	1	1	0	(5)
(6)	0	1	1	1	0	1	0	(1)
(5)	0	1	1	0	1	1	1	(6)
(3)	1	1	0	0	1	0	0	(3)
(4)	1	1	0	1	1	1	0	(5)
(6)	1	1	1	1	0	1	0	(1)
(5)	1	1	1	0	0	1	0	(1)
(7)	1	0	0	0	1	0	1	(4)
(8)	1	0	0	1	1	0	1	(4)
(2)	1	0	1	1	1	0	1	(4)
(1)	1	0	1	0	0	1	1	(2)

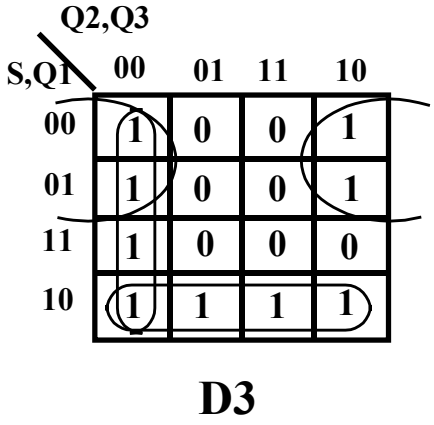


Disaster struck! If the **SAFE** input is logic **1** (safe crossing) and the system finds itself in state **3** then it will be stuck in state **3**. We cannot allow this, we have to go back and change some "don't care" bit(s) since we chose one value for it but could have chosen another. The state in trouble is State **3**, flip-flop outputs **100** and K-map entry **1100**. Looking at the K-maps, we can see that by changing the **0** indicated for this term in the K-map for **D3** to a **1** will cause minimal damage (i.e. will add one extra term to the expression.

The changed table is shown below. If we check in the transition table the entry for **1100** will change to **1101** and the system will move to State **4** from State **3** regardless of the **SAFE** input. We have repaired our system. All other illegal states ultimately end up in a proper state so we have a working system.

It would be possible to make a bit safer system by requiring that all illegal states must go immediately to State 3 or State 6 (**R1=R2=1**) in which case we have to go back to the K-maps and change other "don't care" entries to satisfy these conditions. We will not complicate our problem with this extra work.

	S	Q1	Q2	Q3	D1	D2	D3	
(7)	0	0	0	0	1	0	1	(4)
(8)	0	0	0	1	1	0	0	(3)
(2)	0	0	1	1	1	0	0	(3)
(1)	0	0	1	0	0	1	1	(2)
(3)	0	1	0	0	1	0	1	(4)
(4)	0	1	0	1	1	1	0	(5)
(6)	0	1	1	1	0	1	0	(1)
(5)	0	1	1	0	1	1	1	(6)
(3)	1	1	0	0	1	0	1	(4)
(4)	1	1	0	1	1	1	0	(5)
(6)	1	1	1	1	0	1	0	(1)
(5)	1	1	1	0	0	1	0	(1)
(7)	1	0	0	0	1	0	1	(4)
(8)	1	0	0	1	1	0	1	(4)
(2)	1	0	1	1	1	0	1	(4)
(1)	1	0	1	0	0	1	1	(2)

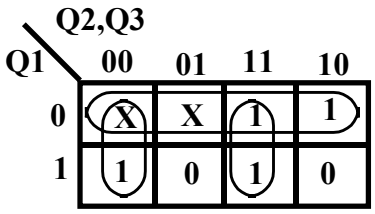


D3

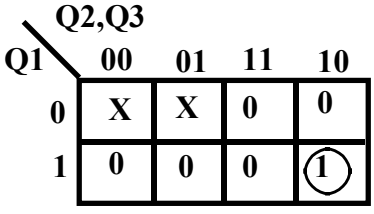
Step 7: Construct the Output Circuits in G

Unfortunately, there are six such circuits but fortunately they have three inputs only (Moore Machine). Their K-Maps can be filled out by the requirements of lights to be either **ON** or **OFF** for each given state. Here again we will start by ignoring the two unused states which will provide "don't care" outputs to find the minimised circuits. Again, filling out the K-maps with the selected **1**s and **0**s will give us the actual operation of the lights for states 7 and 8. We will have to look at these whether they are safe.

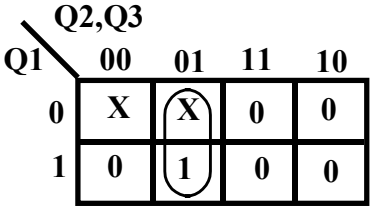
STATE	Q1	Q2	Q3	R1	A1	G1	R2	A2	G2
2	0	1	1	1	0	0	0	1	0
1	0	1	0	1	0	0	0	0	1
3	1	0	0	1	0	0	1	0	0
4	1	0	1	0	0	1	1	0	0
6	1	1	1	1	0	0	1	0	0
5	1	1	0	0	1	0	1	0	0



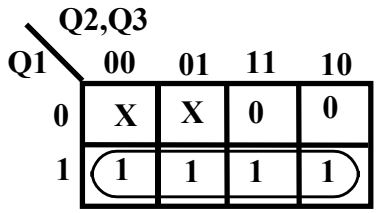
R1 = Q1' + Q2'•Q3' + Q2•Q3



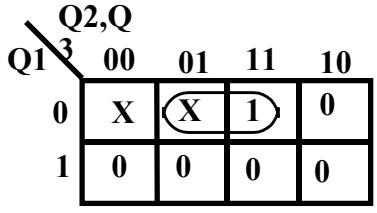
A1 = Q1•Q2•Q3'



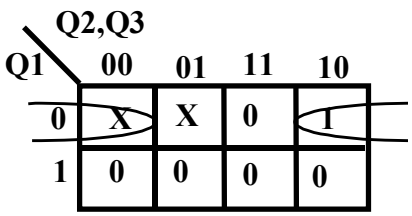
G1 = Q2'•Q3



R2 = Q1



A2 = Q1'•Q3



G2 = Q1'•Q3'

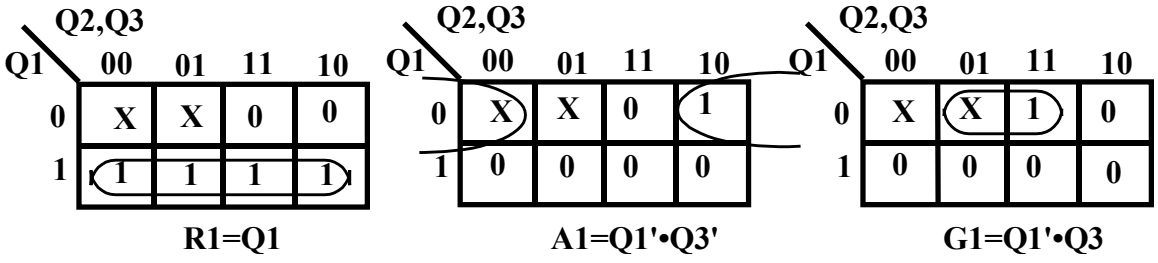
In summary we have the following circuits to build:

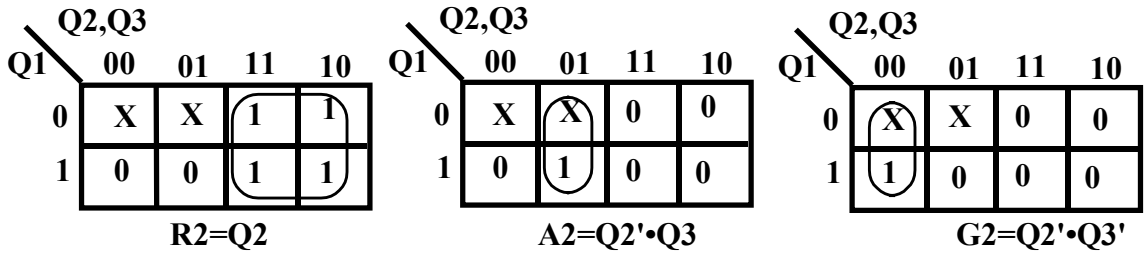
D1 = Q2' + S'•<u>Q1•Q3'</u> + <u>Q1'•Q3</u>
D2 = Q1•Q3 + <u>Q2•Q3'</u>
D3 = <u>Q2'•Q3'</u> + S'•Q3' + S•Q1'
R1 = Q1' + <u>Q2'•Q3'</u> + Q2•Q3
A1 = Q1•<u>Q2•Q3'</u> or Q2•<u>Q1•Q3'</u> The common terms are underlined
G1 = Q2'•Q3
R2 = Q1
A2 = <u>Q1'•Q3</u>
G2 = Q1'•Q3'

A Different State Assignment

If we want to try to find a simpler overall circuit, we may try different flip-flop assignments for the states. One idea is to minimise the output circuitry. We could, for example, make **R1=Q1** and **R2=Q2**, if these simple assignments will give us a correct complete state assignment. The third output, **Q3** has to be assigned such that all used states are distinct. One possible set of assignments are shown below:

Q1	Q2	Q3	R1	A1	G1	R2	A2	G2	STATE
0	0	0	X	X	X	X	X	X	7
0	0	1	X	X	X	X	X	X	8
0	1	1	0	0	1	1	0	0	4
0	1	0	0	1	0	1	0	0	5
1	0	0	1	0	0	0	0	1	1
1	0	1	1	0	0	0	1	0	2
1	1	1	1	0	0	1	0	0	3
1	1	0	1	0	0	1	0	0	6





The output circuits require only two-input **NAND** gates. But of course, we have to redesign the input circuitry with the new flip-flop assignments.

	S	Q1	Q2	Q3	D1	D2	D3	
(7)	0	0	0	0	X	X	X	
(8)	0	0	0	1	X	X	X	
(4)	0	0	1	1	0	1	0	(5)
(5)	0	0	1	0	1	1	0	(6)
(1)	0	1	0	0	1	0	1	(2)
(2)	0	1	0	1	1	1	1	(3)
(3)	0	1	1	1	0	1	1	(4)
(6)	0	1	1	0	1	0	0	(1)
(1)	1	1	0	0	1	0	1	(2)
(2)	1	1	0	1	0	1	1	(4)
(3)	1	1	1	1	X	X	X	
(6)	1	1	1	0	X	X	X	
(7)	1	0	0	0	X	X	X	
(8)	1	0	0	1	X	X	X	
(4)	1	0	1	1	0	1	0	(5)
(5)	1	0	1	0	1	0	0	(1)

D1

S,Q1 \ Q2,Q3	00	01	11	10
00	X	X	0	1
01	1	1	0	1
11	1	0	X	X
10	X	X	0	1

D2

S,Q1 \ Q2,Q3	00	01	11	10
00	X	X	1	1
01	0	1	1	0
11	0	1	X	X
10	X	X	1	0

D3

S,Q1 \ Q2,Q3	00	01	11	10
00	X	X	0	0
01	1	1	1	0
11	1	1	X	X
10	X	X	0	0

This circuit seems to be simpler than the first one.

D1 = S'•Q2' + Q3'

D2 = S'•Q1' + Q3

D3 = Q2' + Q1•Q3

R1 = Q1

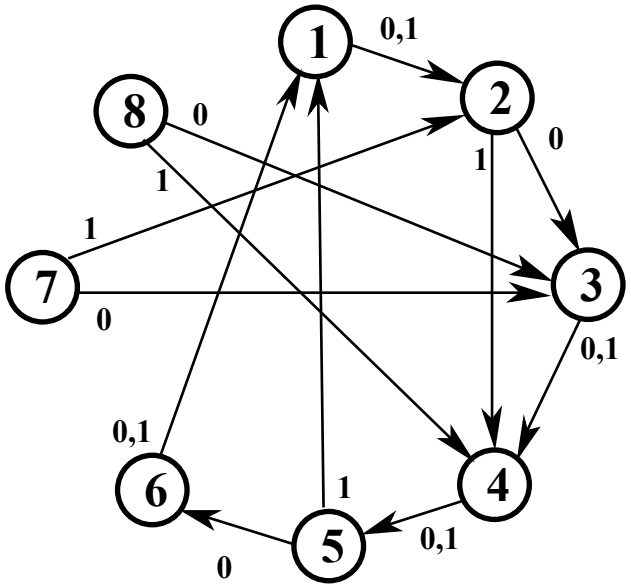
A1 = Q1'•Q3'

G1 = Q1'•Q3

R2 = Q2

A2 = Q2'•Q3

G2 = Q2'•Q3'



And the final circuit is: