COMS10015 lab. worksheet #8

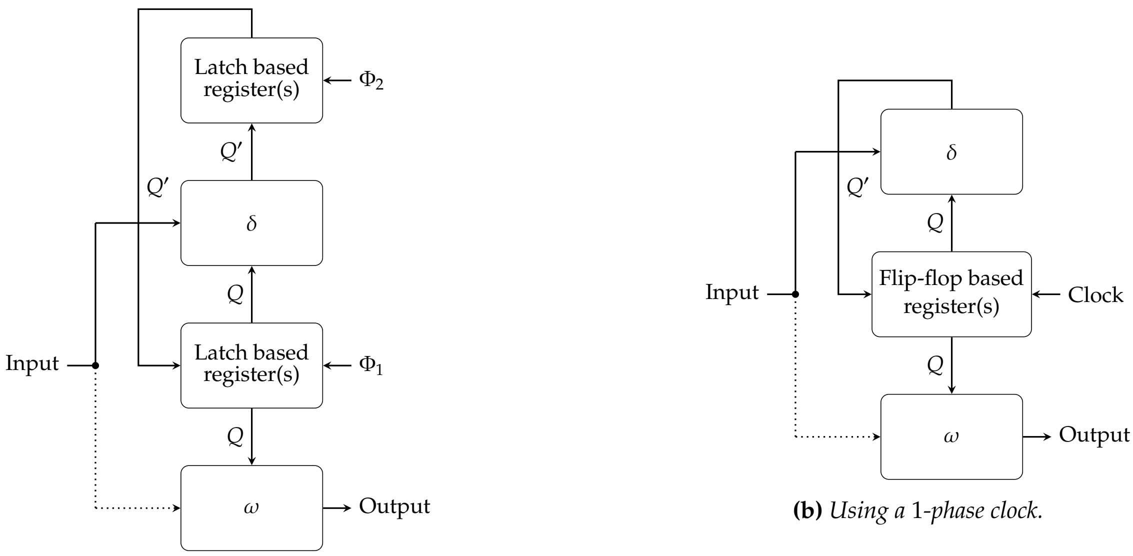
Although some questions have a written solution below, for others it will be more useful to experiment in a hands-on manner (e.g., using a concrete implementation). The file

https://assets.phoo.org/COMS10015_2025_TB-4/csdsp/sheet/lab-08_s.tar.gz

supports such cases.

§1. C-class, or core questions

▷ **S1[C].** First we need to take stock of the problem itself: there is basically one input (the emergency stop button, denoted rst) and six outputs (the traffic light values, denoted M_g, M_a and M_r for the main road and A_g, A_a and A_r for the access road). We *know* that Figure 1 can act as a starting point, in the sense that we just need to fill in each block with a problem-specific implementation.



(a) *Using a 2-phase clock.*

(b) *Using a 1-phase clock.*

Figure 1: *Two generic FSM frameworks (for different clocking strategies) into which one can place implementations of the state, δ (the transition function) and ω (the output function).*

Next we try to develop a precise description of the FSM behaviour. We need 7 states in total: $S_0, S_1, \ldots, S_5$ represent steps in the normal traffic light sequence, and S_6 represents an extra emergency stop state. Figure 2 shows both a tabular and diagrammatic description of the transition function; in essence, it is similar to the counter example (in the sense that it cycles from S_0 through to S_5 and back again) if $rst = 0$, but if $rst = 1$ in *any* state then we move to the S_6. As an aside, however, it is vital to view this description is *one* among several derived from what is (by design) a rather imprecise question. Put another way, we have made several assumptions and/or decisions. One example is the decision to use a separate emergency stop state, and have the FSM enter this as the next state of *any* current state if $rst = 1$; the red lights are both forced on as a result of being in the emergency stop state, therefore, rather than by rst per se. Another valid approach might be to have ω depend on rst as well (rather than only Q, turning it from a Moore-based into a Mealy-based FSM) and forcing the red lights on as soon as $rst = 1$ irrespective of what state the FSM is in. This is arguably more attractive, in the sense that now the emergency stop is instant: we no longer need to wait for the next clock cycle when the next state is latched. Likewise, we have opted to make the first state listed in the question (i.e., green on the main road and red on the access road) the initial state; since the sequence is cyclic this decision seems a little arbitrary, so other options (plus what state the FSM restarts in after an emergency stop) might also seem reasonable.

Accepting the above, however, we next follow standard practice by translating the description into an implementation. Since $2^3 = 8 > 7$ we can represent the current and next states via 3-bit integers $Q = \langle Q_0, Q_1, Q_2 \rangle$ and

$Q' = \langle Q'_0, Q'_1, Q'_2 \rangle$ where

$$\begin{array}{lclcl}
S_0 & \mapsto & \langle 0,0,0 \rangle & \equiv & 000_{(2)} \\
S_1 & \mapsto & \langle 1,0,0 \rangle & \equiv & 001_{(2)} \\
S_2 & \mapsto & \langle 0,1,0 \rangle & \equiv & 010_{(2)} \\
S_3 & \mapsto & \langle 1,1,0 \rangle & \equiv & 011_{(2)} \\
S_4 & \mapsto & \langle 0,0,1 \rangle & \equiv & 100_{(2)} \\
S_5 & \mapsto & \langle 1,0,1 \rangle & \equiv & 101_{(2)} \\
S_6 & \mapsto & \langle 0,1,1 \rangle & \equiv & 110_{(2)}
\end{array}$$

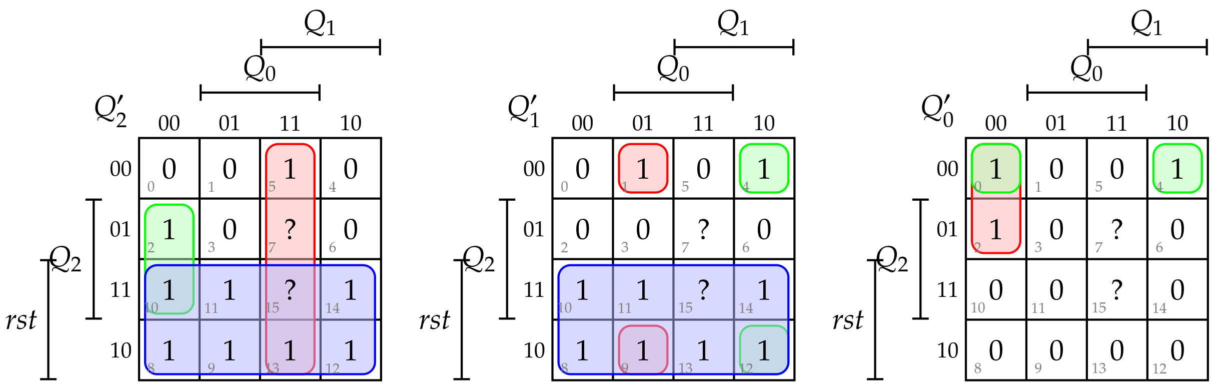
and we have one unused state (namely $S_7 \mapsto \langle 1,1,1 \rangle$). As a result, the input and output registers can each be implemented by using three 1-bit storage components, i.e., either D-type latches or flip-flops.

Now we have a concrete value for each abstract state label, we can expand the tabular description of the FSM into a (lengthy) truth table:

				δ			ω					
rst	Q_2	Q_1	Q_0	Q'_2	Q'_1	Q'_0	M_g	M_a	M_r	A_g	A_a	A_r
0	0	0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	1	0	0	1	0	0	0	1
0	0	1	0	0	1	1	0	0	1	0	1	0
0	0	1	1	1	0	0	0	0	1	1	0	0
0	1	0	0	1	0	1	0	0	1	0	1	0
0	1	0	1	0	0	0	0	1	0	0	0	1
0	1	1	0	0	0	0	0	0	1	0	0	1
0	1	1	1	?	?	?	?	?	?	?	?	?
1	0	0	0	1	1	0	1	0	0	0	0	1
1	0	0	1	1	1	0	0	1	0	0	0	1
1	0	1	0	1	1	0	0	0	1	0	1	0
1	0	1	1	1	1	0	0	0	1	1	0	0
1	1	0	0	1	1	0	0	0	1	0	1	0
1	1	0	1	1	1	0	0	1	0	0	0	1
1	1	1	0	1	1	0	0	0	1	0	0	1
1	1	1	1	?	?	?	?	?	?	?	?	?

Although this *looks* intimidating, we can use decomposition to consider the transition function and the output function *and* then each output signal within them separately:

- The transition function δ is just three Boolean expressions, one for each Q'_i, using rst, Q_2, Q_1 and Q_0 as input. The Karnaugh maps

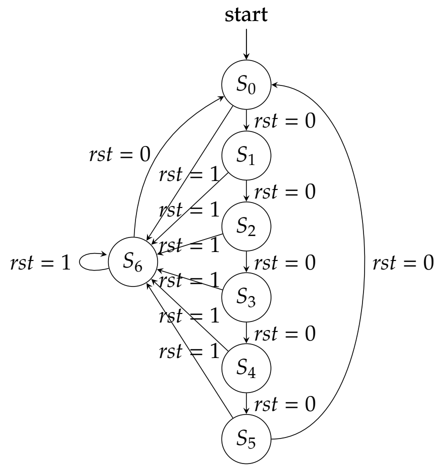


  can be used to produce said expressions:

$$\begin{array}{lcl}
Q'_2 & = & (\; rst \;) \vee \\
 & & (\; Q_2 \wedge \neg Q_1 \wedge \neg Q_0 \;) \vee \\
 & & (\; Q_1 \wedge Q_0 \;) \\
 & & \\
Q'_1 & = & (\; rst \;) \vee \\
 & & (\; \neg Q_2 \wedge \neg Q_1 \wedge Q_0 \;) \vee \\
 & & (\; \neg Q_2 \wedge Q_1 \wedge \neg Q_0 \;) \\
 & & \\
Q'_0 & = & (\; \neg rst \wedge \neg Q_1 \wedge \neg Q_0 \;) \vee \\
 & & (\; \neg rst \wedge \neg Q_2 \wedge \neg Q_0 \;)
\end{array}$$

	δ		ω					
Q	Q'		M_g	M_a	M_r	A_g	A_a	A_r
	$rst = 0$	$rst = 1$						
S_0	S_1	S_6	1	0	0	0	0	1
S_1	S_2	S_6	0	1	0	0	0	1
S_2	S_3	S_6	0	0	1	0	1	0
S_3	S_4	S_6	0	0	1	1	0	0
S_4	S_5	S_6	0	0	1	0	1	0
S_5	S_0	S_6	0	1	0	0	0	1
S_6	S_0	S_6	0	0	1	0	0	1

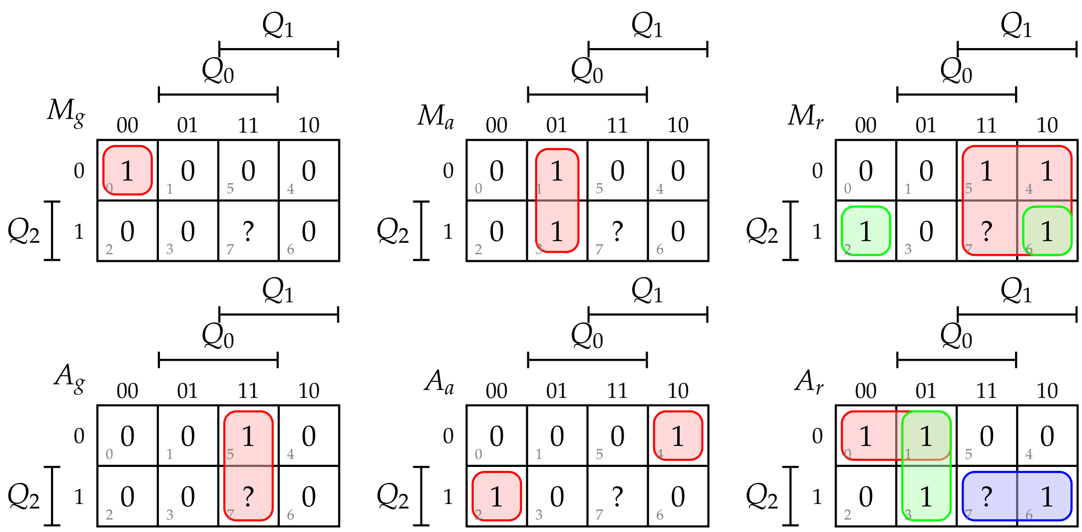
(a) *A tabular description.*



(b) *A diagrammatic description.*

Figure 2: *An FSM for the traffic light controller.*

- The output function ω is just six Boolean expressions, one for each M_i and A_j, using rst, Q_2, Q_1 and Q_0 as input. The Karnaugh maps



can be used to produce said expressions:

$$
\begin{array}{lcl}
M_g & = & (\neg Q_2 \wedge \neg Q_1 \wedge \neg Q_0) \\
M_a & = & (\neg Q_1 \wedge Q_0) \\
M_r & = & (Q_1) \vee \\
 & & (Q_2 \wedge \neg Q_0)
\end{array}
\qquad
\begin{array}{lcl}
A_g & = & (Q_1 \wedge Q_0) \\
A_a & = & (\neg Q_2 \wedge Q_1 \wedge \neg Q_0) \vee \\
 & & (Q_2 \wedge \neg Q_1 \wedge \neg Q_0) \\
A_r & = & (\neg Q_2 \wedge \neg Q_1) \vee \\
 & & (\neg Q_1 \wedge Q_0) \vee \\
 & & (Q_2 \wedge Q_1)
\end{array}
$$

The final step is to implement everything in LogisimEvo. Although this requires some effort, conceptually it means completing Figure 1 with 1) instantiations of the input and output registers, plus 2) instantiations of δ and ω, i.e., a translation of each expression into corresponding logic gates.

§2. R-class, or revision questions

▷ **S2[R].** There is a set of solutions available at

https://assets.phoo.org/COMS10015_2025_TB-4/csdsp/sheet/misc-revision_s.pdf

§3. A-class, or additional questions

▷ **S3[A].** There is no associated solution for this question, because it is somewhat open-ended with respect to 1) the goal or challenge presented, and/or 2) the assumptions and decisions you make, and therefore the design space of viable solutions.