This part of the assignment can be solved with a sequence of fairly simple building blocks. It requires you to be able to distinguish between two situations, in one case, the previous three clock-cycles (“thought-cycles”) have contained less than three events (“Thought-probes”). In the other case, the previous three clock cycles have contained three or more events.

In order to detect the number of events that have occurred in the previous three clock cycles, at any time, it is convenient to remember the number of events that have occurred during each of the last three clock cycles separately. Then, as each new clock cycle begins, we can throw away the information that pertains to four cycles ago.

A shift register is a suitable component for performing this temporary-memory function. If it has three stages, then it can remember information pertaining to three clock cycles.

But how many bits wide do each stage of the shift register have to be? As the number of events that can be detected in a clock cycle is in the range 0 to 3, each piece of information can be represented by a two bit number. Thus the shift register needs to be two-bits wide. This leads to the next part of the solution. How do the four individual lines output from the thought-probe detector get stored in a two-bit wide shift register? The outputs from the detector represent the values 0 to 3, inclusive, so feeding the lines from the thought-probe detector into an encoder will generate the appropriate input to the Shift Register.

Next, it is necessary to determine whether the condition for sounding the klaxon has occurred. That will happen if the sum of the numbers stored in the Shift Register exceeds two. So we need a circuit for detecting that condition. I’ve chosen a simple – and probably inefficient – approach. An adder produces the sum of the values in the Shift register, and a detector determines whether the sum exceeds two.

A number of students have designed detectors that operate directly on the output of the Shift Register, detecting bit combinations that correspond to the activation condition. That approach is fine, too.

Here’s a block diagram of my solution.

The individual component of this device should be drawn out, down to the level that has previously been described in lectures. That is, the architecture of the Shift register and the Adder should be shown, because they’re not the same as the devices we’ve looked at in the lecture course, but the structure of the individual flip flops and full adder circuits (if you’ve used full adders as building blocks to make the adder circuit – there are approaches that don’t use a full adder; in that case, the whole adder circuit should be drawn) that they contain do not need to be drawn out. The encoder and the >2 detector are new, so they should be drawn in full.

Other solutions that would work should attract full marks. However, marks would be deducted for a working solution that violates ordinary standards of good practice – a circuit with a grossly excessive number of gates, for example, or a circuit diagram that is so poorly drawn as to be ambiguous.
The diagram above shows my solution to the problem. Again, it is not necessarily the only solution. Any ASM (the assignment specifies that an ASM is required) that would work, subject to normal design
standards, like a reasonable number of states) would be accepted. You are expected to design using the
diagrammatic conventions of the ASM notation.

Here is a Transition Table for the ASM above, and a set of corresponding Boolean expressions for
the outputs. Of course, I don’t expect you to get exactly these Boolean expressions, because there are
many acceptable ASM diagrams that differ from mine. These should, however, give you an idea of the
sort of thing I expect.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Condition</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Open_door</th>
<th>Close_door</th>
<th>Go_to_horrible</th>
<th>Go_to_terrible</th>
<th>Reset_CFH</th>
<th>Reset_CFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Door_closed + ~Door_closed. ~cargo_for_horrible. ~cargo_for_terrible</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>~Door_closed. Cargo_for_horrible + ~Door_closed.. cargo_for_terrible</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>~Door_closed</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Door_closed</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>~At_horrible</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>At_horrible</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Door_closed + ~door_closed. ~cargo_for_horrible. ~cargo_for_terrible</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>~Door_closed. Cargo_for_horrible + ~Door_closed.. cargo_for_terrible</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>~Door_closed</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Door_closed</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>~At_terrible</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>At_terrible</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

\[ A_n = \neg A_p B_p C_p \cdot (\text{door\_closed} . \text{Cargo\_for\_horrible} + \neg \text{door\_closed} . \text{cargo\_for\_terrible}) + \neg A_p \neg B_p \neg C_p + \]
\[ B_n = \neg A_p - B_p C_p \cdot \text{~At\_terrible} \]
\[ C_n = \neg A_p - B_p C_p \cdot \text{(Door\_closed + ~door\_closed} . \text{~cargo\_for\_horrible. ~cargo\_for\_terrible}) + \]
\[ \neg A_p - B_p C_p \cdot \text{~At\_Terrible} \]

\[ \text{open\_door} = \neg A_p - B_p C_p \cdot \text{(Door\_closed + ~Door\_closed} . \text{~cargo\_for\_horrible. ~cargo\_for\_terrible}) + \]
\[ \neg A_p B_p C_p \cdot \text{Door\_closed + ~door\_closed} . \text{~cargo\_for\_horrible. ~cargo\_for\_terrible}) \]

\[ \text{close\_door} = \neg A_p - B_p C_p \cdot \text{~door\_closed} + \]
\[ A_p - B_p C_p \cdot \text{~door\_closed} \]

\[ \text{go\_to\_horrible} = \neg A_p B_p C_p \cdot \text{~at\_horrible} \]

\[ \text{go\_to\_terrible} = A_p - B_p C_p \cdot \text{~at\_terrible} \]
reset_cfh = \neg A_p B_p \neg C_p \text{at} \_\text{horrible}
reset_cft = A_p \neg B_p C_p \text{at} \_\text{terrible}

If you chose to implement the ASM using MUXes, then it would be acceptable to write out the Boolean expressions using a notation like this

$A_n =$ state$_3$ . (door\_closed . Cargo\_for\_horrible + \neg door\_closed . cargo\_for\_terrible) + state$_4$ + state$_5$ . \neg At\_terrible

$B_n =$ state$_1$ . door\_closed + state$_2$ + state$_3$ . (Door\_closed + \neg door\_closed . \neg cargo\_for\_horrible. \neg cargo\_for\_terrible)

$C_n =$ state$_0$ . (\neg Door\_closed. Cargo\_for\_horrible + \neg door\_closed . cargo\_for\_terrible) + state$_1$ . \neg door\_closed + state$_2$ . at\_horrible + state$_3$ . (Door\_closed + \neg door\_closed . \neg cargo\_for\_horrible. \neg cargo\_for\_terrible) + state$_5$ . \neg At\_Terrible

open\_door = state$_0$ . (Door\_closed + \neg Door\_closed. \neg cargo\_for\_horrible. \neg cargo\_for\_terrible) + state$_3$ . (Door\_closed + \neg door\_closed . \neg cargo\_for\_horrible. \neg cargo\_for\_terrible)

close\_door = state$_1$ . \neg door\_closed + state$_4$ . \neg door\_closed

go\_to\_horrible = state$_2$ . \neg at\_horrible

go\_to\_terrible = state$_5$ . \neg at\_terrible

reset\_cfh = state$_3$ . at\_horrible
reset\_cft = state$_5$ . at\_terrible