COMSM1302
Overview of Computer Architecture
Worksheet on FSMs

Kerstin Eder

September 23, 2021

Overview and workshop goals

This worksheet accompanies L07, which provides an introduction to *Finite State Machines*. Your first task in this workshop is to develop a 3-bit counter machine. Your second task is to design a finite state machine for a simple Vending Machine and to derive as well as optimise the control logic for this Vending Machine.

This workshop will give you several opportunities to practice the development of finite state machines and the optimisation of logic expressions. To make this lab more fun and rewarding, it is recommended that you work in small informal groups of two to three students, rather than individually. It would also be great to see the state-transition diagrams for your counter machines and your Vending Machine FSM.

We hope you have fun and that this lab helps you understand how to design FSMs.

1 Counter Machines

1.1 Familiarisation with a 2-bit counter

Figure 1 shows a state-transition diagram sketch for a 2-bit counter. The labels on the state-transitions list the input (either 0 or 1) followed by the two output bits that is the current value of the counter.

Inspect this 2-bit counter and figure out how it works. Note that this 2-bit counter "rolls over", i.e. when it reaches 11 it returns to 00 and starts counting from there. You may wish to develop a state-transition table for this 2-bit counter as a warm-up exercise.

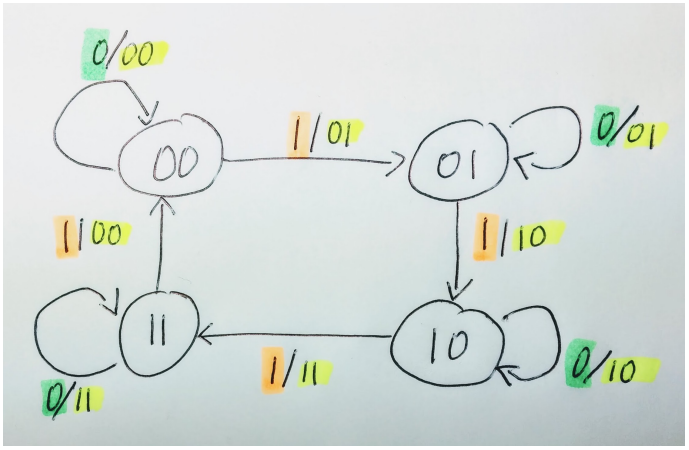


Figure 1: State-transition diagram for a 2-bit counter

Based on your understanding of the 2-bit counter, your task is to develop a 3-bit counter and to experiment with a variety of different versions of such a 3-bit counter.

1.2 Basic version of a 3-bit counter

Develop a basic version of the 3-bit counter for which you clearly specify the input and output and the states as well as the state encoding.

Note that there are different options for the output, e.g. the output is a single bit that tells you whether the input has been on, or the output is a 3-bit value that tells you how often the input was enabled, i.e. the value of the counter.

Develop the state-transition diagram for your 3-bit counter as well as the state-transition table.

Derive the output and next state logic for your 3-bit counter and optimise these.

Extra: It is possible to implement the logic for this 3-bit counter in Logisim. If you have time you may want to try this. However, please concentrate first on the tasks that are not marked "Extra".

1.3 3-bit counter version with reset

Extend your basic version with a second input that resets the counter to zero each time it is enabled, no matter whether or not the original COUNTER input is enabled.

Again, develop the state-transition diagram for your 3-bit reset counter as well as the state-transition table.

Derive the output and next state logic for your 3-bit reset counter and optimise these.

Congratulate yourself for completing this task.

1.4 Optional Extra: Extended version

Develop a new version of your counter that counts up in multiples of two.

Well done for doing extra!

1.5 Optional Extra: Forwards and backwards counter

Develop a version of your counter that counts forwards and backwards by one.

Well done for doing extra!

2 Vending Machine

Your task is to develop a finite state machine for a simple vending machine and to derive from this the expressions for the control logic of the next state function and the output.

The vending machine sells chocolate bars for 15 pence. It takes 5 pence and 10 pence coins, but only one at a time. The vending machine can give 5 pence change. Figure 2 shows the block diagram of this vending machine including inputs and outputs.

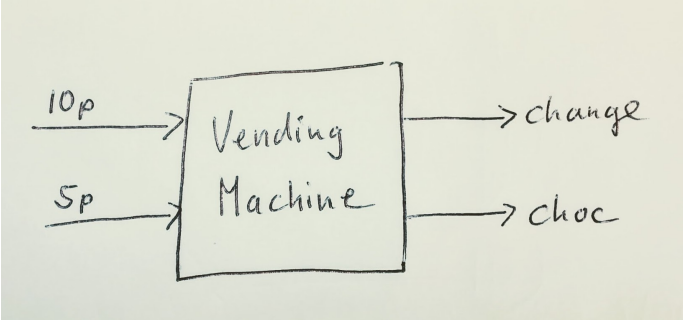


Figure 2: Block diagram of vending machine

Your finite state machine design for this vending machine should contain:

- the state-transition diagram and
- the state-transition table.

Based on these you should be able to develop the optimised expressions for the next state logic and the output logic.

2.1 State-transition diagram for Vending Machine

Develop the state-transition diagram for your vending machine.

First, consider carefully what the inputs and outputs are. Then consider how many states you need and how these could be encoded. To keep things simple you may want to opt for a total of three or four states.

Discuss the options in your group, including advantages and disadvantages.

Once you have decided on the states you intend to use, add the state transitions in a systematic way. For each state, consider all possible inputs.

Assess your state-transition diagram. Make sure you have encoded each state, i.e. assigned each state a unique bit pattern. There should be one transition for each possible input for each state - check this.

Show your state-transition diagram to your TA.

2.2 State-transition table for the Vending Machine

Based on your state-transition diagram, develop the state-transition table for your vending machine. The table should have columns for the current state and the inputs, followed by columns for the next state and the outputs. You may want to draw it out on a larger sheet of paper or use a spreadsheet if you prefer. Table 1 gives you a potential starting point. In this table, states have been encoded using two bits, S_1 and S_0 for the current state and S_1' and S_0' for the next state.

Capture your state-transition diagram in the state-transition table in a systematic way so that you can easily identify which transition in the diagram corresponds to which line in your table.

Current State		Inputs		Next State		Outputs		
S_1	S_0	10p	5p	S_1'	S_0'	Choc	Change	Notes
0	0	0	0	0	0	0	0	initially empty
0	0	0	1	0	1	0	0	5p inserted into initial state
0	0	1	0	1	0	0	0	10p inserted into initial state
...								

Table 1: FSM transition table header. Use as many state bits as you feel is appropriate.

2.3 Control logic for the Vending Machine

In order to obtain optimised logic expression for each bit in the next state and the output, you will need to transfer the relevant columns from your state-transition table into Karnaugh-maps. You will need one K-map for each bit in the next state and the output logic.

Tips

The intention of this workshop is not to find *the* answer, but to encourage you to explore the area of FSMs. Modern systems, from PCs to fitness trackers (and everything in-between and beyond), can be considered a composition of several FSMs to form a larger, more sophisticated system. In other words, FSMs are important. FSMs are everywhere, in an abstract sense, at least.

FSMs are not just important in hardware design. In general, the structure of an FSM can be used as a starting point also for software development. In fact, the application of formal methods is much simpler for code that implements an FSM, than for unstructured code.

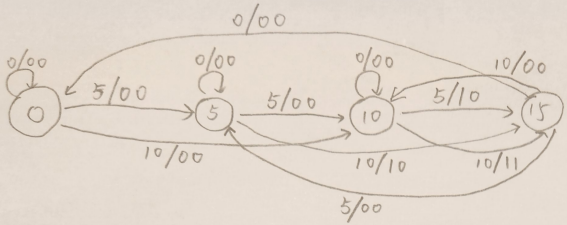
Feel free to ask questions and share thoughts with fellow students, TAs and lecturers.

Questions

If you have any questions, please ask. Make use of the TAs, but do not expect them to give you complete solutions, they are there to guide you, not do the work for you. Please be patient. You may need several attempts to make progress.

When requesting help, TAs will expect you to show them what you have done so far (no matter how sketchy) and they will ask you to clearly explain your reasoning. This makes it easier for the TAs to help you. For this reason, priority will be given to students who can show and explain how far they have got.

Well done for completing the workshop on FSMs.



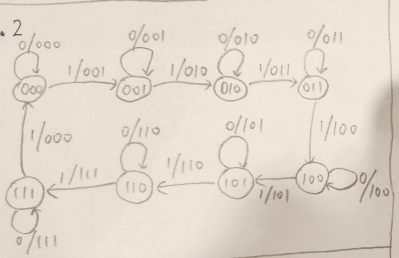
S_1	S_0	I_1 (10¢)	I_0 (5¢)	S_1'	S_0'	O_1 (choc)	O_0 (change)
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	1	1	0
1	0	0	0	1	0	0	0
1	0	0	1	1	1	1	0
1	0	1	0	1	1	1	1
1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	0
1	1	1	0	1	0	0	0

$I_1 I_0$ \ $S_1 S_0$	S_1' 00	01	11	10
00	0	0	0	1
01	0	1	0	1
11	X	X	X	X
10	0	1	1	1

1.1

S_1	S_0	I	S_1'	S_0'	O_1	O_0
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	1	0	1	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	0	0	0	0

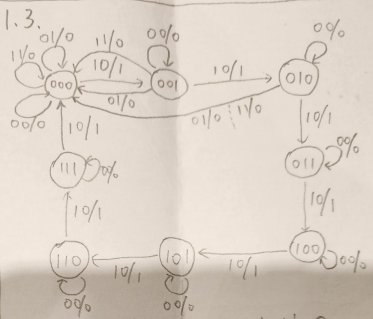
1.2



1.3

S_2	S_1	S_0	I	S_2'	S_1'	S_0'	O_2	O_1	O_0
1	1	1	0	1	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0

1.3.



S_2 S_1 S_0 I_1 I_0 S_2' S_1' S_0' O

1.3

S_2	S_1	S_0	I	S_2'	S_1'	S_0'	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0
0	0	1	1	0	1	1	0	1	1
0	1	0	0	0	1	0	0	1	0
0	1	0	1	0	1	1	0	1	1
0	1	1	0	1	0	0	1	0	0
0	1	1	1	1	0	0	1	0	0
1	0	0	0	1	0	1	1	0	1
1	0	0	1	1	0	1	1	0	1
1	0	1	0	1	1	0	1	1	0
1	0	1	1	1	1	0	1	1	0
1	1	0	0	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1