CSCB58 Lab 3
Part A: Finite State Machines

Introduction

In this part, you will design the control system for a vehicle gate that allows cars into a parking lot.

Desired Behaviour

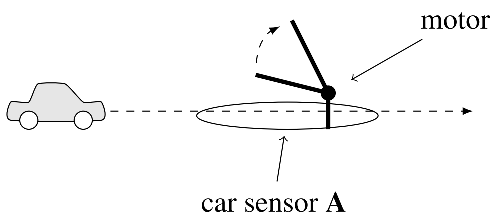


Figure 1: Gate diagram showing sensor **A** and motor controlled by **L** and **R** (input signal **B** not shown here).

The system is shown in Figure 1 and should behave as follows:

- Normally (initially) the gate will be held in the low position to block passage.
- The control system receives a signal **A**, which is 1 if a car is at, near, or crossing the gate. **A** is 0 if there is no car present. It also receives a signal **B** which is explained below.
- When a car is at the gate, the control system should raise it so the car can go through.
- Ass soon as the car can no longer be picked by the sensor (likely because it has passed the gate), the control system must lower the gate.
- The control system has two outputs which control the gate motor. To raise the gate, you should set the output signal **R** to 1 which will tell the engine to raise it. To lower the gate, you should set the output signal **L** to 1.
- It takes a long time to raise and lower the gate, a lot more than one clock cycle. While the gate is being moved, the relevant signal **R** (or **L**) must be held at 1 so the gate does not stop in the middle.
- The control system must stop running the motor once the gate has reached the correct position (high or low). In other words, **R** (or **L**) should be set to 0 once the gate has stopped moving. To help with that, an input signal **B** will be 1 whenever the gate has reached (or is currently in) one of the two final positions (high or low), and 0 if the gate still moving and has not reached the final position.
- Maintain safety: at all times the control system must never lower the gate while a car is passing through, to avoid causing damage to cars or people. If a car is passing though the gate while it is being lowered, raise the gate immediately.

Try to build a robust controller! It should work well and be safe. The specs tell you what should be done, but they may not cover all possible cases.

However, for this lab you do not need to consider the issues of race conditions / unsafe state transitions / hazards in your flip-flop assignments.

Guidance

We are going to separate the combinatorial logic to two sub-circuits: "statelogic" which implements the state machine transitions but not the outputs, and "outputlogic" which implements the logic for the two outputs.

Part I: State Logic

1. Draw the state transition diagram for the gate control system. Be as complete as possible and make sure to draw all possible transitions.
    - A correct solution must use the minimum number of states that implement the desired behaviour correctly. Remember we are ignoring issues of unsafe transitions.
    - You need not specify outputs at this stage.
    - Make sure to specify the initial state.
    - Hint: for some transitions, you may may not care about some of the inputs. For example, suppose for some transition you only care that B is 1, and you don't care whether A is 0 or 1. To save space, you can mark such transitions in the diagram using "A = X, B=1" or even "AB = X1", since you don't care about A for that transition.
    - For drawing, you can use PowerPoint or Google Docs or any drawing software you like, but make sure the diagram is clear and that you submit a **PDF** file.

   [TASK] **Submit the state transition diagram to Quercus as a PDF file**.

2. Write down the state table, including inputs, current state, and next state (but not outputs). At this point use state names, not flip-flop assignments.

   [TASK] **Submit the state table**.

3. Think: What is the minimal number of flip-flops you need for this state machine? Remember that we are ignoring unsafe transitions and race conditions here.

4. Using the minimal number of flip-flops, assign flip-flop values for each state. List the assignments: for each state write the bits for that state (there is no need to submit the updated transition table – we just want the mapping).

   [TASK] **Submit the flip-flop assignments**.

5. Write down the full state table (i.e., truth table) for your circuit based on the state transition diagram and flip-flop value assignment. Use K-maps to derive the Boolean expressions for next states. Name the input pins for states as **F3**, **F2**, **F1** and so on (depending on how many flip-flops you need), and name the output pins for states **newF3**, **newF2**, **newF1** and so on.

   [TASK] **Submit the truth table. (no need to submit K-maps and the Boolean expressions)**.

6. Create a sub-circuit in Logisim called "statelogic" in Logisim and implement the combinatorial logic. Make sure to include the needed input and output pins for reading/writing to the flipflops.

Part II: Output Logic

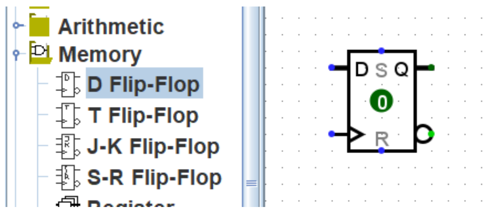
1. **Think:** Is this a Mealy or Moore machine? In other words, does the outputs of the state machine depend on the state only or also on the current input?
2. Write down the full output truth table for your circuit based on the state transition diagram and flip-flop value assignment. Use K-maps to derive the Boolean expressions for the control system outputs.

   [TASK] Submit the truth table for the output logic (no need to submit K-maps and expressions).
3. Create a sub-circuit in Logisim called "output logic" in Logisim and implement the combinatorial logic. Make sure to include the needed input and output pins.

Part III: Bringing it together

Implement the state machine in the main circuit, including all logic, state, and buttons for clock and reset. The main part of the circuit implement the state machine: flip-flop for state, buttons/switches for inputs, LEDs for outputs, and of course you will use the combinatorial logic you implemented as sub-circuits. For your state and output logic sub-circuits, you are allowed to use all the usual built-in gates, as many as you need.

1. Place one instance of your "statelogic" circuit and one instance of your "outputlogic".
2. Place D flip-flops for the state. Use the built-in Logisim flip-flop shown below (you will find this component in the explorer pane under **Memory** → **D Flip-Flop**).

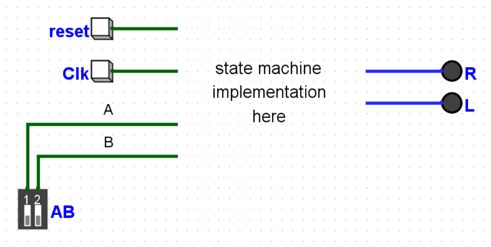


3. Connect the flip-flops to the combinatorial logic circuits.
4. Place a dip switch (**Input/Output** → **Dip switch**) with 2 switches: one for A and one for B, and connect them to the right places in the combinatorial sub-circuits.
5. Place one button (**Input/Output** → **Button**) called "Clk". This will simulate a single clock pulse every time you press and release it (try it!). Connect it to the clock input of the flip flops.
6. A reset input: To make the circuit easier to test, we will add an input that resets the FSM to the initial state anytime you want.

   Place another button called "reset" for this. If you want a flip-flop to be reset to 0 when "reset" is pressed, connect the button to the R input of the flip-flop (bottom of the flipflop). If you want a flip-flop to be set to 1 when "reset" is pressed, connect the button to the S input at the top of the flipflop. Connect the button only to one of them (S or R), and leave the other unconnected.

   Background: The Logisim D flip-flop feature *asynchronous active-high set/clear*: you can set their state to either 0 and 1 regardless of the clock (hence, asynchronous) and this reset is activated whenever the reset input is 1 (hence active-high). You could implement this kind of logic yourself in our own D flip-flop circuits by adding some AND or OR gates that bypass the "gate" part of the circuit.

7. The inputs and outputs in the main circuit should look something like the following



8. Test your circuit and fix issues! You will need to demonstrate that it works correctly to your TA.

   [TASK] Save the whole design and submit the .circ file to Quercus

Summary of what to submit

You need to submit a zip file that includes the following:

1. A PDF (or several PDFs) with:
   - Your name and student number.
   - state transition diagrams (part I)
   - state table (part I)
   - flipflop assignments (part I)
   - Truth table for the state logic (part I)
   - Truth table for the output logic (part II)
2. A single Logisim .circ file that all the circuits: "statelogic", "outputlogic", and main.

Part B: Datapath

Your task this will will be to build a control unit that computes $AX^2 + BX + C$ using the 8-bit datapath shown below. For this lab we have already built the datapath for you! All you need to do is the design and implement the control unit.

Make sure to download the file `lab3_base.circ` from Quercus. Use it file as the base for your solution.

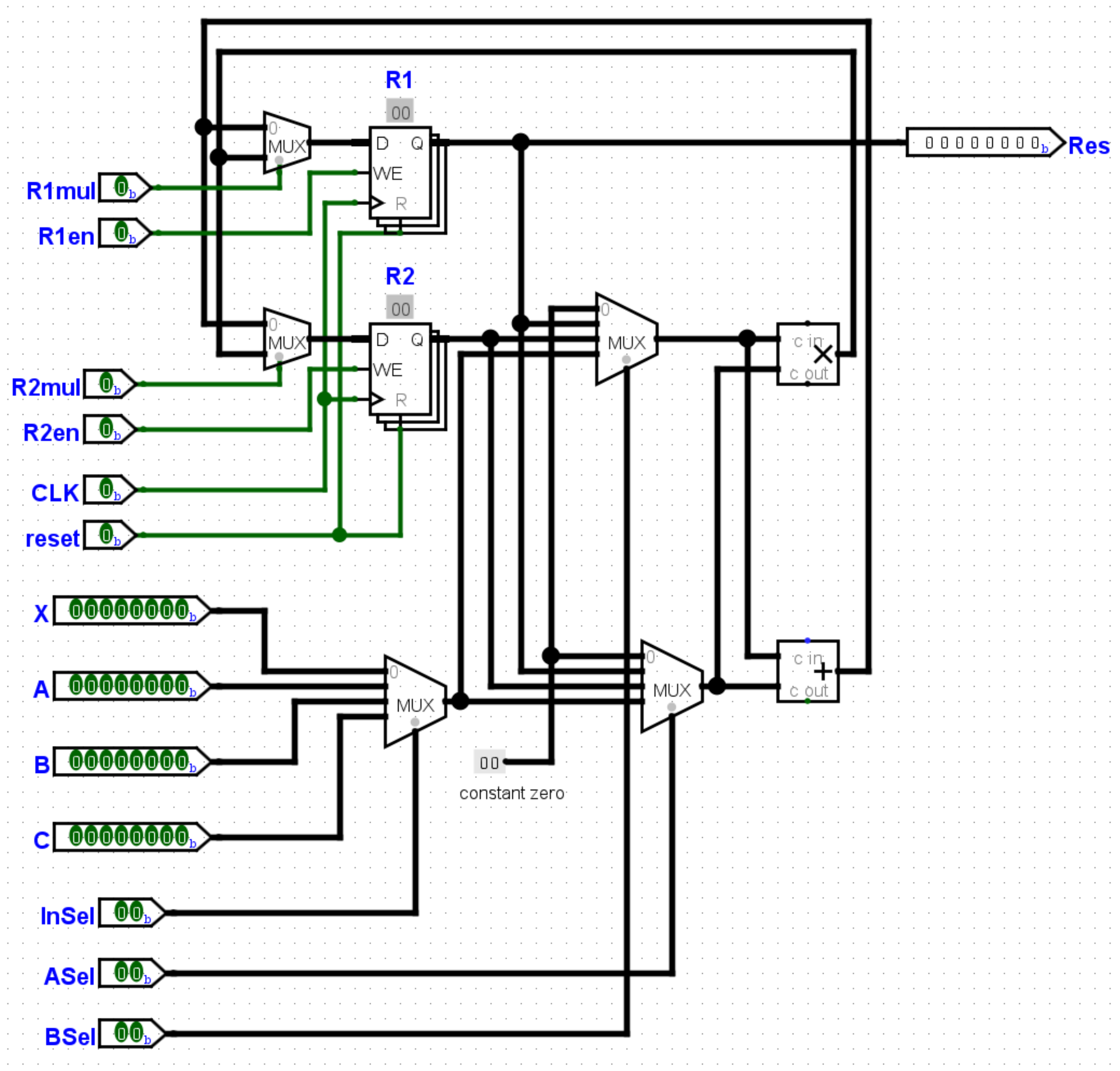


Figure 2: The datapath used for this lab

1 Introduction

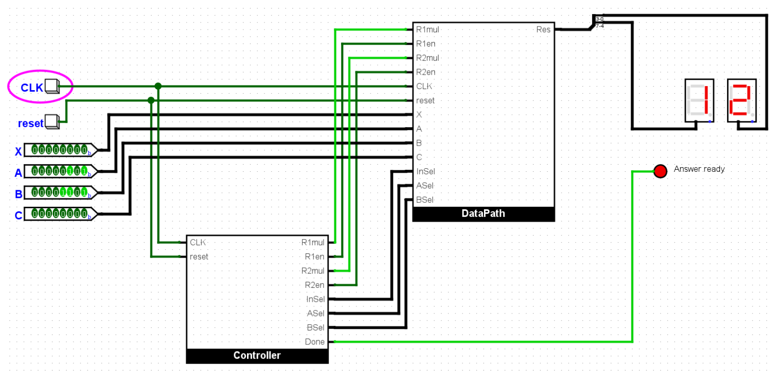
First, open `lab3_base.circ` in Logisim.

You will see that the datapath is implemented for you in the "DataPath" subcircuit. We have implemented a sample control unit that uses this data path to compute $A + B$ in 3 cycles.

Try it:

1. Use the poke tool to set A and B to some numbers, for example 5 (00000101) and 13 (00001101). Make sure that the result will not overflow 8-bits.
2. Use the poke tool to click the "reset" button to reset the state of the circuit.
3. Use the poke tool to click "CLK" 3 times, thereby advancing the clock 3 times.

You should see that the "Answer ready" light has turned on, and the result (shown in base 16) is 12, which is 18 in decimal. This should look something like the following



The `lab3_base.circ` file includes the following circuits:

- "main" – wires up the datapath, controller, buttons, and input/output. Looks like the image above.
- "DataPath" – the datapath as shown in the first page.
- "ControllerLogic" – the combinatorial logic for state and output logic of the FSM. Currently, it implements $A + B$ but you will modify it.
- "Controller" – the control unit (FSM) that sends the control signal. It uses ControllerLogic and just adds the necessary flipflops. You may need to modify it.

2 Design

Your first task is to design a control unit that computes Res $= AX^2 + BX + C$ using this datapath. This datapath has two registers (R1 and R2), an adder (the square marked as "+" in the right side of the datapath), a multiplier (marked as "$\times$"), and bunch of muxes that control what goes into each of these components.

Requirements:

- At the end of the computation, your FSM should make the datapath stop, hold its state, and output Done=1 to tell the user the result is ready. The FSM should stay in the holding state with the output available in Res and with Done=1 until a "Reset" arrives. If not yet done, output Done=0.

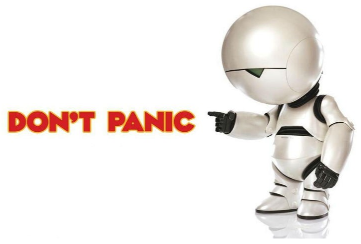
- You can assume the input is correct and will be held constant throughout the entire computation.

- The user is expected to set up the inputs then press Reset at the beginning of every computation. Make sure to do the right thing using the “Reset” signal.
- The best design will finish the computation using the minimum number of steps, since the more steps you have in the control unit (i.e., states), the more flips flops you’ll need. A design that uses more flipflops than necessary may not receive full marks. Note that you will need additional flipflops than what is included in the example FSM.

Guidance:

1. Don’t panic. You can do this.
2. Look at the datapath. Figure out what you kind of basic steps you can actually do with it, given the wiring, muxes, and components available.
3. Split $AX^2 + BX + C$ operation into a sequence of basic steps that you can do with the datapath. Make sure these are actually possible with the datapath! Sometimes we wish we could do something, but the wiring in the datapath does not actually allow it.
4. Convert the sequence of steps to a table that shows for each step: (a) the step number (start with zero); (b) what this step does; (c) what the values in R1 and R2 be after this step; (d) what are the control signals you need to achieve that; and (e) what is the number of the next step.

   [TASK] Submit this table as a PDF to Quercus.
5. Note that (a), (d) and (e) together make up the truth table for your FSM! You may need to add a Done signal too.
6. Given that truth table, you know know how many flipflops you’ll need.



As an example, here is the table for the example FSM that computes $A + B$ (this is not necessarily the fastest way to compute $A + B$ using this datapath). The column “operation” is not part of the truth table, it is just here for our benefit.

Step	Operation	In R1	In R2	R1mul	R1en	R2mul	R2en	InSel	ASel	BSel	Done	NextStep
0	$R1 = A$	A	?	0	1	x	0	01	11	00	0	1
1	$R2 = B$	A	B	x	0	0	1	10	11	00	0	2
2	$R1 = R1 + R2$	$A + B$	B	0	1	x	0	10	01	10	0	3
3	Hold	$A + B$	B	x	0	x	0	xx	xx	xx	1	3

Note the use of don’t-cares (x) when we don’t care about a particular MUX signal. On common case this can happen is that if the register being “fed” by this MUX is not loading (en=0). For example, consider InSel in step 2 in the table above. It is currently 10, but we **could** have used xx instead, since for step 2 the output of the mux InSel controls will be ignored by the components downstream.

3 Implementation

Modify "ControllerLogic" and optionally "Controller" to implement your FSM:

1. You can change "ControllerLogic" (including pins) as much as you want.
2. You can change, add, or remove components in the "Controller" subcircuit. However but you may not change (or remove, or add) any of the input/output pins. They are connected in the main circuit, which we already wired for you.
3. Do not change the DataPath or main circuits. Make sure you do all your changes to the "Controller" and "ControllerLogic" sub-circuit. If you want more practice in using K-maps, you can also do it manually.
4. Test your circuit. Note it is very easy to cause multiplication to overflow (especially with AX^2) so make sure to use appropriately small inputs so that $AX^2 + BX + C$ does not overflow.

[TASK] Save your circuit as `lab3.circ` and submit to Quercus.

Summary of tasks

1. Design the FSM and submit the table of steps and output signals.
2. Implement and test the circuit in Logisim. Go back to previous steps if you've made a mistake.
3. Submit zip file to Quercus with the table of steps and the circuit.
4. Demonstrate the use of your circuit to the TA and explain why it is working correctly.

Evaluation

As always, marks are based not only on submitted work but also on oral examination by TA. You need to be able to make reasonable explanations about any details to the TA.

Solution	Submitting everything on time	1 mark
	Part A:	
	Design	2 marks
	Implementation	2 marks
	Part B:	
	Design	3 marks
	Implementation	1 marks
Understanding	TA oral score	1 to 4

Final lab marks (up to 8) are determined by multiplying your solution subtotal by the oral score (1–4) and dividing by 4:

$$\text{total} = \text{solution marks} \times \frac{\text{oral score}}{4}$$