Lab 8: De Morgan Equivalent Gates

Perhaps you have noticed that while a NAND gate is defined as having an output of 0 when all its inputs are 1, it can also be described as having an output of 1 when any of its input are 0. A NAND gate can either be described as an AND gate with an inverted output or an OR gate with inverted inputs. In 1847, Augustus De Morgan figured this out and very elegantly stated, "The negation of a conjunction is the disjunction of the negations. The negation of a disjunction is the conjunction of the negations."

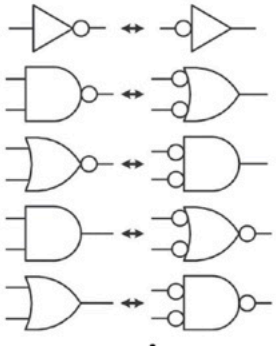
Here it is described in its algebraic form.

$\sim(A_0 \,\&\, A_1) \leftrightarrow \sim A_0 \mid \sim A_1$
$\sim(A_0 \mid A_1) \leftrightarrow \sim A_0 \,\&\, \sim A_1$

Here are the equivalents for AND and OR functions.

$A_0 \,\&\, A_1 \leftrightarrow \sim(\sim A_0 \mid \sim A_1)$
$A_0 \mid A_1 \leftrightarrow \sim(\sim A_0 \,\&\, \sim A_1)$

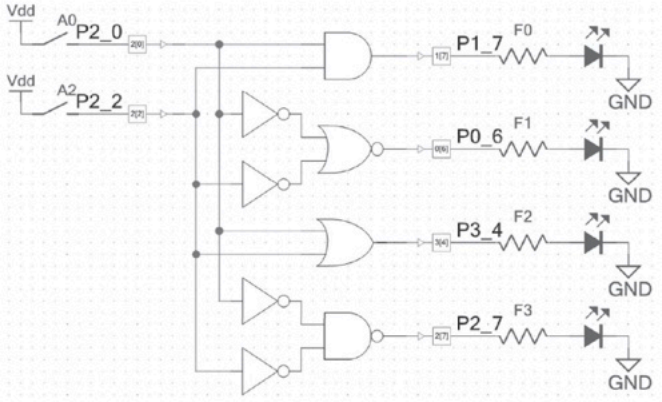
Here are the De Morgan equivalents in logic gate format.



Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the MyStuff tab of the component catalog, and drag in A_0, A_2, F_0, F_1, F_2, and F_3.
- Go to the Cypress tab, and get a NAND gate, a NOR gate, an AND gate, and four NOT gates.
- Make sure to assign the correct pins to your inputs and outputs.

Connect them together, as shown below.



- Build this project, and download it to your test board.

If you have done this project correctly:

- F_0 will be only on when both the inputs are on.
- Being a De Morgan equivalent, F_1 should match F_0's behavior.
- F_2 will be one when any of the two inputs are on.
- Being a De Morgan equivalent, F_3 should match F_2's behavior.

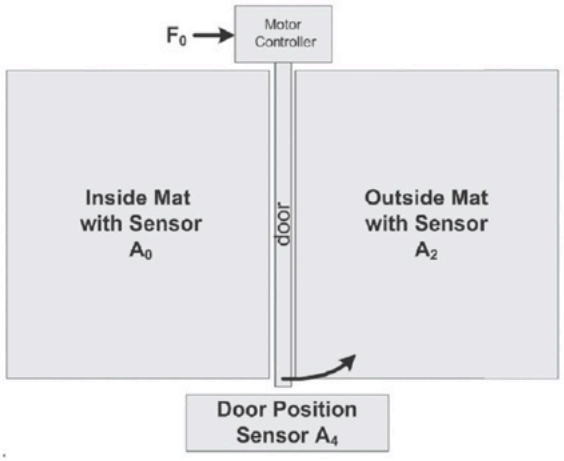
Save this project as a macro. Name it "Lab8," and store it in the MyStuff tab under the MyLabs section.

You have completed the eighth lab.

Lab 9: Combinational (or Combinatorial) Logic

Combination (or combinatorial) logic means exactly what it sounds like. It is a combination of the previously discussed logic gates used to design a specific control application. It will consist of inputs, outputs, and the logic to implement the desired function.

For this example, the application is a controller for an automatic door opener, as shown below.



This particular door controller has three inputs and one output. It is required to allow someone to safely exit from the inside to the outside while prohibiting someone outside from going inside. Sensors A_0 and A_2 provide an output of 0 when the mat is empty and a 1 when occupied. The door position sensor (A_4) provides an output of 0 when the door is closed and 1 when it is not. F_0 outputs a signal to the motor controller, so 0 commands the door to close, and 1 commands the door to open.

There are eight different binary combinations of the three inputs, as shown in the table below.

Inputs			Output	Comments
A4	A2	A0	F0	
0	0	0	0	*Keep door closed when inside mat is empty, outside is empty, and door is closed.*
0	0	1	1	*Open door when inside mat is occupied, outside mat is empty, and door is closed.*
0	1	0	0	*Keep door closed when inside mat is empty, outside mat is occupied, and the door is closed.*
0	1	1	0	*Keep door closed when inside mat is occupied, outside mat is occupied, and the door is closed.*
1	0	0	0	*Close door when inside mat is empty, outside mat is empty, and door is open.*
1	0	1	1	*Keep door open when the inside mat is occupied, the outside mat is empty, and the door is open.*
1	1	0	1	*Keep door open when the inside mat is empty, the outside mat is occupied, and the door is open.*
1	1	1	1	*Keep door open when the inside mat is occupied, the outside mat is occupied, and the door is open.*

Of the eight possible combinations, there are four that cause an output of 1:

$(\sim A_4 \,\&\, \sim A_2 \,\&\, A_0)$
$(A_4 \,\&\, \sim A_2 \,\&\, A_0)$
$(A_4 \,\&\, A_2 \,\&\, \sim A_0)$
$(A_4 \,\&\, A_2 \,\&\, A_0)$

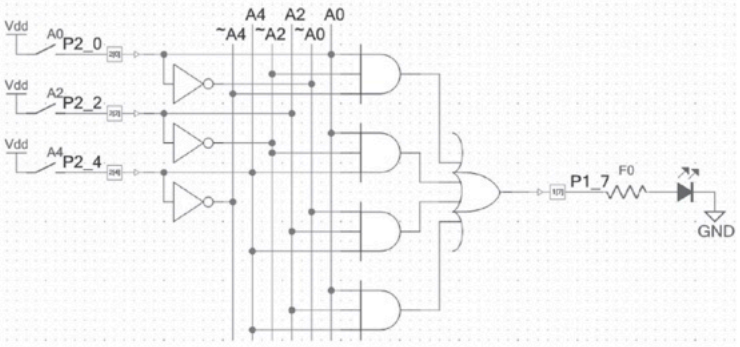
Any of these combinations will cause an output of 1, so the Boolean equation for this function is as follows:

$F_0 = (\sim A_4 \& \sim A_2 \& A_0) | (A_4 \& \sim A_2 \& A_0) | (A_4 \& A_2 \& \sim A_0,) | (A_4 \& A_2 \& A_0)$

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the MyStuff tab of the component catalog, and drag in A_0, A_2, A_4, and F_0.
- Go to the Cypress tab, and get any of the logic gates need to implement the design shown below.
- Make sure to assign the correct pins to your inputs and outputs.

Connect them together, as shown below. I have drawn this schematic in bus format because, with six different options for inputs, it is easy to make a connection mistake. I used the text icon to add the six comments to the schematic.



- Build this project, and download it to your test board.

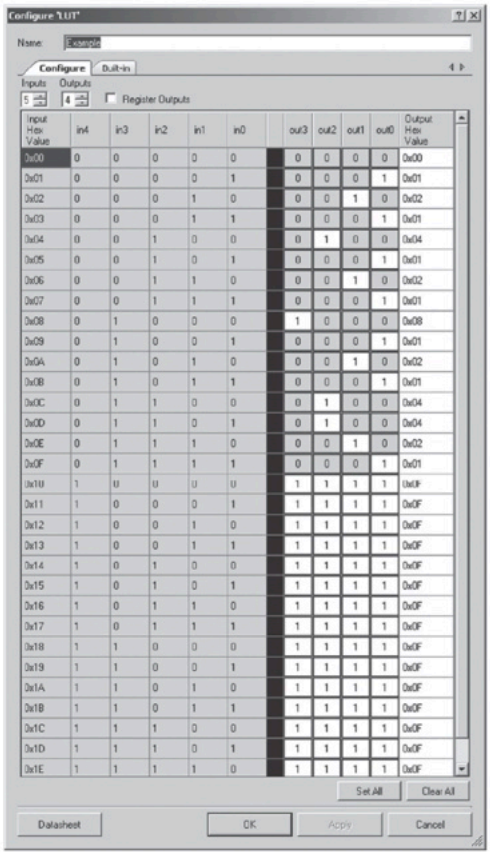
If you have done this project correctly, F_0 will follow the eight different combinations of the three inputs. Test to verify.

Save this project as a macro. Name it "Lab9," and store it in the MyStuff tab under the MyLabs section.

You have completed the ninth lab.

Lab 16: Logic Implementation, Even Easier, with LUTs

By now, you either find Karnaugh maps intriguing or tedious. Trust me when I say that, in ten years, you will only find them tedious. An easy way to implement combinational logic is to take your truth table and put it in an electronic lookup table (LUT). A LUT is a logic device that allows the logic term to be entered in a table form. Originally they were implemented with electrically programmable, read-only memories (EPROM). A 2708 (1 K by 8 EPROM) could be used to implement a LUT with ten inputs and eight outputs. They are considered the first step toward programmable logic. With PSoC, the LUTs are implemented with its programmable logic. Below is a LUT component configured to implement the five-input Karnaugh map example in the fourteenth lab.



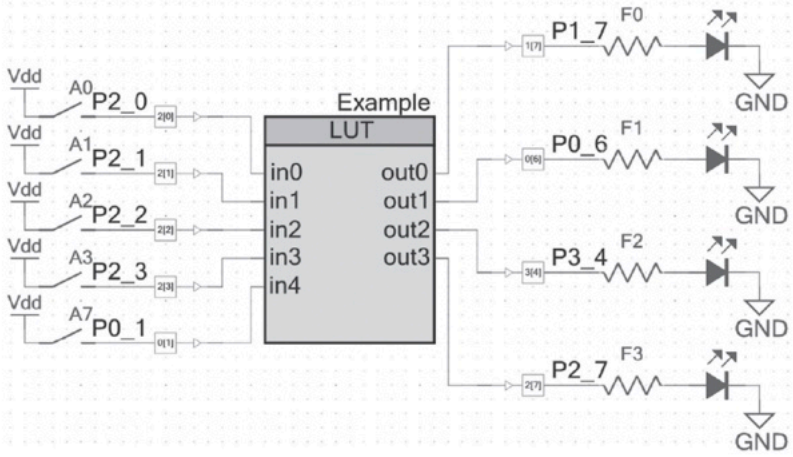
Set the number of inputs and outputs, and enter the right values into the output table.

If you are a hobbyist and you would rather forget the whole logic reduction and use LUTs exclusively, then knock yourself out. They are easy to understand and change. Because Creator will synthesize this design into programmable logic, there is no hardware overhead using LUTs.

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the component catalog, and drag in LUT and any other components shown below.
- Configure the LUT, as shown in the previous figure.

Connect them together, as shown below.



- Build this project, and download it to your test board.

If you have done this project correctly, the four outputs will follow the thirty-two different combinations of the five inputs. Test to verify.

Wasn't this lab easier to implement than the fourteenth lab? When I want to build combinatorial logic quickly, I use LUTs.

Save this project as a macro. Name it "Lab16," and store it in the MyStuff tab under the MyLabs section.

You have completed the sixteenth lab.

Lab 17: Digital Circuits with Memory, the Latch

All the examples so far have been combinatorial. That is, the designs have different combinations of only the inputs' present values.

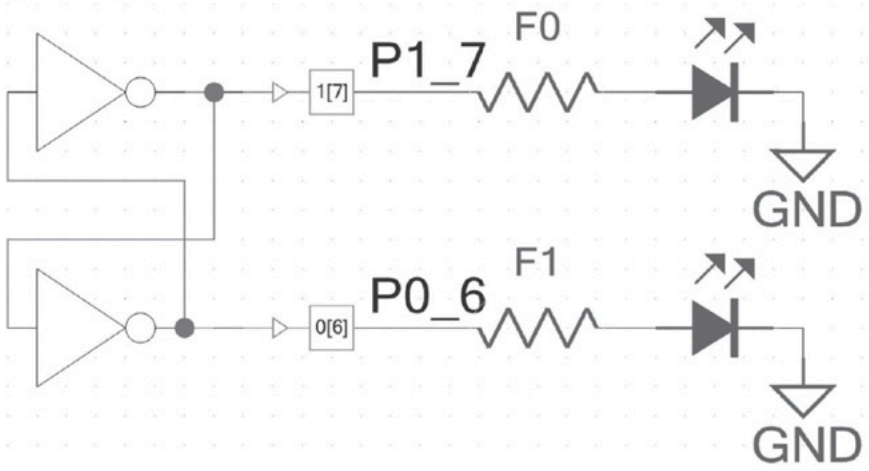
It has not been possible to design a circuit that has an output of some previous stimulus. This is a good time to introduce the concept of a latch. A latch is constructed by feeding back the output of some digital circuitry as an input that allows at least two stable states to be achieved. The simplest latch is constructed with two inverters and is shown below.

The output of each inverter drives the input of the other. One of the outputs is 1, which causes the output of the other to be 0, which causes the output of the original inverter to remain 1. This feedback loop is latched in a stable state and will permanently stay that way. If the original inverter's output were 0, then the feedback will keep it at that value. Unfortunately, while there are two different stable states, there is no way to select which one or to switch states. Still, it's a start.

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the component catalog, and drag in all the components shown below.
- Make sure to assign the correct pins to your inputs and outputs.

Connect them together, as shown below.



- Build this project, and download it to your test board.

Note that Creator will give you a warning that you may have built an unintentional latch. Ignore this warning. The latch you made is very intentional.

If you have done this project correctly, only one of the two LEDs will be on and will remain on. Leave it alone for a while. When you come back, you will see it is in the same state. Press and release the reset button on your PSoC board. The latch comes back to the same state. Test to verify.

Save this project as a macro. Name it “Lab17,” and store it in the MyStuff tab under the MyLabs section.

You have completed the seventeenth lab.

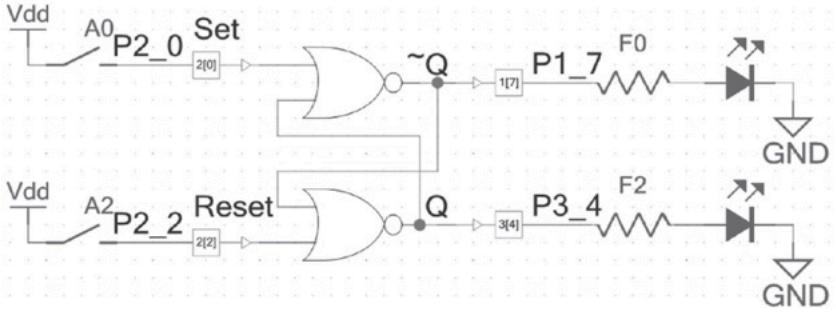
Lab 18: The Selectable SR Latch

Except for educational purposes, an uncontrollable latch is pretty much useless. Suppose you used NOR gates instead of inverters. When the extra inputs are 0, it functions just like the inverter latch in the previous lab. However, if you make the extra input on the top NOR gate 1, it will force the gate's output to be 0. This 0 output feeds back to the other NOR gate to make its output 1. This output feeds back to the top NOR gate to reinforce the 0 output, regardless of what the other input value is. The circuit is latched again. Take the input back 0, and the outputs remain latched in this same state. This is a symmetrical circuit; applying a 1 on the bottom input will cause the circuit to latch in the reserve state. Make both inputs 1, and both outputs are 0, but this is not a stable state. As soon as both inputs are set 0, the outputs will go to one 1 and the other 0. This is called a selectable set/reset (SR) latch.

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the component catalog, and drag in any components shown below.
- Make sure to assign the correct pins to your inputs and outputs.

Connect them together, as shown below.



- Build this project, and download it to your test board.
- If you have done this project correctly:
- Press A_0; F_0 will turn off, and F_2 will turn on.
- Release A_0, and the LEDs remain in the same state.
- Press A_2; F_2 will turn off, and F_0 will turn on.
- Release A_2, and the LEDs remain in the same state.
- Press both buttons, and neither LED is on.
- Simultaneously release both buttons, and only one LED will turn on.

Test to verify.

Save this project as a macro. Name it "Lab18," and store it in the MyStuff tab under the MyLabs section.

You have completed the eighteenth lab.

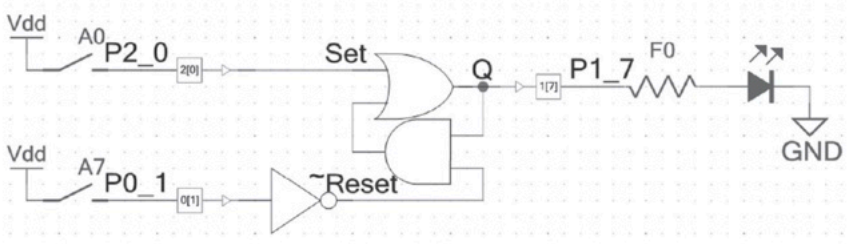
Lab 19: The Selectable S~R Latch

What is interesting about this latch is that it does not use inverting gates to construct it. It is a set, ~reset (S~R) latch, and it allows the output to be 1 when both inputs are 1 and 0 when both inputs are low. When the set is 0 and the ~reset is 1, it remains latched in its previous state. This little combination has many applications when combined with analog circuitry.

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the component catalog, and drag in any component you see below. (The AND gate can be flipped by right clicking on it, going to "shape," and then selecting "flip horizontal.")
- Make sure to assign the correct pins to your inputs and outputs.

Connect them together, as shown below.



- Build this project, and download it to your test board.

If you have done this project correctly:

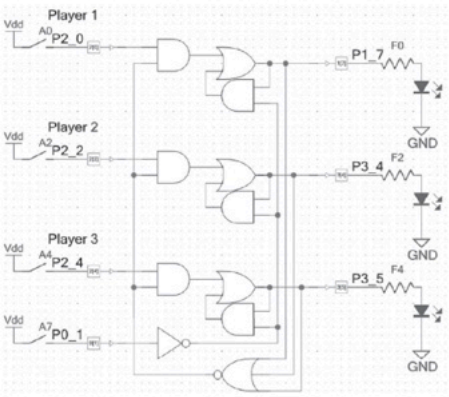
- Press A_0, and F_0 will turn on.
- Release A_0, and F_0 remains on.
- Press A_7, and F_0 turns off.
- Release A_7, and F_0 remains off.

Test to verify.

Save this project as a macro. Name it “Lab19,” and store it in the MyStuff tab under the MyLabs section.

You have completed the nineteenth lab.

Lab 20: I'll Take PSoC for $200, Alex



This is a fun little project I call "jeopardy buttons." There are four inputs, three players, and a reset. Each player has its own output that goes high if they are the first to press their button. Each channel has an S~R latch to grab the signal from a pressed button. All the latch outputs are "NOR-ed" together, so the first one set will lock out the inputs, with the AND gate placed in front of each latch, and allow no other outputs to be set. The reset will turn any of them off. If you replace the buttons with optical switches, you can use it for pinewood derby tracks.

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the component catalog, and drag in any component you need.
- Make sure to assign the correct pins to your inputs and outputs.
- Connect them all together.
- Build this project, and download it to your test board.

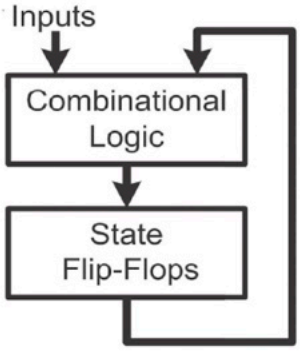
If you have done this project correctly, the reset will reset all the latches. The first of any of the three buttons pressed will set its latch and disable all the inputs. Test to verify.

Save this project as a macro. Name it “Lab20,” and store it in the MyStuff tab under the MyLabs section.

You have completed the twentieth lab.

Lab 25: The State Machine (Inputs and States)

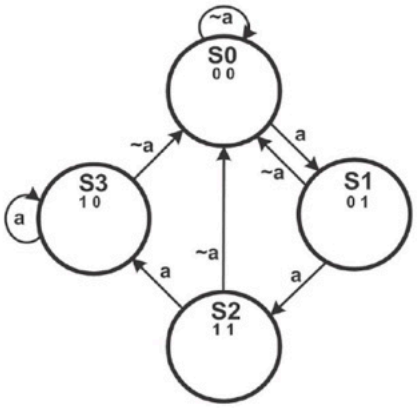
Digital circuitry is often used to control a process or implement an algorithm. These controllers are designed with sequential logic and finite state machines. A finite state machine (or state machine for short) is a computational model that defines all the necessary states to implement a process. A state machine consists of defined inputs, outputs, and a finite number of states. There are several different ways of handling the outputs, but the inputs and states are all handled the same way. A state machine can only be in one state at a time. This is called the "present state." It stays in that state until it receives a particular triggering event or condition. The state machine then transitions to the next desired state, and that becomes the present state. View the block diagram below.



Neglecting the output for now, a state machine consists of inputs, combinational logic, and flip-flops to store the state.

The combinational logic used the inputs and the present state value to determine the next state value. Although not shown in the diagram, the flip-flops require a common clock.

The tool used to design state machines is called a state diagram. A sample is shown below.



Each diagram will have a finite number of states. Each state will be assigned a value, but S0 is reserved for the initial state. Underneath each state number are the values assigned to the flip-flops for that particular state. These values are generally assigned to make the combinational logic as simple as possible. In this state diagram, only four states are defined, so it can be implemented with two flip-flops. (More can be used, as will be seen in later labs, but the minimum required in this case is two.)

This example looks at an input data stream "a" and only outputs when the present input and two previous inputs are one. This kind of circuit is used to "deglitch" a data stream.

The states have the following definitions:

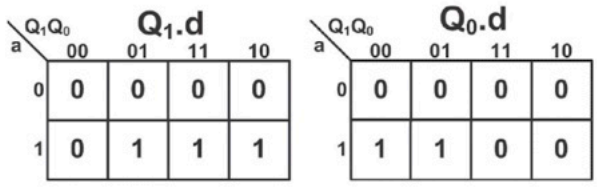
- S0—previous data was 0
- S1—previous two data values were 1and 0
- S2—previous three data values were 1, 1, and 0
- S3—at least the previous three data values are all 1

This information will later allow you to decide when the output should be 1. What is needed now is the logic to implement this design.

This state diagram has a single input and two flip-flops (Q_1, Q_0). Take the diagram, and use the information to fill out a truth table. It should look like this.

Present State			Next State	
Q_1	Q_0	a	Q_1.d	Q_0.d
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	0
1	1	1	1	0

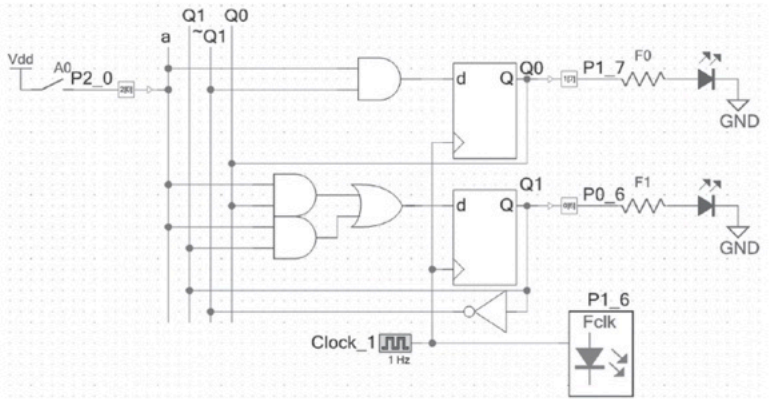
Using this table, construct a three-variable Karnaugh map for each of the two flip-flop's inputs. They should look like this.



The rest of the steps follow simple logic reduction and implementation.

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the component catalog, and drag in any components shown below.
- Make sure to assign the correct pins to your inputs and outputs.
- Connect them together, as shown below.



- Build this project, and download it to your test board.

If you have done this project correctly:

- The states (flip-flop outputs) will cycle as shown in the state diagram.
- The flip-flop outputs are synchronized to the raising edge of the clock.

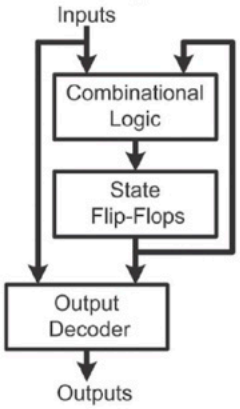
Test to verify.

Save this project as a macro. Name it "Lab25," and store it in the MyStuff tab under the MyLabs section.

You have completed the twenty-fifth lab.

Lab 26: The Class-A (Mealy) State Machine

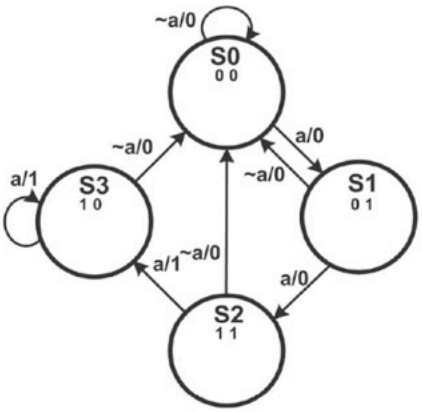
Of course state machines are going to require outputs, but there are several different schools of thought on how it is best done. The first method was named after George Mealy, who presented the concept in a 1955 paper, "A Method for Synthesizing Circuits." With this type of state machine, the outputs are a function of the inputs and the present state. View the block diagram below.



This state machine consists of the following:

- inputs
- combinational logic
- flip-flops to store the state
- logic to convert the present state and inputs to the required outputs

The state diagram will have the same states and transition arrows. The difference is that the output data is added to the trigger conditional value. See the state diagram below.



Note that each transition arrow now includes the value of the output. The output is 1 for S2 when "a" is 1 and for S3 when "a" is 1. The Karnaugh map for the output should look like this.

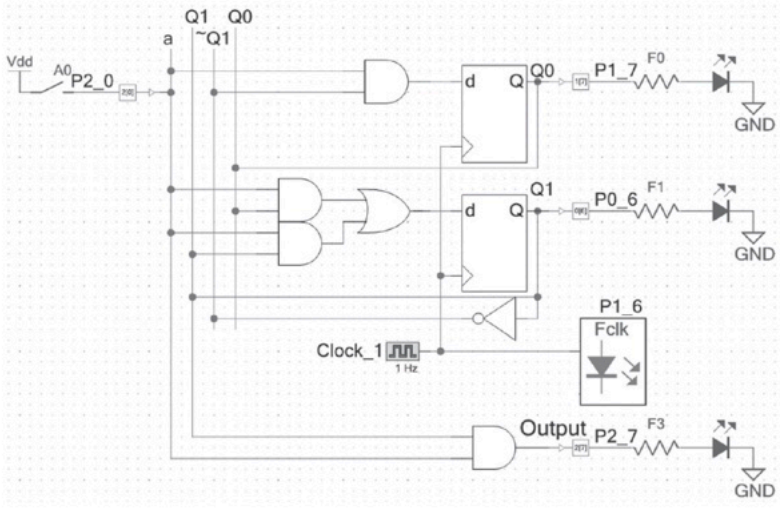
a \ Q_1Q_0	00	01	11	10
0	0	0	0	0
1	0	0	1	1

Output

The rest of the steps follow simple logic reduction and implementation.

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the component catalog, and drag in Lab25, F3, and a dual-input AND gate.
- Make sure to assign the correct pins to your inputs and outputs.
- Connect them together, as shown below.



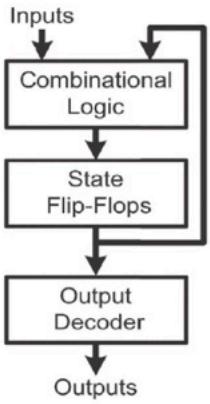
Lab 27: The Class-B (Moore) State Machine

A goal of Mealy's paper was to make state-machine design as automatic as the generation of combinational logic. I realized he considered state machines a super set of combinational logic when he wrote:

> A switching circuit is a circuit with a finite number of inputs, outputs, and (internal) states. Its present output combination and next state are determined uniquely by the present input combination and the present state. If it has one internal state, we call it a combinational circuit; otherwise, we call it a sequential circuit.

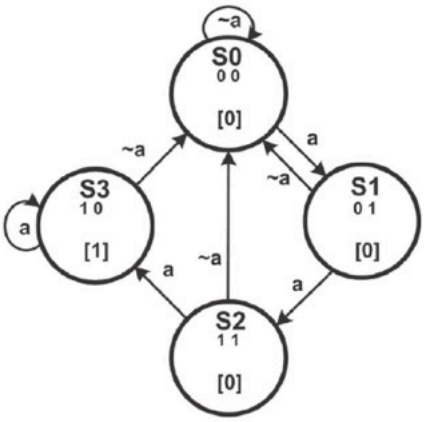
It is no wonder that he defined the outputs as a function of the inputs and the present state. If you consider the state machine to be different than combinational logic, you will realize that the outputs of a state machine are uniquely determined by the present state. The method of using only the sates to determine the output is named after Edward F. Moore, who presented the concept in his 1956 paper, "Gedanken-Experiments on Sequential Machines."

With this type of state machine, the outputs are a function only of the present state, as shown in the block diagram below.



This state machine consists of inputs, combinational logic, flip-flops to store the state, and logic to convert the present state to the required outputs.

The state diagram, shown below with its output is shown in each state.



Note that each state now has its output value listed. For this state machine, the output is 1 only for S3. This makes the Karnaugh map embarrassingly simple.

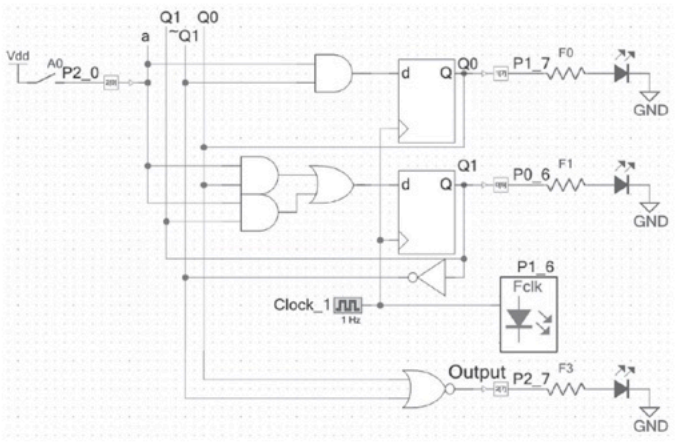
a \ Q_1Q_0	00	01	11	10
0	0	0	0	0
1	0	0	1	1

Output

From here it is just simple logic reduction and implementation. Instead of reducing it down to Q1 & ~Q0, I used the De Morgan equivalent of ~(~Q1 |Q0). An inverter was already used in the state-machine logic to invert Q1.

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the component catalog, and drag in Lab26 and a two-input NOR gate.
- Make sure to assign the correct pins to your inputs and outputs.
- Connect them together, as shown below.



- Build this project, and download it to your test board.

If you have done this project correctly:

- The states (flip-flop outputs) will cycle as shown in the state diagram.
- The output is on only when the state machine is in S3.
- While in state S3 and the clock is low, briefly release and press A0. The output should still remain on while the input is transient.

Test to verify.

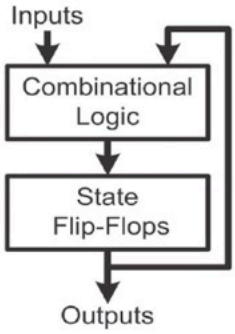
Save this project as a macro. Name it "Lab27," and store it in the MyStuff tab under the MyLabs section.

You have completed the twenty-seventh lab.

Lab 28: The Class-C (Moore) State Machine with Registered Output

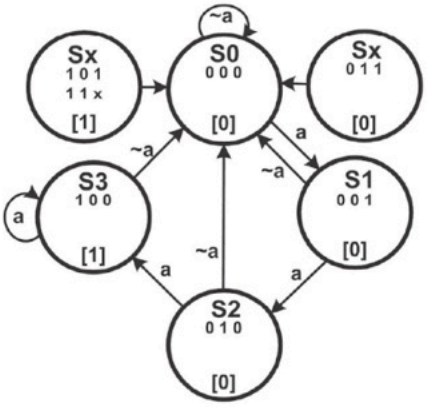
In the last lab, the output was determined exclusively by the present state of the state machine. However, this does not make the output synchronous to the clock. In the last example, the output was set to be 1 when in S3 (1, 0). The problem is that all the flip-flops do not flip exactly at the same moment in state-machine transitions. This transition between two states may momentarily be in an unintended state. In this case, when S2 (1, 1) goes to S0 (0, 0), it will briefly be in S1 (0, 1) or S3 (1, 0). If it is the latter, then the output will briefly be 1, producing a glitch. If it is there, you won't see it with an LED, but you will with an oscilloscope. This will not be a problem if transitions between states only change one bit (Gray code) at a time. For this example, that is not an option.

The solution is to synchronize the output with a flip-flop and build the output control into the state machine.



This state machine consists of inputs, combinational logic, and flip-flops to store the state and outputs.

The state machine must be designed so that a single flop-flop is reserved for each output. This will, in most cases, require adding more flip-flops than the minimum 2^n states. The advantage is that the outputs will be synchronous. With extra, undefined states, there may be chances for tighter reduction of the logic. The state diagram shown below has three flip-flops, with Q2 reserved for the output.



Three flip-flops create eight possible states. Only four are used, and the other four unused states are tied back to the initial state. The state values have been judicially selected, so Q2 is also the output.

This state diagram has a single input and three flip-flops (Q_2, Q_1, Q_0). Take the diagram and use the information to fill out a truth table. It should look like this.

Present State				Next State			
Q_2	Q_1	Q_0	a	Q_2.d	Q_1.d	Q_0.d	Comments
0	0	0	0	0	0	0	*S0 to S0*
0	0	0	1	0	0	1	*S0 to S1*
0	0	1	0	0	0	0	*S1 to S0*
0	0	1	1	0	1	0	*S1 to S2*
0	1	0	0	0	0	0	*S2 to S0*
0	1	0	1	1	0	0	*S2 to S3*
0	1	1	0	0	0	0	*Sx to S0*
0	1	1	1	0	0	0	*Sx to S0*
1	0	0	0	0	0	0	*S3 to S0*
1	0	0	1	1	0	0	*S3 to S3*
1	0	1	0	0	0	0	*Sx to S0*
1	0	1	1	0	0	0	*Sx to S0*
1	1	0	0	0	0	0	*Sx to S0*
1	1	0	1	0	0	0	*Sx to S0*
1	1	1	0	0	0	0	*Sx to S0*
1	1	1	1	0	0	0	*Sx to S0*

Using this table, construct a four-variable Karnaugh map for each of the three flip-flop's inputs. They should look like this.

Q_2.d

Q_2Q_1 \ Q_0 a	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	0	0	0
10	0	1	0	0

Q_1.d

Q_2Q_1 \ Q_0 a	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

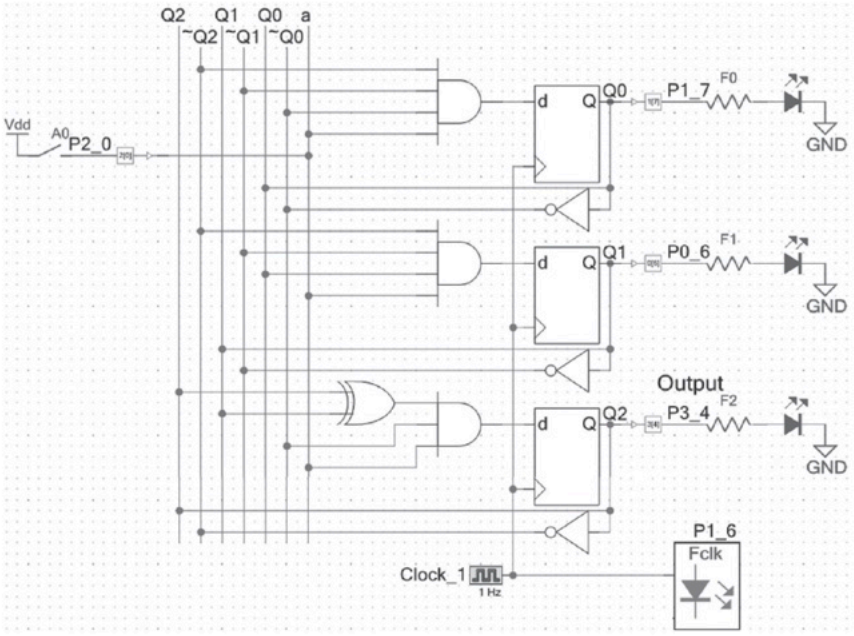
Q_0.d

Q_2Q_1 \ Q_0 a	00	01	11	10
00	0	1	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

The rest of the steps consist of simple logic reduction and implementation.

Instructions

- Open your test bench, and delete anything in the TopDesign schematic.
- Go to the component catalog, and drag in the components shown below.
- Make sure to assign the correct pins to your inputs and outputs.
- Connect them together, as shown below.



- Build this project, and download it to your test board.

If you have done this project correctly:

- The states (flip-flop outputs) will cycle, as shown in the state diagram.
- The output (Q2) is on only when the state machine is in S3.
- While in state S3 and the clock is low, briefly release and press A0. The output should remain on while the input is briefly released.

Test to verify.

Save this project as a macro. Name it "Lab28," and store it in the MyStuff tab under the MyLabs section.

You have completed the twenty-eighth lab.