

Instructions:

LOGIC TUTOR



VERSION: 1

In partnership with



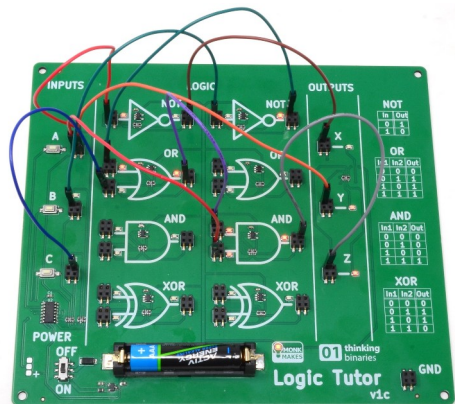
Learn about digital logic by making your own working circuits

- * Fully soldered and ready to use
- * Indicator LEDs on every input and output
- * Powered from a single AAA battery
- * No computer needed

Requires 1 x AAA Battery
(not included)

The kit includes:

- * Logic Tutor circuit board
- * 20 connecting wires
- * Instruction booklet
- * Self-adhesive feet



By: David Whale and Simon Monk

WARNING: Only use the jumper wires to connect terminal blocks. Do NOT connect directly to the battery. To do so may cause a short-circuit and result in the wire getting dangerously hot.

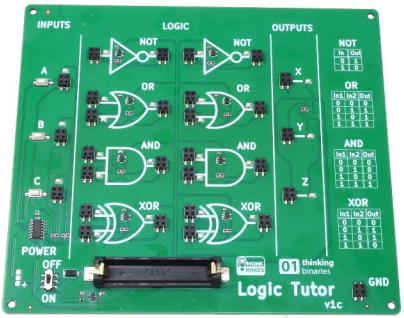
TABLE OF CONTENTS

Introduction.....	3
Parts.....	3
The Logic Tutor.....	4
How To Use.....	5
Wiring Test.....	6
NOT Gate.....	7
OR Gate.....	8
AND Gate.....	9
Truth Tables.....	10
Project 1a – Garden Controller.....	11
Schematic Diagrams.....	12
Project 1b – Garden Controller (pump).....	13
Designing, Documenting and Testing.....	14
Project 2a – Home Alarm System.....	15
Project 2b – Alarm (With Negative Logic).....	16
XOR Gate (Hallway Light).....	17
Project 3a – Combination Lock (Basic).....	18
Gate Equivalence.....	19
Making a NOR Gate.....	20
Project 3b – Combination Lock (Improved).....	21
Wiring Complex Circuits.....	21
Project 3c – Combination Lock (Try).....	22
Making a NAND Gate.....	23
Making a 3-way OR Gate.....	24
Making a 3-way AND Gate.....	25
Gate Reference.....	26
Boolean Equations.....	27
Project 1b Wiring – Garden Controller.....	28
Project 2b Wiring – Home Alarm System.....	29
Project 3c Wiring – Combination Lock.....	30
Troubleshooting.....	31
Learning.....	32
About The Authors.....	32
Books by Simon Monk.....	32
Your Notes Here.....	33
Monk Makes.....	36

INTRODUCTION

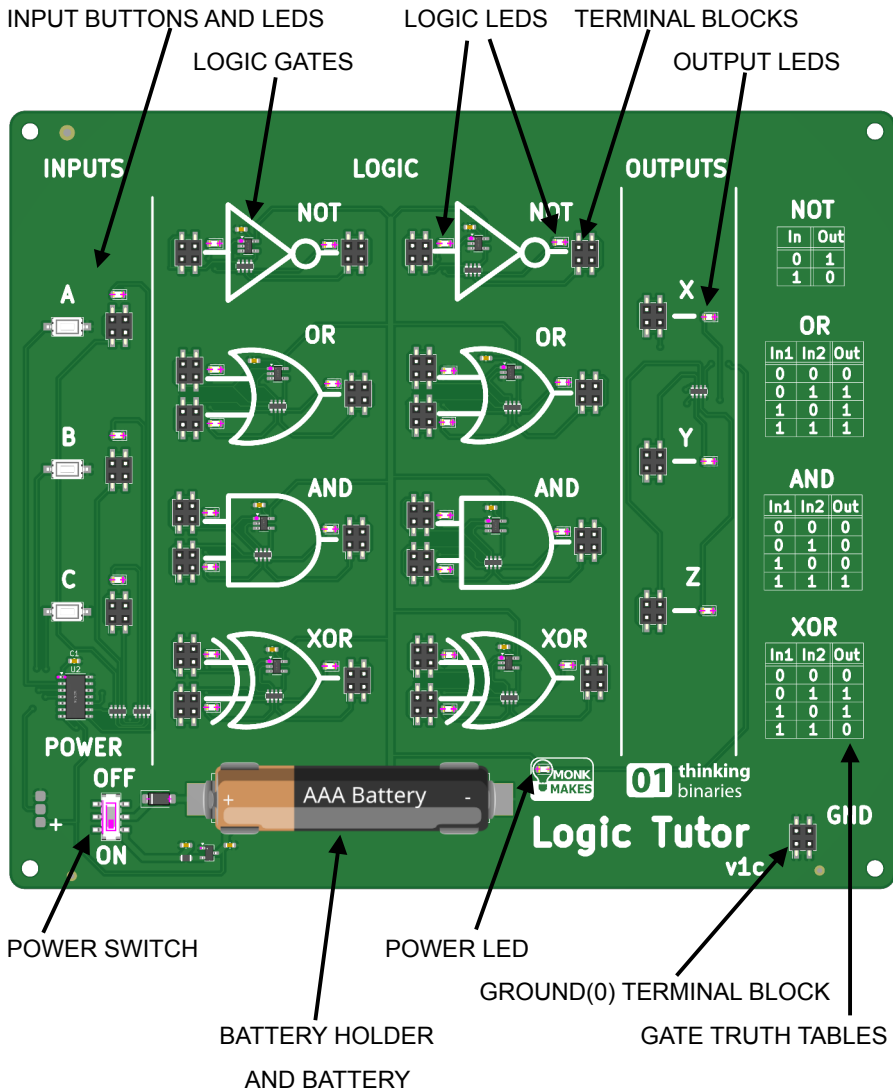
The MonkMakes Logic Tutor is a pre-assembled board that enables you to learn about and experiment first-hand with digital logic. If you use a computer then you have *used* digital logic, but wouldn't it be amazing if you could learn about how the simplest of logic gates used in computers can be combined to make your own circuits? In this booklet, you will learn about logic gates, how to connect them to make bigger circuits, and how to design and test your own circuits. You will build 3 complete projects, and this could be just the start of your journey into digital logic!

PARTS

Qty	Item	Picture
1	Logic Tutor	
4	Rubber feet	
20	Jumper wires (2 bundles of 10)	

* Battery not included – requires 1 x AAA battery

THE LOGIC TUTOR



* (battery not included)

Hint: A logic-gate (or "gate") is the normal name for each of these individual parts of a digital logic circuit. It is called a "gate" because it either lets a digital signal through, or it stops it. So it is a bit like a garden gate at the edge of your garden that is open or closed.

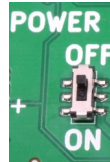
How To Use

1. The first thing to do with your Logic Tutor is to fit one AAA battery – look carefully at the battery pack and insert it so that the bobble on the top (the +) is on the left side, next to the power switch.



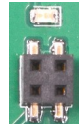
2. Move the small switch at the bottom left of the board to the ON position. You should see the LED inside the MonkMakes logo light up.

(You will also see two of the LEDs near the NOT gates at the top light up – this is normal).

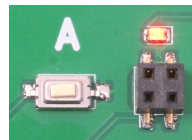


3. The jumper wires can be pushed into the little black square terminal blocks to connect to the gate or the button or LED that is near it.

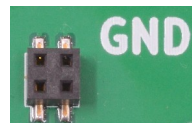
Each terminal block is a group of 4 holes, and you can add up to 4 jumper wires to each terminal block and they will be electrically connected together.



4. To test your circuit, press the A B or C buttons – the buttons are latching, so press them once to turn on, again to turn off – the LEDs next to the buttons will light up when the button is 'on'.



5. The GND terminal block can be used wherever you want a permanent OFF(0) signal. It can also be used to join two or more Logic Tutor boards together with your friends to make bigger circuits.

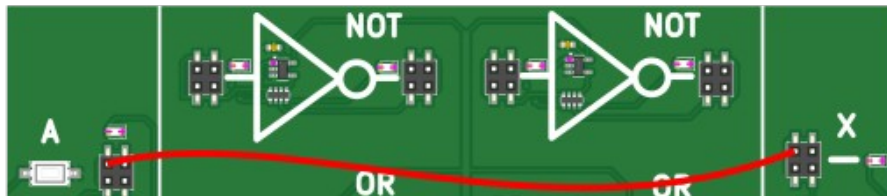


NOTE: When you turn the board off by moving the switch to the OFF position, the buttons forget what state they were in; the board will always power up with all 3 buttons in the OFF state.

NOTE: Peel the rubber feet off the backing paper and stick on the underside of the board in each corner, to make your board more stable when placed on a table.

WIRING TEST

1. Before building digital circuits with your Logic Tutor, let's first test that the buttons and output LEDs work properly. This can be done by connecting a jumper cable between the A terminal block and the X terminal block, as in the following diagram, and is the simplest circuit you could build. (It doesn't matter which of the 4 holes on a terminal block you use, each set of 4 holes are connected on the circuit board).



* Note: It doesn't matter which colour wires you use, although it might be easier to spot wiring errors if you use the same colours as used in the diagrams.

Q. Think of a story to describe this system, such as "A is the fridge door sensor and X is the fridge light. When the door is opened, the light goes on, when the door is closed, the light goes off".

Other things to try

2. Test the other inputs and outputs, wire B to Y and C to Z.
3. Try connecting one input to two outputs (e.g. A to both the X and the Y LEDs) and test what happens when you press the A button.
4. Try connecting one input to three outputs (e.g. A to X,Y,Z), test again with the A button.

Think of new stories that describe your each of your test circuits, and fill in the ideas table.

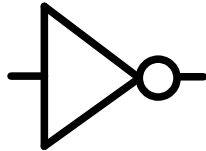
IDEAS: Stories to describe each of your circuits

Circuit #	Story
1	
2	
3	
4	

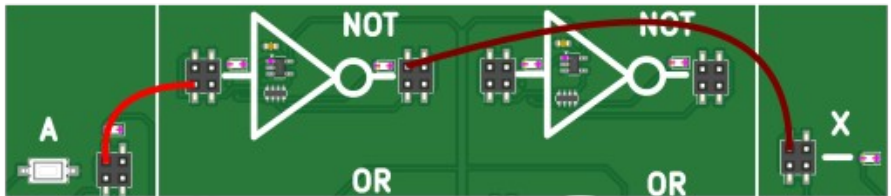
NOT GATE

The simplest logic gate is the NOT-gate – it is sometimes also called an *inverter* because it inverts the input signal to generate the output signal. OFF on the left hand side of the gate (the input) becomes ON on the right hand side of the gate (the output); conversely, ON on the left hand side of the gate becomes OFF on the right hand side of the gate.

Here is the logic symbol of the NOT gate. You can also see there are two independent NOT gates on your Logic Tutor.



Now wire up the left hand NOT gate to the A button on its left hand side (input) and the X LED to the right hand side of the NOT-gate (output), like in the diagram below:



Turn your board on and press the A button several times to see the gate working. Then fill in the results table with your findings, and invent a story to describe what the purpose of your NOT gate circuit could be.

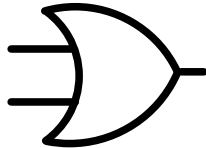
RESULTS: NOT-gate

A	X
OFF	
ON	
Story	

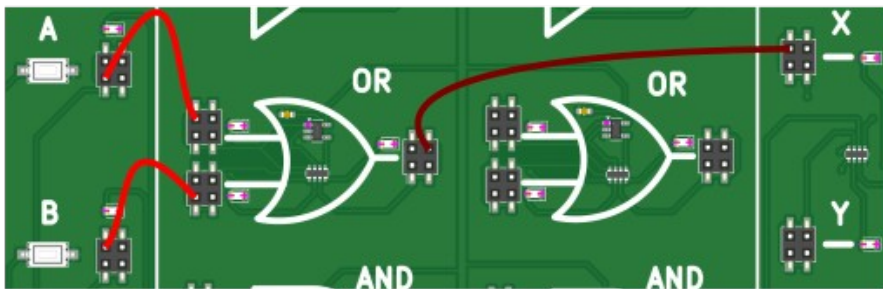
OR GATE

The next gate to experiment with is the OR-gate. This gate combines two inputs to create a single output; its output will be ON if *any* of the inputs are ON – so that's Input 1 OR Input 2 (OR, both at the same time).

Here is the logic symbol of the OR-gate. You can also see there are two independent OR-gates on your Logic Tutor.



Now wire up the left hand OR gate to the A and B buttons on its left hand side (input) and the X LED to the right hand side of the OR-gate (output), like in the diagram below:



Turn your Logic Tutor on and try all combinations of the A, B buttons to see the OR-gate working. Then fill in the results table with your findings, and invent a story to describe what the purpose of your OR-gate circuit could be.

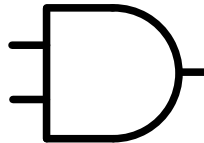
RESULTS: OR-gate

A	B	X
OFF (0)	OFF (0)	
OFF (0)	ON (1)	
ON (1)	OFF (0)	
ON (1)	ON (1)	
Story		

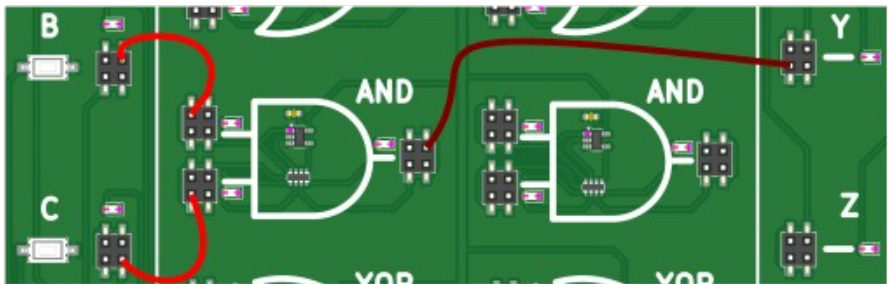
AND GATE

For now, let's finish with the AND-gate. This gate combines two inputs to create a single output; its output will be ON if *both* of the inputs are ON – so that's Input 1 AND Input 2.

Here is the logic symbol of the AND-gate. You can also see there are two independent AND-gates on your Logic Tutor.



Now wire up the left hand AND-gate to the A and B buttons on its left hand side (input) and the X LED to the right hand side of the AND-gate (output), like in the diagram below:



Turn your board on and try all combinations of the A, B buttons to see the AND-gate working. Then fill in the results table with your findings, and invent a story to describe what the purpose of your AND-gate circuit could be.

RESULTS: AND-gate

A	B	X
OFF (0)	OFF (0)	
OFF (0)	ON (1)	
ON (1)	OFF (0)	
ON (1)	ON (1)	
Story		

TRUTH TABLES

As your circuits get bigger and more complex, it is useful to be able to document what they do, independently of the actual gates that make up the circuit – a bit like a specification of the behaviour of the system (without saying exactly how it is going to be constructed). In digital logic, this is called a *truth table*, because it lists all the ‘known truths’ about the system.

This then allows you to test your circuit against this specification, and spot and fix any errors. As you will see later, it also allows you to design *equivalent circuits* that are made from different gates but that operate in an identical way.

A *truth table* is a list of all possible input combinations on the left (the inputs) followed by the desired outputs on the right (the outputs). So really, the results tables you have been recording already are just truth tables.

You can also see that the truth tables for each of the 4 types of gate on the Logic Tutor are printed on the right hand side of the board, next to each gate.

DIGITAL LOGIC

Up until now we have been talking about ON and OFF, because the buttons and LEDs are ON or OFF. You might also see them listed as 0 for OFF and 1 for ON, because digital (or binary) logic works with 0 and 1 – these are sometimes also called a ‘bit’ (for “**binary digit**”). (At the electronics level, OFF is 0 Volts and ON is a positive voltage).

BOOLEANS

There are also times that engineers like to talk about *true* for ON(1) and *false* for OFF(0), and this language was designed by George Boole. It is called Boolean logic. Digital logic and Booleans and binary are all interchangeable here, a signal can only ever be one of two logical values. (If you measured a logic-1 with a meter, it will be a positive voltage, but you might get slightly different readings).

Now, complete the 3 truth-tables below, using the results from your earlier experiments. Use 0 for OFF/false and 1 for ON/true. (Note, truth tables always have a double-line separating the inputs on the left, from the outputs on the right).

RESULTS: Truth Tables

NOT	
In	Out
0	1
1	

OR		
In1	In2	Out
0	0	0
0	1	
1	0	
1	1	

AND		
In1	In2	Out
0	0	0
0	1	
1	0	
1	1	

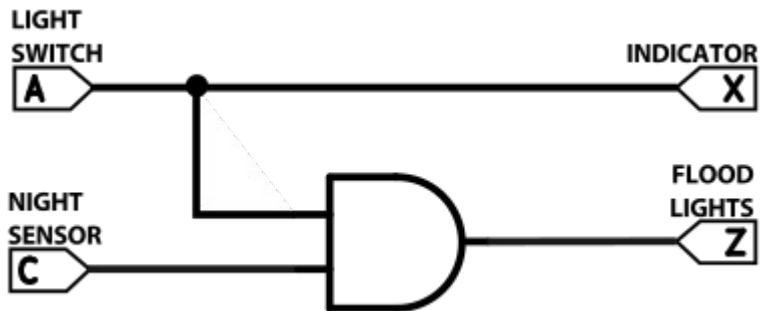
PROJECT 1A – GARDEN CONTROLLER

PURPOSE: The garden controller is used to control floodlights in the garden.

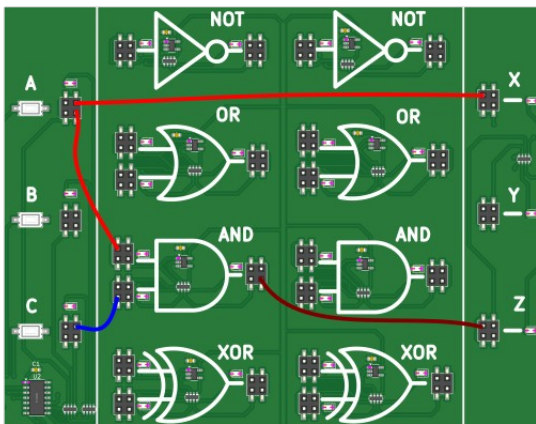
INPUTS: A = The *light switch* is a button that people change to turn the floodlights on. B = The *night sensor* is an automatic sensor that gives an ON signal whenever it is dark outside.

OUTPUTS: X = The *indicator* is a small light next to the light switch that indicates the floodlights have been requested by the light switch. Y = The *flood lights* signal is ON to turn the flood lights on.

LOGIC: The *indicator* always follows the *light switch*. The *flood lights* are ON only if it is night-time and the *light switch* is on.



Here is the wiring diagram for the circuit. Now build and test it, and fill in the truth table by testing all combinations of inputs.



A	C	X	Z
0	0		
0	1		
1	0		
1	1		

* Note: You could also label C as a LIGHT_SENSOR and connect it via a NOT-gate to the AND-gate input. See later, where you will learn about negative logic.

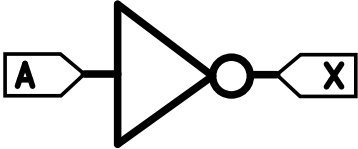
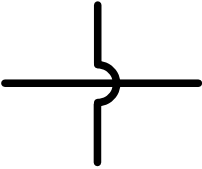
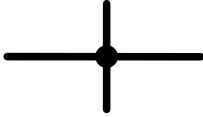
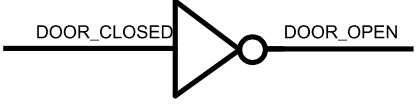
SCHEMATIC DIAGRAMS

A *schematic* is like a plan for a circuit; it describes how all the components in the circuit are connected together and how they are connected to the outside world, but without describing physically where the components are located. Digital logic engineers will draw a schematic as a way to plan out a circuit, and then later on they will decide how to wire it up on a circuit board.

When a circuit design gets bigger, your schematic will look more complex, because there are more logic gates to connect together, and therefore there are more connecting wires. Sometimes there are lots of wires and the circuit begins to look a bit of a mess!

DRAWING SCHEMATICS

Digital logic design engineers use a number of techniques to make their diagrams more readable. Let's look at a few here:

1. <i>Left to right</i> – Inputs go on the left, with outputs on the right; this is so that you can 'read' a circuit just like a book, from left to right, and follow the input signals through the circuit to the final outputs.	
2. <i>Crossing wires</i> – these have a little 'bridge' that hops over another wire, to make it clear that there is no electrical connection at the point they cross.	
3. <i>Multiple connections</i> – these have a little black dot; the dot visually confirms that the two wires really are connected and that they represent the same signal.	
4. <i>Labelling intermediate signals</i> – Sometimes the 'inner wiring' of a complex circuit can also have signal labels, to clarify exactly what that intermediate signal represents.	

PROJECT 1B – GARDEN CONTROLLER (PUMP)

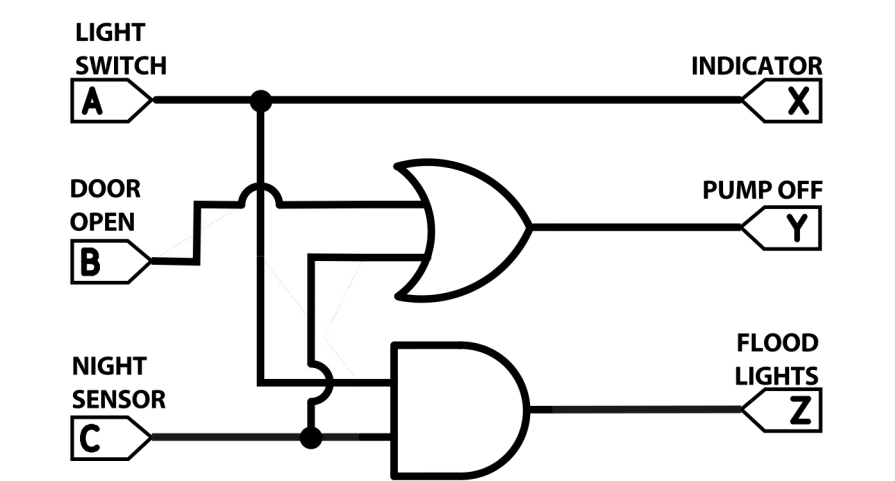
Your garden is being upgraded, and you are adding a fountain pump. So, the circuit additionally has some extra logic:

PURPOSE: A fountain pump makes the garden look nice, but you don't want it annoying your neighbours or for visitors to the garden to get wet.

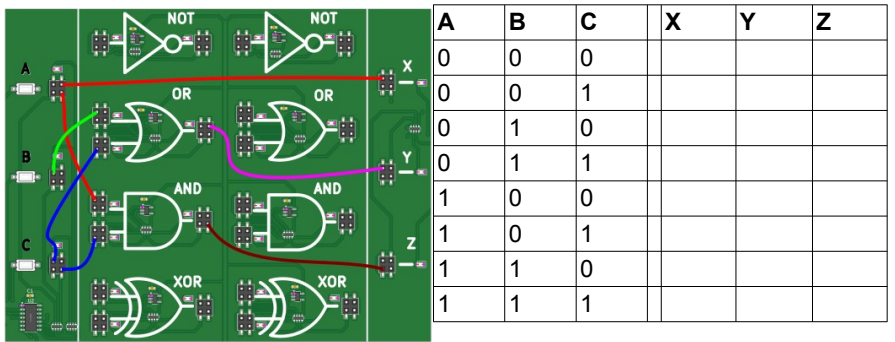
INPUTS (new): B = The *door open* sensor is ON when the door has been opened.

OUTPUTS (new): Y = The *pump off* is ON to stop the pump from running.

LOGIC (new): When the *door is open*, or when it is *night-time*, the *fountain pump* will be stopped.



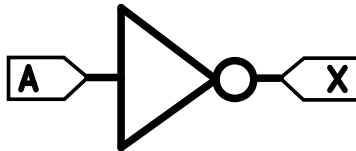
Add the extra wires like below, and test that the pump goes off at the right times.



DESIGNING, DOCUMENTING AND TESTING

Digital logic design engineers will use a variety of diagrams and methods to design, build, and test their circuits.

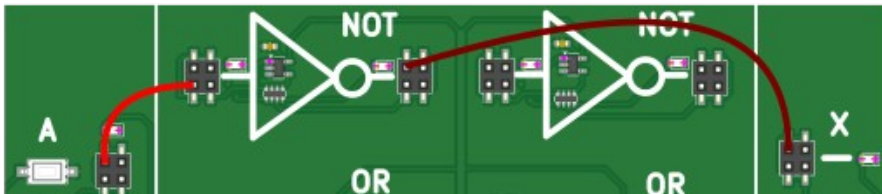
1. A *schematic* is like a plan for the circuit; it describes how all the components in the circuit are connected together and how they are connected to the outside world, but without describing physically where the components are located. Here is an example of a schematic diagram, for a NOT-gate circuit.



2. A *truth table* describes the expected behaviour of a circuit, and it can be used both as part of the design, but also as part of the testing. Here is the truth table for the above NOT-gate circuit.

A	X
0	1
1	0

3. A *wiring diagram* is a detailed assembly diagram; it is logically equivalent to the schematic but also tells you where to place specific components and wires. Here is a wiring diagram for the above NOT-gate circuit.



In the remainder of this booklet, we will just show the schematic, and it is your task as a designer to work out how to wire it up on your Logic Tutor. But don't worry, if you get stuck, there are a complete set of wiring diagrams at the back of this booklet that you can refer to.

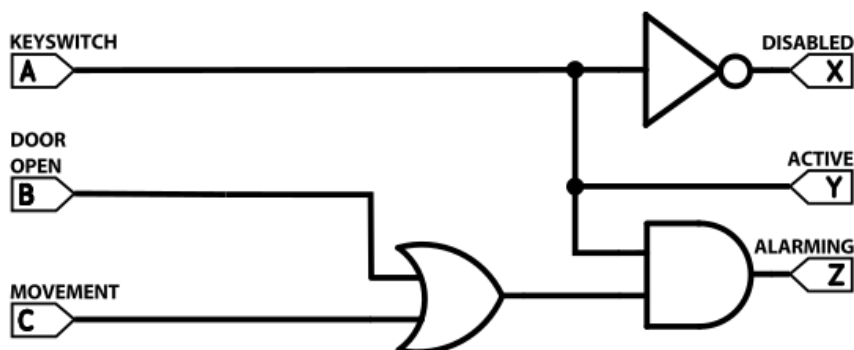
PROJECT 2A – HOME ALARM SYSTEM

PURPOSE: The home alarm system is used to protect your house from intruders.

INPUTS: A = *keyswitch*, B = *door sensor*, C = *movement sensor*.

OUTPUTS: X = *disabled* indicator, Y = *active* indicator Z = *alarming* indicator.

LOGIC: The *disabled* indicator is ON when the *keyswitch* is OFF. The *active* indicator is ON when the *keyswitch* is ON. The *alarming* indicator is ON when the *door sensor* says the door is open(ON), or when the *movement sensor* says that there is movement(ON) and the *keyswitch* has enabled the system.



RESULTS: Home Alarm System #1

A	B	C	X	Y	Z
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

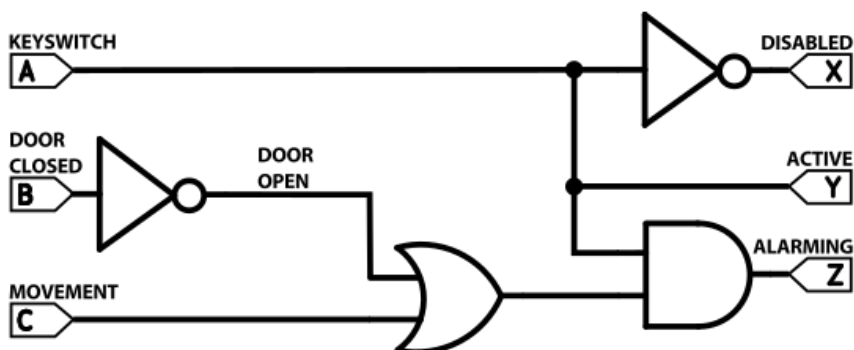
PROJECT 2B – ALARM (WITH NEGATIVE LOGIC)

Normally in alarm systems, a door sensor will provide a *true* signal (ON) when the door is closed. This is so that if an intruder cuts the wires going to the door sensor, the alarm will sound because the system has been tampered with.

In your alarm system, the logic currently treats the door sensor as a *door opened* signal (ON when the door is open, OFF when the door is closed). It would make sense to invert this signal here, and it is called this *negative logic*.

You can easily make this change by inserting a NOT-gate between B and the input to the OR-gate.

Here are the modifications to make to your circuit for the new NOT gate.



Change the wiring on your board to match the change below, and fill in the new results table.

RESULTS: Home Alarm System #2, using negative logic on B (door sensor)

A	B	C	X	Y	Z
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

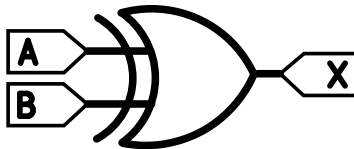
XOR GATE (HALLWAY LIGHT)

You might have noticed there is an extra pair of gates on the Logic Tutor that you haven't used yet, right at the bottom of the board? They look a bit like an OR-gate with the curvy left hand side, but there is an extra line on them – what do they do?

A normal OR-gate is really called an *inclusive-OR-gate*, because the output is ON if *any* input is ON (including *both* inputs at the same time).

Sometimes it is useful to have an OR-gate that doesn't include the 'both inputs ON' part – and that is called an *exclusive-OR-gate* ("exclusively one input or the other, but not both").

Wire up your Logic Tutor like the schematic below, and fill in the results table.



Note, the XOR-gate can also be considered to be a *comparator*, because it compares the two inputs to see if they are different. If they are different, the output is ON, if the two inputs are the same, the output is OFF.

Can you see how this could be used like a two-way hallway light switch, where one switch is upstairs and one switch is downstairs? Work through your truth table for the XOR-gate and imagine yourself going up and down the stairs turning the light on and off at each stage, to convince yourself this is correct.

RESULTS: XOR-gate truth table

A	B	X
0	0	
0	1	
1	0	
1	1	

PROJECT 3A – COMBINATION LOCK (BASIC)

PURPOSE: The combination lock can be used to enable and disable a home alarm system.

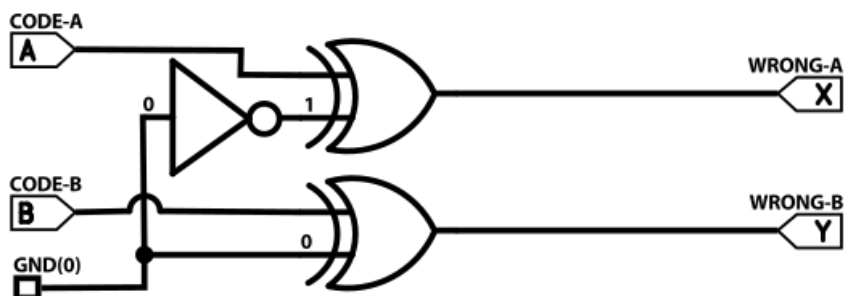
INPUTS: A = The *first digit* of the combination to try. B = The *second digit* of the combination to try.

OUTPUTS: X= The *first digit is wrong*. Y = The *second digit is wrong*.

LOGIC: If the first digit(A) is wrong, X(WRONG-A) is ON. If the second digit(B) is wrong, Y(WRONG-B) is ON.

To make the combination lock, you will use two XOR-gates, one gate for each digit of the combination. Because there is only two XOR-gates on the Logic Tutor, this will be a 2-digit combination lock, and the digits are binary (0 or 1); it's a bit like a button pad that has buttons that either push in or pop out.

Because an XOR-gate is really a comparator, if one of its inputs is the button and the other input is connected to the desired code, the output will be ON if the two are different/wrong. So, your combination lock is configurable based on what logic levels you apply to the *other* inputs of the XOR-gates. A configured 0 means that the code must be a 0, and a configured 1 means that the code must be a 1.



Build the circuit above, but see how the GND(0) terminal block is used to configure one digit to a 0, and then a NOT-gate inverts that GND(0) into a 1 for the other digit. Test your circuit and fill in the results table.

RESULTS: Basic combination lock truth table

A	B	X	Y
0	0		
0	1		
1	0		
1	1		

GATE EQUIVALENCE

The outputs of the combination lock sort of give the game away, because they show the intruder which digits are wrong. It would be much better to have a single signal, and wouldn't it be better if it was a CORRECT signal, so that it could be used directly to drive a latch that opens on the door?

When designing digital logic circuits, two circuits are equivalent if their truth tables are the same in every row. So, let's look at the type of gate needed to convert two WRONGs into a CORRECT:

WRONG A	WRONG B	WRONG	CORRECT
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Looking at the previous gate truth tables you worked out earlier, can you see that the WRONG column is the same as an OR gate, because the truth table is the same on all rows.

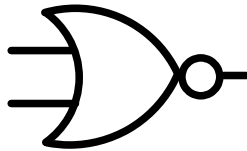
OR-gate truth table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

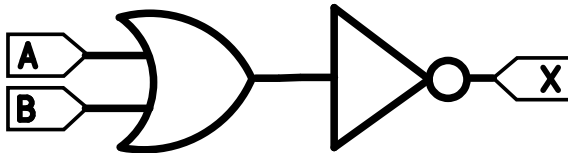
The only difference between the desired truth table, and the OR-gate truth table, is that the final output is the wrong way round. Is it possible to fix that problem?

MAKING A NOR GATE

You can buy a chip that is a NOR gate, and here is its logic symbol:



But you don't want to modify the Logic Tutor. Fortunately you can make a NOR-gate by joining together an OR-gate and a NOT-gate like this:



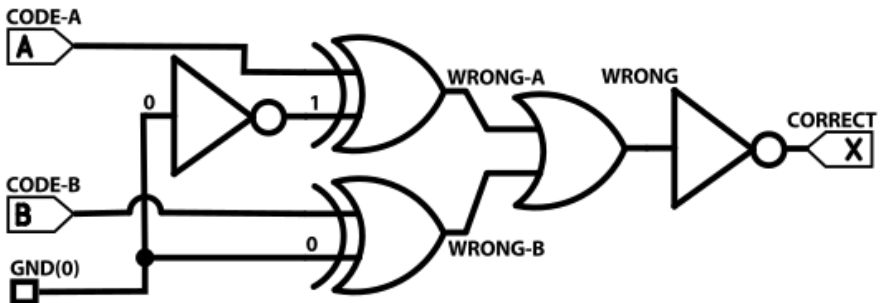
Build this new circuit and test it, and fill in the results table.

RESULTS: Home-made NOR-gate

A	B	X
0	0	
0	1	
1	0	
1	1	
Story		

PROJECT 3B – COMBINATION LOCK (IMPROVED)

Now you have a NOR-gate (NOT + OR) which turns two WRONG signals into a CORRECT signal, let's add this into the combination lock.



Build the new circuit, and fill in the results table.

RESULTS: Improved combination lock, using NOR gate.

A	B	X
0	0	
0	1	
1	0	
1	1	

WIRING COMPLEX CIRCUITS

The digital circuits in this final project are getting quite complex to wire now. You can look at the wiring diagrams in the back of this booklet to help.

However, when electronics engineers are wiring up complex circuits by hand, they will often work from the schematic, and work very systematically, adding one wire at a time and then marking it off on the schematic with a tick, or colouring over it with a highlighter pen.

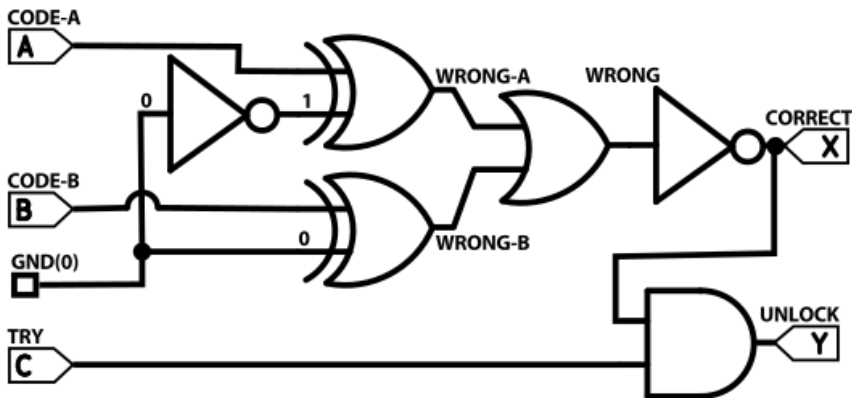
Where there are multiple gates of the same type, you can number or label the gates in the schematic to match those on the circuit you are wiring; in this case, where there are two XOR-gates, label one as L for left, and the other as R for right.

Finally, if part of your circuit doesn't seem to be working, remember that each gate on the Logic Tutor board has an LED on every input and every output, so that you can 'follow the logic through the board' as you are testing it, and you might spot a wiring error that way.

PROJECT 3c – COMBINATION LOCK (TRY)

It would be better if your combination lock worked like a real lock, and you had to press a TRY button to try the combination, as this would slow down an intruder who was trying all combinations.

You can do this by adding an AND-gate, so that the 'try this code' button enables and disables the combination output LEDs. Using an AND-gate in this way provides a sort of master-override for any circuit, so that regardless of what other logic circuit you connect to the AND-gate, the *try* button must be ON for the output of the AND-gate to be ON.



Build and test your modified circuit, and fill in the results table.

RESULTS: Improved combination lock with added TRY button.

A	B	C	X	Y
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Note: In this circuit, the X(CORRECT) output is still connected. You can use this while testing and debugging your circuit.

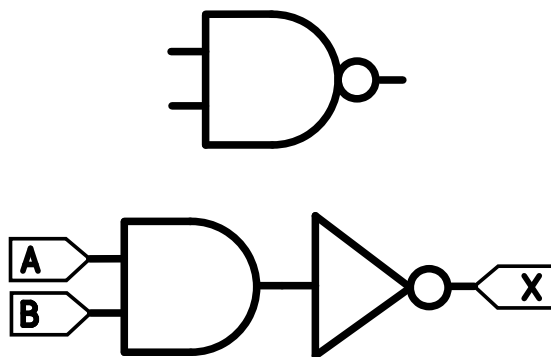
But, when everything is working you might choose to disconnect the wire to X, so that your intruder doesn't get an extra clue to what the unlock code is.

Anything that slows down an intruder working through all combinations, makes your circuit a much better design.

IDEA: If you have a friend that also has a Logic Tutor, you might like to work with them to join your combination lock to their home alarm system, so that entering the correct combination replaces their key-switch and disables the alarm.

MAKING A NAND GATE

Sometimes you may need to use negative logic on an output. Just like with the OR + NOT to make a NOR-gate, you can do the same with the AND-gate, AND + NOT = NAND. Here is the logic symbol for a NAND-gate, and the schematic for the equivalent circuit as you can build it on the Logic Tutor.



Build and test your circuit, and fill in the results table.

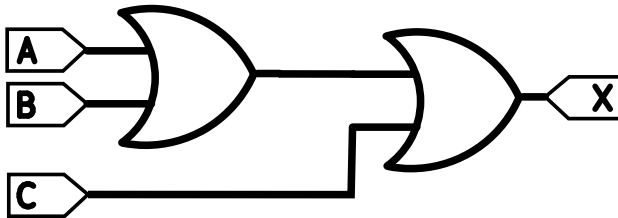
RESULTS: NAND-gate Truth Table

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0
Story		A is the front door sensor, B is the back door sensor. X is a warning light. Sensors output a '1' when a door is closed, and a '0' when a door is opened. If any or both doors are opened, the warning light comes on. Only when both of the doors are safely closed does the warning light go off. You could build this circuit on another LogicTutor and attach it to the DOOR OPEN signal of the home alarm system.

Fun fact! You can make any other gate out of a combination of just NAND-gates – engineers use this when designing silicon chip logic circuits, and will often talk about an 'all-NAND solution'.

MAKING A 3-WAY OR GATE

Sometimes you might want to build bigger circuits that check that any one of 3 inputs is ON – this is called a 3-way OR-gate, and you can build it with two OR-gates.



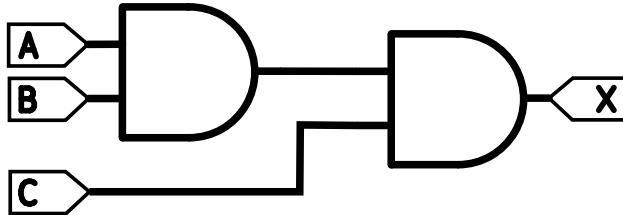
Build and test your circuit, and fill in the results table.

RESULTS: 3-Way OR-gate Truth Table

A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	
Story		There are three fire alarm trigger panels in a building. If any number of the fire alarm trigger panels are triggered, the fire alarm sounds.	

MAKING A 3-WAY AND GATE

Sometimes you might want to build bigger circuits that check that all 3 inputs are ON – this is called a 3-way AND-gate, and you can build it with two AND-gates.



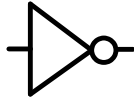
Build and test your circuit, and fill in the results table.

RESULTS: 3-Way AND-gate Truth Table

A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	
Story			A quiz system has a button for each of 3 players. All 3 players must press their button to say that they have finished writing their answers, before the quiz host stops a timer and checks if their answers are correct.

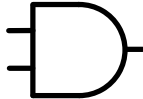
GATE REFERENCE

NOT	In	Out
	0	1
	1	0



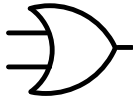
The output is the opposite of the input.

AND	In 1	In 2	Out
	0	0	0
	0	1	0
	1	0	0
	1	1	1



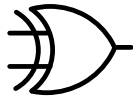
The output is a 1 when both inputs are 1.

OR	In 1	In 2	Out
	0	0	0
	0	1	1
	1	0	1
	1	1	1



The output is a 1 when any inputs are 1.

XOR	In 1	In 2	Out
	0	0	0
	0	1	1
	1	0	1
	1	1	0



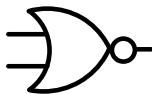
The output is a 1 when the inputs are different.

NAND	In 1	In 2	Out
	0	0	1
	0	1	1
	1	0	1
	1	1	0



The output is a 0 when both inputs are 1.

NOR	In 1	In 2	Out
	0	0	1
	0	1	0
	1	0	0
	1	1	0



The output is a 0 when any input is a 1.

XNOR	In 1	In 2	Out
	0	0	1
	0	1	0
	1	0	0
	1	1	1



The output is a 1 when both inputs are the same.

* The Logic Tutor doesn't have NAND, NOR or XNOR gates, but you can build your own by adding a NOT-gate to the AND, OR, XOR gates.

BOOLEAN EQUATIONS

You have seen how a truth table completely describes the behaviour of a digital logic circuit, but there is another method that can be used as a design description; it is called a *boolean equation*, named after George Boole, who invented Boolean notation. It is a very compact way to describe digital circuits.

A boolean equation is very much like a mathematical equation, it states that the logic on the left hand side of the equals sign is the same as the logic on the right hand side.

For signal names you can use the A B C X Y Z letters from the Logic Tutor (or you could use the long-form names of the project like KEYSWITCH and DOOR_OPEN).

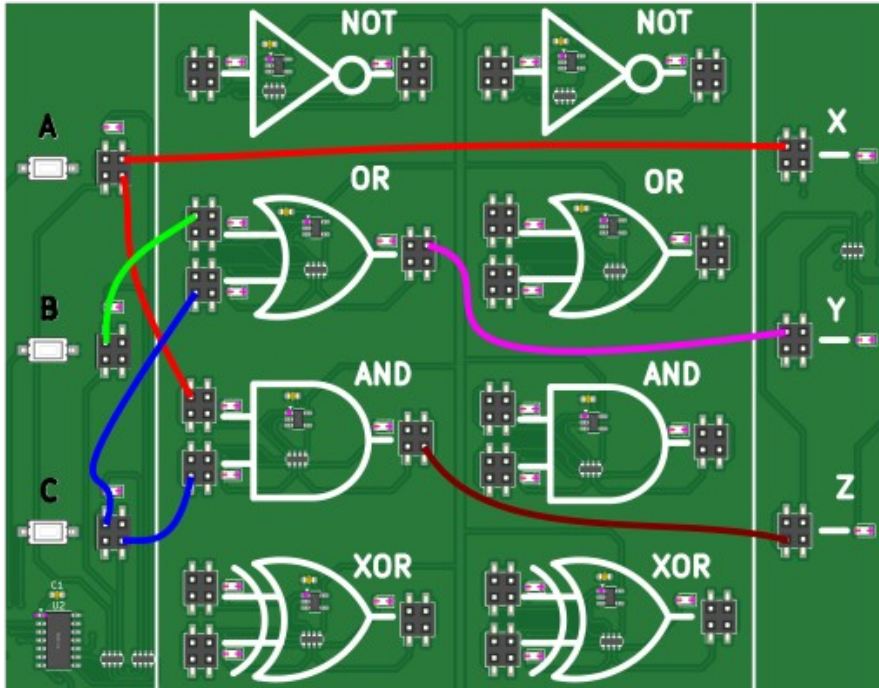
Anything with a horizontal bar over the top of it is a NOT of that symbol.

EXAMPLES

Equation	Meaning
$X = A$	X is identical to A. When A changes, X changes.
$X = A + B$	X is A OR'ed with B.
$X = A \cdot B$	X is A AND'ed with B.
$X = A \oplus B$	X is A XOR'ed with B.
$X = \overline{A}$	X is the NOT of A.
$X = \overline{A + B}$	X is A NOR B (NOT OR)
$X = \overline{A \cdot B}$	X is A NAND B (NOT AND)
$X = \overline{A \oplus B}$	X is A XNOR B (NOT XOR)
$X = A + \overline{B}$	X is A OR NOT B

The boolean equations for the 3 projects are included in the final wiring diagrams so that you can see them. You won't use them here, but digital circuit designers use these all the time, and some of the other resources on the companion website will use boolean equations to perform some amazing magic on the circuit designs!

PROJECT 1B WIRING – GARDEN CONTROLLER



Signal	Label	Description
A	LIGHT_SWITCH	'1' when the switch is on to request the lights.
B	DOOR_OPEN	'1' when the door has opened.
C	NIGHT_SENSOR	'1' when the sensor senses it is dark.
X	INDICATOR	'1' to indicate the lights are requested.
Y	PUMP_OFF	'1' to turn the pump off (inhibited).
Z	FLOOD_LIGHTS	'1' to turn the flood lights on.

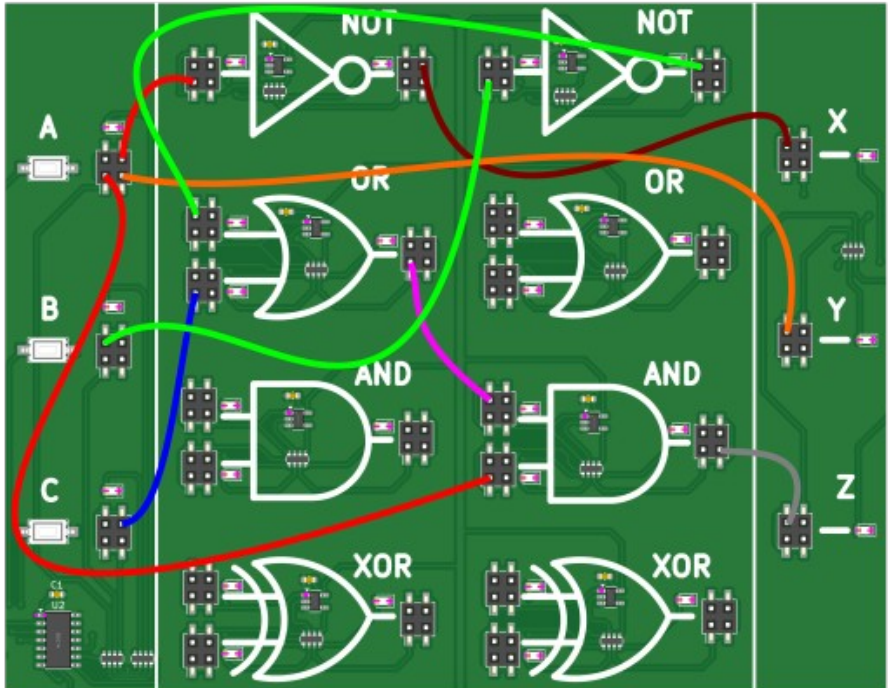
BOOLEAN EQUATIONS

$$X = A$$

$$Y = B + C$$

$$Z = A \cdot C$$

PROJECT 2B WIRING – HOME ALARM SYSTEM



Signal	Label	Description
A	KEYSWITCH	'1' when the key-switch enables the alarm.
B	DOOR_CLOSED	'1' when the door is closed.
C	MOVEMENT	'1' when movement has been detected.
X	DISABLED	'1' to indicate the alarm is disabled/safe.
Y	ACTIVE	'1' to indicate the alarm is active/monitoring.
Z	ALARMING	'1' to indicate the alarm is sounding.

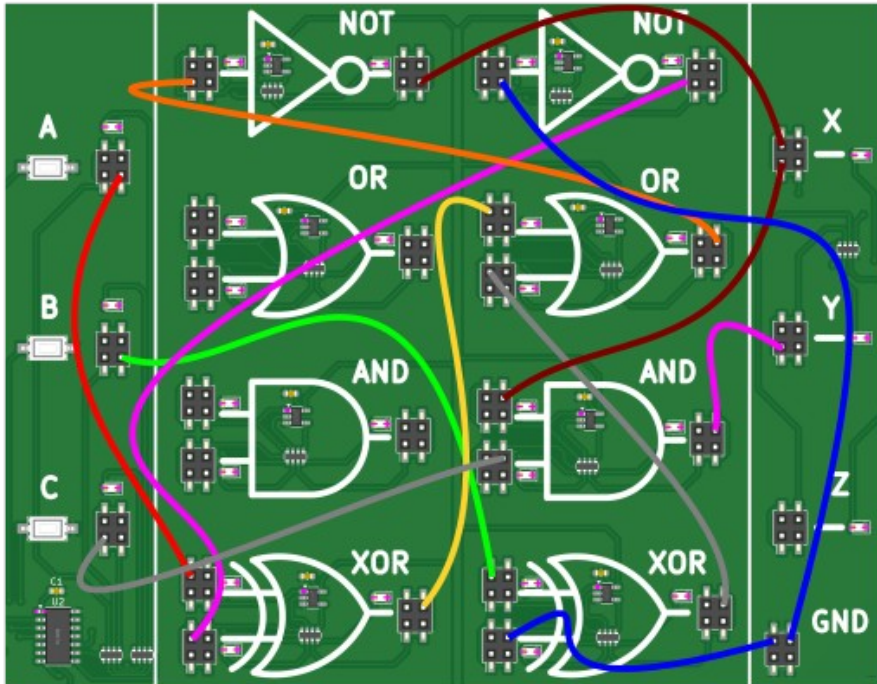
BOOLEAN EQUATIONS

$$X = \bar{A}$$

$$Y = A$$

$$Z = A \cdot (B + C)$$

PROJECT 3C WIRING – COMBINATION LOCK



Signal	Label	Description
A	CODE_A	'1' or '0' for the first part of the combination.
B	CODE_B	'1' or '0' for the second part of the combination.
C	TRY	'1' to check the combination and unlock if correct.
X	CORRECT	'1' to indicate the code is correct. (for debugging).
Y	UNLOCK	'1' to unlock the door latch.
Z	(NOT USED)	

BOOLEAN EQUATIONS

$$X = \overline{(A \oplus 1)} + (B \oplus 0)$$

$$Y = X \cdot C$$

TROUBLESHOOTING

None of the LEDs are lighting – Make sure the power switch is ON. Is the battery running low? If so, try a new battery. Have you fitted the battery the correct way round? If the MonkMakes logo LED is lit, the power to the board is correct, so check your circuit. Remember that when you power the Logic Tutor OFF, the buttons will forget their state and always power up in the OFF state. Make sure there are no short circuits on your board that are connecting together a 1 output with a 0 output.

The NOT-gate LEDs light up, even without a wire fitted – this is normal. It is because $\text{NOT}1 = 0$ and $\text{NOT}0 = 1$, so you would expect either the input LED to be ON, or the output LED to be ON.

I accidentally broke a jumper pin inside the terminal block – Don't worry, the board was designed to cope with this possibility. There are holes in the back of the circuit board under each terminal block, so just push another jumper pin in and the broken bit will fall out the back of the circuit board. We have added a few spare jumper leads to the kit in case this happens.

When I power up the Logic Tutor the buttons are not in the same state as when I last used it – This is normal. To keep the cost of the board as affordable as possible, the buttons are momentary-push buttons and there is a little extra chip on the board that simulates latching buttons. Every time you power up the Logic Tutor, all 3 buttons will automatically be in the OFF state and their LEDs will be OFF.

One of the gates isn't operating according to its truth table – This can happen if you accidentally connect together two outputs from different gates. Don't worry, no damage will be done. But turn the power off and check and correct your wiring.

I can't work out the wiring diagram from the logic diagram – you can find the full set of wiring diagrams in the back of this booklet. We didn't put them alongside the later projects, because we think it is better that you learn how to build a circuit from the schematic.

My circuit isn't working as I expected – debugging faulty circuits is a good skill to learn. Be systematic and follow through the logic carefully from left to right, looking at the LEDs on each of the gates and checking that they are doing what you expect. Test your circuit by trying out each combination in its truth table one by one, and if you spot a line in the truth table that doesn't match, check the wiring around the gate that generates that specific output.

The B LED is flashing – Well done, you found the easter-egg that we added to this board – can you work out what the flashing pattern represents?

If you are still stuck, please email support@monkmakes.com.

LEARNING

OTHER MATERIALS

You can find other resources to learn about digital logic on the product web page here:

www.monkmakes.com/logic_tutor

Look out for project worksheets on the website that give you more ideas of things to build with your Logic Tutor.

Teachers! Our project worksheets are downloadable and customisable, so you can download and modify them to use with your students. Also look out for our teacher guide booklet.

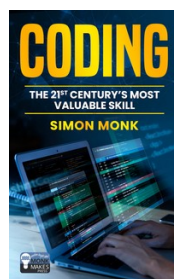
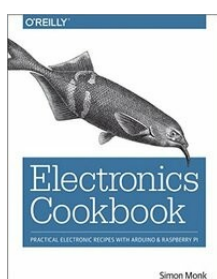
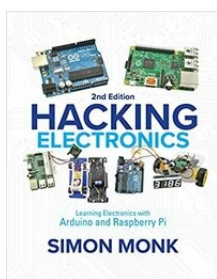
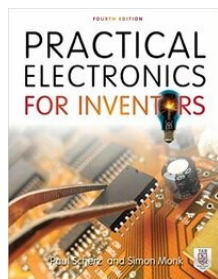
Get in contact if you are interested in buying a class set of Logic Tutors for your teaching. e-mail: sales@monkmakes.com

ABOUT THE AUTHORS

David Whale is a STEM Ambassador (volunteer) who helps schools and teachers throughout the UK to inspire young people to study technology. He has spent over 40 years designing software that goes inside electronic products, and has authored and edited a number of successful technology books and magazines. He also collaborates with MonkMakes, writing resources for some of their products.

Simon Monk designs products for MonkMakes, a UK-based manufacturer of electronic and hobbyist kits and PCBs. All of our kits are manufactured in Lancashire in the UK. Simon has also written many books about electronics and programming. You can read about this at www.simonmonk.org and about MonkMakes at www.monkmakes.com.

BOOKS BY SIMON MONK



YOUR NOTES HERE

MONK MAKES

CONTACT

For more information about this kit, the product's home page is here:

https://monkmakes.com/logic_tutor

As well as this kit, MonkMakes produce all sorts of kits and gadgets to help with your maker projects. Find out more, as well as where to buy here:

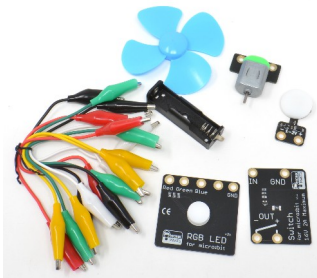
<https://www.monkmakes.com/products>

You can also follow MonkMakes on Facebook and Instagram @monk_makes_ltd.

OTHER KITS



Solar Experimenters Kit



Electronics Kit 2 for micro:bit



CO2 Dock for micro:bit