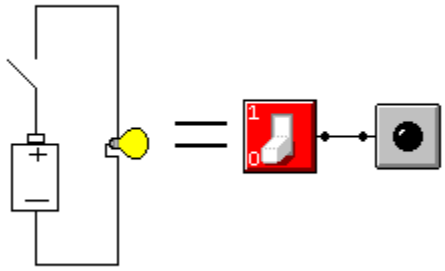
Introduction

Welcome to one of the most simplest, most powerful, most universal languages known (Digital Logic). Most digital logic drawing systems are just that, drawing systems. Recently, some logic drawing systems allow circuits to be activated as they are drawn for testing purposes. But the input and output to these systems are typically files of numbers. The MultiMedia Logic Simulator has taken this one step further and introduced devices that connect directly to your computers' real devices (e.g. Keyboard, Screen, Serial Ports) including MultiMedia ones (PC speaker, Wave, Bitmaps).

The intent of this system is not to necessarily build the logic circuit you design. The intent is to use what you build, to allow experimentation, to learn and to have fun.

What is a MultiMedia Logic Simulator

Lets start with, what is a logic simulator? A logic simulator is a high level simulation of a "digital" circuit, the kind used in a computer. At this level of simulation we ignore the details of the "real world" such as power for circuits, how fast circuits are, ground signals, cost and etc. These details are ignored so that you can deal directly with the logical aspect of the problem at hand. In the "real world", when you're ready to build your circuit you can worry about these details. Here is a simple example of real world versus the logic simulated world.



The great thing about the MultiMedia Logic Simulator is you can use your logic circuit without building real hardware, in fact you can connect it to (interface it with) real hardware (such as your keyboard, speaker, sound card, robot or another Logic Simulator).

States of a node

The MultiMedia Logic Simulator is a 6 state simulator. A node can be in the following states:

0	LO (OFF or False)
1	HI (ON or True)
U	UNKNOWN (Neither True or False)
-	Pulled down, acts like 0 but can be over driven
+	Pulled up, acts like 1 but can be over driven
?	Floating, acts like U but can be over driven

What is UNKNOWN used for? In many circuits there are conditions where the simulator cannot determine if the state should be 0 or 1. You can think of the old saying "Innocent Until Proven Guilty". A node is UNKNOWN until it is proven to be 1 or 0 by the simulator. What this does is flush out initialization problems in your circuits. There are cases where the simulator cannot behave like an imperfect "real world" device and you need to override this behavior. You can do this by using a Force 0 or Force 1 property on a NODE device.

The +, - and ? are only produced by the Tristate Device and processed by the Bus Device. You don't need to understand these until you get to more advanced circuits.

Modes of Operation

The Multimedia Logic Simulator can be in several different modes of operation.

- Ready mode, ready to start drawing or start simulator.
- Drawing mode, devices can be added, deleted, moved, wired and modified.
- Running mode, devices can be operated, heard, and seen.
- Pause mode, examine circuits by stepping through them one cycle at a time.

Click  to enter Drawing mode.

Click  to start the simulation.

Click  to stop a simulation that is running.

Click  to pause a simulation that is running.

Click  to reset a simulation that is paused.

Click  to run the simulation for one cycle and then pause.

Simulation Setup

The simulator can handle unconnected inputs in several ways. The conservative mode is to assume all unconnected inputs are being driven with Unknown (U). This mode flushes out possible problems with your circuit. The problem with this is, that it is tedious to tie off unused inputs with grounds. So the simulator offers both modes and more. The mode is kept in the project file.

IF Input is unconnected

THEN consider it Unknown (U)
THEN consider it LO (0)
THEN consider it Pulled Down (-), it can still be over driven
THEN consider it Floating (?), it can still be over driven

To change the mode see Menu->Simulate->Setup...

Tutorial

This describes how to build a very simple circuit.

- Start a new circuit by clicking on  from the toolbar.
- Bring up the Tool Palette by clicking on  from the toolbar.
- Select the Switch by clicking on  from the palette.
- Place a switch by clicking the left mouse button on schematic.
- Select an LED by clicking on  from the palette.
- Place an LED by clicking the left mouse button to the right of the switch.
- Select the Wiring device by clicking on  from the palette.
- Click the left mouse button down on the output node (black dot) and hold it.
- Drag the mouse over to the LED input node (black dot) and let mouse go.
- Select the Pointer Tool by clicking on  from the palette.
- Double click on the center of the LED  .
- Change the color to Yellow and Click OK.
- Disable the Tool Palette by clicking on  from the toolbar.
- Run the simulator by clicking on  from the toolbar.
- Click on the Switch  on the schematic and notice what happens.
- Stop the simulator by clicking on  from the toolbar.

This tutorial is designed to teach you the basics of using the simulator. It is not an attempt to teach logic design.

Pointer Tool 

The pointer tool is the most common tool you will use. Use it to:

- Select a Device (single click on device).
- Select a Wire(s) (single click on end of wire(s)).
- Set Device Properties (double click on device).
- Move a Device (left mouse button down on device and then drag).
- Select a group of Devices (left mouse button and drag around devices).
- Delete Device(s) (Select Device(s) and hit <DEL> key).
- Copy Device(s) (Select Devices(s) and go to Edit Menu->Copy).
- Delete Wire(s) (Select Wire(s) and hit <DEL> key).
- Toggle Select (Hold <Shift> down while selecting).
- Clone Devices(s) (Select Devices(s) and Hold <Ctrl> down while dragging).

Note: Wires cannot be moved, you must delete them and wire again. However, they will be rerouted if you move a Device with wires connected.

Wiring Tool 

The Wiring tool allows you to connect devices.

Rules:

- For most devices, all the inputs nodes should be connected to operate.
- You always wire outputs to inputs.
- Outputs are on the right side of the device.
- Inputs are on the left side of the device.
- Outputs can have many connections.
- Inputs can only accept one connection.

To Wire a device:

- Select the Wiring Tool.
- Click the Left Mouse Button down (and hold it down) on a Node (black dot).
- Now drag it to another Node on another device and let mouse button go.

To Remove a Wire:

- Select the Pointer Tool.
- Click the left mouse button on either end of the wire.
- Hit <DEL>.

See Also:

Node Tool

See Examples:

WIRE.LGI

Node Tool •

The Node tool allows you to make your wiring more aesthetically pleasing. It also allows you to override the Unknown state with 0 or 1 and allows you to add pullups or pulldowns.

Rules:

- Only one device may drive the node.
- The node may drive as many devices as you wish.
- You may connect as many nodes together as you wish.
- It is best to start from the driving device and work towards the inputs.
- If you connect two standalone nodes together a direction will be set for you.

To Select a Node once connected:

- Click the left mouse button near the node device, but not on the black dot.

See Also:

Wiring Tool

See Examples:

WIRE.LGI

Options:

If Unknown (Default Pass): Pass, Force 1 or Force 0.
If Floating (Default Pass): Pass, Pullup or Pulldown.

Probe Tool 

The probe allows you to examine the circuit for debugging purposes.

To use the probe:

- Run the Simulator (it may also be paused).
- Select the probe by clicking on  from the toolbar.
- Now click on any Node (black dot) the probe will reflect its state.

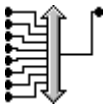
Rules:

- The probe may be used while the simulator is paused.
- You will get any information on an unconnected node.

The probe has 7 different displays:

	Probe is not connected.
0	Probe sees a 0 state.
1	Probe sees a 1 state.
U	Probe sees a Unknown state.
-	Probe sees a Pull Down state.
+	Probe sees a Pull Up state.
?	Probe sees a Float state.

If the state of the node changes while the probe is connected the probe will show its new state.

Bus Tool 

The Bus Tool is a control device. It is the only device that process multiple drivers (Tristate Gates). The Bus Device has no delay.

Rules:

- To pullup or pulldown the bus place a Node Tool just at the output.
- You should only drive the Bus Tool with Tristate Gates.

Behavior:

IF the bus sees 1 or more drivers THEN
    IF the bus sees more than one input driving THEN the output is U.
    IF the bus sees exactly one driver THEN the output is that drivers state.
otherwise
    IF the bus sees 1 or more pulldowns and no pullups THEN the output is -.
    IF the bus sees 1 or more pullups and no pulldowns THEN the output is +.
    IF the bus sees 1 or more pullups and 1 or more pulldowns THEN the output is ?

See Also:

Tristate Gate

See Examples:

TRISTATE.LGI

Options:

Width (Default 2): 2, 4 or 8.

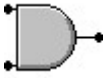
Changing Device Options

Most devices support options by double clicking on the device with the pointer device while in Draw mode.

Managing Pages 

You can have as many pages in your project as you wish. To change pages click on  (next page) or

 (previous page) from the toolbar. The last page with a device on it determines how many pages are in your project. To connect circuits across pages use the Signal Sender Device and Signal Receiver Device.

AND Gate 

The AND Gate performs a Logical AND of the input signals. The NAND device is one of the most simplest devices yet one of the most powerful. You can build almost anything from NAND gates.

Behavior (as an AND):

> IF all inputs are 1 THEN the output is 1
> IF any inputs are 0 THEN the output is 0
> otherwise the output is U

Behavior (as a NAND):

> IF all inputs are 1 THEN the output is 0
> IF any inputs are 0 THEN the output is 1
> otherwise the output is U

See Also:

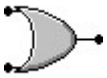
OR Gate

See Examples:

AND.LGI

Options:

Invert output (Default False): True (NAND) or False (AND).
Number of inputs (Default 2): 2, 3, or 4.

OR Gate 

The OR Gate performs a Logical OR of the input signals.

Behavior (as an OR):

> IF all inputs are 0 THEN the output is 0
> IF any inputs are 1 THEN the output is 1
> otherwise the output is U

Behavior (as a NOR):

> IF all inputs are 0 THEN the output is 1
> IF any inputs are 1 THEN the output is 0
> otherwise the output is U

See Also:

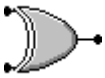
AND Gate

See Examples:

OR.LGI

Options:

Invert output (Default False): True (NOR) or False (OR).
Number of inputs (Default 2): 2, 3, or 4.

XOR Gate 

The XOR Gate performs a Logical Exclusive OR of the input signals.

Behavior (as an XOR):

> IF not exactly 1 input is 1 THEN the output is 0
> IF exactly 1 input is 1 THEN the output is 1

Behavior (as an XNOR):

> IF not exactly 1 input is 1 THEN the output is 1
> IF exactly 1 input is 1 THEN the output is 0

See Also:

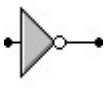
OR Gate
AND Gate

See Examples

XOR.LGI

Options:

Invert output (Default False): True (XNOR) or False (XOR).
Number of inputs.

Inverter/Buffer Gate 

The Inverter Gate performs a logical invert of the input signal. The Buffer Gate just passes the input through. The Buffer Gate may seem useless at first glance, but it isn't. It can be used to synchronize arrival of signals to a device. For example, lets say one input to a device goes through 2 AND gates and another only goes through 1. The path going through 1 gate will arrive 1 cycle too soon (this is not always a problem). To correct this, a dummy device (a buffer) can be added to the path to synchronize the two paths.

In real circuits Buffers are often used to increase the driving capability of signal (an amplifier).

Behavior (as an Inverter):

> IF the input is 0 THEN the output is 1
> IF the input is 1 THEN the output is 0
> otherwise the output is U

Behavior (as a Buffer):

> IF the input is 0 THEN the output is 0
> IF the input is 1 THEN the output is 1
> otherwise the output is U

See Also:

AND Gate
OR Gate

See Examples:

SPIN.LGI
INVERT.LGI

Options:

Invert output (Default False): True (Buffer) or False (Inverter).

Oscillator Device 

The Oscillator device is a powerful device that is often used to “drive” or “pump” your circuit. It is also often referred to as the Clock or Waveform device. It can be thought of as a switch (hooked to power source) that is being flipped on and off at a constant rate. You can adjust how long the switch is up relative to how long it is down. If the Oscillator device is too fast for your application you can use a Timer Device and control the speed in real time.

Behavior:

IF the output is 0 THEN
    IF “Low Cycles” have been executed THEN output is 1
    otherwise output is 0
IF the output is 1 THEN
    IF “High Cycles” have been executed THEN output is 0
    otherwise output is 1

See Also:

Counter Device
Timer Device

See Examples:

SPIN.LGI
OSCILAT.LGI

Options:

Cycles Low. (default 1): 1 to n
Cycles High. (default 1): 1 to n

LED Device 

The LED is a simple device that reports to you, visibly, what its input signal is doing. That is, of course, 0 (off), 1 (on) or U (unknown).

Behavior:

> IF the input is 1 THEN LED is on.
> IF the input is 0 THEN LED is off.
> IF the input is U THEN LED is red striped.

See Also:

8 Segment LED

See Examples:

LED.LGI
HI.LGI

Options:

Color (Default Red): Red, Yellow or Green

Select Device 

The select device is a 16 position rotary switch input device. It can be used as both an interactive input device or as an alternative to using Ground and Plus devices to hardwire some inputs by setting its initial state. For example is you wanted to lock the ALU device in compare mode set the Initial state to 4 and connect the Selector device to the ALU device function select lines.

Behavior:

When the user clicks on the select switch while the simulator is running it will:

IF Output n < F (hex) THEN Output a n+1.
IF Output n = F (hex) THEN Output a 0.

It will return to its initial state when the simulation is reset.

See Also:

Switch Device
Ground Device
Plus Device
Keypad Device

See Examples:

SELECT.LGI

Options:

Initial State (Default 0): 0 (hex) through F (hex).

Switch Device 

The switch device is a 2 state input device. It can be used as both an interactive input device or as an alternative to using Ground and Plus devices to hardwire some inputs by setting its initial state.

Behavior:

When the user clicks on the switch while the simulator is running it will:

Output a 1 IF the output is 0.
Output a 0 IF the output is 1.

If the switch is a momentary switch, it will return to its original state when the mouse is let go.
If the switch is not a momentary switch, it will not return to its original state when the mouse is let go.

See Also:

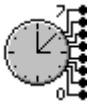
Oscillator Device
Ground Device
Plus Device

See Examples:

MEMORY.LGI

Options:

Momentary (Default False): True (Momentary) or False (Throw)
Normally on (Default False): True (On) or False (Off)

Clock Device 

The clock device is an input device that can present system clock data into your circuit. For example, your circuit could fire up every hour and do its thing and wait for another hour. The Clock device gives you access to the system clock as Seconds, Minutes, Hours, Days of Month, Months or Years. The clock device will only output one of these at a time. But you may have multiple clock devices to access multiple pieces of clock information.

Behavior:

When the corresponding clock information changes it will:

Output its value to the outputs.

See Also:

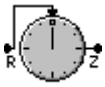
Oscillator Device

See Examples:

CLOCK.LGI

Options:

Type (Default Seconds): Seconds, Minutes, Hours, Days of Month, Months or Years

Timer Device 

The timer device is an input device that can be used to present real time events to your circuit. The timers pulse will not drift, it will fire exactly at n*Delay regardless of machine load or circuit complexity as long as each cycle completes before Delay time has expired.

Behavior:

IF the input “R” (Reset) is 0 THEN

    IF the “DELAY” duration has been expended:

    Output a 1 to the output and reset to the DELAY.
    Otherwise Output a 0 to the output.

IF the input “R” (Reset) is 1 THEN

    Output a 0 to the output and reset to the DELAY.

See Also:

Oscillator Device
Clock Device

See Examples:

TIMER.LGI

Options:

Delay in milliseconds (Default 1000): 1 to n.

Buzzer Device 

The Buzzer Device is an output device that drives the PC speaker. You can choose frequency and duration of the tone.

Behavior:

When a pulse (0 to 1) is seen on the input, the PC speaker will beep.
The simulator will stop until the Duration of time has expired.

See Also:

Sound Wave Device

See Examples:

BUZZER.LGI

Options:

Frequency in hertz (Default 1000): 20 to 20,000.
Duration in milliseconds (Default 20): 1 to n.

8 Segment LED Device 

The 8 segment LED is a device that reports to you, visibly, what its input signals are doing. It comes in two flavors 5 input or 8 input. The 5 input mode will translate the inputs such that the appropriate hexadecimal digit is displayed, in 8 input mode you control which segments are on or off.

Behavior (as 5 input):

When the input is 0 - 15 the LED device will show the digit 0 - F (hex).
If any input is Unknown a U will be displayed.

Behavior (as 8 input):

Each input is routed to the segment that will be On when its input is 1.

See Also:

LED Device

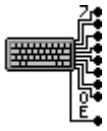
See Examples:

LED8.LGI
SPIN.LGI

Options:

Type (Default 5 Inputs): 5 Inputs or 8 Inputs.

Keyboard Device



The Keyboard device allows you to hook your computers keyboard into the circuit. Click on the Keyboard device while the simulator is running to enable it. Once enabled all keyboard input will go to this device. You may have more that one keyboard in your circuit, but only one can be enabled at any given time. You must disable any enabled keyboards before trying to enable another.

Behavior: (in ASCII mode)

The "E" output will pulse when a key is hit or auto repeated. The value of the outputs will reflect the ASCII translated code (e.g. Shift-A will send ASCII code for the letter "A").

Behavior: (in Virtual Key Code mode)

The "E" output reflects that a Key is down. The value of the outputs will reflect the untranslated Virtual Key Code (Shift-A will send a Virtual Key Code of the Shift Key and the Virtual Key Code of the A Key).

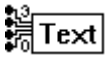
ASCII Display Device

See Examples:

KEYBOARD.LGI

Options:

ASCII Mode (Default False): True (ASCII Mode) or False (Virtual Key Code Mode).

Text 

The Text device is both a “static” documentation tool and an Output device (depending on style). As a “Single” style it allows you to label an input devices function etc. As a “Multiple” style it allows you to display any Text based on inputs from your circuit. The circuit can choose up to 16 different Strings based on its 4 inputs. See also Set Font which effects ALL Text devices.

Behavior (as a Single):

What you see is what you get.

Behavior (as a Multiple):

The line (starting at 0) within the specified file corresponding to the 4 inputs will be displayed.
IF any input is Unknown THEN the Text “Unknown” will be displayed.

See Also:

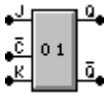
Fonts

See Examples:

TEXT.LGI
TEXT2.LGI

Options:

Text (Default “Text”): Text to be displayed.
Style (Default Single): Single or Multiple.
File (Default NUL): Path to Filename.TXT

FlipFlop Device 

The FlipFlop is the most primitive storage device (it remembers). You can think of it as a memory that can only hold 1 bit of data (0 or 1).

Master Slaves are internally made up from 2 Storage cells rather than just one. When the Clock is not asserted the transfer from the 1st stage to the 2nd is enabled. The JK is considered a Master Slave FlipFlop.

When the "Edge Triggered" clock is enabled the inputs are read only when the clock transitions from the unasserted state to the asserted state. Where as when the "Edge Triggered" clock is disabled (Level Sensitive) the inputs effect the device as long as the clock is in the asserted state.

Behavior: (Common to all non Master Slaves)

I0 Represents the internal state.
I0 is initialized to U

IF I0 is 1 THEN Q is 1 and Q-bar is 0
IF I0 is 0 THEN Q is 0 and Q-bar is 1
otherwise Q is U and Q-bar is U

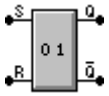
I0	Q	Q-BAR
0	0	1
1	1	0
U	U	U

Behavior: (Common to all Master Slaves)

I0 Represents the internal state of the 1st stage.
I1 Represents the internal state of the 2nd stage.
I0 and I1 are initialized to U

IF I1 is 1 THEN Q is 1 and Q-bar is 0
IF I1 is 0 THEN Q is 0 and Q-bar is 1
otherwise Q is U and Q-bar is U

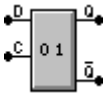
I1	Q	Q-BAR
0	0	1
1	1	0
U	U	U

Behavior: (as a Reset Set) 

IF S is 0 and R is 0 THEN I0 is unchanged.
IF S is 1 and R is 0 THEN I0 is set to 1.
IF S is 0 and R is 1 THEN I0 is set to 0.
IF S is 1 and R is 1 THEN I0 is set to U.

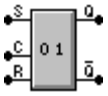
S	R	I0

0	0	I0
1	0	1
0	1	0
1	1	U

Behavior: (as a Data Latch) 

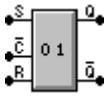
IF C is 0 THEN I0 is unchanged.
IF C is 1 THEN I0 is set to D.

C	D	I0
0	X	I0
1	X	D

Behavior: (as a Clocked Reset Set) 

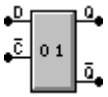
IF C is 0 THEN I0 is unchanged.
IF C is 1 and S is 0 and R is 0 THEN I0 is unchanged.
IF C is 1 and S is 1 and R is 0 THEN I0 is set to 1.
IF C is 1 and S is 0 and R is 1 THEN I0 is set to 0.
IF C is 1 and S is 1 and R is 1 THEN I0 is set to U.

C	S	R	I0
0	X	X	I0
1	0	0	I0
1	1	0	1
1	0	1	0
1	1	1	U

Behavior: (as a Master Slave Reset Set) 

IF C is 1 THEN I0 is unchanged and I1 is set to I0.
IF C is 0 and S is 0 and R is 0 THEN I0 is unchanged.
IF C is 0 and S is 1 and R is 0 THEN I0 is set to 1.
IF C is 0 and S is 0 and R is 1 THEN I0 is set to 0.
IF C is 0 and S is 1 and R is 1 THEN I0 is set to U.

C	S	R	I0	I1
1	X	X	I0	I0
0	0	0	I0	I1
0	1	0	1	I1
0	0	1	0	I1
0	1	1	U	I1

Behavior: (as a Master Slave Data Latch) 

IF C is 1 THEN I0 is unchanged and I1is set to I0.
IF C is 0 THEN I0 is set to D and I1 is unchanged.

C	D	I0	I1
1	X	I0	I0
0	X	D	I1

Behavior: (as a JK) 

IF C is 1 THEN I0 is unchanged and I1 is set to I0.
IF C is 0 and S is 0 and R is 0 THEN I0 is unchanged.
IF C is 0 and S is 1 and R is 0 THEN I0 is set to 1.
IF C is 0 and S is 0 and R is 1 THEN I0 is set to 0.
IF C is 0 and S is 1 and R is 1 THEN I0 is set to Not I0.

C	S	R	I0	I1
1	X	X	I0	I0
0	0	0	I0	I1
0	1	0	1	I1
0	0	1	0	I1
0	1	1	Not I0	I1

See Also:

Memory Device

See Examples:

FLIPFLOP.LGI
*FF.LGI

Options:

Style (Default Reset Set): Reset Set ... JK.
Edge triggered (Default False): True (Edge Triggered) or False (Level Sensitive).

Keypad Device 

The keypad device is a convenient input device that can generate one of 16 values.

Behavior:

When the user clicks on a key the corresponding key value will be presented to the Outputs.
IF a key is down (depressed with Left Mouse Button) E will Output 1.
otherwise E will Output 0.

The “E” Output should be used as an indicator to know when to use (consume) the output of the device.

See Also:

Keyboard Device

See Examples:

KEYPAD.LGI

Options:

Matrix outputs: (future)

Sound Wave Device 

The Sound Wave device is and output device that can play .WAV files.

Behavior:

When the input changes from 0 to 1 the wave file is played.
IF Wait is enabled the simulation will stop until the sound wave is completed.
Otherwise simulation will continue simultaneously.

Rules:

- Only one sound can be playing at any given time.
- If a sound wave is already playing when a new sound wave fires the previous sound will be terminated.
- This device requires that you have a Sound Card properly configured to play .WAV files.

See Also:

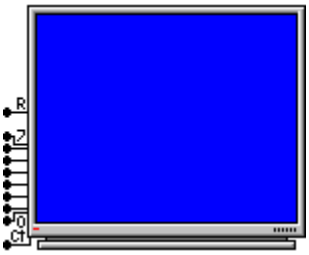
Buzzer Device

See Examples:

COUNTER.LGI

Options:

File (Default NUL): Path to Filename.WAV
Wait for completion (Default True): True (Wait) or False (Don't Wait)

ASCII Display Device 

The ASCII Display device is an Output device that emulates a simple terminal device. The display will hold 16 characters across of 8 rows down.

Behavior:

IF R is 1 the screen is reset.
Otherwise the device will perform the following:

When the clock input “C” changes from 0 to 1 the device will perform the following:

IF the inputs evaluate to <CR> (Hex 0D) the cursor is moved to the next line.
IF the inputs evaluate to <BkSp> (Hex 08) it will erase the previous character.
Otherwise the character is appended to the current line.
IF you send more than 16 characters to a line it will return to column 1 and erase the line.

See Also:

Keyboard Device

See Examples:

DISPLAY.LGI

Options:

None:

Ground Device 

The Ground Device is fixed output device that always generate a 0. You can use one ground to ground many inputs.

Behavior:

This device always outputs a 0.

See Also:

Plus Device
Switch Device

See Examples:

MEMORY.LGI

Options:

None:

Plus Device

The Plus Device is fixed output device that always generate a 1. You can use one Plus to turn on (also known as Pull-up) many inputs.

Behavior:

This device always outputs a 1.

See Also:

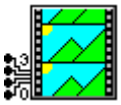
Ground Device
Switch Device

See Examples:

DEVICES.LGI

Options:

None:

Bitmap Device 

The Bitmap device is a very powerful Output device. It allows you to display any Bitmap based on inputs from your circuit. The circuit can choose up to 16 different bitmaps from its 4 inputs. You can also hook a counter to this device to sequence through a set of similar bitmaps to get an animation effect (e.g. A person running could show that your circuit is in a running state).

The Bitmap name must be of the form ????0.BMP and the simulator will assume ????1.BMP up to ????15.BMP could exist. You do not need all 16 bitmaps. The simulator will only look for bitmaps based on your inputs.

Example:

TRAFFICLIGHT0.BMP
TRAFFICLIGHT1.BMP
TRAFFICLIGHT2.BMP

And only use the input connections 0 and 1. But you must ground input connections 2 and 3.

The Bitmap Device can act as a "fixed" bitmap for the purposes of documentation or as a "Playground" for a Robot Device.

Rules:

- Each bitmap should be the same size but is not required.
- Bitmaps must be 16 color bitmaps.
- It is best to draw the first from scratch and copy it as a base for the rest.

Behavior:

Based on the 4 inputs the string 0 in ????0.BMP is replaced in the filename you specify and displayed. The simulator will have to read the files from disk the first time, but will cache them subsequently.

See Also:

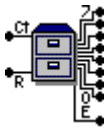
ASCII Display Device
Robot Device

See Examples:

BITMAP.LGI

Options:

Style (Default Single): Single or Multiple.
File (Default NUL): Path to ????0.BMP
Robot Playground (Default False): True (Playground) or False (Not a Playground).

Read File Device 

The Read File Device allows you to read in large storage amounts.

Behavior:

When the clock input “C” changes from 0 to 1 a character or ASCII Hex line (depending on format) is read from the file and fired to the outputs. You can reset the Read File Device at any time with the input “R” set to 1. Reset means it will start reading from the beginning of the file. If end of file is reached the output “E” will be driven to 1 until a reset.

See Also:

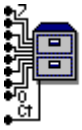
Write File Device
File Formats

See Examples:

FILECOPY.LGI

Options:

File (Default NUL): Path to Filename.Ext
Format (Default Binary): Binary or ASCII Hex

Write File Device 

The Write File Device allows you to write out large storage amounts.

Behavior:

When the clock input “C” changes from 0 to 1 the data on the inputs is written to the file. The file remains open until a simulator reset.

See Also:

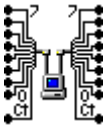
Read File Device
File Formats

See Examples:

FILECOPY.LGI

Options:

File (Default NUL): Path to Filename.Ext
Format (Default Binary): Binary or ASCII Hex

Network Device 

The Network Device is both an input and output device. It allows you to connect your logic simulation to one or more other logic simulations (and some other TCP/IP socket based applications) over a network.

Rules:

- When you wish the Network Device to initiate a connection, select "Make a Call to" and fill in to whom.
- When you wish the Network Device to accept a connection, select "Listen for a Call".
- You can have as many connections (Network Devices) as your Networking software will support.
- Each Network Device must use a unique port.
- The "Listen for a Call" machine must be running before the "Make a Call to" machine.
- A single machine can "Listen for a Call" on one port and "Make a Call to" on another port.
- A single machine can have multiple logic simulations running and communicate with one another.

Behavior:

Each time you start the simulator a call is made to the specified machine (or listened for) on the specified port. Once a connection is made, the Icon will change from a grayed out remote computer to a colored one. If the Icon reverts back to grayed, it indicates one of the remote machines has stopped their simulator or the network has failed.

When the clock input "C" changes from 0 to 1 the data on the inputs is written to the remote machine.

When data arrives from the remote machine the outputs will be driven with that data and the clock output "C" will change from 0 to 1. On the following cycle the output clock "C" will transition back to 0 (it can be used as an edge trigger for another device).

If more than one byte of data arrives within a single simulation cycle, each byte will be queued to subsequent cycles in the order received.

See Also:

Networking Setup Basics

See Examples:

NETWORK.LGI

Options:

Port (Default 4096): 0 to n
Direction (Default Make a Call to): Make a Call to or Listen for a Call

Networking Setup Basics

Networking allows MMLogic to communicate with the rest of the world. There are numerous forms of networking and MMLogic uses just one them. This form is known as TCP/IP sockets (this is primarily what is used to build the Internet). MMLogic can receive data from other computers or send data to other computers. The other computer does not need to be running MMLogic although it will be very likely.

MMLogic Network Device assumes that the TCP/IP protocol is available and that it is available through what is known as 32bit WinSock (WSOCK32.DLL). All TCP/IP protocols have to be explicitly installed and setup on all platforms. Some TCP/IP stacks available at no cost are:

32bit TCP/IP protocol for Windows 95 (included).
32bit TCP/IP protocol for Windows NT (included).
16bit TCP/IP protocol for Windows For WorkGroups 3.11 from Microsoft (public domain).
16bit TCP/IP protocol for Windows 3.x known as Trumpet (public domain).
32bit TCP/IP protocol for Windows ??? known as Trumpet (public domain).

Win32s includes a WSOCK32.DLL that maps to any available 16bit WinSock (WinSock.DLL) [this is not confirmed as yet].

TCP/IP networks can run over a wide variety of devices, a modem running PPP, a LapLink cable running direct connect, Twisted Pair, or etc. Compuserve Information Service (CIS), Microsoft Network (MSN), America OnLine and private Internet Service Providers (ISPs) all offer TCP/IP access to the rest of the world. What this means is that your friend can be on the west cost and you can be on the east coast and both run MMLogic and operate on each others circuits or build games that you can both participate in. Assuming you can find one another over the Internet.

Tools of the trade (outside of MMLogic):

DNS (Domain Name Service): DNS is an integral part of using TCP/IP protocols. TCP/IP in itself only understands machine addresses that look something like 21.100.200.5 (note each number field must be less than 255). But numbers have no meaning and are hard to remember for us humans, so machines are assigned names. DNS is basically machines on the network that maintain databases of these mappings of names to numbers. DNS machines talk to each other to share information. MMLogic can use DNS names or TCP/IP addresses when trying to make a connection. DNS is not required to setup a small TCP/IP network keep reading.

PING: Ping is a simple program (usually command line) used as a simple test of network connectivity. Most TCP/IP implementations include a PING program. To use it just enter "PING remotenode" where remotenode is the machine you'd like to test connectivity to. PING will tell if it can reach that machine and, if it can, how long it will take. Until PING works don't bother trying any other TCP/IP based software (including MMLogic Network Devices). PING will accept both forms of the address (the DNS name or the TCP/IP address). If the PING fails with the name then try the number (if you know it). If the number works but the name does not, it usually indicates a DNS problem.

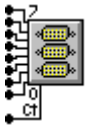
No DNS: If there is no DNS server, you can use what is known as a HOSTS file (usually kept in C:\Windows). Microsoft frequently installs a sample HOSTS file called HOSTS.SAM (copy HOSTS.SAM to HOSTS. and edit it). It is simply a database for mapping name addresses to numbers. If using a HOSTS file then each machine on the network should use a copy of the same HOSTS file with the name to mapping of all the machines in the network. You can

have both DNS and a HOSTS file, usually the HOSTS file is checked before DNS is attempted.

Ports and Services: Services is to Ports as DNS is to Network Addresses. Services map "Service Names" like (TELNET, FTP, POPMAIL) to port numbers. You may find a SERVICES file in C:\Windows and you can browse it if you like. MMLogic does not support the Service name to port number, it only uses Socket port numbers. But if you wish to connect to a service such as TELNET or act as a TELNET server then just look it up in the services file. In general, port numbers under 1024 are reserved for things such as TELNET and FTP. It is best to use socket numbers greater than 1024 and less than 5000.

Networks: If it is a private network under your total control you can do what ever you want. If they are all Windows PC's then you'll probably have to do without a DNS server (and use a HOSTS file). Generally DNS servers are UNIX or NT machines. If your part of a larger network you should contact your network administrators. If your already networked using Netware, it is common to run both TCP/IP and Netware at the same time.

Internet Service Providers: Some ISP's do not give you a name, in this case you can add your own name to the HOSTS file (on both machines) or just use the TCP/IP address. Many ISP's assign your TCP/IP address "on-the-fly" each time when you login. In Windows 95 you can run WINIPCFG.EXE to find out what your address is for your current connection. Some ISP's completely "wrap" your network access up into one application that gives all the services you need and might not offer WinSock access. When asking ISP's for help, tell them that you want to run something like NetScape ®. If NetScape works then MMLogic's Network Device should also work (calling anyway). Allowing you to listen for a call is more complex, the other application needs to be able to find you by name or by address.

Port Out Device 

The Port Out Device is a very powerful output device. It allows you to connect to real devices in your computer such as the COM, LPT or other special hardware. You can connect 2 Computers through a LapLink ® cable.

Behavior:

When the clock input "C" changes from 0 to 1 the data on the inputs is written to the port.

Warning: Writing to an improper address can cause your system to crash.
Note: this device does not work on Windows NT.

See Also:

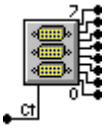
Port In Device.

See Examples:

PORTIN.LGI
PORTLED4.LGI

Options:

Port (Default 0): 0 to n
Initial state (Default 0): 0 to n

Port In Device 

The Port In Device is a very powerful input device. It allows you to connect to real devices in your computer such as the COM, LPT or other special hardware. You can connect 2 Computers through a LapLink ® cable.

Behavior:

When the clock input "C" changes from 0 to 1 a byte is read from the port and fired to the outputs.

Warning: Even reading an improper address can cause your system to crash.
Note: this device does not work on Windows NT.

See Also:

Port Out Device.

See Examples:

PORTOUT.LGI
PORTLED4.LGI

Options:

Port (Default 0): 0 to n
Initial state (Default 0): 0 to n

Signal Sender Device 

The Signal Sender Device is used to send your signal to other locations (or pages) without using a wire. It will be received by a Signal Receiver Device, of the same name, with no simulator delay; it will act as a solid wire. A Signal Sender Device may only drive one Signal Receiver. All Signal Sender/Receiver pairs must be uniquely named by you (checks will be performed when you start the simulator). To send a signal, without delay, to multiple pages, a driving device may drive as many Signal Senders as you wish. Signal names are case sensitive (.e.g. "AA" will NOT match with "aa").

Behavior:

When the driving wire fires a state into the input of the Signal Sender, the Output wire of the corresponding Signal receiver will drive that same state, all within the same cycle.

See Also:

Signal Receiver Device

See Examples:

SIGNAL.LGI

Options:

Signal Name (Default "None"): Any String

Signal Receiver Device 

The Signal Receiver Device is used to receive your signal from other locations (or pages) without using a wire. It will be sent by a Signal Sender Device, of the same name, with no simulator delay; it will act as a solid wire. A Signal Receiver Device may only be driven by one Signal Sender. All Signal Sender/Receiver pairs must be uniquely named by you (checks will be performed when you start the simulator. Signal names are case sensitive (e.g. "AA" will NOT match with "aa").

Behavior:

The Output of the Signal Receiver will fire with the state driven onto the input wire of the corresponding Signal Sender device, all within the same cycle.

See Also:

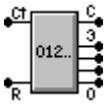
Signal Sender Device

See Examples:

SIGNAL.LGI
PAGES.LGI

Options:

Signal Name (Default "None"): Any String

Counter Device 

The Counter Device is a simple 4 bit counter. You can gang counters together to build larger counters.

Behavior:

When the clock input changes from 0 to 1 the counter increments by 1. If the counter has reached its limit (15 decimal) it pulses in the Carry Out "C" signal. The Carry out signal can be used as the Clock to trigger another counter. So you can gang together as many counters as you like to build larger counters. The 4 outputs represent the current value of the counter. You can reset the counter at any time with the "R" set to 1.

See also:

Random

See Examples:

COUNTER.LGI
COUNTER2.LGI

Options:

Initial Value (Default 0): (Future)

Pause Simulator Device 

The Pause Simulator Device is a unique control device. It allows you to pause the simulator based on state in your circuit.

Behavior:

When the input is 1 the simulation is Paused. You can continue the simulation as long as the 1 condition is removed.

See Also:

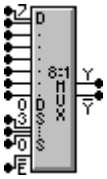
Modes of Operation
Probe Tool

See Examples:

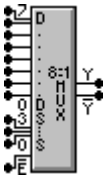
PAUSE.LGI
PAUSE2.LGI

Options:

None:

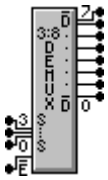
Multiplexer (Mux) Device 

The Multiplexer/DeMultiplexer devices are standard control devices. They allow you to select one of the multiple sources onto a single output (Mux) or select a single source onto one of multiple outputs (DeMux). The Multiplexer is also referred to as a data selector and the DeMultiplexer is also referred to as a Decoder.

Behavior (as Mux): 

IF E is 1 THEN Y is 0 and Y-Bar is 1.
IF E is 0 THEN Y is Dn and Y-Bar is Not Dn.
Where n = Select lines.

E	S2	S1	S0	Y	Y-BAR
1	X	X	X	0	1
0	0	0	0	D0	Not D0
0	0	0	1	D1	Not D1
0	0	1	0	D2	Not D2
0	0	1	1	D3	Not D3
0	1	0	0	D4	Not D4
0	1	0	1	D5	Not D5
0	1	1	0	D6	Not D6
0	1	1	1	D7	Not D7

Behavior (as DeMux): 

IF E is 1 THEN D0 - D7 is 1.
IF E is 0 THEN D0 - D7 is 1 Except Dn is 0.
Where n = Select lines.

E	S 2	S 1	S0	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
1	X	X	X	1	1	1	1	1	1	1	1

0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	0	1
0	0	1	0	1	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	0	1	1	1
0	1	0	0	1	1	1	0	1	1	1	1
0	1	0	1	1	1	0	1	1	1	1	1
0	1	1	0	1	0	1	1	1	1	1	1
0	1	1	1	0	1	1	1	1	1	1	1

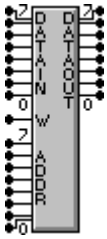
See Also:

See Examples:

MUXES8.LGI
MUXES4.LGI
MUXES2.LGI

Options:

Style (Default 8 to 1 Mux): 8 to 1 Mux ... 2 to 1 DeMux.

Memory Device 

The Memory Device is Control device. It can store up to 256 bytes of data. You can increase the data width by ganging up more memory devices all using the same address. The memory can be optionally initialized with data from a file (upon each reset of simulator). If it is not initialized with a file then it is initialized with all Unknown (U).

Behavior:

IF the Write Enable input "W" is 1 THEN
    The data inputs are written to the address selected on the address inputs.
    IF any address line is Unknown THEN the entire memory is written with Unknown (U).
    IF any data input line is Unknown THEN it will be read as Unknown (U).
IF the Write Enable input "W" is 0 THEN
    The data outputs are driven with what is stored in memory specified by the address inputs.
    IF any address line is Unknown THEN the data outputs are driven with an Unknown (U).

See Also:

FlipFlop Device
File Formats

See Examples:

MEMORY.LGI

Options:

Filename (Default NUL): Path to Filename.Ext

File Formats

The simulator supports 2 file formats for several devices (e.g. Memory, Read File and Write File). The 2 formats are Binary mode and ASCII Hex mode.

ASCII Hex mode:

nn
nn
nn
.
.
.
nn

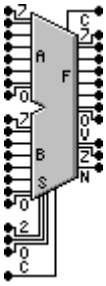
Where nn is 2 Hexadecimal digit's (presented as ASCII text followed by a <CR>), the file can contain any number of lines (Memory device only reads 256). Each line represents 8 bits worth of data. For the memory device the first line will go into address 0, the second line into address 1 and so on.

Example:

FF
AA
00
12

Binary Mode:

In binary mode the files are read or written raw. Each byte in the file is read "as is" including line control characters such as <CR> and <LF>. Any file can be read or written in binary mode and is the most efficient with respect to size of the files and speed of reading or writing.



Arithmetic Logic Unit (ALU) device

The ALU is an 8 bit control unit. It can perform Addition, Subtraction, Multiplication, Division, Shifting and Comparing.

Behavior (Common to all functions):

IF the result is 0 THEN Z is 1 otherwise Z is 1
IF the result overflows or underflows THEN V is 1 otherwise V is 0
N is 0 (future)

Behavior (IF S is 0):

Output to F is A + (B + Cin)
IF the result overflows THEN Cout is 1 otherwise Cout is 0

Behavior (IF S is 1):

Output to F is A - (B + Cin)
IF the result underflows THEN Cout is 1 otherwise Cout is 0

Behavior (IF S is 2):

Output to F is A * B
Cout is 0

Behavior (IF S is 3):

Output to F is A / B
Cout is 0

Behavior (IF S is 4):

Output to F is A = B (output is 0 for False and 1 for True)
Cout is 0

Behavior (IF S is 5):

Output to F is A < B (output is 0 for False and 1 for True)
Cout is 0

Behavior (IF S is 6):

Output to F is A << B (A left shifted by B) fill with Cin
Cout is the last (upper) bit shifted out of A.

Behavior (IF S is 7):

Output to F is A >> B (A right shifted by B) fill with Cin
Cout is the last (lower) bit shifted out of A.

See Also:

Counter Device

See Examples:

ALU.LGI

Options:

None:

Bit Bucket Device 

The Bit Bucket Device is simplest output device. It does absolutely nothing. If it does nothing why have it? The reason is that it documents that you are not using an output versus you just forgot to hook something up.

Behavior:

When any input is 1 it does nothing.
When any input is 0 it does nothing.

See also:

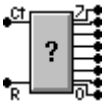
Ground Device.
Plus Device.

See Examples:

BITBUCK.LGI
MEMORY.LGI

Options:

None:

Random Device 

The Random Device is unique input device, it is an 8 bit Random generator. It could be used for testing or part of game circuit or anything you like.

Behavior:

When the clock input "C" changes from 0 to 1 a random number is fired to the outputs. You can reset the Random Device at any time with the input "R" set to 1. Reset means it will return to the first Random number generated (the seed).

See Also:

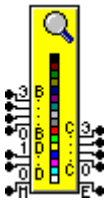
Counter Device.

See Examples:

RANDOM.LGI
HI.LGI

Options:

Seed (Default 0): 0 (make the seed itself random using system time) to n.

Robot Device 

The Robot device is a very basic machine. You have to build the hardware to give it any intelligence. It is also linked to the Bitmap Device. The Bitmap Device acts as the Robots "Playground" you must enable a Bitmap Device to be a Playground (see the properties of the Bitmap Device).

The Robot Bitmap name must be of the form ????0.BMP and the simulator will assume ????1.BMP up to ????15.BMP could exist. You do not need all 16 bitmaps. You may link bitmaps to direction simply by driving the direction inputs with the same value as the low bits of the bitmaps inputs.

Example:

CAR0.BMP
CAR1.BMP
CAR2.BMP
CAR3.BMP

The Color Dark Yellow (Red=128, Green=128, Blue=0) is reserved as a transparent color. Anywhere this color is used (in robot bitmaps only) it will show the "Playground" bitmap background through (i.e. robots do not have to be square you could have a donut shaped robot).

Rules:

- Each bitmap should be the same size but is not required.
- Bitmaps must be 16 color bitmaps.
- It is best to draw the first from scratch and copy as a base for the rest.
- You must have a Bitmap Device playground.
- You can have up to 16 robots play in a playground.

Behavior (Common to all styles):

When the clock input "M" (move) changes from 0 to 1 the robot will perform the following:

The "Robot" is the driven one step in the direction specified by the D inputs.
A bitmap is displayed based on the B inputs.

IF D is 0 the robot moves East
IF D is 1 the robot moves West
IF D is 2 the robot moves South
IF D is 3 the robot moves North

IF the "Hotspot" moves off the "playground" E is 1
otherwise E is 0

When the robot changes direction it will "rotate" around the "Hotspot".
Each time the simulator starts the "Hotspot" will be placed at the "Initial Position" on the playground.

Each time the simulator starts the robots look for the “nearest” playground to play in. Nearest means it will look on the same page as the robot is on then it will look on other pages. Multiple robots can play on the same playground. Multiple robots can be “stacked” on top of one another to combine capabilities (i.e. a robot that has 2 “eyes” and 2 “pens”).

Behavior (Style Seeing):

The C outputs describe the Color under the “Hotspot” of the robot.

Behavior (Style Drawing):

IF P is 1 draw a dot at the “Hotspot” of color based on inputs C.
otherwise don’t draw.

See Also:

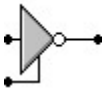
Bitmap Device

See Examples:

ROBOT1.LGI
ROBOT2.LGI
ROBOT3.LGI

Options:

File (Default NUL): Path to ????0.BMP
Hotspot (Default Center): Center … Custom.
Initial Position (Default Center): Center .. Custom.
Style (Default Seeing): Seeing or Drawing.

Tristate Gate 

The Tristate Gate is a powerful control device. It is the only device that generates floating states (+, - and ?). A Tristate gate will normally be connected to the Bus device. The purpose of the Tristate Gate is to allow multiple sources (one at a time) to drive an input.

Behavior (as a Tristate Inverter):

IF the E (enable) input is 1 (0 if bubbled)
IF the input is 0 THEN the output is 1
    IF the input is 1 THEN the output is 0
otherwise the output is U
otherwise the output is ?

Behavior (as a Tristate Buffer):

IF the E (enable) input is 1 (0 if bubbled)
IF the input is 0 THEN the output is 0
    IF the input is 1 THEN the output is 1
    otherwise the output is U
otherwise the output is ?

See Also:

Inverter Gate
Bus Tool

See Examples:

TRISTATE.LGI

Options:

Invert output (Default False): True (Tristate Buffer) or False (Tristate Inverter).
Assert LO (Default False): True (Assert LO) or False (Assert HI).