

# DLS

## **The digital logic simulator game**

Sandbox mode manual

v0.14.0

2017-04-11

# Table of Contents

<b>Introduction</b>	<b>3</b>
<b>1. User Interface</b>	<b>4</b>
1.1 Main toolbar	4
1.2 Simulation/Circuit info	5
1.3 Gate/component toolbar	6
1.3.1 Component library	7
1.3.2 Notes	8
1.3.3 Scripted components	9
1.3.4 Numeric Input	9
1.3.5 Numeric Output	10
1.3.6 Push button	10
1.3.7 7-segment display	10
1.3.8 Wire mergers and splitters	11
1.3.9 Bus	11
1.3.10 ROM	12
1.3.11 Static RAM	13
1.3.12 Clocks	13
1.3.13 Switches	14
1.3.14 LED Matrix	14
1.3.15 Pixel display	15
1.3.16 Tunnels	15
1.3.17 Basic logic gates	16
1.4 Testbench editor	17
1.5 Console	17
1.6 Logic Analyzer	18
1.7 Wiring	18
<b>2. Components</b>	<b>19</b>
<b>3. Scripted components</b>	<b>20</b>
3.1 The init() function	20
3.2 The simulate() function	20
<b>4. Testbenches</b>	<b>22</b>
<b>A. Build-in Gate/Component Information</b>	<b>24</b>
<b>B. Release History</b>	<b>25</b>

# Introduction

Welcome reader! Since you are reading this I assume you already know what DLS is about and you are searching for information on some part of the UI. So I'll keep this as short as possible.

DLS is a “**time-driven event-based multi-delay 3-value digital logic simulator**” (if there's such classification). Let's try to break it up in pieces to better understand how it works:

- **Time-driven** means that the circuit time is advanced forward based on a user-specified target speed, which is measured in circuit nanoseconds per real second (ns/s). The simulation **always advances** to a new state if there are pending signals in the circuit's queue. This means that even **unstable** or **asynchronous** circuits can be correctly simulated.
- **Event-based** means that **a gate is simulated only if one of its inputs changes value**. Otherwise the previous output is considered valid and used as input to all connected gates/components.
- **Multi-delay** means that each build-in gate/component type has its own **propagation delay** which is always an integer greater than or equal to 1.
- **3-value** means that in addition to the 0 and 1 logic levels, there's an extra logic value (U for Undefined) which is used as either Z (high impedance) or X (undefined) signals, depending on its origin.

You can change the target simulation speed through the Simulation/Circuit info popup (see [section 1.2](#)). You can go as low as 1ns/s, which can be used to visually debug signal propagations through a circuit, and as high as 1s/s, in case you want to let it run as fast as possible. Note that the target simulation speed might not be achieved so the real simulation speed will differ. It depends on the complexity of your circuit, but it shouldn't affect the final results.

What might affect your results is the selected clock speed, in case your circuit is a clocked (sequential) circuit. You have to take care to use a slow enough clock to allow your signals to arrive and stored at the destination registers before the next clock tick is triggered. Clocks can go as slow as 1Hz and as fast as 500MHz. Note that the clock speed is measured in simulation time and not real time.

E.g. a 500MHz clock simulated at a target speed of 1ns/s will switch levels every second. The same is true for a 1MHz clock with a target speed of 500ns/s. On the other hand, a 10MHz clock on a 100us/s simulation will switch levels 2000 times during 1 real second (assuming your circuit is simple enough to achieve the target simulation speed).

That's all you need to know about how the simulator works. Now go and make something great :)

# 1. User Interface

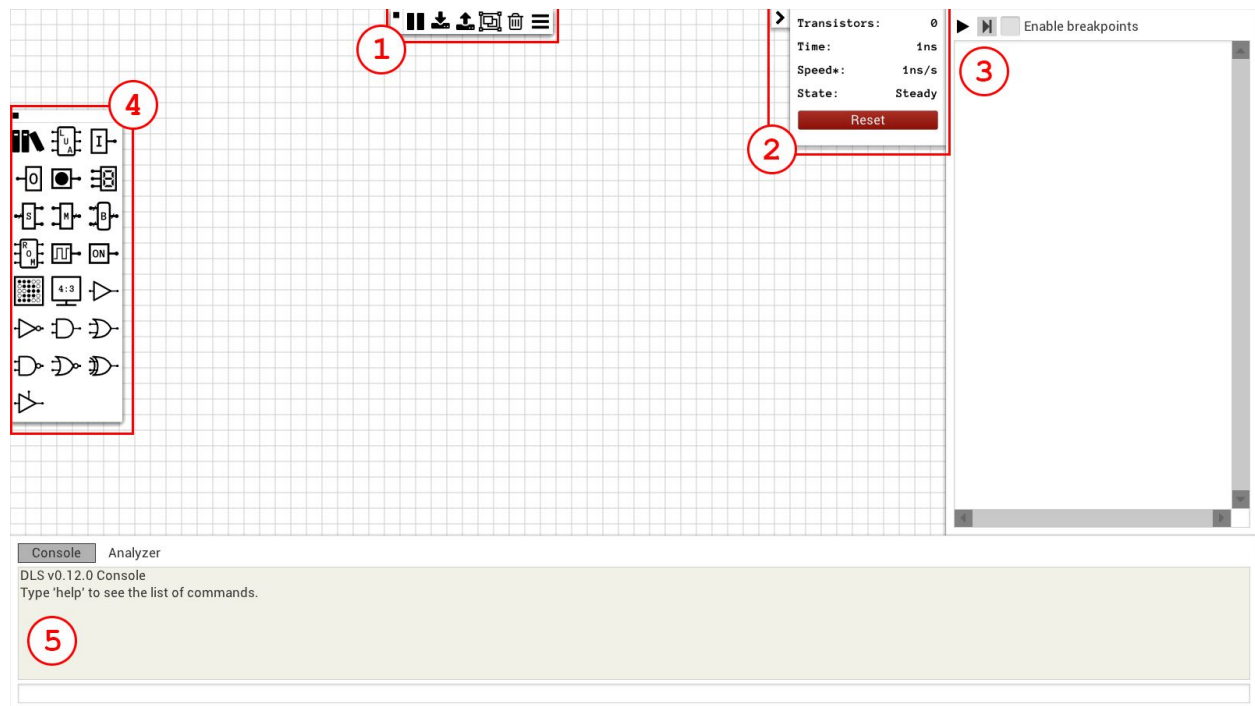


Fig. 1: Main user interface

Figure 1 shows DLS' main user interface (UI), with the testbench editor and the console opened.

1. Main toolbar
2. Simulation/Circuit info
3. Testbench editor (press F2 to open)
4. Gate/component toolbar
5. Console/Logic Analyzer window (press F1 to open)

In the following sections, the various parts of the UI will be described.

## 1.1 Main toolbar



Fig. 2: Main toolbar

Figure 2 shows the main toolbar. From left to right:

- **Start/Stop simulation:** When enabled (pause symbol is visible) every time the circuit changes (e.g. input values change, wires added or removed, etc.), outputs are

recalculated. When disabled (play symbol is visible) no simulation is performed. For example, stopping the simulation is an easy way to simultaneously stop all clocks in a circuit (in case you have more than one), or change multiple input values at once.

- **Save schematic:** Save the current schematic on disk (\*).
- **Load schematic:** Load a previously saved schematic from disk (\*).
- **Componentize circuit:** Create a new component from the current schematic. For details on how components work see [chapter 2](#).
- **Clear schematic:** Clears the current schematic.
- **Menu:** Show the pause menu from which you can return to the main menu or exit DLS.

(\*) Depending on your platform, schematics are saved at the following locations:

- Windows: %USERPROFILE%\Documents\DLS\schematics (your Documents folder)
- Linux: <installation folder>/schematics, where <installation folder> is the folder inside which you extracted DLS.
- OS X: ~/Documents/DLS/schematics (your Documents folder).

## 1.2 Simulation/Circuit info

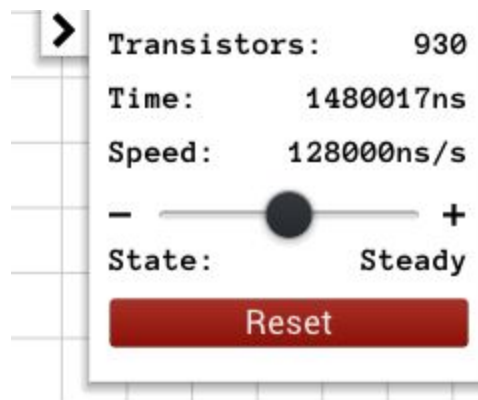


Fig. 3: Simulation/Circuit info

Figure 3 shows the simulation and circuit information popup. You can toggle the popup by pressing the arrow button on its left side.

- **Transistors** shows the current number of equivalent CMOS transistors in the circuit.
- **Time** is the current simulation time of the circuit. Clicking on it changes the units, cycling between ns, us, ms and seconds.
- **Speed** displays either the current (real) simulation speed of the circuit, or the desired/target speed. Clicking on it will switch between the two. When showing the target simulation speed, you can use the slider to change it. You can go as low as 1ns/s and as high as 1s/s.
- **State** shows the current state of the circuit. It will either be **Steady** or **Transient**. Steady means that the circuit has finished processing its event queue and all outputs will remain unchanged until an input changes value. Transient means that there are more events

queued to be processed at a later timestep and outputs will probably change once the circuit returns to steady state.

- By using the **Reset** button the circuit's event queue is cleared and time is reset to 0.

### 1.3 Gate/component toolbar

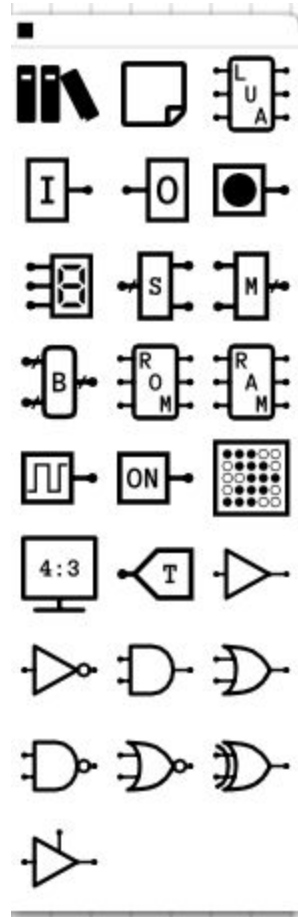


Fig. 4: Gate/component toolbar

Figure 4 shows the gate/component toolbar (from now on just gate toolbar). From left to right, top to bottom:

- [Component library](#)
- [Note](#)
- [Scripted component](#)
- [Numeric input](#)
- [Numeric output](#)
- [Push button](#)
- [7-segment display](#)
- [Wire splitter](#)
- [Wire merger](#)

- [Bus](#)
- [ROM](#)
- [Static RAM](#)
- [Clock](#)
- [Switch](#)
- [LED Matrix](#)
- [Pixel display](#)
- [Tunnels](#)
- [Buffer](#)
- [Inverter \(NOT\)](#)
- [AND](#)
- [OR](#)
- [NAND](#)
- [NOR](#)
- [XOR](#)
- [Tri-state buffer](#)

### 1.3.1 Component library

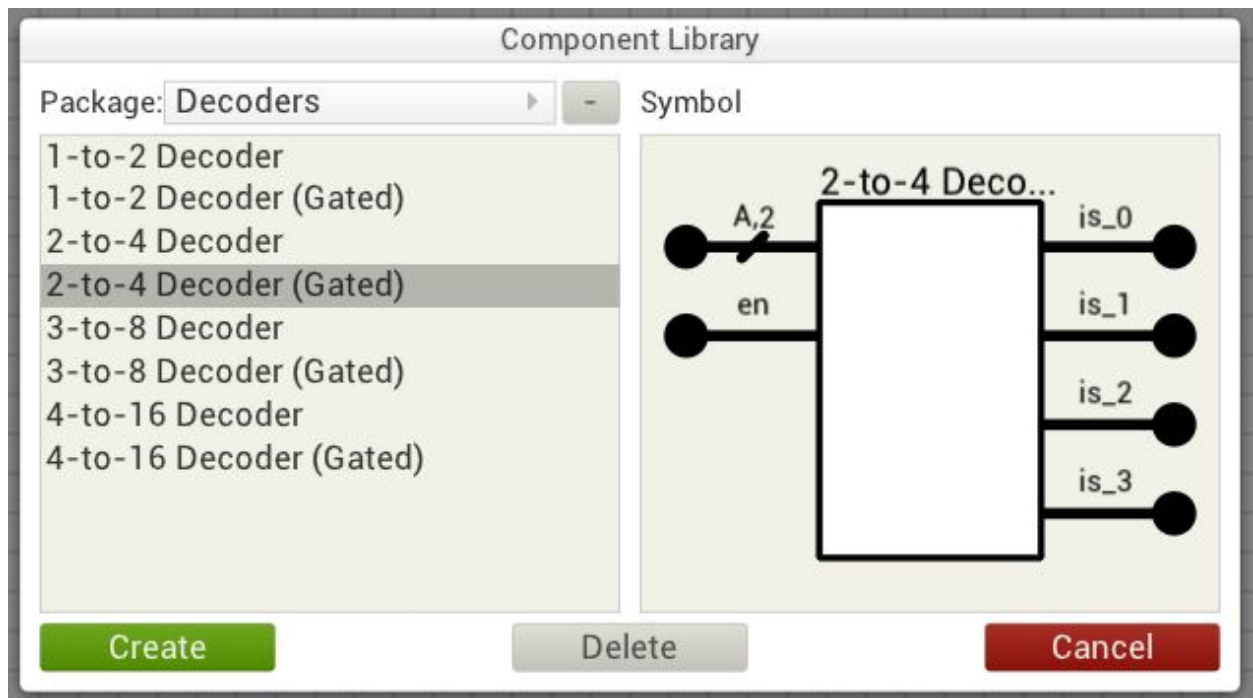


Fig. 5: The component library

Figure 5 shows your component library. Components are separated into packages. When starting DLS for the first time, a default empty package is already created for you, which you cannot delete.

The list on the left side shows the components from the selected package. On the right side, a preview of the selected component is shown.

Selecting Create will create a new instance of the selected component, which you can place on the grid. The Delete button removes the selected component from the library and Cancel closes the dialog.

Right clicking on a component already placed in the circuit brings up extra options. One of them is Inspect which lets you inspect the component's circuit.

**Note:** Currently there's no way to edit a component. If you want to change the internals of a component you have to re-componentize the circuit from the Main toolbar, once you've made your changes, and manually replace the component in the parent circuit.

### 1.3.2 Notes

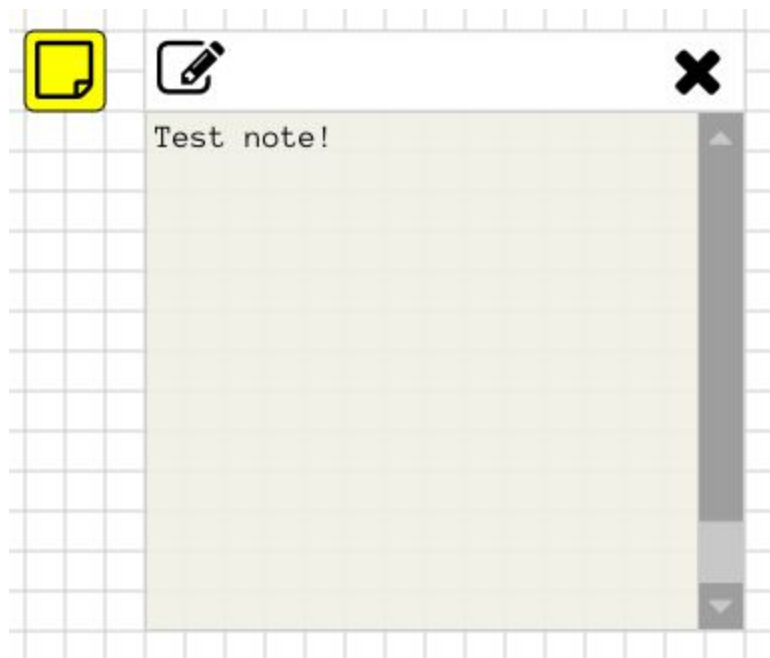


Fig 6: Notes in the schematic

You can add notes to your schematics the same way you add gates and components. Initially, the new note will be closed (yellow symbol in fig. 6). Left clicking on the note will expand it (as seen on the right in figure 6).

The expanded note viewer has fixed dimensions but if your text is longer than what the window can show, a vertical scrollbar will appear.

When saving your schematics, the state of the note (opened or closed) is preserved so you can keep important notes opened and the rest closed.

Finally, notes are drawn last when rendering the schematic, so they will always appear on top of the circuit (gates/components/wires) independent on the order they were created.



### 1.3.3 Scripted components

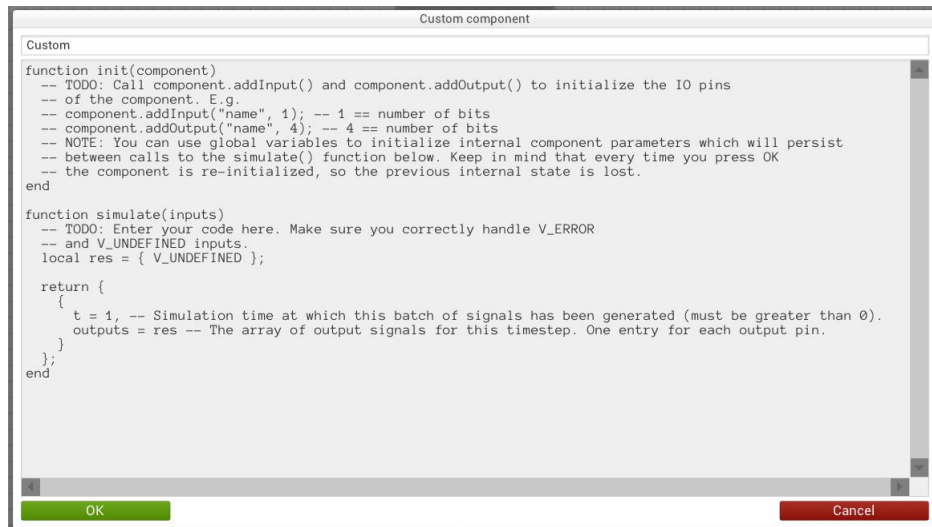


Fig. 7: Scripted component dialog

Figure 7 shows the scripted component editor. On the top there's a single line edit box for the name of the component. Below it's the editor for the component's code. Clicking OK creates a new component and Cancel closes the dialog. For details on how scripted components work see [chapter 3](#).

### 1.3.4 Numeric Input

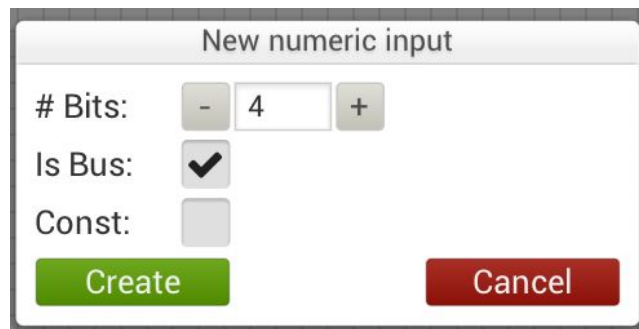


Fig. 8: Numeric input configuration

Figure 8 shows the configuration dialog for a new numeric input. You can open this dialog by right clicking on the toolbar button. The first parameter is the width of the input port in bits. If the **Is Bus** checkbox is checked, the new input port will have 1 wide pin. Otherwise, it'll have multiple 1-bit pins. **Const** means that the input port is a constant value and it won't be part of the generated component. Constant inputs can be used to set signals to predefined values.

You can interact with the numeric input port by either left clicking on its value (increases the value by 1), or by using the mouse wheel while the cursor is over the value (scrolling up or

down increases or decreases the value, respectively). You can also right click on it and select the “Edit value” from its context menu.

Finally, numeric inputs have multiple display formats. Depending on the width of the input, these are unsigned (u), signed (s), hex (h) and ASCII (a) ('a' is available only for 8-bit inputs). You can cycle through the various formats by clicking on the small letter at the top left corner of the input port.

### 1.3.5 Numeric Output

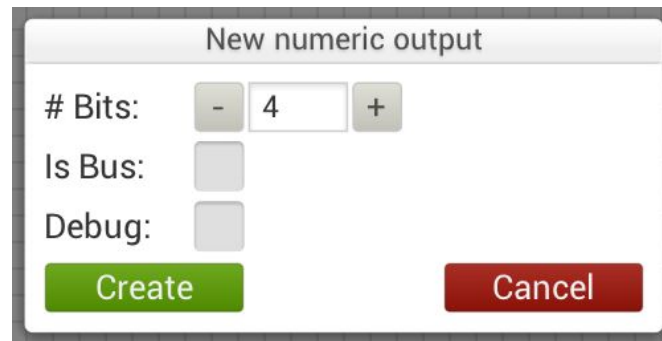


Fig 9: Numeric output configuration

Similar to the numeric input port is the numeric output port (figure 9). The only difference is that instead of being **Const**, output ports are marked as **Debug**. When **Debug** is checked the output port won't be part of the generated component. Debug numeric output ports can be used to observe intermediate signals using the [Logic Analyzer](#).

### 1.3.6 Push button

Push buttons are 1-bit input ports. Their only configurable parameter is whether they will be Const or not. When placed in a schematic, you can interact with the push button using your mouse. As long as the button is pressed, it's value will be 1. Otherwise, it'll automatically revert to 0.

### 1.3.7 7-segment display

7-segment displays are another type of output port. They have 8 1-bit pins (1 for each segment plus a pin for the dot). Whenever a pin is HIGH, the corresponding segment turns on. Otherwise it remains off. Figure 10 shows an instance of a 7-segment display.

As an output port it can be configured by right clicking on the toolbar button. The only configuration option is whether the port will be a debug port or not. Note that even if the 7-segment display is marked as Debug, its value will not appear in the Logic Analyzer.

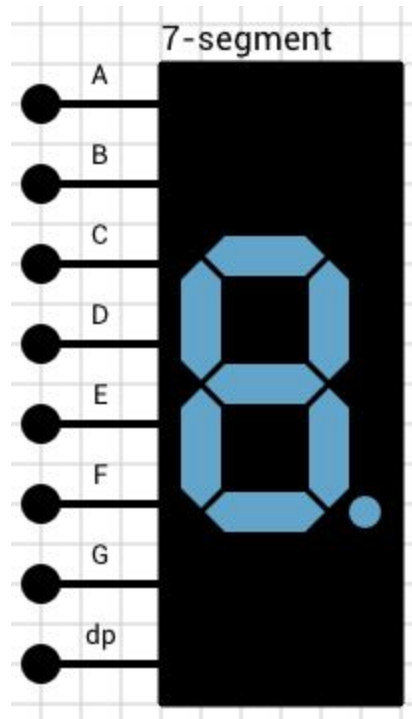


Fig 10: 7-segment display

### 1.3.8 Wire mergers and splitters

Wire mergers are used for combining multiple 1-bit signals into a larger signal. Wire splitters, on the other hand, are used for breaking a wide signal into multiple 1-bit signals. The only configurable option for both of these components is the number of bits.

### 1.3.9 Bus



Fig 11: Bus configuration

Buses are special components used to select one of many signals. A bus has multiple inputs and only one output, all of them having the same size. The output is always equal to the **one and only non-Undefined input**. If multiple inputs to a bus have a non-Undefined value, the bus output is Undefined. Figure 11 shows the bus configuration dialog. The minimum number of inputs is 2.

### 1.3.10 ROM

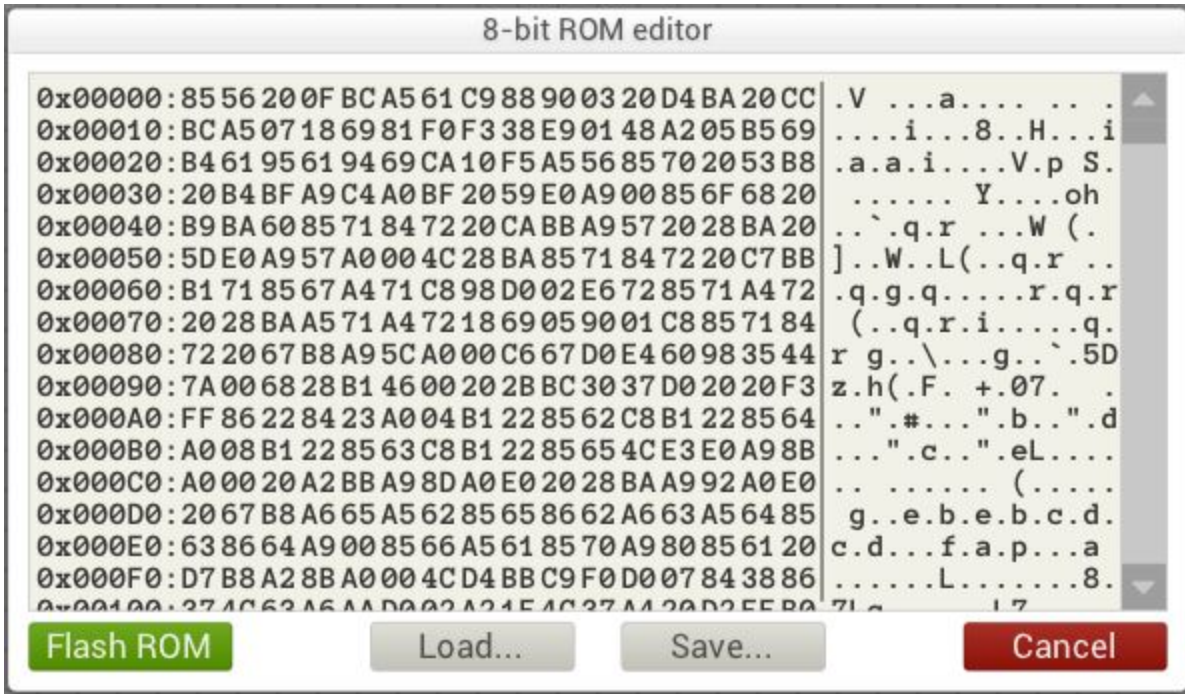


Fig. 12: The ROM editor

ROMs are components used for storing immutable data. Each ROM contains M N-bit words. Words can be up to 64 bits wide and the maximum number of words is 64k.

They have 1 input which is the address of the word you want to read and at least 1 output corresponding to the selected word data. In case the word size is larger than 16 bits the component has multiple outputs. From top to bottom each output holds the data from the LSB to the MSB of the selected word.

ROMs are combinational memory components (i.e. there's no clock input). Every time the '*addr*' input changes, the outputs are updated to reflect the newly selected word.

The context menu (right click) of a ROM component has an 'Edit values...' option. Selecting it will show the ROM editor (fig. 12). From there you can change the contents of the ROM, either by typing or by loading an external file. You can also save the current contents of the ROM to a file for later use.

ROM files are saved in the 'roms' folder, under the user's data folder (see [section 1.1](#) for the exact location depending on your OS). If you want to load your own data into a ROM, place your file into the above folder and then load it through the ROM editor. ROM files must have a 'rom' extension.

### 1.3.11 Static RAM

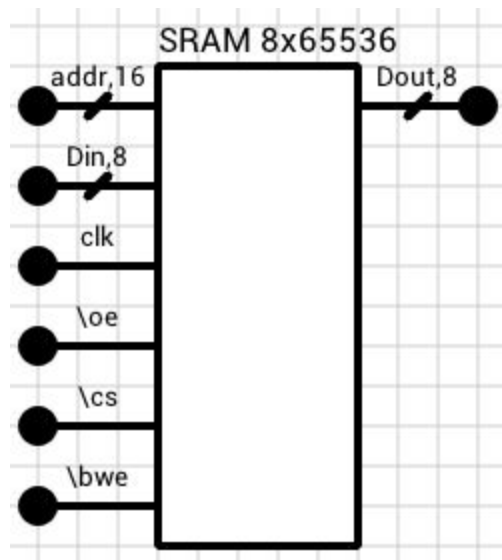


Fig 13: 64k x 8-bit Static RAM component

Unlike ROMs, SRAMs (Static Random Access Memory) are volatile synchronous memory components. Their contents do not persist on disk. They can be used to store runtime data for a circuit.

SRAMs have several different pins. Currently, the SRAM component behaves like a **synchronous flow-through standard write SRAM** chip. All inputs are latched on the rising edge of the supplied clock (*clk*) and the addressed memory location is read/written as some later timestep. The read delay of an SRAM component is currently fixed and equal to 5ns.

*\oe* (Output Enable) is an active low signal used to control whether *Dout* will hold the data from the selected address. When *\oe* is HIGH, *Dout* is equal to UNDEFINED. *\oe* is an asynchronous signal.

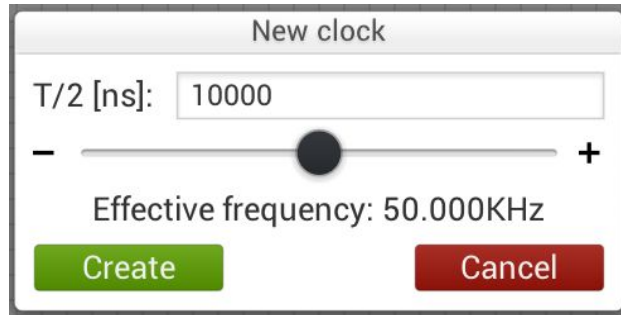
*\cs* (Chip Select) is an active low signal used to control whether the rest of the inputs will be latched on the next rising edge of the clock or not.

*\bwe* (Byte Write Enable) is an active low signal used to control whether the addressed memory location will be written on the next rising edge of the clock or not.

### 1.3.12 Clocks

Clocks are 1-bit input ports which oscillate between 0 and 1 when activated. Figure 14 shows the clock configuration dialog. The only available parameter is the duration of the half clock cycle ( $T/2$ ), measured in nanoseconds. The minimum clock frequency is 1Hz (tick every 500ms) and the maximum clock frequency is 500MHz (tick every 1ns).

You can change the default clock frequency by using the slider. Dragging the handle to the left decreases the value. Dragging it to the right increases it. The dragging distance affects the rate of change of the parameter.



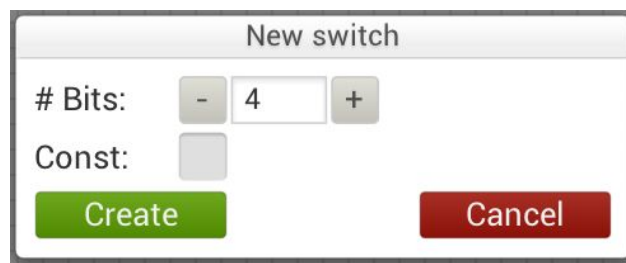
A dialog box titled "New clock" with a text input field for "T/2 [ns]" containing the value "10000". Below the input field is a slider control with a black knob in the center, flanked by minus and plus signs. Under the slider, the text "Effective frequency: 50.000KHz" is displayed. At the bottom are two buttons: a green "Create" button and a red "Cancel" button.

Fig. 14: Clock configuration

Clocks can be either stopped or running. When a clock is stopped you can toggle its value the same way you change the value of a numeric input port. When the clock is running, you cannot interact with it.

Finally, clocks cannot be marked as Const.

### 1.3.13 Switches



A dialog box titled "New switch" with a "# Bits:" label followed by a minus button, a text input field containing "4", and a plus button. Below this is a "Const:" label followed by an unchecked checkbox. At the bottom are two buttons: a green "Create" button and a red "Cancel" button.

Fig 15: Switch configuration

Switches are multi-pin input ports. Figure 15 shows their configuration dialog. A switch has multiple 1-bit pins which you can control using the individual switch controls inside the port. Const has the same meaning as with the other input port types.

### 1.3.14 LED Matrix



A dialog box titled "New LED matrix" with a "Size:" label followed by a minus button, a text input field containing "4", a plus button, an "x" label, another minus button, a text input field containing "2", and a final plus button. Below this is a "Debug:" label followed by an unchecked checkbox. At the bottom are two buttons: a green "Create" button and a red "Cancel" button.

Fig 16: LED matrix configuration

A  $N \times M$  LED matrix contains  $M$  rows by  $N$  columns of LEDs. There's one  $N$ -bit pin for each one of the  $M$  rows. The maximum size of an LED matrix is  $16 \times 16$ .

### 1.3.15 Pixel display

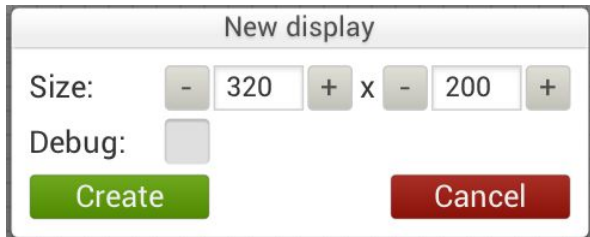


Fig 17: Pixel display configuration

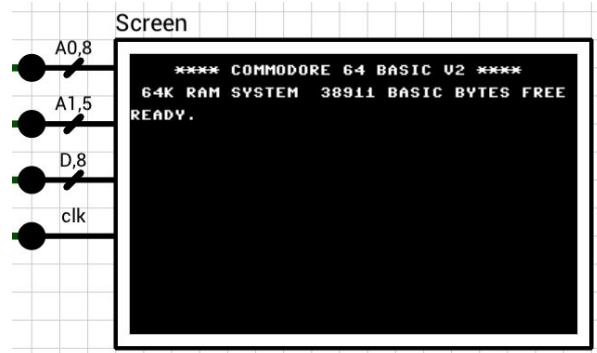


Fig 18: A 320x200 1bpp display

Figure 17 shows the configuration dialog of a Display and figure 18 shows an example of a 320x200 display.

Displays are sequential components (i.e. they have a clock input) with internal memory. The  $A_x$  inputs combined make up the address of the first pixel you want to write to.  $D$  is the data of the 8 pixels you want to write and  $clk$  is the clock. Whenever  $clk$  goes from LOW to HIGH, the data found at the  $D$  input are written to the corresponding pixels, starting at address  $A$  (LSB of the address is the LSB of  $A0$ ).

Currently, displays support only 1bpp. You can clear the display to black or a random color for each pixel via its context menu.

### 1.3.16 Tunnels

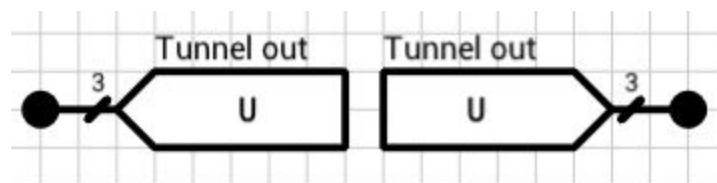


Fig 19: An output tunnel (left) and its corresponding input (right)

Tunnels behave like numeric I/O ports. You can only create output tunnels via the gate toolbar, the same way you create numeric output ports.

You create input tunnels via the output tunnel's context menu (right click on it). All input tunnels get the same name as their parent.

Cloning an input tunnel is the same as creating a new input from the same output tunnel.

Cloning an output tunnel is the same as creating a new output tunnel via the gate toolbar. Cloned output tunnels have no connection to their original tunnels.

If the current selection includes both an input tunnel and its parent, cloning the selection will create a new output tunnel and a new linked input tunnel. There will be no connection to the original ports.

Finally, tunnels do not appear on the final component. They might behave like I/O ports but they aren't actually I/O ports.

### 1.3.17 Basic logic gates

The last 8 buttons of the gate toolbar are the basic logic gates. Left clicking on any one of those will place a new instance under the mouse cursor. Right clicking opens the configuration dialog. From there you can configure the number of inputs as well as the number of bits for the gate. AND, OR, NAND, NOR have a variable number of inputs (from 2 to 16). Buffer, NOT, XOR and Tri-state buffers have a fixed number of inputs.

For a brief explanation on the purpose of each gate, place the mouse cursor on the corresponding toolbar button and wait for the tooltip.

Buffers are the only gates with configurable propagation delay. Figure 20 shows the buffer configuration dialog. All other gates have standard propagation delays, which depend on their number of inputs. For a complete list see [Appendix A](#).

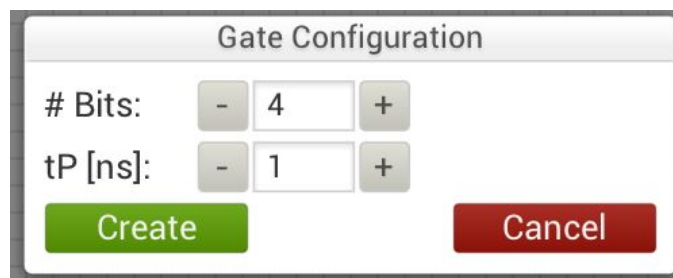


Fig 209: Buffer configuration



## 1.4 Testbench editor

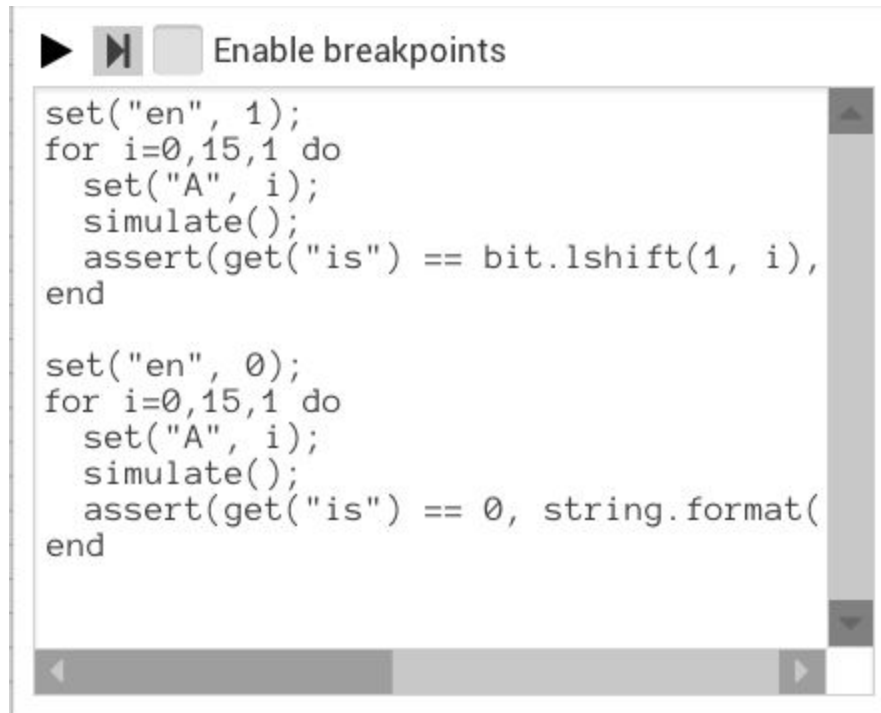


Fig. 21: Testbench editor panel

The testbench editor UI is shown in figure 21. You can open the testbench editor by pressing F2 on the keyboard. On the top there are 2 buttons and a checkbox. The first button starts and stops the testbench. The second button resumes testbench execution until the next breakpoint, if breakpoints are enabled. For more details on testbenches see [chapter 4](#).

## 1.5 Console

Figure 22 shows the console. You can open it by pressing F1 from the keyboard. You can use the console for printing debug messages from scripted components or testbenches.

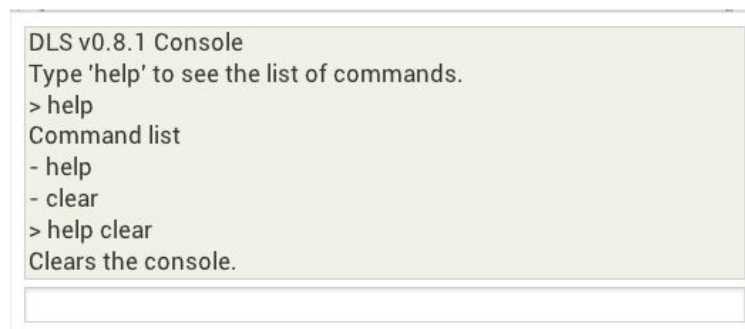


Fig. 22: Console

## 1.6 Logic Analyzer

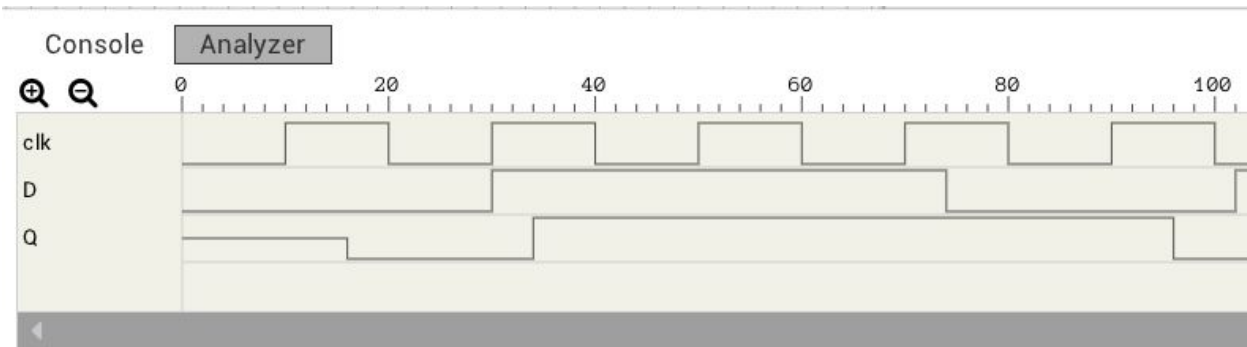


Fig 23: Logic Analyzer

Figure 23 shows the logic analyzer. You can open it by pressing F1 (the same key as the Console) and clicking on the Analyzer button at the top of the panel.

The logic analyzer collects and displays the values of all **debug numeric outputs** through time. In order to monitor the value of a wire/output pin you have to connect a debug output port to it. The analyzer can be used to find delays in your circuits and for debugging their behavior (finding hazards and glitches).

Clicking the Reset button from the [Simulation info](#) popup resets the history of all the debug outputs, as well as the logic analyzer timeline.

## 1.7 Wiring

Creating wires is simple. Clicking on any output pin will start a new wire. After that, you can place as many points you want on the grid. Finally, click on the input pin you want this wire to be connected to, to finish it.

Using the right mouse button while drawing a new wire removes the last point. If the last point was the first point of the wire (the one on the output pin), the wire is destroyed. Alternatively, you can press Esc to cancel the current wire.

You can select wires by left clicking on them. Right clicking will bring up the context menu from where you can either create a new wire up to this point (T-junction) or delete the selected wires.

## 2. Components

You can create new components by clicking the “Componentize circuit” button from the Main toolbar. The circuit should have at least 1 input port, 1 output port and 1 gate/component in order to be turned into a component. Clicking the “Componentize circuit” button will bring up the dialog shown in figure 24.



Fig. 24: New component dialog

The edit box at the top of the dialog is used as the name of the new component. Next, you can select inside which package you want to add the new component, from a list of available packages. Clicking the + button will open a new dialog from which you can create a new package.

The 2 checkboxes should be self-explanatory. If “Create new instance” is checked, once the OK button is clicked, a new instance of the component will be ready to be added to the circuit. “Clear schematic” clears the current schematic.

The new component will have as input pins all variable input ports (those not marked as Const). Each input port pin (independent of its width) will be shown as an input pin on the new component. The same is true for all non-debug output ports.

Finally, you can inspect the internal circuit of a component by selecting Inspect from its context menu (right click on the component).

## 3. Scripted components

Scripted components can be used for creating complex behaviours, or in situations where simulation speed matters. The scripting language used in DLS is Lua (LuaJIT v2.0.4 with BitOp library, for binary operations).

Each scripted component should have at least 2 functions specified; `init()` and `simulate()`.

### 3.1 The `init()` function

This function is used for one-time initialization of the component. It has one argument (named “`component`” in the initial code), which is an temporary Lua object used for specifying the component’s inputs and outputs.

The “`component`” object has 2 member functions, `addInput()` and `addOutput()`. Both functions take the same number (2) and type of arguments. The first argument should be a string, specifying the name of the input pin. The second argument is the width of the pin, in bits.

Script 1 below shows how to create a component with 2 inputs (“A” and “B”) and 1 output (“Y”), all 1-bit wide.

```
function init(component)
    component.addInput("A", 1);
    component.addInput("B", 1);
    component.addOutput("Y", 1);
end
```

Script 1: Initializing a scripted component with 2 inputs and 1 output.

The `init()` function is a good place to also initialize internal component state. You can use global variables which will remain valid for the lifetime of the component. Note that the `init()` function is called every time you press the OK button.

### 3.2 The `simulate()` function

The `simulate()` function is where the actual component logic should be placed. Every time an input changes value, the `simulate()` function is called. It has one argument (named “`inputs`” in the initial code), which is an array of numbers, corresponding to each one of the inputs, in the order they are created in the `init()` function, and should return an array of objects, one for each internal timestep of the simulation.

Inputs can take any valid integer value, depending on the width of the input pin, or the special value of `V_UNDEFINED`.

`V_UNDEFINED` means that either the input pin isn't connected to any output pin from another component/gate/port, or the wire is in high impedance state (hi-Z).

`simulate()` should return an array of objects. Each element in the array corresponds to one internal timestep of the component. Each timestep "object" should have 2 members. One named "t" which is the internal timestep value, greater than or equal to 1, and the second named "outputs" which should be an array of numbers with the values of each output pin for this timestep, in the order they were created in the `init()` function.

Script 2 shows an example using the inputs and outputs created in Script 1 to model a 1-bit AND2 gate. It also shows the `print()` function which is available to the scripts for printing debug information to the console.

```
function simulate(inputs)
  local a = inputs[1];
  local b = inputs[2];
  local res = { V_UNDEFINED };

  if (a == 0 or b == 0) then
    res[1] = 0;
  elseif (a ~= V_UNDEFINED and b ~= V_UNDEFINED) then
    res[1] = bit.band(a, b);
  end

  return {
    { t = 1, outputs = res }
  };
end
```

Script 2: 1-bit AND2 gate implemented as a scripted component.

## 4. Testbenches

Testbenches are Lua scripts which can control the inputs and check the outputs of a circuit. They are used to verify if a circuit behaves as it's expected. The table below shows a list of functions available in testbenches, to control the circuit.

Function	Description
<code>set(string inputName, number value)</code>	Sets the input named <code>inputName</code> to the specified value. Push buttons keep their value until it's set again (like if you were holding the mouse button down). You cannot set the value of a clock. See <code>tick()</code> for details. If the input port doesn't exist, the function does nothing.
<code>get(string outputName)</code>	Returns the value of the output port named <code>outputName</code> . Only works with Output port and LED Matrix components. If the output port does not exist, the returned value is <code>V_UNDEFINED</code> .
<code>simulate()</code>	Simulates the circuit using the current input values.
<code>tick(string clockName)</code>	Ticks the clock named <code>clockName</code> . One call to this function forces the clock to switch levels, so a full clock cycle is executed after two consecutive calls to <code>tick()</code> .
<code>assert(boolean condition, string message)</code>	Check if the condition is true, and if it's not the message is printed to the console, and the testbench terminates.
<code>print(string message)</code>	Prints a message to the console.

Table 2: List of functions available for use in a testbench.

Script 3 below shows a simple testbench for the scripted component we created in the previous section. In order to run it, connect 2 1-bit input ports to the circuit to the component's input pins and an output port to the output pin.

```
set("A", 0);  
set("B", 0);
```

```
simulate();
assert(get("Y") == 0, "Invalid output, expected 0");

set("A", 1);
set("B", 1);
simulate();
assert(get("Y") == 1, "Invalid output, expected 1");

set("A", 0);
set("B", V_UNDEFINED);
simulate();
assert(get("Y") == 0, "Invalid output, expected 0");

print("Testbench completed successfully");
```

Script 3: Testbench for the scripted component from scripts 1 and 2.

**NOTE:** While the testbench is running, you cannot interact with the circuit.

## A. Build-in Gate/Component Information

Table A.1 below shows the delay of each one of the basic gates based on its number of inputs. The number of bits isn't taken into account because each individual bit is processed in parallel to the rest.

Currently (as of version 0.11.0) 1 gate delay equals to 1 nanosecond, which is also the smallest amount of time a circuit can advance forward in time. As a result, the maximum effective clock frequency is 500MHz (2 gate delays are required for a complete clock cycle).

Gate	Number of Inputs								
	1	2	3	4	5	6	7	8	N
Buffer	var	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
NOT	1	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
AND	N/A	1	2	2	3	3	3	3	$\log_2(\text{nextPowOf2}(N))$
OR	N/A	1	2	2	3	3	3	3	$\log_2(\text{nextPowOf2}(N))$
NAND	N/A	1	2	2	3	3	3	3	$\log_2(\text{nextPowOf2}(N))$
NOR	N/A	1	2	2	3	3	3	3	$\log_2(\text{nextPowOf2}(N))$
XOR	N/A	3	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Tristate Buffer	N/A	1	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table A.1: Basic logic gate delays (in arbitrary time units)

Table A.2 shows the delay of each build-in component you can create from the I/O dialog.

Component	Delay
Bus	1
Wire Merger	1
Wire Splitter	1
ROM	Number of Address bits

Table A.2: Delays for the components from the I/O dialog.





## B. Release History

### **v0.14.0 (2017-04-11)**

1. NEW: Notes
2. NEW: Faster renderer
3. NEW: Faster simulator
4. NEW: Removed error values. V\_ERROR is no longer defined in scripted components.
5. FIX: Clicking on read-only edit boxes was crashing the app.
6. FIX: Connecting the output of a gate/component to an input of the same gate/component is forbidden (was causing problems with wires).
7. FIX: Random crash related to constant input ports.

### **v0.13.0 (2017-02-03)**

1. NEW: Undo system
2. NEW: Signal tunnels to connect distant parts of a circuit without making a mess of wires.
3. NEW: Static RAM component (synchronous, flow-through, standard write SRAM)
4. NEW: Configurable LED color
5. NEW: Wire thickness based on number of bits.
6. NEW: Edit buffer propagation delay
7. NEW: Increased maximum unzoom level and made the grid adaptive (to avoid rendering too many lines at small zoom levels)
8. NEW: Increased maximum target simulation speed to 1s/s.
9. FIX: Buffers didn't check for UNDEFINED and ERROR values
10. FIX: Component bounding boxes were wrong (didn't take into account constant/debug I/O ports).
11. FIX: Large range slider change speed is now dependent on the range.
12. FIX: Large range slider +/- labels turned into buttons to quickly change the current value.
13. FIX: Giving a name to a new component and then opening the new package dialog was crashing the app.
14. FIX: When returning to the parent schematic the previous view is restored.
15. FIX: Vertical scrollbar in listboxes with N+1 elements, where N is the number of visible elements.

### **v0.12.0 (2016-12-21)**

1. NEW: Select multiple components and wires by dragging the mouse on the grid, or by holding down the Shift key.
2. NEW: Rotate gates and some of the build-in components using the mouse wheel.
3. NEW: Right clicking on a wire brings up its context menu (doesn't remove the wire anymore).
4. NEW: New Buffer gate with configurable delay (up to 50ns).
5. NEW: Gate/component toolbar has been moved to the left of the screen and all build-in components and gates are displayed. No more I/O and misc gates dialog.

6. NEW: Toolbars can now be unpinned and automatically fold when the cursor isn't near them.
7. NEW: Reduced the size of the schematic file format. Now each component in a schematic is saved only once, independent of the number of instances in the schematic.
8. FIX: Fixed the bounding boxes of some gates/components.
9. FIX: Display textures are now aligned correctly inside the component rectangle for all resolutions.
10. FIX: Stream levels with 0-terminated sequences have random values but the sequence lengths are fixed.

#### **v0.11.2 (2016-11-27)**

1. NEW: Added the total number of transistors in the current circuit to the Simulation/Circuit info panel.
2. FIX: Fixed the description of the Tri-state buffer tutorial level about buses and Undefined values.
3. FIX: When the Simulation/Circuit info panel was visible the Escape key didn't work (didn't bring up the Pause menu).

#### **v0.11.1 (2016-11-19)**

1. FIX: When adding or removing wires from inputs to components/gates, the gate's input pin wasn't set to the correct value.
2. FIX: Truth table levels didn't handle signed I/O correctly during verification.
3. FIX: If you created a new component in the sandbox and then started a level to use the component, the component wasn't included in the package. You had to restart the game.

#### **v0.11.0 (2016-10-26)**

1. NEW: Simulator internals have changed. Circuits are now flattened to their basic logic gates (instead of working on a tree of components) and simulation time is advanced forward if there are any events queued. Steady-state isn't required in order to finish simulating a circuit, so both unstable (i.e. oscillators) and asynchronous circuits should work correctly.
2. NEW: Basic logic analyzer. Every debug output is monitored and displayed in the analyzer.
3. NEW: You can now clear Display components to black or random color via their context menu.
4. FIX: Histogram bucket size was wrong and extreme values weren't displayed properly.
5. FIX: Multi-bit OR gate gave wrong results when one of its inputs was equal to 1.
6. FIX: Inspecting a component while the parent circuit had a testbench, caused the testbench to reset when returning to parent.
7. FIX: During level editing, when the user created a new component and then inspected another component, when returning to the new component's schematic the main menu toolbar was wrong.

#### **v0.10.1 (2016-09-28)**

1. NEW: Level statistics. Compare your results against other players!
2. NEW: Added a red border around the schematic editor which is displayed only if the last circuit simulation was unstable.
3. NEW: Credits dialog
4. FIX: Window icon (Windows + Linux)
5. FIX: Fixed tootip location when it got out of screen.
6. FIX: Package names were getting mixed up when deleting packages from the library.
7. FIX: Background image was not rendered correctly with UI scale other than 1.0, in pause menus
8. FIX: Editing the schematic after an unstable simulation was crashing the app.

#### **v0.10.0 (2016-09-16)**

1. NEW: The first 37 levels!
2. NEW: Respect user's keyboard layout (e.g. QWERTY, Dvorak, AZERTY) when writing into text fields. At the moment all shortcuts (Ctrl + C/V/X) require pressing the corresponding QWERTY key.
3. NEW: Buses and tristate buffers (TSB) do not produce error values any more. When the control input of a TSB is UNDEFINED, its output is UNDEFINED. Buses forward the first non-UNDEFINED value to the output (from top to bottom).
4. NEW: New main menu UI.
5. NEW: You can cancel wire creation by pressing the ESC key.
6. FIX: Fixed a bug which appeared when undoing wire point placement. If the mouse moved a bit while clicking the right mouse button, the operation wasn't executed.
7. FIX: When deleting gates/components without any wires attached, the confirmation dialog isn't displayed.
8. FIX: You can now change clock values using the mouse, just like regular numeric inputs, when the clock isn't ticking.
9. FIX: Fixed multi-input gate delays. The more the inputs the larger the delay. See Appendix A from the manual for more details.
10. FIX: If there's no string in the clipboard the app was crashing when pasting into a text field.
11. FIX: Log file wasn't sent along with the crash dump (Windows only)

#### **v0.9.0 (2016-07-16)**

1. NEW: ROMs can have word size up to 64 bits.
2. NEW: Multi-bit multi-input standard gates.
3. NEW: Load/save ROM data from/to external files.
4. NEW: 7-segment displays
5. FIX: Changing any window related option which caused bgfx to the reset, ended up crashing the game if there was previously a display in the circuit and a display was added after the reset.

#### **v0.8.1 (2016-04-28)**

1. NEW: Added a small manual describing the sandbox (this document).
2. FIX: Calling internal DLS functions from Lua scripts ended up crashing the application if the parameter types weren't correct.
3. FIX: Converting input ports was causing a crash when the port had more than 1 wire.
4. FIX: No 6 from previous release didn't behave correctly with testbenches.
5. FIX: Components in packages weren't sorted correctly.
6. FIX: Testbenches are not correctly saved within components.
7. FIX: New ROMs are now reset to 0.
8. FIX: Push button Debug parameter renamed to Const.

#### **v0.8.0 (2016-04-13)**

1. NEW: Increased the max. number of bits per wire/pin to 16, and the max. number of ROM bytes to 64k.
2. FIX: Minimizing and restoring the window was causing the V-Sync flag to be reset.
3. FIX: ROM editor window contents were getting out of bounds.
4. FIX: When returning to a parent schematic, sometimes the schematic wasn't centered correctly on screen.
5. FIX: Disabled auto UI scale by default to prevent bugs with wrong physical screen dimensions reported by the OS.
6. FIX: When adding or removing wires (directly or indirectly), the circuit is forced to be simulated to prevent undefined values.

#### **v0.7.1 (2016-03-14)**

1. FIX: Deleting components from the library caused a crash.
2. FIX: Printing too many messages to the console caused a crash.

#### **v0.7.0 (2016-03-13)**

1. NEW: Added testbenches (1 per schematic). You can open the testbench editor by pressing F2. Testbenches are saved along with the schematic (.sch v1.4).
2. NEW: Replaced Duktape with LuaJIT (Lua 5.1 with BitOp library). Unfortunately, your previous circuits written in JS will no longer load/work.
3. NEW: Added V-Sync option in the Options dialog.
4. FIX: Console now opens with the F1 key.
5. FIX: Windows: Schematics and packages are now saved under your Documents\DLS folder (%USERPROFILE%\Documents\DLS). You can move your old schematics/packages from %APPDATA%\DLS into this folder.
6. FIX: bgfx now uses the appropriate renderer for each platform (DX under Windows, OGL under Linux/OSX).
7. FIX: Fixed a bug in the renderer which occurred when the number of vertices per frame exceeded 65536.

8. FIX: OSX: Auto UI scale is calculated based on the physical monitor dimensions returned by GLFW.
9. FIX: Several UI fixes and improvements.

#### **v0.6.1 (2016-01-24)**

1. NEW: Uses bgfx for handling rendering details on all platforms. Currently only the OpenGL backend of bgfx is used on all 3 platforms, but this might change later (e.g. DX backend on Windows and Metal backend on OSX).
2. NEW: OSX and Linux builds. The Linux build has been tested on Ubuntu and it's expected to work on any Debian based distro (don't take my word on that :)). The OSX build has been tested on El Capitan. If the game is in alpha stage, these 2 builds should be considered pre-alpha at the moment. Please report any bugs either on the itch.io page or twitter (@jdryg).

#### **v0.6.0 (2016-01-15)**

1. NEW: Switches and push buttons.
2. NEW: LED matrix display.
3. NEW: 1-bit per pixel display with internal memory (resolutions from 32x32 up to 800x800).
4. NEW: Adjustable clock frequencies (from 1 up to 60Hz). A special value of 0 indicates that the clock will tick as fast as possible (normally at 60Hz with VSync enabled).
5. NEW: (EXPERIMENTAL) Scripted components. You can now build custom components with JavaScript. Duktape JS engine is used internally.
6. NEW: Tri-state buffers and buses.
7. NEW: Conversions between the various input and outputs port from the context menu.
8. NEW: In-game console for tweaking and monitoring of scripted components (via print() calls). You can open the console using the ''' button.
9. FIX: Editbox fixes (numpad support, mouse selection and caret placement, etc.)
10. FIX: Added a read-only editbox to the top of the save/load schematic dialogs which shows the folder used to store the schematics.
11. FIX: Various other bug fixes and improvements.

#### **v0.5.0 (2015-12-10)**

1. NEW: New and improved UI.
2. NEW: Inline editing of input port values.
3. NEW: Debug output ports. They function as normal output ports but no output pin is added in the generated component.
4. NEW: Convert between 1-bit input ports and clock.
5. NEW: Convert between const/variable input and normal/debug output ports.
6. NEW: Added a "delete package" button in the library window to remove unwanted packages.
7. FIX: Various bug fixes and memory improvements.

#### **v0.4.2 (2015-11-23)**

1. FIX: The simulator gave wrong results/outputs for certain circuits with large delays.
2. FIX: If you cleared the schematic while drawing a new wire (e.g. by clicking on the Clear Schematic button on the main toolbar), the game crashed.

#### **v0.4.1 (2015-11-15)**

1. FIX: Const inputs were incorrectly visible in the generated component.
2. FIX: Switching packages from the Library dialog sometimes caused a crash.
3. FIX: Combobox item height didn't match listbox item height (wrong selection at the edges of comboboxes).

#### **v0.4.0 (2015-11-14)**

1. NEW: Rotate gates and bus splitters/mergers using the R key.
2. NEW: Added a ROM component (max size: 8 data bits x 8 address bits = 256 bytes).
3. FIX: Deleting schematics via the Load Schematic dialog has been fixed.
4. FIX: Automatic zoom has been limited. When loading circuits with (e.g.) only 1 gate, the zoom factor got too big.
5. FIX: You can now zoom with the mouse wheel if the cursor is on a gate, if the gate doesn't consume the mouse wheel scroll (i.e. inputs ports)
6. FIX: Fixed scrollbar heights to correctly show the scrolling region.
7. FIX: When deleting a component from a package, the preview is removed.
8. FIX: Schematic filenames are sanitized before saving the schematic in order to remove invalid filename characters. All invalid characters are replaced with dashes.

#### **v0.3.1 (2015-11-09)**

1. FIX: Fixed a bug which appeared when deleting an input gate when it was already connected to some other gates, and then simulating the circuit.

#### **v0.3.0 (2015-11-08)**

1. NEW: Switched to binary files for schematics and packages. The game still supports the old file format, so your old schematics are still valid. Packages get automatically converted to the new format, on the first run.
2. NEW: Changed Edit to Inspect in component's context menu. From now on, in order to change the inspected component's circuit, you have to componentize/save the inspected circuit, change it and replace it in the parent schematic, manually. This has been done in order to simplify inspection.
3. NEW: You can now change input port values with the mouse wheel, while the cursor is on the value. This helps when working with inputs with more than 4 bits.
4. NEW: Ctrl+dragging a gate/component clones the component.
5. FIX: When loading or inspecting a schematic/component, the view automatically zooms to fit the new schematic.
6. FIX: Removed a bug which allowed the user to rename a gate/component to an empty name.

7. FIX: When switching packages in the library dialog, the previously selected component gets unselected.
8. FIX: Zooming in/out with the cursor is performed only if the cursor isn't on top of a component/gate. This was required for change no. 4 above.
9. FIX: When inspecting a component, the new schematic is loaded asynchronously.
10. FIX: Several performance and memory optimizations. Loading times and memory requirements have been improved.

#### **v0.2.0 (2015-10-31)**

1. NEW: Added UI scale option. You can scale the UI from 50% up to 300% in case it's too small on your monitor (e.g. hdpi displays). The automatic algorithm tries to calculate the monitor's DPI and scale the UI to match the dimensions of a 96dpi display.
2. NEW: Schematic loading has been made asynchronous.
3. NEW: Pan/zoom using the keyboard. Use WSAD or the arrow keys to pan, Q/Z or numpad +/- to zoom.
4. NEW: Added tooltips to gate toolbar buttons. If you hover over a button, you get information about the specific gate.
5. NEW: You can now start creating a new wire by holding down the Ctrl key and clicking on an existing wires.
6. NEW: Added a Delete button in the Load Schematic dialog which lets you delete previously saved schematics.
7. NEW: When popping a circuit from the stack, a confirmation dialog has been added to let you choose whether you want to replace the old IC with the new one, or revert all possible changes. This message is displayed even if you didn't touch the child circuit. This will fixed in a future version.
8. FIX: Completely disable interaction with the schematic, if a modal dialog is open.
9. FIX: Bus wires are now green when all their bits are HIGH, and red when all their bits are LOW. Intermediate values are displayed as orange.
10. FIX: Loading a previously saved schematic now correctly loads input values.
11. FIX: Changed bus splitter/merger gate size to align with the grid.
12. FIX: There was a point near the top left of the screen where all input/output pins of the gates in the toolbar became hot. If you clicked on it, the game crashed because those gates aren't part of a schematic.
13. FIX: When adding a new wire to a gate input pin, the previously connected wire (if any) is deleted.
14. FIX: When clicking on an output pin which already has wires, none of the existing wires is selected.

#### **v0.1.0 (2015-10-28)**

1. Initial release