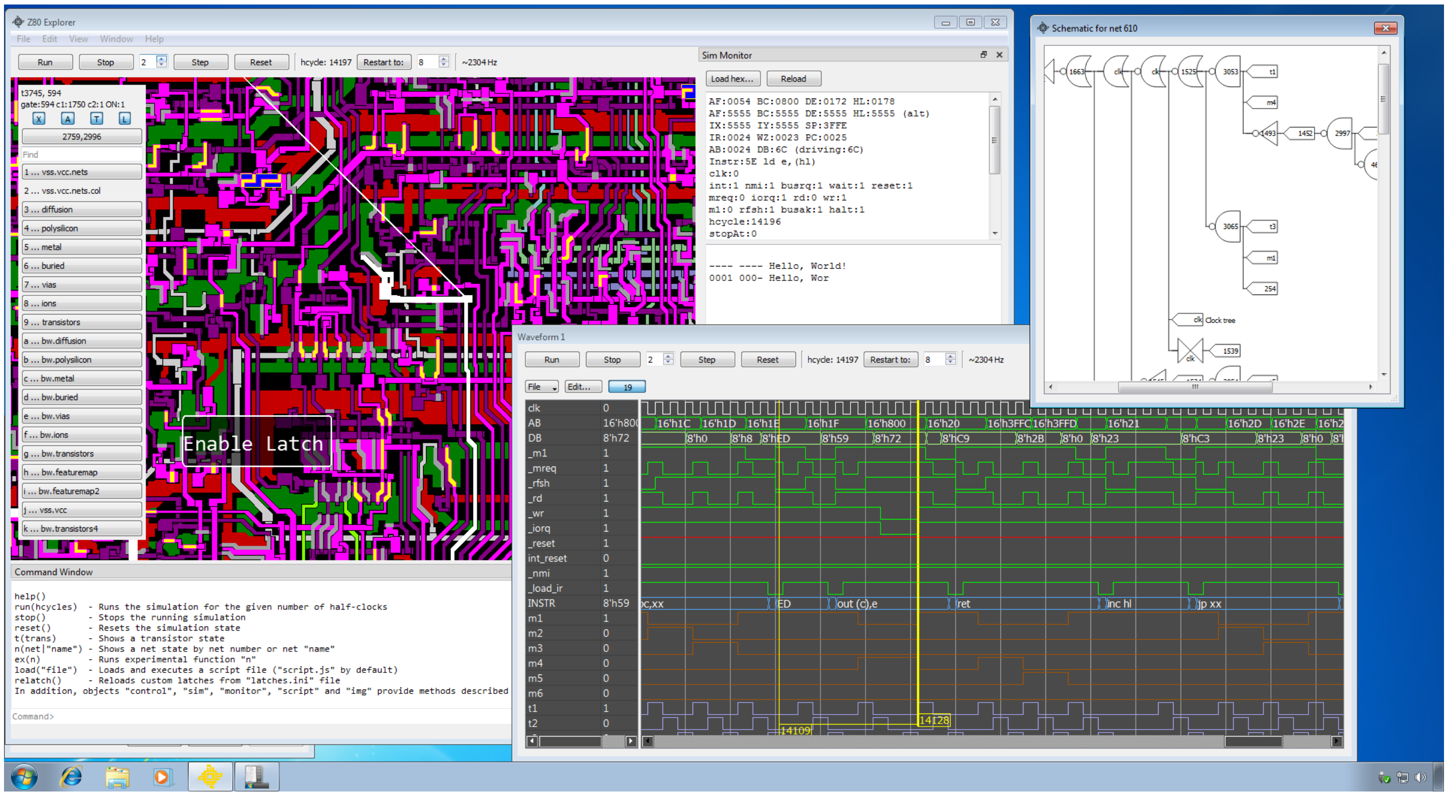
Z80 Explorer

A Z80 Netlist-level Simulator



```
AF:0054 BC:0800 DE:0172 HL:0178
AF:5555 BC:5555 DE:5555 HL:5555 (alt)
IX:5555 IY:5555 SP:3FFE
IR:0024 WZ:0023 PC:0025
AB:0024 DB:6C (driving:6C)
Instr:5E ld e,(hl)
clk:0
int:1 nmi:1 busrq:1 wait:1 reset:1
mreq:0 iorq:1 rd:0 wr:1
m1:0 rfsh:1 busak:1 halt:1
hcycle:14196
stopAt:0
```

```
---- ---- Hello, World!
0001 000- Hello, Wor
```

Command Window

```
help()
run(hcycles)  - Runs the simulation for the given number of half-clocks
stop()        - Stops the running simulation
reset()       - Resets the simulation state
t(trans)      - Shows a transistor state
n(net|"name") - Shows a net state by net number or net "name"
ex(n)         - Runs experimental function "n"
load("file")  - Loads and executes a script file ("script.js" by default)
relatch()     - Reloads custom latches from "latches.ini" file
In addition, objects "control", "sim", "monitor", "script" and "img" provide methods described
Command>
```

https://baltazarstudios.com/z80explorer

Contents

Overview

Z80 Explorer is a Z80 netlist-level simulator capable of running Z80 machine code. Its goal is to be an educational tool with features that help reverse engineer and understand this legendary CPU better.

This document is updated to describe version 1.05 of the tool.

Installation

This application does not need installation. Extract it to a folder of your choice and run it.

Main Application View / Image View

The main application window shows a view of the original NMOS Z80 chip die with various layers. This view lets you see various chip features like nets, transistors, and vias. You can combine those layers into a composite view. It is like an X-ray of the chip die.

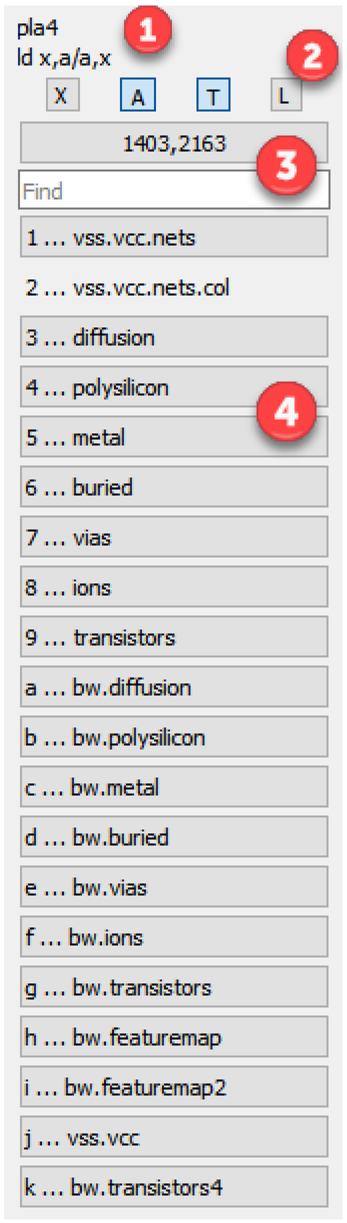
The main application recognizes these keyboard hotkeys:

Main window keyboard assignments	
Ctrl + Q	Open a new Image View window (up to 4)
Ctrl + W	Open a new Waveform View window (up to 4)
F2	Edit net names (rename and delete names)
F3	Edit definitions of buses
F4	Edit Watchlist (list of nets with history data)
F5	Edit custom image annotations
F6	Edit custom nets colors
F10	Show or hide Application Log window
F11	Show or hide Command Window
F12	Show or hide Sim Monitor window

These are the keyboard hotkeys for the image view:

Image view keyboard assignments	
1 … 9 …	Select one of the images
CTRL + 1 … 9 …	Adds (composite XOR) selected image on top of the previous one(s)
F1	Cycle zoom modes: Fill, Fit, Identity (1:1), Scale
Cursor arrows	Pan up/down/left/right (also use mouse to pan)
PgUp/PgDown	Zoom (also use mouse wheel to zoom)
X	Show or hide active nets
SPACE	Show or hide annotations
T	Show or hide active transistors
. (period)	Cycle to show all transistors (SHIFT + “.” period)
L	Show or hide detected latches
ESC	Progressively clear Find net, Selected nets
N	Show of hide net names (visible when zoomed in)

On the left side of the image view is the overlay consisting of several sections:



(1) As you move the mouse cursor over the chip image, this section shows some information about the current net.
(2) Four toggle buttons show or hide features on the image (see below).
(3) This area shows the mouse coordinates on the image and the "Find" option to look for the nets, symbols, or transistors.
(4) You can select to show or combine the list of available image layers by pressing the corresponding key(s).

You can combine images by holding the CTRL key while pressing a key associated with another layer. For example, if you want to see the diffusion and poly layers along with their buried contacts, you would press "3", then hold CTRL and press "4", then "6". Then, release CTRL.

Layer "2" is identical to layer "1" but has nets and bus coloring applied. The coloring is described in the section below.

Many layers are duplicated in black/white, and those have "bw." prefix in their name. Some features are more pronounced when merging the monochrome images into a desired view.

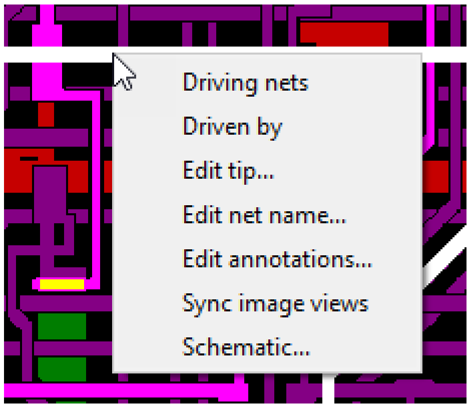
By clicking on the coordinates button (above the "Find") (3), you can enter the image coordinates and center the image at the exact location.

The four buttons on the top (2) are:
[X] – Show or hide all active nets (nets whose current state during the simulation is logic "1"). The equivalent keyboard shortcut is "X".
[A] – Show or hide annotations. The keyboard shortcut is SPACE.
[T] – Highlight (in yellow) all active transistors. The keyboard shortcut is "T". An additional keyboard shortcut, "." (period), toggles to show you all the transistors and not just the active ones.
[L] – Show or hide detected latches. The keyboard shortcut is "L".

"Find" lets you search for different features: nets by number (for example, "398"), nets by name ("m1"), transistors ("t2232"), and buses ("AB").

The feature, if found, flashes on the screen and stays highlighted. Buses have only their first net highlighted. Press ESC once to clear the highlight; press ENTER key to flash the last highlight again.

As you move your mouse over the image, double-click on a net to select it, then right-click to open a context menu. The menu shows options based on the selected net and the context:

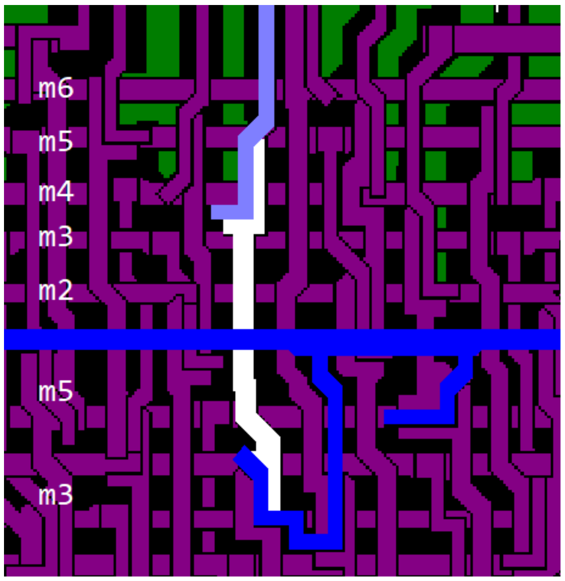


The “Sync image views” option syncs all additional views to this image location and zoom factor.
The sections below describe other options.

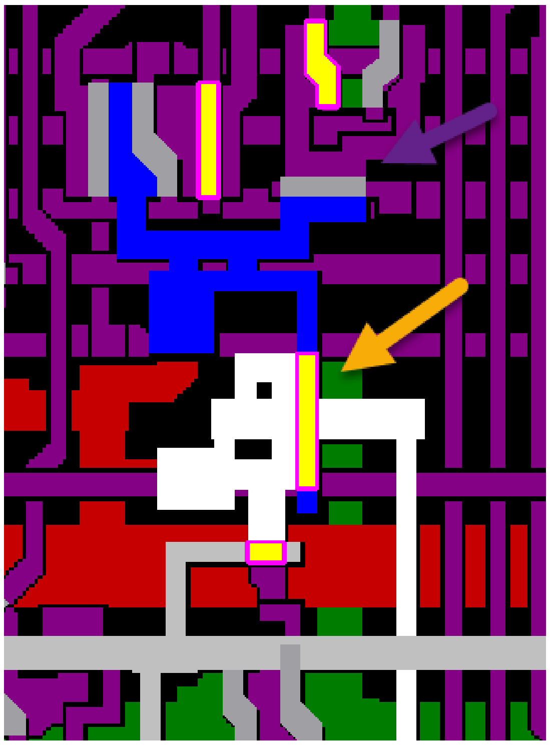
Driving/Driven

“Driving nets” and “Driven by” are two search operations on the netlist that show which nets are gated by the selected signal and which nets contribute to the selected signal. These options perform a shallow search in that they report only the first nets adjacent to the transistors and disregard serially laid-out gates such as NAND. These options are great for quickly tracing the signals.
As they are being identified, the relevant nets are marked with increasingly lighter blue so they can be visually discerned as being part of that group. You can use these options to trace a control signal up and down its chain of transistors and to find out what causes it to change the state.

Example: net 3105 shows “Driven by” nets in blue, while the primary net (3105) is highlighted in white for the best contrast.



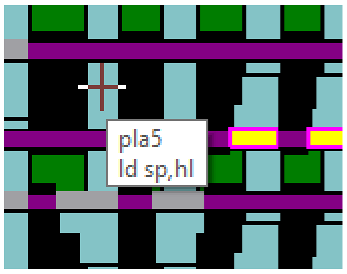
The picture below shows active transistors in yellow and inactive in gray. To show the transistors, press the "T" key. Sometimes, it is helpful to see all transistors lit; press "." (period) key for that. In the image, white, having the best contrast, shows the primary selected net, blue are dependent nets and gray below is the "clk" line. The clock net is always colored gray.



Hint: Un-select a net by double-clicking on an empty (black) space between the nets, or hit the ESC key to deselect progressively.

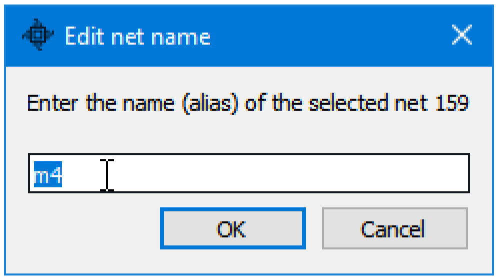
Edit Tips

Tips are another way to help annotate nets. Since the net names are short and often abbreviated, tips provide the mechanism to expand on the meaning of a signal. Tips act as "tooltips": as you hold your mouse over a net with a tip assigned to it, the tip shows as a mouse tooltip. Several nets have tips predefined in the default Z80 resource file. Some of the most useful ones are the PLA signal wires, for which the tips show the opcode group that a PLA wire decodes.



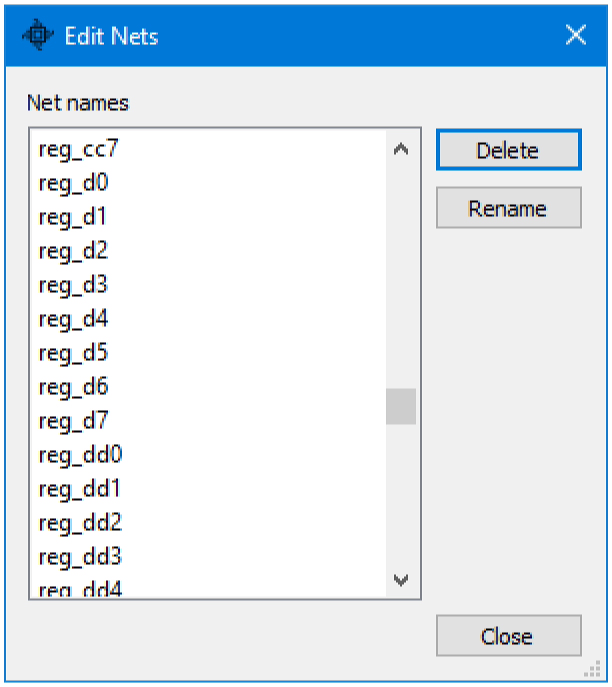
Net Names

Most of Z80's nets are unnamed. They can be identified only by their net number. This application requires naming the nets being tracked (watched) in the simulation history and the waveform graph. Use this context menu to name and rename a selected net quickly. You can also delete the net name by clearing the input field and clicking OK.



Changes to net names are immediately visible in any waveform view that uses them.
Naming a net is a frequent operation as you figure out what it does (before adding it to the waveform view and rerunning the simulation). For example, you can give a net a temporary name (like "n287" for the net number 287), and then you can add it to the waveform view. Later, you can delete the net name if you don't need it any longer or rename it to something more descriptive.

Another way to manage net names is to open the Edit Nets dialog with the application shortcut F2.



You can delete or rename one or more net names.

Adaptive Annotations

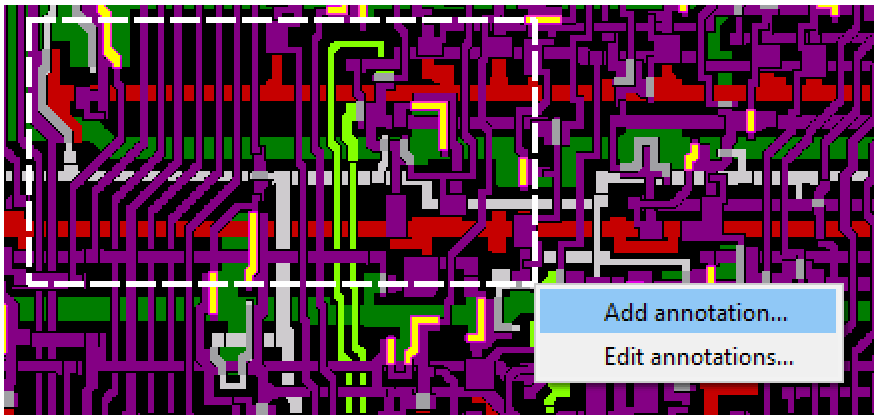
Custom (or user) annotations are text descriptions positioned over an image to mark a feature or show some part of interest. It is “adaptive” since, as you zoom in, larger annotations disappear to reveal smaller ones (otherwise, large text would be in the way when you zoom in.) The point at which an annotation appears and disappears depends on the size of the text and the zoom level.

You can add and edit annotations in several ways. The simplest and most intuitive way is to right-click and drag the mouse to make a selection on the image where you want your annotation to be placed. The size of this area also roughly defines the initial annotation text size, which you can adjust once the Edit Annotation dialog appears.

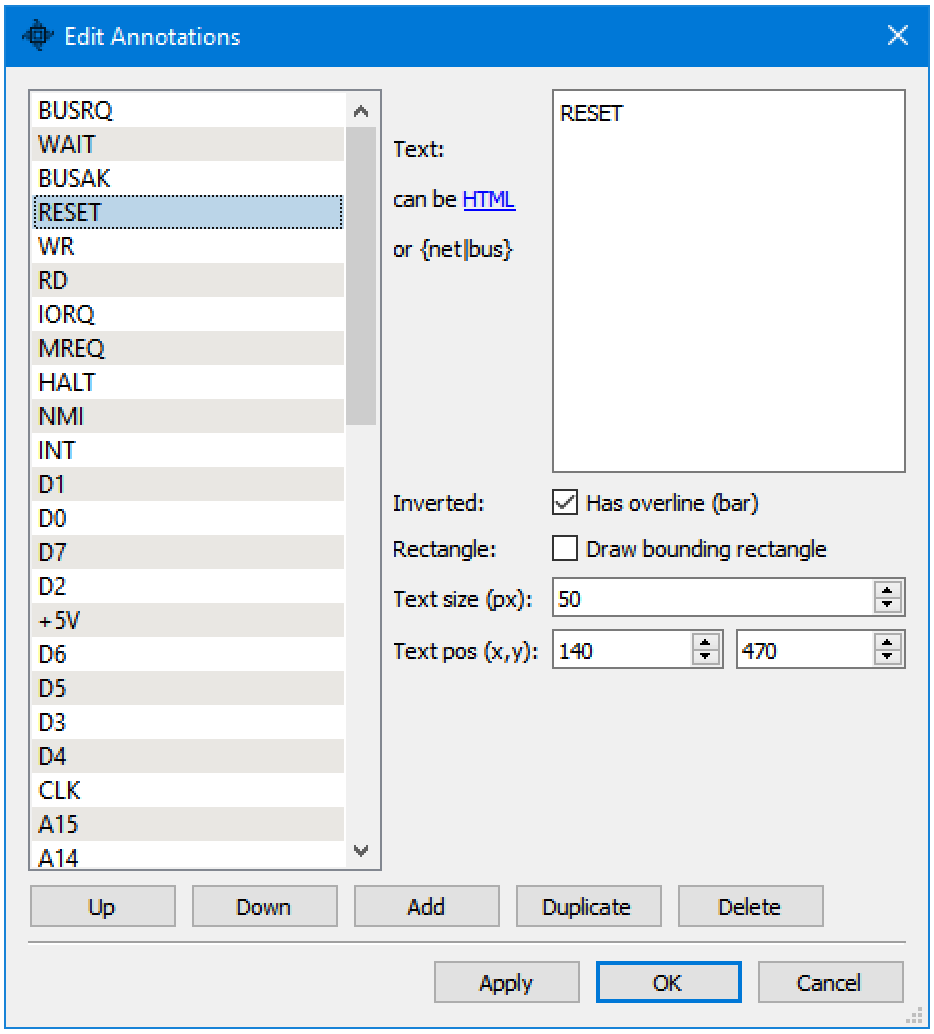
Alternatively, the application shortcut key F5 opens the Edit Annotations dialog.

If you hold down a Shift key while making a selection, the selection rectangle snaps to a grid. This is useful when you are creating adjacent annotations which need to be aligned.

Using the mouse, you can select several existing annotations, and after you choose the option “Edit Annotations…”, the dialog opens with all the annotations within your selection box, all selected. Annotations can also be positioned outside the image area. Those always show and do not fade out as you zoom.



After you add an annotation, you can edit and fine-tune it if desired. Most of the time, you will re-adjust the text sizes and positions. As you adjust these parameters, click on the “Apply” button to preview the changes.



Text: can be HTML or {net|bus}: RESET

Inverted: ☑ Has overline (bar)

Rectangle: ☐ Draw bounding rectangle

Text size (px): 50

Text pos (x,y): 140 470

The annotation text itself can include HTML-style tags (or Markdown formats). This reference website shows supported tags: https://doc.qt.io/qt-6/richtext-html-subset.html. Click on the hyperlink "can be **HTML**" within the dialog to open that reference web page in your browser.

The option "Inverted" or "overline" is helpful to tag pins with inverted input: it places a bar on top of the name.

The option "Rectangle" draws a bounding rectangle using the geometry of your original mouse selection area. This bounding rectangle cannot be changed without recreating the annotation.

By selecting multiple annotations from the list on the left, you can modify them all at once; for example, you can set the text size or one of the positions of several annotations to the same value.

You can have more than one set of annotations. While the default set loads on the application startup ("annotations.json"), you can drag and drop a file containing another set of annotations onto the Image View. Any edits will be saved to that other file. For example, "annot_internals.json" is an example of another annotation file: it contains markups of features that are less abstract than the default annotations file.

The annotation text also supports **macros**, which are simple substitutions of named nets and buses for their value. Anywhere in the annotation text, you can refer to a net or a bus, enclose its name with a set of curly brackets (for example: “{DBUS}”), and the annotation shows its current value.

A resource file, “annot_functional.json,” is an example of such a ‘functional’ annotation showing the runtime values of all major buses and latches.

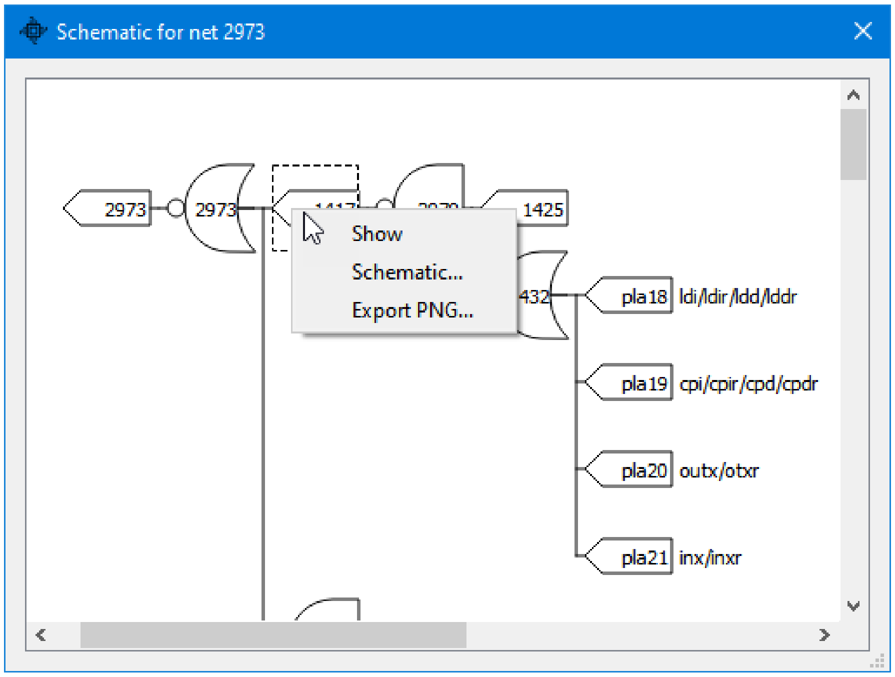
Since all internal data buses in Z80 carry inverted values, an option was added to invert the value of a net/bus when you add tilde ~ in front of a name, as in “{~VBUS}”. The value displayed is inverted, and a symbol ~ is shown to make that unambiguous. This output format is consistent with the Waveform View’s “Ones’ Complement” format.

Schematic View

This option shows the net schematic. At this time, it is experimental and far from perfect; it is a work in progress. The selected net is traced back to all the nets contributing to its state. This traversal ends with certain terminating nets, which are chosen as reasonably good endpoints:

- Power, ground, and clock networks
- PLA signals
- Internal buses (ab, db, ubus, vbus)
- A few predefined nets like “int_reset” (internal reset signal)
- Detected and custom-defined latches

In addition, the traversal ends when a contributing net has already been processed to prevent possibly infinite loops. The leaf nodes display their tip text (if defined).



Use the mouse to pan and zoom the view. When you double-click on a logic gate, the Image view shows its feature location on the die (you may want to zoom out the Image view to see it). The corresponding context menu option is “Show”.

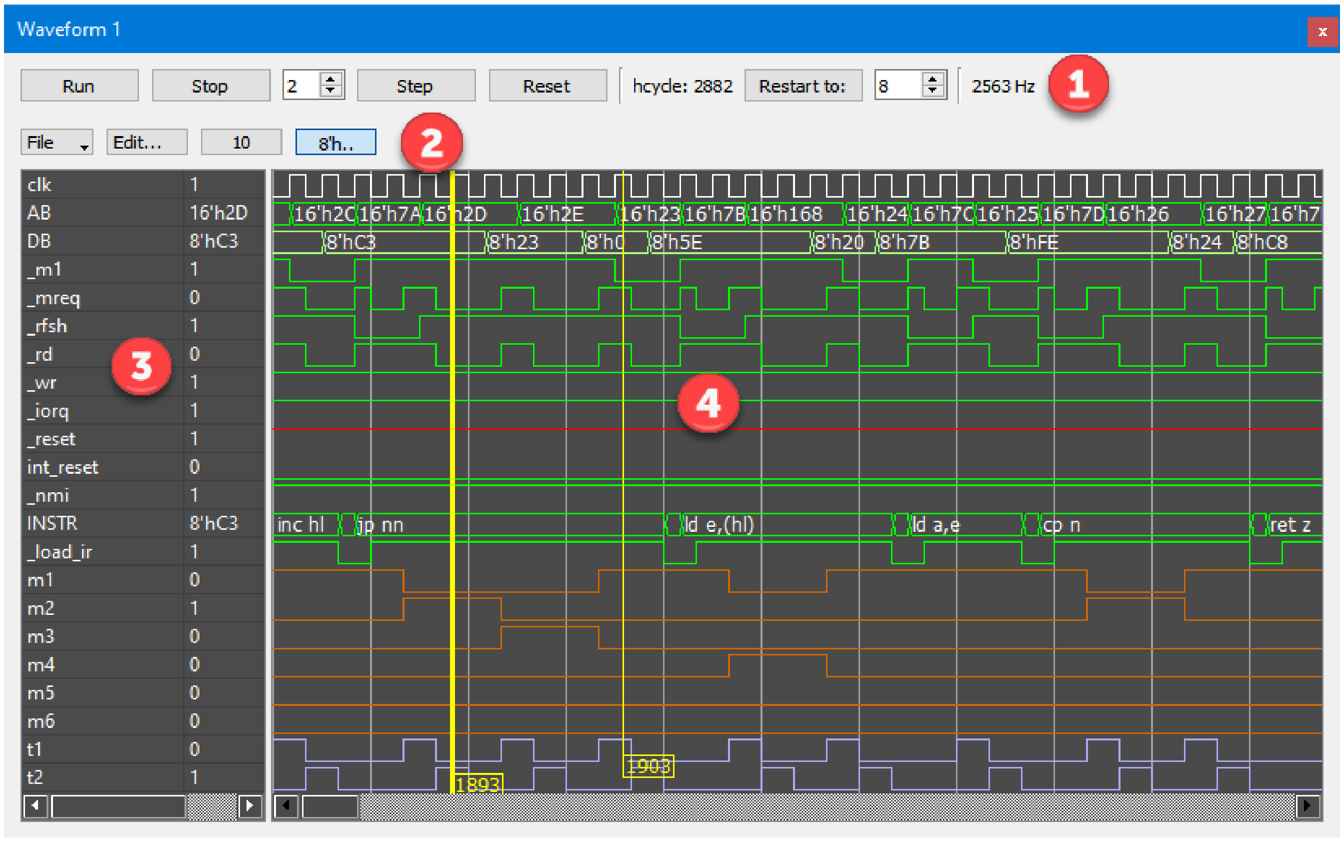
The context menu “Schematic…” creates a new schematic view, starting at a selected gate.

The generated logic network tree is compacted: redundant inverters are coalesced with downstream gates if possible (for example, inverter + NOR gate are collapsed into a single OR gate). This results in a somewhat smaller network. If that is not desirable, you can skip the compacting step and generate the original network if you hold down the Ctrl key while selecting the “Schematic…” menu item. In that case, the title of its window show “(not optimized”) to confirm your choice.

Note: The logic parser code can detect certain features, but it is fairly generic - and not foolproof. Trying to automatically reverse-assemble and create a schematic diagram from a chip that has been heavily hand-optimized is a difficult problem. Use this view only as a reference while still “reading” the traces yourself and using the “Driven by” option.

Waveform View

When you run a simulation, selected signal values are captured over time. The Waveform view provides a view into the history of those signals and buses even as the simulation is running:



hcycle: 2882

2563 Hz

clk	1
AB	16'h2D
DB	8'hC3
_m1	1
_mreq	0
_rfsh	1
_rd	0
_wr	1
_iorq	1
_reset	1
int_reset	0
_nmi	1
INSTR	8'hC3
_load_ir	1
m1	0
m2	1
m3	0
m4	0
m5	0
m6	0
t1	0
t2	1

Different sections of the waveform window are:

1. The simulation toolbox.
2. The file menu lets you save and load custom views (the list of nets and buses you are observing, along with their display formats). You can create several different views and load them as you need to. You can also export and save the image as a PNG file.
   The Edit button opens the Edit dialog described below.
   The push button next to Edit shows the distance between the two cursors, measured in half-clock values. When that button is depressed, the two cursors link together, and the sliding one moves the other along.
   The button marked with "8h'.." toggles bus value decorations, the bus width (in Verilog format), on and off.
3. This column shows the list of nets and buses that you are watching.
4. The main pane shows the history of all the nets and buses that you are watching. There are 2 cursors available, which you can position at any point to read out the net values.

Use the mouse to work with the waveform window:

- The scroll wheel zooms in and out in the timeline (horizontally)
- The scroll wheel with the Ctrl key pressed enlarges the view vertically
- Pan left and right by dragging the pane to the extent of the available data
- Double-click to position one of the cursors (you can also click on the bottom where the cursor "flags" are to bring a cursor)
- Hold the cursors and move them

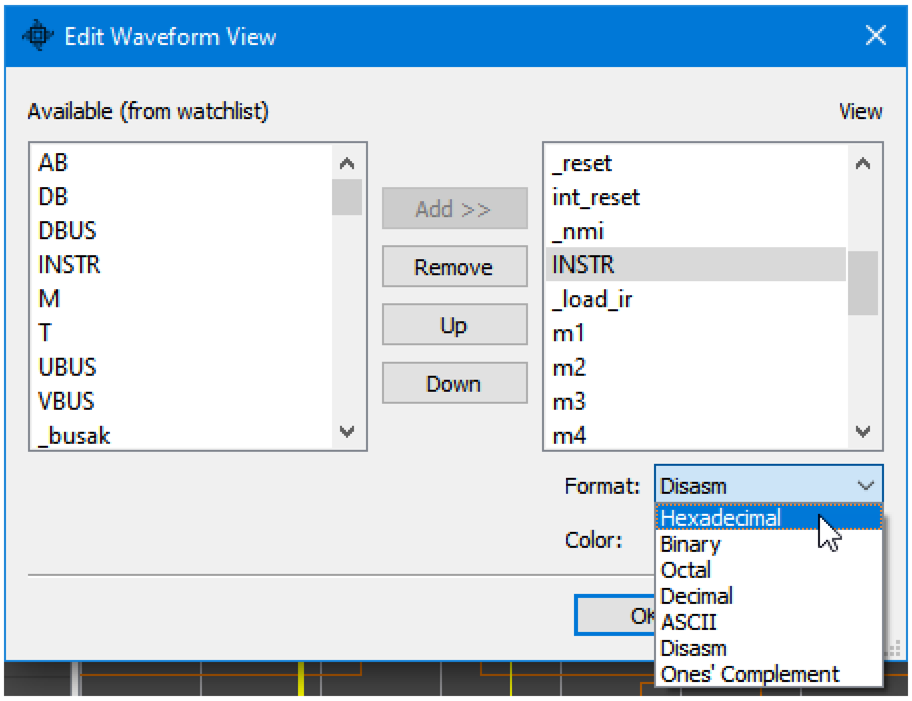
You can have up to 4 separate waveform windows. Each remembers its list of nets when you close the application. You can open additional windows by pressing the Ctrl + W keyboard shortcut or via the "Window" menu.

There are a few things to keep in mind:

- For any nets and buses to be available, they first need to be added to the Watchlist
- That also means they need to be named (see "Net Names")
- The signal history is a rolling window containing 1000 half-cycles of data (sample points)

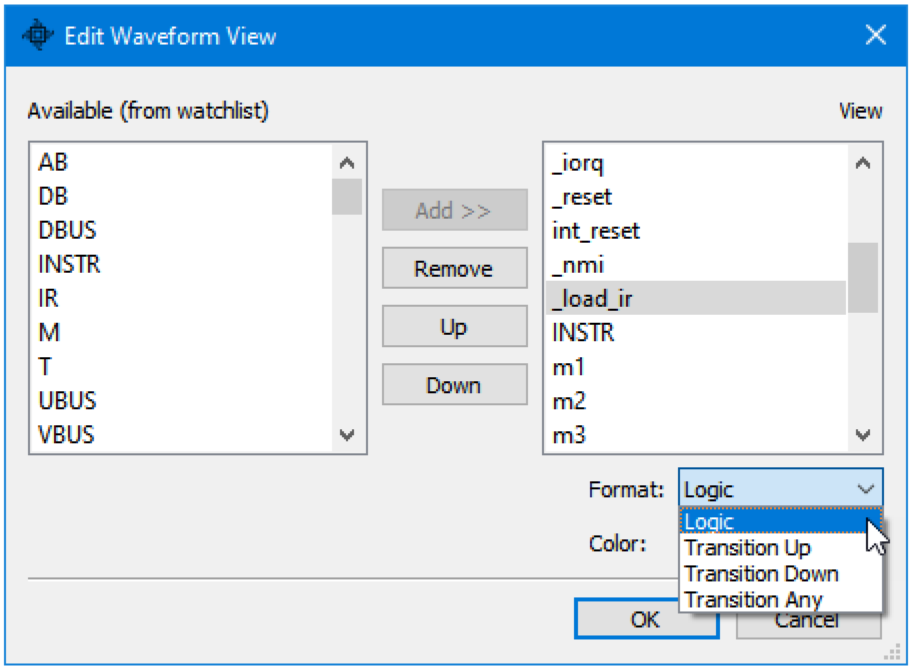
The Edit dialog provides a way to select which nets and buses you want to graph and the format and color of each item. This application's convention is that the net names are lowercase, and bus names are uppercased.

The runtime values of the buses can be shown in hexadecimal, binary, octal, decimal or ASCII, 1's Complement, and a simple Z80 dis-assembly format (which is used for instruction register):



The One's Complement format is useful when displaying internal data buses (VBUS, UBUS,…) since the bits on those buses are inverted.

Nets, only having a logical "0" or "1" value, can also have their transitions tagged:



The list of Waveform view keyboard assignments:

Left/Right	Pan left/right
Up/Down	Enlarge and shrink the view vertically.
PgUp/PgDown	Zoom (also use the mouse wheel to zoom)

Waveform view mouse actions:

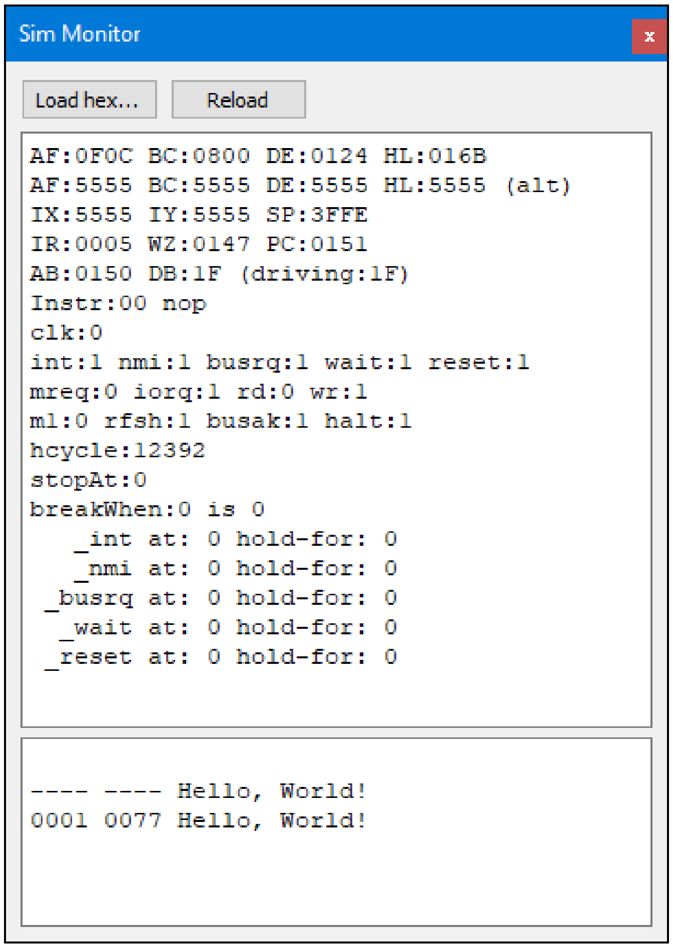
LB hold	Pan left-right; move cursors
Wheel	Zoom in and out into the history of data
Wheel + Ctrl	Enlarge and shrink the view vertically.

Sim Monitor

This window shows the simulation state and provides a terminal-like output for the executing Z80 programs to write to.

While the information listed on the top should be obvious (current values of Z80 registers and external chip pins), the values on the bottom, “stopAt” and “breakWhen” directly correspond to the values set by the scripting commands “mon.stopAt()” and “mon.breakWhen()”. Those commands, when used, stop the simulation at a specified cycle number and/or when a particular net assumes the specified value. The value of zero means the trigger has not been set or it has been cleared.

The lines listing the pin values (“_int”, …) correspond to the monitor’s memory-mapped control area: Z80 programs executing inside the simulator can cause action on these input pins. See “Simulation Environment” chapter for more details.



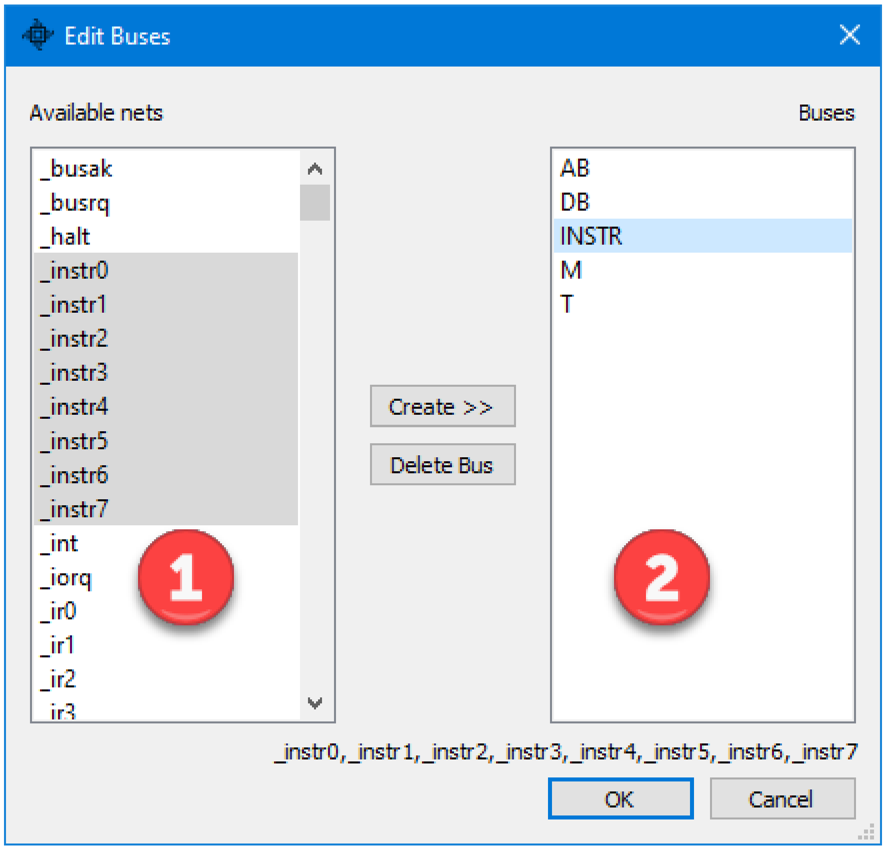
```
AF:0F0C BC:0800 DE:0124 HL:016B
AF:5555 BC:5555 DE:5555 HL:5555 (alt)
IX:5555 IY:5555 SP:3FFE
IR:0005 WZ:0147 PC:0151
AB:0150 DB:1F (driving:1F)
Instr:00 nop
clk:0
int:1 nmi:1 busrq:1 wait:1 reset:1
mreq:0 iorq:1 rd:0 wr:1
m1:0 rfsh:1 busak:1 halt:1
hcycle:12392
stopAt:0
breakWhen:0 is 0
    _int at: 0 hold-for: 0
    _nmi at: 0 hold-for: 0
  _busrq at: 0 hold-for: 0
   _wait at: 0 hold-for: 0
  _reset at: 0 hold-for: 0
```

```
---- ---- Hello, World!
0001 0077 Hello, World!
```

The bottom portion displays ASCII characters the executing Z80 program sent over its IO-mapped port 0x0800. It ignores LF (ASCII code 10) in CR/LF sequence. Writing the value of 4 (ASCII EOD, “End-of-Transmission”) to that port also causes the simulation to stop.

Edit Buses

You can combine signals into buses. Buses are two or more nets that share some logical function. The application has a few predefined buses. Bus names can be edited, or new buses can be created via the Edit Buses dialog. Press F3 at any time (or select Buses from the application's Edit menu) to open this dialog:



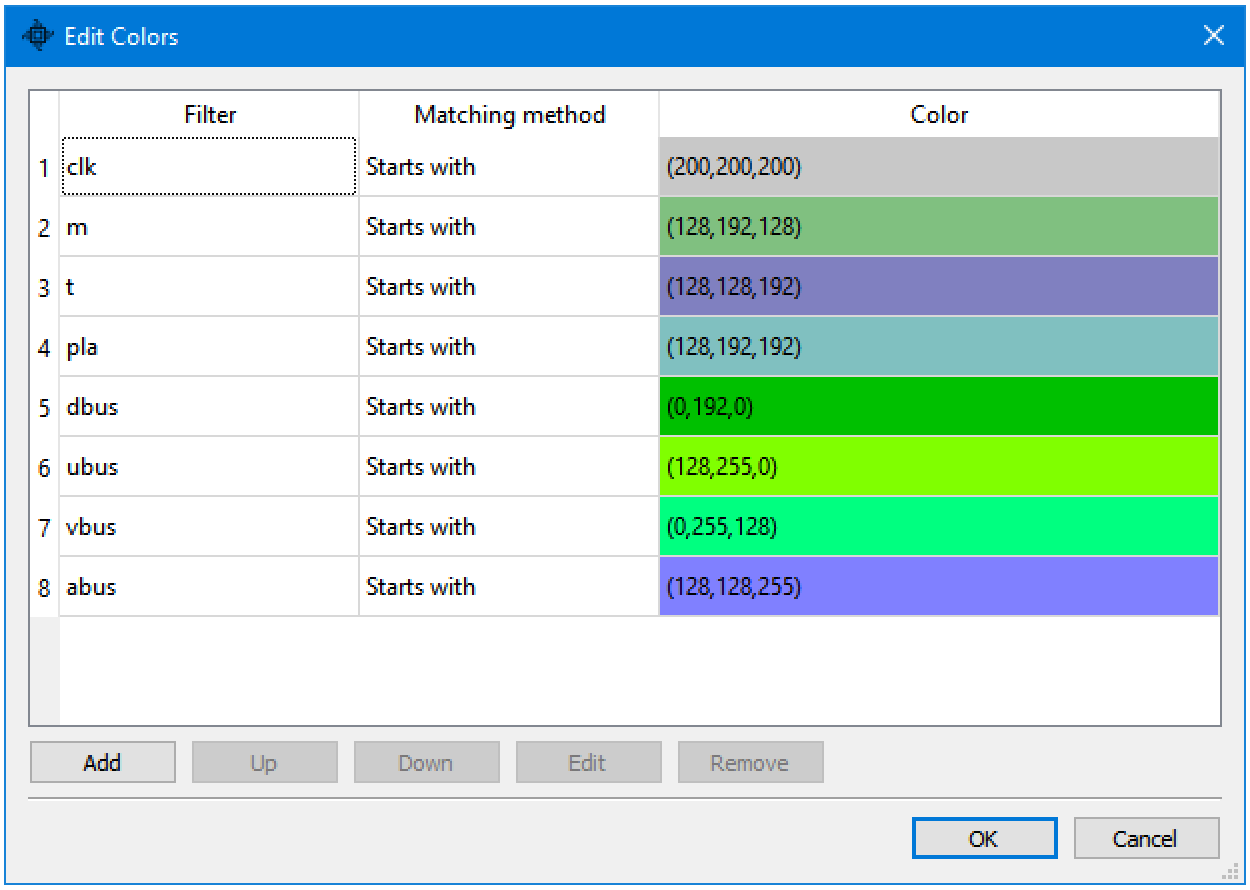
_instr0,_instr1,_instr2,_instr3,_instr4,_instr5,_instr6,_instr7

On the left is a list of (named) nets from which you can make up a bus. Press and hold the CTRL key to select each successive net. After selecting two or more, click "Create >>" to create a bus. The selected signals combine into a bus in the order you selected them.
By the application convention, all buses are written in uppercase, while the other signals (nets) are lowercase. Currently, you cannot rename a bus; the only way to effectively do that is to delete it and recreate it using a new name.

Edit Colors

Define custom colors for the selected nets shown in the image view layer number “2”. The buses and nets to be colored are selected by the means of (1) a filter, and (2) the corresponding matching method, which can be one of:

1. Exact match: the “Filter” name has to match exactly, for the color to the applied
2. Starts with: the net name needs to start with the specified word
3. Regex: use a standard regular expression to pick which nets to color
4. Net number: simply write the net number to which to apply the color

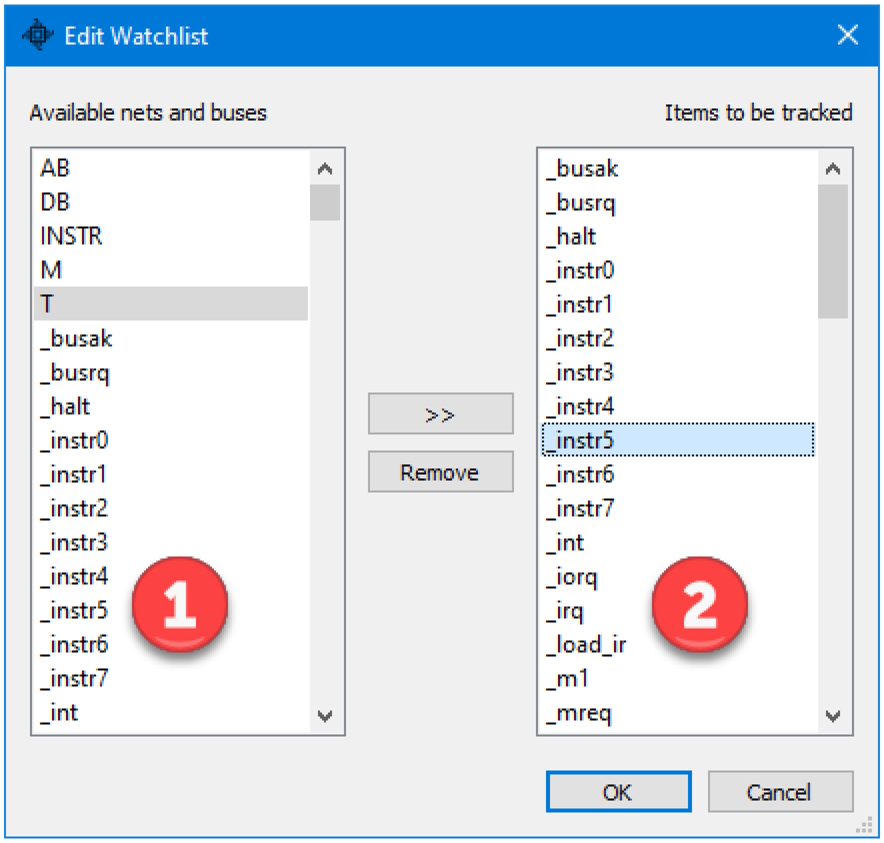


	Filter	Matching method	Color
1	clk	Starts with	(200,200,200)
2	m	Starts with	(128,192,128)
3	t	Starts with	(128,128,192)
4	pla	Starts with	(128,192,192)
5	dbus	Starts with	(0,192,0)
6	ubus	Starts with	(128,255,0)
7	vbus	Starts with	(0,255,128)
8	abus	Starts with	(128,128,255)

You can select multiple color entries, reposition them in the list, or remove them. The color position is important because a net can match multiple color entries, but only the first match is used.

Edit Watchlist

This dialog has an important function: selecting which nets will be available to various application parts - like a waveform view - to function properly. Although the Z80 processor has almost 8000 nets, it is practical to work with only a small subset. For a net to be watched or tracked, it must first be named (see “Net Names” section). After you name a net, it is automatically added to the watchlist. This dialog lets you fine-tune which items to track and which to leave out for the simulation performance reasons.

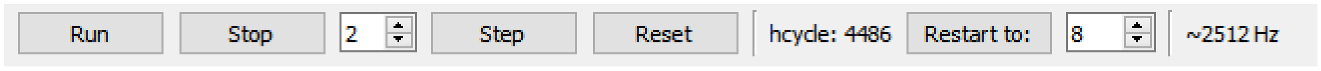


All named (and available) nets that are available to be tracked are listed on the left, and on the right side are the currently tracked ones. If you include a bus, the program adds all the nets that make up that bus. You can select multiple entries on each side to speed up the process.

Running a Simulation

On start, the application loads a “hello_world.hex” test file with Z80 code to print “Hello, World”. The program is loaded into a simulated address space and ready to execute every time you start *Z80 Explorer*. You can edit “init.js” to change the startup behavior.

The simulation toolbar lets you control the simulation:



The “Run” button starts the simulation.
The “Stop” button pauses the simulation. The atomic simulation step is a half-cycle. You can use a single step (“Step” button), or you can adjust the number of half-clocks to step by changing the value in the spinbox in front of the “Step” button.
The “Reset” button clears the history and restarts the CPU. It simulates a valid power-on sequence, running for exactly 8 half-cycles (4 full cycles), after which the RESET pin goes high. You can see that process in a waveform window.
The value on the far right shows the approximate frequency of the simulation.

The Simulation Environment

Z80 machine code runs in a simple simulated environment. The control monitor program, or “Sim Monitor,” contains a 64K RAM buffer that maps into the simulated address space. Programs (in the format of Intel HEX files) are loaded into that RAM and executed. The address to which the files are loaded is specified in a HEX file (see Intel HEX - Wikipedia). For consistency and repeatability, RAM is cleared before loading or reloading HEX data; unsuccessful loads result in the RAM remaining all zeroes (equivalent to NOP instructions).

The “Load hex…” option loads a program in the HEX file format, or you can drag and drop a file onto the Sim Monitor panel.

The “Reload” option reloads the last recently loaded file. This option provides a useful shortcut when you must make frequent changes to a test code and then reload it quickly.

After loading or reloading a program, you need to reset the CPU.

The monitor provides the following services **to the simulated code** by means of a specific memory-mapped area starting at the address 0xD000. You can write Z80 assembly code and use these triggers from within that code. The “test” subfolder contains several Z80 test programs and the “zmac” assembler (for Windows) to create Intel HEX files suitable to load into the simulator. You can, for example, arm NMI to happen and stop the simulation, and then you can observe what happens inside the CPU. Similarly, you can single-step by half-clock and observe how the CPU accepts and handles events.

There are two ways to interact with the simulation monitor:

1) Via direct memory access – these offsets are defined in the “trickbox.inc” include file (residing in resource test folder):

Memory Address	R/W	trickbox.inc	Description
0xD000	W	tb_stop	Writing any value to this address stops the simulation
0xD002	R/W	tb_cyc_stop	Half-cycle number at which to stop the simulation
0xD004	R	tb_cyc_low	Current clock half-cycle number (low 16 bits)
0xD006	R	tb_cyc_high	Current clock half-cycle number (high 16 bits)
0xD008	R/W	tb_int_at	Non-zero cycle number at which to assert INT pin
0xD00A	R/W	tb_int_pc	Non-zero PC address at which to assert INT pin
0xD00C	R/W	tb_int_hold	Number of cycles to hold INT asserted (default is 6)
0xD00E	R/W	tb_nmi_at	Non-zero cycle number at which to assert NMI pin
0xD010	R/W	tb_nmi_pc	Non-zero PC address at which to assert NMI pin
0xD012	R/W	tb_nmi_hold	Number of cycles to hold NMI asserted (default is 6)
0xD014	R/W	tb_busrq_at	Non-zero cycle number at which to assert BUSRQ pin
0xD016	R/W	tb_busrq_pc	Non-zero PC address at which to assert BUSRQ pin
0xD018	R/W	tb_busrq_hold	Number of cycles to hold BUSRQ asserted (default is 6)
0xD01A	R/W	tb_wait_at	Non-zero cycle number at which to assert WAIT pin
0xD01C	R/W	tb_wait_pc	Non-zero PC address at which to assert WAIT pin
0xD01E	R/W	tb_wait_hold	Number of cycles to hold WAIT asserted (default is 6)
0xD020	R/W	tb_reset_at	Non-zero cycle number at which to assert RESET pin
0xD022	R/W	tb_reset_pc	Non-zero PC address at which to assert RESET pin
0xD024	R/W	tb_reset_hold	Number of cycles to hold RESET asserted (default is 6)

When accessing those simulation control addresses, use only 16-bit wide loads and stores (“ld (**),hl”). Reading to or writing from *any other address* simulates regular RAM behavior.

Output pins (INT,NMI,BUSRQ,WAIT and RESET) can be programmatically asserted (set to 0) either by specifying the clock cycle at which to trigger them (write to a “tb_xxx_at” word) or by setting the PC address at which to trigger (write to a “tb_xxx_pc” word). You can use one of those two ways, but not both.

2) Via IO address space access (using “out” instructions):

INs and OUTs to an IO address behave as if the IO space is another 64K segment of writable memory, so an IN from an IO address ‘n’ returns a value that a previous OUT might have set to that address ‘n’; otherwise, it reads as 0xFF by default. Such behavior lets a test program pre-set certain IO locations for future reads.

The IO memory map is set to all 0xFFs before a new program loads.

Two addresses are treated in a special way to enable the use of short Z80 in/out instructions “in a,(n)” and “out (n),a”. The high address byte is ignored when the low IO address byte is 0x80/0x81.

IO Address	IN/OUT	Description
`0x80`	OUT	Write a character to terminal. If the character is ASCII 4 (EOT or End-of-Transmission), stop the simulation
`0x81`	OUT	Byte to be presented on the data bus during the interrupt sequence in IM0 and IM2 modes

Command Window and Scripting

Command Window provides an interactive interface to the JavaScript-based back end. It exposes selected methods internal to the application. Although you do not need to use this interface to work with the application at the basic level, scripting allows you to do more in-depth analysis.

The command editor keeps the history of your commands, which can be retrieved by pressing the cursor up or down keys. Hit the ESC key to clear the command line (or click on the "X" icon on the right side of the input line). Pressing the PgUp key shows the history buffer content in the application log window.

Drag and drop any JavaScript (with the extension ".js") file onto the Command Window to load it. That is equivalent to issuing 'load("filename.js")' command.

Here is a list of base commands:

`load("file.js")`	Loads and executes a JavaScript file ("script.js" if no name is provided)
`run(hcycles)`	Runs the simulation for a given number of half-cycles of the clock; use 0 to run until explicitly stopped
`stop()`	Stops the running simulation
`reset()`	Resets the simulation state
`t(transistor number)`	Shows a transistor state Example: t(8609)
`n(net number or "name")`	Shows a net state by net number or net "name" Example: n("t5") or n(512)
`eq(net)`	Computes and shows the logic equation that drives a given net Example: n("t5") or n(512)
`print("string")`	Prints a string message
`relatch()`	Reloads all custom latches from "latches.ini" file
`save()`	Saves all changes to all custom and config files

The following commands relate to the execution monitor module and operate on the virtual machine within which a simulated Z80 is running:

Object "monitor"	Methods
`mon.loadHex("filename")`	Loads a HEX file into simulated memory, which will be cleared before loading
`mon.loadBin("filename", address)`	Merges a binary file into sim memory at the given address. Memory will *not* be cleared before loading (allows for merging data).

mon.saveBin(“filename”, address, size)	Saves the content of the simulated memory to a file, starting at the given address, and saving “size” bytes
mon.echo(ascii)	Echoes ASCII code to the monitor output terminal
mon.echo(“string”)	Echoes string to the monitor output terminal
mon.readMem(addr)	Reads a byte from the simulated memory
mon.writeMem(addr,value)	Writes a byte to the simulated memory
mon.readIO(addr)	Reads a byte from the simulated IO space
mon.writeIO(addr,value)	Writes a byte to the simulated IO space
mon.stopAt(hcycle)	Stops the simulation at a given half-cycle number
mon.breakWhen(net,value)	Stops the simulation when a given net number becomes 0 or 1
mon.set(“name”,value)	Sets an output pin (*) to a value
mon.setAt(“name”,hcycle,hold)	Activates (sets to 0) an output pin (*) at specified half-cycle, and holds it for the number of half-cycles
mon.setPC(“name”,addr,hold)	Activates (sets to 0) an output pin (*) when PC equals the address, and hold it for the number of half-cycles
mon.enabled = 1|0 true|false	(variable) Enables or disables monitor’s memory mapped services at the address 0xD000
mon.rom = value	(variable) Designates the initial number of bytes for the read-only memory region; default 0. Example: monitor.rom = 8192 will designate 0-8191 as non-writable region

* Output pins: "int", "nmi", "busrq", "wait", "reset"

The following commands interact with the image window of the main application (not with any extra image views):

Object “image”	Methods
img.setLayer(“id”)	Sets the layer id (“1”...”k”)
img.addLayer(“id”)	Adds the layer id (“1”...”k”) to the one(s) already set
img.setZoom(value)	Sets the zoom value (from 0.1 to 10.0)
img.setPos(x, y)	Moves the image to coordinates; x: 0 – 4700, y: 0 – 5000
img.find(“feature”)	Finds and shows the named feature Ex. img.find(“260”)
img.show(x,y,w,h)	Highlight a rectangle at given coordinates and width, height. You can read those values in the log window after selecting an image area using the mouse.
img.state()	Prints the current image view position and zoom to the log window as a command string suitable to copy and use later to restore the exact image view
img.annot(“filename.json”)	Loads a custom annotation file to all image views

When the application starts, it loads a JavaScript startup script called “init.js”, which resides in the resource folder. That script defines a few helpful functions and loads a “Hello, World” Z80 program. Run it by clicking the toolbox’s “Run” button.

The init script also redefines monitor and image class methods into more straightforward JavaScript functions, so it is only optional to type the object prefix. For example, you may write “state()” instead of “img.state()”. That only works for functions, not for variables.

Ensure the JavaScript contains no long-running code since the scripting engine blocks the rest of the application from running. That does not include running the simulation.

Notes and Tidbits

The application stores its settings (windows positions, sizes, …) in the Windows registry at this path:
`HKEY_CURRENT_USER\Software\Baltazar Studios, LLC\Z80Explorer`
On Linux, it stores them in a folder under your user’s home:
`~/.config/Baltazar Studios, LLC`

In this order, hitting the ESC key clears highlighted nets, “driven” nets, and the selected net.

The application supports touch and multi-touch devices. You can scroll (drag) and pinch-to-zoom on images and graphics views.

The application tries to detect latches in the netlist by a simple heuristic of finding two adjacent nets gating each other. While that alone detects many latches, many are still undetected. Hence, the resource file “latches.ini” provides a way to define additional latches as you discover them. It specifies a list of two transistor numbers for each additional latch. See the file itself for an example of how to specify a latch. You can edit it while the application is running. After changing it externally, you must reload it by typing “relatch()” in the script command window.

Everything in the simulation is measured in half-cycles: the high and low clock are two distinct states. The documentation, however, interchangeably uses the terms “half-cycle”, “hcycle” and even “cycle” for brevity.

List of Resource Files

Many resource files have a convenient JSON format (a text file format), and you can edit them.

File	Use
annotations.json	List of annotations and their properties
annot_internals.json	Alternate annotation set; load by dropping onto the Image view
annot_functional.json	Another annotation set; showing major functional buses and nets

`colors.json`	`List of color definitions, filters and matching methods`
`init.js`	`Startup script file`
`latches.ini`	`List of additional latches beyond those that are auto-detected`
`netnames.js`	`List of custom, added, net and bus names`
`tips.json`	`List of user net tips`
`watchlist.json`	`List of nets that are being watched / tracked`
`waveform-*.json`	`State of each of the four waveform windows`

Known Issues

JavaScript engine:
Ensure the JavaScript contains no long-running code since the scripting engine blocks the rest of the application from running. That does not include running the simulation.

Schematic view:
Visual anomalies: (1) does not display tip text for latches, (2) longer tip texts might get clipped
Functional issues: Logic tree parser is a work in progress; the application may decode some nodes incorrectly. Always check.

Credits

This application heavily builds on the work done by the Visual6502 team: Chris Smith, Ed Spittles, Pavel Zima et al.: http://www.visual6502.org
The Visual 6502 team also created main Z80 image layers and many initial net names.