Z80 Explorer is a Z80 netlist-level simulator capable of running Z80 machine code and an educational tool with features that help reverse engineer and understand this chip better.

This document describes version 1.02 of the tool.

Contents

IMPORTANT: Before running the application

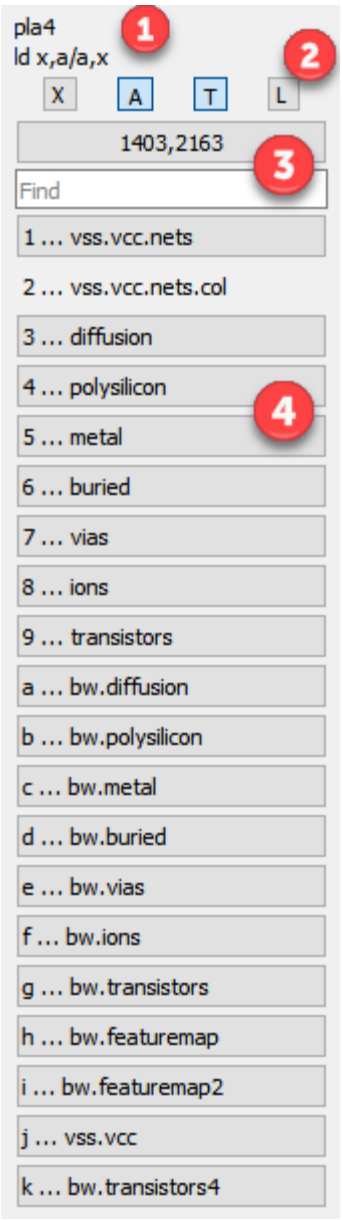
The application is separate from the Z80 resources it uses.
Download Z80 resources from here: https://github.com/gdevic/Z80Explorer_Z80
Extract two 7z files: “layermap.7z” and “segvdefs.7z”. On Windows, use any of the many 7z utilities and on Linux, use “p7zip -d layermap.7z segvdefs.7z”.

When you run the application for the first time, it may ask you to select the folder where these resources are located. Pick that folder.

Image View

The main application window shows a view of the original, NMOS, Z80 chip die layers. This view lets you see various chip features like nets, transistors and vias.
You can open up to 4 additional image views as separate windows, each showing you a different combination of layers, by pressing Ctrl+Q, or via the application menu.
It also lets you compose several layers into one so you can see the traces and vias in different combinations. This is like an X-ray view of the chip die.

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



(1) As you move the mouse over chip features on the image, this section will show you some of the feature's information.
(2) Four toggle buttons enable/disable features on the image (see below)
(3) Shows the mouse coordinates on the image and a Find option to look for the nets, symbols or transistors
(4) Shows the list of available images which you can select or combine by pressing the corresponding key

You can combine images by holding the CTRL key while pressing another key associated with a 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 it has nets and bus coloring applied to it. The coloring is described in a 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.

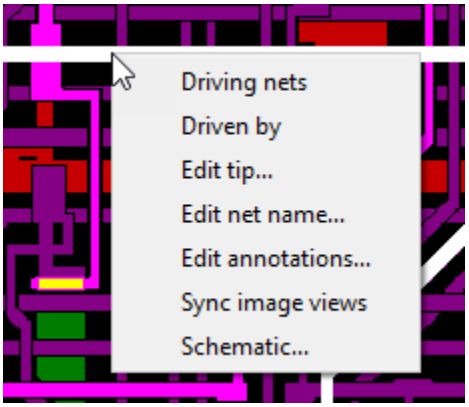
By clicking on the coordinates button (above the "Find") (3), you can enter the image coordinates and center the image at the exact location.

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. Keyboard shortcut is SPACE.
[T] – Highlight (in yellow) all active transistors. 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. 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, will flash on the screen and stay highlighted. Buses will have only their first net highlighted. Press ESC once to clear the highlight; press ENTER key to flash the last highlight again.

As you navigate over the image, double-click on a net to select it, and then right-click to open a context menu. The menu will show different options based on the selected net or the mouse context.



These options are described in the sections that follow.
Sync image views option will sync all additional image views (opened by Ctrl-Q) to the same layer location and zoom factor.

These are the keyboard hotkeys for the main application:

Main window keyboard assignments	
Ctrl + Q	Open a new Image View window (to up to 4)
Ctrl + W	Open a new Waveform View window (to 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:

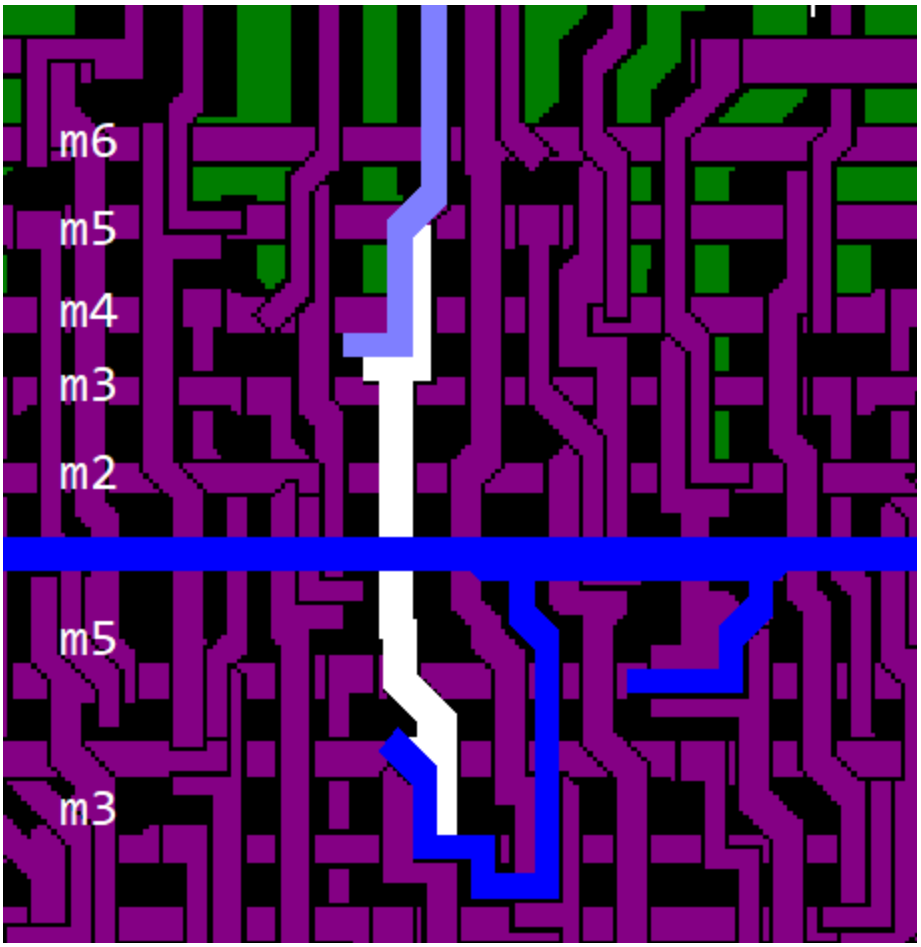
Image view keyboard assignments	
1 … 9 …	Selects one of the images
CTRL + 1 … 9 …	Adds (composite XOR) selected image on top of the previous one(s)
F1	Cycles zoom modes: Fill, Fit, Identity (1:1), Scale
Cursor arrows	Pan up/down/left/right (also use mouse to pan)
PgUp/PgDown	Zoom (also use mouse wheel 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)

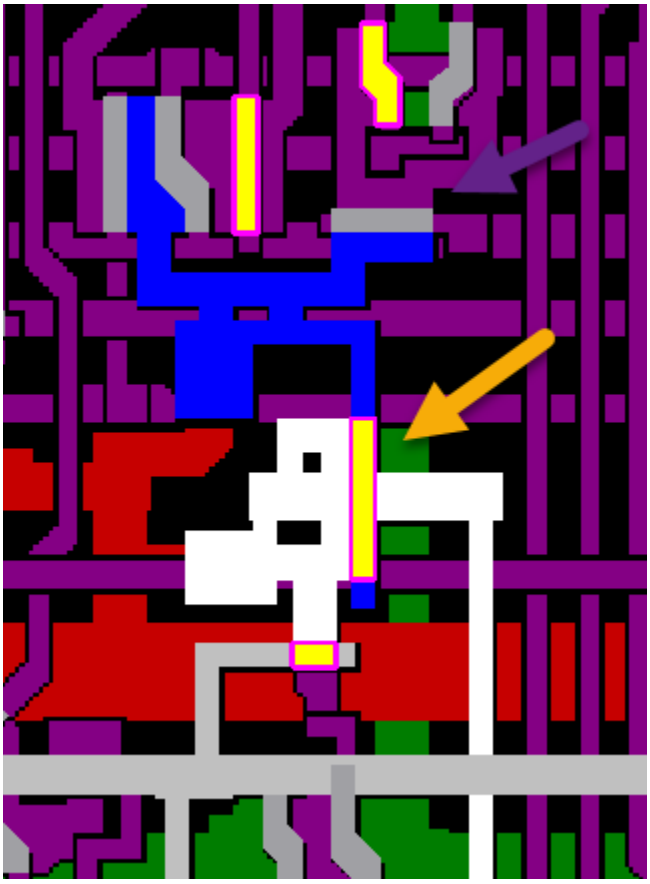
Driving/Driven

"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 very first nets adjacent to the transistors and will disregard serially laid-out gates such is NAND, for example, but they are great for quickly tracing the signals.
The relevant nets, as they are being identified, are marked with increasingly lighter color of 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 showing "Driven by" nets, in blue, while the primary net (3105) is highlighted in white, to have the best contrast.



A picture below shows active transistors in yellow and inactive in gray. To show the transistors, press the “T” key. Sometimes it is useful 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. Clock net will always be colored gray.



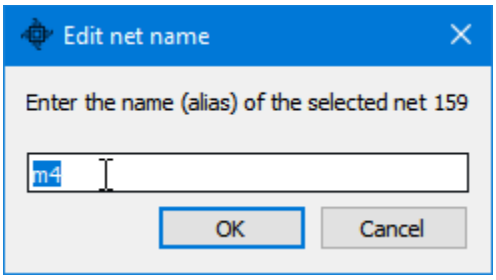
Hint: Un-select a net by double-clicking on an empty (black) space in between the nets, or hit the ESC key to progressively deselect.

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 which has a tip assigned to it, the tip will be shown as a mouse tooltip. Several nets have tips predefined in the default Z80 resource file. One of the most useful ones are the PLA signal wires for which the tip shows the opcode group that a PLA wire decodes.

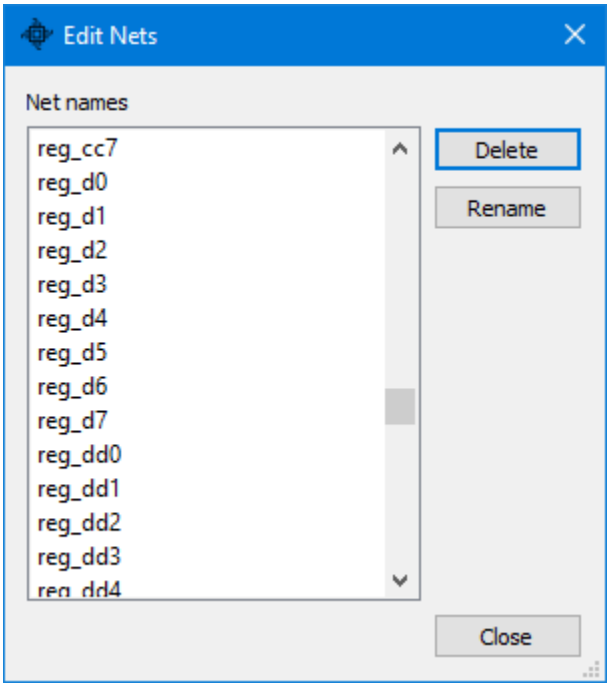
Net Names

Most of Z80's nets are un-named. They can be identified only by their net number. This application requires that the nets being tracked (watched) in the simulation history and waveform graph, be named. Use this context menu to quickly name - and rename - a selected net. You can also delete the net name if you clear the input field and click OK.



Changes to a net name will propagate to any waveform view where it might have been used. Naming a net temporarily is a frequent operation as you are figuring out what a net does (and 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 you can rename it to something more descriptive.

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



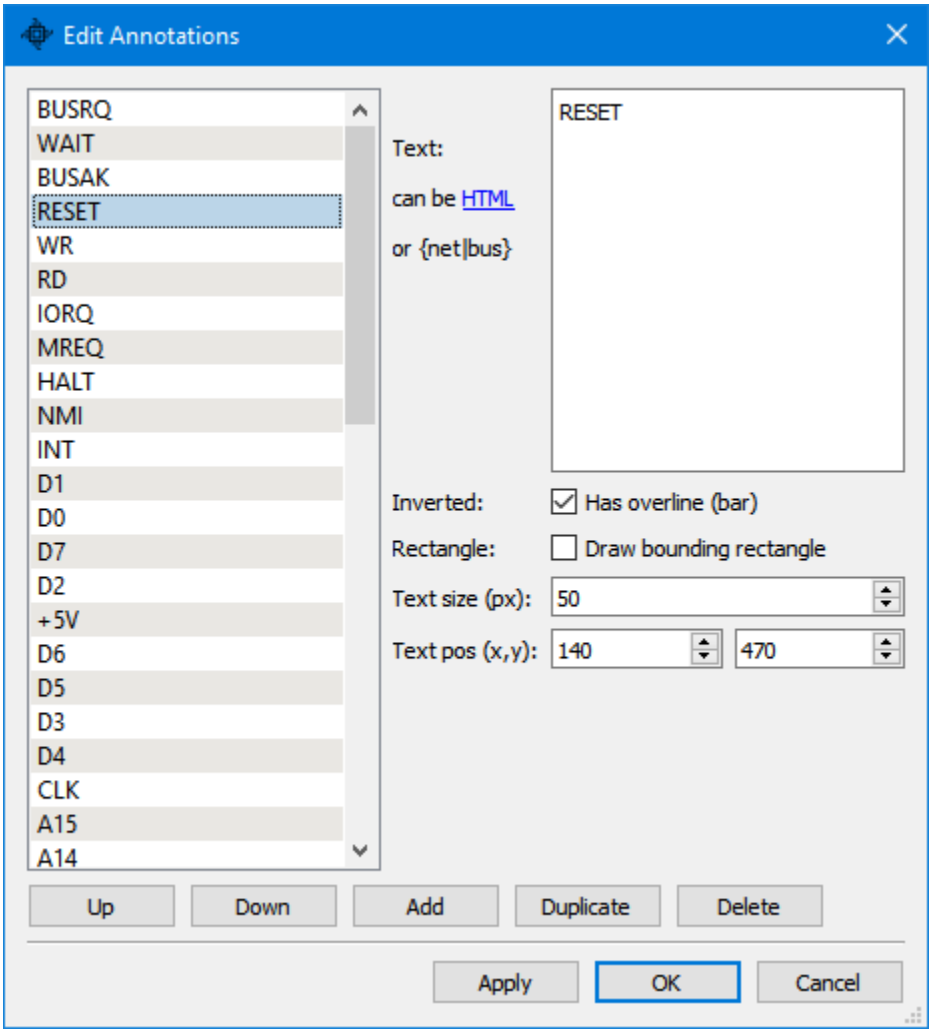
You can delete one or more net names or you can rename each individual net.

Adaptive Annotations

Custom (or user) annotations are the 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, application shortcut key F5 will open the Edit Annotations dialog.
If you hold down a Shift key while making a selection, the selection rectangle will snap 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 chose the option "Edit Annotations…", the dialog will open with all the annotations within your selection box, all selected. Annotations can also be positioned outside the image area. Those will always be shown and will not fade out as you zoom.

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



The annotation text itself can also be made of a good number of HTML-style tags (or Markdown formats). This reference website shows supported tags: https://doc.qt.io/qt-5/richtext-html-subset.html; you always can click on the hyperlink "can be **HTML**" within the dialog to open that reference web page in your browser.
"Inverted" or "overline" option is useful to tag pins with inverted input: it places a bar on top of the name.
Rectangle, if enabled, will draw 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 of several annotations to the same value.

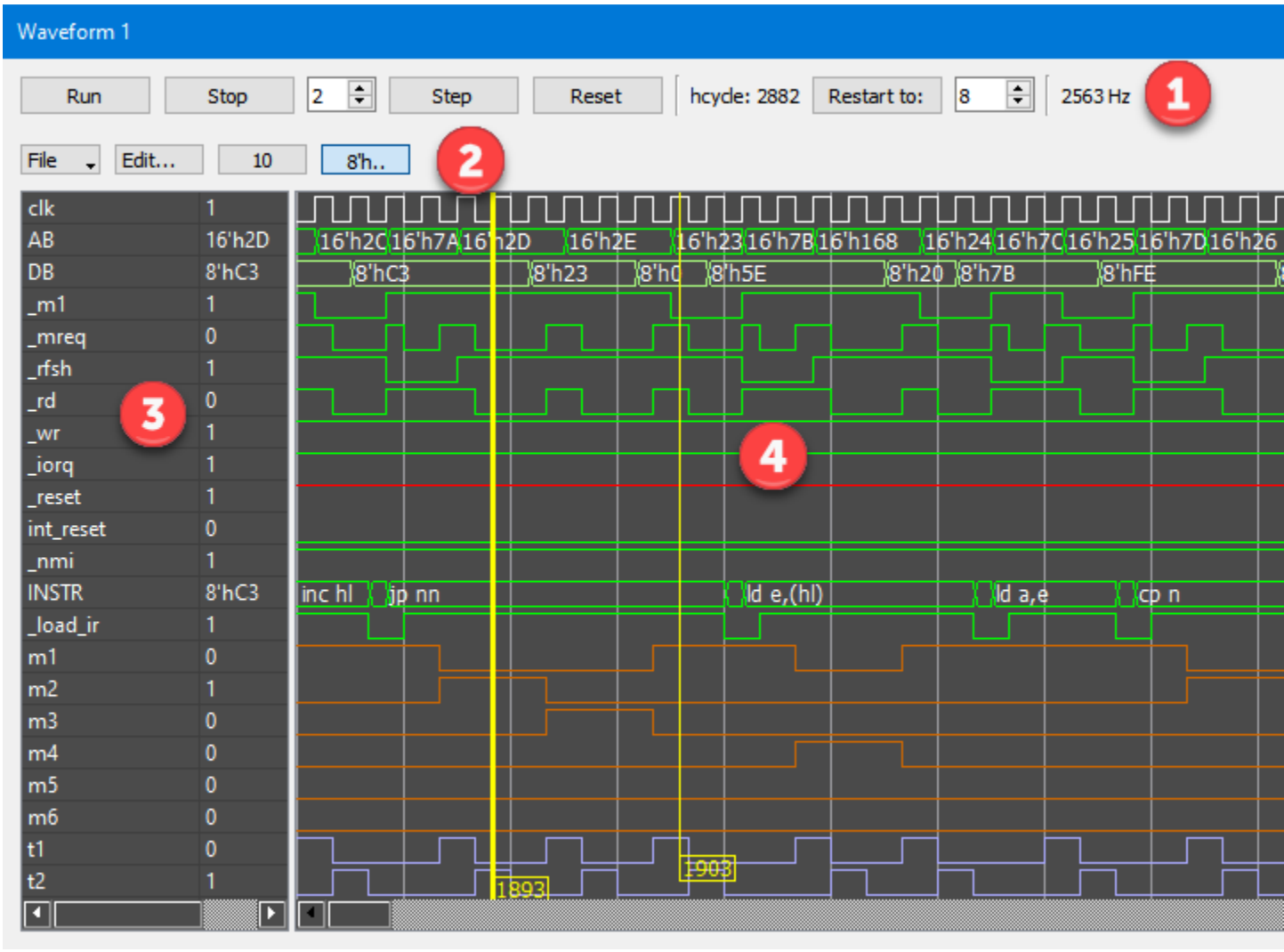
You can have more than one set of annotations. While the default set will load on application startup ("annotations.json"), you can simply drag and drop onto the Image View a file containing another set of annotations. From there on, any edits will be saved into that, another file. For example, "annot_internals.json"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

Alternatively, you can use script command, img.annot("file name") to load any annotation json file.

In addition to being HTML-aware, annotation text supports macros which are simple substitutions of named nets and buses to 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 will show its current value.
A resource file, "annot_functional.json", is an example of such '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 will be inverted and ~ will be shown to make that unambiguous. This output format is consistent to the Waveform View's "Ones' Complement" format.

Waveform View

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



Different sections of the waveform window are:

1. Simulation toolbox (the same simulation toolbox is also present on the Image view windows).
2. File menu lets you save and load custom views (the list of nets and buses that you are observing along with their display formats). You can create a number of different views and load them as you need to. You can also export, save the image as a PNG file.
   Edit button opens the Edit dialog described below.
   The push button next to Edit shows the distance, measured in half-clock values, between the two cursors. If you click on that button, the two cursors will be linked together and sliding one will also move the other one along.
   The button marked with "8h'.." toggles bus value decorations, the bus width (in Verilog format), on and off.
3. The list of nets and buses that you are watching.

4. The history of all nets and buses that you are watching. There are 2 cursors available which you can position at any point to readout the net values.

Use the mouse to work with the waveform window:

- Scroll wheel zooms in and out in the timeline (horizontally)
- Scroll wheel with 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 cursors "flags" are to bring a cursor)
- Hold the cursors and move them

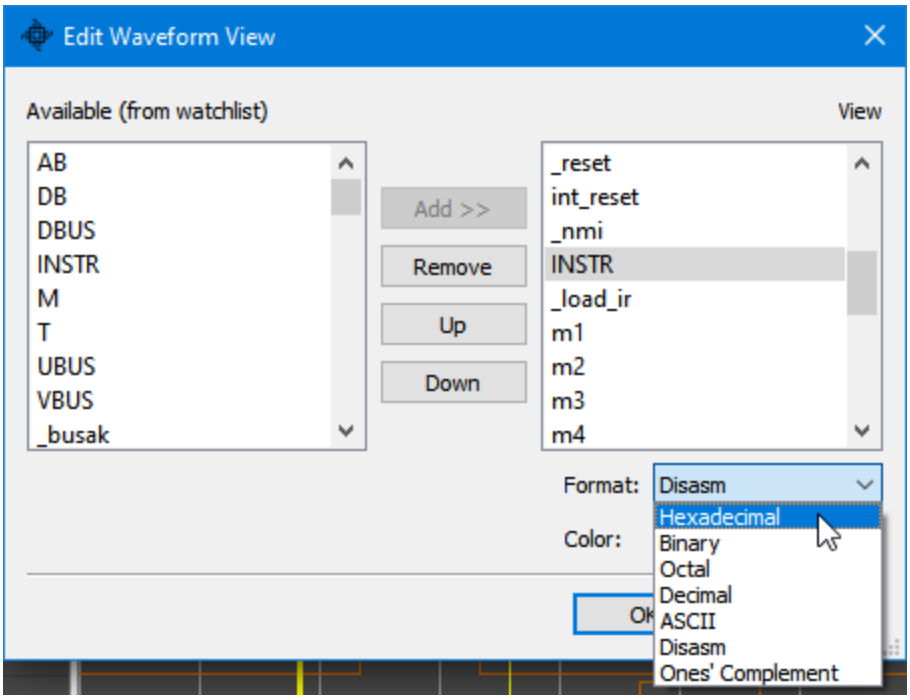
You can have up to 4 separate waveform windows and each will remember its list of nets when you close the application. You can open additional windows by pressing Ctrl + W keyboard shortcut or via the "Window" menu.

There are 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)

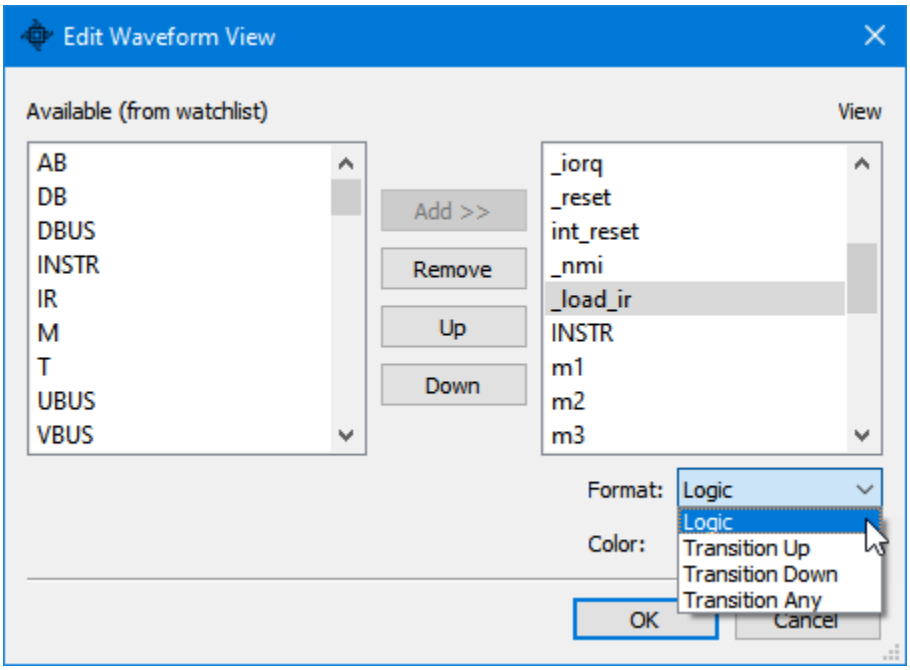
Edit dialog provides a way to select which nets and buses you want to graph and the format and color of each individual item.
This application's convention is that the net names are lowercased and bus names are uppercased. Depending on whether you select a bus or a net, you can choose from different formats.

Buses can be shown in hexadecimal, binary, octal, decimal or ASCII, 1's Complement, and a simple Z80 dis-assembly format (used for instruction register):



One's Complement 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:



Waveform view keyboard assignments:

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

Waveform view mouse actions:

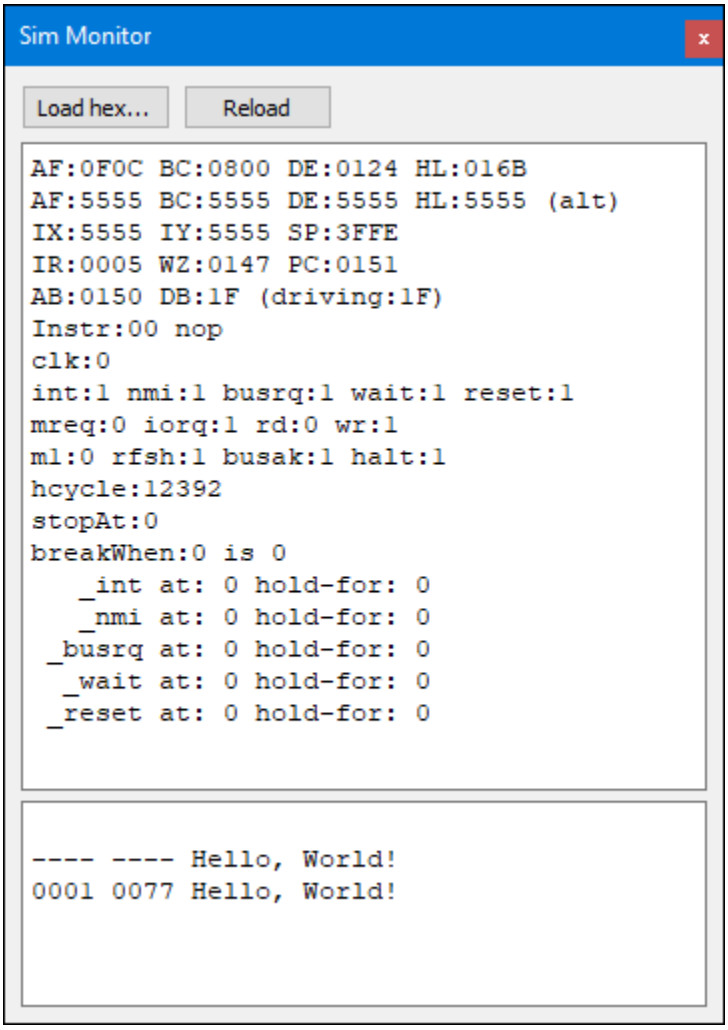
LB hold	Pan left right; move cursors
Wheel	Zoom in and out into the history of data
Wheel + Ctrl	Enlarge and shrink the view vertically

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 “monitor.stopAt()” and “monitor.breakWhen()”. Those commands, when set, will stop the simulation at a specified cycle number and/or when a certain net assumes the specified value. 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.



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

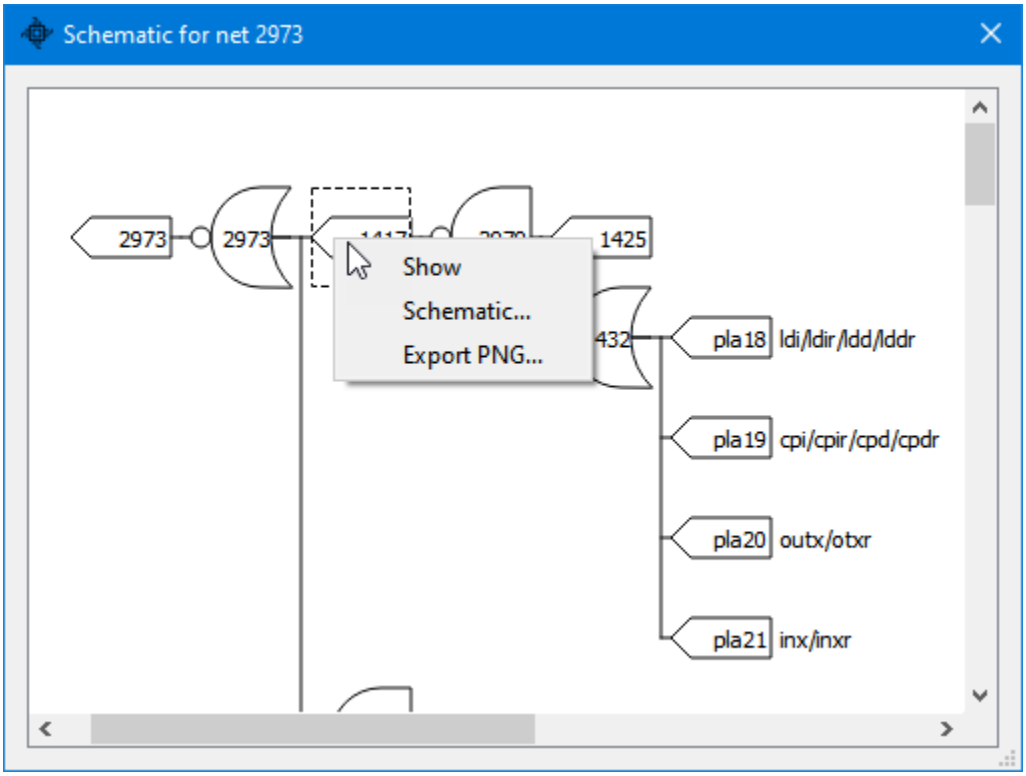
Schematic View

This option shows the net schematic. This is purely experimental and far from perfect; it is a work in progress.
The selected net is being traced back through all the nets that contribute to its state. This traversal ends with certain terminating nets which are chosen as reasonably good end points:

- 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, to prevent possibly infinite loops, the traversal ends when a contributing net has already been processed.
The leaf nodes will display their tip text, if they have any defined.



Use the mouse to pan and zoom the view. When you double-click on a logic gate, the Image view will show its feature location on the die (you may want to zoom out the Image view too see it). The corresponding context menu option is “Show”.
You can create a new schematic view, starting at a selected gate, via the context menu option “Schematic…”.

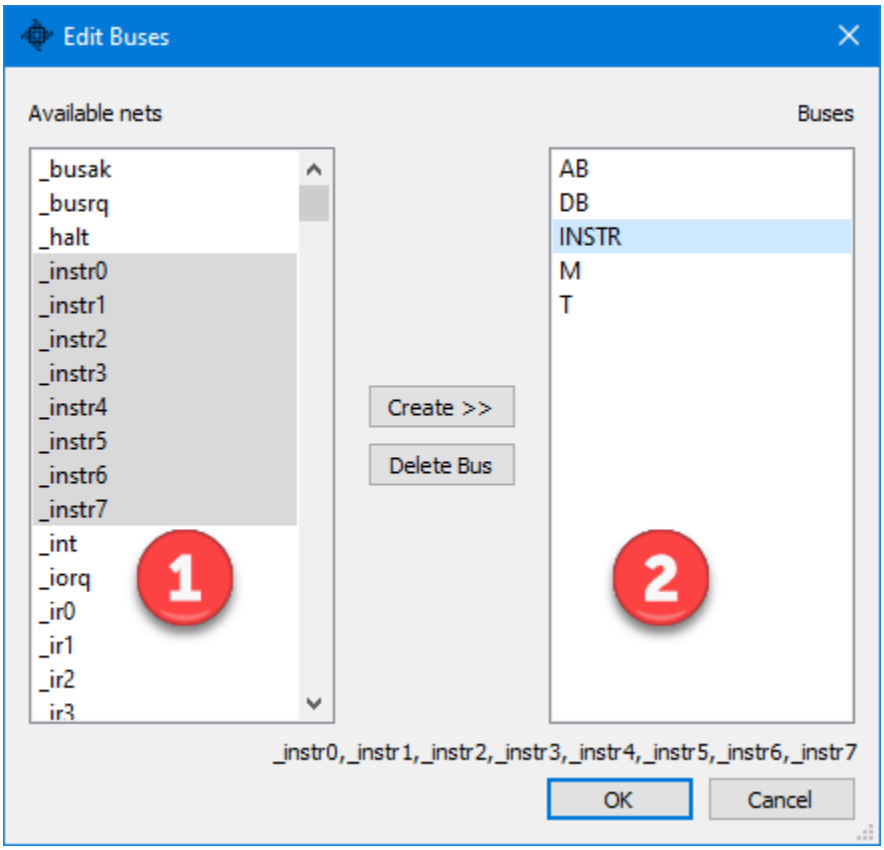
The generated logic network tree is opportunistically compacted: redundant inverters are coalesced with downstream gates if possible (for example, inverter + NOR gate would be collapsed into a single OR gate). This results in a somewhat smaller network. If that is not desirable, you can skip the compacting 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 the window will show “(not optimized”) to confirm your choice.

Note: The logic parser code detects a number of features but it is also fairly generic - and not full proof. 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 "Driven by" option.

Edit 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 created via the Edit Buses dialog. Press F3 at any time (or select Buses from the application's Edit menu) to open this dialog:

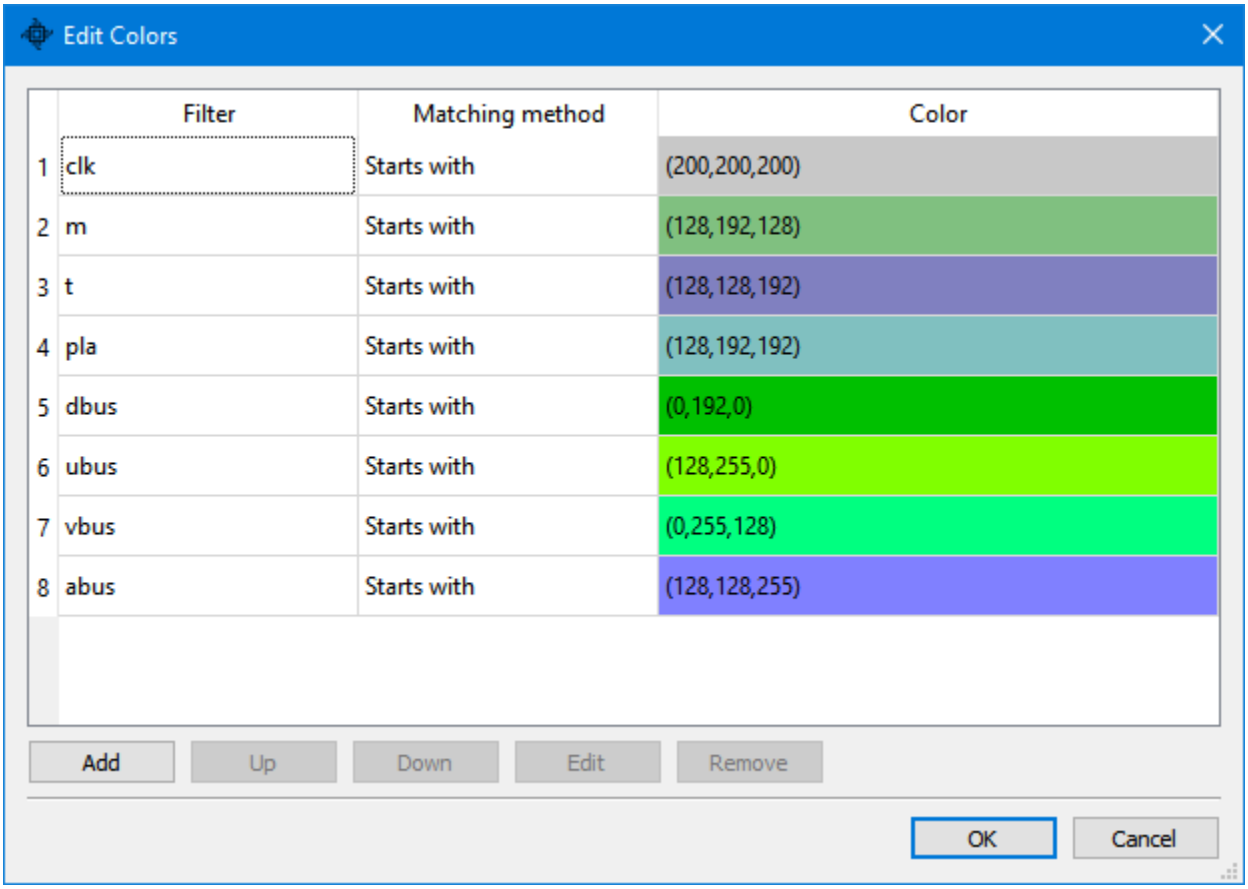


On the left is a list of all named nets from which you can make up a bus. Select two or more signals and click on "Create >>" to create a bus. By the application convention, all buses are uppercased while the other signals (nets) are lowercased.
Currently, you cannot rename a bus; the only way to effectively 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

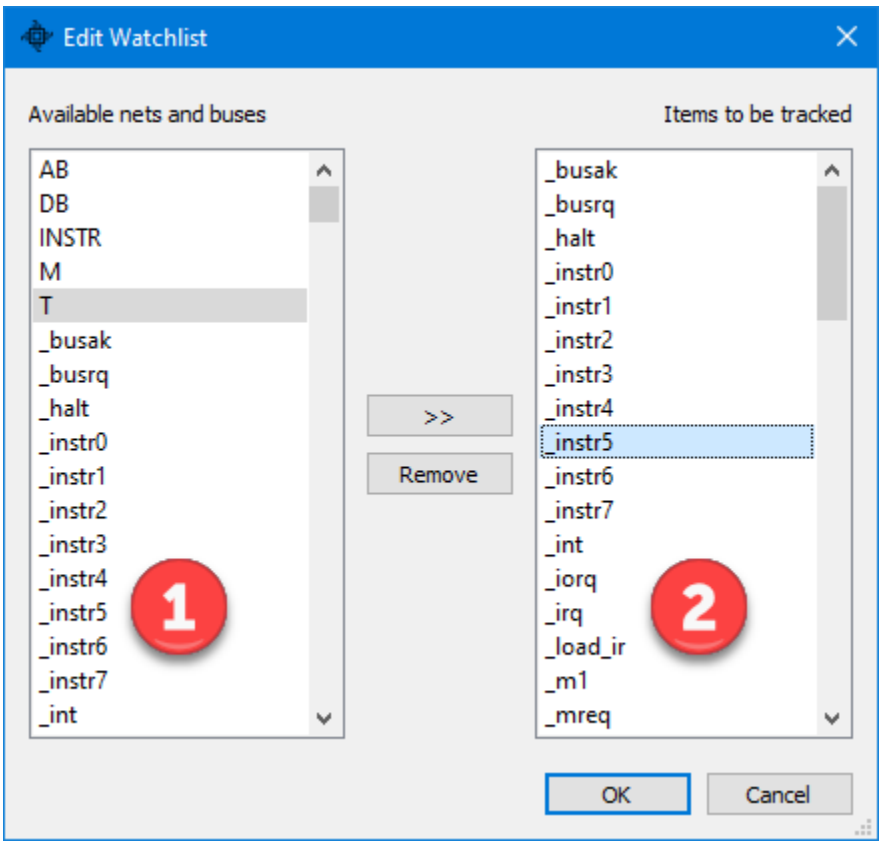


You can select multiple color entries and reposition them in the list, or remove them. Color position is important because a net could be matching more than one color entry, but only the first match with be used.

Edit Watchlist

This dialog has an important function to select which nets will be available to various application parts - like a waveform view - to function properly. Although Z80 processor has almost 8000 nets, it is practical to work with only a small subset of them. In order for a net to

be watched, or tracked, it first needs to 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.

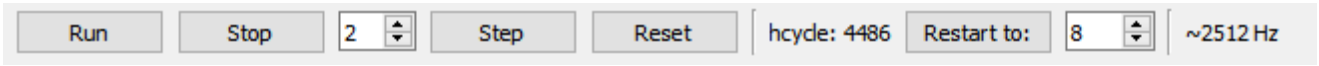


On the left are listed all named (and available) nets that could be tracked and on the right side are currently tracked ones. If you include a bus to be tracked, the program will add all the nets that make up that bus. You can select multiple entries on each side to speed up the process.

Running a Simulation

By default, the application loads a test file, "hello_world.hex" with Z80 code to repeatedly print "Hello, World". That file is loaded into a simulated address space and ready to execute every time you start *Z80 Explorer*. This startup behavior can be changed by editing "init.js" file.

The simulation toolbar lets you control the simulation:



Click on the “Run” button to run the simulation.
Click on the “Stop” to pause the simulation. The basic simulation step is a half-cycle. You can 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.
Clicking on the “Stop” will also stop any running script evaluation.
“Reset” will clear the history and restart the CPU as it were from the power-on and run it for exactly 8 half-cycles (4 full cycles) after which the RESET pin goes high. You can observe that process in a waveform window.
The value on the far right is showing the approximate frequency that the simulation is running.

Simulation Environment

Machine code runs in a simple simulated environment. The control monitor program, or “Sim Monitor”, contains a 64K RAM memory buffer that maps into the simulated address space. Programs (in the format of Intel HEX files) are loaded into that RAM memory 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 memory is completely cleared before loading or re-loading HEX data; unsuccessful loads result in the RAM memory remaining all zeroes (equivalent to NOP instructions).
You can load a HEX file by using the “Load hex…” button, or you can simply drag and drop a file onto the Sim Monitor panel.
In addition, scripting object “monitor” provides extended commands to also load (and save) binary files into the simulated RAM.
Clicking on the “Reload” button will reload the last recently loaded file which is a useful shortcut when you need to make frequent changes to a test code and reload it quickly.

After loading or re-loading a program, you do need to reset the CPU.

The monitor provides the following services **to the simulated code**, by the 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 “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 simulation, then you can observe what happens inside the CPU. Similarly, you can single-step by half-clock time period and observe how the CPU is accepting and handling events.

Via direct memory access – these offsets are defined in “trickbox.inc” include file in resource’s 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)

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

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, at the same time.

Via IO address space access ("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' will return a value that might have been set by a previous OUT to that address 'n'; otherwise, it will read as 0xFF by default. Such behavior lets a test program preset certain IO locations for future reads.

Like RAM buffer, the IO memory map is also cleared to 0xFF before a new program loads.

Two addresses are treated as special to enable use of short Z80 in/out instructions forms "in a,(n)" and "out (n),a". When the low IO address byte is 0x80/0x81, the high address byte is ignored.

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

Scripting

Command Window provides an interactive interface to the JavaScript-based back end.
There are many commands which you can use; some of them are briefly described when you type "help()" command, and others are implemented as part of the functional, class-like interface.
The command editor keeps the history of your commands which can be retrieved by pressing the cursor up and cursor down keys. Hit 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 will show you the content of the history buffer in the application log window.

Here is a list of root commands:

run(hcycles)	Runs the simulation for a given number of half-cycles of the clock; set 0 to run indefinitely
stop()	Stops the running simulation
reset()	Resets the simulation state
t(transistor number)	Shows a transistor state
n(net number or "name")	Shows a net state by net number or net "name"
load("file.js")	Loads and executes a JavaScript file ("script.js" if no name is provided)
relatch()	Reloads all custom latches from "latches.ini" file

In addition, these are the objects that provide additional methods; commands are tied to these object classes:

Object "control"	Methods
control.doRunsim(ticks)	Starts simulation for "ticks" number of half-clocks, 0 to stop
control.doReset()	Resets simulation
control.save()	Saves all changes to all custom and config files

Object “sim”	Methods
sim.hcycle	(variable) Returns the current simulation half-cycle count
sim.hz	(variable) Returns the estimated simulation frequency
sim.eq(net)	Computes and shows the logic equation that drives a given net

Object “monitor”	Methods
monitor.loadHex(“filename”)	Loads a HEX file into simulated memory, which will be cleared before loading
monitor.loadBin(“filename”, address)	Loads a binary file into sim memory at the given address. Memory will *not* be cleared before loading!
monitor.saveBin(“filename”, address, size)	Saves the content of the simulated memory to a file, starting at the given address, and saving “size” bytes
monitor.echo(code)	Echoes ASCII code to the monitor output terminal
monitor.echo(“string”)	Echoes string to the monitor output terminal
monitor.readMem(addr)	Reads a byte from the simulated memory
monitor.writeMem(addr,value)	Writes a byte to the simulated memory
monitor.readIO(addr)	Reads a byte from the simulated IO space
monitor.writeIO(addr,value)	Writes a byte to the simulated IO space
monitor.stopAt(hcycle)	Stops the simulation at a given half-cycle number
monitor.breakWhen(net,value)	Stops the simulation when a given net number becomes 0 or 1
monitor.set(“name”,value)	Sets an output pin to a value (*)
monitor.setAt(“name”,hcycle,[hold])	Activates (sets to 0) an output pin (*) at the specified half-cycle, and hold it for the optional number of half-cycles
monitor.setPC(“name”,addr,[hold])	Activates (sets to 0) an output pin (*) when PC equals the address, and hold it for the optional number of half-cycles
monitor.enabled = 1|0 true|false	(variable) Enables or disables monitor’s memory mapped services at the address 0xD000
monitor.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"

Object "script"	Methods
script.response("string")	Writes a string to the scripting terminal
script.exec("cmd")	Executes a scripting command

Object "img" works with the image on the main app window, not with any extra image views that may be opened.

Object "img"	Methods
img.setImage(num)	Sets the image number
img.setImage(num,1)	Adds the image number 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 the JavaScript startup script called "init.js", which should be present in the Z80 resource folder. That script loads a "Hello, World" Z80 program. You can immediately run it by clicking on the toolbox's "Run" button.
You can load your own JavaScript files by using the "load()" command.
If your script unexpectedly executes for too long, or it gets stuck in a loop, clicking on the toolbox's "Stop" button will kill any currently running script.

Notes and Tidbits

Application settings (windows positions, sizes, …) are stored in the Windows registry at this path:

`HKEY_CURRENT_USER\Software\Baltazar Studios, LLC\Z80Explorer`
On Linux, they are stored in this folder under your user's home:
`~/.config/Baltazar Studios, LLC`

Hitting the ESC key will clear, in this order: highlighted nets, "driven" nets and selected net.

On a high-DPI monitor you can either run a batch file "highDPI.bat" or use Windows 10 application menu: Compatibility -> Change High DPI settings -> Override (on the bottom) -> System (Enhanced), whichever gives you better results.

Touch and multi-touch devices are supported by various views handling the scroll (drag) operations and also pinch-to-zoom.

Application tries to detect latches in the netlist by a rather simple heuristic of finding two adjacent nets gating each other. While that alone detects a fair number of latches, many are not detected. Hence, the resource file "latches.ini" provides a way to define additional latches as you discover them. Simply specify two transistor numbers that make up a 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, reload it by typing "relatch()" in the script command window.

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

List of Resource Files

As mentioned at the beginning, Z80Exploer is using a set of resource files which needs to be downloaded and may need to be periodically updated from a git repository. These resources broadly fit into two groups: Z80 chip data and Z80Explorer application state. This section describes the latter group.

Many resource files are kept in a convenient JSON format (which is a text file format) and can easily be edited by hand. For most of these files, there is rarely such a need since the application provides a UI to 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

Only “init.js” and “latches.ini” files could be safely edited while the application is running. Other files may be overwritten on the application exit.

Known Issues

The simulation run-count is 32-bit wide and will wrap to 0 and stop simulation if left running for a number of days (around a week on my PC). Simply click on the “Run” button to continue the simulation.
Schematic view:
Visual anomalies: (1) will not display tip text for latches, (2) longer tip texts might get clipped
Functional issues: Logic tree parser is a work in progress; 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
All Z80 image layers are also from the Visual 6502 team as well as many initial net names.

Revision History

Since you need to match the software version with the data repo, and perhaps also update your own tests and other files, this revision history outlines the most important changes.

Version 1.02

- Trickbox: changed fields names from tb_int_**len** to tb_int_**hold** for all 5 pins (int,nmi,busrq,wait and reset.)
- Trickbox: added another way to assert input pins by matching a PC register value.
- IO address to write a character out has changed to a shorter form 0x80. All relevant tests have been updated.
- New, short, IO address of 0x81 allows to set the byte to output to data bus during the interrupt sequence. Also, a new relevant test has been added, “test_ints.asm”.