WaveDrom

Rendering Beautiful Waveforms from Plain Text

Aliaksei Chapyzhenka
Jonah Probell

www.wavedrom.com

Abstract

WaveDrom is a tool for rendering digital timing diagrams, and other technical visualization, as SVG or PNG images from an intuitive plain text language. It is easy to learn, easy to use, and browser-based. WaveDrom is free open source software.

http://wavedrom.com/

This paper provides simple and complex examples of code and its resulting graphics. Also provided are guidelines on integrating WaveDrom into source code, diffing waveforms, parameterizing waveforms, describing and visualizing constraints, expressing design intent, and generating assertions.

Table of Contents

Introduction

Digital timing diagrams ("waveforms") are the type of diagram that best describe how electronic signals do, or should, behave. Chip designers naturally draw waveforms on whiteboards, and routinely view waves when debugging simulations. When called upon to write a specification, we know our word processing options. However, there has never been a simple, widely used tool for rendering waveforms. That is until now.

WaveDrom is a tool for rendering waveforms as SVG or PNG images.[1] It defines a language that uses textual symbols, which are easy to create with a keyboard, to represent the types of information that waveforms present. WaveDrom follows the software development trend of running applications in browsers for automatic portability across operating systems. The WaveDrom editor (online and offline version) is WYSIWYG. The effect of changes to text describing waveforms is immediately rendered, without a compilation step. Furthermore, since WaveDrom waveforms are described as text, they can be embedded in source code comments, in source control metadata, and diffed between versions.

WaveDrom was first created by this paper's author, Aliaksei Chapyzhenka. It is available online, as desktop application (Windows, Linux, OSX), and open source on github.[2] WaveDrom is well documented online with a fun interactive tutorial.[3] Others have contributed ideas, and WaveDrom has been extended for:

- drawing schematic diagrams
- generating register field documentation
- a translator to TikZ for TeX / LaTeX typesetting[4]

This paper intends to stimulate ideas for other extensions and other uses.

Problems with current practices

Waveforms are a common, well understood, industry standard way to express hardware behavior. When done well, they effectively communicate ideas visually, in a readable way.[5]

Some readers of this paper draw waveforms as ASCII text. Some draw diagrams manually within a word processor's graphics tools. Some copy-paste lines and hexagons in drawing programs. Some cleverly use cell outlining features in a spreadsheet, and some will even write Verilog with #delay

---

[1] Scalable Vector Graphics (SVG) is a vector-based graphics format recognized by many document editing programs. Portable Network Graphics (PNG) is a raster graphics format, recognized by many document editing programs. PNG is comparable to the Joint Photographic Experts Group (JPEG) format.

[2] https://github.com/drom/wavedrom

[3] http://wavedrom.com/tutorial

[4] TikZ is a description format for vector graphics rendered in TeX and LaTeX documents. Describing timing diagrams in TikZ is not intuitive. See http://www.texample.net/tikz/examples/timing-diagram/.

[5] Tufte, Edward. Envisioning Information, 1990, and
Tufte, Edward. The Visual Display of Quantitative Information, 2001.

statements, run a simulation with a trace file, and take screenshots of the traces in a waveform viewer GUI. Various commercial and freeware tools are available for download and installation on PCs to produce waveforms, but none is widely used.

All such methods have important limitations:

1. Not everyone is a visual artist, especially at work, and especially during routine work
2. It takes time to make a good drawing
3. Different authors draw with different and inconsistent styles
4. Drawing waveforms directly intermingles presentation and content
5. It is difficult to version control, change control, and compare waveforms
6. Waveforms are not interactive
7. It is difficult to assemble or concatenate multiple waveforms
8. Waveforms are not parameterized

As a result, designers avoid drawing waveforms where they could be helpful. When designers draw waveforms, they require editors and technical writers to redo the drawings for clarity.

WaveDrom resolves all of these limitations. Furthermore, because WaveDrom uses a textual language it can serve as a modeling language that computers can parse. Designers at several companies use WaveDrom descriptions as part of hardware design entry, design capture, or test case entry. Furthermore, some designers parse simulation traces to generate WaveDrom waveforms for selected signals.

Inspiration

The concept of using textual (markup) language as a source for diagram rendering engine is not new. Many domain specific language exist to drive or hint diagram rendering process. There is full spectrum of language approaches.

One approach is to provide some domain specific computer language that can be utilize standard computer language processing tools like lexer and parser, and processed than as abstract syntax tree. For example DOT to describe graphs or PlantUML to describe UML diagrams.

Another approach is to employs the fact that block of monospace font text can have meaningful alignment that can be understood by the computer program. The ASCII graphics type descriptions like ditaa or shaape rely on fact of 2D alignment to render sophisticated block diagrams with desired spacial relationship of it's parts, and certain color, shape, size effects.

Some of the diagram languages employ standard mark languages (typically XML) to capture structure and content. For example: CML (Chemical Markup Language) For graphic rendering of the molecular structure of chemical compounds, or MathML (Mathematical Markup Language) to describe mathematical notations. JSON is new emerging standard for data object representation replacing XML in many areas.

Exchange format (WaveJSON)

WaveJSON is the compact exchange format used by WaveDrom to render waveforms.[6] It is based on the JSON language syntax. WaveJSON is an Embedded Domain Specific Language.[7] It uses the industry-standard JSON format with additional semantics embedded into the JavaScript object structure syntax.[8]

WaveJSON satisfies the following criteria for a useful waveform exchange format:

- Text format
  - Plain text editor can be used
  - Waveform code can be inserted into HDL or software code as comment
  - Code can be inserted into wiki, markdown, literate, or Asciidoctor documents[9]
- Machine readable
  - JSON data object notation language
  - Has JSON schema: https://github.com/wavedrom/wavedrom-schema that can be used for data validation.
- Domain specific language (DSL)
  - Great expressive power within the domain
  - Exhibits minimum redundancy of domain specific knowledge definition
  - Is an extension of a general purpose language (i.e. JavaScript)
- Open Source format and tools
  - WaveDrom : WaveJSON → SVG rendering engine in JavaScript https://github.com/drom/wavedrom
  - WaveDromEditor : WaveJSON editor for Desktop and Browser https://github.com/wavedrom/wavedrom.github.io

Image rendering

The first and most natural use for WaveDrom is rendering of images for publishing purposes.[10] Here is a code example:

---

[6] The WaveJSON format specification is at https://github.com/drom/wavedrom/wiki/WaveJSON

[7] Mernik, Marjan, Jan Heering, and Anthony M. Sloane. "When and how to develop domain-specific languages." *ACM computing surveys (CSUR)* 37.4 (2005): 316-344.
Knuth, Donald Ervin. *The texbook*. Vol. 1993. Reading, Massachusetts: Addison-Wesley, 1986.
Spinellis, Diomidis. "Notable design patterns for domain-specific languages."*Journal of systems and software* 56.1 (2001): 91-99.

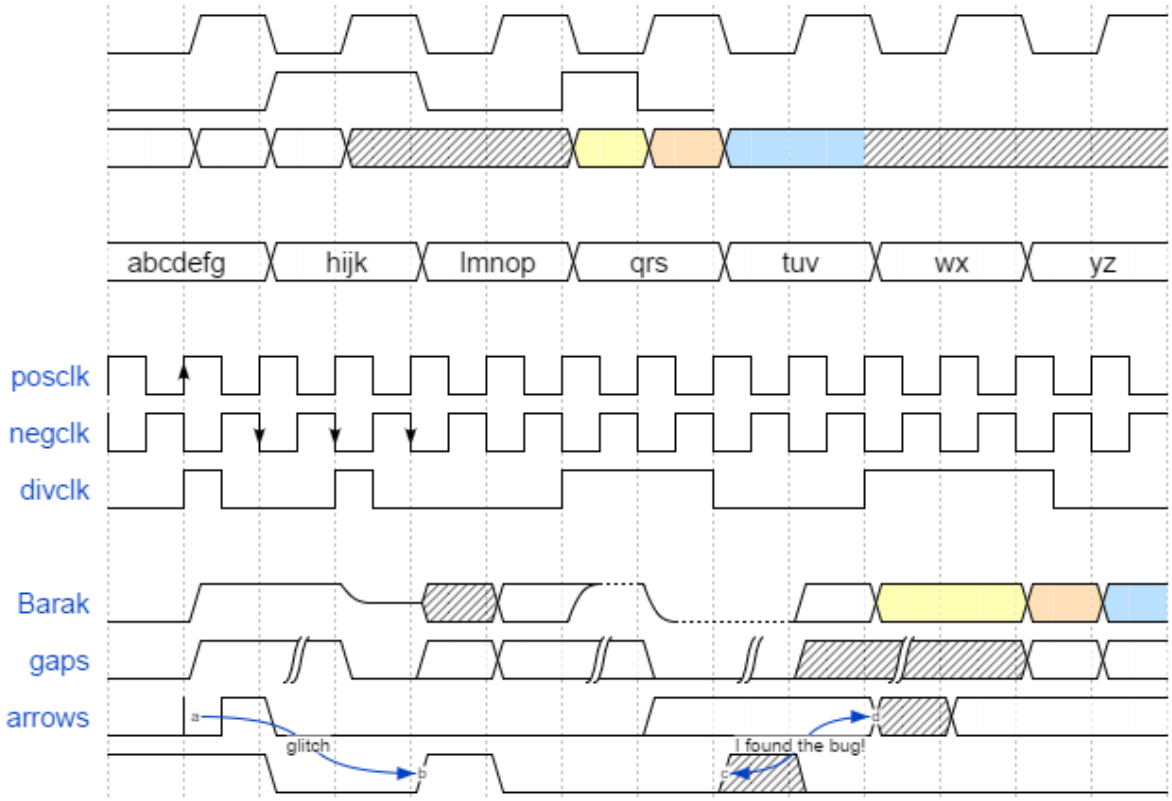
[8] JavaScript Object Notation (JSON) is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. More information at http://json.org/.

[9] Knuth, Donald E. "Literate programming." *CSLI Lecture Notes, Stanford, CA: Center for the Study of Language and Information (CSLI), 1992* 1 (1992).
Asciidoctor diagram extension at https://github.com/asciidoctor/asciidoctor-diagram

[10] The online editor is available at http://wavedrom.com/editor.html.
The desktop application is available at https://github.com/wavedrom/wavedrom.github.io/releases.

```
{signal: [
  {wave: '01010101010101'}, // toggling
  {wave: '0.1.0.hl'},       // dot(.) holds a value, h/l for high and low
  {wave: '222xxx345.6789'}, // multi-bit, X, 345 = colors, 6789 == x
  {wave:''}, // blank line

  // text
  {wave: '2.2.2.2.2.2.2.',
   data:["abcdefg","hijk","lmnop","qrs","tuv","wx","yz"]},
  {wave:''},

  // names and clocks
  {name: "posclk", wave: 'pPp..........' }, // capital letters for arrows
  {name: "negclk", wave: 'n.N..n........' },
  {name: "divclk", wave: 'lplpl.h.l.h.pl' },
  {wave:''},

  // fun
  {name: "Barak",  wave: '01.zx=ud.23.45'},

  // gaps
  {name: "gaps",   wave: '01|022|0|x|.22' },

  // arrows with nodes and edges
  {name: "arrows", wave: '0n0....2..x2..',
                   node: '.a........d' },
  {                wave: '1.0.10..x0....',
                   node: '....b...c'   },
  ],
edge:['a~>b glitch',
  'c<~>d I found the bug!',
  ],
}
```

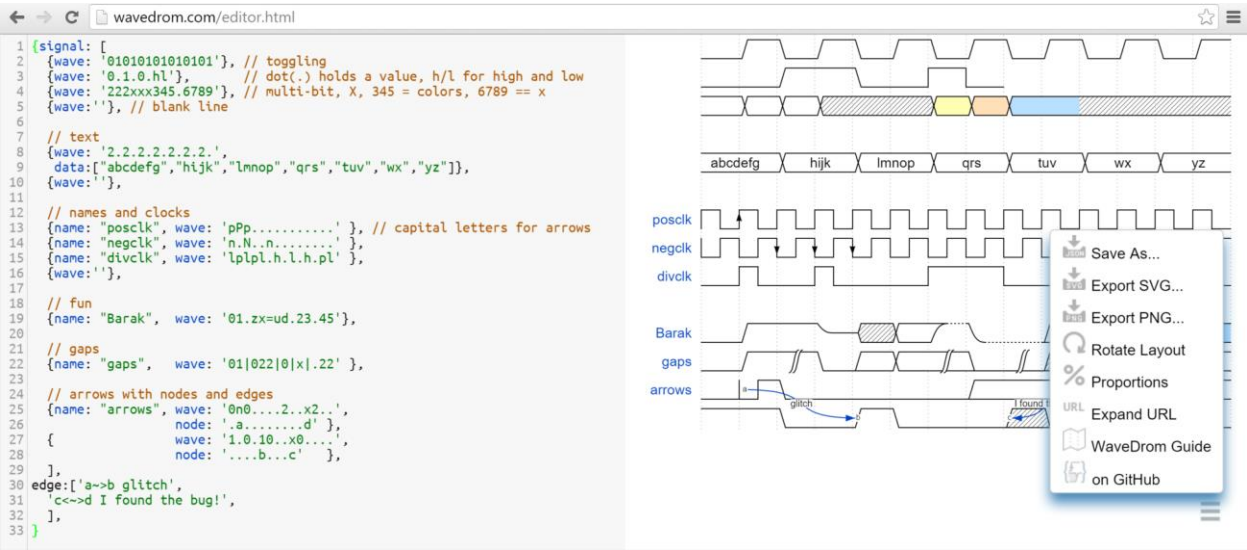
It renders this waveform:



The online tutorial has more examples and more sophisticated examples.

WaveDrom render produces SVG or PNG. Standard skins can be used, but custom skins are supported for specific stylistic formatting. WaveDrom allows text to have XML style attributes.[11] WaveDrom has a two panel editor that renders WaveJSON source into SVG image in realtime. The panels can be rotated to show the source and waveform horizontally or vertically. A user menu is in the lower right.

[11] Package available at https://github.com/drom/tspan

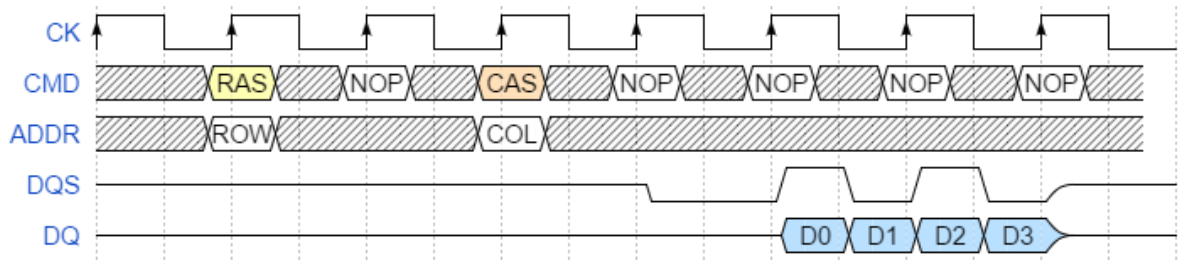


The instant feedback of real time rendering gives a designer an ability to draw the intended waveform and correct mistakes very quickly.

The following source code describes a DDR DRAM access:

```
{ signal: [
  { name: "CK",   wave: "P.......",      period: 2  },
  { name: "CMD",  wave: "x.3x=x4x=x=x=x=x",
    data: "RAS NOP CAS NOP NOP NOP NOP", phase: 0.5 },
  { name: "ADDR", wave: "x.=x..=x........",
   data: "ROW COL",                     phase: 0.5 },
  { name: "DQS",  wave: "z.......0.1010z." },
  { name: "DQ",   wave: "z.........5555z.",
   data: "D0 D1 D2 D3" }
]}
```

Below is the rendered waveform.

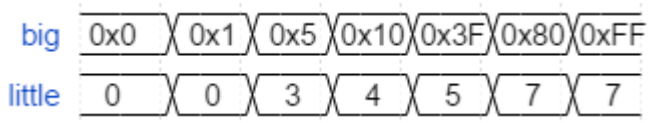


Waveforms in code comments

The following Verilog module code includes a WaveDrom waveform in a code comment:

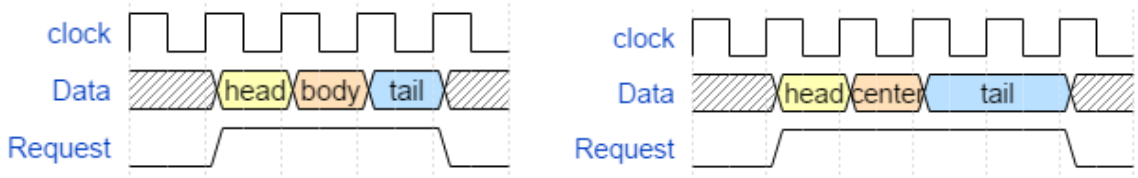
```
module reducer(big, little)
   input  [7:0] big;
   output [2:0] little;

/* Waveform of example expected behavior
{signal: [
  {name: 'big',
   wave: '=======',
   data: '0x0 0x1 0x5 0x10 0x3F 0x80 0xFF'},
  {name: 'little',
   wave: '=======',
   data: '0 0 3 4 5 7 7'},
],}
*/
   always @(*) begin
      little = 0;
      case(big)
         8'h02:  little = 1;
         8'h04:  little = 2;
         8'h08:  little = 3;
         8'h10:  little = 4;
         8'h20:  little = 5;
         8'h40:  little = 6;
         8'h80:  little = 7;
      endcase
   end
endmodule
```

The waveform code can be rendered for purposes of documentation. It would look like this.



Diffing waveforms

Though waveforms are the best way to display timing information in two dimensions, even when placed side by side it is very difficult to visually compare one to another to identify changes or differences. However, by representing waveforms as text, it is much easier to identify the difference between visual waveforms.

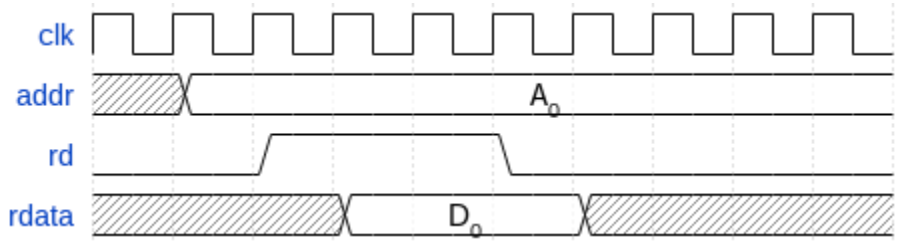


Similarly, when a simulation or logic analyzer dumps a large amount of trace data as text, it is difficult to understand the diff in context. As long as the trace dump is in WaveDrom format, or can be easily converted to WaveDrom format by a script, then rendering the waveform as an image and looking at the point of the difference makes it much easier to understand.

Parameterization

WaveDrom is built on JavaScript. Therefore, it is extensible. The example below is a waveform for a parameterizable memory block. It can return data in the same cycle, or with some latency. A JavaScript functions makes the read latency parameterizable, based on the variable `rdlatency`.

```
(function (o) {
  var rdata = 'x.';
  rdata += '.'.repeat(o.rdlatency);
  rdata += '=..x';
  rdata += '.'.repeat(4 - o.rdlatency);
  return {
    signal: [
      {name: 'clk',   wave: 'p.........'},
      {name: 'addr',  wave: 'x=........', data: 'A<sub>0'},
      {name: 'rd',    wave: '0.1..0....'},
      {name: 'rdata', wave: rdata, data: 'D<sub>0'}
    ]
  };
})({rdlatency: 1}) /* <-- change this */
```



Representing constraints in WaveJSON

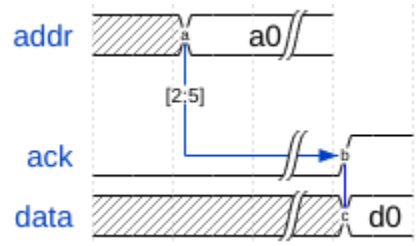
WaveJSON has several features that can be used to describe behavioural constraints in the (human | computer) (readable | writable) language. Therefore, these features can facilitate human-machine interaction:

- [signal[].wave] property represents one character per cycle, so it is possible to represent alignment of several signals.
- Vertical bar (|) character represents gaps in the timing.
- [signal[].node] property allows placing anchors aligned with [signal[].wave] in time.
- [edge] property describes dependencies between the nodes with optional textual attributes.

Consider the following code.

```
{signal: [
  {name: 'addr', wave: 'x=|',  node: '.a..', data: 'a0'}, {},
  {name: 'ack',  wave: '0.|1', node: '...b'},
  {name: 'data', wave: 'x.|=', node: '...c', data: 'd0'}
],
  edge: ['a|->b [2:5]', 'b-c']
}
```

That yields the following waveform.



Several system constraints can be drawn and analyzed from this description by human or computer. For example:

1. one may expect `ack` to rise in 2 to 5 cycles after `addr` is asserted.
2. The `data` signal should carry some payload, and be asserted on the same cycle.

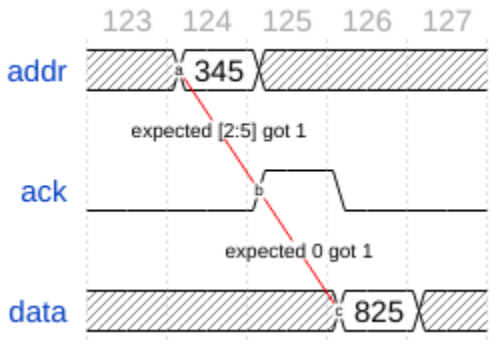
If the designer describes that expected behavior, then a computer can use it to:

- Create a test case, and generate testbench code that behaves like the description, varying parameters on the edges
- Assert the constraint during simulation or for formal analysis
- In the case of a mismatch (case failure), a computer can back-annotate the waveform, showing the difference with the actual behaviour, and use it for reporting.

A report may contain multiple failing cases that look like this:

```
{signal: [
  {name: 'addr', wave: 'x=x..', node: '.a..', data: '345'}, {},
  {name: 'ack',  wave: '0.10.', node: '..b'}, {},
  {name: 'data', wave: 'x..=x', node: '...c', data: '825'}
],
  edge: ['a--b expected [2:5] got 1', 'b--c expected 0 got 1'],
  head: {tock: 123}
}
```

rendered form:



The report may have a timestamp or cycle number in a waveform header or footer section as a reference to the simulation time.

Specific edges may contain more failure information.

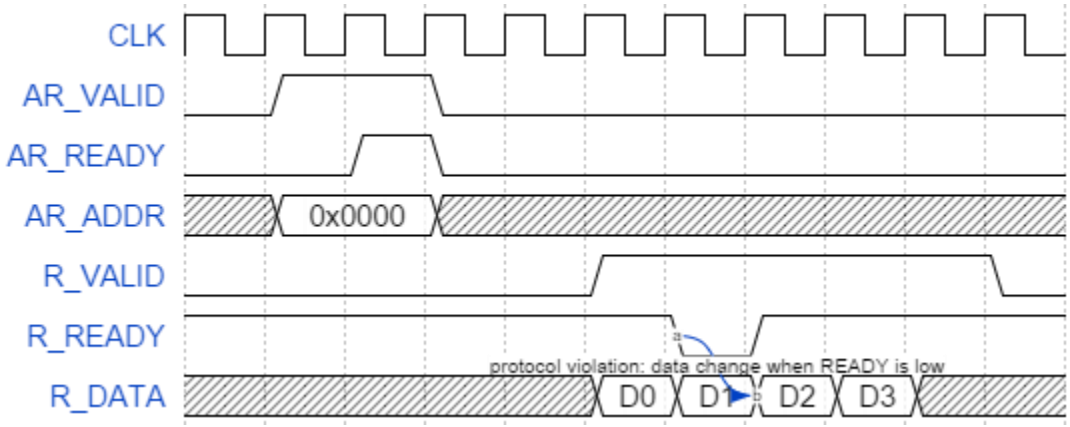
Below is code to represent a common protocol violation on an AMBA AXI read transaction.[12] Text such as this could be generated by protocol checkers as part of verification IP packages:

```
{ signal: [
  { name: "CLK",      wave: 'p.........', },
  { name: "AR_VALID", wave: '01.0.......', },
  { name: "AR_READY", wave: '0.10.......', },
  { name: "AR_ADDR",  wave: 'x=.x.......',
    data: "0x0000",                       },
  { name: "R_VALID",  wave: '0....1....0'  },
  { name: "R_READY",  wave: '1.....01...',
```

[12] See http://www.arm.com/products/system-ip/amba-specifications.php

```
                      node: '......a....', },
  { name: "R_DATA",   wave: 'x....====x.',
    data: "D0 D1 D2 D3",
                      node: '.......b...', },
],
  edge:['a~>b protocol violation: data change when READY is low'],
}
```

Below is the rendered waveform.

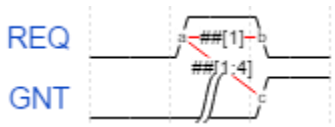


Generating assertions

It is possible for a user to express constraints using WaveJson in a way that they can both be rendered visually and translated, such as with a Perl script, into standard SystemVerilog assertions. There are an infinite number of ways to express constraints in WaveJson that make different trade-offs between readability and extensibility. Users should choose their format based on the range of types of assertions they intend to create, and what display format they find most readable.

The authors of this paper are not recommending any particular syntax for assertion expression. Creating a script to perform a translation to SystemVerilog assertions is left as an exercise to the reader. Below are some possible examples.

The following diagram is rendered from the following WaveDrom code (color added for emphasis).



```
{ signal: [
    {name: 'REQ', wave: '010', node: '.ab'},
    {name: 'GNT', wave: '0|1', node: '..c'}
  ],
  edge: ['a|>b ##[1]',
         'a|>c ##[1:4]']
}
```

A simple translation of the WaveDrom code yields a property assertion for each edge such as the following SystemVerilog assertions.

```
assert property (REQ |-> ##[1] ~REQ); // a|b
assert property (REQ |-> ##[1:4] GNT); // a|c
```

The translation uses the state, 1 or 0, of the wave corresponding to each node value to determine the state of the assertion implication (~ inversion). The translator uses SystemVerilog assertion syntax in the WaveDrom edge text. Other implementations could use any syntax appropriate for describing assertions. This opens a new level of assertion expression possibilities.

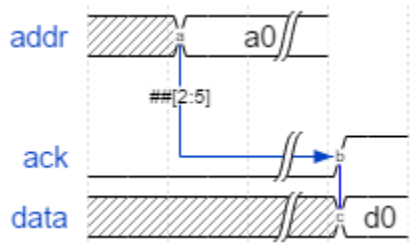
In another example, the following WaveDrom code

```
{signal: [
  {name: 'addr', wave: 'x=|',  node: '.a..', data: 'a0'}, {},
  {name: 'ack',  wave: '0.|1', node: '...b'},
  {name: 'data', wave: 'x.|=', node: '...c', data: 'd0'}
],
  edge: ['a|->b ##[2:5]', 'b-c']
}
```

yields

```
assert property (addr |-> ##[2:5] ack);
assert property (ack |-> data);
```

The rendered diagram looks like this.



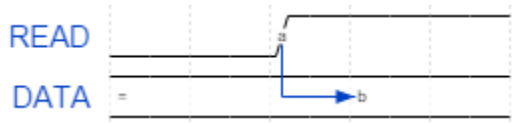
The dot character ('.') in WaveJson means repetition of the value of the previous cycle, which naturally maps to the SystemVerilog $stable construct.

```
{ signal: [
  { name: "READ", wave: "0.1..", node: '=.a..' },
  { name: "DATA", wave: "=...", node: '=..b.' },],
  edge: ['a|->b']
}
```

One particular translation of the code above finds that the `b` node corresponds to a `.` in the `DATA` wave. Since the `a` node represents a transition to `1` in the `READ` wave, the following SystemVerilog assertion results.

```
assert property (READ |=> $stable(DATA);
```

The assertion waveform is rendered below.



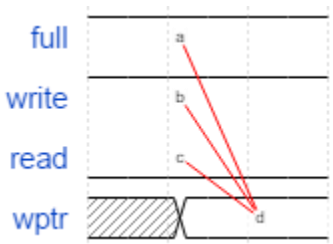
The translator converts the following WaveJson code

```
{signal: [
  {name: 'full',  wave: '1..',  node: '.a..'},
  {name: 'write', wave: '1..',  node: '.b..'},
  {name: 'read',  wave: '0..',  node: '.c..'},
  {name: 'wptr',  wave: 'x=.',  node: '..d.'},
],
  edge: ['a&->d', 'b&->d', 'c&->d']
}
```

to the following SystemVerilog assertion.

```
/* full && write && !read |=> $stable(wptr) */
```

Note that node `d` is the right hand side node for each edge, implying that the state of its signal (`wptr`) is the target of the assertion. The assertion waveform is rendered below.



Other users

The web version of WaveDrom is used by tens to hundreds of users at each of: Intel, AMD, Apple, Synopsys, Analog Devices, Nvidia, Cisco, National Semi, TI, ARM, Nvidia, HP, Marvell, Hynix, Infineon, Altera, Avago, Qualcomm, Volkswagen, Ford, and others.

Many others outside of the semiconductor industry use or have adapted WaveDrom.

Tutorial by Dave Vandenbout: http://www.xess.com/blog/timing-diagrams-made-easy/

Hackaday comparison to other waveform drawing applications:
http://hackaday.com/2015/05/25/need-timing-diagrams-try-wavedrom/

PSHDL Verilog coding game: http://waves.pshdl.org/

Yosys synthesis Verilog code verification game:
http://www.clifford.at/yosys/nogit/YosysJS/snapshot/demo03.html

Coroutine Co-simulation Test Bench (COCOTB) simulation trace to waveform usage:
http://potential.ventures/beautiful-documentation.html

BeagleBone BeagleLogic logic analyzer result display: http://beaglelogic.github.io/webapp/

A simple function to add wavedrom waveforms into an ipython notebook:
https://github.com/witchard/ipython-wavedrom

A Trac plugin: https://github.com/jun66j5/trac-wavedromplugin

Asciidoctor Diagram extension: https://github.com/asciidoctor/asciidoctor-diagram

What will you do?

Conclusion

Current practices for drawing waveforms do not support a fast, easy, error-free workflow. WaveDrom does so with an intuitive plain text language for describing waveforms. A free, open source editor renders clear diagrams with a consistent format. WaveDrom text is easy to embed in source code comments. Furthermore, WaveDrom text allows for diffing waveforms. Waveforms can also be parameterized. Waveforms becomes a source for describing and for visualizing constraints. Edges within waveforms naturally indicate relationships, and can be translated to other expression syntaxes such as SystemVerilog assertions. A lot of companies already use WaveDrom in their design work. Shouldn't you?