Hitchhiker's Guide to the WaveDrom

Fork me on GitHub

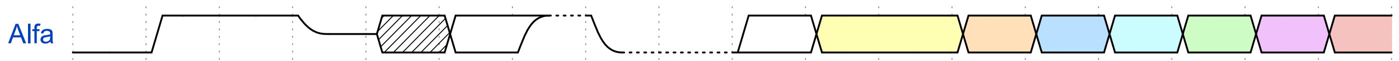
WaveDrom is a JavaScript application. WaveJSON is a format that describes Digital Timing Diagrams. WaveDrom renders the diagrams directly inside the browser. Element "signal" is an array of WaveLanes. Each WaveLane has two mandatory fields: "name" and "wave".

Step 1. The Signal

Lets start with a quick example. Following code will create 1-bit signal named "Alfa" that changes its state over time.

```
{ signal: [{ name: "Alfa", wave: "01.zx=ud.23.456789" }] }
```

Every character in the "wave" string represents a single time period. Symbol "." extends previous state for one more period. Here is how it looks:



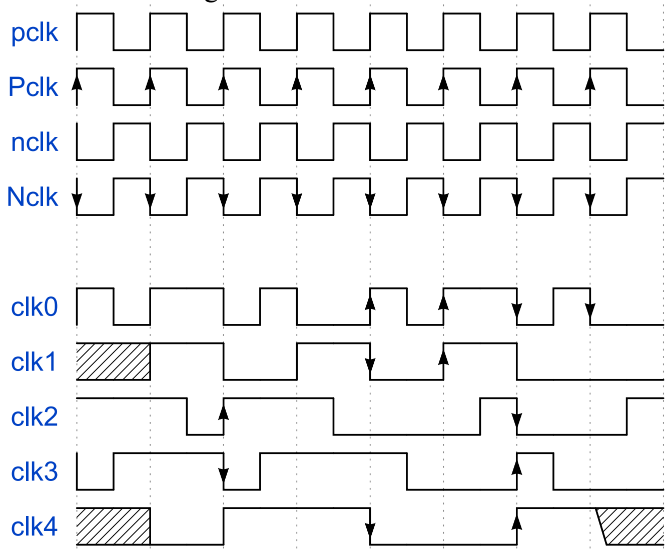
Edit Me >>

Step 2. Adding Clock

Digital clock is a special type of signal. It changes twice per time period and can have positive or negative polarity. It also can have an optional marker on the working edge. The clock's blocks can be mixed with other signal states to create the clock gating effects. Here is the code:

```
{ signal: [
  { name: "pclk", wave: 'p.......' },
  { name: "Pclk", wave: 'P.......' },
  { name: "nclk", wave: 'n.......' },
  { name: "Nclk", wave: 'N.......' },
  {},
  { name: 'clk0', wave: 'phnlPHNL' },
  { name: 'clk1', wave: 'xhlhLHl.' },
  { name: 'clk2', wave: 'hpHplnLn' },
  { name: 'clk3', wave: 'nhNhplPl' },
  { name: 'clk4', wave: 'xlh.L.Hx' },
]}
```

and the rendered diagram:

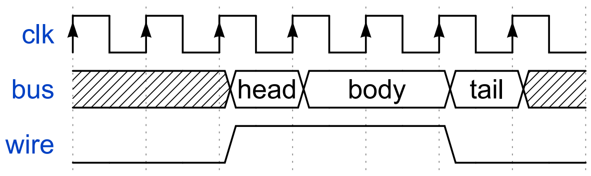


Edit Me >>

Step 3. Putting all together

Typical timing diagram would have the clock and signals (wires). Multi-bit signals will try to grab the labels from "data" array.

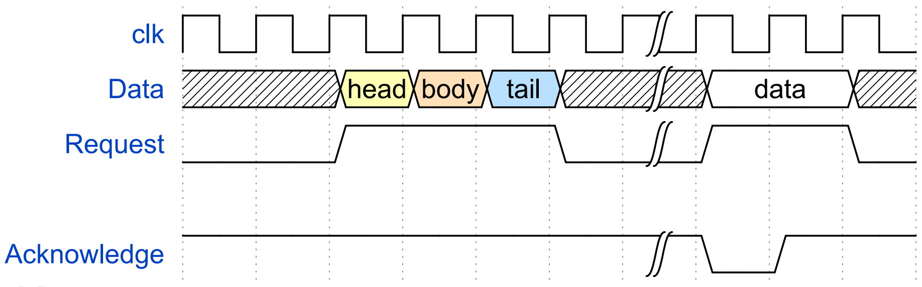
```
{ signal: [
  { name: "clk",  wave: "P......" },
  { name: "bus",  wave: "x.==.=x", data: ["head", "body", "tail", "data"] },
  { name: "wire", wave: "0.1..0." }
]}
```



Edit Me >>

Step 4. Spacers and Gaps

```
{ signal: [
  { name: "clk",         wave: "p.....|..." },
  { name: "Data",        wave: "x.345x|=.x", data: ["head", "body", "tail", "data"
  { name: "Request",     wave: "0.1..0|1.0" },
  {},
  { name: "Acknowledge", wave: "1.....|01." }
]}
```

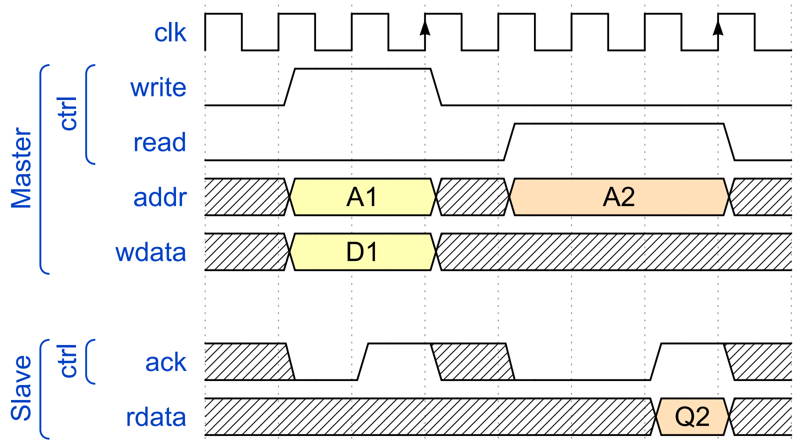


Edit Me >>

Step 5. The groups

WaveLanes can be united in named groups that are represented in form of arrays. ['group name', {...}, {...}, ...] The first entry of array is the group's name. The groups can be nested.

```
{ signal: [
  {    name: 'clk',   wave: 'p..Pp..P'},
  ['Master',
    ['ctrl',
      {name: 'write', wave: '01.0....'},
      {name: 'read',  wave: '0...1..0'}
    ],
    {  name: 'addr',  wave: 'x3.x4..x', data: 'A1 A2'},
    {  name: 'wdata', wave: 'x3.x....', data: 'D1'   },
  ],
  {},
  ['Slave',
    ['ctrl',
      {name: 'ack',   wave: 'x01x0.1x'},
    ],
    {  name: 'rdata', wave: 'x.....4x', data: 'Q2'},
  ]
]}
```



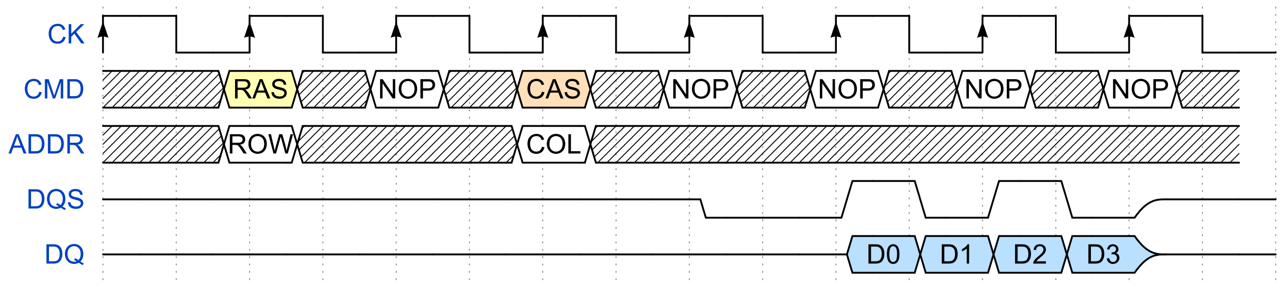
Edit Me >>

Step 6. Period and Phase

"period" and "phase" parameters can be used to adjust each WaveLane.

DDR Read transaction

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



Edit Me >>

Step 7.The config{} property

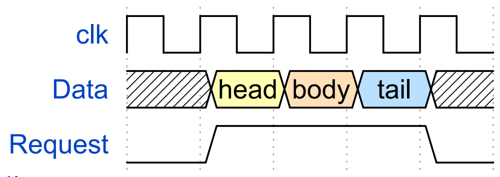
The config:{...} property controls different aspects of rendering.

hscale

config:{hscale:#} property controls the horizontal scale of the diagram. User can put any integer number greater than 0.

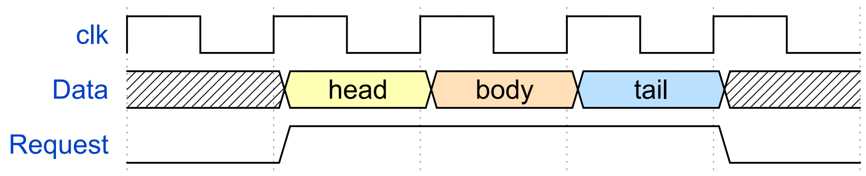
```
{ signal: [
  { name: "clk",     wave: "p...." },
  { name: "Data",    wave: "x345x",  data: ["head", "body", "tail"] },
  { name: "Request", wave: "01..0" }
  ],
  config: { hscale: 1 }
}
```

hscale = 1 (default)



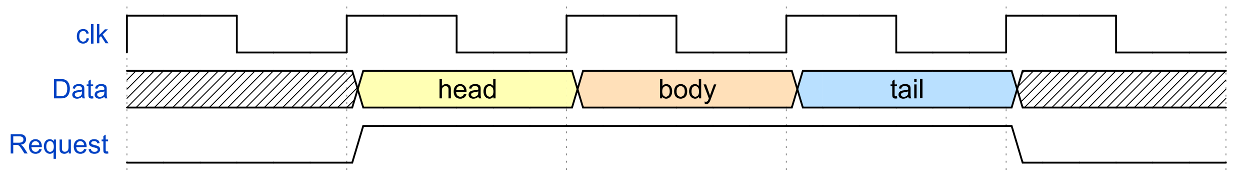
Edit Me >>

hscale = 2



Edit Me >>

hscale = 3



Edit Me >>

skin

config:{skin:'...'} property can be used to select the WaveDrom skin. The property works only inside the first timing diagram on the page. WaveDrom Editor includes two standard skins: 'default' and 'narrow'

head/foot

head:{...} and foot:{...} properties define the content of the area above and below the timing diagram.

tick

-adds timeline labels aligned to vertical markers.

tock

-adds timeline labels between the vertical markers.

text

-adds title / caption text.

every

-render ticks and tocks only once every N cycle

```
{signal: [
  {name:'clk',         wave: 'p....' },
  {name:'Data',        wave: 'x345x', data: 'a b c' },
  {name:'Request',     wave: '01..0' }
],
 head:{
   text:'WaveDrom example',
   tick:0,
   every:2
 },
 foot:{
   text:'Figure 100',
   tock:9
 },
}
```

WaveDrom example

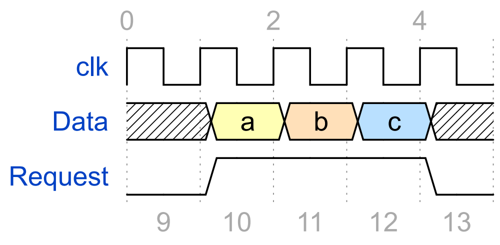


Figure 100

[Edit Me >>](#)

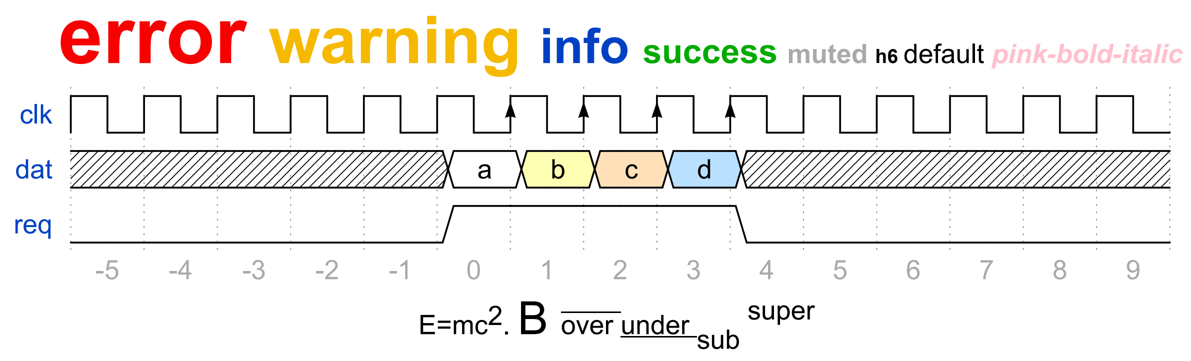
head/ foot text has all properties of SVG text. Standard SVG tspan attributes can be used to modify default properties of text. [JsonML](#) markup language used to represent SVG text content. Several predefined styles can be used and intermixed:

h1 h2 h3 h4 h5 h6 -- predefined font sizes.

muted warning error info success -- font color styles.

Other SVG tspan attributes can be used in freestyle as shown below.

```
{signal: [
  {name:'clk', wave: 'p.....PPPPp....' },
  {name:'dat', wave: 'x....2345x.....', data: 'a b c d' },
  {name:'req', wave: '0....1...0.....' }
],
head: {text:
  ['tspan',
    ['tspan', {class:'error h1'}, 'error '],
    ['tspan', {class:'warning h2'}, 'warning '],
    ['tspan', {class:'info h3'}, 'info '],
    ['tspan', {class:'success h4'}, 'success '],
    ['tspan', {class:'muted h5'}, 'muted '],
    ['tspan', {class:'h6'}, 'h6 '],
    'default ',
    ['tspan', {fill:'pink', 'font-weight':'bold', 'font-style':'italic'}, 'pink-b
  ]
},
foot: {text:
  ['tspan', 'E=mc',
    ['tspan', {dy:'-5'}, '2'],
    ['tspan', {dy: '5'}, '. '],
    ['tspan', {'font-size':'25'}, 'B '],
    ['tspan', {'text-decoration':'overline'},'over '],
    ['tspan', {'text-decoration':'underline'},'under '],
    ['tspan', {'baseline-shift':'sub'}, 'sub '],
    ['tspan', {'baseline-shift':'super'}, 'super ']
  ],tock:-5
}
}
```


error warning info success muted h6 default pink-bold-italic
clk
dat
req
a
b
c
d
-5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9
E=mc2. B over under sub super

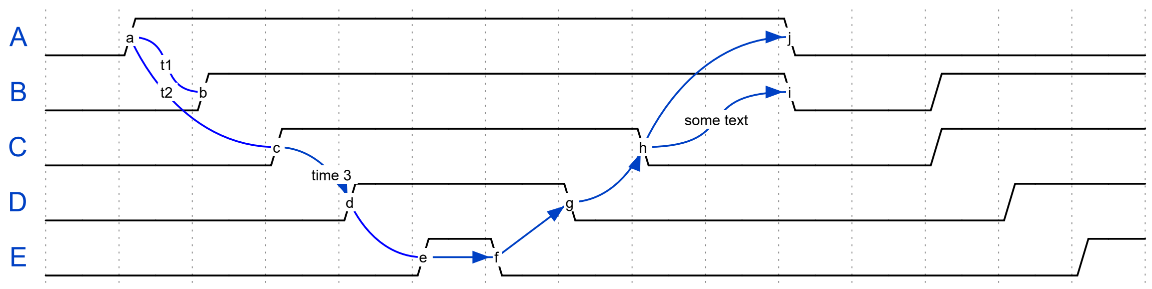
Edit Me >>

Step 8. Arrows

Splines

```
 ~    -~
<~>  <-~>
 ~>   -~>  ~->
```

```
{ signal: [
  { name: 'A', wave: '01........0....',  node: '.a........j' },
  { name: 'B', wave: '0.1.......0.1..',  node: '..b.......i' },
  { name: 'C', wave: '0..1....0...1..',  node: '...c....h..' },
  { name: 'D', wave: '0...1..0.....1.',  node: '....d..g...' },
  { name: 'E', wave: '0....10.......1',  node: '.....ef....' }
  ],
  edge: [
    'a~b t1', 'c-~a t2', 'c-~>d time 3', 'd~-e',
    'e~>f', 'f->g', 'g-~>h', 'h~>i some text', 'h~->j'
  ]
}
```



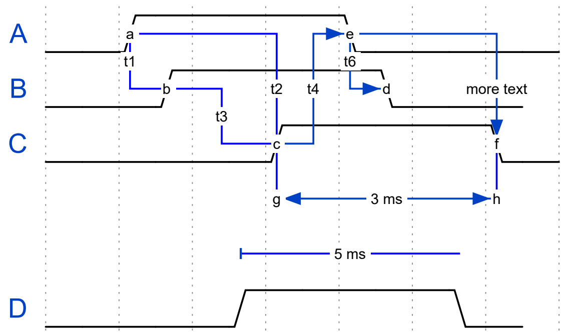
Edit Me >>

Sharp lines

```
 -   -|   -|-
<-> <-|> <-|->
```

```
-> -|> -|-> |->
+
```

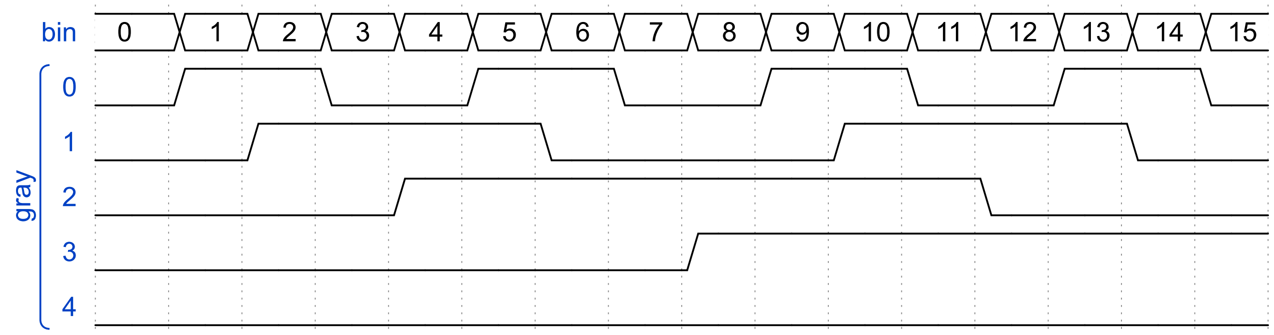
```
{ signal: [
  { name: 'A', wave: '01..0..',  node: '.a..e..' },
  { name: 'B', wave: '0.1..0.',  node: '..b..d.', phase:0.5 },
  { name: 'C', wave: '0..1..0',  node: '...c..f' },
  {                              node: '...g..h' },
  {                              node: '...I..J',  phase:0.5 },
  { name: 'D', wave: '0..1..0',  phase:0.5 }
  ],
  edge: [
    'b-|a t1', 'a-|c t2', 'b-|-c t3', 'c-|->e t4', 'e-|>f more text',
    'e|->d t6', 'c-g', 'f-h', 'g<->h 3 ms', 'I+J 5 ms'
  ]
}
```



Edit Me >>

Step 9. Some code

```
(function (bits, ticks) {
  var i, t, gray, state, data = [], arr = [];
  for (i = 0; i < bits; i++) {
    arr.push({name: i + '', wave: ''});
    state = 1;
    for (t = 0; t < ticks; t++) {
      data.push(t + '');
      gray = (((t >> 1) ^ t) >> i) & 1;
      arr[i].wave += (gray === state) ? '.' : gray + '';
      state = gray;
    }
  }
  arr.unshift('gray');
  return {signal: [
    {name: 'bin', wave: '='.repeat(ticks), data: data}, arr
  ]};
})(5, 16)
```


bin
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
gray
0
1
2
3
4