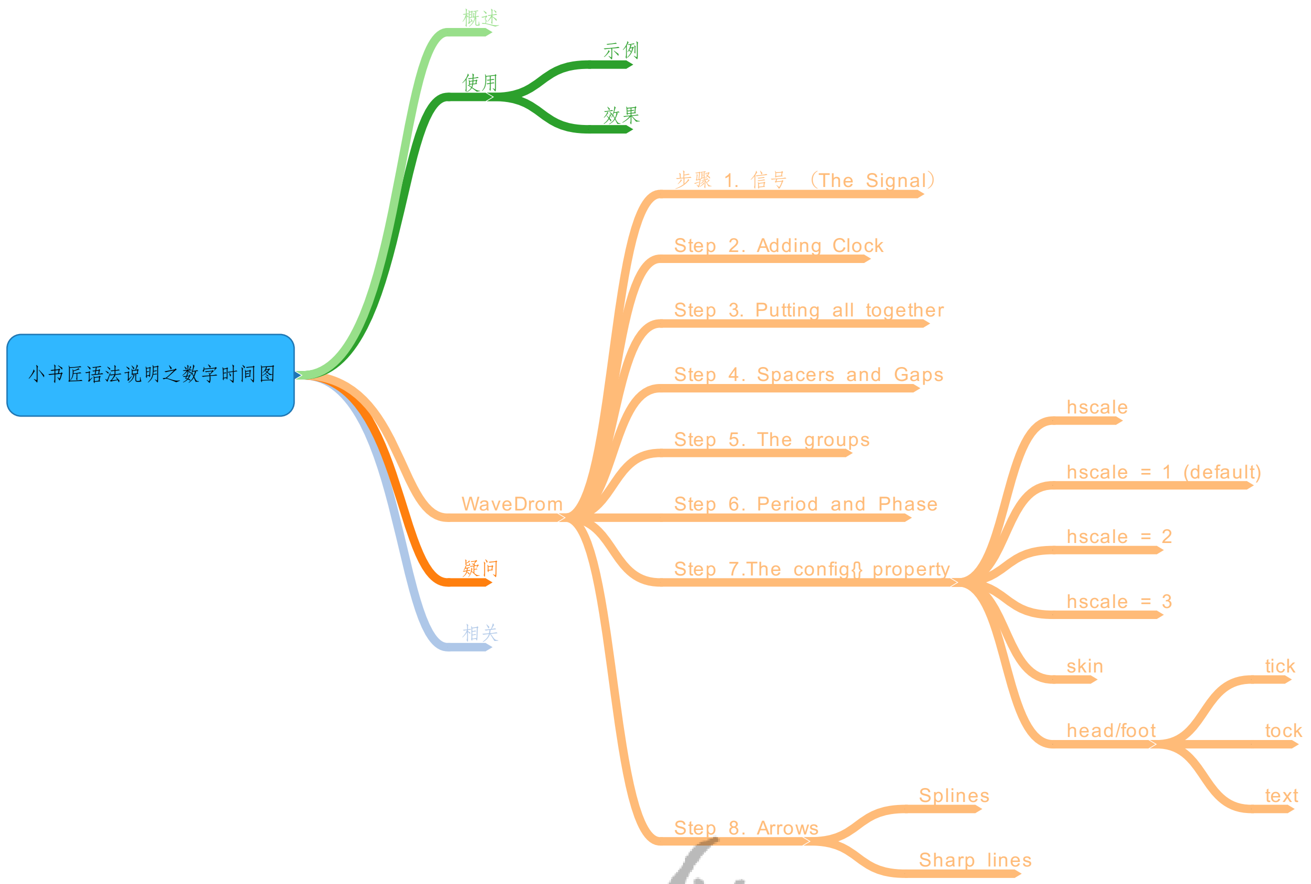
小书匠语法说明之数字时间图

小書匠

目录



概述

WaveDrom Digital timing diagram 语法用于显示数字时间图表。该语法不是标准的 commonmark， 目前只有小书匠提供了对该语法的支持。

> WaveDrom draws your Timing Diagram or Waveform from simple textual description.
> It comes with description language, rendering engine and the editor.

使用

元数据标识: **grammar_wavedrom**

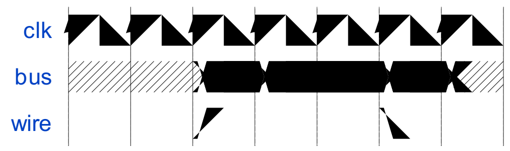
想要使用该语法，需要在**设置>扩展语法** 里把**WaveDrom Digital timing diagram 语法**选项打开。或者在每篇文章的元数据里通过 **grammar_wavedrom** 进行控制。系统默认关闭**WaveDrom Digital timing diagram 语法**语法功能

更详细的语法使用，可以参考这里 https://wavedrom.com/

示例

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

效果



WaveDrom

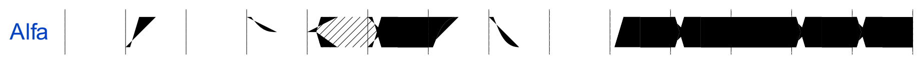
WaveDrom 是一个 javascript 应用。 WaveJSON 是一种描述数字时间图(Digital Timing Diagrams)的语言。 通过 WaveDrom 可以直接在浏览器内显示数字时间图表。元素 "信号(signal)" 是一个包含了 WaveLanes 的数组。每个 WaveLane 有两个必选字段: "name" 和 "wave"。

步骤 1. 信号 （The Signal）

Following code will create 1-bit signal named "Alfa" that changes its state over time.

下面的代码将创建一个名为 "Alfa" 的 1-bit 信号，并随着时间显示状态变化

~~~
``` wavedrom!
{ signal: [{ name: "Alfa", wave: "01.zx=ud.23.45" }] }
```
~~~



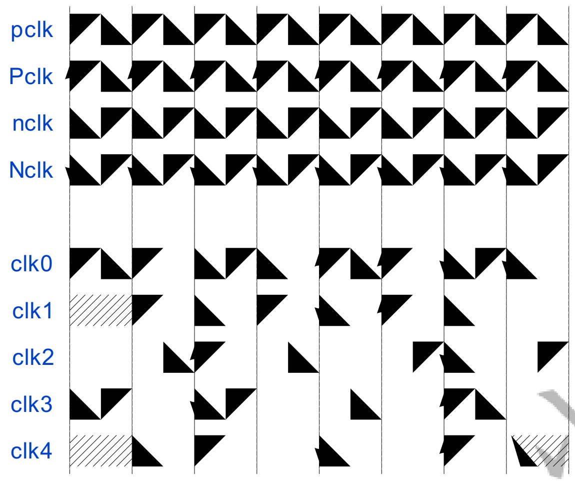
在 “wave” 字段内，每个字符表示一个单独时间点，符号"." 继承了前一个符号的状态。

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:

~~~
``` wavedrom!
{ 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' },
]}
```
~~~

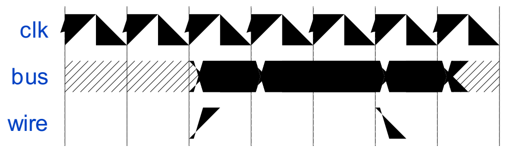
and the rendered diagram:



Step 3. Putting all together

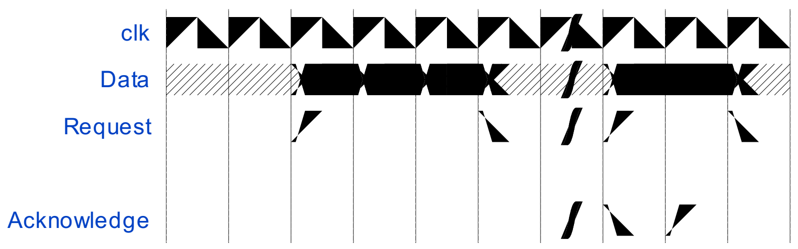
Typical timing diagram would have the clock and signals (wires). Multi-bit signals will try to grab the labels from `"data"` array.

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



Step 4. Spacers and Gaps

~~~
``` wavedrom!
{ 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." }
]}
```
~~~



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.

~~~
``` wavedrom!
{ 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'},
  ]
]}
```
~~~

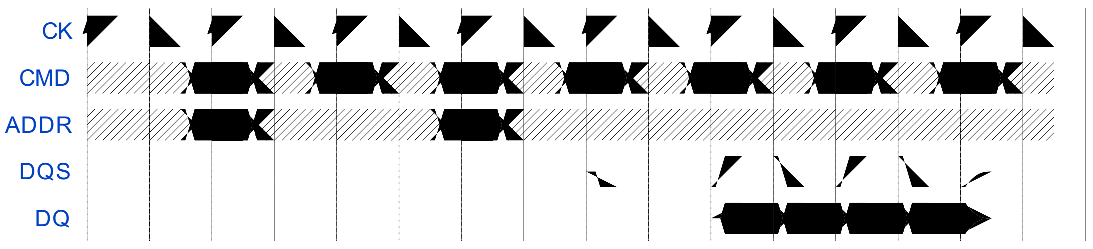
SyntaxError: Unexpected token] in JSON at position 262

Step 6. Period and Phase

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

DDR Read transaction

~~~
```wavedrom!
{ signal: [
  { name: "CK",   wave: "P.......",                                              pe
  { 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",                      p
0.5 },
  { name: "DQS",  wave: "z.......0.1010z." },
  { name: "DQ",   wave: "z.........5555z.", data: "D0 D1 D2 D3" }
]}
```
~~~



Step 7.The config{} property

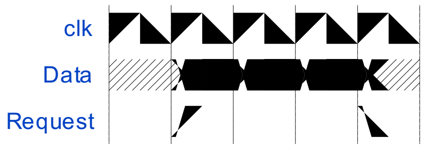
The `config:{...}` property controls different aspects of rendering.

hscale

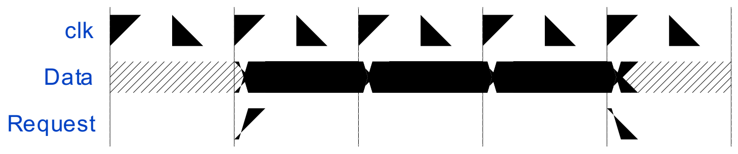
`config:{hscale:#}` property controls the horizontal scale of the diagram. User can put any integer number greater than 0.

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

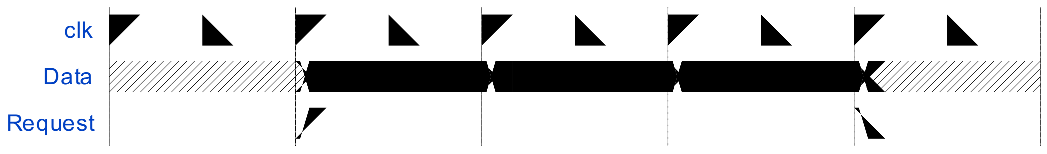
hscale = 1 (default)



hscale = 2



hscale = 3



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.

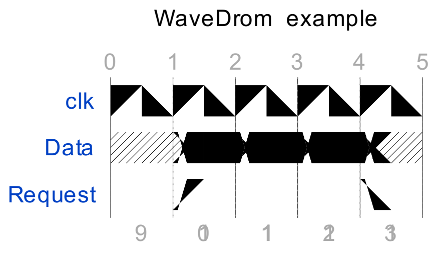
tock

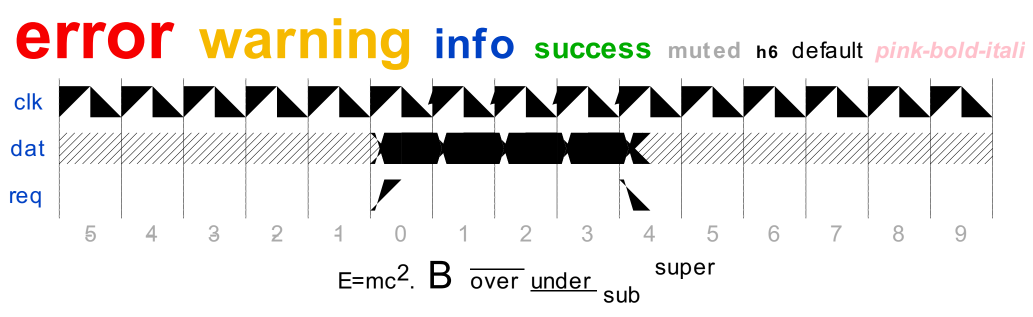
-adds timeline labels between the vertical markers.

text

-adds title / caption text.

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

```
    tock:9
  },
}
```
```


WaveDrom example
0
1
2
3
4
5
clk
Data
Request
9
0
1
2
3

Figure 100

`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.

````
``` wavedrom!
{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-
bold-italic']
  ]
},
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
}
}
```
````
````

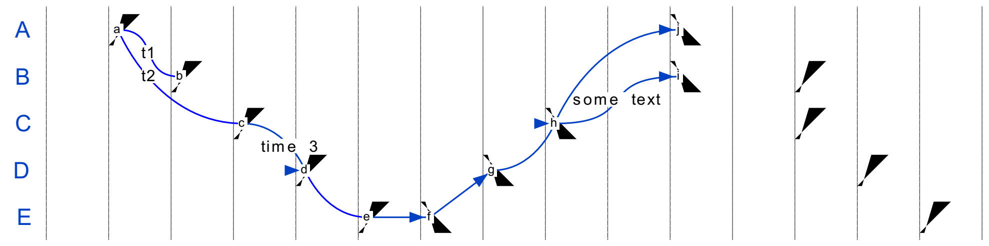


## Step 8. Arrows

### Splines

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

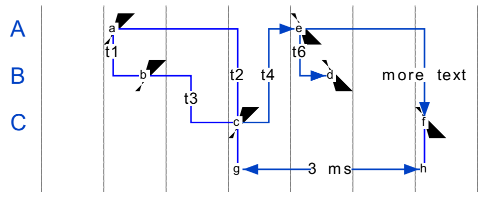
~~~
``` wavedrom!
{ signal: [
  { name: 'A', wave: '01........0....',  node: '.a........j' },
  { name: 'B', wave: '0.1.......0.1..',  node: '..b.......i' },
  { name: 'C', wave: '0..1....0...1..',  node: '...c....h..' },
  { name: 'D', wave: '0...1..0.....1.',  node: '....d..g...' },
  { name: 'E', 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'
  ]
}
```
~~~



### Sharp lines

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

~~~
``` wavedrom!
{ 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' }
  ],
  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'
  ]
}
```
```



# 疑问

# 相关
```