

The Package

A \LaTeX Package for Timing Diagrams

Version v0.7f – 2017/12/20

Martin Scharrer

martin@scharrer-online.de

WWW: <http://latex.scharrer-online.de/tikz-timing>

CTAN: <http://www.ctan.org/pkg/tikz-timing>

Contents

1	Introduction	5
1.1	Changelog	6
1.2	Dependencies	9
2	Usage	10
2.1	Timing Characters	10
2.2	Macro for use in Text Mode	13
2.3	Macro for use inside TikZ-Pictures	14
2.4	Table for Timing Diagrams	16
2.5	Macros for use inside the Character String	22
2.6	Meta-Characters	24
2.7	Floating timing diagrams with captions	27
3	TikZ Keys for Styles, Settings and Actions	28
4	Libraries for Further Characters	34
4.1	Arrows	35
4.2	Either High or Low	36
4.3	Overlays	37
4.4	Clock Arrows	38
4.5	Column Type	39
4.6	Nice Timing Tables	40
4.7	Counter Character	41
4.8	Advanced Nodes	45
4.9	Compatibility Macros for <code>ifsym</code> package	48
4.10	Intervals (experimental)	49


4.11 Beamer Overlay Support (experimental)	50
5 Examples	52


List of Examples


- 1 Initial Characters, Modifiers, TikZ Keys 53
- 2 tikztimingtable without extracode 54
- 3 tikztimingtable with extracode 54
- 4 timing inside general tikzpicture 55
- 5 Using In-Line Nodes to draw Relationships. 55
- 6 Using In-Line Nodes to draw Marker Lines. 56
- 7 Adjusting Diagram Parameters and using Advanced In-Line Nodes to draw Marker Lines. 57
- 8 SR flip-flop timing diagram 58
- 9 SPI Interface Timing 59

1 Introduction

This package uses the [pgf]tikz package to produce timing diagrams inside text or tikzpicture environments. Also a tabular-like environment is provided to produce a larger timing diagram with multiple labeled signals and the possibility to add own drawing material. Additional optional functionality is provided by libraries.

The signal levels of the timing diagram can be given by corresponding characters/letters like ‘H’ for *Logical High* or ‘L’ for *Logical Low*. So e.g. ‘{HLZXD}’ gives ‘

Recurring character combinations can be repeated using character groups (‘3{hlz}’ \rightarrow ‘

Additional features are the inclusion of in-line TikZ styles (‘H ;[orange] L’ \rightarrow ‘

5

1.1 Changelog

v0.3 from 2009/04/24

- First released version

v0.4 from 2009/05/03

- Added output routine which combines successive occurrences of the same character. This improves screen display quality and reduces rendering time and file size.
- Removed own macros for lowercase characters. They are now handled by the uppercase macros which receive half of the width. Exceptions are possible like for the ‘m’ character.
- Added parser for rows in `tikztimingtable`. This makes the syntax much more stable. Also replaced row counter with TikZ coordinates which is more user-friendly.
- User macros to draw grids and lines inside table.
- In-line Nodes, e.g. to mark positions inside the diagram.

v0.4a from 2009/05/05

- Added `\tablerules` macro. Changed default style of inline nodes to `coordinate`.

v0.5 from 2009/05/15

- Added PGF shape for timing diagrams. Added meta-characters. Changed ‘M’ character to use PGF decorations. Added special ‘B’ character to reduce width of next character. Changed `\timing` syntax to include an ‘at’ before the coordinate. Bug fix for use with the ‘calc’ package.

v0.6 from 2009/07/27

- Added “forward” modifier ‘F’ as reverse version of the “backward” modifier ‘B’.
- Added support for lower-case modifiers “b”, ‘f’ and n’.
- Added libraries for characters ‘A’/‘W’ for arrows and ‘E’ for uncertain low-to-high and high-to-low transitions.

v0.6a from 2009/07/28

- Added library for overlay modifier ‘O’.

v0.7 from 2009/12/05

- New libraries:
 - `clockarrows` Library for clock arrows.
 - `columntype` Library providing a timing column type for `tabular`.
 - `nicetabs` Library to format `\tikztimingtable` like a `booktab` `tabular`.
 - `counters` Library to defined counter characters which display an incrementing counter value every time there are used.
 - `advnodes` Library for advanced nodes with multiple anchor points.
 - `ifsym` Library providing the same timing symbols and characters as the `ifsym` package when loaded with the `electronic` option.
- Additional experimental libraries:
 - `interval` Library to change color of ‘ZL’, ‘ZH’ etc. transitions to indicate borders of an interval.
 - `beamer` Library providing some marginal beamer overlay support.
- `overlays` library:
 - Overlays can now be cascaded, i.e. an overlay can be inside another one.
 - The second braces around the second part are now optional.
 - Fixed issues with ‘T’ and ‘C’ characters inside overlays.
- Meta-characters can now have arguments.
- Added more variety for in-line options: ‘[[]]’, ‘[+ +]’ and ‘[|]]’.
- Handling of in-line options and nodes got modified. Options are now placed directly where placed and are valid until the next ‘;’. Please note that `[/utils/exec={..}]` now needs to be written as `[|/utils/exec={..}|]`. Otherwise it is re-executed every time the drawing path is renewed.

- Added star version of `\tablegrid`.
- Added background to ‘E’ character (`either` library).
- Some fixes for placing of ‘D{ }’ texts.
- Fixed wrong slopes (e.g. `lslope` instead of `zslope`) for some transitions.
- Major changes on internal character definition macros, parser and output routine.
- Fixed problems with expanding code content in user input.
- The `\texttiming` macro now uses a `\timing` macro internally.
- The `\timing` macro is now only defined inside `tikzpictures`. This includes `tikztimingtable`.
- Added TikZ style `timing/draw grid` for grids behind `\timing` macros.
- Replaced macros `\texttimingbefore`, `\texttimingafter` and `\texttiminggrid` with TikZ settings ‘`timing/before text`’, ‘`timing/after text`’ and ‘`timing/draw grid`’.
- Added separators ‘`timing/outer sep`’ around `\texttiming`.
- Graphical improvements for ‘double line’ characters like ‘D’, ‘U’ and ‘E’. The whole character including both edges is drawn in a single process.
- Character width can now be scaled using `wscale`.
- Character width can now be calculated by placing code inside ‘`$ $`’.
- Fixed issue with `\horlines` macro.
- The `tikztimingtable` environment and associated macros got enhanced:
 - The content is no longer read as macro argument and can now include paragraphs.
 - Multiple `extracode` sections can be now included between rows, not only a single section at the very end.

- A `extracode` environment has been added. Both macro and environment have now an optional argument for TikZ settings.
- Added `\tableheader` macro to label both columns. The `\tablerules` macro got adjusted to detect the header line and draw also a middle line.
- Added `background` environment to draw things in the background.
- Fixed broken optional argument of `\tablegrid`.
- Added macro `\marknodes` and associated `debug/nodes` style to mark in-line nodes for debug purposes/orientation during the diagram creation.

v0.7d from 2011/01/09

- Fix for end macro of `extracode` environment to support `etoolbox`'s environment hooks.

v0.7e from 2017/12/10

- Fixed `advnodes` library to support current PGF version.

v0.7e from 2017/12/10

- Documentation update: added usage as float with caption due to user request.

v0.7f from 2017/12/20

- Documentation update: added description of several existing styles.

1.2 Dependencies

...

2 Usage

2.1 Timing Characters

The logic levels are described by so called *timing characters*. Actually all of them are letters, but the general term *character* is used here. Table 2.1 shows all by default defined logic characters and Table 2.2 all possible two-character transitions. Additional functionality is provided by the *modifiers* shown in Table 2.3.

Table 2.1: Timing Characters

Character	Description	Diagram	Transition Example
H	High		
L	Low		
Z	High Impedance		
X	Undefined / Don't Care		
D	Data / Double		
U	Unknown Data		
T	Toggle		
C	Clock (no slope)		
M	Metastable Condition		
G	Glitch (zero width)		
S	Space (nothing)		




Table 2.2: Overview over all transitions.

from \ to	H	L	Z	X	M	D	U	T	C
H									
L									
Z									
X									
M									
D									
U									
T									
C									

Table 2.3: Modifiers for Timing Characters.

Modifier Syntax	Description
$D\{D\}$	Produces transition between two data values. <i>E.g.</i> : ‘D{D}’ →
$D\{\langle Text \rangle\}$	Adds $\langle text \rangle$ into a data signal using a node. <i>E.g.</i> : ‘D{A}D{B}’ →
$D\{[\langle TikZ Settings \rangle]\langle Text \rangle\}$	Adds $\langle text \rangle$ using the given node $\langle settings \rangle$. <i>E.g.</i> : ‘D{[blue]A}’ →
$\langle number \rangle \langle character \rangle$	Sets width of next signal to given number. Half of it if character is in lower case. <i>E.g.</i> : ‘2.6H5.21’ →
$\langle integer \rangle \{ \langle characters \rangle \}$	Repeats the given characters $\langle int \rangle$ times. <i>E.g.</i> : ‘5{h1}’ →
$\{ \langle characters \rangle \}$	Encloses characters in a local scope. Options inside are only local to the scope. This also applies to the effect of ‘;’ and similar modifiers. <i>E.g.</i> : ‘H {[blue] LH} L’ →
$\langle number \rangle B$	Subtracts the given number from the width of the next character. “Backwards” <i>E.g.</i> : ‘H.5BL’ →
$\langle number \rangle F$	Adds the given number to the width of the next character. “Forwards” <i>E.g.</i> : ‘H.5FL’ →
$N[\langle Settings \rangle](\langle Name \rangle)\{\langle Content \rangle\}$	Adds node at current position. All three arguments are optional. <i>E.g.</i> : ‘H N(a1) L’ →
$[\langle TikZ Keys \rangle]$	Executes given TikZ settings during the drawing process. This settings will be re-executed when the internal drawing path is renewed which can cause side-effects. <i>E.g.</i> : ‘H[blue]LH’ →
$[\langle TikZ Keys \rangle]$	Executes given TikZ settings during the drawing process like [] but does not re-executes them. <i>E.g.</i> : ‘D{.} [/utils/exec={\def \m {...}}] D{.} D{.}’ →
$![\langle TikZ Keys \rangle!]$	Executes given TikZ settings during the parsing process. Because this makes only sense for internal settings the default path is ‘/tikz/timing’, not ‘/tikz’ like in all other settings macros. <i>E.g.</i> : ‘H[!wscale=2.5!]LH’ →
$[[\langle TikZ Keys \rangle]]$	Executes given TikZ settings first during the parsing process and again during the drawing process. This is for settings which are needed for width calculations and again for the drawing code, e.g. the slope values. <i>E.g.</i> : ‘H[[timing/slope=.5]]L \$\slope \$H’ →
$!\{\langle code \rangle\}$	Places given code into the internal tikzpicture. See Example 1.
$@\{\langle code \rangle\}$	Executes the given code immediately during the parsing process. This can be used to change parsing parameters. To execute code during the drawing process use [/utils/exec= $\langle code \rangle$] instead. <i>E.g.</i> : ‘L @{\setwscaler {2}} H’ →

Table 2.3 – continued from previous page

Modifier Syntax	Description
$\langle \mathit{math\ expression} \rangle \$$	Takes a valid pgfmath expression (See pgf manual), evaluates it and places the result back in the input string so it can be used as width for the next character. The macros $\backslash slope = \backslash lslope$, $\backslash dslope$, $\backslash zslope$ and $\backslash wscale$ can be used to access the corresponding values. <i>E.g.</i> : 'D{ } \$ \dslope \$ D{ } D' → 
;	Renews the internal drawing path which ends the scope of all options given by []. <i>E.g.</i> : 'H; [blue]L;H' → 
,	Same as ';', but timing specific options (atm.: slopes and line width) are restored for the new path. <i>E.g.</i> : '[line width=1pt]L,H;L' → 

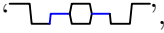
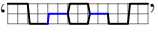
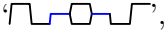
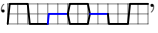
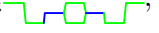
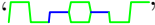
2.2 Macro for use in Text Mode

```
\texttiming[<initial character/TikZ Settings>]{<characters>}
```

This macro places a single timing diagram line into the current text. The signals have the same height as a uppercase letter (like ‘X’) of the current font, i.e. they scale with the font size. The macro argument must contain only valid logic characters and modifiers which define the logical levels of the diagram line.

An initial character can be given as an optional argument. No logic level will be drawn for this character. Instead it will be used to define the initial position of the signal so that the diagram line will start with a transition from the initial to the first character. However, if the optional argument holds more than a single character it is taken as TikZ settings for the diagram. The initial character can then be given using the key ‘`timing/initchar=<char>`’.

Examples:

```
\texttiming{HLZDZLH}    gives , with grid: .
\texttiming[L]{HLZDZLH} gives , with grid: .
\texttiming[green]{HLZDZLH} gives 
\texttiming[green, timing/initchar=L]{HLZDZLH} gives 
```

```
\texttimingbefore    Deprecated!                (defaults to: <empty>)
\texttimingafter     Deprecated!                (defaults to: <empty>)
```

These two macros are executed before and after every timing diagram line created by `\texttiming` macro inside the same `tikzpicture` environment and can be used to add drawing macros. The argument of the `\texttiming` macro is already processed before any of these macros are expanded, therefore these macros can access the width of the diagram.

These macros should not be used directly in newer code but instead the new TikZ styles ‘`timing/before text`’ and ‘`timing/after text`’. For backward compatibility these styles default to the two macros.

(Deprecated) Example: `\let\texttimingbefore\texttiminggrid` adds a grid into the background of the `\texttiming` diagram.

```
\texttiminggrid     Deprecated!
```

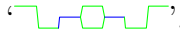
This macro should only be used inside `\texttimingbefore` or `\texttimingafter` and draws a grid of the full size of the `\texttiming` diagram. For newer code the TikZ styles ‘`timing/draw grid`’ and ‘`timing/no grid`’ should be used instead, e.g. `\tikzset{timing/intext/.append style={timing/draw grid}}` or simply enable the grid globally for all in-text and other timing diagrams with `\tikzset{timing/draw grid}`.

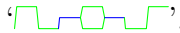
2.3 Macro for use inside TikZ-Pictures

```
\timing[TikZ Keys] at (TikZ Coordinate) {[initial character]}characters};
```

This macro does the same as `\texttiming` but is designed to be used inside a `tikzpicture` environment and only there. Like normal TikZ macros (`\path`, `\drawn`, `\node`) it allows an optional argument with TikZ settings and an optional TikZ-coordinate. However, a own argument parser, not the one used by TikZ, is used to detect and read these optional arguments. Therefore the order of the arguments is mandatory and must not be reversed. This small limitation might be overcome in future versions of this package.

Please note that the optional initial character may be given *inside* and at the very start of the mandatory argument, not before it. This is necessary because of several technical reasons.

Example: `\tikz \timing [green] at (1,2) {HLZDZLH};` gives .

Example: `\tikz \timing [green] at (1,2) {[L]HLZDZLH};` gives .

Timing Shape Anchors

Every timing diagram line produced by `\timing`, which includes the rows in `tikztimingtable`, is also a PGF shape (node) with several anchors. These are shown in Figure 2.1. The shape is very similar to the standard `rectangle` shape but does not provide a `text` anchor. In addition to the standard points of the compass anchors of TikZ the three logic levels `low`, `mid` and `high` can be used in combination with `start`, `mid` and `end`. An extra `origin` anchor is located at the lower left, also called `south west` corner where the diagram originates. The two anchors called `start` and `end` are provided to mark the start and end of the timing signal. There are either located at the low, middle or high logic level dependent on the used first (or initial) and last timing character.

In order to use the timing node it has to be named which can be done using the `'name=<name>'` option inside the optional argument. The rows of a `tikztimingtable` are automatically named as `'row<row number>'` where the first row has the number 1.

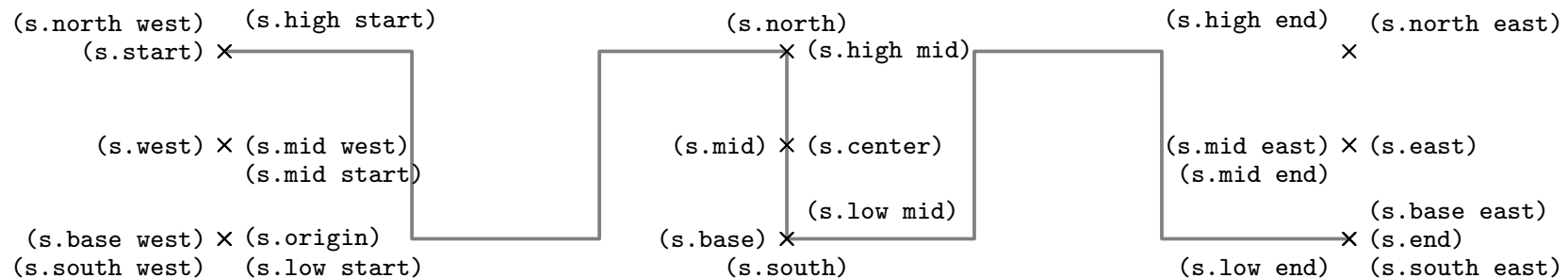


Figure 2.1: Timing Shape Anchors. The `start` and `end` anchors mark the start and end of the timing signal.

2.4 Table for Timing Diagrams

```
\begin{tikztimingtable}[\langle TikZ settings for whole table \rangle]
  {\langle Signal Name \rangle} & [\langle Init. Char./TikZ Keys for Row \rangle] \langle Characters \rangle \\
  \dots
\extracode % Optional
  \langle additional code \rangle
\end{tikztimingtable}
```

This environment can be used to typeset multi-line timing diagrams. The syntax is like the one for a `tabular` environment with two columns. The first column is for the signal name and the second one are the logic characters which would be placed inside the argument of a `\texttiming` or `\timing` macro. If the second column starts with an optional argument it is either taken as initial character if it holds only a single character or as row wide settings otherwise. The whole table will be drawn inside a `tikzpicture` environment using multiple `\timing` and `\node` macros for the timing signals and their names, respectively. Additional `tikz` drawing code can be insert at the end of the table using `\extracode`.

```
\extracode [TikZ Keys]
```

This macro is only defined inside a `tikztimingtable` environment. In earlier versions of this package it could only be used after the last table line (i.e. after a `\\`). If used there all code between it and the `\end{tikztimingtable}` will be placed inside the same `tikzpicture`. This allows to add some drawing lines or a grid to the picture. The macro does *not* start a TikZ scope or a TeX group by itself. The optional `\langle settings \rangle` therefore affect all following code until the end of the picture.

It is also possible to draw something behind the timing diagram by using using the `background` environment or the PGF background layer:

```
\begin{pgfonlayer}{background}... \end{pgfonlayer}
```

```
\endextracode
```

From version 0.7 on it is possible to add further timing rows after an `extracode` section by using `\endextracode`. Everything after this macro is taken as part of a new row. It is allowed to use this macro direct before `\endtikztimingtable`. This makes it possible to use `\extracode` anywhere inside the table, including at the very start before any rows are processed. Early insertion of extra code is necessary for e.g. limiting the bounding box or setting up a clipping path.

New in v0.7


```
\begin{extracode}[\langle TikZ settings \rangle]
  \langle extra drawing code \rangle
\end{extracode}
```

Instead of using `\extracode ... \endextracode`, which is actual plainTeX syntax, this L^AT_EX style environment can be used. Like any environment it creates a TeX group around its content, but no TikZ scope is created to allow drawing code (e.g. clipping paths) to affect the rest of the table. The optional `\langle settings \rangle`, however, only affect the environment content.

New in v0.7

Please note that while `\endextracode` is optional if `\extracode` is used at the end of the table, a `\begin{extracode}` must always be closed by `\end{extracode}`.

Macros for `\extracode` Section

The following macros are only defined inside a `tikztimingtable` after the macro `\extracode`. They are useful for drawing additional material.

```
\tablegrid*[\langle TikZ Keys \rangle]
```

After `\extracode` this macro draws a grid in the background of the table. A separate grid is drawn for each row. The normal version draws all grids with the width of the widest row while the star version draws them with the width of the corresponding row. Because this macro draws material into the `background` layer it must not be placed inside a `pgfonlayer` environment itself.

```
\fulltablegrid[\langle TikZ Keys \rangle]
```

After `\extracode` this macro draws a big grid over all rows in the background of the table.

```
\nrows
```

Returns the number of rows in the current table. Useful for use in `\horlines`.

`\rowdist`
`\coldist`

This macros return the row and column distance. There are useful for drawing additional material relative to the rows and columns. This values can be set (e.g. in the optional argument of the table) using the `timing/rowdist` and `timing/coldist` settings which are explained in Section 3.

`\twidth`

Returns the width (as multiple of the ‘period width’) of the longest timing diagram line in the table. Example: If the longest line would be ‘H 2.3L z’ than `\twidth` would be $1 + 2.3 + 0.5 = 3.8$.

`\horlines` [*TikZ Keys*] {*list*}

Draws horizontal lines, optionally with the given *Settings*, at the base line of the rows given by *list*. The PGF macro `\foreach`¹ is internally used so the list can include not only row numbers as integer but also fractional numbers and the ‘...’ operator to auto-increment the numbers. Please note that all numbers in the list are multiplied by `\rowdist`. If the list is empty the default ‘1,2,...,\nrows’ is used which draws lines for all rows.

`\vertlines` [*TikZ Keys*] {*list*}

Like `\horlines` but draws vertical lines and the listed numbers a relative to the basic width. If the list is empty the default ‘0,1,...,\twidth’ is used which draws lines after every period width.

`\tableheader` [*TikZ Keys*] {*Description Title*} {*Signal Title*}

This macro adds a table head row on top of the table. The two mandatory arguments provide the header text for the description and signal columns.

¹See the `pgf` manual for more details.

```
\tablerules[⟨TikZ Keys⟩]
```

This macro adds top and bottom rules to the table in the same (or at least very similar) way as the `booktabs` package is doing it for normal `tabulars`. The current bounding box is used to calculate the needed rule length, which makes this macro position dependent if further code changes the bounding box. If the `\tableheader` macro was used beforehand it also draws a thinner horizontal line (like `booktabs \midrule`) between the table head and body.

```
\begin{background}[⟨TikZ Keys⟩  
  ⟨drawing commands⟩  
\end{background}
```

This environment can be used to draw material on the `background` layer and is an abbreviation for:

```
\begin{pgfonlayer}{background}  
  \begin{scope}[⟨TikZ Keys⟩  
    ⟨drawing commands⟩  
  \end{scope}  
\end{pgfonlayer}
```

Scaling the Table

The standard ‘`scale`’ setting of TikZ will not scale all parts of the table correctly, e.g. the line width and nodes will keep their original size. However there are scaled relative to the font size (which needs to be set using `timing/font`). If the timing diagrams should be scaled the keys `timing/unit`, `timing/xunit` and/or `timing/yunit` can be used.

Alternatively the table can be scaled using the `\scalebox{⟨factor⟩}{⟨content⟩}` macro from the `graphicx` package or be placed inside a scaled `\node` inside another `tikzpicture` environment.

Positions & Nodes inside the Table

Coordinates

The first row starts at $y = 0$ and the next rows are each $-1*\rowdist$ lower than the previous one. The vertical unit is 1 signal height and the default row distance is ‘2’ ($=2\times\text{signal height}$). This means that a normal table with three rows goes from

$y = +1$ (base line at $0 + 1 \text{ signal height}$) to $y = -4$ (first row: $+0$, second row: -2 , third row: -4). This are relative to the middle of the drawn lines, i.e. the bounding box is $2 \times \frac{\text{line width}}{2} = 1 \times \text{line width}$ higher.

The timing column starts at $x = 0$ and goes into the positive range while scaled using the period width. Example: `HHHh` has a width of 3.5. The label column starts at $x = -\text{coldist}$ and the text is right align with the right border at this position. See Figure 2.2 for an illustration.

Nodes

Each timing line is a timing node (see section 2.3) labeled (not fully correctly) as `row<number>`, where the first row has the number 1 and the last one the number provided in `\nrows`, but can also accessed using the alias `last row`. The corresponding labels are normal rectangle nodes named `label0`, `label1`, ..., `label\nrows`/`last label`.

Both groups of `rows` and `labels` are enclosed in a rectangle node called `all rows` and `all labels`, respectively. These nodes can be used to draw material relative to the rows, e.g. the macros `\tableheader` and `\tablerules` are making use of them. The headers added by `\tableheader` are rectangle nodes names `label header` and `row header` and are placed between the x-coordinates of the inner and outer border of `all labels` and `all rows` respectively. By default the TikZ settings `pos=0` and `anchor=base east`/`anchor=base west`, respectively, are applied to place them in the inner border, but this can be changed using the styles `timing/table/header` and/or `timing/table/label header`/`timing/table/row header`. All nodes are shown in Figure 2.2.

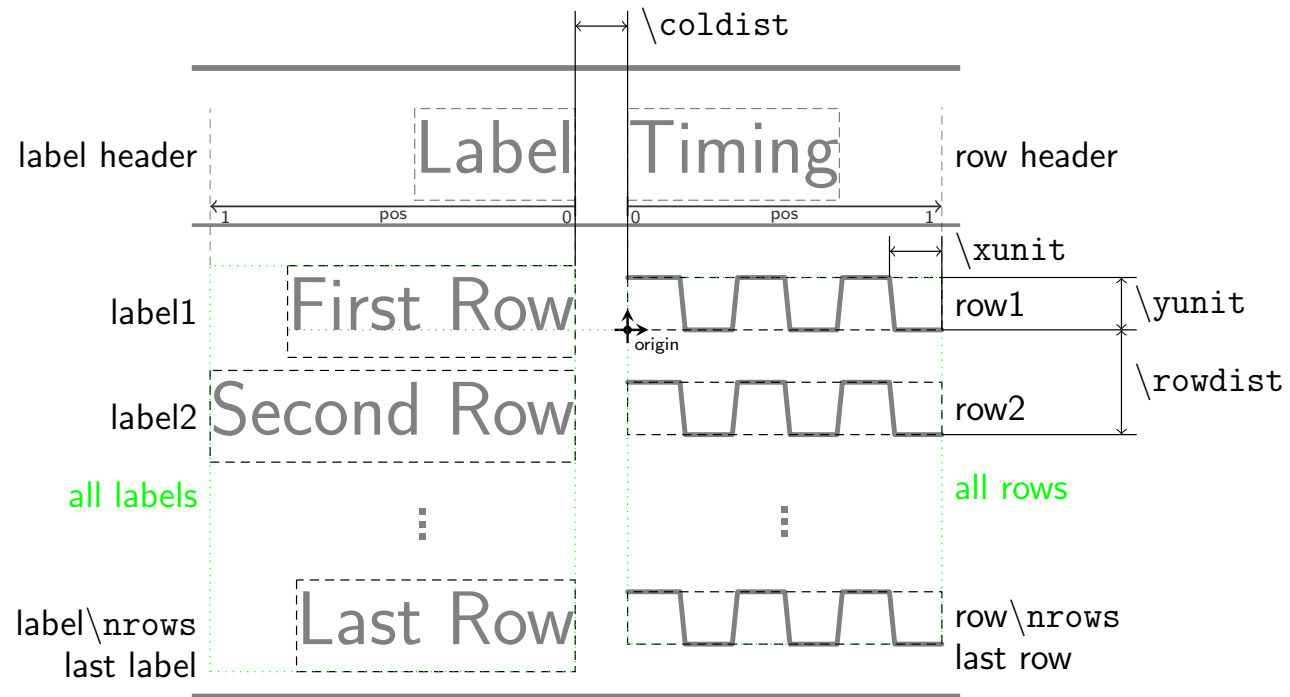


Figure 2.2: Distances and Nodes inside a `tikztimingtable`

2.5 Macros for use inside the Character String

The modifiers ‘@’ and ‘\$’ allow the user to include macros. These macros are evaluated when the `tikz-timing` parser encounters them in the input character string, i.e. before any diagram element is drawn or any single bracket ‘[]’ options are processed. Therefore their values should be set either outside the `tikz-timing` diagram or with the ‘[! .. !]’ or ‘[[..]]’ option blocks.

The following macros are provided for the user.

```
\tikztimingsetwscale{<math expression>}  
\setwscale{<math expression>}
```

This macro, which can be called `\setwscale` for short inside modifier code, sets the `wscale` value. This value is used during the parsing process to scale the width of the characters, e.g. `wscale=3.2` makes 1H as long as 3.2H normally would be. Slopes are not affected, but the ‘width’ values of meta-characters are. It can also be set with the `timing/wscale` TikZ setting. The current value can be accessed using `\wscale`.

New in v0.7

```
\wscale
```

Returns the current width scaling ‘`wscale`’ value.

```
\xunit  
\yunit
```

This dimension registers can be used to access the x- and y-unitlength of the timing diagram. Assignments to these registers do not change the scaling!

```
\slope  
\lslope (alias)
```

Returns the current logic slope, i.e. the slope between L and H levels. Set by the `timing/lslope` or indirectly by the `timing/slope` TikZ setting. See Table ?? for more information.

`\zslope`

Returns the current Z slope. Set by the `timing/zslope` or indirectly by the `timing/slope` TikZ setting.

`\dslope`

Returns the current Z slope. Set by the `timing/dslope` or indirectly by the `timing/slope` TikZ setting.

Examples:

Changing the slope and using its value to calculate the width of a character:

```
\texttiming{ HLHLHL [[timing/slope=.5]] H $\slopesL }
```

gives: 

Changing the width scaling for a certain group of characters:

```
\texttiming{ HL [!wscale=\wscale/3!] 3D{a} Z D Z [!wscale=3*\wscale!] HL }
```

gives: 

2.6 Meta-Characters

It is possible to define recurring groups of characters and modifiers as so called *meta-characters*. These characters are then expanded to the group whenever they appear inside the character list. Please note that like for groups a numeric factor before such a meta-character is taken as a repetition factor not as a width. The meta-character is case sensitive and the other case is not affected by the definition, i.e. the lower- and uppercase versions of one character can have complete different meanings. It is possible to redefine normal characters (only one or both cases) as meta-characters, which suppresses its normal meaning. Using the meta-character in its own definition group causes an infinite loop which will lead to a T_EX error.

```
\tikztimingmetachar{<Meta-Character>[<Number of arguments>]{<Character Group>}
```

This macro defines the given *<meta-character>* to be identical to the given *<character group>*. Alternatively this can also be done using the TikZ style `timing/metachar={<Meta-Character>[<Number of arguments>]{<Character Group>}`.

An empty group deletes the meta-character, which might be necessary in cases when normal characters are temporarily redefined as meta-characters. However, if the group only contains spaces the meta-character is practically ignored.

Because the meta-character is simply expanded to its character list, the first character of this list might be combined with identical characters placed before the meta-character. For example, after a meta-character ‘Y’ got defined as ‘2D{A} 2D{B}’ the characters ‘DY’ will first be expanded to ‘D2D{A} 2D{B}’ and then combined to ‘3D{A} 2D{B}’. This might not be the wanted behaviour and can be avoided by terminating the leading ‘D’ with its own braces: ‘D{Y}’.

Meta-Characters with Arguments

The replacement text of meta-character can now include macro arguments. This allows the creation of more complex and flexible meta-characters. The optional argument *<Number of arguments>* selects the number of macro arguments in the same way it does for `\newcommand`. However, the first argument #1 is always set to the given ‘width’ of the meta-character, i.e. the number value preceding it. All further arguments are read as normal for macros from the text after the meta-character. It is recommended to enclose them in braces.

New in v0.7

The default behaviour of meta-character without arguments is, as mentioned above, to repeat the replacement group by the preceding number (‘width’). This is now archived by defining them internally as `#1{<Character Group>}`, which creates a repetition group. Users which want to include an own argument but still want to repeat the group need to define a meta-character with at least two arguments and define it as `#1{ .. #2 .. }`. If the repetition is not wanted the #1 argument can be used as a real width for one or more group internal characters: `{Y}[1]{Z #1D Z}`, so ‘4Y’ will give ‘Z 4D Z’.

instead of ‘4{Z D Z}’.

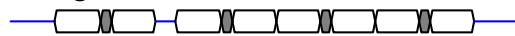
Also the modifier ‘@’ (see Table 2.3) together with the `\setwscale` macro can be used to scale the whole group dependent on the first argument:

`{Y}[1]{ @{\setwscale{#1*\wscale}} Z 2D Z @{\setwscale{\wscale/#1}} }`, so ‘4Y’ is equivalent to ‘4Z 8D 4Z’.

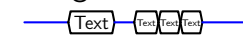
The new ‘\$’ modifier can be used to calculate the width of the group characters: `{Y}[1]{ $#1/3$D{A} $#1/3$D{B} $#1/3$D{C} }`, so ‘4Y’ results in ‘1.333D{A} 1.333D{B} 1.333D{C}’.

Examples:

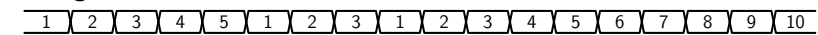
```
\tikztimingmetachar{Y}{2D 0.5U 2D{}} \texttiming{ZZ Y Z 3Y ZZ}
```

gives: 

```
\tikztimingmetachar{Y}{2D{Text}} \tikztimingmetachar{y}{1D{\tiny Text}} \texttiming{ZZ Y Z 3y ZZ}
```

gives: 

```
\newcounter{mycount}
\tikztimingmetachar{Q}{2D{\stepcounter{mycount}\arabic{mycount}}}
\tikztimingmetachar{R}{[| /utils/exec=\setcounter{mycount}{0} |]}
\texttiming{ 5Q R 3Q R 10Q }
```

gives: 

Redefining the glitch ‘G’ character:

```
\tikztimingmetachar{G}{.1T.1T .2B} \tikztimingmetachar{g}{.1T.1T}
\texttiming{ 10{H G L G} H } % With correction of width ‘.2B’
\texttiming{ 10{H g L g} H } % Without correction
\texttiming{ 10{H L } H } % For comparison
```

gives:

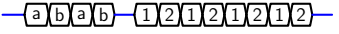


```
\tikztimingmetachar{J}[2]{ 1.8D{#2} .2D{ } }
\texttiming{ D{ } J{A} J{B} J{C} D }
```

gives: 

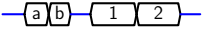
```
\tikztimingmetachar{Y}[3]{#1{ D{\strut #2} D{#3} }}
```

```
\texttiming{ Z 2Y{a}{b} Z 4Y{1}{2} Z }
```

gives: 

```
\tikztimingmetachar{Y}[3]{ .2D .2B #1d{\strut #2} .2D .2B #1d{\strut #3} }
```

```
\texttiming{ Z 2Y{a}{b} Z 4Y{1}{2} Z }
```

gives: 

Mata-chars to set the width scaling. Because the scaling also affects the meta-char width (#1) argument a compensation is needed to achieve absolute values (W) instead of relative ones (w).

```
\tikztimingmetachar{w}[1]{ [! wscale=#1 !] } % relative
```

```
\tikztimingmetachar{W}[1]{ [! wscale=#1/\wscale !] } % absolute
```

```
\texttiming{ HL .2w HLHLH 3w LH 1W LH }
```

gives:



2.7 Floating timing diagrams with captions

The timing diagrams can also be typeset as floats with (or without) a caption like a **figure**. This can be easily achieved by using the `caption` package as shown in the following example document.

```
\documentclass{article}
\usepackage{tikz-timing}
\usepackage{caption}

\DeclareCaptionType{timingdiag}[Timing diagram][List of Timing Diagrams]

\begin{document}
\listoftimingdiags

\begin{timingdiag}[!ht]
  \centering
  (timing code)
  \caption{Some timing diagram caption}
  \label{tim:foobar}
\end{timingdiag}

\end{document}
```

3 TikZ Keys for Styles, Settings and Actions

TikZ itself uses the `pgfkeys` package to define and apply drawing styles and settings. The same method is also used for `tikz-timing` which places all of the keys under the “subdirectory” ‘`timing`’ in the main “directory” ‘`tikz`’, which is the default when `\tikzset` is used. These keys are simply called *TikZ Keys* throughout this manual and can be used in all places where *⟨TikZ Keys⟩* is mentioned, while some only make sense at specific places. Three types of keys are used by this package: styles, settings and actions.

Styles simply define the style in which a certain element is drawn, e.g. in which color or line width. These styles are defined and can be redefined using `\tikzset{⟨style name⟩/.style=⟨value⟩}`. However, while some styles are initially empty, some hold internal settings and therefore user styles should only be added using ‘`.append style=⟨value⟩`’.

Settings are TikZ keys which await an argument and set an internal macro or length. They are like `\setlength` and `\renewcommand`. They should only be used as stated and not be redefined like styles as shown above.

Actions are TikZ keys which perform a drawing or other action on the current element, either directly or by en-/disabling an internal setting which then in turn triggers the drawing process. Therefore some actions can be globally and/or locally but others only make sense if used locally on a single `tikz-timing` macro or environment or even a scope inside a `tikztimingtable`. Actions can be very similar to settings but they always execute code instead of only setting/redefining it. The actions are defined and can be redefined using `\tikzset{⟨action name⟩/.code=⟨code⟩}`

General

TikZ Key	Type	Description
<code>timing</code>	Style	Base settings like signal height and period width.
<code>timing/font=<i></i></code>	Setting	Sets the normal <code>font</code> key and sets <code>x/y</code> keys to 1.6ex.
<code>timing/intext</code>	Style	Used for <code>\texttiming</code> . Depends on <code>timing</code> .
<code>timing/picture</code>	Style	Usable for own <code>tikzpictures</code> to set timing settings.
<code>timing/inline node</code>	Style	Used for nodes created by the <code>N</code> character. Defaults to <code>coordinate</code> .
<code>timing/every char</code>	Style	Used in the <code>\draw TikZ</code> command for every drawn timing character.
<code>timing/every bg</code>	Style	Used for every background path of characters like <code>D</code> .
<code>timing/text format=<i><code></i></code>	Setting	Sets formatting code for the text of characters like <code>D</code> . The code is placed directly for the text wrapped in braces, so that the code can be a macro awaiting the text as an argument. By default the code is empty.
<code>timing/initchar=<i><char></i></code>	Setting	Sets initial character. Only valid as first optional argument in table rows or in <code>\texttiming</code> .
<code>timing/metachar={<i><C></i>}[<i><#arg></i>]{<i><G></i>}</code>	Setting	Sets meta-character <code><C></code> to character group <code><G></code> .
<code>timing/before</code>	Action	Code executed direct before the timing TikZ code inside the internal <code>\tikzpicture</code> .
<code>timing/after</code>	Action	Code executed direct after the timing TikZ code inside the internal <code>\tikzpicture</code> .

Scaling

TikZ Key	Type	Description
<code>timing/unit=<i><length expression></i></code>	Setting	Sets both the x- and y-unitlength.
<code>timing/xunit=<i><length expression></i></code>	Setting	Sets the x-unitlength (<code>\xunit</code>) for the timing diagrams.
<code>timing/yunit=<i><length expression></i></code>	Setting	Sets the y-unitlength (<code>\yunit</code>) for the timing diagrams.
<code>timing/font=<i><code></i></code>	Setting	Can be used to set font macros. Needs to be used instead of the normal TikZ font as it resets unit etc. if it is font sized dependent.
<code>timing/wscale=<i><math expression></i></code>	Setting	Sets the width-scale <code>\wscale</code> by calling <code>\tikztimingsetwscale</code> . See section 2.5 for further details.

Slopes

TikZ Key	Type	Description
<code>timing/slope=<i><0.0-1.0></i></code>	Setting	Sets slope for logic transitions. This also sets <code>dslope=2*slope</code> , <code>zslope=slope/2</code> .
<code>timing/lslope=<i><0.0-1.0></i></code>	Setting	Sets slope for logic transitions only. Default: 0.1
<code>timing/dslope=<i><0.0-1.0></i></code>	Setting	Sets slope for data transitions. Default: 0.2
<code>timing/zslope=<i><0.0-1.0></i></code>	Setting	Sets slope for Z transitions. Default: 0.05

Texttiming

TikZ Key	Type	Description
<code>timing/outer sep=<dim></code>	Setting	Sets outer separation around <code>\texttiming</code> macros.
<code>timing/outer xsep=<dim></code>	Setting	See above. Only X-Coordinate. (Default: 0pt)
<code>timing/outer ysep=<dim></code>	Setting	See above. Only Y-Coordinate. (Default: 0pt)
<code>timing/before text</code>	Action	Code executed direct before the timing TikZ code inside the internal <code>\tikzpicture</code> for <code>\texttiming</code> . This code is executed just after the <code>timing/before</code> code. Defaults to <code>\texttimingbefore</code> .
<code>timing/after text</code>	Action	Code executed direct after the timing TikZ code inside the internal <code>\tikzpicture</code> . This code is executed just before the <code>timing/after</code> code. Defaults to <code>\texttimingafter</code> .

Grid

TikZ Key	Type	Description
<code>timing/grid</code>	Style	Style used for drawing grids. Depends on <code>help lines</code> and <code>timing</code> .
<code>timing/draw grid</code>	Action	Enables background grids for <code>\timing</code> macros.
<code>timing/no grid</code>	Action	Disabled background grids for <code>\timing</code> macros.

Table

TikZ Key	Type	Description
<code>timing/name</code>	Style	Used for the signal name column in <code>tikztimingtable</code> .
<code>timing/table</code>	Style	Used for <code>tikztimingtable</code> . Depends on <code>timing</code> .
<code>timing/table/grid</code>	Style	Used for table grid. Depends on <code>timing/grid</code> .
<code>timing/table/lines</code>	Style	Used for <code>\horlines</code> and <code>\vertlines</code> .
<code>timing/table/rules</code>	Style	Used for <code>\tablerules</code> for top and bottom lines.
<code>timing/table/midrules</code>	Style	Used for <code>\tablerules</code> between table head and body.
<code>timing/table/header</code>	Style	Used for <code>\tableheader</code> . Defaults to <code>timing/name</code> .
<code>timing/table/label header</code>	Style	Used for label header in <code>\tableheader</code> .
<code>timing/table/row header</code>	Style	Used for timing row header in <code>\tableheader</code> .
<code>timing/rowdist=<i><distance></i></code>	Setting	Sets (baseline) distance between rows in a <code>tikztimingtable</code> . Default: 2 (=2×signal height)
<code>timing/coldist=<i><distance></i></code>	Setting	Sets distance between columns in a <code>tikztimingtable</code> . Default: 1 (=1×period width)
<code>timing/before table</code>	Action	Code placed before the tables TikZ code inside the internal <code>tikzpicture</code> .
<code>timing/after table</code>	Action	Code placed after the tables TikZ code at the end of the internal <code>tikzpicture</code> .

Character Styles

TikZ Key	Type	Description
<code>timing/<i><lowercase char></i></code>	Setting	Style for character <i><char></i> . Not used for ‘H’ and ‘L’.
<code>timing/<i><lc char>/background</i></code>	Setting	Background style for characters ‘D’ and ‘U’.
<code>timing/<i><lc char>/text</i></code>	Setting	Text style for character <i><char></i> . Only defined for ‘D’.
<code>timing/text format=<i><macros></i></code>	Setting	Define macros which are placed before the text of ‘D{ <i>text</i> }’. The text is enclosed in braces to allow the last macro to take it as an argument. A <code>\strut</code> is a good choice to ensure a common baseline.

Debug

Some debug settings for users (first group) and the package developer (last group).

TikZ Key	Type	Description
<code>timing/debug/nodes</code>	Action	Enables marking of named inline nodes.
<code>timing/debug/node</code>	Style	Format style for inline node marker label, which itself is a TikZ node.
<code>timing/debug/markcmd=<i><code></i></code>	Setting	TikZ drawing code to draw marker (except label). The code can access the node name as <code>\N</code> .
<code>timing/debug/scope</code>	Style	Format for scope of node markers.
<code>timing/debug/level=<i><integer></i></code>	Setting	Sets debug output level. This is only important for developers.

Other

TikZ Key	Type	Description
<code>timing/expand count=<i><integer></i></code>	Setting	Sets the maximum expand count for the underlying <code>tikzpicture</code> . TikZ only expands the content this number of times. The TikZ default value of 100 is too small for timing diagrams and is changed to 1000 by default. This should be raised (only) if TikZ throws an to-many-expands error.

4 Libraries for Further Characters

All default timing characters described in Table 2.1 are always made available by this package. Further, less-common characters are provided by libraries which are loaded with the macro `\usetikztiminglibrary{<library>}`. This is done to hold the memory usage of this package small and reduce the risk of collisions with user-defined (meta-)characters. The full syntax for the above macro is `\usetikztiminglibrary[<options>]{<library,...>}[<date>]`, like the one for `\usepackage`. The date is used as a version number like for packages and is identical to the date of the `tikztiming` package.

4.1 Arrows

The library ‘`arrows`’ enables two characters ‘`A`’ and ‘`W`’ which draw vertical up and down *ArroW*s. Such arrows are used in timing diagrams to mark special polarized events, like clock edges of another signal.

The width provided with these character is added as whitespace after the “zero-width” arrow: ‘`A2AA`’ results in $\uparrow\uparrow\uparrow$. This space can be avoided by specifying the width to zero: ‘`0A`’. Like the ‘`C`’ and ‘`T`’ characters the subsequent arrow characters are not combined into one.

The arrow tips can be changed using the TikZ styles for this characters. See section 3 for more information. The ‘`A`’ and ‘`W`’ character should only be used which each other, but not together with any other characters except with ‘`S`’ (space).

Table 4.1: Examples for Arrow Characters.

Characters	Resulting Diagram
<code>0A</code>	\uparrow
<code>AAA</code>	$\uparrow\uparrow\uparrow$
<code>3A</code>	$\uparrow\uparrow\uparrow$
<code>3{A}</code>	$\uparrow\uparrow\uparrow$
<code>3A 3A</code>	$\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow$
<code>3a 3a</code>	$\uparrow\uparrow\uparrow$
<code>AW AW</code>	$\uparrow\downarrow\uparrow\downarrow$
<code>3{AW}</code>	$\uparrow\downarrow\uparrow\downarrow\uparrow\downarrow$
<code>3{aw}</code>	$\uparrow\downarrow\uparrow\downarrow$
<code>2S 2A 3W A W</code>	$\uparrow\downarrow\uparrow\downarrow\uparrow\downarrow$

4.2 Either High or Low

The library ‘`either`’ enables the ‘E’ character which stands for ‘*either high or low*’. This character is designed to be used with the ‘H’ and ‘L’ characters to display a uncertainty of a transition. Sometimes a, e.g. low signal can go either to high or stay at low for a certain time before it definitely goes to high. In this case both (or more) possible transitions can be drawn using this character. Like the ‘C’ and ‘T’ characters subsequent ‘E’ characters are not combined into one.

The drawing style can be changed using the `timing/e` and `timing/e/background` TikZ style.

Table 4.2: Examples for the ‘E’ Character.

Characters	Resulting Diagram
L E H	
L D H	
H E L	
H D L	
L E E H	
L 3{.25E} H	
H E E L	
L EEE HH EEE L	
l e e h	
h e e l	
H 2E L	
H 2{E} L	
H 5{e} L	
H E E H	
L E E L	

Settings (<code>timing/e/.cd</code>)	Resulting Diagram (LL EE HH)
<code>.style={dotted,gray}</code>	
<code>background/.style={fill=gray}</code>	

4.3 Overlays

The library ‘overlays’ enables the ‘O’ character which allows the overlaying of timing characters, i.e. different groups of timing characters are drawn on top of each other. This is not to be confused with ‘dynamic’ overlay provided by the presentation class `beamer`. The `tikz-timing` library `beamer` provides some support for such overlays.

The ‘O’ character awaits a set of character enclosed by braces which are drawn as normal. The position before the ‘O’ character is then restored and the following characters are drawn over them. Older versions of this character awaited a second set of characters in braces but this braces are now optional. The exact syntax is:

⟨chars before⟩ O{*⟨background chars⟩*} {*⟨foreground chars⟩*} *⟨chars after⟩*

or, without second set of braces, but equal:

⟨chars before⟩ O{*⟨background chars⟩*} *⟨foreground chars, ...⟩*

It is the responsibility of the user to make sure that the lines drawn by the first set reconnect to the main lines or do something else useful. The modifier ‘;’ can be used to restart the drawn line, e.g. to change to a different color. This is not done automatically to give the user the freedom if and where this should happen. It is recommended to start and end the set with characters identical with the main line to avoid ugly connection points.

Please note that the width of the first argument is ignored and does not count to the total width of the diagram line. The characters following the overlay should therefore be as wide or wider as the one of the overlay, otherwise the bounding box and background grid will be incorrect.

Overlays can be cascaded , i.e. an overlay can be included in the first argument of another overlay.

New in v0.7

Table 4.3: Examples for the ‘O’ Overlay Character.

Characters	Resulting Diagram	Characters	Resulting Diagram
LLL O{HH}{LL} HHH		ZZ O{dDZ}O{DZ}{dZ} ZZ	
LLL O{HHH}{LL} HHH		ZZ 3D O{dDZ}O{DZ}{dZ} ZZ	
LLL O{;[gray]HH.1H;}{LLH} HH		ZZ 3D O{dDZ}O{DZ}{dZ} ZZ	
LL O{L;[gray]HH.1H;}{LLLH} HH		ZZ 3D O{3D} DZZ	
DD{ } O{zd}{D}d 2D		Z O{DD} ZDDD O{DDZZ} DZ 2S	
ZZ O{Z D Z}{Z 1.1M .9Z} ZZ		Z O{6D Z}{Z 4D Z} Z	
ZZ O{d Z}O{DZ}{dD} ZZ		Z O{8D Z}O{Z 6D Z}{2Z 4D 2Z} Z	

4.4 Clock Arrows

New in v0.7

The library ‘clockarrows’ is changing the ‘C’ clock character to contain arrows which mark the rising and/or falling clock edge. By default the rising edges are marked. To simplify the implementation only the transition from a ‘C’ to another ‘C’ character contains the arrows but not transitions from or to different characters, like ‘HCH’ or ‘LCL’.

The arrows can be controlled using the TikZ styles shown in Table 4.4 below. This styles can also be used as library options. The key “directory” ‘timing/c’ must be dropped for options, e.g.

```
\usetikztiminglibrary[rising arrows]{clockarrows}.
```

Table 4.4: TikZ Styles for Clock Arrows.

TikZ Style	Description
<code>timing/c/rising arrows</code>	Mark (only) rising edges with arrows.
<code>timing/c/falling arrows</code>	Mark (only) falling edges with arrows.
<code>timing/c/dual arrows</code>	Mark both rising and falling edges with arrows.
<code>timing/c/no arrows</code>	Do not mark any edges with arrows. (Default)
<code>timing/c/arrow</code>	Style for arrows. Can be modified to change arrow tip etc. (Default: <code>{}</code>)
<code>timing/c/arrow pos=<0-1.></code>	Position of arrows, i.e. its tip, on the edge. May needs adjustment if different arrow tip shapes are selected. (Default: 0.95)
<code>timing/c/arrow tip=<name></code>	Tip shape of arrows. See the PGF manual for the list of arrow tips. (Default: ‘to’)

Table 4.5: Examples for the Clock Arrows.

Settings (<code>timing/c/.cd</code>)	Resulting Diagram (<code>11{C}</code>)	Settings (<code>timing/c/.cd</code>)	Resulting Diagram (<code>11{C}</code>)
<code>rising arrows</code>		<code>arrow pos=.7</code>	
<code>falling arrows</code>		<code>arrow pos=.4</code>	
<code>no arrows</code>		<code>arrow tip=latex</code>	
<code>dual arrows</code>		<code>arrow tip=stealth</code>	

4.5 Column Type

The library ‘`columnType`’ uses the `array` package to define a new `tabular` column type for timing characters. The `tabular` column can then hold timing characters like the `tikztimingtable`. An initial option block ‘`[...]`’ is taken as initial character or diagram line wide settings. The main difference between these two table types is that `tikztimingtable` creates a big common `tikzpicture` with one coordinate system with potential extra drawing code and the column type creates single pictures for each diagram line without common coordinate system.

New in v0.7

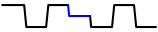
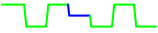
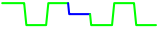
By default the letter ‘T’ and left alignment are used for the timing column type. The TikZ style `timing/columnType={⟨letter⟩}{⟨alignment⟩}` can be used to select different column letters and alignments. The `⟨alignment⟩` represents here the *real* column letter like ‘l’, ‘c’ or ‘r’. Additional column code can be added using the ‘>{⟨code⟩}’ and ‘<{⟨code⟩}’ argument of the `array` package.

More complex column types (e.g. one holding `@{. .}` or `!{. .}` arguments, multiple columns, etc.) must be defined manually using `array`’s `\newcolumnType` macro. The default definition is equal to

```
\newcolumnType{T}{>{\celltiming}l<{\endcelltiming}}.
```

The default ‘T’ definition can be suppressed by using either the library option `notype` or a different type (equal to `timing/columnType`):

```
\usetikzlibrary[type={U}{1}]{columnType}
```

Description	Num	Signal
Example	A	
Example	B	
Example	C	

```

1 \begin{tabular}{lcT}
2   \toprule
3   Description & Num & \multicolumn{1}{l}{Signal} \\
4   \midrule
5   Example & A & HLHZLHL \\
6   Example & B & [green] HLHZLHL \\
7   Example & C & [green] HLHZLHL \\
8   \bottomrule
9 \end{tabular}

```

Figure 4.1: Example for use of Timing Column Type.

4.6 Nice Timing Tables

The library ‘`nicetabs`’ uses the settings of the `array` and `booktabs` packages to improve the layout of `tikztimingtables`. The resulting table matches a `tabular{rT}` table which uses the above packages and the `columntype` library. The table macros `\tabcolsep`, `\arraystretch` and `\extrarowheight` are obeyed.

New in v0.7

The original table layout is designed to produce integer coordinates between the rows to simplify the drawing of extra drawings (see `\extracode`). The improved layout will cause non-integer coordinates, but in-line nodes and the `\rowdist` and `\colldist` macros can be used to draw extra material relatively to the rows.

The TikZ styles ‘`timing/nice tabs`’ (default) and ‘`timing/no nice tabs`’ can be used to activate and deactivate the layout, e.g. for each table. Both settings can be given as a library option, but without the ‘`timing/`’ path.

Table 4.6: Timing tables using ‘nice’ (left) and normal (right) Layout. For comparison a `{tabular}{rT}` table is placed in grey below the left table.

Name	Timing
Example	
Example	
Example	

Name	Timing
Example	
Example	
Example	

4.7 Counter Character

The library ‘counters’ allows the easy definition of meta-characters which implement up-counters. These characters show the counter value using the ‘D{ }’ character and increment it by one every time they are used. A second character can be defined to (re-)set the counter to a specific value. The counter values can be decimal (base-10, default), hexadecimal (base-16) or any other base from 2 to 36. By default the lower case version of the counter character is defined to produce the same output only with half the width.

New in v0.7

*Extended in
v0.7a*

Counter characters are defined using the TikZ key ‘timing/counter/new={char=<char>,<settings>}’ which can also be written as ‘timing/new counter’. The <settings> are TikZ keys themselves and are shown by Table 4.7. One or more ‘new’ keys (path ‘timing/counter’ removed) can be given as library options. The counter values are global like normal L^AT_EX counters. They should be reset in every timing diagram line before they are used.

Counter Style

The styles ‘timing/counter/<char>’ and ‘timing/counter/<char>/text’ (both initially empty) are used to format the graphic and text style of this counter, respectively. Because the ‘D{ }’ character is used internally the above styles need to change the corresponding ‘D’ styles. This changes are only local and do not affect further ‘real’ ‘D’ characters.

The settings ‘fg style’, ‘bg style’ and ‘text style’ can be used to quickly define the foreground (i.e. line), background and text style of the counter. While the ‘text style’ setting simple sets the ‘timing/counter/<char>/text’ style, the other two are a shortcut for

```
\tikzset{timing/counter/<char>/.style={%
  timing/d/.style={<fg style>},
  timing/d/background/.style={<bg style>},
}}
```

Additional Macros

```
\tikztimingcounter{<char>}
\tikztimingsetcounter{<char>}{<pgfmath expression>}
```

The value of counter *<char>* can be read or set using this macros.

Examples:

Counter character ‘Q’ with base 16 (hexadecimal). ‘R’ resets the counter. The counter value should be in blue text typer font.

```
\tikzset{timing/new counter={char=Q,base=16,reset char=R}}
\tikzset{timing/counter/Q/text/.style={font=\ttfamily,blue}}
\texttiming{ 3Q 3{Q} R 12{Q} 2R Q qq OR 3{Q} }
```

gives:

0	1	2	3	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Table 4.7: Settings for Counter Meta-Characters

Key name	Description
<code>char=<char></code>	Defines the given <code><char></code> to be a counter meta-character. If it is a upper case character the lower case character will produce the same output but with the half width, as long this is not overwritten with the <code>half width char</code> key.
<code>half width char=<char></code>	Defines the given <code><char></code> to be the half width version of the counter value. By default this is the lower case version of the counter character given with <code>char</code> . An empty value for <code><char></code> deactivates the definition of a half width character.
<code>reset char=<char></code>	Defines the given <code><char></code> to (re-)set the counter value to the ‘width’ of the character, i.e. the number preceding it. The lower case version of the reset <code><char></code> is not defined.
<code>reset type=<width—arg—both—Both></code>	Defines the type of the reset character, i.e. how the reset value is obtained. <ul style="list-style-type: none"> <code>width</code> Width is reset value: ‘<code><value><char></code>’, e.g. ‘OR’. Value can not be negative. <code>arg</code> Reset value is provided as argument: ‘<code><char>{<value>}</code>’, e.g. ‘R{-1}’. <code>both</code> Uppercase <code><char></code> is <code>width</code>-type, lowercase <code><char></code> is <code>arg</code>-type reset char. <code>Both</code> Lowercase <code><char></code> is <code>width</code>-type, uppercase <code><char></code> is <code>arg</code>-type reset char.
<code>base=<Num 2-36></code>	Defines the numeric base of the counter. If not used the base 10 is used.
<code>increment=<pgfmath expression></code>	Sets the increment which is added every time the counter character is used. This can be a formula which result is truncated to a integer. The current counter value can be referenced as <code>\N</code> . The increment can be negative which causes the counter to count down. Default: 1
<code>max value=<pgfmath expression></code>	Sets the maximum counter value. Default: <i>not set</i>
<code>min value=<pgfmath expression></code>	Sets the minimum counter value. Default: <i>not set</i>
<code>wraps=<true—false></code>	If set to <code>true</code> the counter wraps around, i.e. it counts to the minimum value when counting over the maximum value or the other way around if <code>increment</code> is negative. Initial value: <code>false</code> . Default value: <code>true</code>
<code>bg style=<TikZ style(s)></code>	Sets the background style of the counter.
<code>fg style=<TikZ style(s)></code>	Sets the foreground (line etc.) style of the counter.
<code>text style=<TikZ style(s)></code>	Sets the text style of the counter.
<code>text format=<TEXcode></code>	Sets the format code of the counter value. This should be a macro which receives the counter value as first argument.

Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
Bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000
Oct	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017	000
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0

```

1 \scalebox{2}{%
2 \begin{tikztimingtable}
3   Dec & [timing/counter/new={char=Q,max value=15, wraps,text style={font=\scriptsize}}] 33{Q}d\\
4   Bin & [timing/counter/new={char=Q,max value=15,base=2, digits=4,wraps,text style={font=\tiny,scale
5     =.8}}] 33{Q}d\\
6   Oct & [timing/counter/new={char=Q,max value=15,base=8, digits=3,wraps,text style={font=\tiny}}]
7     33{Q}d\\
8   Hex & [timing/counter/new={char=Q,max value=15,base=16, wraps,text style={font=\scriptsize}}] 33{Q}
9     d\\
10  \extracode
11  \begin{background}[shift={(0.1,0)},help lines]
12    \vertlines{}
13  \end{background}
14 \end{tikztimingtable}
15 }%
```

4.8 Advanced Nodes

New in v0.7

The library ‘advnodes’ changes the in-line nodes, i.e. the ‘N’ character, to provide multiple transition dependent and independent node anchors shown in Table 4.8.

Most transitions provide the three logic level anchors ‘low’, ‘mid’ and ‘high’ which lie on the drawn timing line. Transitions of ‘double line’ characters like ‘D’ can have two low and/or high level anchors which are called ‘low2’ and ‘high2’.

To align marker lines over multiple rows inside a `tikztimingtable` a set of transition independent node anchors are provided: ‘LOW’, ‘MID’, ‘HIGH’. This anchors lie all at the begin of the transition at the appropriate logic levels. With the normal coordinate-like in-line nodes the vertical node (center) position has to be taken into account, while this advanced anchors do this automatically.

Often marker lines should start and end with a little vertical offset from the timing diagram to be more distinguishable. For this the two anchors ‘TOP’ and ‘BOTTOM’ exist. They lie above and below of ‘HIGH’ and ‘LOW’, respectively. The vertical distance can be set using ‘`timing/nodes/offset=<dim. or number>`’.

Please note that the node *center* of advanced nodes will be different for some transitions. The ‘center’ anchor (used by default if no node anchor is provided) will be placed at the logical center of the transition, i.e. mostly in the middle of it. In order not to break existing diagrams which use nodes as references to draw additional material an TikZ styles is provided to select between the old and new node centers. This styles can be used globally or locally as required. The two explicit anchors ‘new center’ and ‘old center’ are also provided. For existing documents with diagrams using normal nodes it is recommended to switch to simple nodes or to the old node centers globally or select such a style for each old diagram.

The following TikZ settings can be used with the library. The node style settings affect all new nodes after them. The center settings affect all following *references* (e.g. ‘(NodeName)’ or ‘(NodeName.center)’) of advanced nodes. The settings can be used as library options with the ‘`timing/nodes/`’ part removed.

TikZ Setting	Description
<code>timing/nodes/advanced</code>	Selects advanced in-line nodes. (library default)
<code>timing/nodes/simple</code>	Selects simple coordinate-style in-line nodes. (package default)
<code>timing/nodes/new center</code>	Center of in-line nodes is in the new position. (default for advanced)
<code>timing/nodes/old center</code>	Center of in-line nodes is in the old position. (always on for simple)
<code>timing/nodes/offset</code>	Sets offset for TOP and BOTTOM anchors. Can be a dimension with unit or a factor to the current y unit. (default: 0.25)

Examples:

`\usetikztiminglibrary[simple]{advnodes}` loads the library with nodes default to the old ‘simple’ style.

`\usetikztiminglibrary[old center]{advnodes}` loads the library with advanced nodes with have the center at the same place as the normal simple nodes. This is a good “compatibility mode” for existing pre-v0.7 diagrams.

`\begin{tikztimingtable}[timing/nodes/simple]` starts a timing table which uses simple nodes.

`\begin{tikztimingtable}[timing/nodes/.cd,advanced,old center]`

starts a timing table which uses advanced nodes with old node centers.

Table 4.8: Transition Dependent Anchor Points of Advanced Nodes

	$\begin{matrix} \text{to} \\ \swarrow \\ \text{from} \end{matrix}$	L	H	X	D	E	
L							
H							
X							
D							
D{A}							
E							
OLE							
OHE							

Legend:

- low
- mid
- high
- low2
- mid2
- high2
- × LOW
- × MID
- × HIGH
- new center
- + old center

4.9 Compatibility Macros for `ifsym` package

New in v0.7

The library ‘`ifsym`’ provides macros and timing styles to emulate the behaviour of the `ifsym` package when loaded with the `electronic` option. The `ifsym` package was an early inspiration to this package and this library allows the usage of `ifsym` style timing symbol macros and characters (`\textifsym{characters}`) which uses `\texttiming[timing/ifsym]{characters}` which are described in Table 4.9 and Table 4.10, respectively. This is useful if old `ifsym` timing diagrams should be reused. The `tikz-timing` replacements are a very close match and do not need a special font to be installed. The graphic quality should be equal or better than the original. The intermixing of `ifsym` and `tikz-timing` style characters in a `\textifsym` macro (the one provided by this library, not the one from the `ifsym` package) is supported but it is not guaranteed to work 100% properly. Please note that the ‘M’ character is defined to use ‘X’ in black.

The library can be loaded with one of the options ‘`provide`’ (default), ‘`new`’, ‘`renew`’ or ‘`off`’, respectively. These select if the macros should be defined using `\providecommand`, `\newcommand`, `\renewcommand` or not at all. This can be useful if the `ifsym` package is loaded beforehand.

Table 4.9: `ifsym` style Timing Symbol Macros
















Macro	Symbol	Description (trivial)
<code>\RaisingEdge</code>		Raising Edge
<code>\FallingEdge</code>		Falling Edge
<code>\ShortPulseHigh</code>		Short Pulse High
<code>\ShortPulseLow</code>		Short Pulse Low
<code>\PulseHigh</code>		Normal Pulse High
<code>\PulseLow</code>		Normal Pulse Low
<code>\LongPulseHigh</code>		Long Pulse High
<code>\LongPulseLow</code>		Long Pulse Low

Table 4.10: `ifsym` style Timing Characters (from `ifsym` manual)

Character	Symbol	Description
<code>l, h</code>		Short low or high level signal.
<code>L, H</code>		Long low or high level signal.
<code> </code>		Transition/glitch between L/H or H/L levels.
<code>m, d</code>		Short middle or double level signal.
<code>M, D</code>		Long middle or double level signal.
<code><, <<</code>		Short or long slope between middle and double level.
<code>>, >></code>		Short or long slope between double and middle level.

4.10 Intervals (experimental)

This library is under development and might change in the future.

The library ‘interval’ allows the drawing of intervals using the ‘H’, ‘L’, ‘Z’ and ‘X’ logic levels. It provides modified definitions of X and Z transitions (e.g. ‘LX’, ‘XH’) where the transition edges can be coloured either way to indicate interval borders.

New in v0.7

The interval borders can be set using the ‘timing/interval=*<settings>*’ TikZ style. The *<settings>* are ‘lo’ (left-open), ‘lc’ (left-closed) and ‘ro’ (right-open), ‘rc’ (right-closed), which build the following combinations: ‘{lo,ro}’, ‘{lc,ro}’, ‘{lo,rc}’ and ‘{lc,rc}’. However, every of them can also be set on its own, e.g. ‘timing/interval={lc}’ simply sets the interval to ‘left-closed’ without changing the right border.

The key ‘timing/interval/normal’ (alternative: ‘timing/interval={normal}’) sets the transitions back to their default.

Examples:

	ro	rc
lo		
lc		

A meta-character can be defined for quick changes if necessary:

```
\tikztimingmetachar{Y}{[timing/interval={lo,ro}]}
\tikztimingmetachar{y}{[timing/interval/normal]}
\texttiming[timing/draw grid]{ LZH Y HZH y LZH Y LZL }
\texttiming[timing/draw grid]{ LXH Y HXH y LXH Y LXL }
gives: 
```

A further alternative is to use a meta-character with an argument. Note that ‘#2’ must be used, because ‘#1’ holds the width, which is irrelevant here. This definition is also provided by the ‘timing/interval/metachar=*<Character>*’ key.

```
\tikztimingmetachar{I}[2]{[timing/interval={#2}]}
or \tikzset{timing/interval/metachar=I}
\texttiming[timing/draw grid]{ LZH I{lo,rc} HZH I{ro} LZH I{normal} LZL }
gives: 
```

4.11 Beamer Overlay Support (experimental)

This library is under development and might change in the future.

The library ‘beamer’ provides (at the moment) marginal support for overlays of the `beamer` class. It allows the usage of beamer *overlay specifications* (`<spec>`) inside the timing string. However, the current simple implementation might not work properly and cause strange results. The support is designed for use inside `tikztimingtable`. See the `beamer` manual for more informations about overlays specifications.

New in v0.7

Usage

Insert an overlay specification, e.g. `<number>`, inside the timing string. It will affect the rest of the timing characters of the current diagram line. Unfortunate due to the global nature of overlays it also affects the rest of the table. Therefore all diagram lines should end with a specification which turns overlays off, i.e. `<*>` or `<0->`. Otherwise strange results can occur (e.g. wrong/missing background graphics).

Example Timing Table Row & H L <2> Z L H <3> D{last} Z <*> \\

Display Rows Stepwise

The rows of a `tikztimingtable` can be uncovered row-by-row using the way shown below. The signal names must be enclosed into a `\mbox` because `\uncover` needs to be inside horizontal mode. Instead of `\uncover` another beamer overlay command like `\only` or `\invisible` can be used. To highlight the signal name use `\alert<...>{signal name}` inside `\uncover`. At the moment there is no simple way to highlight the timing lines.

```
\begin{tikztimingtable}
  \mbox{\uncover<+>{Signal Name 1}} & <.-> HL <*> \\
  \mbox{\uncover<+>{Signal Name 2}} & <.-> HL <*> \\
  % ...
  \mbox{\uncover<+>{Signal Name n}} & <.-> HL <*> \\
\end{tikztimingtable}
```

Display Columns Stepwise

Different sections ('columns') of timing diagrams in a `tikztimingtable` can be uncovered stepwise using the way shown below. In this example the second section/column will be uncovered in the second slide. The first is always visible. Further sections/columns can be uncovered in further slides.

Please note that the total width of the table is constant and e.g. `\tablerules` will always cover the full width independent of overlays.

```
\begin{tikztimingtable}
  Signal Name 1 & 10D{Sec. 1} <2> 10D{Sec. 2} <*> \\
  Signal Name 2 & 10D{Sec. 1} <2> 10D{Sec. 2} <*> \\
  % ...
  Signal Name n & 10D{Sec. 1} <2> 10D{Sec. 2} <*> \\
\end{tikztimingtable}
```

Overlay Extra Code

The `beamer` overlay specifications can be used inside the `\extracode` section like in a normal `tikzpicture` environment. However, in both cases strange results will occur if the end of the environment is hidden by an overlay specification. Due to this reason it is recommended to only use overlay commands which affect only an argument, like `\only<...>{<code>}`, or to place a plain `\onlayer` before the end of the environment.

```
\begin{tikztimingtable}
  Signal Name 1 & 10D{Sec. 1} <2> 10D{Sec. 2} <*> \\
  % ...
\extracode
  % either
  \draw<2> (0,0) -- (2,0); \only<3> { ... }
  % or
  \onlayer<2>
  % and then at the very end:
  \onlayer % or \onlayer<*>
\end{tikztimingtable}
```

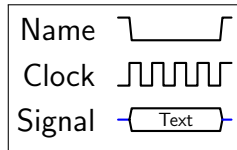
5 Examples

This section shows some examples by putting either the full source code or only the needed characters beside the graphical result. Please note that the displayed syntax is the one of `\timing` where the initial character is declared as optional argument (`[<char>]`) *inside/together* with the logic characters. The syntax of `\textttiming` is identical except the initial character is given as a normal optional argument before the characters argument. All examples except Example 1 are attached in compilable form to this PDF.

Example 1: Initial Characters, Modifiers, TikZ Keys

Characters	Resulting Diagram
HLXZDUTC	
cccc	
tttt	
[c]cccc	
4{c}	
4c4c	
4{1.8c}	
[d] 4{5D{Text}} 0.2D	
3.4H 0.6L	
DDDUUUDDD	
DDD{ }DUUDD	
8{2D{\hexcountmacro }}	
3{2{0.25X 2.5D .25Z}}	
DDD{ } 3{0.2D{}}	
DDD{ } 3{0.2D{}} 0.4D{ } 0.6D{ } DDD	
HHHLLH SSSS HLLHHL	
HHGHHGGHLLGLLGH	
ZZ G ZZ G XX G X	
LLL 2{0.1H 0.1L} 0.6H HH	
LLL [timing/slope=0.05] 4{.05H .05L} 0.6H HH	
LLL 0.4U 0.6H HH	
[L] [timing/slope=1.0] HL HL HL HL HL	
LLLLL !{-- +(.5,.5) -- ++(1,0)} HHHHHH	
LLLLL [/utils/exec={\somemacro \code }] HHHHHH	
LL [green] HH [brown] XX LL ZZ [orange] HH	
[] [line width=1pt] HLXZDU [line width=0.1pt] HLXZDU	
[] [line width=1pt] HLXZDU , [line width=0.1pt] HLXZDU	
[] [line width=1pt] HLXZDU ; [line width=0.1pt] HLXZDU	

Note: Optional argument must be placed before macro argument if `\texttiming` is used.

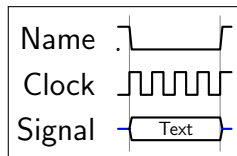


```

1 \begin{tikztimingtable}
2   Name & hLLLLh \\
3   Clock & 10{c} \\
4   Signal & z4D{Text}z \\
5 \end{tikztimingtable}

```

Example 2: `tikztimingtable` without `\extracode`.

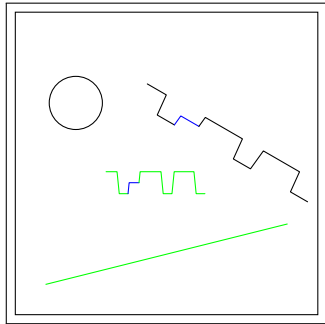


```

1 \begin{tikztimingtable}
2   Name & hLLLLh \\
3   Clock & 10{c} \\
4   Signal & z4D{Text}z \\
5 \extracode
6   \draw (0,0) circle (0.2pt); % Origin
7   \begin{pgfonlayer}{background}
8     \vertlines[help lines]{0.5,4.5}
9   \end{pgfonlayer}
10 \end{tikztimingtable}

```

Example 3: `tikztimingtable` with `\extracode`.

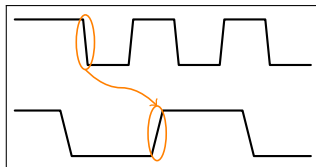


```

1 \begin{tikzpicture}[x=4cm,y=4cm]
2 \draw (0,0) rectangle (1,1);
3 \draw (0.2,0.7) circle (10pt);
4 \begin{scope}[green]
5 \draw (0.1,0.1) -- +(0.8,0.2);
6 \timing at (0.3,0.4) {hlzhhlhhl};
7 \end{scope}
8 \timing [rotate=-30]
9 at (0.4,0.7) {HLZHHLHHL};
10 \end{tikzpicture}

```

Example 4: `\timing` inside general `tikzpicture`.

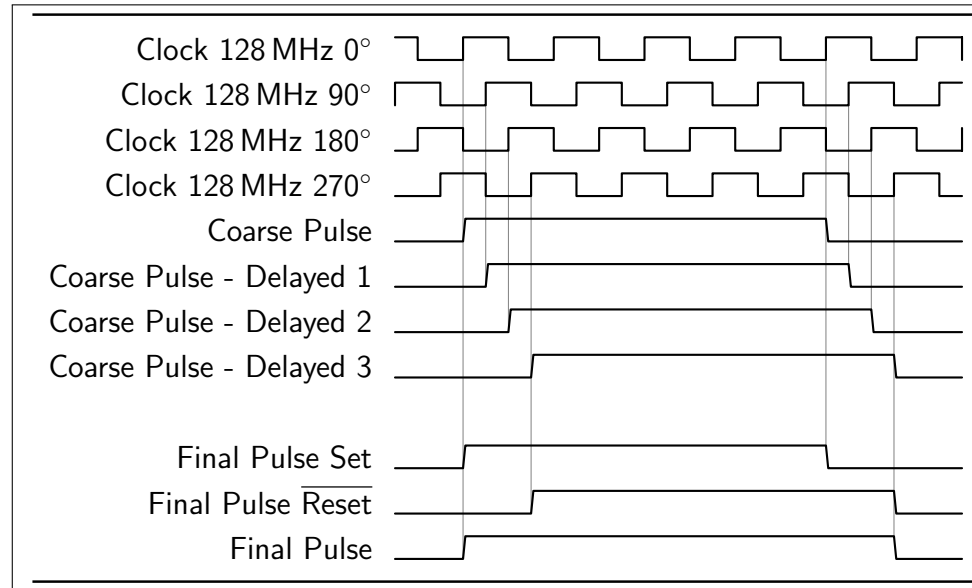


```

1 \Huge
2 \begin{tikzpicture}[timing/picture,thick,
3 timing/nodes/advanced]
4 \timing at (0,2) {hH N(A) LHLHL};
5 \timing[timing/slope=.25] at (0,0)
6 {HLL N(B) HHL};
7 \draw [orange,semithick]
8 (A.mid) ellipse (.2 and .6)
9 (B.mid) ellipse (.2 and .6);
10 \draw [orange,semithick,->]
11 ($ (A.mid) - (0,.6) $)
12 parabola [bend pos=0.5]
13 ($ (B.mid) + (0,.6) $);
14 \end{tikzpicture}

```

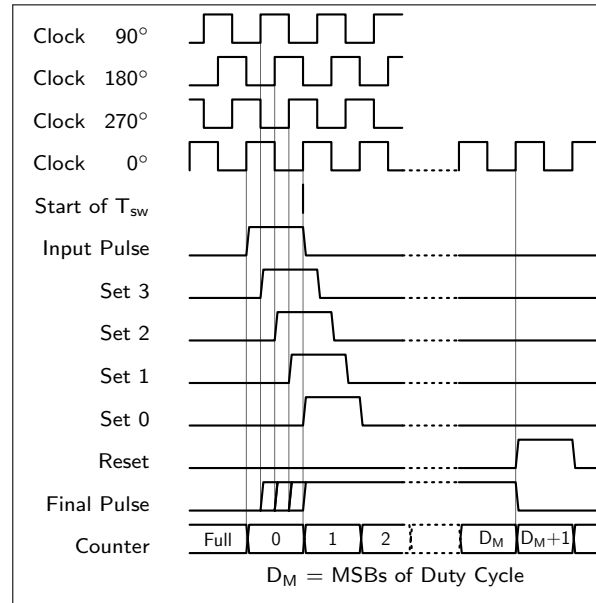
Example 5: Using In-Line Nodes to draw Relationships.



```

1 \def\degr{${}^\circ}
2 \begin{tikztimingtable}
3   Clock 128\MHz 0\degr & H 2C N(A1) 8{2C} N(A5) 3{2C} G \\
4   Clock 128\MHz 90\degr & [C] 2{2C} N(A2) 8{2C} N(A6) 2{2C} C \\
5   Clock 128\MHz 180\degr & C 2{2C} N(A3) 8{2C} N(A7) 2{2C} G \\
6   Clock 128\MHz 270\degr & 3{2C} N(A4) 8{2C} N(A8) 2C C \\
7   Coarse Pulse & 3L 16H 6L \\
8   Coarse Pulse - Delayed 1 & 4L N(B2) 16H N(B6) 5L \\
9   Coarse Pulse - Delayed 2 & 5L N(B3) 16H N(B7) 4L \\
10  Coarse Pulse - Delayed 3 & 6L 16H 3L \\
11                                     \\
12  Final Pulse Set & 3L 16H N(B5) 6L \\
13  Final Pulse  $\overline{\mbox{Reset}}$  & 6L N(B4) 16H 3L \\
14  Final Pulse & 3L N(B1) 19H N(B8) 3L \\
15 \extracode
16   \tablerules
17   \begin{pgfonlayer}{background}
18     \foreach \n in {1,...,8}
19       \draw [help lines] (A\n) -#6(B\n);
20   \end{pgfonlayer}
21 \end{tikztimingtable}

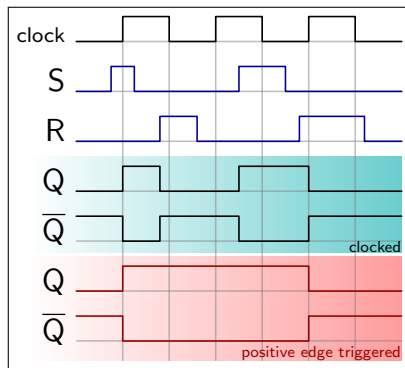
```

```

1 \def\degr#1{\makebox[2em][r]{#1}\ensuremath{{}^{\circ}}}
2
3 \begin{tikztimingtable}[%
4     timing/dslope=0.1, timing/.style={x=2ex,y=2ex}, x=2ex,
5     timing/rowdist=3ex,
6     timing/name/.style={font=\sffamily\scriptsize},
7     timing/nodes/advanced,
8 ]
9 Clock \degr{90} & 1 2{C} N(A1) 5{C} \\\
10 Clock \degr{180}& C 2{C} N(A2) 4{C} c\\
11 Clock \degr{270}& h 3{C} N(A3) 4{C} \\\
12 Clock \degr{0} & [C] 2{C} N(A0) 2{C} N(A4) 3{C}c ;[dotted]
13             2L; 2{C} N(A5) 3{C} \\\
14 Start of T$_{\text{sw}}$ & 4S G N(start) \\\
15 Input Pulse & 2.0L 2H 3.5L ;[dotted] 2L; 5L \\\
16 Set 3 & 2.5L 2H 3.0L ;[dotted] 2L; 5L \\\
17 Set 2 & 3.0L 2H 2.5L ;[dotted] 2L; 5L \\\
18 Set 1 & 3.5L 2H 2.0L ;[dotted] 2L; 5L \\\
19 Set 0 & 4.0L 2H 1.5L ;[dotted] 2L; 5L \\\
20 Reset & 7.5L ;[dotted] 2L; 2L N(reset) 2H 1L \\\
21 Final Pulse & 2.5L N(B1) e N(B2) e N(B3) e 3.5H; [dotted]

```



```

1 \definecolor{bgblue}{rgb}{0.41961,0.80784,0.80784}%
2 \definecolor{bgred}{rgb}{1,0.61569,0.61569}%
3 \definecolor{fgblue}{rgb}{0,0,0.6}%
4 \definecolor{fgred}{rgb}{0.6,0,0}%
5 \begin{tikztimingtable}[timing/slope=0,
6   timing/coldist=2pt,xscale=2.05,yscale=1.1,semithick]
7   \scriptsize clock & 7{C}\
8   S & .75L h 2.25L H LL1 [fgblue]\
9   R & 1.8L .8H 2.2L 1.4H 0.8L [fgblue]\
10  Q & L .8H 1.7L 1.5H LL\
11  $\overline{\mbox{Q}}$ & H .8L 1.7H 1.5L HH\
12  Q & LHHHLL[fgred]\
13  $\overline{\mbox{Q}}$ & HLLLLHH[fgred]\
14 \extracode
15 \makeatletter
16 \begin{pgfonlayer}{background}
17   \shade [right color=bgblue,left color=white]
18     (7,-8.45) rectangle (-1,-4.6);
19   \shade [right color=bgred,left color=white]
20     (7,-12.8) rectangle (-1,-8.6);
21   \begin{scope}[gray,semitransparent,semithick]
22     \horlines{}
23     \foreach \x in {1,...,6}
24       \draw (\x,1) -- (\x,-12.8);
25     % similar: \vertlines{1,...,6}
26   \end{scope}
27   \node [anchor=south east,inner sep=0pt]
28     at (7,-8.45) {\tiny clocked};
29   \node [anchor=south east,inner sep=0pt,fgred]
30     at (7,-12.8) {\tiny positive edge triggered};
31 \end{pgfonlayer}
32 \end{tikztimingtable}%

```

Example 8: SR flip-flop timing diagram. Redrawn from image

http://commons.wikimedia.org/wiki/File:SR_FF_timing_diagram.png

