The ForSyDe–LaTeX utilities

George Ungureanu
Department of Electronic Systems
KTH Royal Institute of Technology
Stockholm, SWEDEN

February 28, 2018

Abstract

This is the reference manual for the LaTeX utilities used in the context of ForSyDe . All packages and their API features are documented here.

1 Introduction

This library was developed as an effort to standardize symbols and graphical primitives in documents related to ForSyDe , but also to provide tools and utilities for user convenience. ForSyDe is a high-level design methodology aiming at synthesizing correct-by-construction systems through formal means. For more information check `https://forsyde.ict.kth.se`.

The library contains the following main packages:

- `forsyde-tikz` : is a collection of PGF and TikZ styles, graphical primitives and commands for drawing ForSyDe process networks;
- `forsyde-math` : is a collection of math symbols used in the ForSyDe formal notation. It is mainly focused on the ongoing ForSyDe–Atommethodology;
- `forsyde-plot` : provides utilities for plotting ForSyDe signals;
- `forsyde-legacy` : API for the previous versions of this library.

2 Installation & usage

The most straightforward way to install ForSyDe–LaTeX is to use the provided GNU Make script command:

```
make install
```

which installs the packages in `TEXMFLOCAL` if you have write access or `TEXMFHOME` otherwise. Refer to `https://en.wikibooks.org/wiki/LaTeX/Installing_Extra_Packages` for more information about the two enviroment variables.

Alternatively, there are three main alternatives to manually install the libraries:

1. copy the contents of `forsyde-latex/src` and `forsyde-latex/fonts` in their appropriate path under `TEXMFHOME` or any standard loading path, as specified by your LaTeX compiler.

2. compile your document with the variable `TEXINPUTS` set to `/path/to/forsyde-latex/src/`. If you intend to use `forsyde-math` characters, you need to generate the fonts under `forsyde-latex/fonts` using a `METAFONT` tool suite, and afterwards compile your document with the variable `TEXFONTS` set to `/path/to/forsyde-latex/fonts/`;

3. copy the contents of `forsyde-latex/src` and `forsyde-latex/fonts` in the same folder as your document and compile normally.

To include any of the packages enumerated in the introduction, you cal load the `forsyde` package with the appropriate option:

```
\usepackage[option]{forsyde}
```

where `option` is

- `tikz` for loading the `forsyde-tikz` library
- `math` for loading the `forsyde-math` library
- `plot` for loading the `forsyde-plot` library
- `legacy` for loading the `forsyde-legacy` library

When loaded without an option, this package only provides some general commands for typesetting and logos:

Command	**Expands to**
`\ForSyDe`	ForSyDe
`\ForSyDeLaTeX`	ForSyDe–LaTeX
`\ForSyDeAtom`	ForSyDe–Atom

Contents

1 Introduction **1**

2 Installation & usage **1**

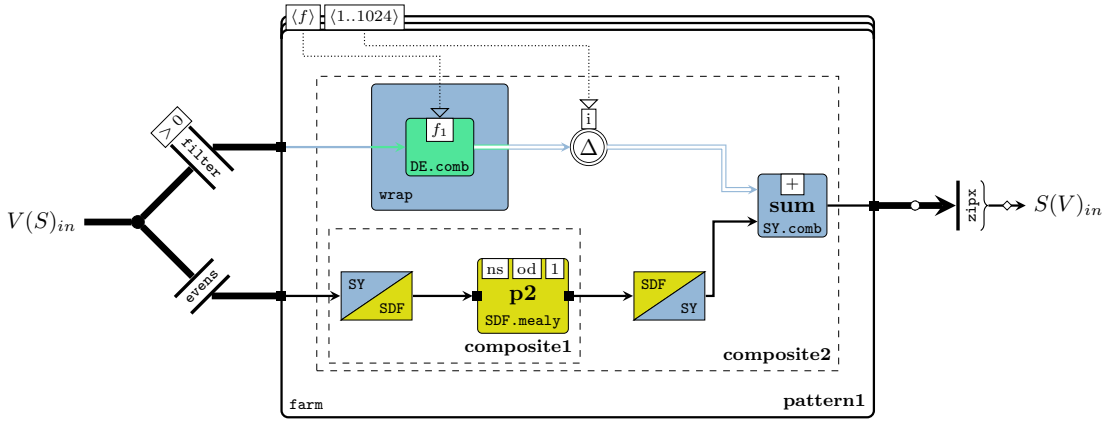
3 The `forsyde-tikz` package **5**
3.1 Main nodes . . . 5
3.1.1 Draw commands . . . 5
3.1.2 Shapes . . . 9
3.1.3 Helper options . . . 10
3.2 Paths . . . 11
3.2.1 Helper commands . . . 13
3.3 Utility commands . . . 13
3.4 Further customizing the environment . . . 14

4 The `forsyde-math` package **16**
4.1 Operator symbols . . . 16
4.2 Math commands . . . 17
4.3 Font map . . . 17

5 The `forsyde-plot` package **19**
5.1 The `signalsSY` environment . . . 19
5.2 The `signalsDE` environment . . . 20
5.3 The `signalsCT` environment . . . 21

6 Libraries of additional pictures **23**
6.1 The `forsyde.pictures.layered` library . . . 23

7 The `forsyde-legacy` package **25**
7.1 `forsyde-tikz v0.3` or prior . . . 25
7.2 `forsyde-pc v0.3` or prior . . . 25




```
\documentclass{standalone}
\usepackage[tikz]{forsyde}
\begin{document}
\begin{tikzpicture}[]
  \trans     [transition, type=evens, rotate=-45]                    (a)             {};
  \trans     [transition, type=filter, rotate=45, nf=1, f1={$>0$} ] (b) <0,2>       {};
  \standard  [process, nf=1, type=comb, moc=de, right of=b, xshift=3cm](c)           {};
  \interface[right of=a, xshift=2cm]                                  (d1)            {sy}{sdf};
  \standard  [process, f={ns;od;1}, type=mealy, moc=sdf, right of=d1](d)             {p2};
  \interface[right of=d]                                              (e)             {sdf}{sy};
  \basic     [primitiven, right of=c, f=i, xshift=1.5cm]              (f)             {$\Delta$};
  \standard  [process, ni=2, anchor=w2, xshift=2cm,
              f=$+$, moc=sy, type=comb]                               (g) <$(e)!.5!(f)$> {sum};
  \cluster   [embed, moc=sy, type=wrap, inner sep=15pt]               (h) <(c)>       {};
  \cluster   [composite,  inner ysep=13pt]                            (i) <(d1)(d)>   {composite1};
  \cluster   [composite, ni=2, inner xsep=5pt]                        (j) <(g)(i)(h)> {composite2};
  \cluster   [farmstyle, type=farm,
              f={$\langle f \rangle$;$\langle 1 .. 1024 \rangle$}]    (k) <(j)>       {pattern1};
  \trans     [zipx, right of=g,xshift=2cm]                            (l)             {};
  % additional info nodes
  \node[connector] (con) at ($ (a)!.5!(b)-(1,0) $){};
  \node (in)  at ($ (con)-(1.5,0) $){$V(S)_{in}$};
  \node (out) at ($ (l)+(1.5,0) $)  {$S(V)_{in}$};
  % singal/vector edges
  \path[v] (con) edge (in) edge (a.w1) edge (b.w1);
  \path (b.e1) edge[intersect=k-west, as=b-c1] (c) edge[intersect=h-west, as=b-c2] (c)
        (b-c1) edge[v]  (b.e1)  edge[s=sy,srcport]   (b-c2)
        (b-c2) edge[s=de,->]                         (c)
        (f)    edge[trans={<-,sn=sy}{h-east}{sn=de}] (c)
               edge[sn=sy,-|-=.8,->]                 (g.w1)
        (a.e1) edge[trans={v,dstport}{k-west}{s,->}] (d1)
        (d)    edge[<-, srcport, s]                  (d1)
               edge[s, srcport, ->]                  (e)
        (e)    edge[s,-|-,->]                        (g.w2)
        (g.e1) edge[intersect=k-east, as=g-l]        (l)
               edge[s]                               (g-l)
        (g-l)  edge[v,srcport,token=scalar,->]       (l)
        (l)    edge[s,token=vector,->]               (out);
  % function edges, suggest the passing of functions as arguments
  \path[f] (k-f.s1) edge[|-|=.3,->] (c.north) (k-f.s2) edge[|-|=.2,->] (f-f.n1);
\end{tikzpicture}
\end{document}
```

3 The forsyde-tikz package

This library is used to draw systems modeled as process networks with ForSyDe , similar to the one on the previous page. A ForSyDe process network is drawn like any other TikZ figure, inside a `tikzpicture` environment. The `options` provided are key variables defined by the `tikz` package.

environment

```
\begin{tikzpicture}[options]
          content...
\end{tikzpicture}
```

environment variables

Apart from the standard keys, `forsyde-tikz` provides the following variables:

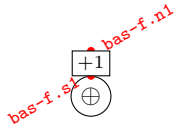
`no moc color` : disables process coloring according to their MoC.
`no moc label` : disables process labeling according to their MoC.
`label style=` : font style for the process type labels. Default is `\textbf`.
`type style=` : font style for the process name labels. Default is `\scriptsize\textit`.
`function style=` : font style for the functions. Default is `\scriptsize`.

3.1 Main nodes

Using `forsyde-tikz`, process networks are drawn using a combination of commands and style options for customizing shapes.

3.1.1 Draw commands

Inside the `tikzpicture` environment, one can use a set of draw commands[1] which are used as templates for placing shapes and information provided as arguments. In the following listing, the gray arguments are optional while the black ones are mandatory. The library also provide as set of helper styles for setting several options at once, presented in 3.1.3.




```
\basic[shape=atom shape, nf=1, f1=$+1$] (bas) <0,0> {$\oplus$};
```

Figure 1: The `\basic` draw command. The red dots are custom anchors, accessed with `[name]-f.[anchor]`.

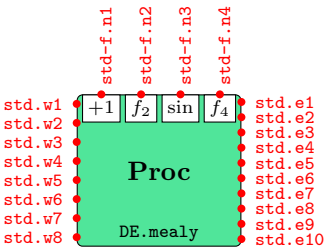
`\basic`[options](name)<position>{label};

The basic node is the simplest depiction of a function (e.g., process, atom) with maximum one argument. It is the most lightweight graphical primitive, and does not contain port anchors. It can be customized with the following keys (check 3.1.3 for composite options):

`shape=` any TikZ style option for the node shape. Check section 3.1.2 for a list of library-provided shapes. Default is `rectangle`.

[1] All draw commands are defined in the `forsyde.nodes` TikZ library, which is loaded by default by the `forsyde-tikz` package, but can also be loaded independently using the command `\usetikzlibrary{forsyde.nodes}` in the document preamble.

`nf=[0..1]` the number of passed functions. Default is `0`. If `nf>1` then `nf=1`.
`f1=` function label. Appears as a box above the main shape in case `nf > 0`. Default is f_1.
`anchor=[anchor]` center point for shape. Default is `center`.
`xshift=[distance]` shift position in X direction. Default is `0pt`.
`yshift=[distance]` shift position in Y direction. Default is `0pt`.




```
\standard[shape=leaf shape, hasmoc, moc=de, ni=8, no=10, type=mealy, nf=4, f1=$+1$, f3=$\sin$,
    inner ysep=15pt] (std) <0,0> {Proc};
```

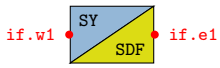
Figure 2: The `\standard` draw command. The function anchors are accessed with `[name]-f.[anchor]` and the port anchors are accessed with `[name].[anchor]`

`\standard[options](name)<position>{label};`

The standard node is a more complex depiction of a ForSyDe block. It contains anchors for each port, shows a label and the constructor name, and its field is colored according to a MoC (if this is the case). It can be customized with the following keys (check 3.1.3 for composite options):

`class=[sy|de|ct|sdf|blackbox|none]` The class of the node. This option affects the field color and the constructor label. Default is `none`.
`hasmoc` flag for saying that this node is a process, therefore it is associated with a MoC. It need to be provided otherwise the global flags will ignore MoC-related options (e.g. `nomoccolor`).
`shape=` any `TikZ` style option for the node shape. Check section 3.1.2 for a list of library-provided shapes. Default is `rectangle`.
`type=` the type / constructor of that particular ForSyDe block. It shows below the main label and it is affected by the `class` argument.
`npw=[0..10]` number of ports on the left side of the node, i.e. the number of left port anchors.
`ni=[0..10]` alias for `npw`, stands for "number of inputs".
`npe=[0..10]` number of ports on the right side of the node, i.e. the number of right port anchors.
`no=[0..10]` alias for `npe`, stands for "number of outputs".
`nf=[0..4]` the number of passed functions. Default is `0`.
`f1=` first function label. Appears as a box in the upper part in case `nf > 0`. Default is f_1.
`f2=` second function label. Appears as a box in the upper part in case `nf > 1`. Default is f_2.
`f3=` third function label. Appears as a box in the upper part in case `nf > 2`. Default is f_3.
`f4=` fourth function label. Appears as a box in the upper part in case `nf > 3`. Default is f_4.
`anchor=[anchor]` center point for shape. Default is `center`.
`xshift=[distance]` shift position in X direction. Default is `0pt`.
`yshift=[distance]` shift position in Y direction. Default is `0pt`.
`rotate=[number]` the rotation angle (in degrees) of the node. Default is `0`.

`rotate shape=[number]` the rotation angle (in degrees) of the `shape`. Defailt is `0`.
`inner ysep=[distance]` the distance between the function node, the label node and the type node. Default is `3pt`.

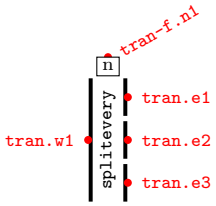



```
\interface[] (if) <0,0> {sy}{sdf};
```

Figure 3: The `\trans` draw command. The function anchors are accessed with `[name]-f.[anchor]` and the port anchors are accessed with `[name].[anchor]`

`\interface`[options](name)<position>{domain left}{domain right};
This command draws an interface from one domain to another (e.g. a MoC interface). It can be customized with the following keys (check 3.1.3 for composite options):

`anchor=[anchor]` center point for shape. Default is `center`.
`xshift=[distance]` shift position in X direction. Default is `0pt`.
`yshift=[distance]` shift position in Y direction. Default is `0pt`.
`rotate=[number]` the rotation angle (in degrees) of the node. Default is `0`.



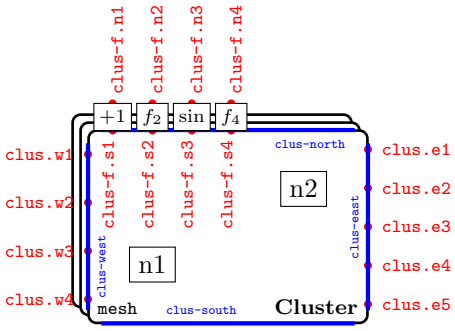

```
\trans[shape=trans shape v1v3, ni=1, no=3, nf=1, f1=n, type=splitevery, inner ysep=5pt] (tran)
    <0,0> {};
```

Figure 4: The `\trans` draw command. The function anchors are accessed with `[name]-f.[anchor]` and the port anchors are accessed with `[name].[anchor]`

`\trans`[options](name)<position>{};
This command draws a transversal node over a path (usually 90 degrees), which symbolizes a structural transition of the path's type (e.g. permutation). The label is ignored, but the brackets are necessary to mark the end of the arguments. It can be customized with the following keys (check 3.1.3 for composite options):

`shape=` any `TikZ` style option for the node shape. Check section 3.1.2 for a list of library-provided shapes. Default is `rectangle`.
`type=` the type / constructor of that particular ForSyDe block. It is shown as the main label, rotated at 90 degrees.
`npw=[0..10]` number of ports on the left side of the node, i.e. the number of left port anchors.
`ni=[0..10]` alias for `npw`, stands for "number of inputs".
`npe=[0..10]` number of ports on the right side of the node, i.e. the number of right port anchors.

`no=[0..10]` alias for `npe`, stands for "number of outputs".
`nf=[0..4]` the number of passed functions. Default is `0`.
`f1=` first function label. Appears as a box in the upper part in case `nf > 0`. Default is f_1.
`f2=` second function label. Appears as a box in the upper part in case `nf > 1`. Default is f_2.
`f3=` third function label. Appears as a box in the upper part in case `nf > 2`. Default is f_3.
`f4=` fourth function label. Appears as a box in the upper part in case `nf > 3`. Default is f_4.
`anchor=[anchor]` center point for shape. Default is `center`.
`xshift=[distance]` shift position in X direction. Default is `0pt`.
`yshift=[distance]` shift position in Y direction. Default is `0pt`.
`rotate=[number]` the rotation angle (in degrees) of the node. Default is `0`.
`rotate shape=[number]` the rotation angle (in degrees) of the `shape`. Defailt is `0`.
`inner ysep=[distance]` the distance between the function node, the label node and the type node. Default is `3pt`.




```
\node[draw] (n1) at (0,0) {n1};
\node[draw] (n2) at (2,1) {n2};
\cluster[shape=generic skel shape, ni=4, no=5, type=mesh, nf=4, f1=$+1$, f3=$\sin$, inner sep=15
    pt] (clus) <(n1)(n2)> {Cluster};
```

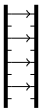
Figure 5: The `\cluster` draw command. The function anchors are accessed with `[name]-f.[anchor]`, the port anchors are accessed with `[name].[anchor]` and the cluster outer shape edges are named `[name]-[position]`

`\cluster[options](name)<list of clustered nodes>{label};`

This command draws a cluster around other nodes. Instead of a position, it requires a list of nodes to fit. Apart from the functions and port anchors as seen in the previous commands, this command provides the edges of the outer shape cluster as names. These names can be used further in computing intersection points. It can be customized with the following keys (check 3.1.3 for composite options):

`class=[sy|de|ct|sdf|blackbox|none]` The class of the node. This option affects the field color and the constructor label. Default is `none`.
`hasmoc` flag for saying that this node is a process, therefore it is associated with a MoC. It need to be provided otherwise the global flags will ignore MoC-related options (e.g. `nomoccolor`).
`shape=` any `TikZ` style option for the node shape. Check section 3.1.2 for a list of library-provided shapes. Default is `rectangle`.
`type=` the type / constructor of that particular ForSyDe block. It shows below the main label and it is affected by the `class` argument.

`npw=[0..10]` number of ports on the left side of the node, i.e. the number of left port anchors.
`ni=[0..10]` alias for `npw`, stands for "number of inputs".
`npe=[0..10]` number of ports on the right side of the node, i.e. the number of right port anchors.
`no=[0..10]` alias for `npe`, stands for "number of outputs".
`nf=[0..4]` the number of passed functions. Default is `0`.
`f1=` first function label. Appears as a box in the upper part in case `nf > 0`. Default is f_1.
`f2=` second function label. Appears as a box in the upper part in case `nf > 1`. Default is f_2.
`f3=` third function label. Appears as a box in the upper part in case `nf > 2`. Default is f_3.
`f4=` fourth function label. Appears as a box in the upper part in case `nf > 3`. Default is f_4.
`anchor=[anchor]` center point for shape. Default is `center`.
`xshift=[distance]` shift position in X direction. Default is `0pt`.
`yshift=[distance]` shift position in Y direction. Default is `0pt`.
`rotate=[number]` the rotation angle (in degrees) of the node. Default is `0`.
`rotate shape=[number]` the rotation angle (in degrees) of the `shape`. Defailt is `0`.
`inner xsep=[distance]` the inner separation in X direction. Default is `3pt`.
`inner ysep=[distance]` the inner separation in Y direction. Default is `3pt`.



```
\node[ports e8w8, inner ysep=18pt] (inside) at (2,0) {};
\cluster[shape=trans shape v1v1,inner sep=2pt]<(inside)>{};
\drawconduit{inside}{8}{8};
\foreach \i in {1,3,...,8} {\draw[very thin, ->] (inside.w\i) -- (inside.e\i);}
```

Figure 6: The `\drawconduit` draw command.

`\drawconduit[length]{node with ports}{num. ports west}{num. ports east};`
This command draws short lines outside a node with port anchors, that might be interpreted as "conduits" or "ports". It is used as helper for drawing, for example, custom-shaped transition patterns, like in the figure above.

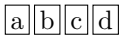
3.1.2 Shapes

`node shapes`

ForSyDe-TikZ provides a collection of raw shapes[2] either as TikZ styles or as low-level PGF drawings. They are often used as arguments for drawing commands. Below is a list with them and their usage:

Shape	Drawing	Description
`atom shape`		meant to be used with the `\basic` command, it is a good candidate for depicting ForSyDe atoms or other primitive blocks.
`nary atom shape`		similar to `atom shape`, but suggests an n-tuple of blocks.

[2] All shapes are defined in the `forsyde.shapes` TikZ library, which is loaded by default by the `forsyde-tikz` package, but can also be loaded independently using the command `\usetikzlibrary{forsyde.shapes}` in the document preamble.

`leaf shape`		meant to be used with the `\standard` command, it is a good candidate for depicting abstractions of complex semantics (e.g. black boxes).
`nary leaf shape`		similar to `leaf shape`, but suggests an n-tuple of blocks.
`comp shape`		meant to be used with the `\cluster` command, it is a good candidate for depicting exposed hierarchical blocks such as composite processes.
`nary comp shape`		similar to `leaf shape`, but suggests an n-tuple of blocks.
`ports e[0..10]w[0..10]`		these are multiple shapes that have no drawing, but define additional anchors[3] for ports on the east (`.e[index]`) and west (`.w[index]`) sides of the node. It is meant to fit around an existing shape.
`func[0..4]`		these shapes create a series of boxes with text and for each box two anchors: north (`.n[index]`), south (`.s[index]`). They are used to show functions as input arguments.
`dp shape`		meant to be used with the `\cluster` command, this shape suggests a data parallel skeleton.
`pipe shape`		meant to be used with the `\cluster` command, this shape suggests a pipeline skeleton.
`merge shape`		meant to be used with the `\cluster` command, this shape suggests a reduce/recur skeleton.
`generic skel shape`		meant to be used with the `\cluster` command, this shape suggests generic skeleton which implies a recursive composition of functions/blocks.
`trans shape v[1..8]v[1..8]`		meant to be used with the `\trans` command, this shape suggests a transition in the structure of `n` input vectors resulting in `n'` output vectors.
`trans shape s1v1`		meant to be used with the `\trans` command, this shape suggests a transposition of a vector to/from a stream.
`trans shape v1gv1`		meant to be used with the `\trans` command, this shape suggests a grouping/merging to/from a vector of vectors.

3.1.3 Helper options

node helper keys

The `forsyde-tikz` library provides a large set of styles or function keys[4] that can be used for user convenience as options in the draw commands shown in section 3.1.1.

Key	Description/Expands to
`moc=x`	`hasmoc, class=x`
`inner sep=x`	`inner xsep=x, inner ysep=x`
`left of=x`	positions left of node `x`, with an edge-to-edge distance = 1cm
`right of=x`	positions right of node `x`, with an edge-to-edge distance = 1cm
`above of=x`	positions above of node `x`, with an edge-to-edge distance = 1cm

[3] apart from the ones inherited from `rectangle`

[4] The helpers are defined in the `forsyde-tikz` package itself, separated from the TikZ libraries.

`below of=x`	positions below of node `x`, with an edge-to-edge distance = 1cm
`f={a;b;c;d}`	parses a string of functions separated by `;` and sets `nf`, `f1`, `f2`, `f3` and `f4` accordingly
`primitive`	`shape=atom shape`
`primitiven`	`shape=nary atom shape`
`process`	`shape=leaf shape, hasmoc`
`processn`	`shape=nary leaf shape, hasmoc`
`composite`	`shape=comp shape`
`compositen`	`shape=comp shape`
`embed`	`shape=leaf shape, hasmoc, inner sep=15pt`
`farmstyle`	`shape=dp shape, inner xsep = 15pt, inner ysep = 20pt,`
`pipestyle`	`shape=pipe shape, inner xsep = 15pt, inner ysep = 20pt,`
`skeleton`	`shape=generic skel shape, inner xsep = 15pt, inner ysep = 20pt,`
`transition=v2v3`	`shape=trans shape v2v3`. Default argument is `v1v1`.
`zipx`	`transition=s1v1, rotate shape=180, type=zipx,`
`unzipx`	`transition=s1v1, type=unzipx,`

3.2 Paths

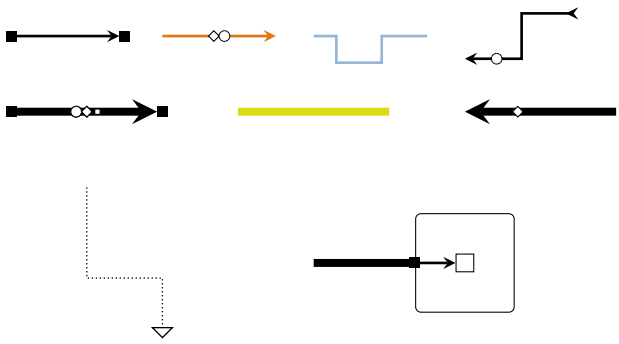
Data flow in `forsyde-tikz` is represented as (directed) edges between components. These are regular TikZ paths[5], customized with a series of keys provided by the library. There are obvious advantages for using the TikZ infrastructure, such as being able to further customize paths outside the scope of this library, or reuse existing keys (e.g. the arrow tips, shapes). The custom port or function anchors presented in the previous section come in handy when considering port-to-port connections.

`path keys`

The library provides the following custom keys:

- `s=[MoC]` draws a signal. If the global options allow it, passing a MoC string to this key will color the signal accordingly.
- `sn=[MoC]` draws an n-tuple of signals. If the global options allow it, passing a MoC string to this key will color the signal accordingly.
- `v=[MoC]` draws a vector. If the global options allow it, passing a MoC string to this key will color the vector accordingly, suggesting that it contains signals of their respective MoC.
- `vn=[MoC]` draws an n-tuple of vectors. If the global options allow it, passing a MoC string to this key will color the vector accordingly, suggesting that it contains signals of their respective MoC.
- `f` draws an edge used for showing how functions and parameters are passed from higher order functions downwards into the hierarchy.
- `fn` similar to `f`, but for n-tuples of functions.
- `srcport` draws a port symbol at the source of the path.
- `dstport` draws a port symbol at the destination of the path.
- `token=[token-string]` draws symbols for depicting the data structures carried by ForSyDe signals or vectors. The accepted tokens strings are `scalar|vector|function`. To draw tuple structures you have to separate token keywords by `-` (dash). E.g.: 3-tuple of scalar, vector and function can be drawn with `token=scalar-vector-function`. To depict functions

[5] All paths are defined in the `forsyde.paths` TikZ library, which is loaded by default by the `forsyde-tikz` package, but can also be loaded independently using the command `\usetikzlibrary{forsyde.paths}` in the document preamble.



```
\node[draw] (a) at (6, -1) {};
\cluster[embed] (cl) <(a)> {};
\path (0,2)   edge[s, srcport,->,dstport]  (1.5,2)
      (2,2)   edge[s=ct,-> , token=vector-scalar]  (3.5,2)
      (4,2)   edge[s=sy, -|-|-=0.2:0.6, deviate=-10pt] (5.5,2)
      (6,1.7) edge[s, <-<, token=scalar, token pos=0.2, -|-] (7.5,2.3)
      (0,1)   edge[v,token=scalar-vector-function,srcport,->,dstport] (2,1)
      (3,1)   edge[v=sdf] (5,1)
      (6,1)   edge[v, <-, token=vector,token pos=0.35] (8,1)
      (1,0)   edge[f, ->, |-|=0.6] (2,-2)
      (4,-1)  edge[trans={v,dstport}{cl-west}{s,->}] (a);
```

Figure 7: Different paths created with the library provided styles

which take other data types as arguments, you can group some tokens between parentheses. E.g. a 2-tuple of a scalar and a function which takes a vector and a scalar as arguments can be drawn with `token=scalar-(vector-scalar)`.

`token pos=[0.0 .. 1.0]` the position between the start node/anchor and the end node/anchor of the token symbols' center position. Default is `0.5`.

`intersect=[path-name]` finds the first intersection with the path named `[path-name]`, and creates a coordinate from it, named by default `int` if no other name was given with the `as` key.

`as=[coordinate-name]` names the intersection coordinate found with the `intersect`.

`trans={source-path-style}{intersection-path-name}{destination-path-style}` is a helper key which finds the intersection of the current path with `intersection-path-name`, splits the current path into two, decorated with `source-path-style` and `destination-path-style` respectively.

`-|-=[0.0 .. 1.0]` will create a horizontal-vertical-horizontal spline in the path. It may be accompanied by a number which determines the position of the 90 degree angle.

`|-|=[0.0 .. 1.0]` will create a vertical-horizontal-vertical spline in the path. It may be accompanied by a number which determines the position of the 90 degree angle.

`-|-=[0.0 .. 1.0]` will create a horizontal-vertical-horizontal spline in the path. It may be accompanied by a number which determines the position of the 90 degree angle.

`-|-|=[0.0 .. 1.0]` will create a horizontal-vertical-horizontal-vertical spline in the path. It may be accompanied by a number which determines the position of the 90 degree angle.

`-|-|-=[0.0 .. 1.0]:[0.0 .. 1.0]` will create a horizontal-vertical-horizontal-vertical-horizontal spline in the path. It may be accompanied by two numbers separated by `:` which determine the position of the 90 degree angles.

`|-|-|=[0.0 .. 1.0]:[0.0 .. 1.0]` will create a vertical-horizontal-vertical-horizontal-

vertical spline in the path. It may be accompanied by two numbers separated by `:` which determine the position of the 90 degree angles.

`deviate=` is a length representing the deviation from the straight path in case of complex splines (`-|-|`,`-|-|-` and `|-|-|`).

3.2.1 Helper commands

The following commands can be alternatively used to draw paths, in case the complexity of the `\path` TikZ command is not needed. The MoC can be passed to the `moc` style key.

```
\signal[style] (from) arrow-tip (to);
\signaln[style] (from) arrow-tip (to);
\vector[style] (from) arrow-tip (to);
\vectorn[style] (from) arrow-tip (to);
\function[style] (from) arrow-tip (to);
\functionn[style] (from) arrow-tip (to);
```

3.3 Utility commands

Following is a list of miscellaneous functions and environments provided for user convenience:

`\textleftof{node}{text}`

`\textrightof{node}{text}`

`\textaboveof{node}{text}`

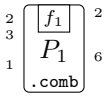
`\textbelowof{node}{text}`

`\ifnodedefined{node}{true expr}{false expr}`
Conditional sub-drawing if a node has been defined or not. Useful for ignoring edge cases when drawing recursive structures with `\foreach`.

`\gettikzx{node}{\xmacro}`
Gets the `X` coordinate of `node` and stores it in `\xmacro`.

`\gettikzy{node}{\ymacro}`
Gets the `Y` coordinate of `node` and stores it in `\ymacro`.

`\gettikzxy{node}{\xmacro}{\ymacro}`
Gets the `X` and `Y` coordinates of `node` and stores them in `\xmacro` and `\ymacro` respectively.




```
\standard[shape=leaf shape, ni=3, no=2, type=comb, nf=1] (n) <0,0> {$P_1$};
\resetportinfo{n}\wpinfo[east]{2}\wpinfo{3}\wpinfo{1}\epinfo{2}\epinfo{6}
```

Figure 8: An example of using the `\resetportinfo`, `\epinfo` and `\wpinfo` commands

`\resetportinfo{node}`

Resets the counters for the `\epinfo` and `\wpinfo` commands which decorates ports with information. These commands can be used for decorating ports, for example, with production and consumption rates in case of SDF processes.

`\wpinfo[anchor]{label}`
Places a label node next to a western port and increases its counter.

`\epinfo[anchor]{label}`
Places a label node next to a eastern port and increases its counter.

3.4 Further customizing the environment

The following commands can be used in the document preamble for changing environment variables.

```
% Colors
\renewcommand{\defaultdrawcolor}{[color]}
\renewcommand{\defaultfillcolor}{[color]}
\definecolor{sycolor}{[coord sys]}{[color coord]}
\definecolor{ctcolor}{[coord sys]}{[color coord]}
\definecolor{decolor}{[coord sys]}{[color coord]}
\definecolor{sdfcolor}{[coord sys]}{[color coord]}
\definecolor{blackboxcolor}{[coord sys]}{[color coord]}}
% line widths
\renewcommand{\compositelinewidth}{[size]}
\renewcommand{\skeletonlinewidth}{[size]}
\renewcommand{\signalpathlinewidth}{[size]}
\renewcommand{\functionpathlinewidth}{[size]}
\renewcommand{\vectorpathlinewidth}{[size]}
% sizes, etc.
\renewcommand{\tokensize}{[size]}
\renewcommand{\halftokensize}{[size]}
```

The ForSyDe system which performs the the Fast Fourier Transform can be defined in terms of atoms as:

$$\texttt{fft}_S\ k\ vs = \texttt{bitrev}_S((stage \diamond kern) \diamond vs) \tag{1}$$

where the constructors

$$stage\ wdt = \texttt{concat}_S \circ (segment \diamond twiddles) \circ \texttt{group}_S\ wdt \tag{2}$$
$$segment\ t = \texttt{unduals}_S \circ (butterfly\ t \diamond) \circ \texttt{duals}_S \tag{3}$$
$$butterfly\ w = ((\lambda\ x_0\ x_1 \rightarrow x_0 + wx_1, x_0 - wx_1) \triangle) \oplus \tag{4}$$

are aided by the number generators

$$kern = \texttt{iterate}_S\ (\times 2)\ 2 \tag{5}$$
$$twiddles = (\texttt{reverse}_S \circ \texttt{bitrev}_S \circ \texttt{take}_S\ (\texttt{lgth}_S\ vs/2))(wgen \diamond \langle 1..\rangle) \tag{6}$$
$$wgen\ x = -\frac{2\pi(x-1)}{\texttt{lgth}_S\ vs} \tag{7}$$

```
\documentclass[preview]{standalone}
\usepackage[math]{forsyde}

\begin{document}
The \ForSyDe system which performs the the Fast Fourier Transform can
be defined in terms of atoms as:

\begin{align}
  \SkelCons{fft}\ k\ vs =&\ \SkelCons{bitrev} ((\id{stage} \SkelFun \id{kern}) \SkelPip vs)
  \intertext{where the constructors}
  \id{stage}\ wdt   =&\ \SkelCons{concat} \circ (segment \SkelFun \id{twiddles})
                       \circ \SkelCons{group}\ wdt  \\
  \id{segment}\ t   =&\ \SkelCons{unduals} \circ (\id{butterfly}\ t\ \SkelFrm)
                       \circ \SkelCons{duals}  \\
  \id{butterfly}\ w =&\ ((\lambda\ x_0\ x_1 \rightarrow x_0 + wx_1, x_0 - wx_1 )\ \BhDef)\
                       \MocCmb \\
  \intertext{are aided by the number generators}
  \id{kern}         =&\ \SkelCons{iterate}\ (\times 2)\ 2 \\
  \id{twiddles}     =&\ (\SkelCons{reverse} \circ \SkelCons{bitrev} \circ \SkelCons{take}\
                       (\SkelCons{lgth}\ vs/2)) (\id{wgen} \SkelFun \SkelVec{1..}) \\
  \id{wgen}\ x      =&\ -\frac{2 \pi (x-1)}{\SkelCons{lgth}\ vs}
\end{align}
\end{document}
```

4 The `forsyde-math` package

This package provides a set of symbols and commands for writing equations mainly for the ForSyDe–Atom theoretical framework.

4.1 Operator symbols

The `forsyde-math` package exports a set of symbols written in the METAFONT language (see 4.3). These symbols are typed in using commands following the naming convention:

```
\<name>              % inside a math environment
\text<name>          % inside a text environment
```

operator symbols

where the symbol name is from the table below[6]:

<name>	5pt	7pt	8pt	9pt	10pt	12pt	14.4pt	17.28pt	20.74pt	24.88pt
BhFun										
BhApp										
BhDef										
BhPhi										
BhDeg										
MocFun										
MocApp										
MocCmb										
MocPre										
MocPhi										
MocDel										
SkelFrm										
SkelPip										
SkelFun										
SkelApp										
SkelRed										
SkelRec										

[6]the symbol naming scheme reflects their semantics defined in the ForSyDe–Atom framework, as two acronyms: first one denoting the layer and the second one denoting the constructor

4.2 Math commands

math commands

There are a couple of macros defined for math environments, mainly for convenience. They are listed below:

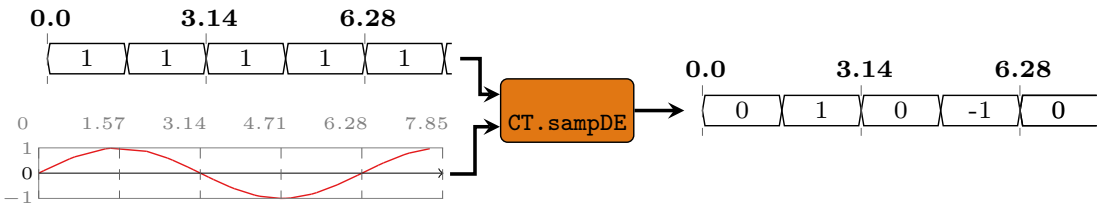
Command	Example	Result	Explanation
`\id{name}`	`$\id{func}$`	*func*	wraps `name` as an identifier, rather than loose characters
`\context{ctx}{f}`	`$\context{\Gamma}{f}$`	$\Gamma \vdash f$	associates context `ctx` to function `f`
`\Constructor{name}{layer}`	`$\Constructor{all}{T}$`	$\texttt{all}_T$	generic infix name for constructor in a user-defined layer
`\BhCons{name}`	`$\BhCons{default}$`	$\texttt{default}_B$	infix name for constructor in the *behavior layer*
`\MocCons{name}`	`$\MocCons{mealy}$`	$\texttt{mealy}_M$	infix name for constructor in the *MoC layer*
`\SkelCons{name}`	`$\SkelCons{mesh}$`	$\texttt{mesh}_S$	infix name for constructor in the *skeleton layer*
`\SkelVec{exp}`	`$\SkelVec{v}$`	$\langle v \rangle$	surrounds `exp` in vector type delimiters, associated with the *skeleton layer*

4.3 Font map

The ForSyDe–Atom operators from 4.1 have been created using METAFONT and have been bundled as a font family called `forsydeatom`. These fonts can be imported in accordance to the LaTeX 2_ε standard. The math symbol font based on this font family is called `atomoperators` and it declares all symbols as binary operators.

In case you need to access the fonts directly (and not through the `forsyde-math` package), here is the mapping of the `forsydeatom` font family:

	´0	*´1*	*´2*	*´3*	*´4*	*´5*	*´6*	*´7*	
´02x	⟑	△	⟑	⟑	⚠				″1x
´03x									
´04x	⨁	⊙	⊛	⊘	⊜	⧋			″2x
´05x									
´06x	⟐	⟐	⟐	⟐	◇	◇			″3x
´07x									
	″8	″9	″A	″B	″C	″D	″E	″F	




```
\documentclass{standalone}
\usepackage[plot,tikz]{forsyde}
\usepackage{filecontents}

\begin{filecontents}{ct-sampde-i1.flx}
  0.0                 : 0              ,  9.983480705722622e-2 : 0.10000000149,
  0.5646425989611361  : 0.60000000894  ,  0.6442172934010967   : 0.70000001043,
  0.9635580777669631  : 1.30000001937  ,  0.9854495936745004   : 1.40000002086,
  0.8632093171439696  : 2.10000003129  ,  0.5984721912401382   : 2.50000003725,
  0.5155014142351443  : 2.60000003874  ,  0.2392493170657283   : 2.90000004321,
  0.1411199729841522  : 3.0000000447   , -0.15774590023230295  : 3.30000004917,
 -0.2555412776230858  : 3.40000005066  , -0.5298363260375504   : 3.70000005513,
 -0.6118581429834538  : 3.80000005662  , -0.8715756491042981   : 4.20000006258,
 -0.9161661569653531  : 4.30000006407  , -0.9999232696597815   : 4.70000007003,
 -0.9961645277800696  : 4.80000007152  , -0.9824524964336662   : 4.90000007301,
 -0.8322681468629263  : 5.30000007897  , -0.7727641178059984   : 5.40000008046,
 -0.46460085726814676 : 5.80000008642  , -8.308743802765294e-2 : 6.20000009238,
  0.31154304920391934 : 6.60000009834  ,  0.40484843505764895  : 6.70000009983,
  0.7289691091159304  : 7.10000010579  ,  0.7936684574683253   : 7.20000010728,
  0.9679197162034261  : 7.60000011324  ,  0.9989413219493093   : 7.90000011771
\end{filecontents}
\begin{filecontents}{ct-sampde-o1.flx}
  0.0 : 0,
  1.0 : 1.570796326794,
  1.793238462856701e-12 : 3.141592653588,
 -1.0 : 4.712388980382,
  0.0 : 6.283185307176,
  1.0 : 7.85398163397,
 -3.0403029981061924e-7 : 8
\end{filecontents}

\begin{document}
\begin{tikzpicture}[]
  \standard[process, ni=2, no=1, moc=ct, type=sampDE](p1){};
  \begin{signalsCT}[name=ct-in, timestamps=1.57, grid=1.57, at={p1.west}, anchor=north east,
      xshift=-.5cm, yshift=-.2cm, xscale=1]{7.86}
    \signalCT*[outline,ordinate=0,ymin=-1,ymax=1]{ct-sampde-i1.flx}
  \end{signalsCT}
  \begin{signalsDE}[name=de-in, timestamps=3.14, grid=3.14, at={p1.west}, anchor=south east,
      xshift=-.4cm, yshift=.2cm]{8}
    \signalDE[trunc,last label=false]{ 1.0 : 0, 1.0 : 1.570, 1.0 : 3.141, 1.0 : 4.712, 1.0 :
        6.283, 1.0 : 7.853, 1.0 : 8 }
  \end{signalsDE}
  \begin{signalsDE}[name=de-out, timestamps=3.14, grid=3.14, outputs={p1.east}]{7.8}
    \signalDE*[trunc]{ct-sampde-o1.flx}
  \end{signalsDE}
  \path[s,-|-,->] (de-in.e1) edge (p1.w1) (ct-in.e1) edge (p1.w2) (p1.e1) edge (de-out.w1);
\end{tikzpicture}
\end{document}
```

5 The `forsyde-plot` package

This package provides environments and plot commands for visualizing the contents of ForSyDe signals. It is meant to be an alternative to GNUplot or other plotting tools, but not necessarily a replacement for these.

5.1 The `signalsSY` environment

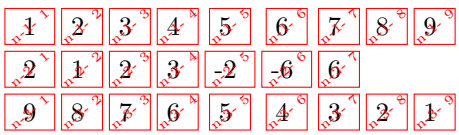
This environment creates a TikZ matrix of events, where each row is the content of a signal, and "synchronous" events are aligned on columns. This environment is suitable for depicting signals where the time/causality between events is implicit from their position in the signal. This includes synchronous signals, but also any dataflow signals.

```
\begin{signalsSY}[options]

  body...
\end{signalsSY}
```

The `body` of the environment usually consists in a series of `\signalSY` commands, but it may also contain any command acceptable within a TikZ `matrix` environment. Notice that it needs an initial line break. The `options` are:

`name=` the name of the TikZ element. Default is `sigplot`.
`at=` the position of the plot within the `tikzpicture`. Default is `(0,0)`.
`xshift=`*dist*, `yshift=`*dist*, `shift=`*coord* a displacement in the respective direction.
`anchor=`*anchor* where this element should be anchored.
`left of=`*coord*, `right of=`*coord*, `above of=`*coord*, `below of=`*coord* positions the element accordingly.
`inputs=`*coord*, `outputs=`*coord* positions the element left of/right of the given coordinate.




```
\begin{signalsSY}[name=n]

  \signalSY{1,2,3,4,5,6,7,8,9}
  \signalSY{2,1,2,3,-2,-6,6}
  \signalSY{9,8,7,6,5,4,3,2,1}
\end{signalsSY}
```

Figure 9: The `\signalSY` draw command. The red boxes are nodes accessed with `[name]-[row]-[column]`.

`\signalSY{events}`

The SY signal is simply a row in a matrix, and the elements can be accessed according to the TikZ matrix of nodes.

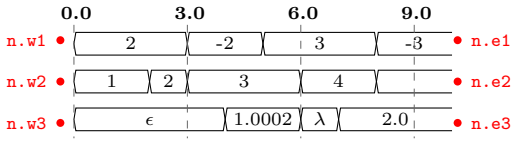
5.2 The signalsDE environment

This environment creates a (simple) plot of DE signals similar to Modelsim or GTKwave, within a `tikzpicture`.

```
\begin{signalsDE}[options]{xmax}
  body...
\end{signalsDE}
```

The body within the environment consists in a series of `\signalDE` commands. It *requires* an `xmax` number which stands for the last timestamp plotted. The `options` are:

`name=` the name of the TikZ element. Default is `sigplot`.
`grid=`*step* draws a dashed line every `step` time(stamps).
`timestamp=`*step* shows the time(stamp) above the plot at every `step`.
`grid and time=`*step* is equivalent to `grid=`*step* and `timestamp=`*step*.
`label pos=` position of the label within the plotted bar. Default is `center`.
`signal sep=`*dist* the vertical distance between two consecutive signals.
`xscale=`*ratio* and `yscale=`*ratio* scales the plot.
`at=` the position of the plot within the `tikzpicture`. Default is `(0,0)`.
`xshift=`*dist*, `yshift=`*dist*, `shift=`*coord* a displacement in the respective direction.
`anchor=`*anchor* where this element should be anchored.
`left of=`*coord*, `right of=`*coord*, `above of=`*coord*, `below of=`*coord* positions the element accordingly.
`inputs=`*coord*, `outputs=`*coord* positions the element left of/right of the given coordinate.




```
\begin{filecontents}{de-input1.flx}
 1:0, 2:2, 3:3, 4:6, 5:8, 5:15
\end{filecontents}
% ...
\begin{signalsDE}[name=n, grid and time=3]{10}
  \signalDE [trunc]{2.0:0, -2.0:3, 3.0002:5, -3:8, 5:15}
  \signalDE*[last label=false]{de-input1.flx}
  \signalDE {$\epsilon$:0, 1.0002:4, $\lambda$:6, 2.0:7, 5:15}
\end{signalsDE}
```

Figure 10: The `\signalDE` draw command. The red dots are custom anchors accessed with `[name]-[anchor]`.

`\signalDE*[options]{events_or_datafile}`

The DE signal command has two versions: a starred (`*`) version taking as argument the path to a file containing the input data, or the non-starred version which takes as argument the actual data (events). In both cases the data needs to be formatted as

```
value1 : timestamp1, value2 : timestamp2, ... , valueN : timestampN
```

where `timestampN` > `xmax`. The `options` are:

`name=` the name of the signal.
`trunc` a flag activating value truncation to integers. Works only of all values of a signal are numbers. Default is `false`.
`last label` a flag enabling the last value in a signal to be printed or not. Default is `true`.

5.3 The `signalsCT` environment

This environment creates a (simple) stacked plot of CT signals within a `tikzpicture`.

```
\begin{signalsCT}[options]{xmax}
  body...
\end{signalsCT}
```

The body within the environment consists in a series of `\signalDE` commands. It *requires* an `xmax` number which stands for the last timestamp plotted. The `options` are:

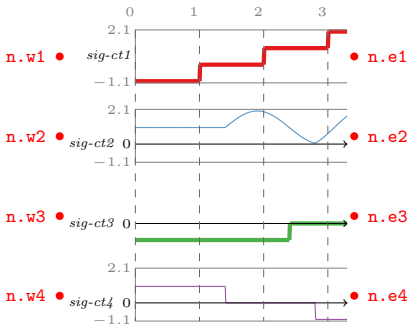
`name=` the name of the TikZ element. Default is `sigplot`.
`grid=`*step* draws a dashed line every `step` time(stamps).
`timestamp=`*step* shows the time(stamp) above the plot at every `step`.
`grid and time=`*step* is equivalent to `grid=`*step* and `timestamp=`*step*.
`label pos=` position of the label within the plotted bar. Default is `center`.
`signal sep=`*dist* the vertical distance between two consecutive signals.
`xscale=`*ratio*, `yscale=`*ratio* scales the plot.
`at=` the position of the plot within the `tikzpicture`. Default is `(0,0)`.
`xshift=`*dist*, `yshift=`*dist*, `shift=`*coord* a displacement in the respective direction.
`anchor=`*anchor* where this element should be anchored.
`left of=`*coord*, `right of=`*coord*, `above of=`*coord*, `below of=`*coord* positions the element accordingly.
`inputs=`*coord*, `outputs=`*coord* positions the element left of/right of the given coordinate.

`\signalCT*[options]{events_or_datafile}`
The CT signal command has two versions: a starred (`*`) version taking as argument the path to a file containing the input data, or the non-starred version which takes as argument the actual data (events). In both cases the data needs to be formatted as

```
value1 : timestamp1, value2 : timestamp2, ... , valueN : timestampN
```

where `timestampN` > `xmax`. The `options` are:

`name=` the name of the signal.
`ordinate=` draws the X axis (ordinate) on the value passed as argument.
`outline` a flag for drawing the outline for a signal. Default is `false`.
`ymax=`*num*, `ymin=`*num* the maximum and minimum values covered by the plot. Default is `ymin=0` and `ymax=1`.
`line style=` a style passed to the drawing tool.




```
\begin{tikzpicture}[]
  \begin{signalsCT}[name=n, grid and time=1, xscale=.7, yscale=.7]{3.3}
    \signalCT [name=sig-ct1, outline, ymin=-1.1, ymax=2.1, line style={ultra thick}]{
      -1.0:0, -1.0:0.9999999776, 0.0:1.009999977376, 0.0:1.9999999552, 1.0:2.009999954976,
      1.0:2.9999999328, 2.0:3.009999932576, 2.0:3.29999992608
    }
    \signalCT*[name=sig-ct2, outline, ymin=-1.1, ymax=2.1,draw ordinate]{ct-input1.flx}
    \signalCT [name=sig-ct3, ymin=-1.1, ymax=2.1,draw ordinate, line style={ultra thick}]{
      -1.0:0,-1.0:2.39999994624,0.0:2.409999946016,0.0:3.29999992608
    }
    \signalCT[name= sig-ct4, outline, ymin=-1.1, ymax=2.1,draw ordinate]{
      1.0:0,1.0:0.009999999776, 1.0:1.39999996864, 0.0:1.409999968416, 0.0:2.79999993728,
      -1.0:2.809999937056, -1.0:3.29999992608
    }
  \end{signalsCT}
```

Figure 11: The `\signalCT` draw command. The red dots are custom anchors accessed with `[name]-[anchor]`.

6 Libraries of additional pictures

The ForSyDe–LaTeX package contains a set of additional libraries containing usual parameterized pictures used in related publications or documentation. Their sources are public and can be imported as regular TikZ libraries. To use them one has to pre-load some general package dependencies, which are taken care of by passing the `pictures` option when loading the `forsyde` package.

```
\usepackage[pictures]{forsyde}
...
\usetikzlibrary{library}
```

where `library` is named along the lines of `forsyde.pictures.(name)`.

The following libraries are not extensively documented but rather listed along with the main commands and an example of usage. For more information check their source code.

6.1 The `forsyde.pictures.layered` library

This library provides commands for depicting the layered structure of constructors, as introduced in the ForSyDe–Atom project.

`\forsydeAtomMakeLayers[options]{list_of_layers}`
Creates a a picture of incremental layers wrapping each other, based on the list of labels, formatted in the following way:

```
inner label 0: outer label 0, inner label 1: outer label 1, ...
```

Each layer `N`, where `N` $\in [0..n]$ has:

- a named referred to by other commands: `layerN`;
- defined coordinates/anchors: `layerN-center`, `layerN-left`, and `layerN-right`;
- defined paths: `layerN-leftpath` and `layerN-rightpath`.

The `options` are:

`width=` default is `3.8cm`;
`height=` default is `1.8cm`;
`length=` default is `1cm`;

`\forsydeAtomSignalArrow[options]{position}`
Creates a a picture of wide arrow representing signal of events. The `position` is where the arrow tip should point to. It has defined the following coordinates/anchors: `arrow-south`, `arrow-north`, `arrow-east`, `arrow-west`, `arrow-center`. The `options` are:

`left indent=` default is `3pt`;
`right indent=` default is `0pt`;
`label shift=` default is `-4ex`;
`layer radius=` default is `2cm`;

`\forsydeAtomSignalVector{interesction_path}`
Is used after a `\forsydeAtomSignalArrow` command to draw at which layer `interesction_path` does a signal is transformed into a structure/vector of signals.

`\forsydeAtomHighlightLayer[color]{layer_number}`
Highlights a layer by filling it with a background color. Default `color` is `red`.

Listing 1: Example usage of `forsyde.pictures.layered`

```
\forsydeAtomMakeLayers[]{%
  Function Layer:Function,%
  Behavior Layer:Behavior,%
  MoC Layer:Process,%
  Skeleton Layer:Process Network}
\node[] (value) at (layer0-center){%
  \usebox{\forsydeAtomValue}};
\node[] (ext-value) at ($(layer1-center)!.5!(layer2-center)$) {%
  \usebox{\forsydeAtomExtValue}};
\node[] (event) at ($(layer2-center)!.5!(layer3-center)$) {%
  \usebox{\forsydeAtomTagExtValue}};
\forsydeAtomSignalArrow[width=4.2cm]{layer3-center}
\node[inner sep=0] (token) at (arrow-center) {%
  \usebox{\forsydeAtomTagExtValue}};
\node[] at ($(token.east)!.5!(arrow-west)$) {%
  \usebox{\forsydeAtomTagExtValue}};
\node[] at ($(token.west)!.5!(arrow-east)$) {%
  \usebox{\forsydeAtomTagExtValue}};
\forsydeAtomSignalVector{layer4-rightpath}
```

The `forsyde.pictures.layered` library also holds a number of shapes defined as `boxes`. These boxes are:

- `\forsydeAtomValue`
- `\forsydeAtomExtValue`
- `\forsydeAtomTagValue`
- `\forsydeAtomTagExtValue`

7 The `forsyde-legacy` package

This package offers an API for the legacy commands defined in older versions of the ForSyDe–LaTeXutilities. This way, documents compiled with old commands can be compiled with the newer versions of their respective library.

7.1 `forsyde-tikz v0.3` or prior

Although from `v0.4` onward the draw commands have been heavily modified, the old commands could be mapped to the new API.

```
\primitive[keys]         {id}{pos}{label}
\primitiven[keys]        {id}{pos}{label}
\leafstd[keys]           {id}{pos}{label}
\leafcustom[keys]        {id}{pos}
\compositestd[keys]      {id}{clustered nodes}{label}
\compositebbox[keys]     {id}{pos}{label}
\patterncluster[keys]    {id}{clustered nodes}{label}
\patternnodestd[keys]    {id}{pos}
\patternnodecustom[keys] {id}{pos}
```

7.2 `forsyde-pc v0.3` or prior

This package is obsolete and used to hold helpers associated to some ForSyDe process constructors.

```
\delay    [moc=,f1=,inner sep=,reverse]        {id}{pos}{label}
\delayn   [moc=,f1=,f2=,inner sep=,reverse]    {id}{pos}{label}
\map      [moc=,f1=,inner sep=,reverse]        {id}{pos}{label}
\comb     [moc=,f1=,inner sep=,reverse]        {id}{pos}{label}
\combII   [moc=,f1=,inner sep=,reverse]        {id}{pos}{label}
\combIII  [moc=,f1=,inner sep=,reverse]        {id}{pos}{label}
\combIV   [moc=,f1=,inner sep=,reverse]        {id}{pos}{label}
\scanl    [moc=,f1=,f2=,inner sep=,reverse]    {id}{pos}{label}
\scanlII  [moc=,f1=,f2=,inner sep=,reverse]    {id}{pos}{label}
\scanlIII [moc=,f1=,f2=,inner sep=,reverse]    {id}{pos}{label}
\scanld   [moc=,f1=,f2=,f3=,inner sep=,reverse]{id}{pos}{label}
\scanldII [moc=,f1=,f2=,f3=,inner sep=,reverse]{id}{pos}{label}
\scanldIII[moc=,f1=,f2=,f3=,inner sep=,reverse]{id}{pos}{label}
\moore    [moc=,f1=,f2=,f3=,inner sep=,reverse]{id}{pos}{label}
\mooreII  [moc=,f1=,f2=,f3=,inner sep=,reverse]{id}{pos}{label}
\mooreIII [moc=,f1=,f2=,f3=,inner sep=,reverse]{id}{pos}{label}
\mealy    [moc=,f1=,f2=,f3=,inner sep=,reverse]{id}{pos}{label}
\mealyII  [moc=,f1=,f2=,f3=,inner sep=,reverse]{id}{pos}{label}
\mealyIII [moc=,f1=,f2=,f3=,inner sep=,reverse]{id}{pos}{label}
\source   [moc=,f1=,f2=,inner sep=,reverse]    {id}{pos}{label}
\filter   [moc=,f1=,f2=,inner sep=,reverse]    {id}{pos}{label}
\hold     [moc=,f1=,inner sep=,reverse]        {id}{pos}{label}
\fillS    [moc=,f1=,f2=,inner sep=,reverse]    {id}{pos}{label}

\zip     [moc=,reverse]{id}{pos}
\zipIII  [moc=,reverse]{id}{pos}
\zipIV   [moc=,reverse]{id}{pos}
\zipV    [moc=,reverse]{id}{pos}
\zipVI   [moc=,reverse]{id}{pos}
\unzip   [moc=,reverse]{id}{pos}
\unzipIII[moc=,reverse]{id}{pos}
\unzipIV [moc=,reverse]{id}{pos}
```

```
\unzipV  [moc=,reverse]{id}{pos}
\unzipVI [moc=,reverse]{id}{pos}

\domaininterface[moc=,reverse]          {id}{pos}
\mocinterface   [mocin=,mocout=,reverse]{id}{pos}

\composite[ni=,no=,inner xsep=,inner ysep=,reverse] {id}{included}{label}
\blackbox [ni=,no=,inner xsep=,inner ysep=,reverse] {id}{included}{label}

\farm     [ni=,no=,inner xsep=,inner ysep=,reverse]                {id}{included}{label}
\farmI    [ni=,no=,f1=,inner xsep=,inner ysep=,reverse]            {id}{included}{label}
\farmII   [ni=,no=,f1=,f2=,inner xsep=,inner ysep=,reverse]        {id}{included}{label}
\farmIII  [ni=,no=,f1=,f2=,f3=,inner xsep=,inner ysep=,reverse]    {id}{included}{label}
\farmIV   [ni=,no=,f1=,f2=,f3=,f4=,inner xsep=,inner ysep=,reverse]{id}{included}{label}
\pipe     [ni=,no=,inner xsep=,inner ysep=,reverse]                {id}{included}{label}
\pipeI    [ni=,no=,f1=,inner xsep=,inner ysep=,reverse]            {id}{included}{label}
\pipeII   [ni=,no=,f1=,f2=,inner xsep=,inner ysep=,reverse]        {id}{included}{label}
\pipeIII  [ni=,no=,f1=,f2=,f3=,inner xsep=,inner ysep=,reverse]    {id}{included}{label}
\pipeIV   [ni=,no=,f1=,f2=,f3=,f4=,inner xsep=,inner ysep=,reverse]{id}{included}{label}
\reduce   [ni=,no=,inner xsep=,inner ysep=,reverse]                {id}{included}{label}
\reduceI  [ni=,no=,f1=,inner xsep=,inner ysep=,reverse]            {id}{included}{label}
\reduceII [ni=,no=,f1=,f2=,inner xsep=,inner ysep=,reverse]        {id}{included}{label}
\reduceIII[ni=,no=,f1=,f2=,f3=,inner xsep=,inner ysep=,reverse]    {id}{included}{label}
\reduceIV [ni=,no=,f1=,f2=,f3=,f4=,inner xsep=,inner ysep=,reverse]{id}{included}{label}

\unzipx     [reverse]         {id}{position}
\zipx       [reverse]         {id}{position}
\unzipv     [reverse]         {id}{position}
\zipv       [reverse]         {id}{position}
\splitatv   [f1=,reverse]     {id}{position}
\catv       [reverse]         {id}{position}
\oddsv      [reverse]         {id}{position}
\evensv     [reverse]         {id}{position}
\reversev   [reverse]         {id}{position}
\groupv     [reverse]         {id}{position}
\concatv    [reverse]         {id}{position}
\filteridxv [f1=,reverse]     {id}{position}
\gatherv    [f1=,f2=,reverse] {id}{position}
\gatherAdpv [f1=,f2=,reverse] {id}{position}
\selectv    [reverse]         {id}{position}
\distributev[f1=,reverse]     {id}{position}
\filterv    [f1=,reverse]     {id}{position}
\getv       [f1=,reverse]     {id}{position}

\visualoddsv   [reverse]{id}{pos}
\visualevensv  [reverse]{id}{pos}
\visualreversev[reverse]{id}{pos}
\visualgroupv  [reverse]{id}{pos}
\visualconcatv [reverse]{id}{pos}
```