

# KlugHDL : a SpinalHDL diagram generator

## Défense de projet d'approfondissement

Sylvain Julmy

Institut Systèmes Industriels  
Master of Science HES-SO

6 février 2017

# Overview of contents

- 1 Context
- 2 Diagrams modelling
- 3 Parsing the AST
- 4 Viewing library
- 5 Diagram visualisation
- 6 Current working state
- 7 Further work

# Schedule

- 1 Context
- 2 Diagrams modelling
- 3 Parsing the AST
- 4 Viewing library
- 5 Diagram visualisation
- 6 Current working state
- 7 Further work

# VHDL

- Hardware description language
- Mostly use with Verilog for programming on the FPGA
- Old, verbose and tricky

# SpinalHDL

SpinalHDL, written as an internal DSL, is used to describe digital hardware and generate the corresponding source code in VHDL (or Verilog).

# VHDL vs SpinalHDL

```
import spinal.core._

class AND extends Component {
  val io = new Bundle {
    val a = in Bool
    val b = in Bool
    val c = out Bool
  }
  io.c := io.a & io.b
}
```

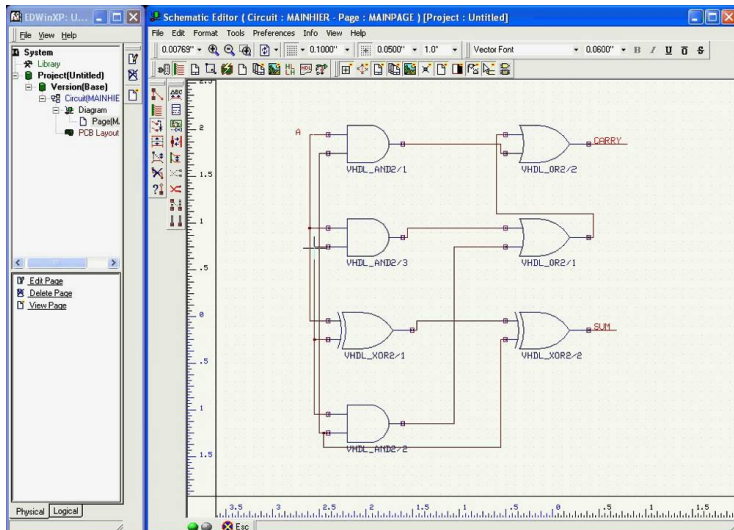
```
entity AND_1 is
  port(
    io_a : in std_logic;
    io_b : in std_logic;
    io_c : out std_logic
  );
end AND_1;

architecture arch of AND_1 is
begin
  io_c <= (io_a and io_b);
end arch;
```

# KlugHDL

An application which is able to analyse a SpinalHDL program and produce a block diagram of the corresponding hardware description.

## KlugHDL

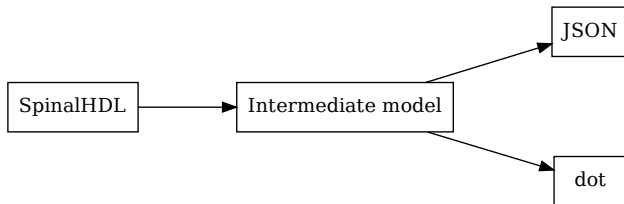




# Schedule

- 1 Context
- 2 Diagrams modelling**
- 3 Parsing the AST
- 4 Viewing library
- 5 Diagram visualisation
- 6 Current working state
- 7 Further work

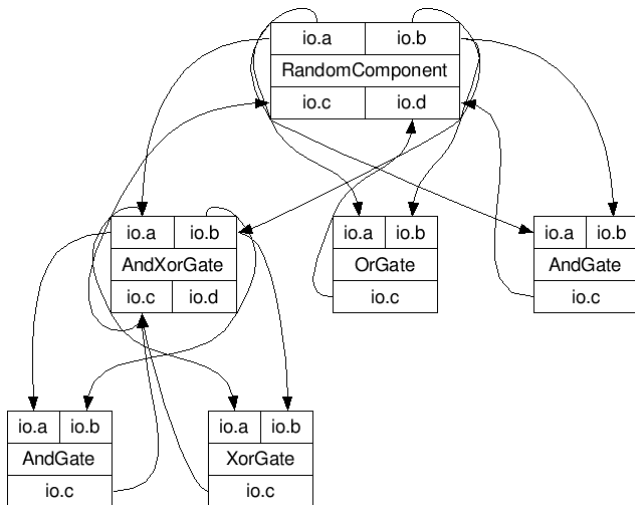
# Architecture



# Visual representation

- Hierarchical layout
- Tree view
- Multiple diagrams

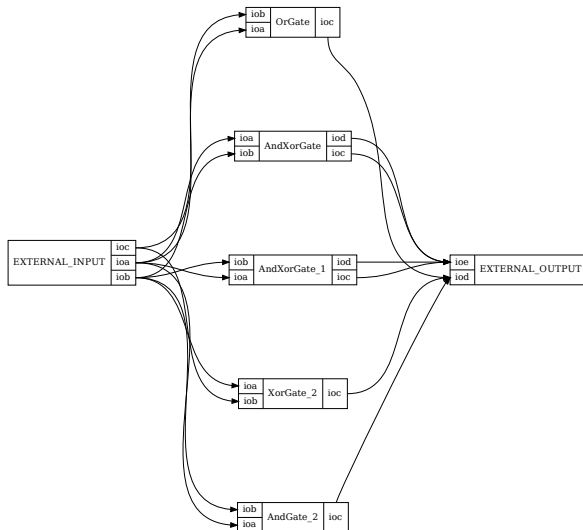
# Hierarchical layout



# Tree view

└ HierarchicComponent
AndGate_2
XorGate_2
└ AndXorGate
XorGate
AndGate
OrGate
└ AndXorGate_1
XorGate_1
AndGate_1

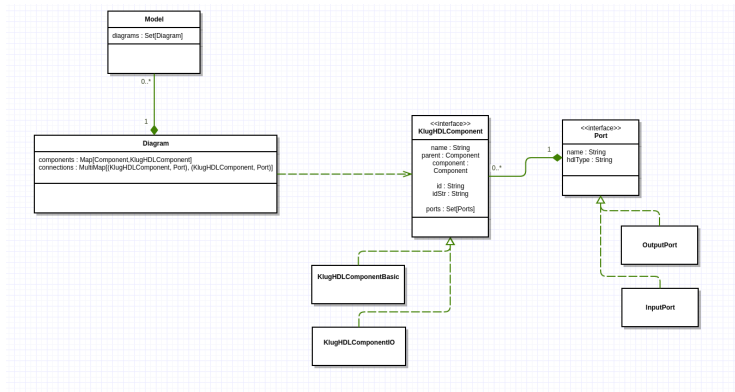
# Multiple diagrams



# Model representation

- Model owns at least one diagram
- Diagram owns one or more component
- Component owns some ports
- Connections are between ports

# Model representation





# Schedule

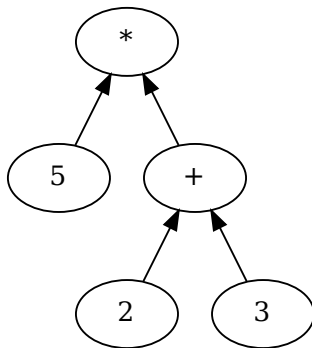
- 1 Context
- 2 Diagrams modelling
- 3 Parsing the AST**
- 4 Viewing library
- 5 Diagram visualisation
- 6 Current working state
- 7 Further work

# AST

AST : Abstract Syntax Tree

# AST

Example :  $5 * (2 + 3)$



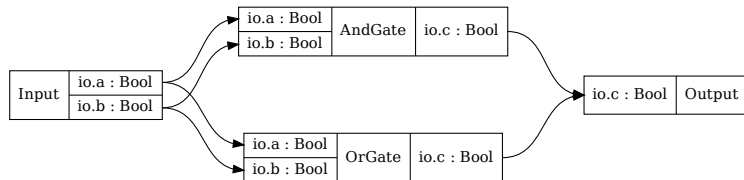
# Process

- Diagram parsing
- Component parsing and generation
- Ports parsing
- Connections parsing

# Schedule

- 1 Context
- 2 Diagrams modelling
- 3 Parsing the AST
- 4 Viewing library**
- 5 Diagram visualisation
- 6 Current working state
- 7 Further work

# Base graph model

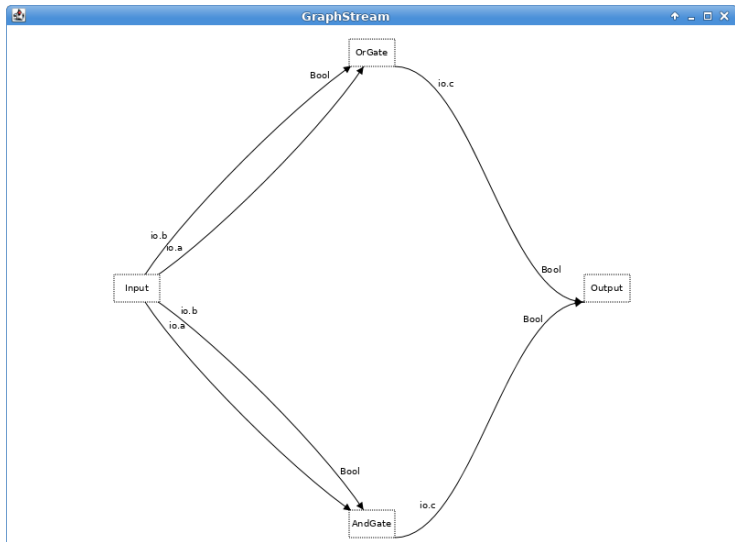


# GraphStream

GraphStream is a Java library used for the modelling and analysis of dynamic graphs.

# GraphStream

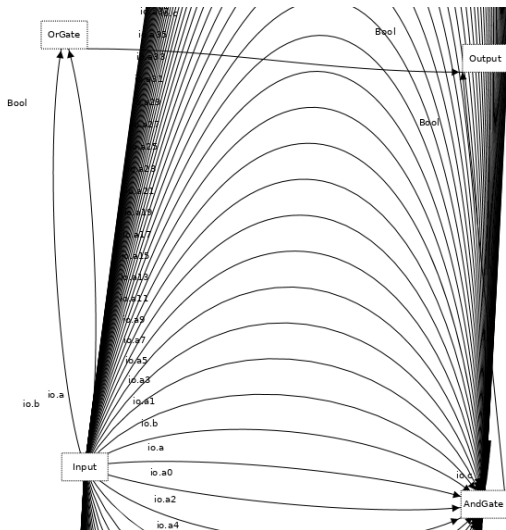
## Implementation of the base graph model





# GraphStream

## Problems



# GraphStream

## Problems

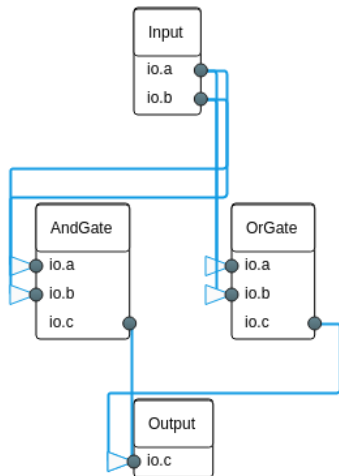
- No port notion
- Label on the edges
- Tricky connection's position

# Draw2D

Draw2D is a HTML5 and Javascript library for visualisation and interaction with diagrams and graphs.

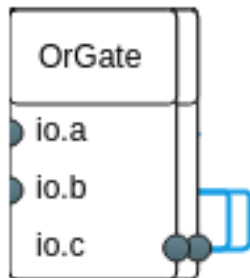
# Draw2D

## Implementation of the base graph model



# Draw2D

## Problems : Layout



# Draw2D

## Problems : Layout

### Solutions :

- Implements our own layout algorithm (for nodes with ports)
- Use the layout engine of another program (DOT,...)
- Use another library, which might be non-free

# Draw2D

## Problems : Layout

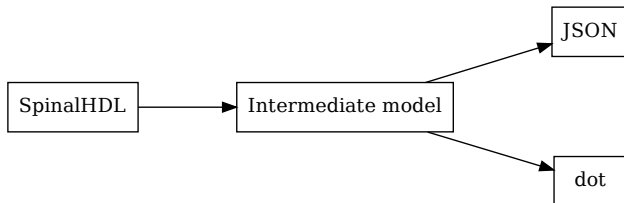
We can't (easily) manipulate the vertex layout with Draw2D.

# Schedule

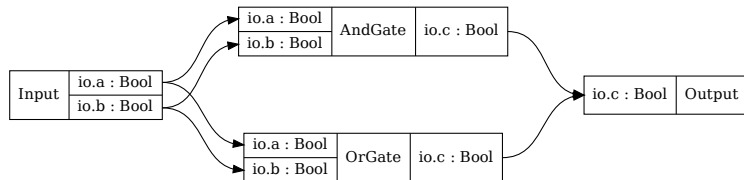
- 1 Context
- 2 Diagrams modelling
- 3 Parsing the AST
- 4 Viewing library
- 5 Diagram visualisation**
- 6 Current working state
- 7 Further work



# Recall



## DOT



## DOT

```

digraph g {
  node [shape=record];
  graph [rankdir=LR,ranksep="1",nodesep="1"];
  AndGate [label="{{<a>io.a : Bool|<b>io.b : Bool}|AndGate|{{<c>io.c :
    ↪ Bool}}}"];
  OrGate [label="{{<a>io.a : Bool|<b>io.b : Bool}|OrGate|{{<c>io.c :
    ↪ Bool}}}"];
  Input [label="{Input|{{<a>io.a : Bool|<b>io.b : Bool}}}"];
  Output [label="{{<c>io.c : Bool}|Output}"];
  Input:a -> AndGate:a;
  Input:b -> AndGate:b;
  Input:a -> OrGate:a;
  Input:b -> OrGate:b;
  OrGate:c -> Output:c;
  AndGate:c -> Output:c;
}

```

# JSON

JSON (JavaScript Object Notation) is a format mostly used by Javascript for object serialisation.

# JSON

```
{
  "tree":{
    "text":"SmallComponent",
    "nodes":[{
      "text":"AndGate"
    },{
      "text":"OrGate"
    }]
  },
  "model":[{
    "diagram":{
      "name":"null",
      "isTopLevel":"true",
      "components":[{
        "name":"SmallComponent",
        "type":"default",
        "ports":[{
          ...
          ...
        ]
      }
    ]
  }
]
```

# JSON

## Advantages

- Common backend for multiple libraries
- JSON is readable by a lot of languages
- Human readable

# Schedule

- 1 Context
- 2 Diagrams modelling
- 3 Parsing the AST
- 4 Viewing library
- 5 Diagram visualisation
- 6 Current working state**
- 7 Further work

# Current working state

## What is working

- Parse and generate the IR for simple SpinalHDL component.
- Generate DOT and JSON file.
- A dynamic visualisation through a browser.



# Current working state

## What isn't working

- Parse and generate the IR for more complex SpinalHDL component.
- Child to parent navigation
- Filter with the tree view

# Schedule

- 1 Context
- 2 Diagrams modelling
- 3 Parsing the AST
- 4 Viewing library
- 5 Diagram visualisation
- 6 Current working state
- 7 Further work**

# Further work

- Parsing and generating the intermediate model for any kind of SpinalHDL component
- Finding a way or improve the library to directly manipulate the Draw2D layout
- Generate the layout information with DOT and Graphviz and add them to KlugHDL

# Conclusion

- Try to offer a tool for SpinalHDL.
- Generate static and dynamic diagram from the SpinalHDL AST.
- Some works need to be done.

Questions ?

Thanks !