University of Applied Sciences Western Switzerland
MSE - Software Engineering

Deepening project

KlugHDL : a VHDL language generator

Author:
Julmy Sylvain

Supervisor:
Mudry Pierre-André

Fribourg, January 6, 2017

Contents

1 Introduction

A program is a sequence of mathematical instruction which are written by a human and executed by a computer. The VHDL langage has been created to describe digital and signals systems such as field-programmable gate arrays and integrated circuits[1].The problem is that VHDL is an old, verbose and tricky language.

SpinalHDL has been created to fight thos disadvantage. The main goal of a DSL (Domain Specific Langage) is to offer a more plaisant way to code and that's what dose SpinalHDL. Has written before, the code is written by a human and by using SpinalHDL, the human became happier than using just the VHDL.

KlugHDL is an another tools to make the human happiest, it's a component diagram generator which offer some

1.1 Context

SpinalHDL is a programming language to describe digital hardware and generate the corresponding in VHDL (or Verilog). SpinalHDL is written in Scala as a DSL and has multiple advantages[2] :

- No restriction to the genericity of your hardware description by using Scala constructs
- No more endless wiring. Create and connect complex buses like AXI in one line.
- Evolving capabilities. Create your own buses definition and abstraction layer.
- Reduce code size by a high factor, especially for wiring. Allowing you to have a better visibility, more productivity and fewer headaches.
- Free and user friendly IDE. Thanks to scala world for auto-completion, error highlight, navigation shortcut and many others
- Extract information from your digital design and then generate files that contain information about some latency and addresses
- Bidirectional translation between any data type and bits. Useful to load a complex data structure from a CPU interface.
- Check for you that there is no combinational loop / latch
- Check that there is no unintentional cross clock domain

The code 1 shows a AND gate written with SpinalHDL with the corresponding generated VHDL code, we could see the similarity between the two code.

1.2 Goal

The goal of the project is to produce an application which is analysing a spinalhdl program in order to produce a block diagramm of the corresponding hardware description. This application should offer the following activities :

```
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;
```

Listing 1: Example of a AND gate written in SpinalHDL and the corresponding generated VHDL code

- Display the complete graph of the programm
- Navigate through all the hierarchical level of the programm (Component inside another component)
- The edges should display some information about the signals :
  - type
  - input or output
  - ...
- A Hierarchical view (like in filesystems or in Quartus)
- A filter functionnality in order to view only some specific component

The application is going to be develop, firstable, in a standalone version and next could be develop into a plugin version for Eclipse or IntelliJ.

1.3 Document overview

TODO

2 Analysis

In this chapter, we would discuss about the "What do we want ?" questions. It means what need the final product based on the Goal of the project. The other part is about the actual state of SpinalHDL. We are not presenting all of the implementation or conception, just a small part which is usefull for this project. FInally, we introduce the concept of Component and AST (Abstract Syntax Tree).

2.1 What do we want ?

Based on the goal of the project at section 1.2, we expose what the end user of SpinalHDL want to do with KlugHDL. The figure 2.1 show the use case of KlugHDL.

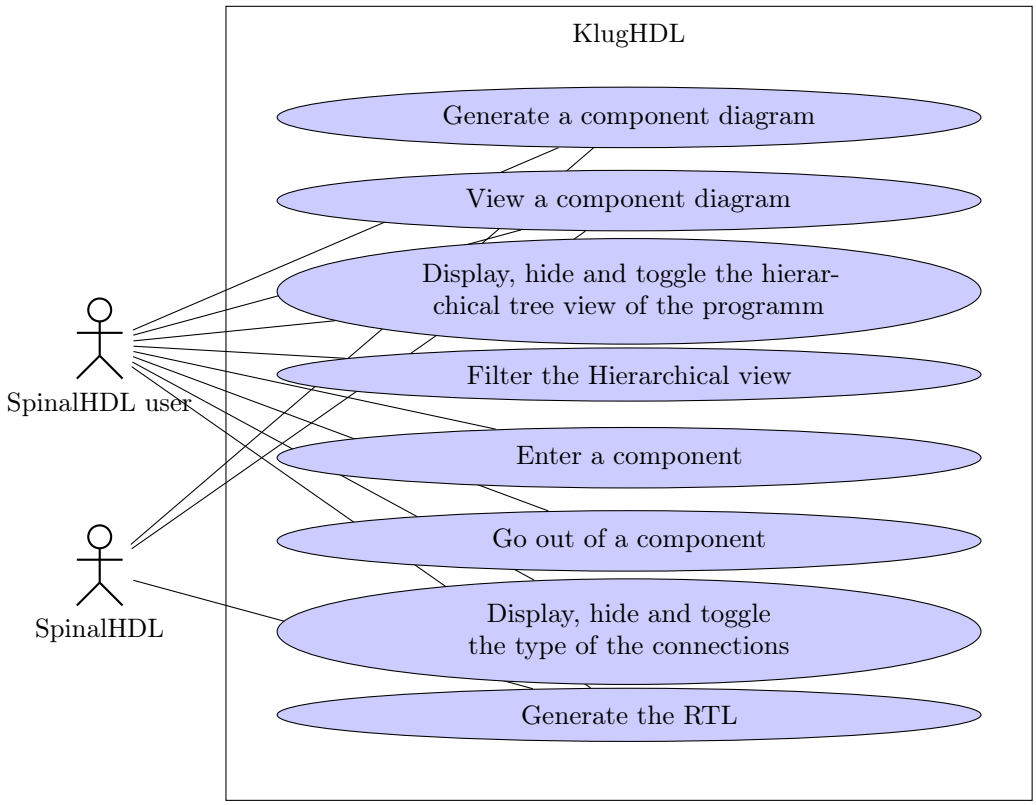

Figure 2.1: Use case of KlugHDL

An important point is the idea of port. If we want to design a HDL diagram, we have to display the types of the connection between the component. We have at least two alternative :

- display the types using the edges
- display the types using some ports attached to the component

We need to figure out which alternative is the best in order to display the type of a connection.

2.2 SpinalHDL

As explain in chapter 1, SpinalHDL is a DSL for VHDL programmation. SpinalHDL run with various internal construction such as an AST or a Component Tree. Those are the majors element needed for the diagram generation.

2.2.1 Component

A SpinalHDL's component is the base class of every SpinalHDL Program, like in the listing 1. Like in Verilog and VHDL, you can define components that could be used to build a design hierarchy. But unlike them, you don't need to bind them at instantiation [2]. That's why there is just one component against the architecture and entity combination in VHDL.

The component is the main part of the SpinalHDL architecture for this project because all the component are linked together with :

- a parent-children relationship
- a brother relationship

A component could contains and communicate with an other component, that's the parent-children relationship, and a component could communicate with the component contained by his parent, that's the brother relationship. This two relation could be seen in listing 2. The component is communication with their children which are communicationg between them.

A component could have other element inside him too, like Area or Function. But they are not relevant for the diagram generation.

2.2.2 Abstract Syntax Tree

An abstract syntax tree is a tree where the node are marked by operator and the leaf are marked by the operand of those operator. For example, the expression $5 * 2 + 3$ has his equivalent AST showed in the figure 2.2.

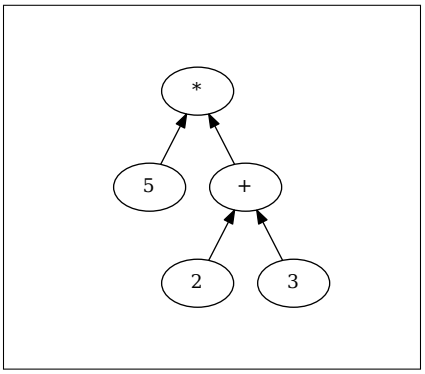

Figure 2.2: AST of the expression $5 * 2 + 3$

The AST is important because he is using to recover all the input and the output of a component, including the source of each input's component. To do that, we just need to recursively walk inside the AST of a SpinalHDL program.

```
class ParentChildrenBrother extends Component
{
    val andGate = new AndGate
    val xorGate = new XorGate
    val andXorGate1 = new AndXorGate
    val orGate = new OrGate
    val andXorGate2 = new AndXorGate

    val io = new Bundle
    {
        val a = in Bool
        val b = in Bool
        val c = in Bool
        val d = out Bool
        val e = out Bool
    }

    andGate.io.a := io.a            // children communication example
    andGate.io.b := io.b
    xorGate.io.a := io.b
    xorGate.io.b := io.c

    orGate.io.a := andGate.io.c     // brother communication example
    orGate.io.b := xorGate.io.c

    andXorGate1.io.a := io.a
    andXorGate1.io.b := io.b

    andXorGate2.io.a := io.a
    andXorGate2.io.b := io.b

    io.d := andGate.io.c & xorGate.io.c & orGate.io.c
    io.e := andXorGate1.io.c | andXorGate1.io.d | andXorGate2.io.c | andXorGate2.io.d
}
```

Listing 2: caption

3 Viewing library

This chapter will present some visualization library which could be use to generate and interract with the component diagram. The comparison between those library is based on the features we want to offer to the KlugHDL user (see chapter 1.2). In order to compare those viewing library, we end by discuting the point of the evaluation and the evaluation of all the library mentinnoed.

For the presentation of each of those library for graph visualization, we would use the same graph showed in figure 3.1.

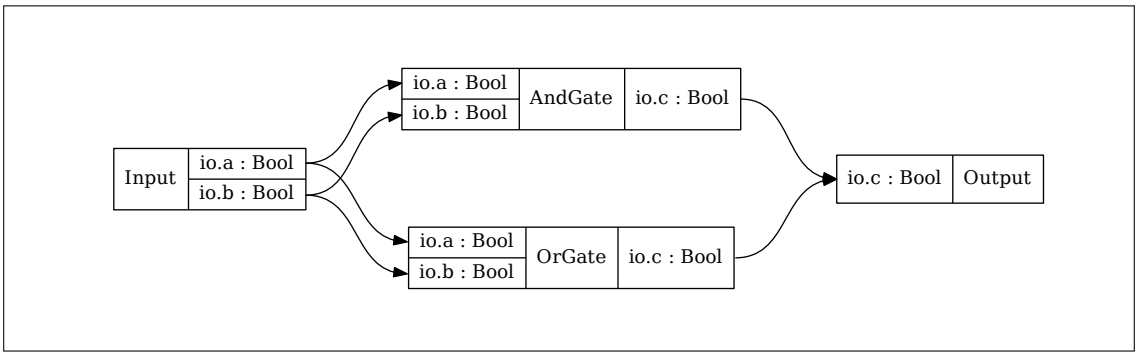

Figure 3.1: The graph we will use in order to compare several Viewing library. This model includes two logical component : a AND and a OR gate and two nodes which are representing the input and output of the parent component

3.1 GraphStream

GraphStream is a Java library for the modeling and analysis of dynamic graphs[3]. The goal of the library is to provide a way to represents graphs and work on it[3]. GraphStream is an active project hosted by the university of Le Havre in France.

3.1.1 Implementation of the base graph model

The figure 3.2 show the base graph model realized with the GraphStream library.

GraphStream allow us to simply create a graph (or a multigraph) in Java like in listing 3.

3.1.2 Remarks

With GraphStream, there is no way to control some visual features which are very interresing for hardware component visualisation : the splines of the edges and the anchor position on the node. There is no way to have a node with sub-node like in the base graph model in figure 3.1.

An another special case is the label on the edges. With GraphStream, we could add some label on the edges like in listings 4, but we can't add multiple ones. To add multiple label we need to use sprite like in listing 5.

```java
public class Example
{
   public static void main(String args[])
   {
       Graph graph = new SingleGraph("Example");
       graph.addNode("A" );
       graph.addNode("B" );
       graph.addNode("C" );
       graph.addEdge("AB", "A", "B");
       graph.addEdge("BC", "B", "C");
       graph.addEdge("CA", "C", "A");
   }
}
```

Listing 3: Modelisation of a simple graph using the GraphStream library

```java
package examples;

import org.graphstream.graph.Edge;
import org.graphstream.graph.Graph;
import org.graphstream.graph.implementations.MultiGraph;

/**
 * GraphStream example :
 * How to label an edge
 * Created by Snipy on 21.10.16.
 */
public class EdgesLabel
{
    public static void main(String[] args)
    {
        System.setProperty("org.graphstream.ui.renderer",
                           "org.graphstream.ui.j2dviewer.J2DGraphRenderer");

        Graph graph = new MultiGraph("Edges label example");
        graph.addAttribute("ui.quality");
        graph.addAttribute("ui.antialias");

        graph.addNode("A");
        graph.addNode("B");
        Edge edge = graph.addEdge("AB", "A", "B", true);
        edge.addAttribute("ui.label", "A --> B");
        graph.display(false);
    }
}
```

Listing 4: TODO : maybe remove ?

```java
package examples;

import org.graphstream.graph.Graph;
import org.graphstream.graph.implementations.MultiGraph;
import org.graphstream.ui.spriteManager.Sprite;
import org.graphstream.ui.spriteManager.SpriteManager;

/**
 * GraphStream example :
 * How to add multiples labels on an edge
 * Created by Snipy on 21.10.16.
 */
public class SpriteEdgesExample
{
    public static void main(String[] args)
    {
        System.setProperty("org.graphstream.ui.renderer",
                           "org.graphstream.ui.j2dviewer.J2DGraphRenderer");

        Graph graph = new MultiGraph("Edge multiple label example");
        graph.addAttribute("ui.quality");
        graph.addAttribute("ui.antialias");

        graph.addNode("A");
        graph.addNode("B");

        graph.addEdge("AB", "A", "B", true);

        /* Add some label using the sprite */

        SpriteManager sman = new SpriteManager(graph);

        // add label 1
        Sprite label1 = sman.addSprite("label1");
        label1.attachToEdge("AB");
        label1.setPosition(0.2, 0, 0);
        label1.addAttribute("ui.label", "Label 1");
        label1.addAttribute("ui.style", "fill-mode:none;");

        // add label 2
        Sprite label2 = sman.addSprite("label2");
        label2.attachToEdge("AB");
        label2.setPosition(0.8, 0, 0);
        label2.addAttribute("ui.label", "Label 2");
        label2.addAttribute("ui.style", "fill-mode:none;");

        graph.display(true);
    }
}
```

Listing 5: TODO : maybe remove ?

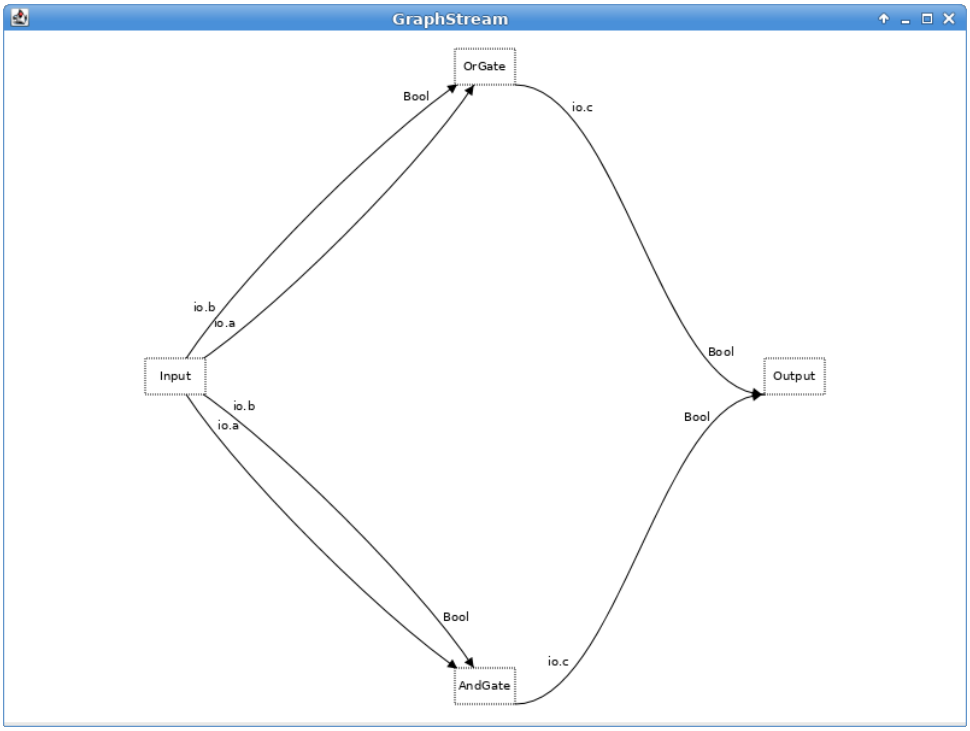

Figure 3.2: The base graph model rendering using graphstream

3.1.3 Conclusion

GraphStream ownes some another features which could be usefull to design an application :

- View integration using Swing
- Human-Computer interaction with the view
- A complete CSS interpretation to personalize the nodes and the edges

The main problem using GraphStream is we need to indicates the component information using label on the edges. The figure 3.2 show four nodes and six connections, but if we have a more lot of edges between nodes, it would be difficult to see the type of the inputs and outputs, like show in figure 3.3.

In order to overpass these problem, we need to visualize the type in an another form : in the graph 3.1 we have the idea to show the inputs and outputs as a port of the component. With GraphStream we can't do that, so we need to an another library which have these idea of port.

3.2 Draw2D

Draw2D is a HTML5 and Javascript library for visualization and interaction with diagrams and graph[4]. The goal of the library is to provide a way to represent diagrams and graph and manipulate those using the mouse or javascript code. Some existing product already use Draw2d for diagrams manipulation :

- Shape Designer[4]

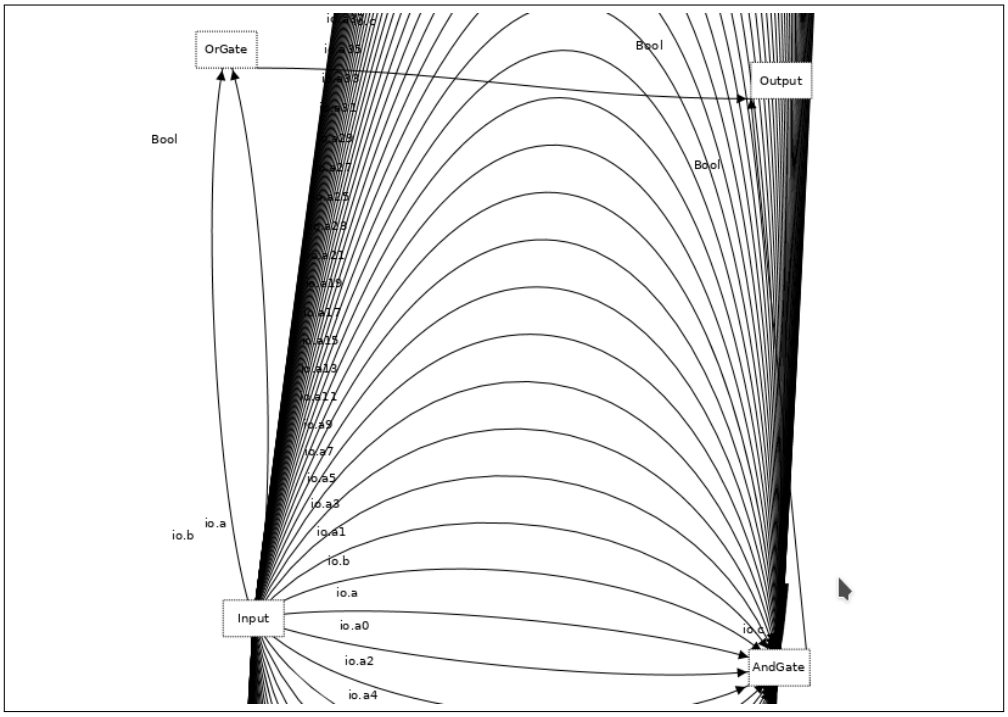

Figure 3.3

- BrainBox[4]
- Sankey State[4]

3.2.1 Implementation of the base graph model

Before implementing directly the basic example of the graph 3.1, we need to extend the library with a new component for our usage. In the chapter 3.1.3 we indicates that's the GraphStream library is lacking the idea of port, but Draw2d already have those.

The listing 6 show the realisation of the base graph model example. Note that the object **ComponentShape** and the functions **ComponentShape.addPort()**,**newConnection()** and **createConnection()** are not part of the Draw2d library, it's an extension added for this project. Those implementation are discuss in the section 6.1.

The code 6 is producing the HTML page see in figure 3.4.

3.3 Graph layout

One functionnality the Draw2D don't have is the layout of the diagram. If we said nothing about the position of the node, Draw2D simply overlap all the nodes and manage nothing for the visualization like in figure 3.5.

In order to produce a visualisable diagram, we need to layout ourself the diagram. We need to mention that's the layout algorithm is a NP-Hard problem in computing theory[5] and for our graph, we have some specialties :

```
var canvas = new draw2d.Canvas("gfx_holder1");

var andGate = new ComponentShape();
var orGate = new ComponentShape();
var input = new ComponentShape();
var output = new ComponentShape();

canvas.installEditPolicy(new draw2d.policy.connection.DragConnectionCreatePolicy({
    createConnection: createConnection
}));

andGate.setName("AndGate");
orGate.setName("OrGate");
input.setName("Input");
output.setName("Output");

andGate.addPort("io.a", "input");
andGate.addPort("io.b", "input");
andGate.addPort("io.c", "output");

orGate.addPort("io.a", "input");
orGate.addPort("io.b", "input");
orGate.addPort("io.c", "output");

input.addPort("io.a", "output");
input.addPort("io.b", "output");

output.addPort("io.c", "input");

canvas.add(input);
canvas.add(output);
canvas.add(andGate);
canvas.add(orGate);

canvas.add(newConnection(input.getPort("io.a"), andGate.getPort("io.a")));
canvas.add(newConnection(input.getPort("io.b"), andGate.getPort("io.b")));
canvas.add(newConnection(input.getPort("io.a"), orGate.getPort("io.a")));
canvas.add(newConnection(input.getPort("io.b"), orGate.getPort("io.b")));
canvas.add(newConnection(andGate.getPort("io.c"), output.getPort("io.c")));
canvas.add(newConnection(orGate.getPort("io.c"), output.getPort("io.c")));
```

Listing 6: The necessary code to produce the base graph model with the Draw2d library. The object **ComponentShape** and the functions **addPort()**, **newConnection()** and **createConnection()** are not part of the Draw2d library, it's an extension added for this project.

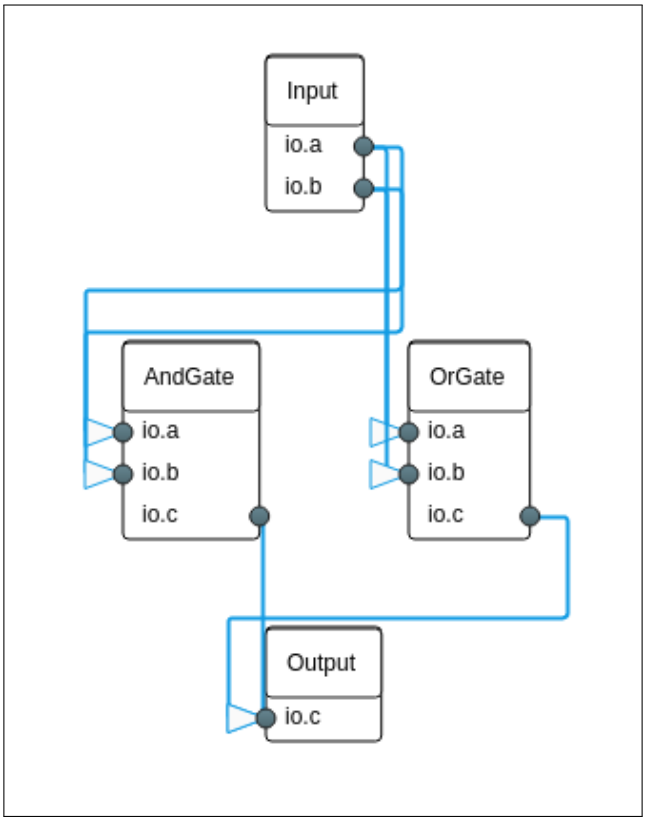

Figure 3.4: Rendering of the base graph model diagram from figure 3.1 using the Draw2D library

- The graph could be cyclic
- The graph is oriented
- Nodes ownes port, so we need to layout the edges on specific part of the graph

The first two specialties aren't a big deal, but the last one is quiet problematic, he involved a special layout algorithm which take care of the ports position on the nodes. Otherwise the visual result could be chaotics like in figure 3.6, in which we can't see the connection between some specific ports.

In order to solve this drawback, we need to layout the graph ourself by using a library. We could also do it ourself by implementing the algorithm for the kind of graph we have, but it's to time-consuming for a project like this.

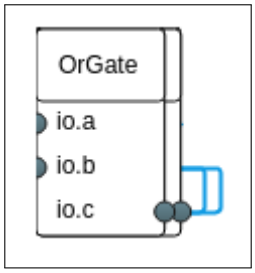

Figure 3.5: When we specify nothing about the node position with Draw2D, the engine just overlaps all the node

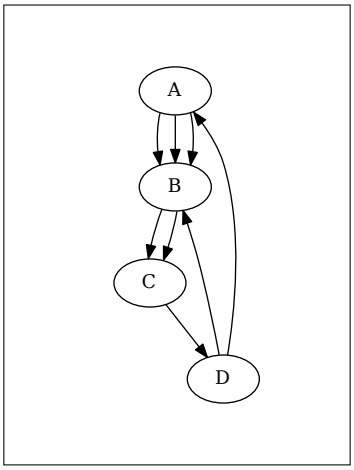

Figure 3.6: Visualization of a multi-graph without port, we could not determine the connection which is representing a certain connection between two component

TODO layout des edges du graph... (questions sur le forum de Draw2d.org)

4 Diagrams modelisation

In order to produce a visual schema from a SpinalHDL program, we need to parse his AST and produce an intermediate model for the diagrams. The reason to use an intermediate model is that we could then build multiple backend generator for multiple output target. The completeme pipeline of the generation is visible in figure **??**.

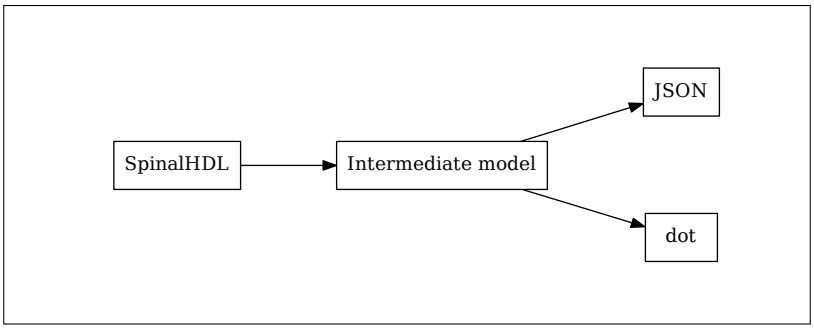

Figure 4.1: This figure show the entire generation pipeline use by KlugHDL in order to generate different output. The intermediate model is use to produce the target code, for example dot or JSON file

We also need to introduce an crucial element to understand how the model is build : the hierarchy visualisation.

4.1 Hierarchy visualisation

The diagrams produces by KlugHDL are kind of special, they are hierarchical ones. That's means we need to find a way to show the hierarchy between the components of a SpinalHDL program. There is multiple way to do this :

- Showing the elements using a hierarchical layout, like in family tree
- Showing the elements one by one (multiple diagrams)
- Showing the elements using a tree view, like in file explorer
- Showing the elements in 3D, the brothers at the same height and the parents higher

4.1.1 Hierarchical layout

The hierarchical layout is the simplest way to achieve the visualisation of the hierarchy. The figure 4.2 illustrate a simple hierarchical layout with some children and parent.

The problem by using such a representation is clear, sometimes the ports of a component is an output type port as well as a input type port. This can be seen at the port "io.c" from the **AndXorGate** component.

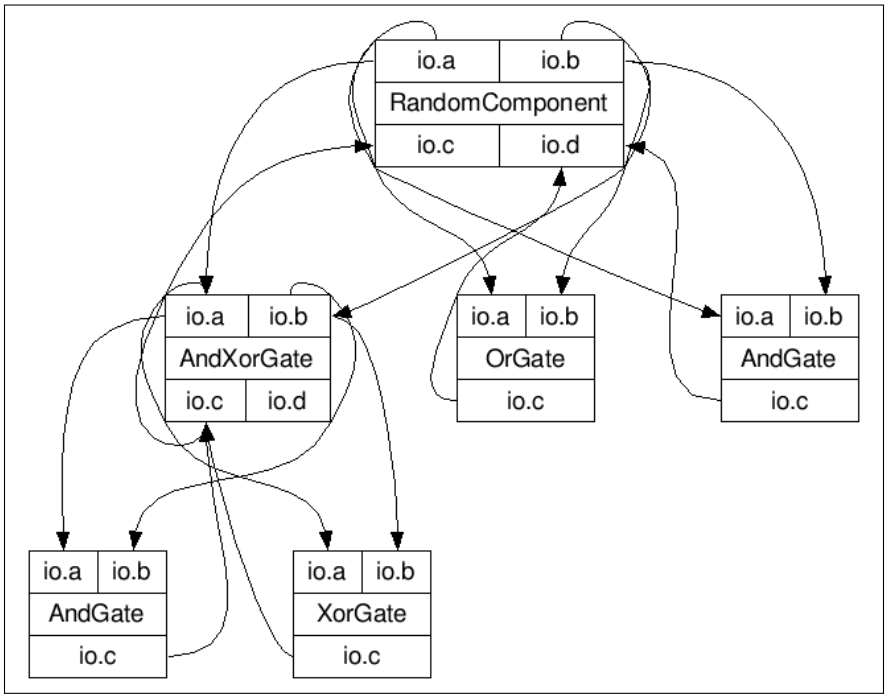

Figure 4.2: The simplest hierarchical visualization

4.2 A visual example

The diagram we want to produce, as explain in 3.3, ownes the following property :

- Oriented
- Cyclic
- Connection between port on nodes and not between nodes

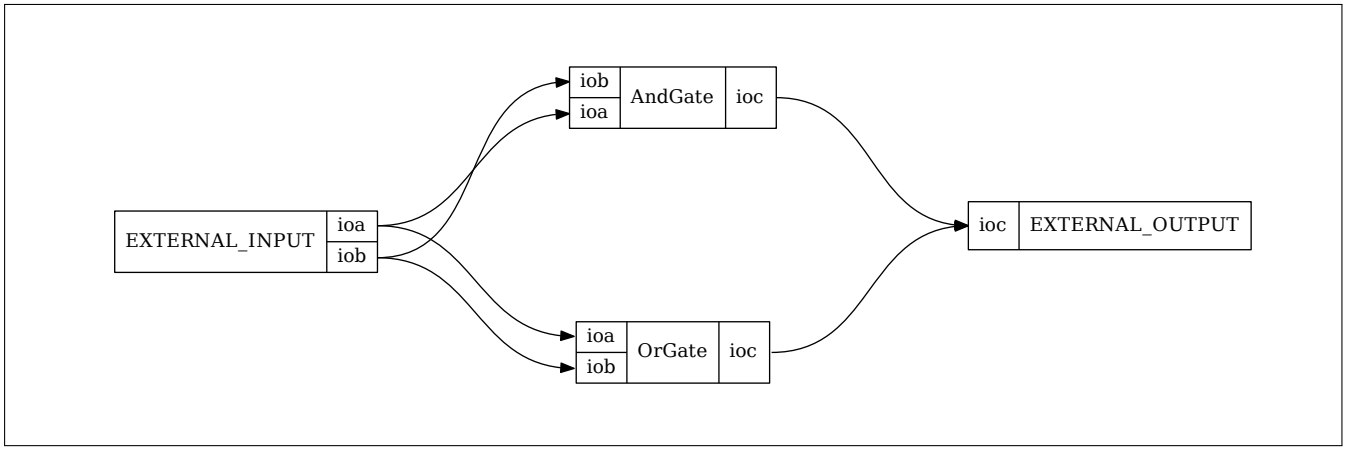

Figure 4.3: A complete example of a KlugHDL diagram, with connection between ports and oriented edges. The EXTERNAL_INPUT and EXTERNAL_OUTPUT are the interface of the component

4.3 Model representation

The model representation is builded using the oriented-object paradigm, the figure 4.4 show the class diagram of the models.

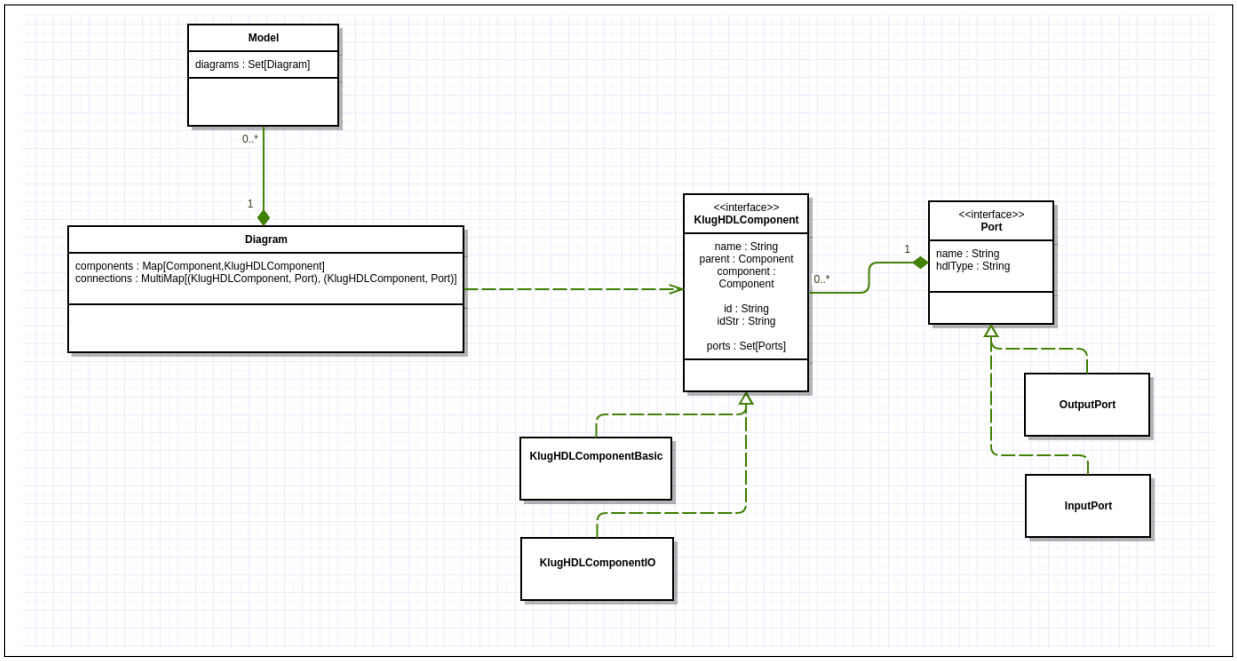

Figure 4.4: The complete class diagram of the intermediate model representation

4.4 Conclusion

The major advantages of such an intermediate representation is the capability to evolve the program in order to produce additionnal output. For now there is just a **dot** and **JSON** backend. It's also provide a way to check the correctness of the diagram after the parsing on the AST : connections with non-existing port for example.

5 Specification

TODO

5.1 Modeling the diagrams

6 Implementation

TODO

Parler du parcours du graphe de composant pour retrouver :

- type des liens
- directions des liens

6.1 Extension of the Draw2d library

TODO

7 Tests

TODO

8 Conclusion

TODO

Bibliography

[1] Wikipedia, VHDL, Online; accessed 05.10.2016, **2016**.

[2] C. Papon, SpinalHDL : About SpinalHDL, Online; accessed 05.10.2016, **2016**.

[3] T. G. Team, GraphStream : a Dynamic Graph Library, Online; accessed on 09.10.2016, **2015**.

[4] A. Herz, Draw2D touch, **2007**, `http://draw2d.org/` (visited on 03/25/2014).

[5] R. Tamassia, *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*, Chapman & Hall/CRC, **2007**.

Appendices

A Facts sheet : Generate a component diagram

Generate a component diagram

1 : Identification summary

Abstract : The user want to generate the component diagram from the code
Authors : Sylvain Julmy

Actors : SpinalHDL user
Creation date : 12.10.2016 **Version** : 0.1
Modification date : 18.10.2016 **Responsible** : Sylvain Julmy

2 : Sequence description

Precondition : The user activate the compilation options

Standard progress :

1. The user indicate to the compiler to launch KlugHDL
2. The user compile the program using the standard Scala compiler
3. A Windows appear and show the component diagram

Alternative sequence :
A1 : Start at point 2 of standard progress

1. There is a compilation error
2. Nothing append by KlugHDL
3. Restart at point 1 of standard progress

Postcondition : The component diagram is displayed

3 : HCI needing (Optional)

A windows that's show the component diagram

4 : Complementary remarks

KlugHDL is not really a standalone application, it's more like an option of SpinalHDL at the compilation.

B Facts sheet : Display, hide and toggle the view of the edge's type

Display, hide and toggle the view of the edges's type

1 : Identification summary

Abstract : The user want to see the types of some edges
Authors : Sylvain Julmy

Actors : SpinalHDL user
Creation date : 12.10.2016 **Version** : 0.1
Modification date : 14.10.2016 **Responsible** : Sylvain Julmy

2 : Sequence description

Precondition : The diagram is generated and open.

Standard progress :

1. The user click on "Show types"
2. The diagram display the types of the edges

Postcondition : The types of the selected edges are display.

3 : HCI needing (Optional)

The system needs the following HCI element :

- A toggable button "Show types"

4 : Complementary remarks

All the edges are affected by this operation

C Facts sheet : Enter a component

Enter a component

1 : Identification summary

Abstract : The user want to enter inside a component (zoom in) in order to show the sub-component

Authors : Sylvain Julmy

Actors : SpinalHDL user
Creation date : 12.10.2016 **Version** : 0.1
Modification date : 18.10.2016 **Responsible** : Sylvain Julmy

2 : Sequence description

Precondition : The diagram is generated and open.

Standard progress :

1. The User double click on a component
2. The system display the inside of the component

Postcondition : The double clicked component is maximize and is now the "root" component of the window.

3 : HCI needing (Optional)

- A visualization of the component and the sub-component

4 : Complementary remarks

Except the leaf component which has no sub-component and the BlackBox, every component are zoomable.

D Facts sheet : Exit a component

Exit a component

1 : Identification summary

Abstract : The user want to exit (zoom out) a component
Authors : Sylvain Julmy

Actors : SpinalHDL user
Creation date : 12.10.2016 **Version** : 0.1
Modification date : 18.10.2016 **Responsible** : Sylvain Julmy

2 : Sequence description

Precondition : The user has entering a component previously.

Standard progress :

1. The user zoom out (using the mouse wheel)
2. The parent component is displaying by the system

Exception sequence :
E1 : Start at point 1 of standard progress

1. The is no parent component for the current one
2. Nothing is changing
3. Case is ended

Postcondition : The user view the inside of the parent component.

3 : HCI needing (Optional)

- A visualization of the component and his parent

4 : Complementary remarks

Except the root component, each component could be zoom out.

E Facts sheet : View a component diagram

View the component diagram

1 : Identification summary

Abstract : The user want to display and view the component diagram of his program
Authors : Sylvain Julmy

Actors : SpinalHDL user
Creation date : 12.10.2016 **Version** : 0.1
Modification date : 21.10.2016 **Responsible** : Sylvain Julmy

2 : Sequence description

Precondition : The RTL has been generated without compilation error and the user has enter the correct option for the compilation.

Standard progress :

1. The user compile is program
2. The diagram is displayed by the system

Postcondition : The diagram is display.

3 : HCI needing (Optional)

- A diagram of the component of the user program

4 : Complementary remarks

The diagram could be generated but not necessarily display by the user.

F Facts sheet : Display, hide, toggle the hierarchical tree view

Display, Hide and Toggle the hierarchical tree view

1 : Identification summary

Abstract : The user want to hide or display the hierarchical tree view
Authors : Sylvain Julmy

Actors : SpinalHDL user
Creation date : 12.10.2016 **Version** : 0.1
Modification date : 21.10.2016 **Responsible** : Sylvain Julmy

2 : Sequence description

Precondition : The diagram is generated and open.

Standard progress :

1. The user click on the "Hierarchical tree view" button
2. The hierarchical tree view is display or hide

Postcondition : The hierarchical view is visible if she was previously hidden and is not visible if she was previously display.

3 : HCI needing

- A tree view of the componenent and their sub-component
- A toggable button "Hierarchical tree view"

4 : Complementary remarks

G Facts sheet : Generate the RTL

Generate the RTL

1 : Identification summary

Abstract : The user want to generate the RTL of his SpinalHDL program
Authors : Sylvain Julmy

Actors : SpinalHDL user
Creation date : 12.10.2016 **Version** : 0.1
Modification date : 21.10.2016 **Responsible** : Sylvain Julmy

2 : Sequence description

Precondition :

Standard progress :

1. The user compile his code using the command `sbt compile`
2. The RTL is generated

Exception sequence :
E1 : Start at point 1 of standard progress

1. There is a compilation error
2. Case is ended

Postcondition : The RTL is generated

3 : HCI needing (Optional)

There is no HCI needing because everything is done by SBT.

4 : Complementary remarks

Generate the RTL is not a job done by KlugHDL but by SpinalHDL.

H Facts sheet : Filter the view

Filter the view

1 : Identification summary

Abstract : The user want to filter the view (including the hierarchical tree view) to see a specific kind of component

Authors : Sylvain Julmy

Actors : SpinalHDL user
Creation date : 12.10.2016 **Version** : 0.1
Modification date : 21.10.2016 **Responsible** : Sylvain Julmy

2 : Sequence description

Precondition : The diagram is generated and open.

Standard progress :

1. The user enter the text specific to a component (class name)
2. The view turn gray the component who arent matching the text

Postcondition : The element who aren't matching the introduce pattern are grayscale.

3 : HCI needing (Optional)

- A writable textfield

4 : Complementary remarks