Center for Embedded and Cyber-Physical Systems
University of California, Irvine

A Tool for Visualization of SystemC Modules

Daniel Mendoza, Rainer Dömer

Technical Report CECS-17-06
November 6, 2017

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

dmmendo1@uci.edu

A Tool for Visualization of SystemC Modules

Daniel Mendoza, Rainer Dömer

Technical Report CECS-17-06
November 6, 2017

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

dmmendo1@uci.edu

Abstract

SystemC is a system level programming language that enables researchers to simulate large models consisting of hierarchies of modules, ports and channels. Although comprehending a hierarchy of modules, ports, and channels is possible from analysis of SystemC source code, this task is difficult and time-consuming. `Visual` *is a tool for visualization of SystemC module hierarchy that can make analyzing source code an easy and convenient task. This report discusses the development and usage of this tool that draws the hierarchy of modules, ports and channels of a given Systemc source file. In this report, we will first discuss how modules, ports and channels are visually represented. Then, we will explain in detail options this tool features. Finally, we will show some examples of SystemC models and their respective visual representation generated by the* `Visual` *tool.*

Contents

1 Introduction **1**

2 Visualization of SystemC Objects **1**
2.1 Modules . . . 2
2.2 Ports . . . 2
2.3 Channels . . . 3

3 Command Line Options and Extra Features **4**
3.1 Command Line Options . . . 4
3.2 Extra Features . . . 4
3.2.1 File Submenu . . . 5
3.2.2 View Submenu . . . 5

4 Experiments and Results **5**
4.1 Mandelbrot . . . 5
4.2 Port Mapping . . . 6
4.3 Play . . . 7
4.4 Noc2x2 . . . 8
4.5 JPEG . . . 9
4.6 Canny . . . 10

5 Conclusion and Future Work **11**

References **11**

A Tool for Visualization of SystemC Modules

Daniel Mendoza, Rainer Dömer
Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
dmmendo1@uci.edu

Abstract

SystemC is a system level programming language that enables researchers to simulate large models consisting of hierarchies of modules, ports and channels. Although comprehending a hierarchy of modules, ports, and channels is possible from analysis of SystemC source code, this task is difficult and time-consuming. `Visual` *is a tool for visualization of SystemC module hierarchy that can make analyzing source code an easy and convenient task. This report discusses the development and usage of this tool that draws the hierarchy of modules, ports and channels of a given Systemc source file. In this report, we will first discuss how modules, ports and channels are visually represented. Then, we will explain in detail options this tool features. Finally, we will show some examples of SystemC models and their respective visual representation generated by the* `Visual` *tool.*

1 Introduction

The development of `Visual` for SystemC Module Visualization is a undergraduate summer project in conjunction with the development of the Parallel SystemC Simulation on Many-Core Architectures project [3]. This tool supports a graphical user interface implemented with the Gtk API and visualizes a specified SystemC source file's Module hierarchy [2], which is drawn using the Cairo API. In order for the tool to receive information about the modules described in a SystemC source file, the tool utilizes the Recoding Infrastructure for SystemC (RISC) API [4], which is built on top of the ROSE compiler infrastructure [1]. The tool essentially gets lists of data about a module that has information about nested modules and thus can recursively iterate through nested lists of child modules to obtain enough information to visualize the hierarchy of the entire SystemC source file. The input SystemC source file may contains thousands of lines of code which can make manually drawing a representation of the modules, ports, and channels described by the code a difficult time-consuming task. Thus the `Visual` tool was made to address this issue and it can automatically generate a visual representation of a SystemC model in a very short period of time.

2 Visualization of SystemC Objects

The RISC API [4] provides data structures that contain information about a SystemC file. The RISC API analyzes a SystemC file and implements RISC objects that represent the SystemC data structures. The RISC objects contain information about the SystemC data structure it represents. For example, a RISC object can represent a SystemC module. The RISC API provides methods to obtain the type and name of the module the RISC object represents. If the module contains more modules within its contents, the RISC object that represents

outer module can provide a list of the inner modules. By recursively iterating through lists of RISC objects, we draw the hierarchy of SystemC modules from the bottom of the hierarchy to the top.

2.1 Modules

Modules are represented by boxes and the name of a certain module appears in the top left corner. Modules of the same class type are filled with the same color.

Drawing the module hierarchy is the main function of the `Visual` tool. Any information provided in the visual representation of a SystemC source file will be provided within a top module. Thus lists of modules are the data structure that the main recursive drawing function iterates through. The recursive function is provided with an outer module and obtains a list of inner modules. If the list of inner modules is empty, we have reached the base case condition of the recursive function and the module is drawn according to the height and width of its contents. If the list of inner modules is not empty, the function will call itself with the inner module as an argument and go deeper into the hierarchy of the SystemC modules. The inner recursive function call will also provide information about the width and height of the inner modules so that after the inner recursive function is finished, the outer module can be drawn according to the size of the contents it contains.

In Figure 1, the top module monitor has three child modules: `screen`, `speakerL`, and `speakerR`. When drawing this module hierarchy, the tool first iterates to the lowest level of the hierarchy and then draws the lowest level module it finds. Thus in Figure 1, the tool finds a list of three child base modules contained in the monitor module and begins to draw left to right. Thus the `screen` module is the first module drawn on the Cairo surface. The width of the base modules are drawn depending on how many characters are in the name of the module for efficiency of space within the drawing. The height of the base modules are fixed values. After the module `screen` is drawn, any sibling of module `screen` is drawn horizontally adjacent to it with a fixed offset value away from module `screen`'s rightmost edge. After iterating through the list of base modules, the recursive functions returns back up the hierarchy and passes information about the base modules' sizes. The module monitor is then drawn with a width that is equal to the sum of the width's of its child modules and the offsets between them with an added offset so that the module monitor is gaurenteed to be large enough to visually contain its child modules.

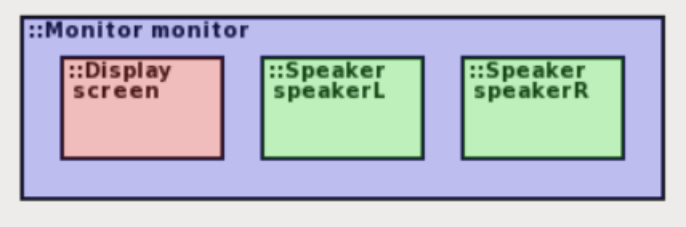


Figure 1: Module Drawing Example

2.2 Ports

Ports are visually represented by dots on the top line of a module box and the name of the module is provided right under the dot it represents. The ports of a module are drawn before the outer limits of a module so that we know the module box is large enough to contain all of the dots that represent the ports within the module. When

a port is drawn, information about the port's horizontal axis value, vertical axis value, and the pointer to the port are stored. This information is then passed up the hierarchy so that a line can be drawn between two ports to represent two ports that are bound.

In Figure 2, the outer module has a port called `AudioRight` that binds to a port called `AudioIn` which is located in the inner module `speakerR`.

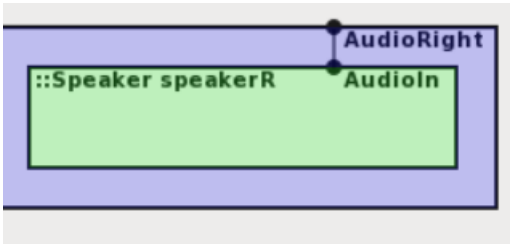


Figure 2: Port Drawing Example

2.3 Channels

Channels within a module are represented by their name followed by a horizontal black line at the top region of the module. Primitive channels are represented in the exact same way as hierarchical channels. When the recursive drawer function reaches a module with a channel, a vertical offset equal to the height of all the channels within the module is added so that the child modules are drawn below the channels. However, even though the offset is added before drawing the child modules, we cannot draw the channels and their connection between ports because we do not know where the ports of the child modules will appear. Thus the recursive function continues down the hierarchy, draws the child modules, passes up information about the child modules' ports and then draws the channels and their connections between ports.

In Figure 3 `VideoStream`, `AudioStreamL`, `AudioStreamR`, `Video`, `AudioLeft`, and `AudioRight` are all channels within the top module. The channel `VideoStream` is bound to a port `VideoStream`.

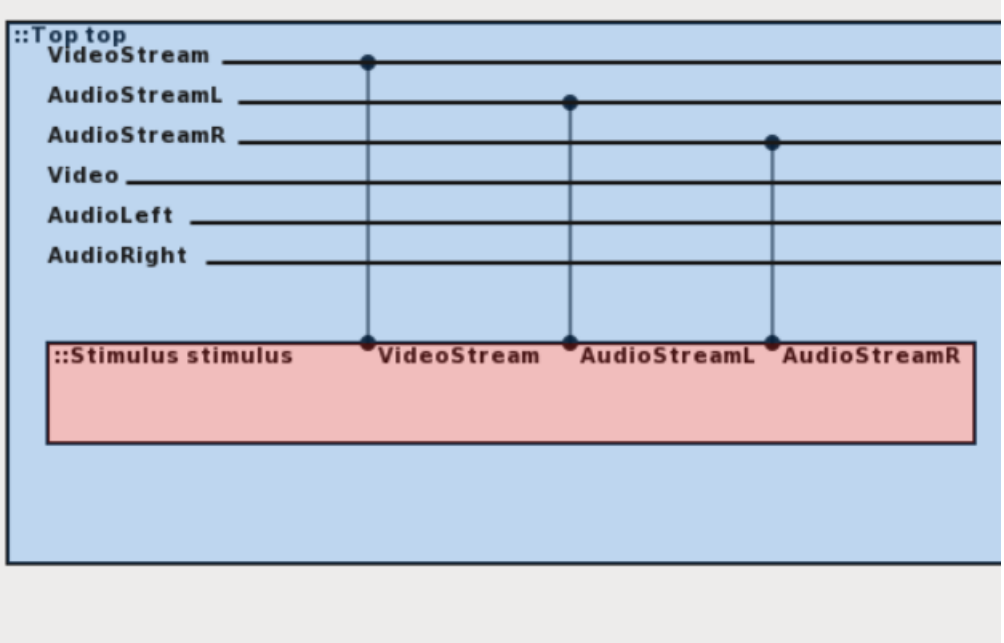


Figure 3: Channel Drawing Example

3 Command Line Options and Extra Features

The developed `Visual` tool includes command line options and extra features within the GUI for the user to customize the drawing of the SystemC module hierarchy. The elements options included in the command line can also be adjusted during runtime with the GUI.

3.1 Command Line Options

First, we will discuss how to use the different command line options and their effects.

-bw Modules are drawn without color.

-tm [module name] Only draw [module name].

-ll [integer] Draw only a certain depth in the hierarchy given by [integer].

-s [floating point number] Scale the drawing size by [floating point number]. if [floating point number] = 0.5, then the size of the drawing is scaled by 50 percent.

-np The module hierarchy will be drawn without ports or channels.

3.2 Extra Features

The options edited in the command line can also be managed with menus provided in the GUI.

3.2.1 File Submenu

The GUI includes a File submenu in the top left corner of the program window. It features two buttons `Print` and `Quit` with their usages listed below.

Print Exports the drawing as a pdf file called `visualization.pdf`.

Quit Exits the program.

3.2.2 View Submenu

The GUI includes a View submenu next to the File submenu. It contains options for editing the drawing of the modules. These options are listed below.

Zoom Scale the drawing size.

Depth Provides options to vary the number of levels drawn in the hierarchy.

No Color Toggles color or no color in the drawing.

Draw Channels and Ports Toggles the drawing of ports and channels.

4 Experiments and Results

In this section we show some examples of the `Visual` tool's outputs. We illustrate eight different examples. These examples are MandelBrot, Port Mapping, Play, Noc2x2, JPEG, Canny5, Canny6, and Canny7.

4.1 Mandelbrot

Figure 4 exemplifies the display of the tool's interpretation of a Mandelbrot renderer. The input SystemC source file consists of over a thousand lines of code, which can make visualizing the electronic model described by the source code tedious from reading only source code or impossible if the reader does not understand SystemC. From the display of the GUI, one can easily discern the module hierarchy of the Mandelbrot renderer without reading the source code.

Figure 4 illustrates the hierarchy of modules without drawing the ports and channels. The drawing of ports and channels can be swithed on in the `View` menu at the top left of the GUI. The top module contains three modules: `stimulus`, `platform`, and `monitor`. The platform then contains three modules: `din`, `dut`, and `dout`.

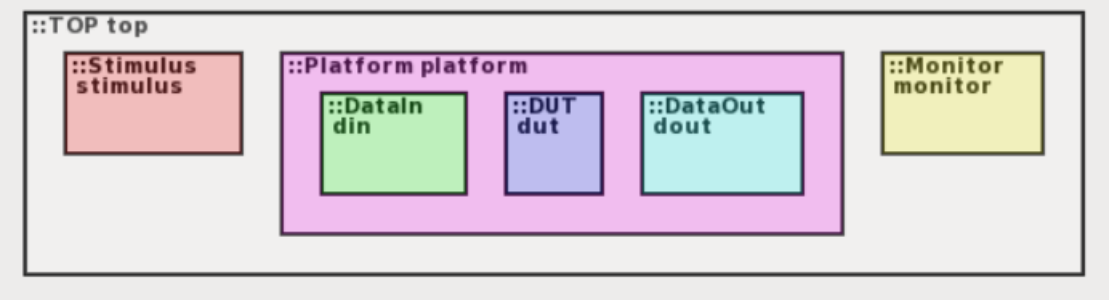


Figure 4: Mandelbrot Renderer Module Hierarchy Example

4.2 Port Mapping

In this next example we draw attention to the port mapping within the module hierarchy. Figure 5 shows a top module `md` containing two child modules left and right. The illustration also displays the connections of ports between child and parent modules. Module `md` has two ports: `outport left md` and `outport right md`. Port `outport left md` is bound to port `outport0` in the lowest level module `m0` contained in module left and port outport right `md` is binded to port `outport0` in module `m0` contained in module `right`.

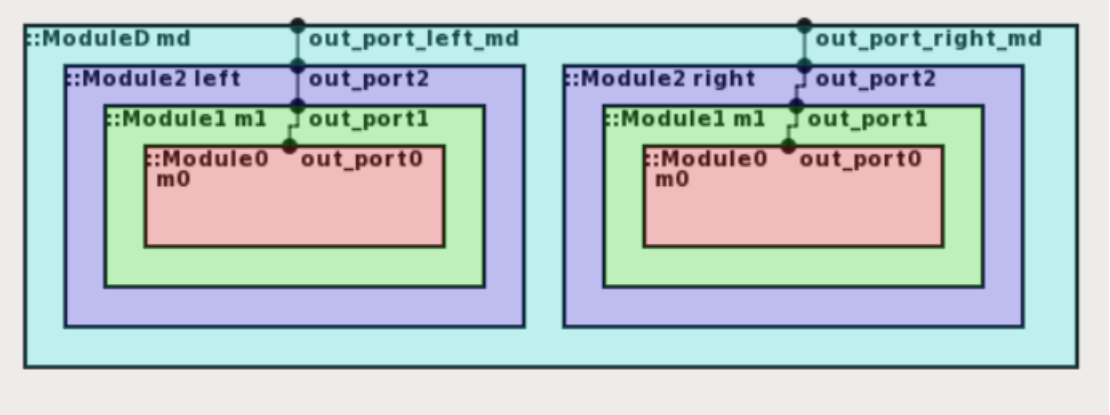


Figure 5: Port Mapping Example

4.3 Play

In this example we show the play model.

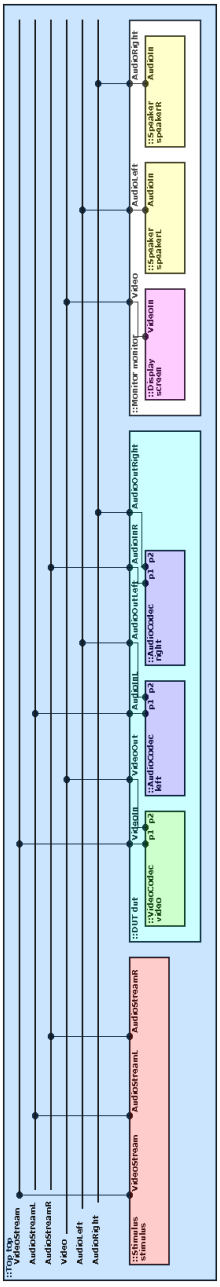


Figure 6: Play Example

4.4 Noc2x2

In this section we exemplify noc2x2.

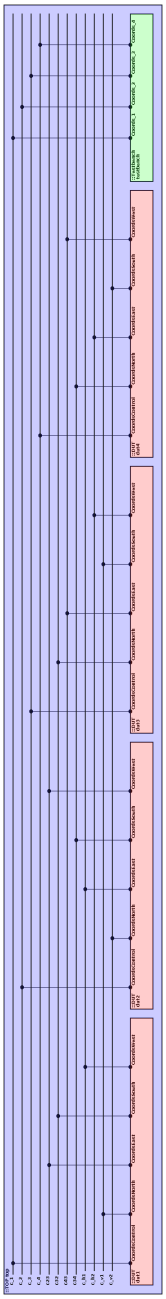


Figure 7: Noc2x2 Example

4.5 JPEG

This example is a JPEG.

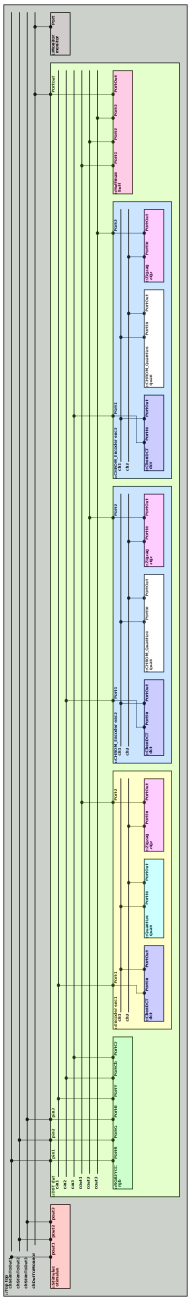
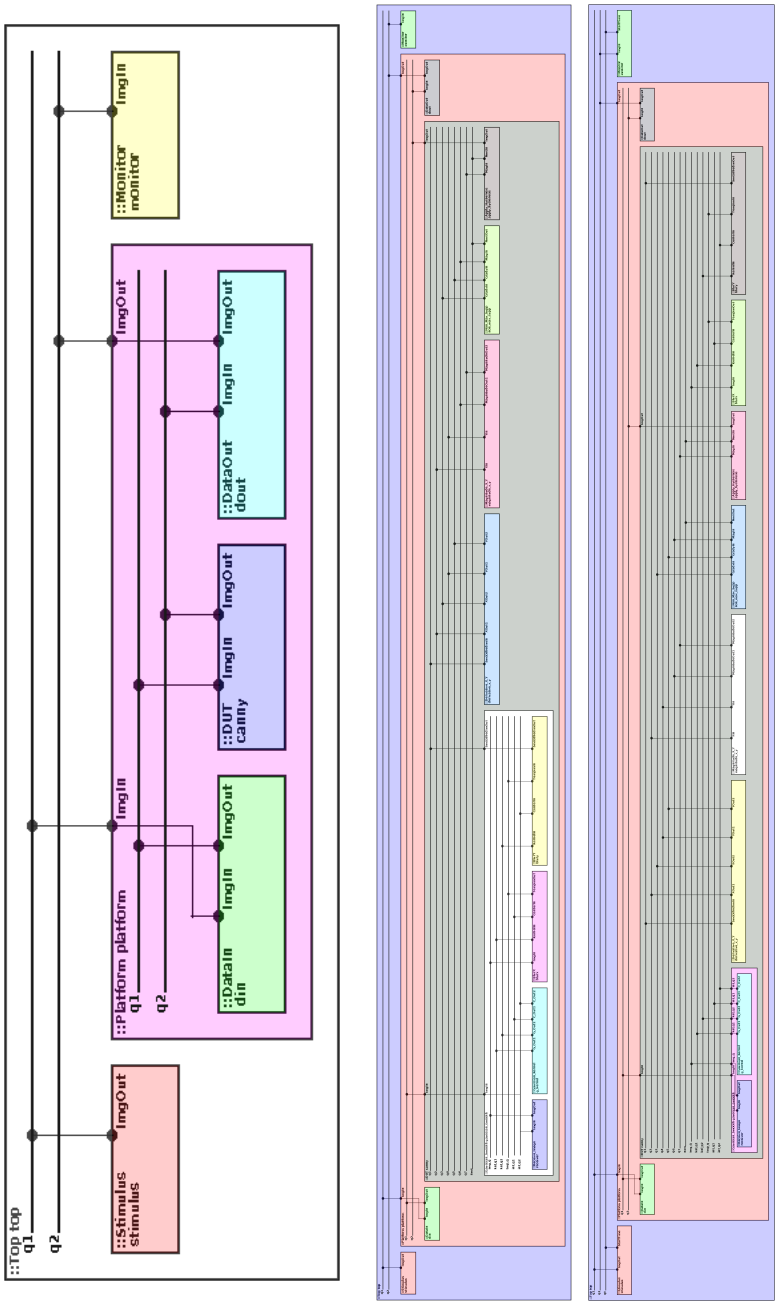


Figure 8: JPEG Example

4.6 Canny

In this section, we illustrate the development process of a SystemC model called `Canny`.



(a) Canny5 (b) Canny6 (c) Canny7

Figure 9: Canny5, Canny6, and Canny7 are the same SystemC project in different stages, with Canny7 being the most developed. From a comparison of Figure 9(a), Figure 9(b), and Figure 9(c), one can observe the developer's addition of modules, ports, and channels of the `Canny` SystemC model.

5 Conclusion and Future Work

In this report we discussed how the `Visual` tool visualizes SystemC modules, ports, and channels. Furthermore, we showed the modifiable options the user has when using the `Visual` tool. We also explored examples such as MandelBrot, Port Mapping, Canny, Play, and JPEG to show practical applications of the tool.

Improvements can be made with the layout of the visualization from the `Visual` tool. The version of the tool discussed in this report draws sibling modules such that they are always horizontally adjacent to each other. This can lead to some visualization in which the horizontal width is much larger than vertical length. Ideally, the most aesthetic drawings would illustrate equal horizontal and vertical length. Thus in future work, we plan to modify the visualization algorithm for sibling SystemC modules so that it supports both vertical and horizontal adjacency.

References

[1] ROSE Compiler Working Group. Rose open source compiler infrastructure. http://www.http://rosecompiler.org.

[2] IEEE Computer Society. *IEEE Standard 1666-2011 for Standard SystemC Language Reference Manual*. IEEE, New York, USA, 2011.

[3] Guantao Liu, Tim Schmidt, Zhongqi Cheng, and Rainer Dömer. Risc compiler and simulator, release v0.4.0: Out-of-order parallel simulatable systemc subset. Technical Report CECS-17-05, Center for Embedded Computer Systems, July 2017.

[4] Guantao Liu, Tim Schmidt, and Rainer Doemer. Recoding Infrastructure for SystemC (RISC) Compiler and Simulator. http://www.cecs.uci.edu/~doemer/risc.html.

[5] SpecC Technology Open Consortium. http://www.specc.org.