A cloud based signal waveform viewer using modern browser technologies

Efstathios Lymperidis
Department of informatics
and telecommunications engineering,
University of Western Macedonia,
Kozani, 50100, Greece,
Email: e.l.d@live.com

Minas Dasygenis
Department of Informatics
and Telecommunications Engineering,
University of Western Macedonia,
Kozani, 50100, Greece,
Email: mdasyg@ieee.org

***Abstract*—Modern age technologies allow us to create applications that are always accessible and available to a user. With the use of such technologies we have achieved the creation of an application that allows a developer to go through the whole process of creating an HDL program in a web environment. The application of the online HDL compiler is a complication of online tools, which include the cloud based waveform viewer. This waveform viewer is a tool allowing the user to have a waveform visualization of a testbench execution with all the signals and their values throughout the execution duration. It is designed to run partly on the server and partly on the client offering a fast, efficient and resource friendly environment. The use of modern web technologies allows us to create an interactive display of the waveforms to the user with a lot of features that provide him with optimal control over it.**

Keywords—cloud application; waveform viewer; HDL tool; modern technologies; waveform visualization;

I. INTRODUCTION

With today's progress in technology and telecommunications we experience an age where online tools come to shine and cloud applications take over their local counterparts. While local software has to be installed in the computer with specific operating systems and may need to use third party libraries, a cloud application is available at all times and accessible not only from any operating system but a wider range of computing systems like tablets or even cell phones.

With the use of modern web technologies, that allows us to create fast and interactive tools in a web environment, we have developed a tool for an online HDL compiler application which provides waveform visualization of a testbench execution. The online HDL compiler is an open source application composed of several tools that allow the users to go through the whole process of developing an HDL program while their data are available to them at any time or place as long as they have access to a browser.

While the development of the HDL compiler application is a collective effort of Dr. Minas Dasygenis and his team [1], the focus of this paper is the signal waveform viewer tool. The cloud based waveform viewer is an efficient graphical application where the user is able to dynamically extract all the relevant information he needs from the signals and their transitions that interest him. The tool is developed to read and analyze the output of a testbench execution and create interactive waveform visualizations for the user. A fixed demonstration is available for testing at http://arch.icte.uowm.gr/hdl/vcdview.php while the next step is the integration of this tool to the online HDL compiler.

The rest of the paper is structured as follows: The next section (Section II) presents some related work, while Section III, describes the structure of the signal waveform viewer tool. In Section IV we present in more details the way the waveform viewer displays, the signals and in Section V the client features are presented and explained. Finally, in Section VI we give the concluding remarks.

II. RELATED WORK

There are a lot of local applications that offer environments for compiling or simulating HDL code such us the ISE Design Suite [2], Icarus Verilog [3], VHDL Simili [4] and GHDL [5] . There also several web applications like the EDA Playground [6] and Verilog Online [7]. All of them though have the limitations of focusing on either compiling or simulating features while local software may require specific operating systems. The cloud based HDL compiler on the other hand creates on online environment that provides the user with tools to go through the whole process of HDL programming on a web environment.

Perhaps one of the most popular local applications that provide an interactive waveform viewer is the Modelsim [8] software. Although a great example for similar tools and even an inspiration to ours, it is still limited to simulation of a HDL code and it is still a local commercial application. We on the other hand are working on creating a complete open source tool available through the web that covers every need of an HDL developer.

III. Structure of the Tool

The signal waveform viewer is structured as a number of steps that include interactions between the client and the server as well as data analysis and processing, as shown in Figure 1. As a first step, when the website is loaded, the client sends a request for the data to the server. On the second step, the server reads as an input a value change dump (VCD) file, provided by the online HDL compiler, processes the information and converts the data in a simple form, easier for the client to process. The data in this form are then saved to a cache file in the server so that they can be retrieved if there is no change to the original VCD file. On the third step the data are sent to the client where the waves are designed on demand. The final step provides the user with features that allow him to interact with the waveform viewer, such as zooming in and out on the time scale, scrolling through the signals, getting information for specific time segments about the viewing signals and exporting their current instance of the waveform viewer as an image.

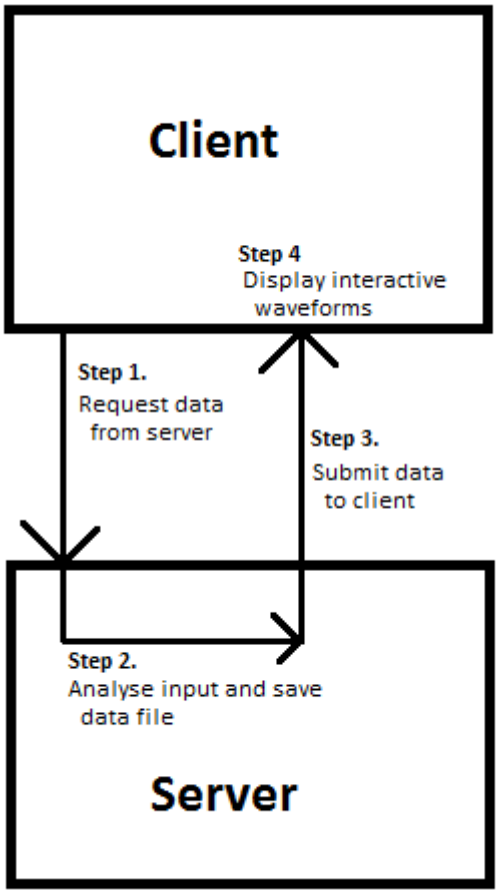

Fig. 1. The client requests data from the server which analyzes the input (VCD file) and returns them in an easy to process format.

A. Data Input

In order to understand the techniques used to import the data, it is essential to understand the input's format. The input in our tool is a VCD file generated by the HDL compiler after running a testbech code to verify the expected output of the HDL program. The VCD file contains basic information about the signals like their name, length in bits, signal type and the modules they belong to, but most importantly it describes the signals' value changes during the testbench execution.

The VCD files contain all that data in the following specific format, an example of which is shown in Figure 2. The header information contains the time scale, the VCD version and the generation date. Below that is the signals' definition containing the signals separated in their modules where they are defined by their name, length, type and each of them is assigned to a specific ASCII character for future reference within the file. The last part of the VCD file is the value changes section which gives us the time of a value change and below that the new value and the signal's character, this format is repeated for each time segment where at least one signal changes value.

```
$date
  Wed Oct 15 20:28:14 2014
$end
$version
  GHDL v0
$end
$timescale
  1 fs
$end
```

```
$scope module label_0_0_0 $end
$var reg 1 ] x $end
$var reg 1 ^ y $end
$var reg 1 _ andout $end
$upscope $end
```

```
#500000
1#
1&
1*
1:
```

Fig. 2. On the top part, we see the header information. In the middle part are some signals declaration and at the bottom part we see some value change information.

B. Signal Information Analysis

As mentioned in the structure, the server's first step is to read the VCD file and analyze the data into a format that is easy for the client to process. This format is called Javascipt Object Notation (JSON) and it is generated server-side using PHP. This allows us to store the VCD data in an easier format for the client offering overall a fast and resource friendly tool as the authors in [9] describe. The JSON data are stored in the server and we can retrieve instead of regenerating them in case there is no change to the VCD file. Although the JSON file is considerably bigger than the VCD file, it is optimal to use a resource such as the server's storage space instead of the server's processing power.

The PHP code uses a function which is called when the client requests the data through Asynchronous Javascript and XML (AJAX) [10] to either generate and send them or retrieve

them from the JSON file and forward them back to the client. Using the `filemtime()` PHP function we can get the time when the last modification was made to the VCD file and decide whether the JSON file is outdated and needs to be generated again.

To generate the JSON data we read the VCD file line by line. Upon each line there are several conditions using regular expressions to identify the line's information. Once we identify the information, we usually trim the unnecessary parts like the VCD keywords and keep those that interest us. To store the signal data an array is used containing the name, length, module (or module path) and type of each signal as well as the time segments and the signal's values on them. Some other data that we keep are the duration of the execution, the time scale and the number of time segments. All that data are then encoded to JSON using the PHP function `json_encode()`, stored into the JSON file and returned to the client.

This technique allows us to retrieve all the data we need by reading the VCD only once and since the JSON file contains the last version of the data, a lot of the times we will not need to regenerate it. Overall we end up using minimum resources but at the same time providing the client side javascript with a data format which is easy and fast for it to process.

IV. Displaying the Signal Waveforms

When the page has loaded, the AJAX to retrieve the JSON data is called. Once the client has the data, we use javascript to display a list of the signals within their modules and to draw them on a HTML 5 Canvas element.

A. Initializing the Waveforms Viewer

Once we have the JSON data, we extract from them the number of time segments in order to draw the selected signals later on at their full length unless an other zoom value is specified.

The next thing we have to do is creating the list of signals available. This is done as shown in Figure 3 by nested "div" elements for the modules with the signal names inside them as "span" elements which trigger an event upon been clicked. This event adds the signal to an array with the signals to be displayed and calls the function to draw the waveforms.

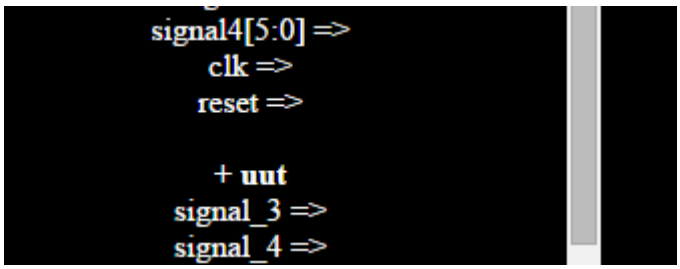

Fig. 3. The list of signals divided by their modules.

B. Drawing the Selected Signals

When the user clicks on a signal from the list created in the initialization, an event is called which adds that signal's symbol in a global javascript array and then calls the function to draw the signals on the HTML 5 Canvas element.

First of all, the values of the first and last time segments that will be displayed are calculated and then, in order to create two dimensional graphics in a canvas with javascript, we need to get an object linked the canvas. This object provides us with the methods and properties required to draw upon the canvas element. These properties allow us to specify colors, font types and size, while the methods allow us to draw lines, shapes or text.

We begin by drawing the name and size (if more than one bit) of the signal. Then we move a set amount of pixels on the "X" axis and begin drawing the signal's waveform. In order to draw the waveform, we go through the signal's values until we reach the time segment where we need to begin displaying. Since the signal's data give us information about the change of value, we can store the value and replace it when a new one appears. To draw these values we need to start a line from the last time segment's coordinates in the canvas to the next. Each possible value for one bit signals has a certain offset in the "Y" axis. When the value changes we update this offset and create a vertical line connecting the end of the last pulse to the next. For non expanded signals with length more than one bit, instead of having a different offset to signify their values, we draw their values as text at the time segments where they change if there is enough space to display it correctly. Their values are enclosed by two horizontal lines, one above and one below. On the time segments where their value changes, we create a vertical line with a small circle of it to specify that change visually.

Once we finish drawing the waveforms, we create another canvas object linked to another canvas element which is not displayed to the user. This canvas element works as a buffer and we create a copy of the original canvas on it. This allows us to add dynamic information on the visible canvas by replacing it with the buffer's contents when needed, instead of redrawing it every time.

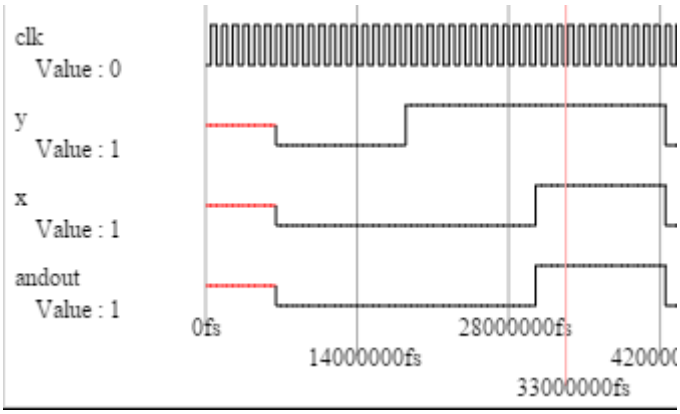

Fig. 4. Display of the signals' waveforms with some additional information.

V. Client Features

There are a number of features that allow the user to interact dynamically with the signals waveform viewer in order

to gain focused and more detailed information about parts of the signals that interest them.

A. Time Scaling

The feature to alter the time scale (zoom in and out) is available to the user through a slider under the canvas. As a default value the whole duration is shown. The way it is developed is by using an event listener to the slider which changes the zoom value and calls the drawing function to update the canvas. It will keep the current time segment where the drawing begins and recalculate how many time segments need to be drawn to the rest of the canvas. The zoom value is percentage based, pointing to the percentage of the signals that is shown. So the starting value is one hundred percent (100%) which is also the maximum value the slider allows while one percent (1%) is the minimum allowed value.

B. Time Scrolling

The user is allowed by holding the right mouse click down and dragging left or right in the canvas to scroll through the signals on the time axis. In order to develop this function we need to disable the default behavior of the right click inside the canvas element when the website has loaded. Then using two event listeners, one for mouse down and one for mouse up, we can calculate the original position of the mouse and make sure the mouse button remains clicked. The next step is to add an other event listener on the canvas for mouse movement which will trigger when the mouse hovers above the canvas element. In that case we can calculate the difference between the starting position of the mouse and the current position and allow the waveforms to start and end at different time segments depending on the movement's direction and speed. Then we also have to make sure that the scrolling keeps the displayed signals within their duration and do not allow them to go lower than zero time and higher than the execution duration.

C. Pointer Information

The pointer information is displayed when the mouse hovers above the canvas, giving the value of each signal and the exact time value of that time segment. The user is also able to assign a more permanent marker by left clicking on the canvas on the time segment he is interested in. Knowing the canvas width dimensions, we can easily calculate if the mouse is within the signals region of the waveform viewer in order to display the additional information.

To display the information to the user on mouse hover, a buffer canvas is used. We do this to avoid recreating the canvas on every event trigger and offer a faster and more efficient function. Once the buffer is copied to the visible canvas, we calculate the value for each signal to that time segment and display it below the signal's name. That time segment is also marked with a vertical red line and the exact time value to help the user understand its position and manipulate it with ease as shown in Figure 4. The more permanent pointer information feature uses the same process with the main differences been that it is displayed on click and it is also copied on the buffer canvas in order to remain into an extracted image.

Using those features, the user has the ability to extract detailed information about the signals at any time segment in a fast and dynamic way.

D. Exporting Image

The last user feature is the ability to export the canvas content as a JPG image. This will allow the user to keep a local copy of the waveform with information that interests them, which they can refer upon at any time. The user is provided with a button that triggers a function which uses a canvas object method that returns a link to the image generated by the canvas.

An important note here is that the canvas we use to generate the image is the buffer. This way, it is made sure that the image will contain only the more permanent elements that are displayed.

VI. Conclusion

Mobility and independence of the machine and third party software are becoming a need in modern days and perhaps even more so in the near future. The tool presented here is designed to deal with that need, offering a web environment accessible by any operating system and even a wide range of devices. The cloud based signal waveform viewer is also part of an application that deals with another problem of HDL programming, the severe lack of software that gives the developer the ability to go through the whole process of creating and testing an HDL program. We have seen here that such applications can not only be developed in a way that makes them accessible at any time and place but they can also be designed providing the users with interactive and dynamic features on match with their local counterparts.

References

[1] M. Dasygenis, "A distributed VHDL compiler and simulator accessible from the web." Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on. IEEE, 2014.

[2] Xilinx, "ISE Design Suite", http://www.xilinx.com/products/design-tools/ise-design-suite.html

[3] S. Williams, "Icarus Verilog," http://iverilog.icarus.com/

[4] Symphony EDA, "VHDL Simili," http://www.symphonyeda.com/products.htm

[5] T. Gingold, A. Lauger, F. Tappero, C. Jarron, "GHDL," http://ghdl.free.fr/

[6] Victor EDA, "EDA Playground," http://www.edaplayground.com/

[7] iVerilog, "Verilog Online", http://iverilog.com/index.php

[8] Mentor Graphics, "Modelsim," http://www.mentor.com/products/fv/modelsim/

[9] Ecma International, "The JSON Data Interchange Format," Standard ECMA-404, October 2013. [Online]. Available : http://www.ecma-international.org/publications/standards/Ecma-404.htm

[10] J. J. Garrett, "Ajax: A New Approach to Web Applications," 2005. [Online]. Available : https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf