WeiGUI

The Simple Verilog GUI Debugger

This application is designed to provide a quick way of creating a GUI debugger for Verilog code. The underlying idea is that the user will edit a configuration file which specifies what signals they want to display and how they want them displayed. The tool will then add the needed hooks to the Verilog code and create a visual display of that data. It is implemented using Python, Tkinter and ttk.

This application is still in development and as such it might be buggy and slow. If you have any questions, feature suggestions or bugs to report, please send email to miaowei@umich.edu

This tool is designed to run under Linux, using version 2.6.6 or later of Python (but not Python 3). You also need to have Tkinter installed. As of Winter 2014 CAEN Linux load can run this application without modification.

Also, we’ve included an example of using the GUI debugger. While the directions below are hopefully complete, if in doubt, you can look to see how things are implemented there.

Installation and Getting Started

You will need to download the zip file for WeiGUI and put the directory supplied in the same directory as your Makefile and that Makefile must be one level above the directory where your testbench is found. The testbench(es) must be in a directory called “testbench”.

You then need to modify your testbench as follows:

1. Insert the line:
   ```
   `include "testbench/dataoutput.v"
   ```
   immediately after any variable declarations.

2. In the initial block, insert the line:
   ```
   f = $fopen("testdata");
   ```

3. In the main loop (the always block that runs the processor), insert the line:
   ```
   displayinclude;
   ```

Once you’ve modified the testbench as described above, all you’ll need to do is create a configuration file and run the GUI debugger. While you won’t be running the debugger at this point, it is simple to do. Just type the following at the command line from the directory with your Makefile.

```
python  guidebugger/gui.py
```
Using the GUI Debugger

After making the above changes to your testbench, you’ll be able to run the debugger. Below is a screen capture which gives you a sense of what the debugger looks like. Just press Next Cycle, Previous Cycle or Jump to Cycle to change the cycle. You can also change the format from hexadecimal to decimal using the radio buttons.

Sample Project

Included with the code, there’s a sample project that allows you to see the debugger in use. This sample project is the same as the project 3 skeleton code, with the configuration file added and changes made to the testbench as described above. As the previous introduction, just type

```
python guidebugger/gui.py
```

in the directory with the project makefile.

You should now see a window similar to the picture above. When a register’s value changes, its background color will become green. When the signal is x, it will be red. When the signal is z, it will be yellow. You can try to use this GUI debugger and change some parameter in the config file before you build your own one.
Protocol of Config file

The core of the application is the configuration file. A single configuration file controls both the GUI debugger’s layout and creates the needed interface to the Verilog.

Before we describe the configuration file itself, we want to warn you of four rules you should be aware of before creating or editing a configuration file for this debugger.

1. NEVER removed or change sequence of those lines with = and [].
2. SOME signal are necessary, like clock name. others are not. It will be specific in protocol
3. EVERY data should belong to a group and a tab. OK to multiple the data to several different group.
4. There can’t be any blank lines.

Now that that is said, let’s describe the layout of the configuration file. We suggest you look at the sample configuration file as you read this section.

Section Name

The configuration file is broken into different sections. Each section name is in brackets. These sections break the file into four parts and are: [Config], [Instruction], [Data], and [Register]. These section headers must all exist and be in the order listed above and each it to be on its own line. Let’s look at each of the sections.

[Config]
The Config section has only two items: Screen Resolution and ClockName.

- **ScreenResolution**: Specifies the initial size of the screen in pixels. This is specified by two numbers with a * between them. For example: ScreenResolution=1024*600 would mean that the initial size of the screen is 1024 by 600. You **must** supply an initial value.
- **ClockName**: The name of your clock as seen in your testbench. You **must** supply a value here. For example: ClockName=clock_count.

[Instruction]
This section allows you to display instructions in a given stage of the processor. While most useful in an in-order processor, it can also be helpful in an out-of-order processor.

- **PCRegName**: the name of the program counter as seen by the testbench. You **must** supply a value here. For example: PCRegName=pipeline_0.if_stage_0.PC_reg
- **InstrName**: This is a comma separated list of pipeline stage names. You **must** supply at least one value here. For example: InstrName=IF, IF/ID, ID/EX, EX/MEM, MEM/WB
- **InstrSignal**: This is the name of each instruction in each stage. There must be the same number of items listed here as in InstrName. For example: InstrSignal=pipeline_0.if_IR_out, pipeline_0.if_id_IR, pipeline_0.i d_ex_IR, pipeline_0.ex_mem_IR, pipeline_0.mem_wb_IR
[Register]
Here you will specify the register file’s name. You can only specify one register file (though it could be the ARF or PRF as desired). Therefore there should be exactly one line in this section. For example:
  pipeline_0.id_stage_0.regf_0.registers[31:0]

[Data]
This is where you will describe and organize the signals you want to display. You can specify different tabs and different groups inside of those tabs. For each tab there must be at least one group. For each group there must be at least one signal. Every signal belongs to a group, and every group belongs to a tab.

Tabs
To indicate that you are starting a new tab, put the tab name you want in curly braces {}. So {data} would indicate that there is a tab called “data”.

Groups
Each data signal (described below) will be organized into groups (boxes in the GUI). These will be in parentheses and indicate where the box is to be placed. So (IF, 0, 0) would indicate the box/group name will be “IF” and it is to be placed in the top row (the first zero) and the left-most column (the second zero). (ID/EX, 3, 1) would indicate the group name is to be ID/EX and it should be placed in the fourth row and the second column.

Signals
These are the values that are to be displayed in the GUI debugger. Each signal can be either a single value, or an array. To specify a single signal, you supply its full name, including the dot notation. For example:
  pipeline_0.id_ex_rega

To state an array, besides its full name, you would also need to specify its size. The format is:
  Signal_name[31:0]
Where 31 is the upper bound of the array and 0 is the lower bound. For example:
  memory.unified_memory[8195:0]

By default the debugger will supply only the name after the prefix (so after the last dot). If you want to display more of the name, you can add an argument to the signal.