PyVHDL Documentation

Release 0.0.1

Vern Muhr

October 26, 2016

Contents

CHAPTER 1

Introduction:

PyVHDL is an open source project for simulating VHDL hardware designs. It cleanly integrates the general purpose Python programming language with the specialized VHDL hardware description language. You can write the testbench for your VHDL design in Python. You can use all the great features of Python to quickly create your testbench. Some advantages of using Python include:

- It's an easy to learn, modern mainstream language.
- Object oriented for better organization of code.
- Simple and powerful formatting of of text output.
- Easy file read and write of formatted data.
- Quality modules available for just about any programming task.

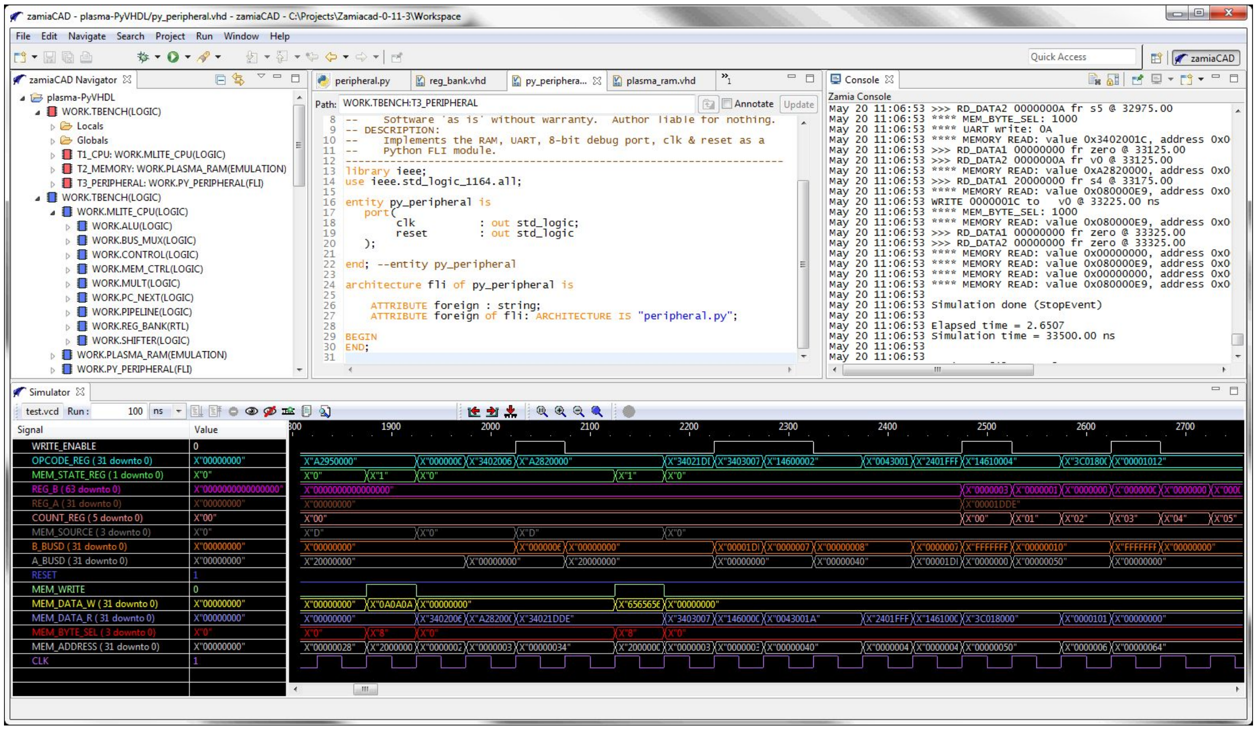
Python can be used for more than just testbenches. You can also create architectures in your VHDL design that are written in Python. Standard VHDL syntax is used to link the architecture to a Python file. Concurrent Python processes can wait for signal transitions, wait for a specified period of time, read values of signals in the design, or update the values of signals with an optional delay. This capability is useful for quickly prototyping the behavior of an entity, or emulating complex behavior like a memory device or UART.

PyVHDL features a unified execution environment. VHDL is translated to Python bytecode, and runs on the same interpreter as the Python code. There is no overhead wasted on passing data and maintaining synchronization between separate Python and VHDL environments. An advantage of using Python is that PyVHDL can utilize the work of other projects in the very active Python community. For example, using the Cython optimising compiler, the speed of the simulator event queue can be improved by more than a factor of seven.

An important goal of PyVHDL is to create a VHDL simulator that the open source community can easily make contributions to. The simulator is written in Python, a language familiar to many programmers. Python is much easier to read and modify compared to most languages. Also, the hurdle of setting up an obscure tool chain environment is not an issue. The PyVHDL project is maintained on GitHub.

The excellent zamiaCAD Eclipse based IDE is packaged with PyVHDL. The zamiaCAD IDE is an open source Eclipse based platform for advanced VHDL hardware design. PyVHDL is tightly integrated with the zamiaCAD IDE. Features provided by zamiaCAD include:

- An environment for creating and managing PyVHDL projects.
- VHDL editing with syntax highlighting and checking.
- Graphical tools for visualizing and navigating the VHDL hierarchy.
- Configuration and running of simulations.
- A waveform viewer for displaying the results of a simulation.
- A VHDL parser used by PyVHDL for analysis and elaboration.



```
--    Software 'as is' without warranty.  Author liable for nothing.
-- DESCRIPTION:
--    Implements the RAM, UART, 8-bit debug port, clk & reset as a
--    Python FLI module.
---------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity py_peripheral is
   port(
        clk          : out std_logic;
        reset        : out std_logic
   );

end; --entity py_peripheral

architecture fli of py_peripheral is

    ATTRIBUTE foreign : string;
    ATTRIBUTE foreign of fli: ARCHITECTURE IS "peripheral.py";

BEGIN
END;
```

```
May 20 11:06:53 >>> RD_DATA2 0000000A fr s5 @ 32975.00
May 20 11:06:53 **** MEM_BYTE_SEL: 1000
May 20 11:06:53 **** UART write: 0A
May 20 11:06:53 **** MEMORY READ: value 0x3402001C, address 0x0
May 20 11:06:53 >>> RD_DATA1 00000000 fr zero @ 33125.00
May 20 11:06:53 >>> RD_DATA2 0000000A fr v0 @ 33125.00
May 20 11:06:53 **** MEMORY READ: value 0xA2820000, address 0x0
May 20 11:06:53 >>> RD_DATA1 20000000 fr s4 @ 33175.00
May 20 11:06:53 **** MEMORY READ: value 0x080000E9, address 0x0
May 20 11:06:53 WRITE 0000001C to   v0 @ 33225.00 ns
May 20 11:06:53 **** MEM_BYTE_SEL: 1000
May 20 11:06:53 **** MEMORY READ: value 0x080000E9, address 0x0
May 20 11:06:53 >>> RD_DATA1 00000000 fr zero @ 33325.00
May 20 11:06:53 >>> RD_DATA2 00000000 fr zero @ 33325.00
May 20 11:06:53 **** MEMORY READ: value 0x00000000, address 0x0
May 20 11:06:53 **** MEMORY READ: value 0x080000E9, address 0x0
May 20 11:06:53 **** MEMORY READ: value 0x00000000, address 0x0
May 20 11:06:53 **** MEMORY READ: value 0x080000E9, address 0x0
May 20 11:06:53
May 20 11:06:53 Simulation done (StopEvent)
May 20 11:06:53
May 20 11:06:53 Elapsed time = 2.6507
May 20 11:06:53 Simulation time = 33500.00 ns
May 20 11:06:53
May 20 11:06:53
```

Signal	Value
WRITE_ENABLE	0
OPCODE_REG (31 downto 0)	X"00000000"
MEM_STATE_REG (1 downto 0)	X"0"
REG_B (63 downto 0)	X"0000000000000000"
REG_A (31 downto 0)	X"00000000"
COUNT_REG (5 downto 0)	X"00"
MEM_SOURCE (3 downto 0)	X"0"
B_BUSD (31 downto 0)	X"00000000"
A_BUSD (31 downto 0)	X"00000000"
RESET	1
MEM_WRITE	0
MEM_DATA_W (31 downto 0)	X"00000000"
MEM_DATA_R (31 downto 0)	X"00000000"
MEM_BYTE_SEL (3 downto 0)	X"0"
MEM_ADDRESS (31 downto 0)	X"00000000"
CLK	1

Fig. 1.1: The zamiaCAD+PyVHDL IDE

Caution: PyVHDL 0.0.1 is an alpha release! Not all VHDL syntax is supported. Un-supported VHDL syntax includes:

- Functions and Procedures.
- Type statements.
- Arrays and structures.
- Libraries written in VHDL (std_logic and std_logic_vectors are supported natively in Python).

Despite these limitations, PyVHDL can simulate the 32-bit, MIPs compatible, plasma CPU design, and pass plasma's instruction set test suite!

CHAPTER 2

Documentation Contents:

2.1 Quickstart

Quickstart table of contents:

2.1.1 Requirements

Note: At this time, zamiaCAD+PyVHDL has been tested on Windows 7 and Windows 10.

- Java Runtime Environment version 1.8. Install the 64 bit version of the JRE.
- Stackless Python 2.7 choose the amd64 version.

Note: Be sure python.exe in on your path.

2.1.2 Install PyVHDL

Download PyVHDL-0.0.1.zip from GitHub here. Extract the contents of the zip file to a location of your choice. I use C:/Projects/PyVHDL

2.1.3 zamiaCAD

As noted above, PyVHDL runs alongside the zamiaCAD Eclipse IDE. zamiaCAD provides an environment for creating VHDL design projects, editing VHDL files, analyzing and elaborating a VHDL design, controlling a simulation, and displaying waveform files generated by a simulation. If you have used the Eclipse framework before, you will feel fairly at home using zamiaCAD. zamiaCAD has many features that facilitate working with VHDL designs. Read this zamiaCAD tutorial to learn more about zamiaCAD's features.

Zamiacad is split into an IDE component, and a VHDL processing component. It is possible to run simulations without the IDE in command line mode using just the VHDL processing component.

Initial zamiaCAD setup

Navigate to the **PyVHDL-0.0.1** directory. Double click on **run_giu.bat**. The zamiaCAD+PyVHDL splash screen should appear:

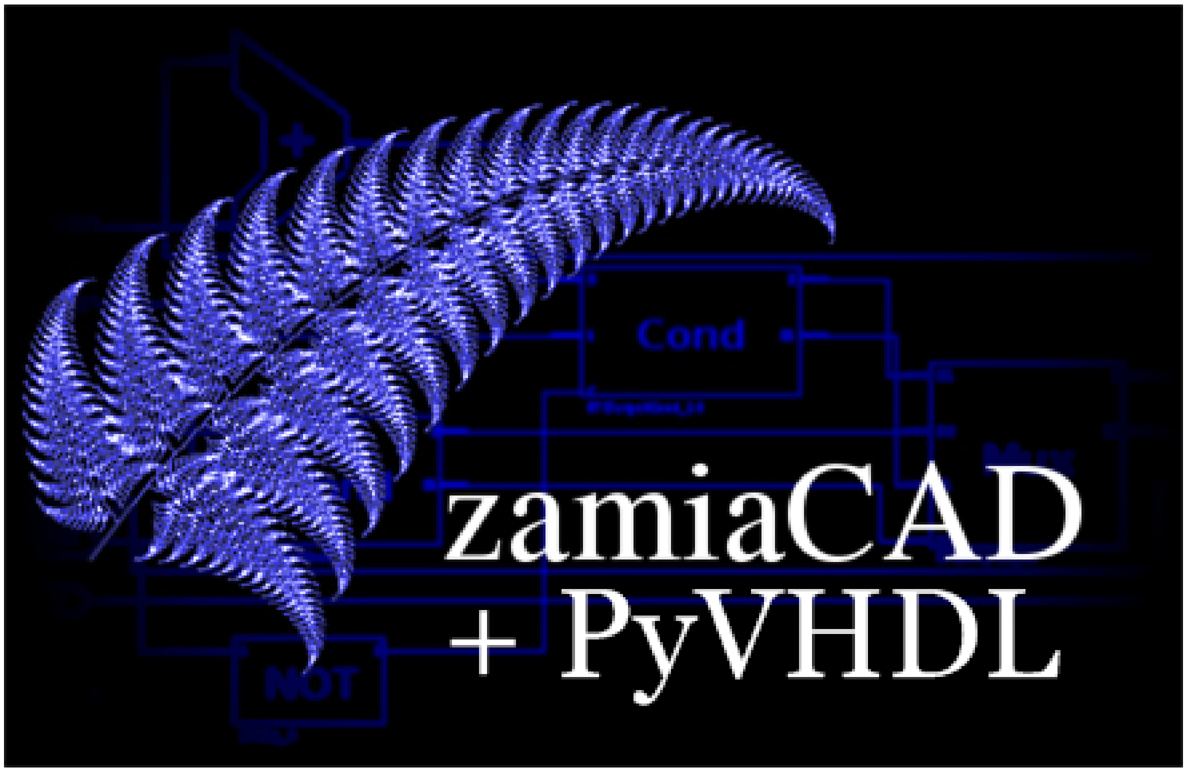


Fig. 2.1: zamiaCAD+PyVHDL Splash Screen

Next a **Workspace Launcher** dialog should appear. Accept the default workspace that appears in the Workspace textbox. It should be a folder named `workspace` in the PyVHDL installation folder:

Click on the **Use this as the default and do not ask again** checkbox, then click on **OK**. The initial zamiaCAD+PyVHDL IDE should appear.

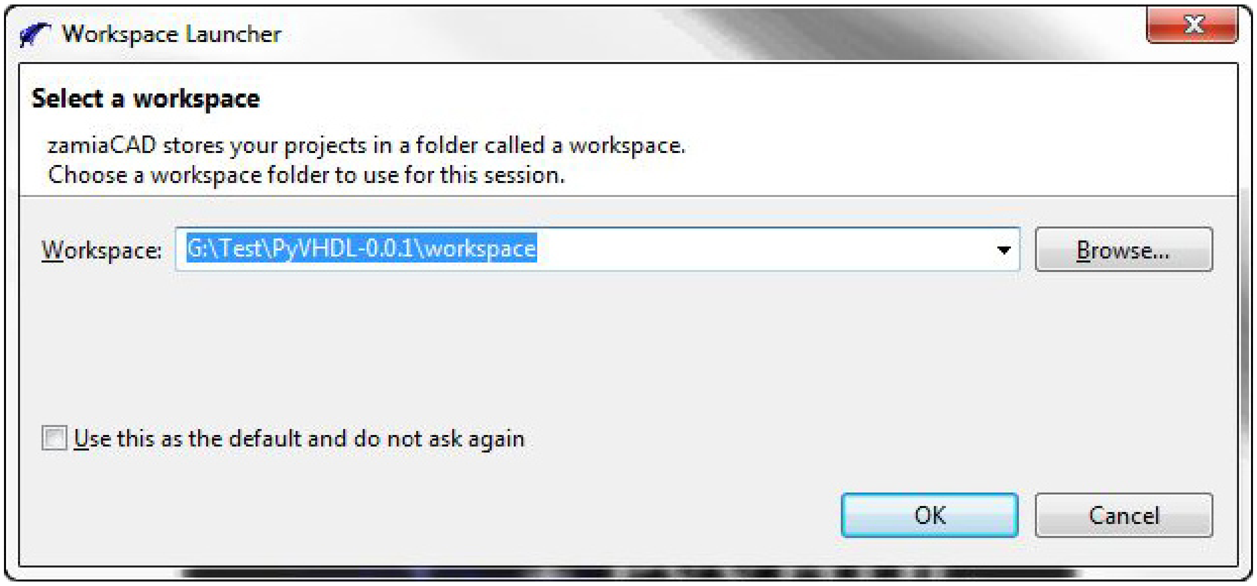

Workspace Launcher
Select a workspace
zamiaCAD stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.
Workspace: G:\Test\PyVHDL-0.0.1\workspace
Browse...
Use this as the default and do not ask again
OK
Cancel

Fig. 2.2: Workspace Launcher Dialog

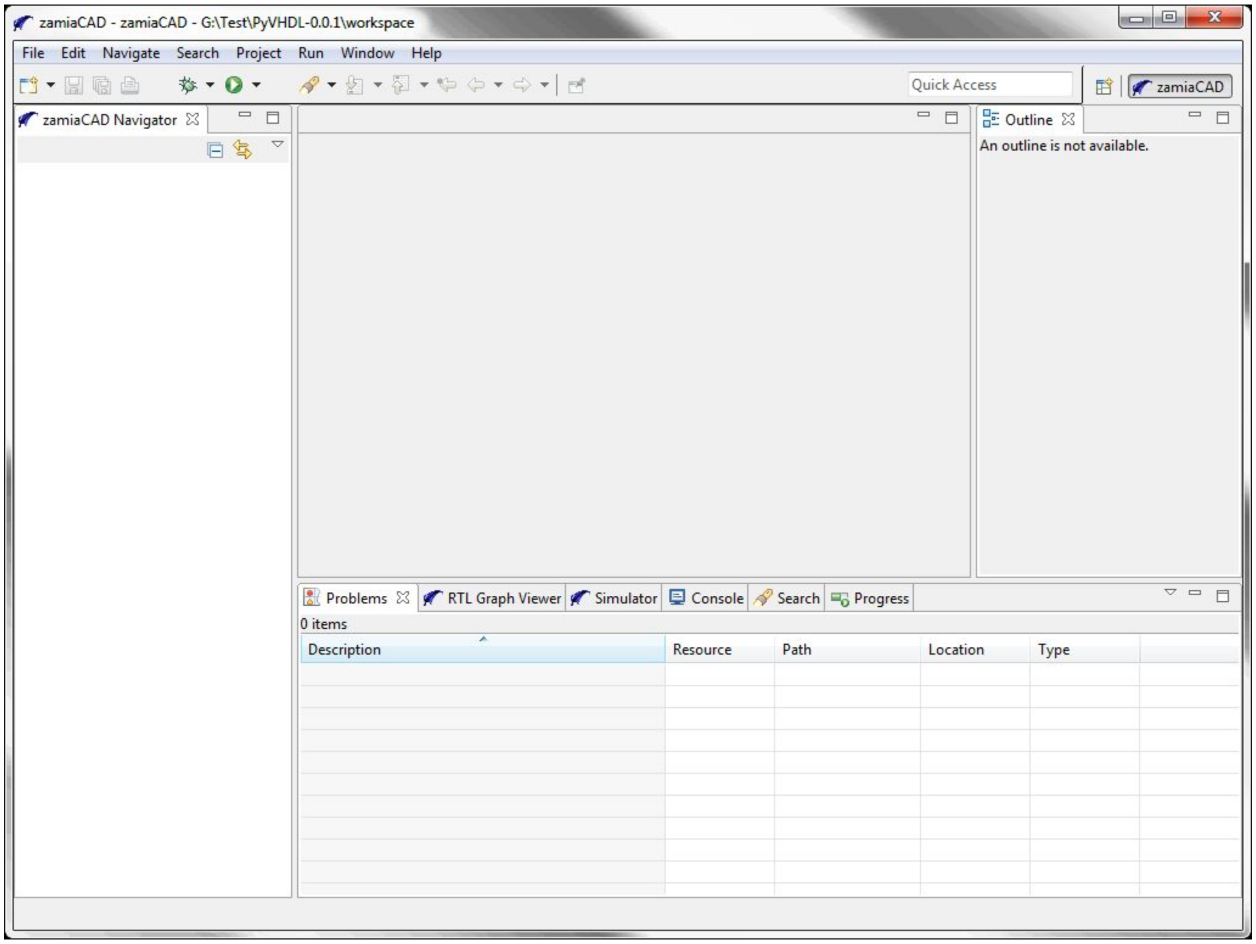


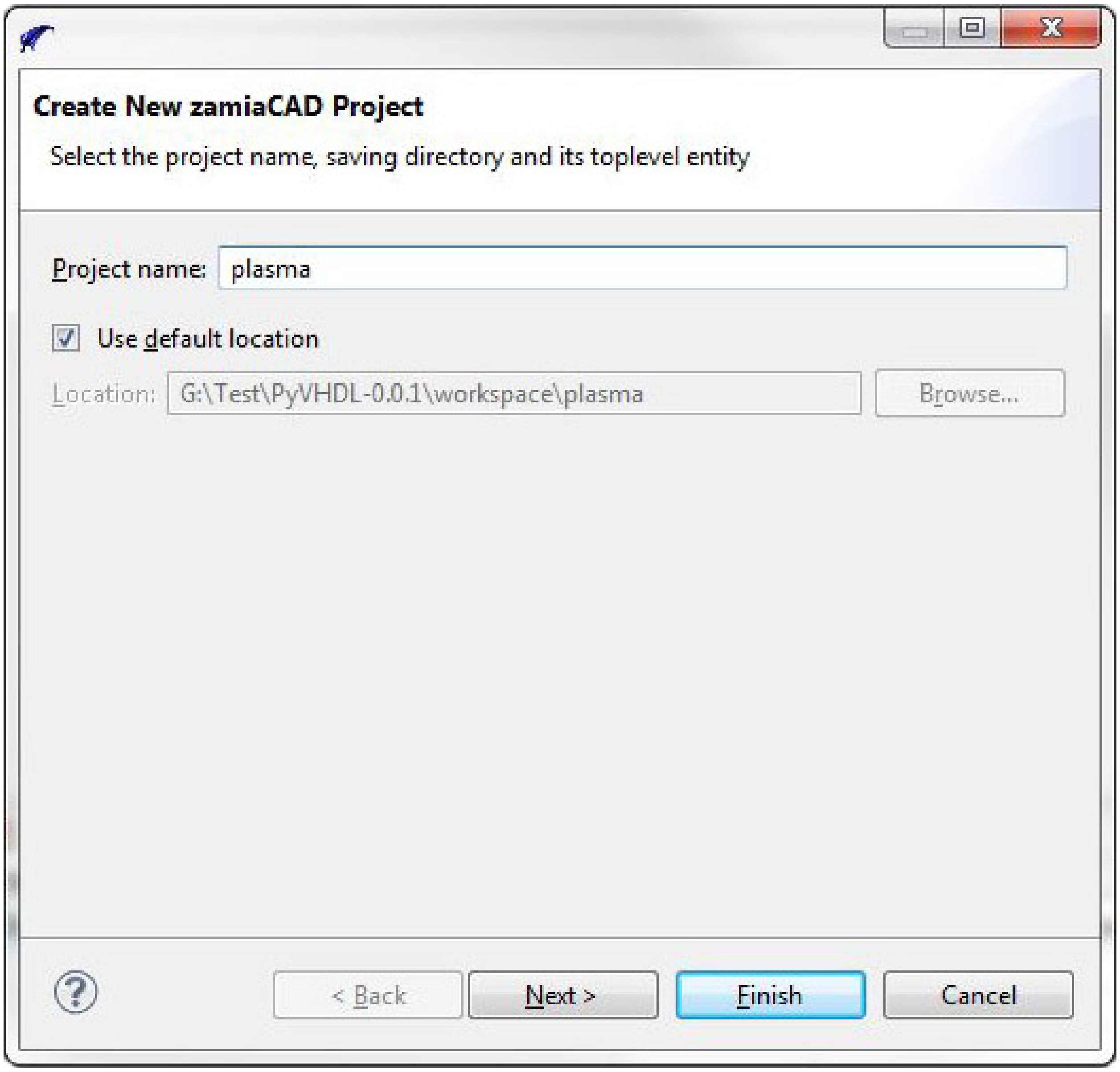
Fig. 2.3: Initial zamiaCAD+PyVHDL IDE

2.1.4 Run the plasma demo

The plasma demo is a zamiaCAD+PyVHDL project for simulating the plasma CPU design. Plasma is a 32-bit MIPs compatible processor. The design is a set of RTL style VHDL files.

Setup a new project

Before the plasma project can be simulated, a new project must be setup in the zamiaCAD Navigator view. Make sure the zamiaCAD perspective is active. It should appear highlighted in the upper right corner of the IDE window. On the top menu select **File > New > zamiacad Project**. The **Create New zamiaCAD Project** dialog will appear.



Create New zamiaCAD Project

Select the project name, saving directory and its toplevel entity

Project name: plasma

[✓] Use default location

Location: G:\Test\PyVHDL-0.0.1\workspace\plasma

Fig. 2.4: Create New zamiaCAD Project Dialog

In the Project name textbox enter `plasma`. Click Finish. A **Do Full Build?** dialog will appear. Click **No**. A project folder labled `plasma` will now appear in the IDE **zamiaCAD Navigator** view.

Import the archived project

The zamiaCAD+PyVHDL IDE allows projects to be archived, and later imported back into the IDE workspace. The folder at **PyVHDL-0.0.1/share/saved-projects** contains the plasma project archived as `plasma-PyVHDL.zip`.

Make sure the plasma project is highlighted in the IDE **zamiaCAD Navigator** view. Right click on the plasma project folder. In the context menu that appears select **Import...**. The Import dialog will appear. Double click on the **General** folder, and select **Archive File**, then click the **Next >** button.

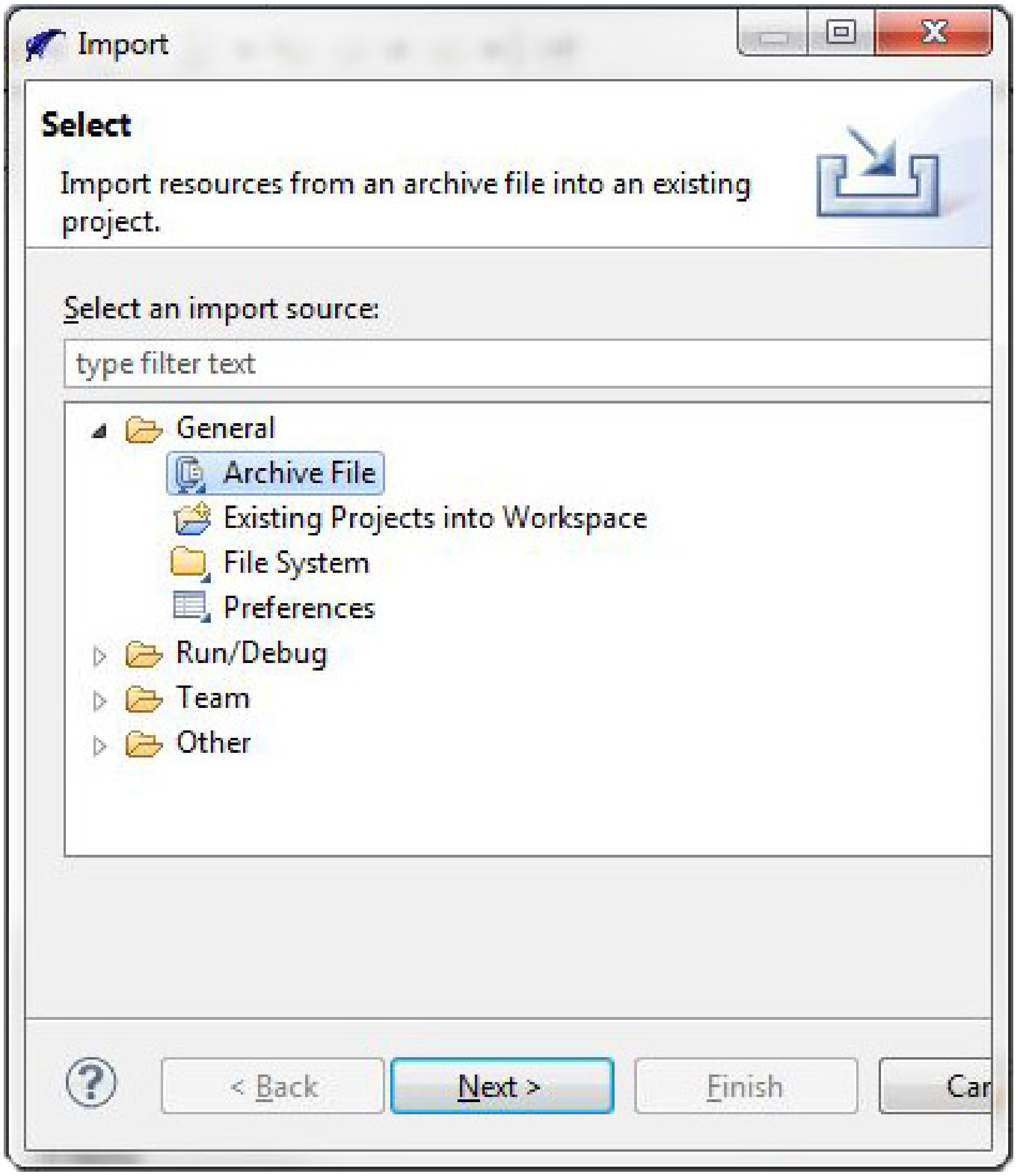
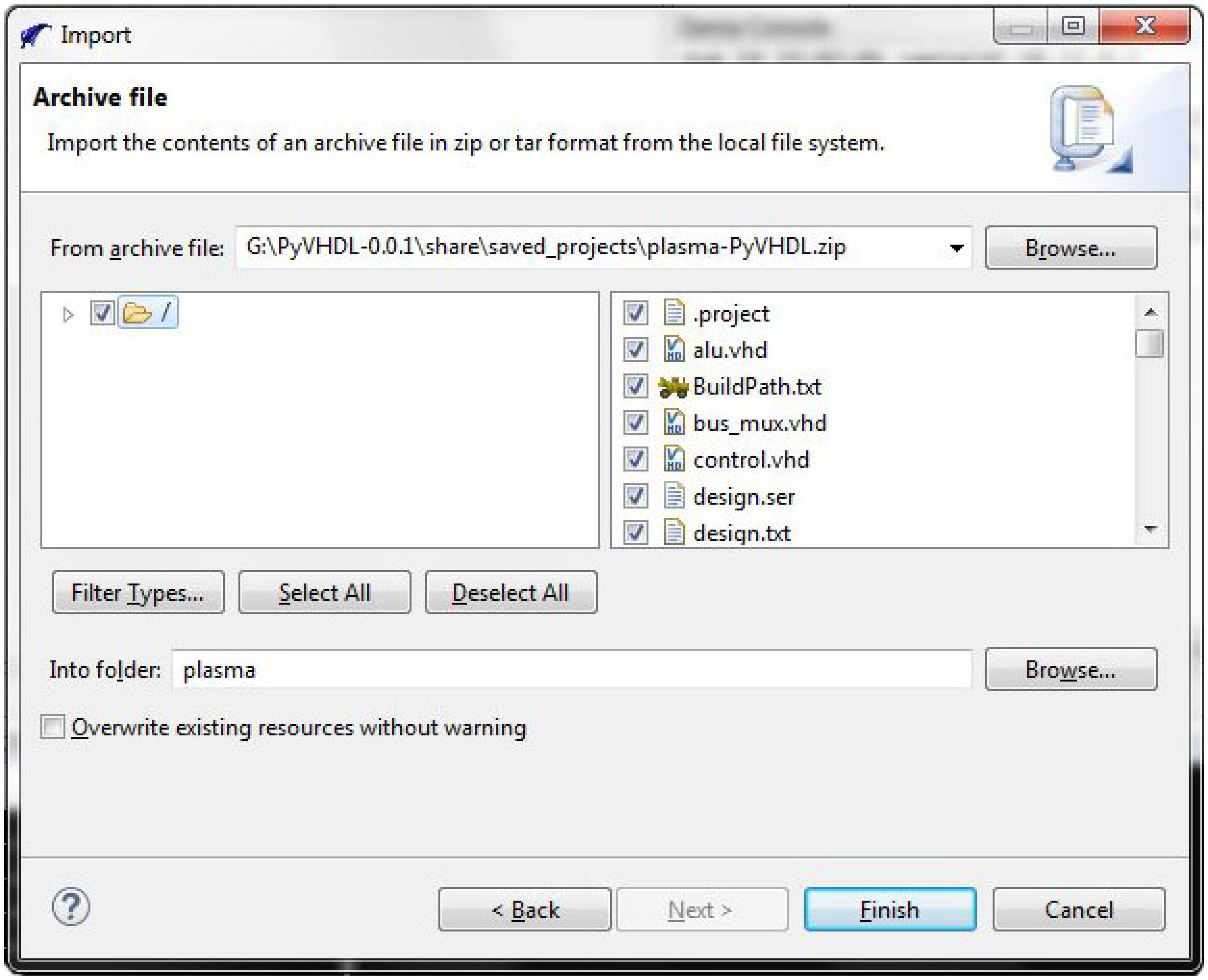


Fig. 2.5: Import Dialog

Next a dialog appears where the import is setup. Click on the **Browse...** button to the right of the **From archive file:** text box. Navigate to **PyVHDL-0.0.1/share/saved-projects** . Select the **plasma-PyVHDL.zip** file. Click the **Open** button. Now the completed Import dialog should look like this:


Import

Archive file

Import the contents of an archive file in zip or tar format from the local file system.

From archive file: G:\PyVHDL-0.0.1\share\saved_projects\plasma-PyVHDL.zip

[✓] /

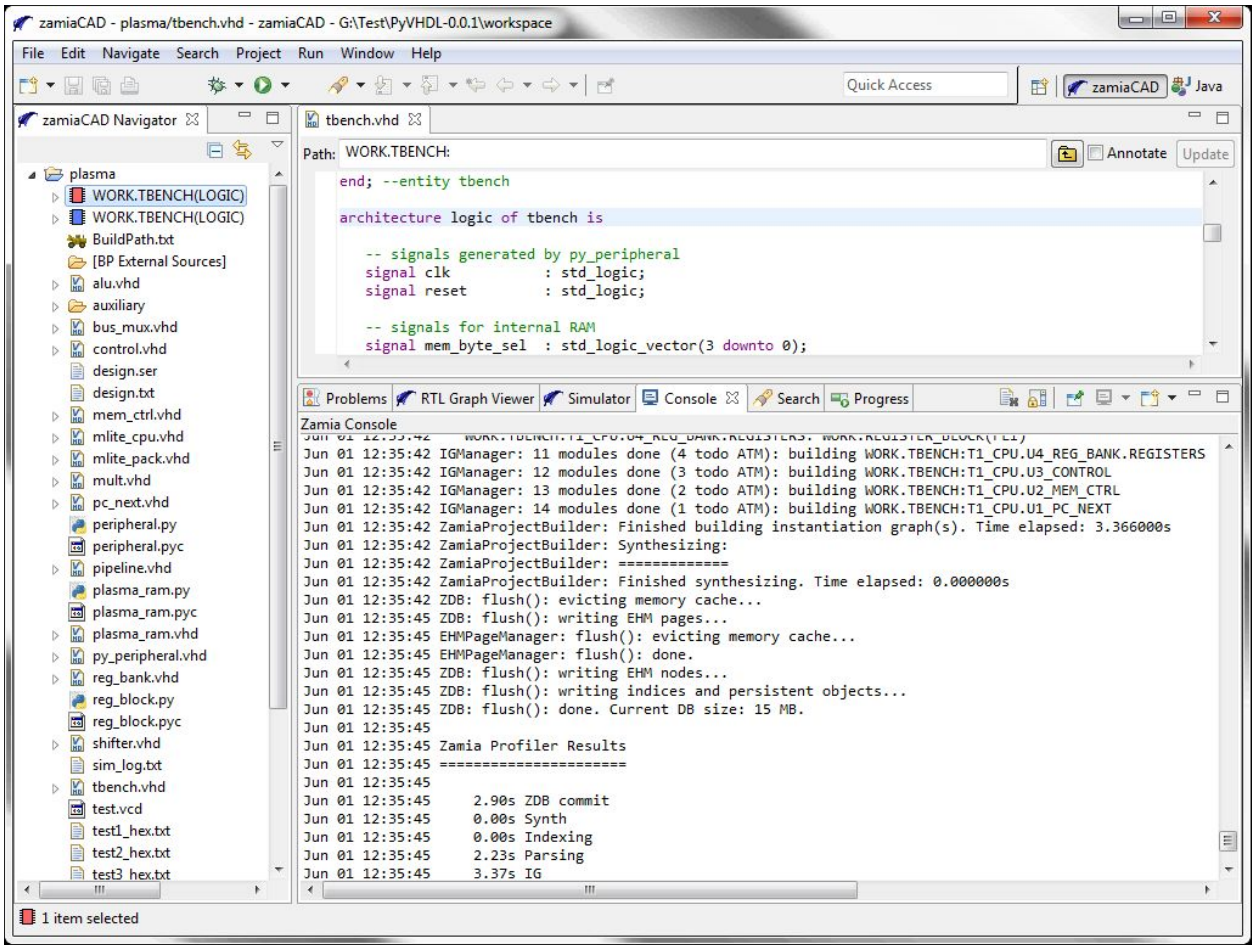
[✓] .project
[✓] alu.vhd
[✓] BuildPath.txt
[✓] bus_mux.vhd
[✓] control.vhd
[✓] design.ser
[✓] design.txt

Into folder: plasma

[ ] Overwrite existing resources without warning

Fig. 2.6: Completed Import Dialog

Click on the **Finish** button. In the Question dialogs that appears, click the **Yes** button. In the **Do Full Build?** dialog, click **Yes**. zamiaCAD will analyze and elaborate the plasma design. A log of that process will be in the **Zamia Console** window. Click on the plasma project folder. The project navigation information will appear below the folder. The red and blue rectangular icons can be opened to view the project design hierarchy. The files that make up the project are listed below the icons. You are now ready to setup and run a PyVHDL simulation.



```
end; --entity tbench

architecture logic of tbench is

   -- signals generated by py_peripheral
   signal clk          : std_logic;
   signal reset        : std_logic;

   -- signals for internal RAM
   signal mem_byte_sel  : std_logic_vector(3 downto 0);
```

```
Jun 01 12:35:42 IGManager: 11 modules done (4 todo ATM): building WORK.TBENCH:T1_CPU.U4_REG_BANK.REGISTERS
Jun 01 12:35:42 IGManager: 12 modules done (3 todo ATM): building WORK.TBENCH:T1_CPU.U3_CONTROL
Jun 01 12:35:42 IGManager: 13 modules done (2 todo ATM): building WORK.TBENCH:T1_CPU.U2_MEM_CTRL
Jun 01 12:35:42 IGManager: 14 modules done (1 todo ATM): building WORK.TBENCH:T1_CPU.U1_PC_NEXT
Jun 01 12:35:42 ZamiaProjectBuilder: Finished building instantiation graph(s). Time elapsed: 3.366000s
Jun 01 12:35:42 ZamiaProjectBuilder: Synthesizing:
Jun 01 12:35:42 ZamiaProjectBuilder: ============
Jun 01 12:35:42 ZamiaProjectBuilder: Finished synthesizing. Time elapsed: 0.000000s
Jun 01 12:35:42 ZDB: flush(): evicting memory cache...
Jun 01 12:35:45 ZDB: flush(): writing EHM pages...
Jun 01 12:35:45 EHMPageManager: flush(): evicting memory cache...
Jun 01 12:35:45 EHMPageManager: flush(): done.
Jun 01 12:35:45 ZDB: flush(): writing EHM nodes...
Jun 01 12:35:45 ZDB: flush(): writing indices and persistent objects...
Jun 01 12:35:45 ZDB: flush(): done. Current DB size: 15 MB.
Jun 01 12:35:45
Jun 01 12:35:45 Zamia Profiler Results
Jun 01 12:35:45 ======================
Jun 01 12:35:45
Jun 01 12:35:45     2.90s ZDB commit
Jun 01 12:35:45     0.00s Synth
Jun 01 12:35:45     0.00s Indexing
Jun 01 12:35:45     2.23s Parsing
Jun 01 12:35:45     3.37s IG
```

Fig. 2.7: The IDE After Plasma Project is Built

Configure a simulation

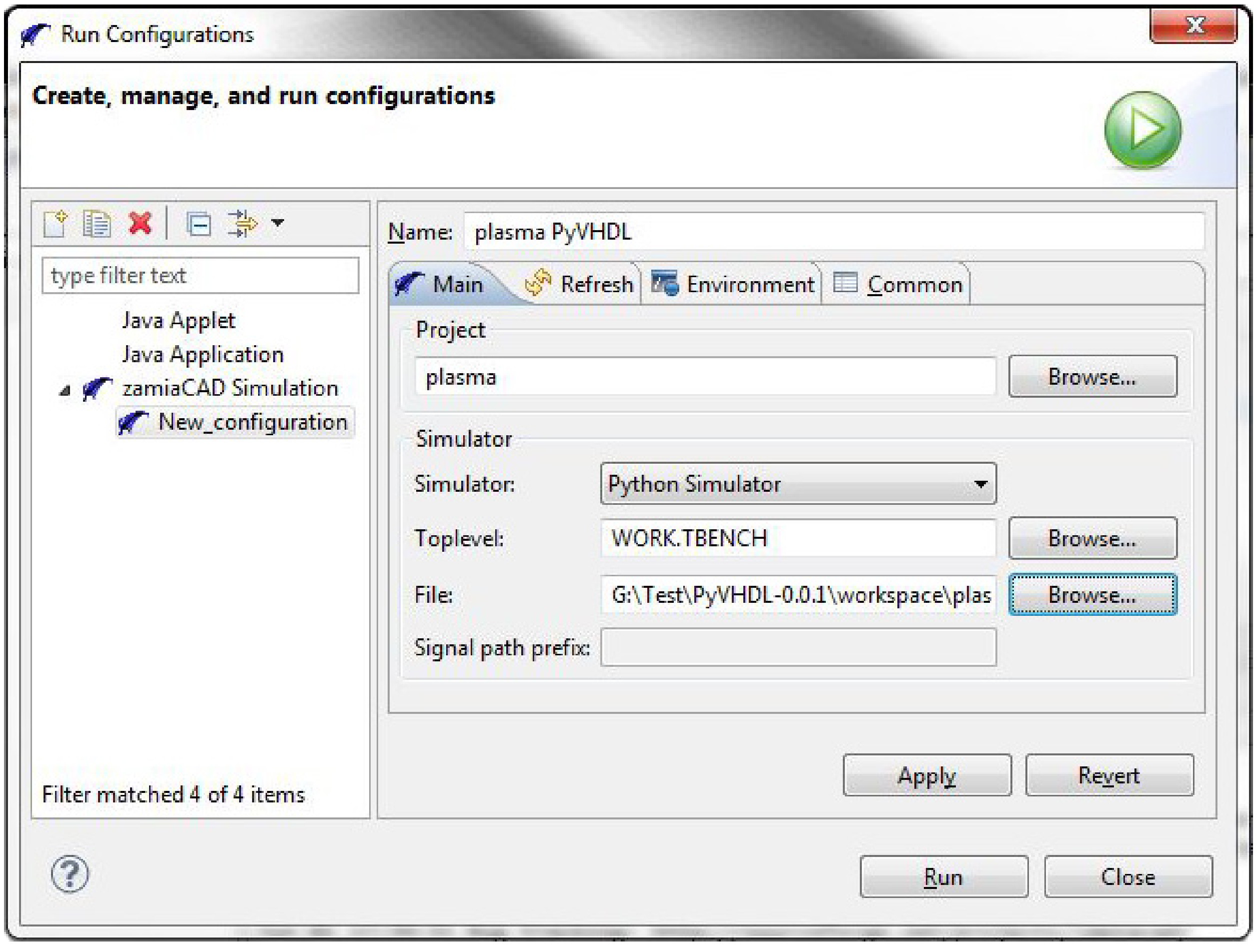
Make sure the plasma project is highlighted. On the IDE menu click **Run > Run Configurations...**. The **Run Configurations** dialog appears. Right click on **zamiaCAD Simulation**, and select **New**.

The right side of the dialog now displays the settings for the new configuration.

- In the **Name:** text box type `plasma PyVHDL`.
- Click the **Browse...** button next to the **Project** text box. Select the plasma project folder, and click **OK**.
- Click on the **Simulator:** dropdown, and select **Python Simulator**.
- Click on the **Browse...** button next to the **Toplevel:** textbox. **TOPLEVEL WORK.TESTBENCH** should be highlighted. Click **OK**.

- Click the **Browse...** button next to the **FIle:** textbox. Navigate to the **PyVHDL-0.0.1\workspace\plasma** folder. Select the **test.vcd** file. Click the **Open** button.
- The **Signal path prefix:** textbox should be empty.

Check that the fields in the **Run Configurations** dialog match the figure below:



Name: plasma PyVHDL

Project: plasma

Simulator: Python Simulator

Toplevel: WORK.TBENCH

File: G:\Test\PyVHDL-0.0.1\workspace\plas

Signal path prefix:

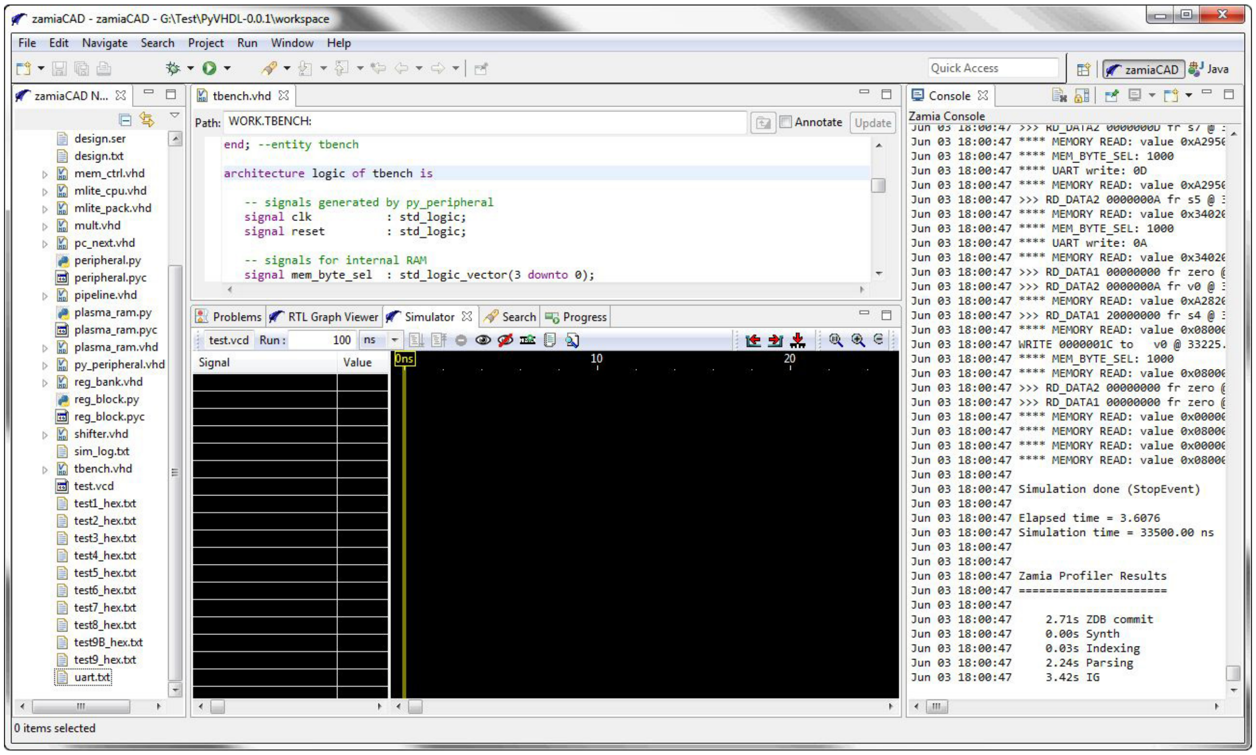
Fig. 2.8: Completed Run Configurations Dialog

Run the simulation

Click the **Run** button. If a **Do Full Build ?** dialog appears, click **Yes**. Open the **Zamia Console** window if it is not open. The text at the bottom of the console window will look similar to this, without the coloring:

```
Jun 03 18:00:47 >>> RD_DATA1 00000000 fr zero @ 33325.00
Jun 03 18:00:47 **** MEMORY READ: value 0x00000000, address 0x000003A8 @ 33325.00
Jun 03 18:00:47 **** MEMORY READ: value 0x080000E9, address 0x000003A4 @ 33375.00
Jun 03 18:00:47 **** MEMORY READ: value 0x00000000, address 0x000003A8 @ 33425.00
Jun 03 18:00:47 **** MEMORY READ: value 0x080000E9, address 0x000003A4 @ 33475.00
Jun 03 18:00:47
Jun 03 18:00:47 Simulation done (StopEvent)
Jun 03 18:00:47
Jun 03 18:00:47 Elapsed time = 2.8060
Jun 03 18:00:47 Simulation time = 33500.00 ns
Jun 03 18:00:47
Jun 03 18:00:47
Jun 03 18:00:47 Zamia Profiler Results
Jun 03 18:00:47 ======================
Jun 03 18:00:47
Jun 03 18:00:47     2.71s ZDB commit
Jun 03 18:00:47     0.00s Synth
Jun 03 18:00:47     0.03s Indexing
Jun 03 18:00:47     2.24s Parsing
Jun 03 18:00:47     3.42s IG
```

Make sure the **Simulator** window is visible. The IDE will look similar to this:



```
end; --entity tbench

architecture logic of tbench is

   -- signals generated by py_peripheral
   signal clk            : std_logic;
   signal reset          : std_logic;

   -- signals for internal RAM
   signal mem_byte_sel  : std_logic_vector(3 downto 0);
```

```
Jun 03 18:00:47 >>> RD_DATA2 0000000D fr s7 @ :
Jun 03 18:00:47 **** MEMORY READ: value 0xA2950
Jun 03 18:00:47 **** MEM_BYTE_SEL: 1000
Jun 03 18:00:47 **** UART write: 0D
Jun 03 18:00:47 **** MEMORY READ: value 0xA2950
Jun 03 18:00:47 >>> RD_DATA2 0000000A fr s5 @ 3
Jun 03 18:00:47 **** MEMORY READ: value 0x34020
Jun 03 18:00:47 **** MEM_BYTE_SEL: 1000
Jun 03 18:00:47 **** UART write: 0A
Jun 03 18:00:47 **** MEMORY READ: value 0x34020
Jun 03 18:00:47 >>> RD_DATA1 00000000 fr zero @
Jun 03 18:00:47 >>> RD_DATA2 0000000A fr v0 @ 3
Jun 03 18:00:47 **** MEMORY READ: value 0xA2820
Jun 03 18:00:47 >>> RD_DATA1 20000000 fr s4 @ 3
Jun 03 18:00:47 **** MEMORY READ: value 0x08000
Jun 03 18:00:47 WRITE 0000001C to   v0 @ 33225.
Jun 03 18:00:47 **** MEM_BYTE_SEL: 1000
Jun 03 18:00:47 **** MEMORY READ: value 0x08000
Jun 03 18:00:47 >>> RD_DATA2 00000000 fr zero @
Jun 03 18:00:47 >>> RD_DATA1 00000000 fr zero @
Jun 03 18:00:47 **** MEMORY READ: value 0x00000
Jun 03 18:00:47 **** MEMORY READ: value 0x08000
Jun 03 18:00:47 **** MEMORY READ: value 0x00000
Jun 03 18:00:47 **** MEMORY READ: value 0x08000
Jun 03 18:00:47
Jun 03 18:00:47 Simulation done (StopEvent)
Jun 03 18:00:47
Jun 03 18:00:47 Elapsed time = 3.6076
Jun 03 18:00:47 Simulation time = 33500.00 ns
Jun 03 18:00:47
Jun 03 18:00:47
Jun 03 18:00:47 Zamia Profiler Results
Jun 03 18:00:47 ======================
Jun 03 18:00:47
Jun 03 18:00:47     2.71s ZDB commit
Jun 03 18:00:47     0.00s Synth
Jun 03 18:00:47     0.03s Indexing
Jun 03 18:00:47     2.24s Parsing
Jun 03 18:00:47     3.42s IG
```

Fig. 2.9: IDE After Simulation Run

To select the signal waveforms to be displayed in the **Simulator** window, click on the **Eye** icon on the **Simulator** menu. The **Select signals to trace** dialog appears. In the **Select signals to be traced** textbox enter *. A list of signals will appear. Select all the signals. Click **OK**. The names and waveforms of the signals will now appear:

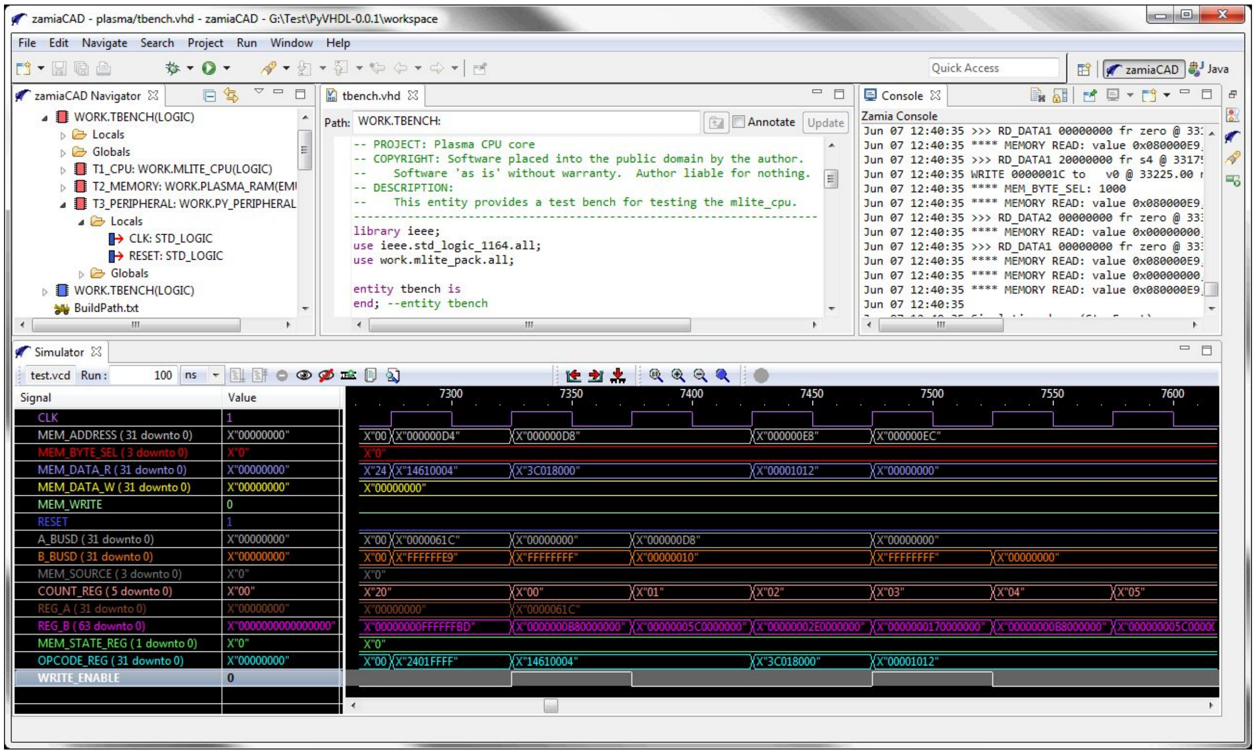
The IDE windows can be resized to show more of the waveforms. You can click on the **Simulator** window magnifying glass menu icons to zoom in, zoom out, or zoom full.

This completes the zamiaCAD+PyVHDL Quickstart tutorial. Read this zamiaCAD tutorial to learn more about the very useful features of the IDE.

2.2 Write a Python testbench

This part of the documentation needs more work! For now, look at the Python files in the **plasma** project. Briefly:

- Setup for the simulation is in the file: `PyVHDL-0.0.1\workspace\plasma\peripheral.py`



```
-- PROJECT: Plasma CPU core
-- COPYRIGHT: Software placed into the public domain by the author.
--    Software 'as is' without warranty.  Author liable for nothing.
-- DESCRIPTION:
--    This entity provides a test bench for testing the mlite_cpu.
---------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.mlite_pack.all;

entity tbench is
end; --entity tbench
```

```
Jun 07 12:40:35 >>> RD_DATA1 00000000 fr zero @ 33
Jun 07 12:40:35 **** MEMORY READ: value 0x080000E9
Jun 07 12:40:35 >>> RD_DATA1 20000000 fr s4 @ 3317
Jun 07 12:40:35 WRITE 0000001C to   v0 @ 33225.00 
Jun 07 12:40:35 **** MEM_BYTE_SEL: 1000
Jun 07 12:40:35 **** MEMORY READ: value 0x080000E9
Jun 07 12:40:35 >>> RD_DATA2 00000000 fr zero @ 33
Jun 07 12:40:35 **** MEMORY READ: value 0x00000000
Jun 07 12:40:35 >>> RD_DATA1 00000000 fr zero @ 33
Jun 07 12:40:35 **** MEMORY READ: value 0x080000E9
Jun 07 12:40:35 **** MEMORY READ: value 0x00000000
Jun 07 12:40:35 **** MEMORY READ: value 0x080000E9
Jun 07 12:40:35
```

Signal	Value
CLK	1
MEM_ADDRESS (31 downto 0)	X"00000000"
MEM_BYTE_SEL (3 downto 0)	X"0"
MEM_DATA_R (31 downto 0)	X"00000000"
MEM_DATA_W (31 downto 0)	X"00000000"
MEM_WRITE	0
RESET	1
A_BUSD (31 downto 0)	X"00000000"
B_BUSD (31 downto 0)	X"00000000"
MEM_SOURCE (3 downto 0)	X"0"
COUNT_REG (5 downto 0)	X"00"
REG_A (31 downto 0)	X"00000000"
REG_B (63 downto 0)	X"0000000000000000"
MEM_STATE_REG (1 downto 0)	X"0"
OPCODE_REG (31 downto 0)	X"00000000"
WRITE_ENABLE	0

Fig. 2.10: IDE Displaying Waveforms

- The plasma registers are implemented in the file: `PyVHDL-0.0.1\workspace\plasma\reg_block.py`. Note the print statements which print register activity messages to the zamiaCAD console.
- 8K of memory for the plasma CPU, and a UART data register are implemented in the file: `PyVHDL-0.0.1\workspace\plasma\plasma_ram.py`. Note how the `initialize(self)` method is used to set up the contents of the memory from a file.

The **Process** class implements a **concurrent** process. The class has 2 important methods:

- **initialize(self)** is called before the simulation starts to do any setup that is needed.
- **run(self, P, S)** is called to start the process running. The `while true:` loop causes the process to continuously run through its code, yielding to other processes when a wait is encountered. The **P** argument is a class whose attributes are the ports defined in the VHDL entity declaration. Signals are accessed as **P.signal_name**. The **signal_name** must be in **UPPER CASE**. the **S** argument is a list containing references to all the signals in the design (*avoid using this for now*).

Each Python file that implements a VHDL architecture has a **setup(tb, P)** function that adds the processes defined in the file to the simulation execution environment. The **tb** argument is a reference to the simulation execution environment. The **P** argument is a string identifying the process.

2.3 License

PyVHDL is an open source project. This means that it is not only available for download free of charge, but users have access to the source code and may contribute to the project.

PyVHDL is copyright 2016 by Vern Muhr and is covered by the GNU General Public License Version 3.

Here is the ZamiaCAD project license.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search