Non-Monitoring Tracing on LEON3 Platform

Author: Zhengyang Liu

Under the Guidance of: Dr. Minjun Seo, Prof. Fadi Kurdahi

目录

0. About this documentation

In this documentation, I will demonstrate my work on LEON3 and its non-intrusive tracing.

In the part of this documentation, I will introduce my work in three main parts:

- 1. Introduction
- 2. Non-Intrusive Tracing Design
- 3.Adding Pcore to LEON3 Library

In part "Introduction", I will briefly introduce LEON3 Platform. For more details, you could refer to the user manuals listed in this documentation.

In part Non-Intrusive Tracing Design, I will demonstrate how to

- Implementing the LEON3 Platform on FPGA Board
- Extending the Trace Interface
- Building a Pcore to Receive the Trace Data

In part "Adding Pcore to LEON3 Library", I will demonstrate how to modify its metadata and add your pcore to the LEON3 library. **If you already have finished designing your pcore and have realized LEON3 on your FPGA board, you can directly jump to this part for the details about the adding pcore to LEON3's library.** It is recommended that following the steps in this documentation first when you try to realize LEON3 on your FPGA board, because based on my experience, I may could help you to save your time.

1. Introduction

1.1 The LEON3 Platform

For the details about the LEON3 microprocessor, you could refer to the user manuals listed below:

- guide.pdf
- grip.pdf
- grlib.pdf

You can download these pdfs on https://www.gaisler.com/index.php/downloads/leongrlib

These user manuals well introduce the LEON3 Platform, configuration steps and all the IPs provided by the Cobham Gaisler and other vendors. For example, the grlib.pdf "Installation" and "LEON/GRLIB quick-start guide" parts introduce how to install GRLIB on your PC and then implement LEON3 on FPGA boards.

You may find these pdfs very useful when you try to realize LEON3 on your FPGA board and add your pcore into LEON3 Platform.

Here I will only introduce some basic and important features about the LEON3.

LEON3 is a 32-bit microprocessor based on SPARC V8 instruction set. It has a 7-stage integer pipeline, an internal instruction trace buffer. It has AMBA AHB and APB bus, with an AHB2APB bridge inside. It also supports the multi-buses structure.

Below is the block diagram of LEON3 microprocessor core.

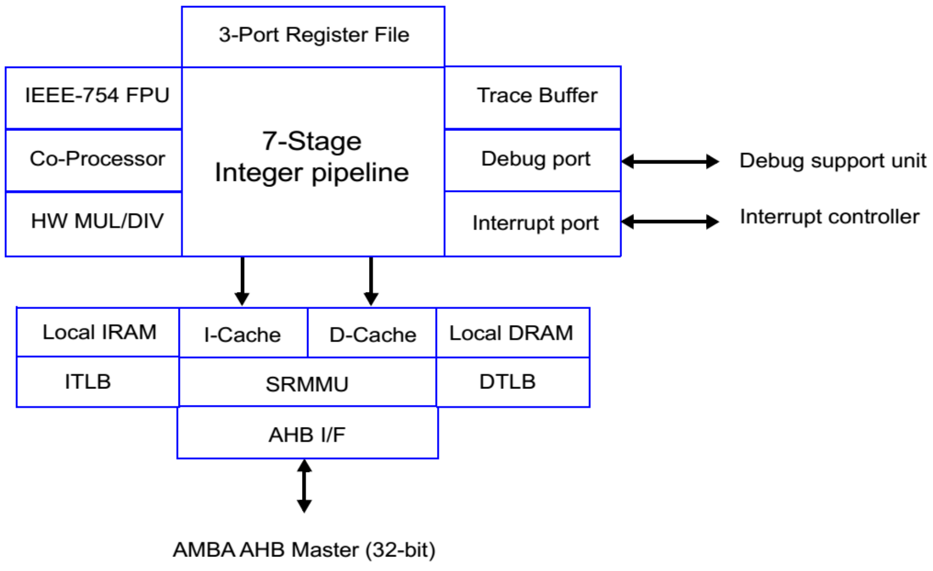


Fig. 1 LEON3 processor core block diagram

For LEON3 Platform level (system level), a template design could have the structure shown in the figure below.

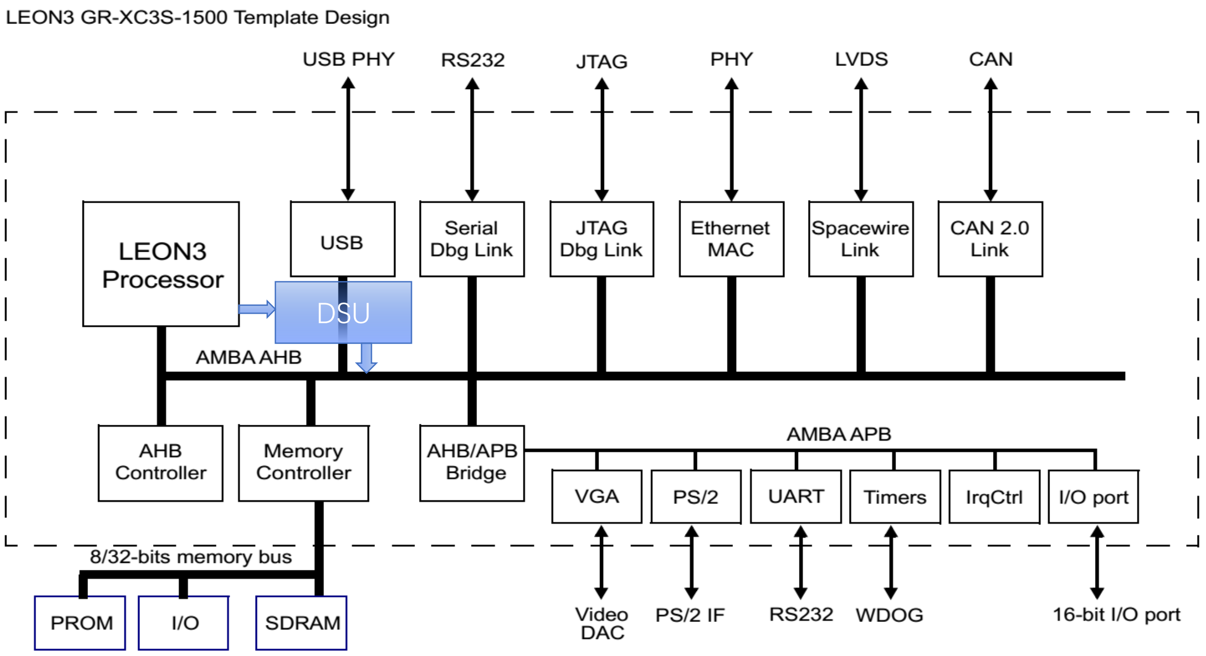


Fig. 2 LEON3 template design

The LEON3 has an AHB bus interface. On AHB bus, and there are several peripherals such as JTAG, Ethernet MAC, Memory Controller and so on. There is also an AHB/APB Bridge connecting the APB bus. There is also a Debug Support Unit (DSU) connecting with LEON3's debug port and AHB bus.

2. Non-Intrusive Tracing Design

2.1 Implementing the LEON3 Platform on FPGA Board

2.1.1 Introduction

This part will demonstrate how to setup the developing environment for LEON3 and finally realize LEON3 platform on your FPGA board.

Files and tools that needed are:

- **Cygwin**, which enable you to use Linux commands on Windows. It is needed for using many commands in the command-line to control and realizing the LOEN3.
  Downloading: setup-x86_64.exe
  Its website: http://cygwin.com/install.html

- **GRLIB**, which contains the LEON3 source code and its IPs.
  Downloading: grlib-gpl-2018.1-b4217.tar.gz
  Its website: https://www.gaisler.com/index.php/downloads/leongrlib

- **GRTools**, which contains BCC, the bare C cross-compiler for LEON3, Grmon, the debugger for LEON3, Eclipse and other sofwares.
  Downloading: GRTools-20180426.exe
  Its website: https://www.gaisler.com/index.php/downloads/grtools

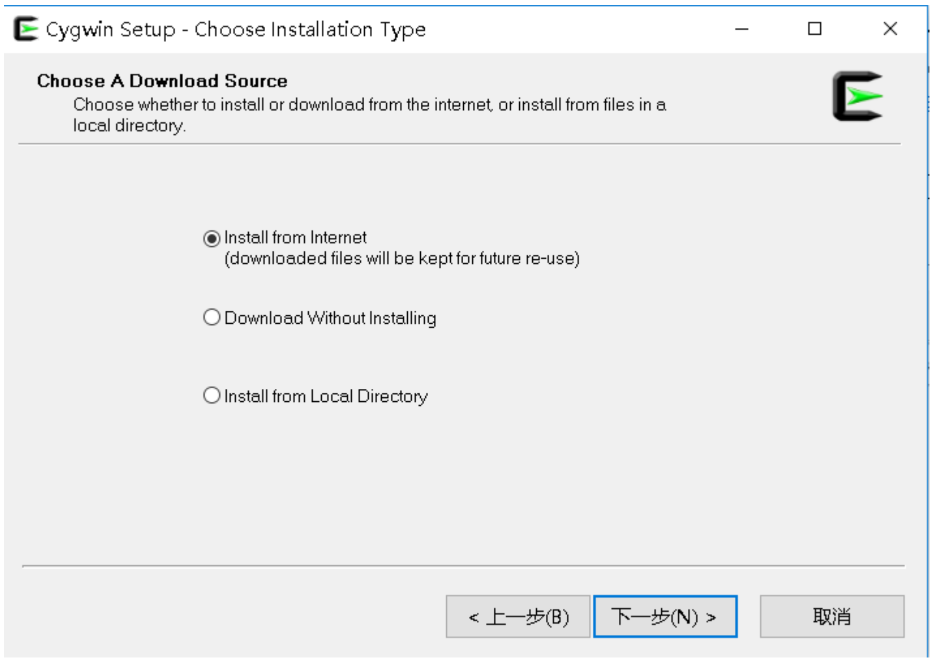
- **FPGA design tools**, for example, Vivado 2018.2. If you are a student and cannot afford the license (Yeah it is super expensive…), you can choose 30-day trial license.

- **Nexys Video Board Files,** if you are using the same FPGA board and same design tool with me (i.e., Vivado), you need to download the board file from the following link. Then follow the instruction given by Digilent to setup these board files. The downloading link is also included in the instructions website. You can also google "Digilent Nexys Video" and enter "Resource Center" on the right side of this webpage.
  Downloading: archive
  Instrucitons: https://reference.digilentinc.com/vivado/installing-vivado/start

2.1.2 Installation of Cygwin

The installation program of Cygwin is not well designed and it is a little user-unfriendly. Being patient in this step…

Run the installing program. Follow the steps in it. For the “Choose A Download Source” page, if you are the first time to install Cygwin, choose “Install from Internet”.



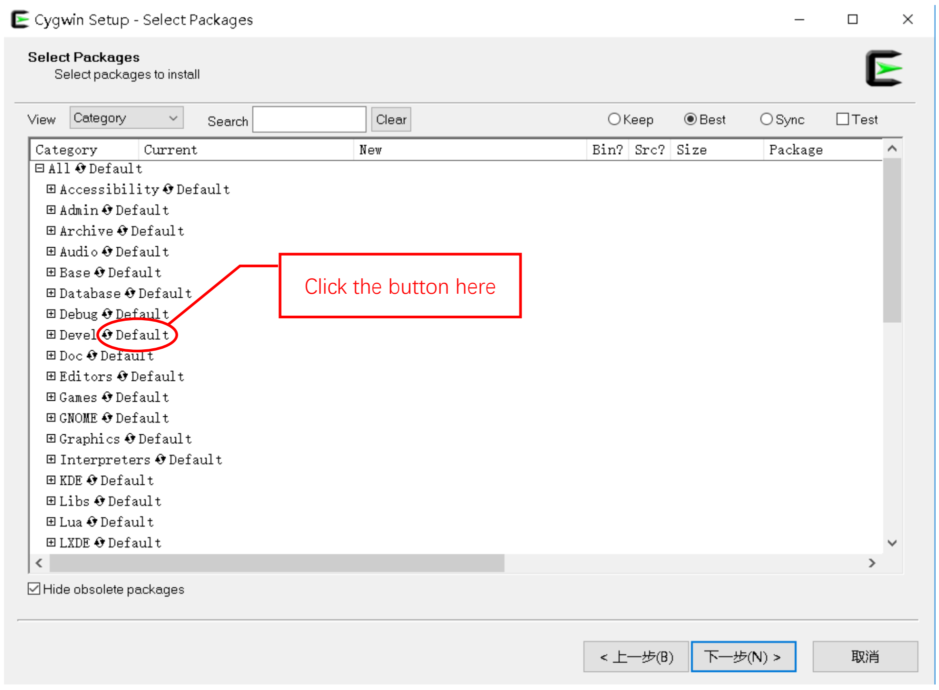
Cygwin Setup - Choose Installation Type

Choose A Download Source
Choose whether to install or download from the internet, or install from files in a local directory.

[✓ Install from Internet
(downloaded files will be kept for future re-use)]
Download Without Installing
Install from Local Directory

Fig. 3 Choose a download source

To install the Cygwin, you need run the install program, selecting the necessary packages. In the “Select Packages” page of the installing program, you can click the button near the category name.



Cygwin Setup - Select Packages

Select Packages
Select packages to install

View: Category   Search: [ ]   Keep [✓ Best] Sync Test

```
Category          Current     New     Bin?  Src?  Size   Package
⊟ All ♻ Default
  ⊞ Accessibility ♻ Default
  ⊞ Admin ♻ Default
  ⊞ Archive ♻ Default
  ⊞ Audio ♻ Default
  ⊞ Base ♻ Default
  ⊞ Database ♻ Default
  ⊞ Debug ♻ Default
  ⊞ Devel ♻ Default
  ⊞ Doc ♻ Default
  ⊞ Editors ♻ Default
  ⊞ Games ♻ Default
  ⊞ GNOME ♻ Default
  ⊞ Graphics ♻ Default
  ⊞ Interpreters ♻ Default
  ⊞ KDE ♻ Default
  ⊞ Libs ♻ Default
  ⊞ Lua ♻ Default
  ⊞ LXDE ♻ Default
```

[✓ Hide obsolete packages]

Fig. 4 Install all packages

The packages needed: all packages in the categories **Archive, Base, Devel, Database, Web and X11**. Because I am not very familiar with every package and its functions, in case of after installing the Cygwin cannot execute some commands, I also installed all packages in the **Editor, Libs and Utils**.

Make sure you have enough hard disk space if you want to install all packages (although this is the safest way). It is said it requires about **70GB** to install all packages.

At the last step of the installing, the install program will look like it is stuck, stopping and doing nothing for a long time. At this time, don't close the installer, you just need to patiently give its time, waiting for about several hours to let it finish the last step of installing…

After waiting for enough time, the Cygwin will finish its installation. Then you can run the Cygwin terminal and issue the command

- cygcheck -c cygwin
- gcc --version
- g++ --version
- make --version
- gdb --version

to check whether the Cygwin is in good status. If it returns some information about the version number and license, it means that your Cygwin is running well now.

2.1.3 Installation of GRLIB

After downloading the GRLIB, you will get a ".tar.gz" file. Follow the instruction in the "2.1 Installation" part of the grlib.pdf to finish the installation of GRLIB. Basically, copying this .tar.gz file pack into desired directory and the issuing the command

- gunzip -c grlib-gpl-2018.1-b4217.tar.gz | tar xf –

or

- tar xvf grlib-gpl-2018.1-b4217.tar.gz

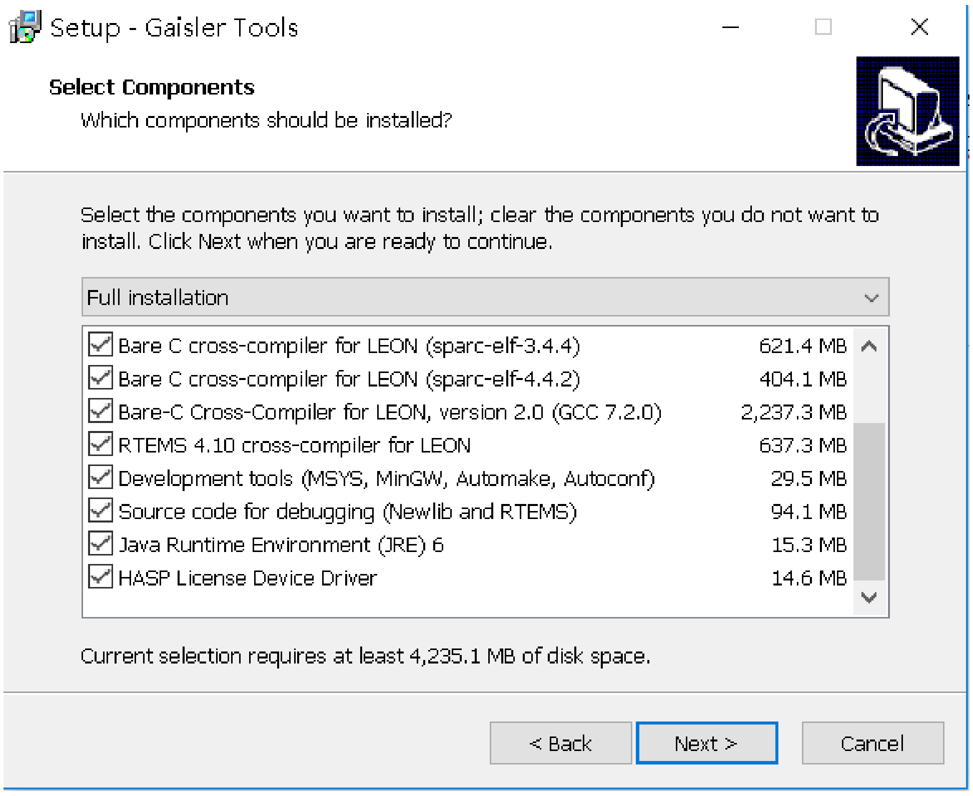
in Cygwin to unzip the file pack. After unzipping it, you will finish the installation of GRLIB. It simple and easy to do this.

Do NOT use WinZip on the .tar.gz file, this will corrupt the files during extraction. If you meet any unexpected troubles or for further details, you could refer to "2.1 Installation" in grlib.pdf. This part also shows the structure of the GRLIB, demonstrating its hierarchy and what kind of files in each subfolder. It is recommended that you should have a look at this part in the user manual.

2.1.4 Installation of GRTools

Run the downloaded GRTools installation program and start the installation. It is strongly recommended that use the default destination location (i.e., the folder where GRTools to be installed).

Then in the next page, you can select all the tools including Eclipse, GRMON debugger, RTEMS, in case that you will need some other software in it for your further purpose.



Setup - Gaisler Tools

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

✓ Bare C cross-compiler for LEON (sparc-elf-3.4.4)	621.4 MB
✓ Bare C cross-compiler for LEON (sparc-elf-4.4.2)	404.1 MB
✓ Bare-C Cross-Compiler for LEON, version 2.0 (GCC 7.2.0)	2,237.3 MB
✓ RTEMS 4.10 cross-compiler for LEON	637.3 MB
✓ Development tools (MSYS, MinGW, Automake, Autoconf)	29.5 MB
✓ Source code for debugging (Newlib and RTEMS)	94.1 MB
✓ Java Runtime Environment (JRE) 6	15.3 MB
✓ HASP License Device Driver	14.6 MB

Current selection requires at least 4,235.1 MB of disk space.

Fig. 5 You can select all these tools here

Then click "Next", you don't need to check the first block "Install GRMON PCI driver" here. You could check the later two blocks. Then you could click "Next" and "Install" and let it start the installation. During the installation, this installer will automatically set the environment variables.

Also, during the installation, it will pop up a window to help you to install the Minimal System. Just simply following it steps to start this installation process.

During the installation of Minimal System, it will pop up a cmd command-line window. Firstly, this popped up will ask you whether you wish to normalize between your MinGW install if any as well as your previous MSYS installs. Type "y" with your keyboard then press "enter" to continue. The cmd window will pause again to ask you whether you have installed MinGW before. Typically, you should type “n” then press enter if you haven’t installed the MinGW.

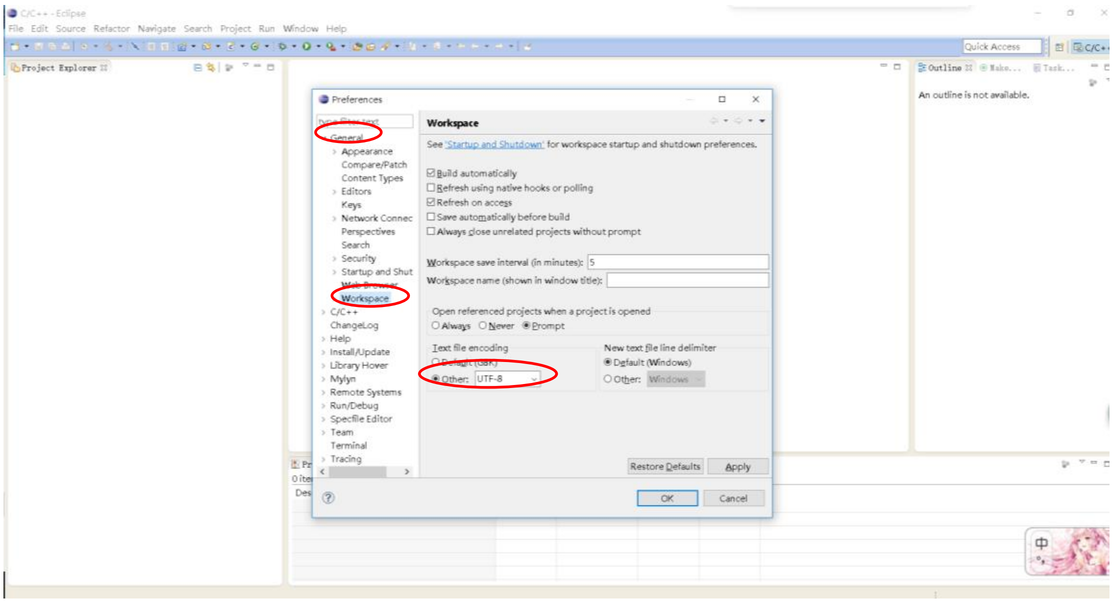
This cmd window will still pause for several time, just press enter and continue. At last it will finish the installation of Minimal System and automatically quit.

Then you will be led back to the GRTools installation window. Click “Finish” to finally finish the whole installation process.

Now, if you want to use Eclipse for the following software developing on LEON3, you need to set the workplace path of the Eclipse. Launching the Eclipse, it will guide you to select a folder as your workplace. Choosing a folder that you want. Try to avoid any space and other non-English characters. For my case, there are some compatibility problems between Eclipse and GRMON, I used Code::blocks and GRMON separately for writing and downloading software to the LEON3.

Note: you can read the grmon.pdf to understand how to use GRMON.

Note: If you want to use these tools—especially the Eclipse in your mother language rather than English (e.g., in Simplified Chinese, Japanese or Korean)—you may need to configure the “Text file encoding” in the Eclipse: launching Eclipse, then Window → Preferences→ General→ Workspace→ Text file encoding. Choose “other”, and set it as “UTF-8”.



Workspace

See 'Startup and Shutdown' for workspace startup and shutdown preferences.

- [✓] Build automatically
- [ ] Refresh using native hooks or polling
- [✓] Refresh on access
- [ ] Save automatically before build
- [ ] Always close unrelated projects without prompt

Workspace save interval (in minutes): 5

Workspace name (shown in window title):

Open referenced projects when a project is opened: Always Never [✓ Prompt]

Text file encoding: Default (GBK) [✓ Other: UTF-8]

New text file line delimiter: [✓ Default (Windows)] Other: Windows

Fig. 6 Configure Text file encoding

After finishing the step above, you should add the Athena to your design tool, for example, Vivado, to the “PATH” system environment variable. You can do this by right click the “This Computer” on your desktop then choose “properties” if you are using Windows. You should also add the path of Cygwin (…\Cygwin\bin) to the “PATH” system environment variable. For the details, you can refer to “2.4.2 Windows with Cygwin” in grlib.pdf

2.1.5 Implementing the LEON3 Platform

Now you have finished the building of the developing environment of LEON3. You can start to use your FPGA design tool to run the design flow (Synthesis, Implementation and generating the bitstream file).

In Cygwin, use command “cd” and enter the directory

- …\grlib-gpl-2018.1-b4217\designs\leon3-your-FPGA-boardname

For example, my FPGA board is Digilent Nexys Video. So, I need to go into the directory

- …\grlib-gpl-2018.1-b4217\designs\leon3-digilent-nexys-video

After you going into this directory, **you should firstly read README.txt** which is in this directory. It demonstrates some very important information about how to implement the LEON3 platform.

According the README.txt, you should firstly issue the command if you are using Xilinx Vivado Design Tool

- export XILINX_VIVADO

to let the mig_7series target work correctly.

If you want to configure the LEON3 platform as well as its bus peripherals you can use XGrlib tool. For the details about the XGrlib, you can refer to the “4.8 XGrlib graphical implementation tool” part in grlib.pdf. In order to launch XGrlib configuration tool, you must issue the command

- Export DISPLAY=:0.0

To successfully launch the XGrlib. Without this command you will encounter a failure. Then click “xconfig” on the upper right of XGrlib window. You can configure the LEON3 platform here. After finishing the configuration with XGrlib, click “Save and Exit” button to exit “xconfig”. Then you will back to the main page of XGrlib.

Now, if this is the first time for you to deal with this design, straightly click button “scripts”. The tool will automatically generate the Vivad project file “leon3-your-FPGA-boardname.xpr”, which is located in path

- …\grlib-gpl-2018.1-b4217\designs\leon3-your-FPGA-boardname\vivado\leon3-your-FPGA-boardname and other necessary files. If you have done this before and generated the Vivado project file, you should firstly click “distclean” button to remove all generated files, then click “scripts” button to re-generate these again.

Note: all this should be done in correct directory mentioned above.

Note: DO NOT OPEN THE ISE PROJECT DIRECTLY! When you finish “make scripts”, you will find an ISE project file (.xise file)**, DO NOT open it directly with your Vivado if you are using Vivado. There maybe some bugs when Vivado tries to convert ISE project into Vivado project, and then you will fail to synthesis and implementation.** Just use “make vivado-launch”, “make vivado” or directly double-click the Vivado project (.xpr file) when you have launched the Vivado with one of these two commands for one time.

Now you should have got the Vivado project file, and you will notice that in your current directory there are many newly generated files. Back to Cygwin command window, issue

- make vivado-launch

to launch the GUI of vivado. If you prefer the command-line mode (non-GUI), just issue command

- make vivado

then the Vivado will launch. You can run the design flow to generate the bitstream files and program your device!

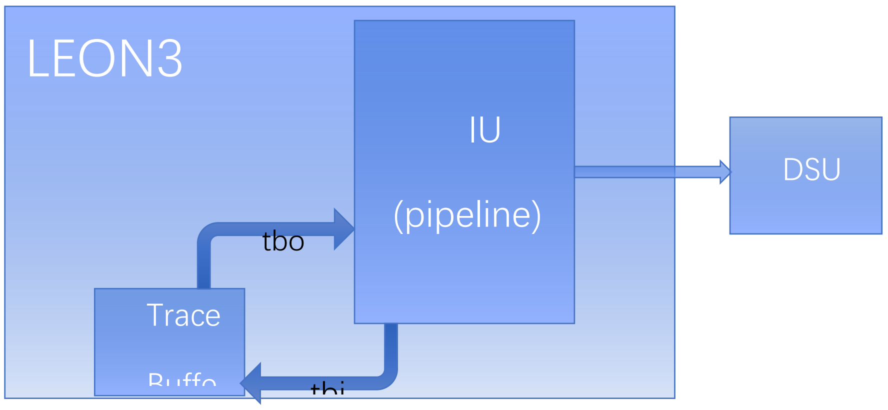
If you don’t want to add your custom hardware (pcore) into LEON3 platform, you could directly run the design flow in your design tool. But if you already designed your pcore and you want to add your pcore into LEON3 platform, you can jump to the part “Adding Your Pcore into LEON3 Platform” in this documentation.

After generating the bitstream file, you can connect your board to your PC and program it.

2.2 Extending the Trace Interface

For some details about this part you could also refer to the corresponding part in slide “Non-Intrusive Tracing on LEON3 Platform.pptx”.

Originally, the trace information is sent out from the integer unit (IU) to trace buffer and stored in LEON3’s own instruction trace buffer which is located inside the LEON3 core. The trace information is buffered, causing a big time-delay until we could see them. So, it is an internal trace interface, which means that the trace will not be directly sent out.



For instantly sending out the trace information, we need to create an external trace interface that can directly send out the trace information. To do this, we acquire the structure shown in the picture below.

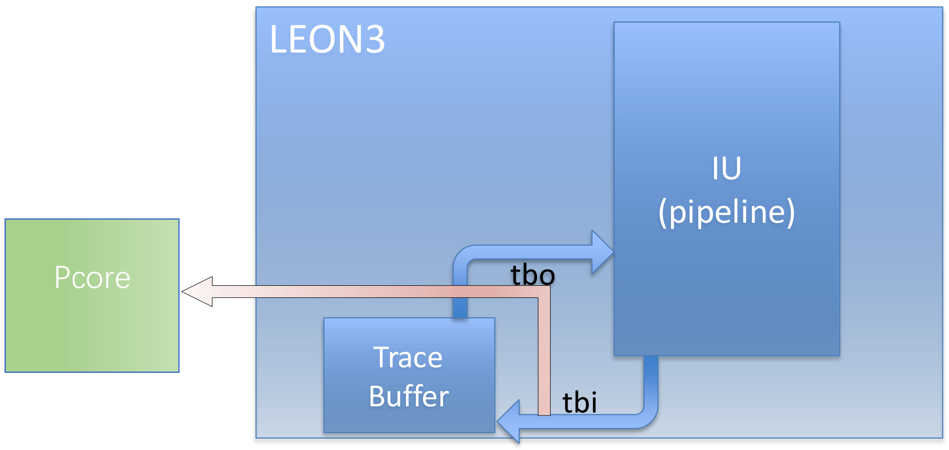


Fig. 8 External trace interface

We directly pull out the input bus of the trace buffer to the outside, creating a new output port on LEON3 core. Our pcore will then connect to this port to gather the trace information. Then trace data can directly go into the pcore. Further data storing, filtering and processing of these trace information will be done in the pcore.

The code below is the instruction trace buffer inside the LEON3 core. The file name is

- …/lib/gaisler/leon3v3/leon3x.vhd

```
-- instruction trace buffer memory
tbmem_gen : if (tbuf /= 0) generate
  tbmem_1p : if (tbuf <= 64) generate
    tbmem0 : tbufmem
      generic map (tech => MEMTECH_MOD, tbuf => tbuf, dwidth =>
      32, testen => scantest)
      port map (gclk2, tbi, tbo, ahbi.testin
            );
    tbo_2p <= tracebuf_2p_out_type_none;
  end generate;
  tbmem_2p: if (tbuf > 64) generate
    tbmem0 : tbufmem_2p
      generic map (tech => MEMTECH_MOD, tbuf => (tbuf-64), dwidth
      => 32, testen => scantest)
      port map (gclk2, tbi_2p, tbo_2p, ahbi.testin
            );
    tbo <= tracebuf_out_type_none;
  end generate;
end generate;
notbmem_gen : if (tbuf = 0) generate
  tbo <= tracebuf_out_type_none;
  tbo_2p <= tracebuf_2p_out_type_none;
end generate;
```

We will use the input "tbi" of the trace buffer, creating a new signal to the outside.

```
signal sig_instr_trace_out  :  std_logic_vector(255 downto 0) :=
        tbi.data;



begin
---------------connect to the new port------
   instr_trace_out_leon3x  <=  sig_instr_trace_out;
--------------------------------------------
```

Then we need a new port declaration of module(entity) leon3x:

```
  port (
    clk        : in  std_ulogic;              -- free-running clock
    gclk2      : in  std_ulogic;              -- gated 2x clock
    gfclk2     : in  std_ulogic;              -- gated 2x FPU clock
    clk2       : in  std_ulogic;              -- free-running 2x clock
    rstn       : in  std_ulogic;
    ahbi       : in  ahb_mst_in_type;
    ahbo       : out ahb_mst_out_type;
    ahbsi      : in  ahb_slv_in_type;
    ahbso      : in  ahb_slv_out_vector;
    irqi       : in  l3_irq_in_type;
    irqo       : out l3_irq_out_type;
    dbgi       : in  l3_debug_in_type;
    dbgo       : out l3_debug_out_type;
    fpui       : out grfpu_in_type;
    fpuo       : in  grfpu_out_type;
    clken      : in  std_ulogic;

    instr_trace_out_leon3x : out std_logic_vector(255 downto 0)
    );
```

Port **instr_trace_out_leon3x** is the new output port of the trace interface, pulling out from the input of instruction trace buffer.

I left out some long comment in the source code to keep this documentation a clear format. You can check this source file for more comment and information.

The entity leon3x is instantiated by entity leon3s. As a result, we need do a similar thing in leon3s. See the code part below. The code below is in the file

- …/lib/gaisler/leon3v3/leon3s.vhd

In the instantiating part, use the new port map of leon3x:

```
    port map (
      clk        => gnd,
      gclk2      => clk,
      gfclk2     => clk,
      clk2       => clk,
      rstn       => rstn,
      ahbi       => ahbi,
      ahbo       => ahbo,
      ahbsi      => ahbsi,
      ahbso      => ahbso,
      irqi       => irqi,
      irqo       => irqo,
      dbgi       => dbgi,
      dbgo       => dbgo,
      fpui       => open,
      fpuo       => fpuo,
      clken      => vcc,
-------------------
      instr_trace_out_leon3x  => instr_trace_out_leon3s
      );
```

Then add a new output port in entity leon3s:

```
 port (
   clk        : in  std_ulogic;
   rstn       : in  std_ulogic;
   ahbi       : in  ahb_mst_in_type;
   ahbo       : out ahb_mst_out_type;
   ahbsi      : in  ahb_slv_in_type;
   ahbso      : in  ahb_slv_out_vector;
   irqi       : in  l3_irq_in_type;
   irqo       : out l3_irq_out_type;
   dbgi       : in  l3_debug_in_type;
   dbgo       : out l3_debug_out_type;
   --new output port of the trace interface, connected to
   --instr_trace_out_leon3x
   instr_trace_out_leon3s  : out std_logic_vector(255 downto 0)
   );
```

Again, entity leon3s is instantiated in top file leon3mp.vhd, we need do the similar thing one more time. See code part below. File name:

- …/designs/leon3-your-FPGA-boardname/leon3mp.vhd

In signal declaration part, you can add signal

```
 signal sig_instr_trace_temp : std_logic_vector(255 downto 0);
```

Then in leon3s instantiating part in top file leon3mp.vhd, use the same new port map:

```
----------------------------------------------------------------------
---  LEON3 processor and DSU -----------------------------------------
----------------------------------------------------------------------

 -- LEON3 processor
 leon3gen : if CFG_LEON3 = 1 generate
   cpu : for i in 0 to CFG_NCPU-1 generate
     u0 : leon3s
       generic map (i, fabtech, memtech, CFG_NWIN, CFG_DSU, CFG_FPU,
                    CFG_V8,
                    0, CFG_MAC, pclow, CFG_NOTAG, CFG_NWP, CFG_ICEN,
                    CFG_IREPL, CFG_ISETS, CFG_ILINE,
                    CFG_ISETSZ, CFG_ILOCK, CFG_DCEN, CFG_DREPL,
                    CFG_DSETS, CFG_DLINE, CFG_DSETSZ,
                    CFG_DLOCK, CFG_DSNOOP, CFG_ILRAMEN, CFG_ILRAMSZ,
                    CFG_ILRAMADDR, CFG_DLRAMEN,
                    CFG_DLRAMSZ, CFG_DLRAMADDR, CFG_MMUEN,
                    CFG_ITLBNUM, CFG_DTLBNUM, CFG_TLB_TYPE,
                    CFG_TLB_REP, CFG_LDDEL, disas, CFG_ITBSZ,
                    CFG_PWD, CFG_SVT, CFG_RSTADDR, CFG_NCPU-1,
                    CFG_DFIXED, CFG_SCAN, CFG_MMU_PAGE, CFG_BP,
                    CFG_NP_ASI, CFG_WRPSR, CFG_REX, CFG_ALTWIN)
      port map (clk   => clkm,
                rstn  => rstn,
                ahbi  => ahbmi,
                ahbo  => ahbmo(i),
                ahbsi => ahbsi,
                ahbso => ahbso,
                irqi  => irqi(i),
                irqo  => irqo(i),
                dbgi  => dbgi(i),
                dbgo  => dbgo(i),
                instr_trace_out_leon3s  => sig_instr_trace_temp   --
                );
    end generate;
```

This 256bits signal(sig_instr_trace_temp) will be later connected to the pcore. Then, you can connect to this signal in your pcore port map.

After modifying these source code, we still need to do one thing. That is, add these new ports into the component declaration .vhd files of these entities. This file is

- …\lib\gaisler\leon3\leon3.vhd

It declares different types of LEON3 and components. Add your new port into it:

```
  component leon3s
  generic (
    hindex    : integer                 := 0;
    fabtech   : integer range 0 to NTECH  := DEFFABTECH;
    memtech   : integer                   := DEFMEMTECH;

    .........many generics.........

    smp       : integer range 0 to 15  := 0;
    cached    : integer                  := 0;
    scantest  : integer                  := 0;
    mmupgsz   : integer range 0 to 5   := 0;
    bp        : integer                  := 1;
    npasi     : integer range 0 to 1  := 0;
    pwrpsr    : integer range 0 to 1  := 0;
    rex       : integer range 0 to 1  := 0;
    altwin    : integer range 0 to 1  := 0
  );
  port (
    clk    : in  std_ulogic;
    rstn   : in  std_ulogic;
    ahbi   : in  ahb_mst_in_type;
    ahbo   : out ahb_mst_out_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : in  ahb_slv_out_vector;
    irqi   : in  l3_irq_in_type;
    irqo   : out l3_irq_out_type;
    dbgi   : in  l3_debug_in_type;
    dbgo   : out l3_debug_out_type;
    --new output port of the trace interface, connected to
    --instr_trace_out_leon3x
    instr_trace_out_leon3s  : out std_logic_vector(255 downto 0)
  );
  end component;
```

Now we finished the extending of external trace interface.

You can also refer the schematic “trace_interface.pdf” about this external trace interface in this GitHub repository.

2.3 Building a Pcore to Receive the Trace Data

2.3.1 About the APB Bus Interface

I built several versions of pcores to realize different functionalities. They all have the APB bus interface, i.e., are all connected to the APB bus. As a result, they can exchange data through the bus. It is helpful because we can print out some text to the GRMON console through the debug link. We can also design some control registers inside our pcore. By set some values into these control registers via APB bus interface, we can let the pcore has different functionalities. For example, we can write two different addresses to two “address control registers” as the start address and end address, to let the pcore only store the trace information that is about the instructions within this address range. You could regard the debug link as a data path between the debug software GRMON on your PC and the LEON3 system. You can use some simple C functions, for example, “printf()”, to let LEON3 print some debugging information to the GRMON console through the debug link.

For further explanation, your desired information is generated in the LEON3 system, for example, your pcore stored some trace information. And your pcore is connected to the APB bus. This data will pass through the AHB/APB bridge and arrive at the AHB bus. At there, these data will finally go to your PC by debug link.

The GRMON concept could be shown as the figure below.

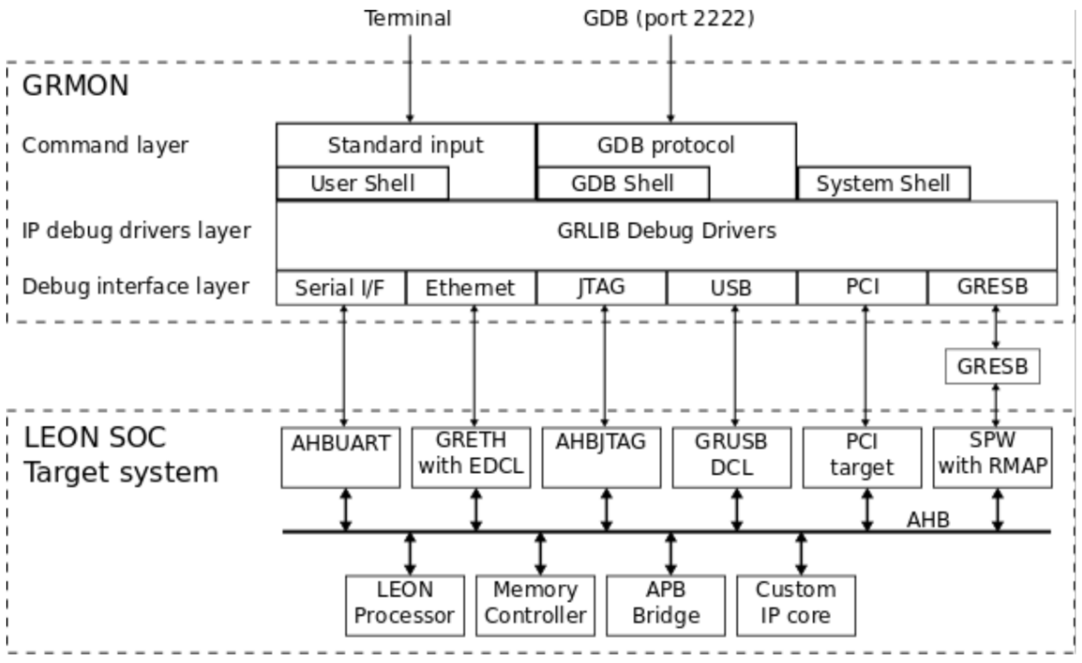


Fig. 9 GRMON concept overview

Every APB peripheral has a Plug and Play configuration information. For the details, it is strongly recommended that referring the part "5.4 AMBA APB on-chip bus" and part "5.5 APB plug&play configuration" in grlib.pdf. It has a detailed demonstration about how to connect your pcore to the APB bus.

2.3.2 The First Pcore

The first pcore is a simple hardware multiplier. It has three registers: two are writable, one is read only. And these three all 32-bit wide. Two for storing the operands and one for storing the result. This pcore is designed for testing whether the connection to APB bus is successful and whether the LEON3 platform is successfully realized.

A simplified block diagram of the structure of this pcore can be shown in the figure below.

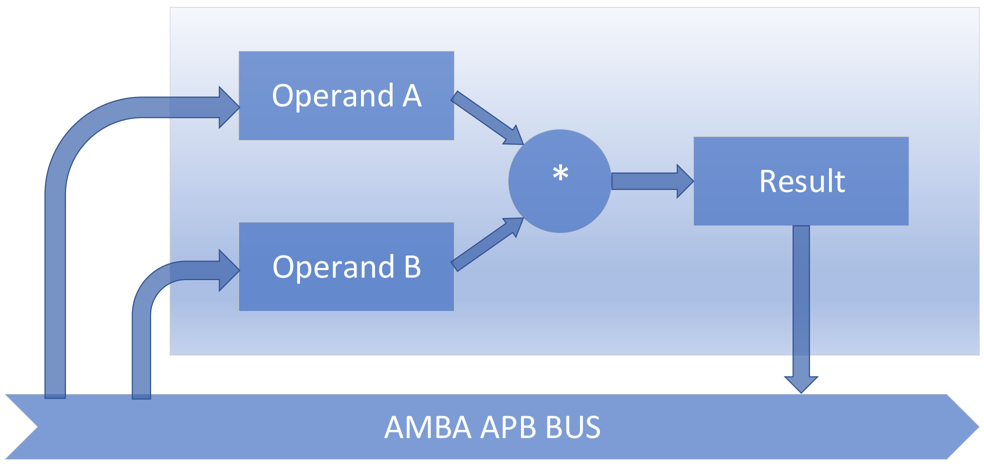


Fig. 10 First pcore

Since it is the first pcore that to be introduced in this documentation, I would like to show its source code and make some explanation.

Code:

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
  --use ieee.std_logic_unsigned.all;

library grlib;
    use grlib.amba.all;
    use grlib.devices.all;
    use grlib.stdlib.all;

library gaisler;
  use gaisler.misc.all;


entity multiplier32bits is
    generic (
                pindex : integer := 0;
                paddr :  integer := 0;
                pmask :  integer := 16#fff#
            );
    port    (
                rst :  in  std_ulogic;
                clk :  in  std_ulogic;
                apbi : in  apb_slv_in_type;
                apbo : out apb_slv_out_type
            );
end entity multiplier32bits;
```

```vhdl
architecture rtl of multiplier32bits is

    constant REVISION : integer := 0;
    constant PCONFIG : apb_config_type := (
        0 => ahb_device_reg (VENDOR_GAISLER, GAISLER_MULTIPLIER32, 0,
                             REVISION, 0),
        1 => apb_iobar(paddr, pmask));

    type registers is record
           reg : std_logic_vector(31 downto 0);
    end record;

    signal a, a_in : registers;
    signal b, b_in : registers;
    signal c, c_in : registers;


    begin
        comb :  process(rst, a, b, c, apbi)
                variable a_readdata : std_logic_vector(31 downto 0);
                variable a_tempv    : registers;

                variable b_readdata : std_logic_vector(31 downto 0);
                variable b_tempv    : registers;

                variable c_readdata : std_logic_vector(31 downto 0);
                --variable c_tempv    : registers;
                variable c_tempv    : std_logic_vector(63 downto 0);



                begin
                    a_tempv := a;
                    b_tempv := b;
                    c_tempv := a.reg*b.reg;
                -- read register
                    a_readdata := (others => '0');
                    b_readdata := (others => '0');
                    c_readdata := (others => '0');
                    case apbi.paddr(4 downto 2) is
                        when "000" => a_readdata := a.reg(31 downto 0);
                        when "001" => b_readdata := b.reg(31 downto 0);
                        when "010" => c_readdata := c.reg(31 downto 0);
```

```
            when others => null;
        end case;
    -- write registers
        if (apbi.psel(pindex) and apbi.penable and
             apbi.pwrite) = '1'
        then
        case apbi.paddr(4 downto 2) is
            when "000" => a_tempv.reg := apbi.pwdata;
            when "001" => b_tempv.reg := apbi.pwdata;
            when others => null;
        end case;
        end if;
    -- system reset
        if rst = '0' then
        a_tempv.reg := (others => '0');
        b_tempv.reg := (others => '0');
        c_tempv     := (others => '0');
        end if;

        a_in <= a_tempv;
        b_in <= b_tempv;
        c_in.reg <= c_tempv(31 downto 0);
        case apbi.paddr(4 downto 2) is
            when "000" => apbo.prdata <= a_readdata; -- drive
                                                 --apb read bus
            when "001" => apbo.prdata <= b_readdata; -- drive
                                                 --apb read bus
            when "010" => apbo.prdata <= c_readdata; -- drive
                                                 --apb read bus
            when others => apbo.prdata <=
            "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
        end case;
    end process;

apbo.pirq <= (others => '0'); -- No IRQ
apbo.pindex <= pindex;       -- VHDL generic
apbo.pconfig <= PCONFIG;      -- Config constant

-- registers
regs : process(clk)
    begin
        if rising_edge(clk) then
        a <= a_in;
        b <= b_in;
```

```
              c <= c_in;
             end if;
          end process;
      -- boot message
      -- pragma translate_off
      bootmsg : report_version
          generic map ("reg_32bits" & tost(pindex) &": Example core
                            rev " & tost(REVISION));
      -- pragma translate_on
end architecture rtl;
```

It has two processes. One is for combinational logic "`comb`", the other is for generating the registers (D flipflops) "`regs`". In "comb" process, we use variables rather than signals to describe the APB protocol algorithm.

Code parts

```
    constant REVISION : integer := 0;
    constant PCONFIG : apb_config_type := (
       0 => ahb_device_reg (VENDOR_GAISLER, GAISLER_MULTIPLIER32, 0,
                               REVISION, 0),
       1 => apb_iobar(paddr, pmask));
```

and

```
      apbo.pirq <= (others => '0'); -- No IRQ
      apbo.pindex <= pindex;        -- VHDL generic
      apbo.pconfig <= PCONFIG;      -- Config constant
```

is for APB Plug&Play configuration. Every pcore that connected to the APB bus should has these code parts.

VHDL case block

```
   case apbi.paddr(4 downto 2) is
      when "000" => apbo.prdata <= a_readdata; -- drive --apb read bus
      when "001" => apbo.prdata <= b_readdata; -- drive --apb read bus
      when "010" => apbo.prdata <= c_readdata; -- drive --apb read bus
      when others => apbo.prdata <=
      "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
   end case;
```

uses a "XXX…XXX" at other conditions. It is recommended to use this output, otherwise it is possible that the synthesis tool will misunderstand your purpose and introduce a latch in the pcore.

You can also refer to part “8.3 Adding an AMBA IP core to GRLIB” in grlib.pdf for a simple code version. It only has one register. It is a good example for the beginners to refer to.

2.3.3 The Second Pcore and the Third Pcore

Since the second and the third pcore are detailly shown in the slides, you could refer to it for detailed information. Also, I wrote many comments in the source code, it would be much easier to understand the source code if it has not any comments.

A point worth mentioning is the address offset value. All the APB address offset value of the addressable registers in these two versions of pcores are listed in the tables in slides. If you finish reading the APB Plug&Play part in the grlib.pdf and already understood it, you will know that the VHDL generic “pmask” decides the address size of the pcore, which is occupied by the pcore on APB bus; generic “paddr” decides the value of start address of this pcore; generic “pindex” can give a number of this pcore to identify this pcore from other pcores or peripherals that have different pindex value.

What’s more, according to the function of “pmask”, the address range size of a APB peripheral is not continuous, it can only be the value listed below.

256Byte, 512Byte, 1024Byte/1KB, 2KB, 4LB, ···, 1MB

When it is 1MB, there is only one APB peripheral and it occupies all the allowed APB address.

As for the address offset, it shows a value of every addressable registers in the pcore. Based on this value, we can calculate the actual address of these addressable registers. See the example below.

Example:
in a pcore on APB bus, paddr = 16#020#, one address offset of a register is 16#2#.
The starting address of AHB/APB bridge is 0x8010_0000.

The starting address of AHB/APB bridge is 0x8010_0000 means that all APB peripherals will occupy the address from 0x8000_0000 to 0x8010_0000.

According to user manual, paddr is the value of bit [19:8] of the staring address for a pcore on the APB bus (part ”5.5.3 Address decoding” in grlib.pdf).

So, the start address of this pcore is

- 0x8000_0000 + 0x0000_2000 = 0x8000_2000.

As a result, the address of the addressable register mentioned in this example is

- 0x8000_2000 + 0x2 = 0x8000_2002.

So, you can declare a volatile unsigned int pointer in C to write and/or read value to/from this register.

```c
volatile unsigned int * example_register; //point to the register
example_register = (unsigned int * )0x80002002;
```

You can see the similar usage in the testing software.

2.3.4 Verifying the Trace Result

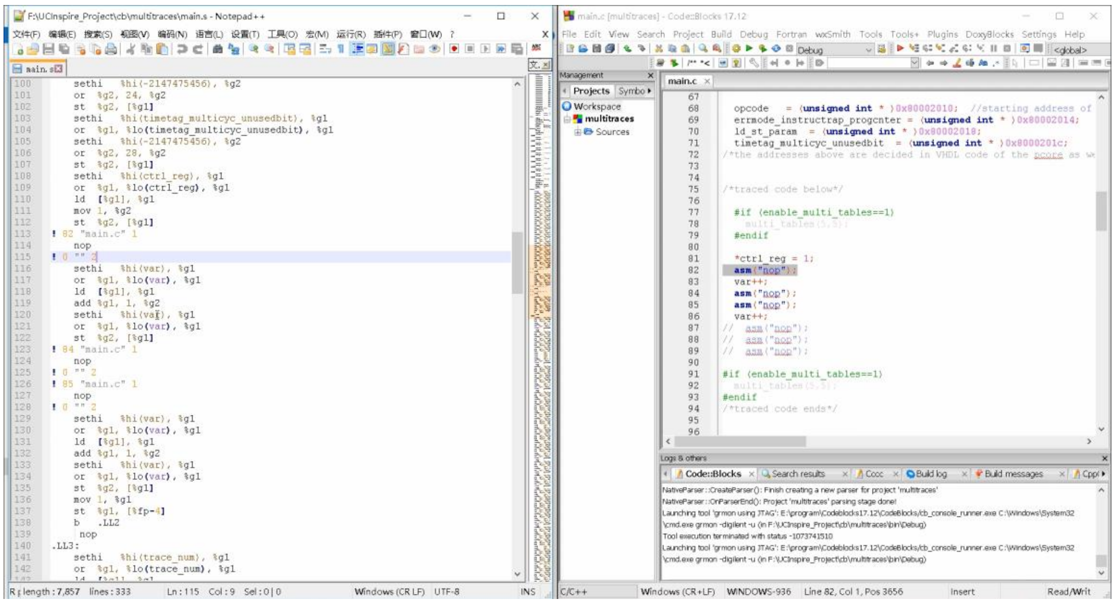
Before moving on, it is important that we should make sure the trace result is correct. To know this, we need to compare the assembly code of the executed software, and trace result and the disassembly information given by the GRMON by using the command "disassemble <function name>" or "disassemble <address value>".

To get the assembly code, we can use the command to control the BCC(bare C cross compiler) to generate this. This command is

- `sparc-gaisler-elf-gcc-7.2.0.exe -O0 -S main.c -o main.s`

Flag -o0 is to let the compiler not to optimize the assembly code; flag -S is used to let the compiler generate the assembly code; flag -o is telling the compiler output the assembly file (.s file).

The comparation result is shown in the figure below.



F:\UCInspire_Project\cb\multitraces\main.s - Notepad++

```
	sethi	%hi(-2147475456), %g2
	or	%g2, 24, %g2
	st	%g2, [%g1]
	sethi	%hi(timetag_multicyc_unusedbit), %g1
	or	%g1, %lo(timetag_multicyc_unusedbit), %g1
	sethi	%hi(-2147475456), %g2
	or	%g2, 28, %g2
	st	%g2, [%g1]
	sethi	%hi(ctrl_reg), %g1
	or	%g1, %lo(ctrl_reg), %g1
	ld	[%g1], %g1
	mov	1, %g2
	st	%g2, [%g1]
! 82 "main.c" 1
	nop
! 0 "" 2
	sethi	%hi(var), %g1
	or	%g1, %lo(var), %g1
	ld	[%g1], %g1
	add	%g1, 1, %g2
	sethi	%hi(var), %g1
	or	%g1, %lo(var), %g1
	st	%g2, [%g1]
! 84 "main.c" 1
	nop
! 0 "" 2
! 85 "main.c" 1
	nop
! 0 "" 2
	sethi	%hi(var), %g1
	or	%g1, %lo(var), %g1
	ld	[%g1], %g1
	add	%g1, 1, %g2
	sethi	%hi(var), %g1
	or	%g1, %lo(var), %g1
	st	%g2, [%g1]
	mov	1, %g1
	st	%g1, [%fp-4]
	b	.LL2
	 nop
.LL3:
	sethi	%hi(trace_num), %g1
	or	%g1, %lo(trace_num), %g1
```

length : 7,857 lines : 333 Ln : 115 Col : 9 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

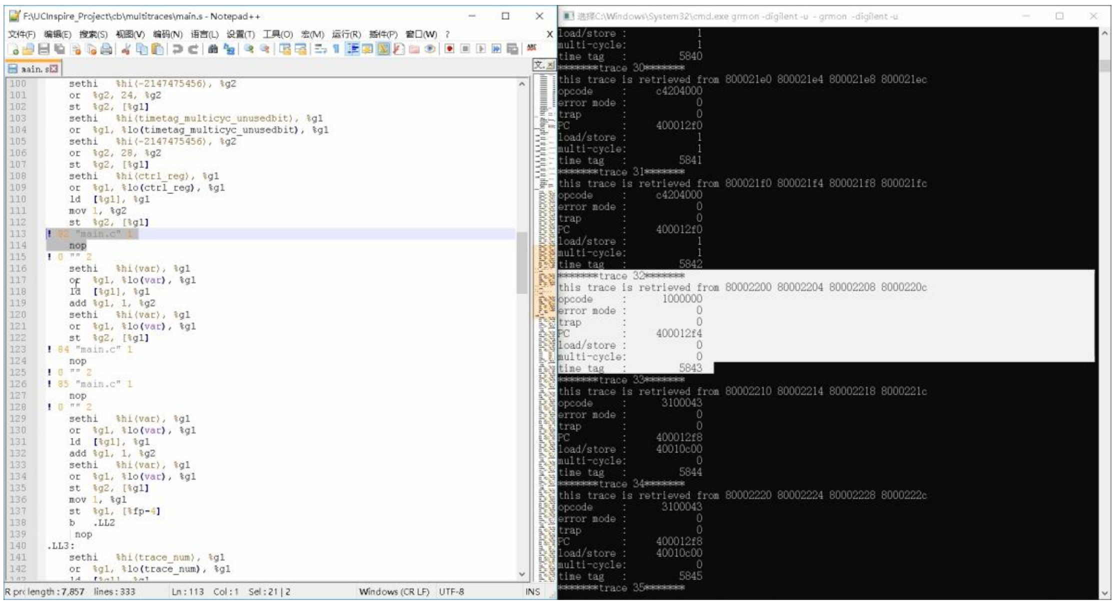
main.c [multitraces] - Code::Blocks 17.12

```c

  opcode   = (unsigned int * )0x80002010;  //starting address of
  errmode_instructrap_progcnter = (unsigned int * )0x80002014;
  ld_st_param  = (unsigned int * )0x80002018;
  timetag_multicyc_unusedbit  = (unsigned int * )0x8000201c;
/*the addresses above are decided in VHDL code of the pcore as w


/*traced code below*/

  #if (enable_multi_tables==1)
    multi_tables(5,5);
  #endif

  *ctrl_reg = 1;
  asm("nop");
  var++;
  asm("nop");
  asm("nop");
  var++;
//  asm("nop");
//  asm("nop");
//  asm("nop");

#if (enable_multi_tables==1)
  multi_tables(5,5);
#endif
/*traced code ends*/
```

```
NativeParser::CreateParser(): Finish creating a new parser for project 'multitraces'
NativeParser::OnParserEnd(): Project 'multitraces' parsing stage done!
Launching tool 'grmon using JTAG': E:\program\Codeblocks17.12\CodeBlocks/cb_console_runner.exe C:\Windows\System32
\cmd.exe grmon -digilent -u (in F:\UCInspire_Project\cb\multitraces\bin\Debug)
Tool execution terminated with status -1073741510
Launching tool 'grmon using JTAG': E:\program\Codeblocks17.12\CodeBlocks/cb_console_runner.exe C:\Windows\System32
\cmd.exe grmon -digilent -u (in F:\UCInspire_Project\cb\multitraces\bin\Debug)
```

C/C++ Windows (CR+LF) WINDOWS-936 Line 82, Col 1, Pos 3656 Insert Read/Writ

Fig. 11 a) C and assembly code

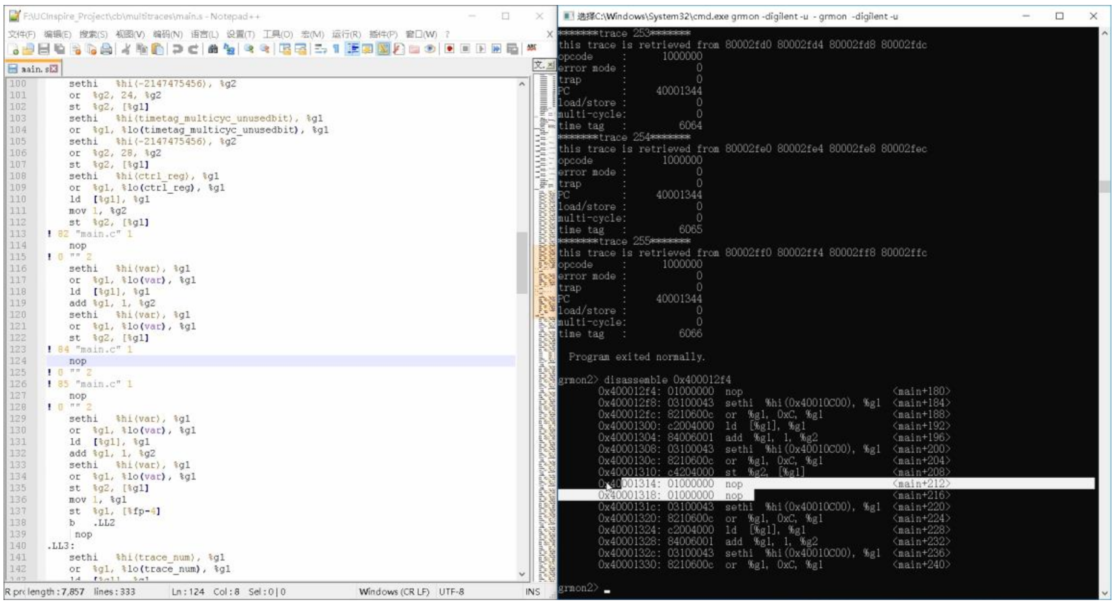

F:\UCInspire_Project\cb\multitraces\main.s - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
main.s
选择C:\Windows\System32\cmd.exe grmon -digilent -u - grmon -digilent -u

```
    sethi   %hi(-2147475456), %g2
    or  %g2, 24, %g2
    st  %g2, [%g1]
    sethi   %hi(timetag_multicyc_unusedbit), %g1
    or  %g1, %lo(timetag_multicyc_unusedbit), %g1
    sethi   %hi(-2147475456), %g2
    or  %g2, 28, %g2
    st  %g2, [%g1]
    sethi   %hi(ctrl_reg), %g1
    or  %g1, %lo(ctrl_reg), %g1
    ld  [%g1], %g1
    mov 1, %g2
    st  %g2, [%g1]
! 82 "main.c" 1
    nop
! 0 "" 2
    sethi   %hi(var), %g1
    or  %g1, %lo(var), %g1
    ld  [%g1], %g1
    add %g1, 1, %g2
    sethi   %hi(var), %g1
    or  %g1, %lo(var), %g1
    st  %g2, [%g1]
! 84 "main.c" 1
    nop
! 0 "" 2
! 85 "main.c" 1
    nop
! 0 "" 2
    sethi   %hi(var), %g1
    or  %g1, %lo(var), %g1
    ld  [%g1], %g1
    add %g1, 1, %g2
    sethi   %hi(var), %g1
    or  %g1, %lo(var), %g1
    st  %g2, [%g1]
    mov 1, %g1
    st  %g1, [%fp-4]
    b   .LL2
     nop
.LL3:
    sethi   %hi(trace_num), %g1
    or  %g1, %lo(trace_num), %g1
```

```
R pro length : 7,857   lines : 333   Ln : 113   Col : 1   Sel : 21 | 2   Windows (CR LF)   UTF-8   INS
```

```
load/store :          1
multi-cycle:          1
time tag   :       5840
*******trace 30*******
this trace is retrieved from 800021e0 800021e4 800021e8 800021ec
opcode     :    c4204000
error mode :          0
trap       :          0
PC         :    400012f0
load/store :          1
multi-cycle:          1
time tag   :       5841
*******trace 31*******
this trace is retrieved from 800021f0 800021f4 800021f8 800021fc
opcode     :    c4204000
error mode :          0
trap       :          0
PC         :    400012f0
load/store :          1
multi-cycle:          1
time tag   :       5842
*******trace 32*******
this trace is retrieved from 80002200 80002204 80002208 8000220c
opcode     :     1000000
error mode :          0
trap       :          0
PC         :    400012f4
load/store :          0
multi-cycle:          0
time tag   :       5843
*******trace 33*******
this trace is retrieved from 80002210 80002214 80002218 8000221c
opcode     :     3100043
error mode :          0
trap       :          0
PC         :    400012f8
load/store :    40010c00
multi-cycle:          0
time tag   :       5844
*******trace 34*******
this trace is retrieved from 80002220 80002224 80002228 8000222c
opcode     :     3100043
error mode :          0
trap       :          0
PC         :    400012f8
load/store :    40010c00
multi-cycle:          0
time tag   :       5845
*******trace 35*******
```

Fig. 11 b) Assembly code and trace result


F:\UCInspire_Project\cb\multitraces\main.s - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
main.s
选择C:\Windows\System32\cmd.exe grmon -digilent -u - grmon -digilent -u

```
    sethi   %hi(-2147475456), %g2
    or  %g2, 24, %g2
    st  %g2, [%g1]
    sethi   %hi(timetag_multicyc_unusedbit), %g1
    or  %g1, %lo(timetag_multicyc_unusedbit), %g1
    sethi   %hi(-2147475456), %g2
    or  %g2, 28, %g2
    st  %g2, [%g1]
    sethi   %hi(ctrl_reg), %g1
    or  %g1, %lo(ctrl_reg), %g1
    ld  [%g1], %g1
    mov 1, %g2
    st  %g2, [%g1]
! 82 "main.c" 1
    nop
! 0 "" 2
    sethi   %hi(var), %g1
    or  %g1, %lo(var), %g1
    ld  [%g1], %g1
    add %g1, 1, %g2
    sethi   %hi(var), %g1
    or  %g1, %lo(var), %g1
    st  %g2, [%g1]
! 84 "main.c" 1
    nop
! 0 "" 2
! 85 "main.c" 1
    nop
! 0 "" 2
    sethi   %hi(var), %g1
    or  %g1, %lo(var), %g1
    ld  [%g1], %g1
    add %g1, 1, %g2
    sethi   %hi(var), %g1
    or  %g1, %lo(var), %g1
    st  %g2, [%g1]
    mov 1, %g1
    st  %g1, [%fp-4]
    b   .LL2
     nop
.LL3:
    sethi   %hi(trace_num), %g1
    or  %g1, %lo(trace_num), %g1
```

```
R pro length : 7,857   lines : 333   Ln : 124   Col : 8   Sel : 0 | 0   Windows (CR LF)   UTF-8   INS
```

```
*******trace 253*******
this trace is retrieved from 80002fd0 80002fd4 80002fd8 80002fdc
opcode     :     1000000
error mode :          0
trap       :          0
PC         :    40001344
load/store :          0
multi-cycle:          0
time tag   :       6064
*******trace 254*******
this trace is retrieved from 80002fe0 80002fe4 80002fe8 80002fec
opcode     :     1000000
error mode :          0
trap       :          0
PC         :    40001344
load/store :          0
multi-cycle:          0
time tag   :       6065
*******trace 255*******
this trace is retrieved from 80002ff0 80002ff4 80002ff8 80002ffc
opcode     :     1000000
error mode :          0
trap       :          0
PC         :    40001344
load/store :          0
multi-cycle:          0
time tag   :       6066

 Program exited normally.

grmon2> disassemble 0x400012f4
       0x400012f4: 01000000  nop                            <main+180>
       0x400012f8: 03100043  sethi  %hi(0x40010C00), %g1  <main+184>
       0x400012fc: 8210600c  or  %g1, 0xC, %g1            <main+188>
       0x40001300: c2004000  ld  [%g1], %g1               <main+192>
       0x40001304: 84006001  add  %g1, 1, %g2             <main+196>
       0x40001308: 03100043  sethi  %hi(0x40010C00), %g1  <main+200>
       0x4000130c: 8210600c  or  %g1, 0xC, %g1            <main+204>
       0x40001310: c4204000  st  %g2, [%g1]               <main+208>
       0x40001314: 01000000  nop                            <main+212>
       0x40001318: 01000000  nop                            <main+216>
       0x4000131c: 03100043  sethi  %hi(0x40010C00), %g1  <main+220>
       0x40001320: 8210600c  or  %g1, 0xC, %g1            <main+224>
       0x40001324: c2004000  ld  [%g1], %g1               <main+228>
       0x40001328: 84006001  add  %g1, 1, %g2             <main+232>
       0x4000132c: 03100043  sethi  %hi(0x40010C00), %g1  <main+236>
       0x40001330: 8210600c  or  %g1, 0xC, %g1            <main+240>

grmon2> _
```

Fig. 11 c) Comparing with disassemble information in GRMON

I use the simple SPARC V8 instruction "nop" and instructions to add 1 to a variable to do this comparation.

In Fig. 11 a) C and assembly code, left part is the assembly code, right part is the C code. We can find that the "nop" instruction and a series of instructions to realize "var++". In Fig.11 b), right part is the trace result. We can find the corresponding part of "nop" (highlight in the right part in this figure). We can also find the corresponding part of "var++", it takes more than one clock cycle so they cannot be shown within one figure; In Fig. 11 c), we can also find the matched code part of the assembly code and the disassemble information of GRMON, which is shown in the right part of this figure.

3. Adding Pcore to LEON3 Library

In this part, I will demonstrate how to add your pcore to LEON3's IP library so that when you generate the Vivado project by using command "scripts" in Cygwin, the script provided by the Cobham Gaisler will automatically scan all the IPs and add them into the LEON3 entity hierarchy.

All the IP files are in the folder "lib". We need to add our pcore VHDL or Verilog files into this folder. LEON3's directory structure can be shown in the figure below.

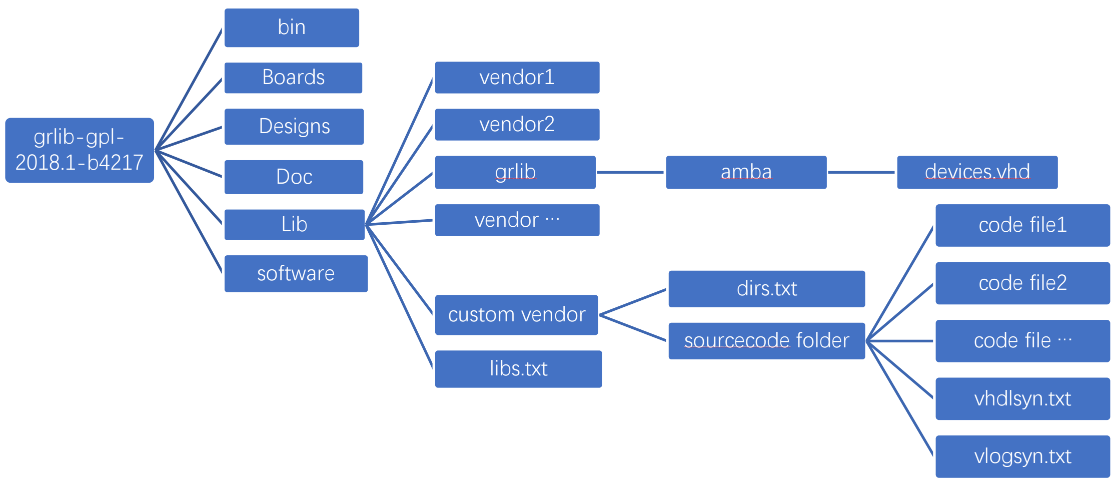


Fig. 12 LEON3 directory structure

Step1: creating a folder in directory /lib and name it with your desired name. This folder can be regarded as a new vendor that contains some pcore VHDL or Verilog files.

Step2: edit "libs.txt" file, adding your new folder name into it. This file includes all folder names of different vendor. Scripts will read this txt file to scan these folders in order to find the files that should be compiled and added into LEON3 design.

Step3: create a subfolder within the folder mentioned in step2, then create a txt file named "dirs.txt". You should give a name to this subfolder, then add this name in the dirs..txt file.

Step4: add you VHDL or Verilog design files into the subfolder mentioned in Step3. Then create txt file vhdlsyn.txt or vlogsyn.txt (based on the hardware description you use). If there are VHDL files as well as Verilog files, then create both two txt files. Then add the design file name into the corresponding txt files.

Step5: create a component/module declaration file (you'd better use VHDL and make it as a VHDL package). Add the declaration file name into the vhdlsyn.txt or vlogsyn.txt. A good website demonstrates how to instantiate Verilog module in VHDL:
ftp://ece.buap.mx/pub/Secretaria_Academica/SDC/Active_HDL_4.2_Student_Version_Installer/Doc/avhdl/Avh00213.htm
(The website above may require an IP from this university to visit…)

Step6: add you vendor information in \lib\grlib\amba\devices.vhd. For example, my vendor name is "trace", it is also the folder name under \lib. The subfolder within \trace is tracesnapshot; my pcore name (entity name) is also tracesnapshot. Then in devices.vhd, we should add the code below:

```
-----------new vendor-------
  constant VENDOR_TRACE     : amba_vendor_type := 16#FF#;
----------------------------
```

It indicates that we have added a new folder that contains our design files as a new vendor library.

And code:

```
--Multi-Trace device ids
  constant TRACE_TRACESNAPSHOT : amba_device_type := 16#001#;
```

will give a device number of our pcore.

And also

```
------------------------------
  constant TRACE_DESC : vendor_description := "Pcores for non-intru
                                               tracing";

  constant trace_device_table : device_table_type := (
     TRACE_TRACESNAPSHOT   => "mypcore for trace testing",
     others               => "Unknown Device              ");

  constant trace_lib : vendor_library_type := (
    vendorid     => VENDOR_TRACE,
    vendordesc   => TRACE_DESC,
    device_table => trace_device_table
    );
------------------------------
```

Save and close this file.

Step7: in top file leon3mp.vhd, include our library, i.e., the folder name under \lib directory, and use our component package name, i.e., the packages name in component declaration file. Then instantiate our pcore in this file.

Step8: back to Cygwin, launch XGrlib click the button "distclean all" and then "scripts", or directly issue the command "make distclean" and then "make scripts" to re-generate the Vivado project. Open this project file in Vivado and check the hierarchy, you should see your pcore has already added in leon3's hierarchy.

In my LEON3 project in this repository, I created a folder named "trace", you may refer to this as an example. Also, you could refer to the part "8 Extending GRLIB" in grlib.pdf

Now you have finished adding your pcore into the LEON3 system.