国立大学法人 電気通信大学 The University of Electro-Communications

Pham Laboratory
Integrated circuit design laboratory

「Course」
RISC-V Computer System Integration

「Lecture 2」 VexRiscv: a simple 32-bit MCU

Hoang Trong Thuc

2023/9

Outline

1. Introduction
2. Git clone, make, and self-test
3. Debug using OpenOCD
4. Briey SoC and debug with Eclipse
5. Practice: make a clock using timer

Outline

1. Introduction
2. Git clone, make, and self-test
3. Debug using OpenOCD
4. Briey SoC and debug with Eclipse
5. Practice: make a clock using timer

1. Introduction (1/2)

Reference link: *(Original)* https://github.com/SpinalHDL/VexRiscv
(Modified) https://github.com/thuchoang90/VexRiscv

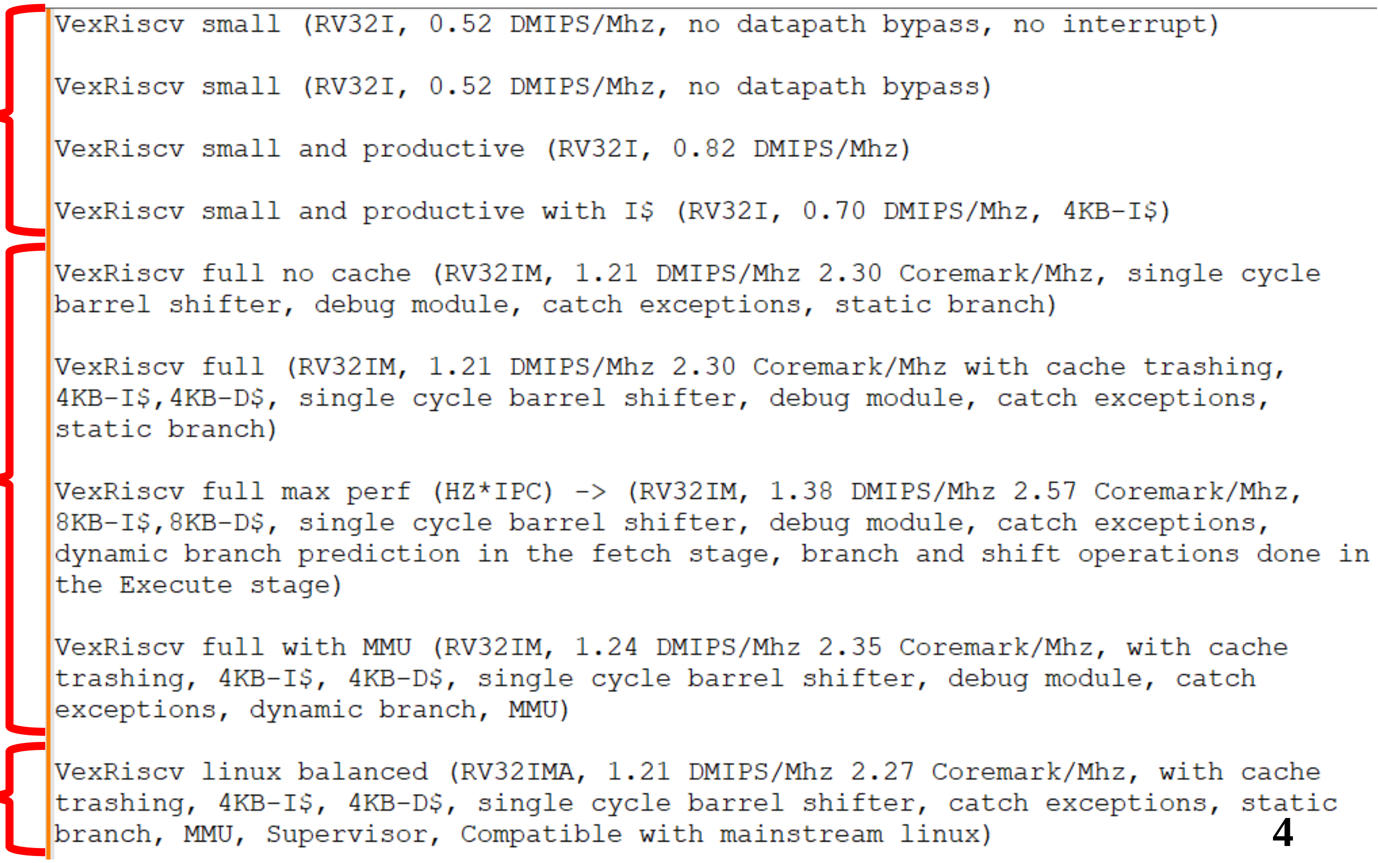
****Note:*** the project was written using the SpinalHDL library *(not the Chisel library)*, but the language is still Scala. Thus, the compile flow is similar to Chisel:

Scala *(+SpinalHDL)* → Java → FIRRTL → Verilog

The **VexRiscv** core has three main build options:

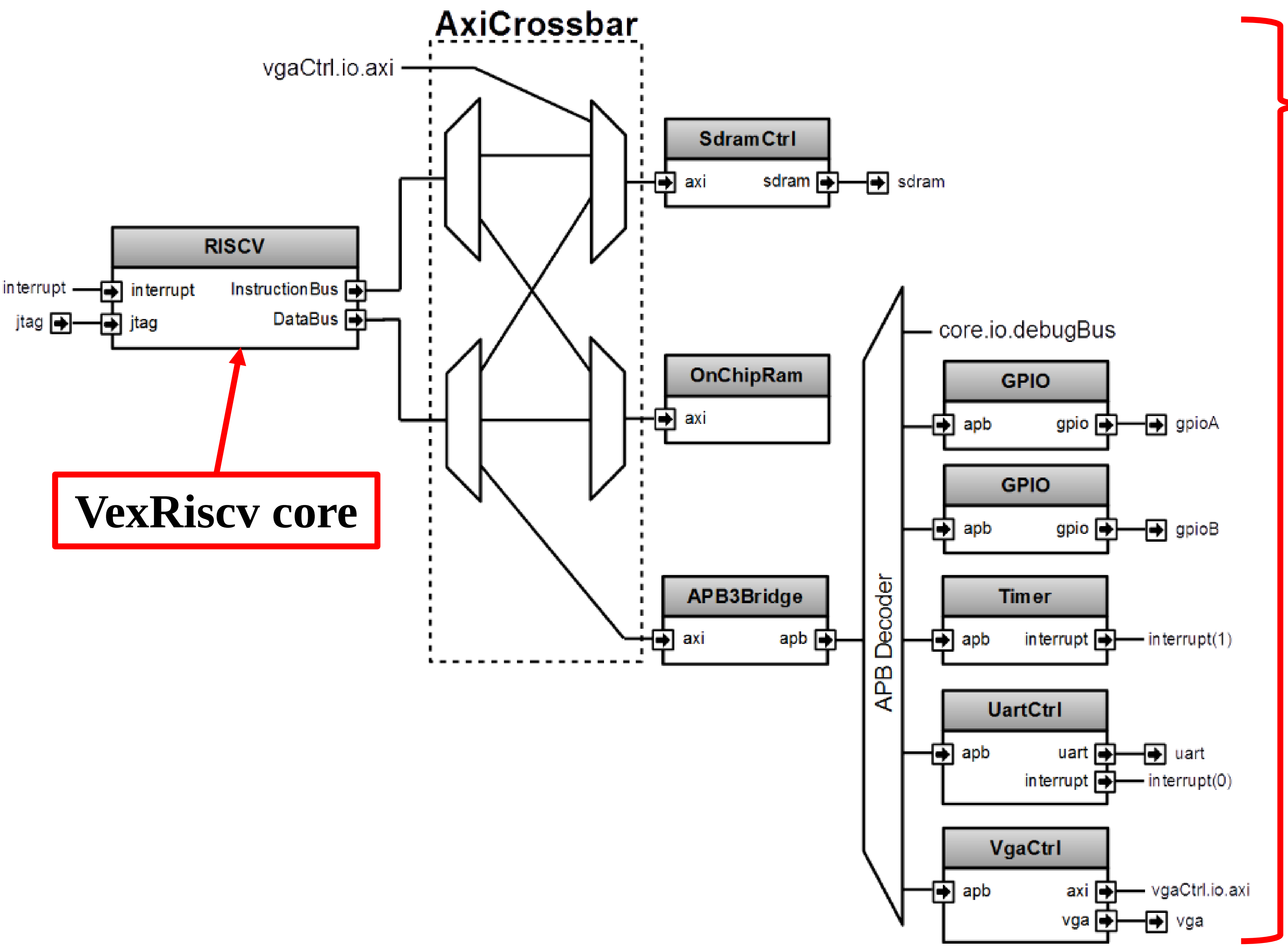
- Small: RV32I
- Full: RV32IM
- Linux: RV32IMA

(and other sub-options)



```
VexRiscv small (RV32I, 0.52 DMIPS/Mhz, no datapath bypass, no interrupt)

VexRiscv small (RV32I, 0.52 DMIPS/Mhz, no datapath bypass)

VexRiscv small and productive (RV32I, 0.82 DMIPS/Mhz)

VexRiscv small and productive with I$ (RV32I, 0.70 DMIPS/Mhz, 4KB-I$)

VexRiscv full no cache (RV32IM, 1.21 DMIPS/Mhz 2.30 Coremark/Mhz, single cycle
barrel shifter, debug module, catch exceptions, static branch)

VexRiscv full (RV32IM, 1.21 DMIPS/Mhz 2.30 Coremark/Mhz with cache trashing,
4KB-I$,4KB-D$, single cycle barrel shifter, debug module, catch exceptions,
static branch)

VexRiscv full max perf (HZ*IPC) -> (RV32IM, 1.38 DMIPS/Mhz 2.57 Coremark/Mhz,
8KB-I$,8KB-D$, single cycle barrel shifter, debug module, catch exceptions,
dynamic branch prediction in the fetch stage, branch and shift operations done in
the Execute stage)

VexRiscv full with MMU (RV32IM, 1.24 DMIPS/Mhz 2.35 Coremark/Mhz, with cache
trashing, 4KB-I$, 4KB-D$, single cycle barrel shifter, debug module, catch
exceptions, dynamic branch, MMU)

VexRiscv linux balanced (RV32IMA, 1.21 DMIPS/Mhz 2.27 Coremark/Mhz, with cache
trashing, 4KB-I$, 4KB-D$, single cycle barrel shifter, catch exceptions, static
branch, MMU, Supervisor, Compatible with mainstream linux)
```

1. Introduction (2/2)



Based on the core processor of VexRiscv, a completed MCU called **Briey SoC** was developed.

Briey SoC includes:

- VexRicv core
- Memory
- Peripherals

Outline

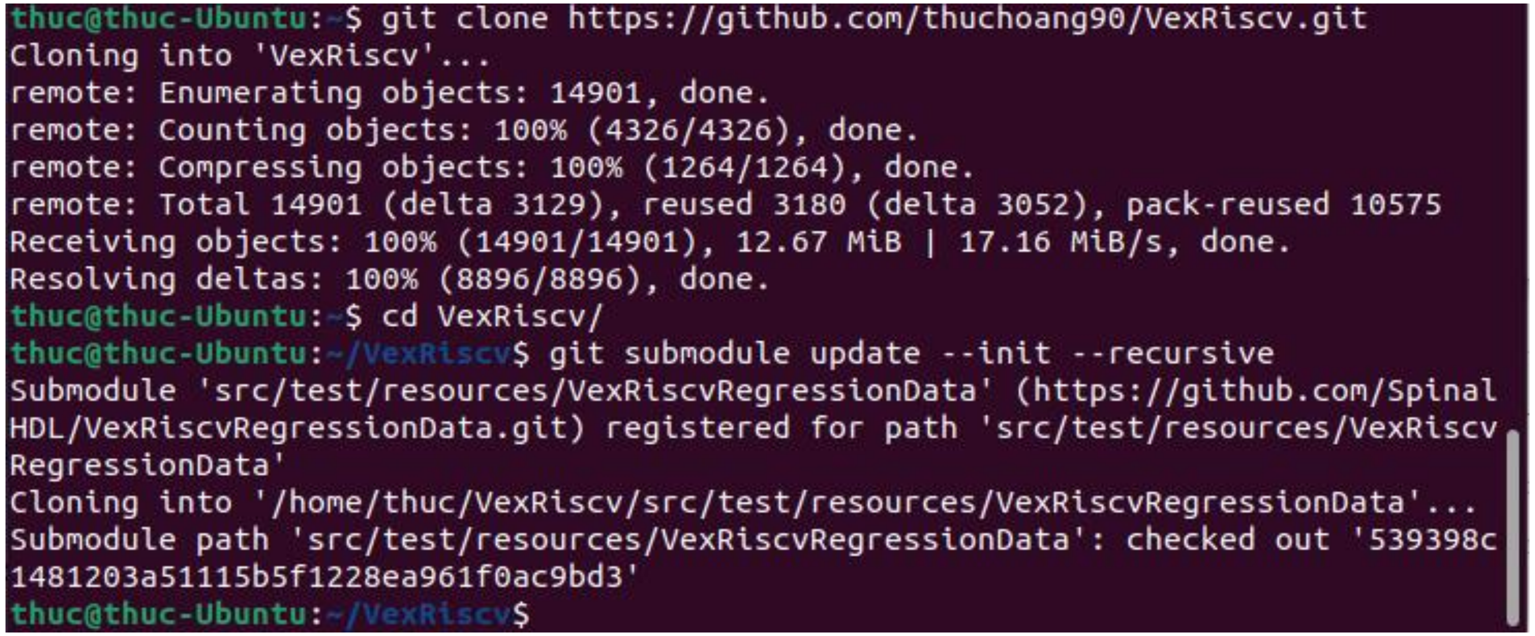
2. Git clone, make, and self-test (1/5) `$ git clone`

Note: you can follow the guide **I.a)** in:
https://thuchoang90.github.io/project/2020/07/23/VexRiscv

First, git clone. From your home folder:

```
$ git clone https://github.com/thuchoang90/VexRiscv.git
$ cd VexRiscv/
$ git submodule update --init --recursive
```

After git clone and submodule update:



```
thuc@thuc-Ubuntu:~$ git clone https://github.com/thuchoang90/VexRiscv.git
Cloning into 'VexRiscv'...
remote: Enumerating objects: 14901, done.
remote: Counting objects: 100% (4326/4326), done.
remote: Compressing objects: 100% (1264/1264), done.
remote: Total 14901 (delta 3129), reused 3180 (delta 3052), pack-reused 10575
Receiving objects: 100% (14901/14901), 12.67 MiB | 17.16 MiB/s, done.
Resolving deltas: 100% (8896/8896), done.
thuc@thuc-Ubuntu:~$ cd VexRiscv/
thuc@thuc-Ubuntu:~/VexRiscv$ git submodule update --init --recursive
Submodule 'src/test/resources/VexRiscvRegressionData' (https://github.com/Spinal
HDL/VexRiscvRegressionData.git) registered for path 'src/test/resources/VexRiscv
RegressionData'
Cloning into '/home/thuc/VexRiscv/src/test/resources/VexRiscvRegressionData'...
Submodule path 'src/test/resources/VexRiscvRegressionData': checked out '539398c
1481203a51115b5f1228ea961f0ac9bd3'
thuc@thuc-Ubuntu:~/VexRiscv$
```

2. Git clone, make, and self-test (2/5) Make

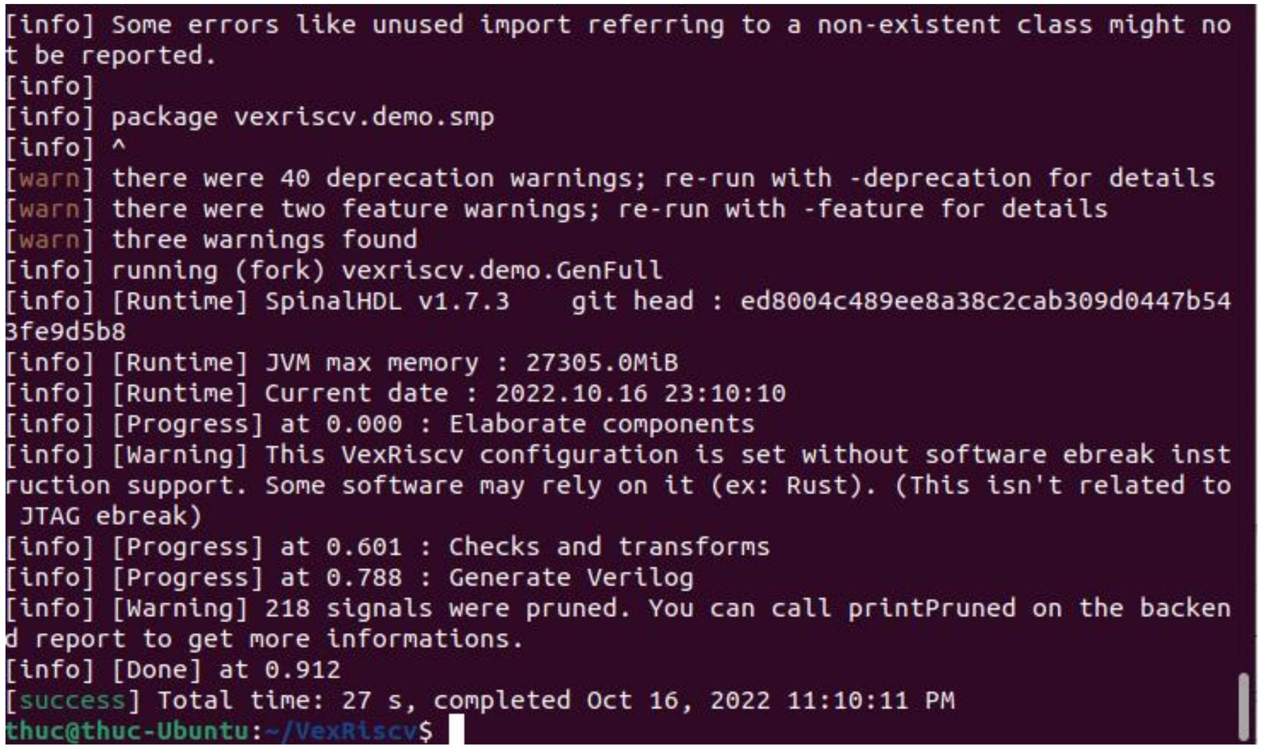
For make,
there are three options:

for smallest CPU:
```
$ sbt "runMain vexriscv.demo.GenSmallest"
```

for GenFull CPU:
```
$ sbt "runMain vexriscv.demo.GenFull"
```

for Linux CPU:
```
$ sbt "runMain vexriscv.demo.LinuxGen"
```

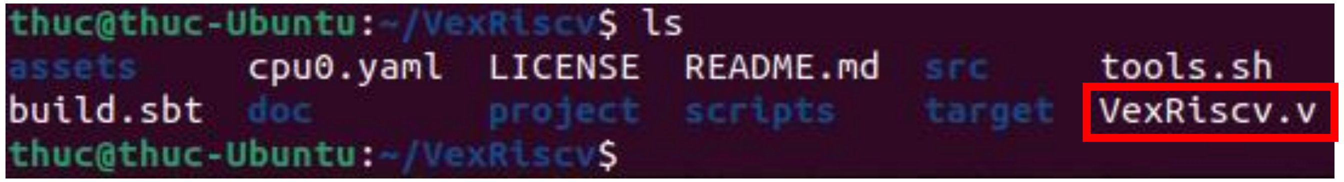
The terminal shows the result
after the **GenFull**:



```
[info] Some errors like unused import referring to a non-existent class might no
t be reported.
[info]
[info] package vexriscv.demo.smp
[info] ^
[warn] there were 40 deprecation warnings; re-run with -deprecation for details
[warn] there were two feature warnings; re-run with -feature for details
[warn] three warnings found
[info] running (fork) vexriscv.demo.GenFull
[info] [Runtime] SpinalHDL v1.7.3    git head : ed8004c489ee8a38c2cab309d0447b54
3fe9d5b8
[info] [Runtime] JVM max memory : 27305.0MiB
[info] [Runtime] Current date : 2022.10.16 23:10:10
[info] [Progress] at 0.000 : Elaborate components
[info] [Warning] This VexRiscv configuration is set without software ebreak inst
ruction support. Some software may rely on it (ex: Rust). (This isn't related to
 JTAG ebreak)
[info] [Progress] at 0.601 : Checks and transforms
[info] [Progress] at 0.788 : Generate Verilog
[info] [Warning] 218 signals were pruned. You can call printPruned on the backen
d report to get more informations.
[info] [Done] at 0.912
[success] Total time: 27 s, completed Oct 16, 2022 11:10:11 PM
thuc@thuc-Ubuntu:~/VexRiscv$
```

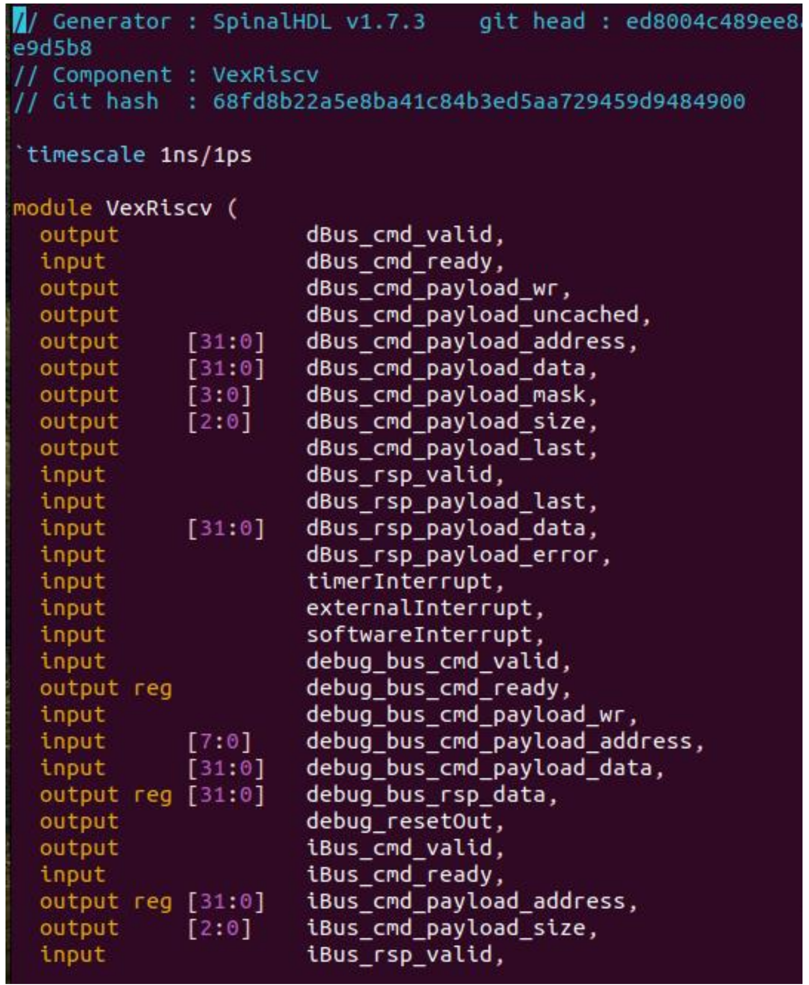
2. Git clone, make, and self-test (3/5) Verilog file

After **GenSmallest** or **GenFull** or **LinuxGen**, a Verilog file is created in the VexRiscv folder:



```
thuc@thuc-Ubuntu:~/VexRiscv$ ls
assets      cpu0.yaml  LICENSE  README.md  src     tools.sh
build.sbt   doc        project  scripts    target  VexRiscv.v
thuc@thuc-Ubuntu:~/VexRiscv$
```

That file contains the processor that we just generated:



```
// Generator : SpinalHDL v1.7.3    git head : ed8004c489ee8
e9d5b8
// Component : VexRiscv
// Git hash  : 68fd8b22a5e8ba41c84b3ed5aa729459d9484900

`timescale 1ns/1ps

module VexRiscv (
  output             dBus_cmd_valid,
  input              dBus_cmd_ready,
  output             dBus_cmd_payload_wr,
  output             dBus_cmd_payload_uncached,
  output     [31:0]  dBus_cmd_payload_address,
  output     [31:0]  dBus_cmd_payload_data,
  output     [3:0]   dBus_cmd_payload_mask,
  output     [2:0]   dBus_cmd_payload_size,
  output             dBus_cmd_payload_last,
  input              dBus_rsp_valid,
  input              dBus_rsp_payload_last,
  input      [31:0]  dBus_rsp_payload_data,
  input              dBus_rsp_payload_error,
  input              timerInterrupt,
  input              externalInterrupt,
  input              softwareInterrupt,
  input              debug_bus_cmd_valid,
  output reg         debug_bus_cmd_ready,
  input              debug_bus_cmd_payload_wr,
  input      [7:0]   debug_bus_cmd_payload_address,
  input      [31:0]  debug_bus_cmd_payload_data,
  output reg [31:0]  debug_bus_rsp_data,
  output             debug_resetOut,
  output             iBus_cmd_valid,
  input              iBus_cmd_ready,
  output reg [31:0]  iBus_cmd_payload_address,
  output     [2:0]   iBus_cmd_payload_size,
  input              iBus_rsp_valid,
```

2. Git clone, make, and self-test (4/5) Self-test

For each case *(smallest*, *full*, *linux)*, there is a self-test script using Verilator.

First, go to:

```
$ cd src/test/cpp/regression/
```

Then, run the Verilator self-test for each case:

for GenSmallest:

```
$ make clean run IBUS=SIMPLE DBUS=SIMPLE CSR=no MMU=no DEBUG_PLUGIN=no
MUL=no DIV=no
```

for GenFull:

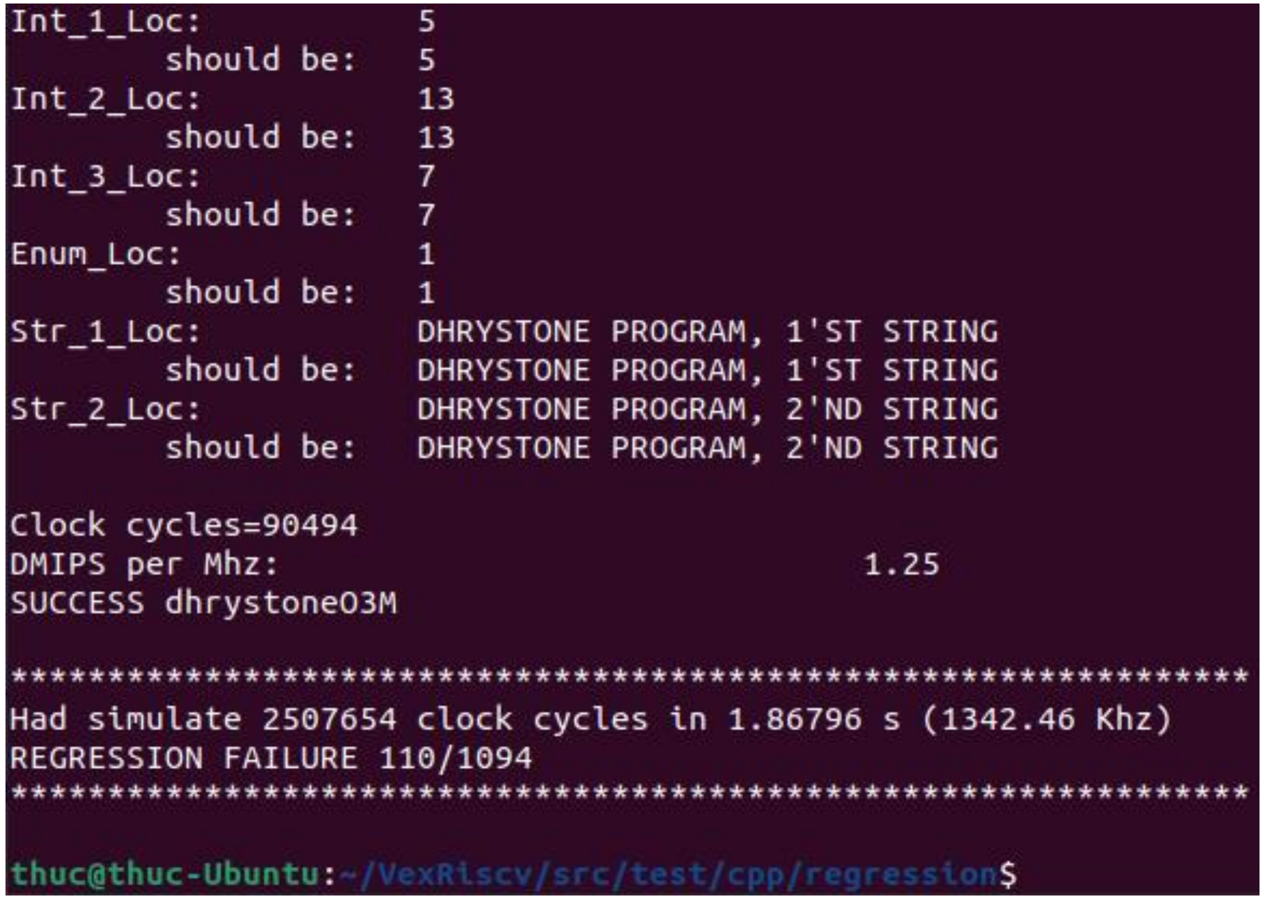
```
$ make clean run
```

for LinuxGen:

```
$ make clean run IBUS=CACHED DBUS=CACHED DEBUG_PLUGIN=STD DHRYSTONE=yes
SUPERVISOR=yes MMU=yes CSR=yes DEBUG_PLUGIN=no COMPRESSED=no MUL=yes
DIV=yes LRSC=yes AMO=yes REDO=10 TRACE=no COREMARK=yes
LINUX_REGRESSION=yes
```

2. Git clone, make, and self-test (5/5) Self-test

The terminal shows the result after the **GenFull** self-test:



```
Int_1_Loc:           5
        should be:   5
Int_2_Loc:           13
        should be:   13
Int_3_Loc:           7
        should be:   7
Enum_Loc:            1
        should be:   1
Str_1_Loc:           DHRYSTONE PROGRAM, 1'ST STRING
        should be:   DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc:           DHRYSTONE PROGRAM, 2'ND STRING
        should be:   DHRYSTONE PROGRAM, 2'ND STRING

Clock cycles=90494
DMIPS per Mhz:                              1.25
SUCCESS dhrystoneO3M

****************************************************************
Had simulate 2507654 clock cycles in 1.86796 s (1342.46 Khz)
REGRESSION FAILURE 110/1094
****************************************************************

thuc@thuc-Ubuntu:~/VexRiscv/src/test/cpp/regression$
```

* Now, repeat the test with **GenSmallest** and **GenLinux**.

Outline

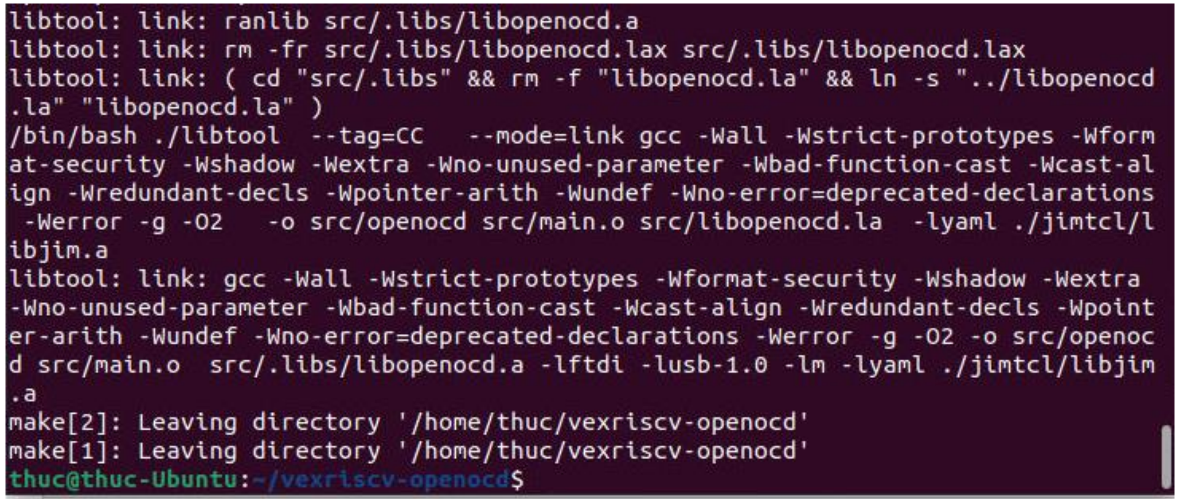
3. Debug using OpenOCD (1/6) OpenOCD

****Note:*** you can follow the guide **I.c)** and **I.d)** in:
https://thuchoang90.github.io/project/2020/07/23/VexRiscv

For debugging, first, we need to prepare the **OpenOCD** folder.
From your home folder:

```
$ git clone https://github.com/SpinalHDL/openocd_riscv.git vexriscv-openocd
$ cd vexriscv-openocd/
$ git submodule update --init --recursive
$ ./bootstrap
$ ./configure --enable-ftdi --enable-dummy
$ make
```

The terminal after **make**:

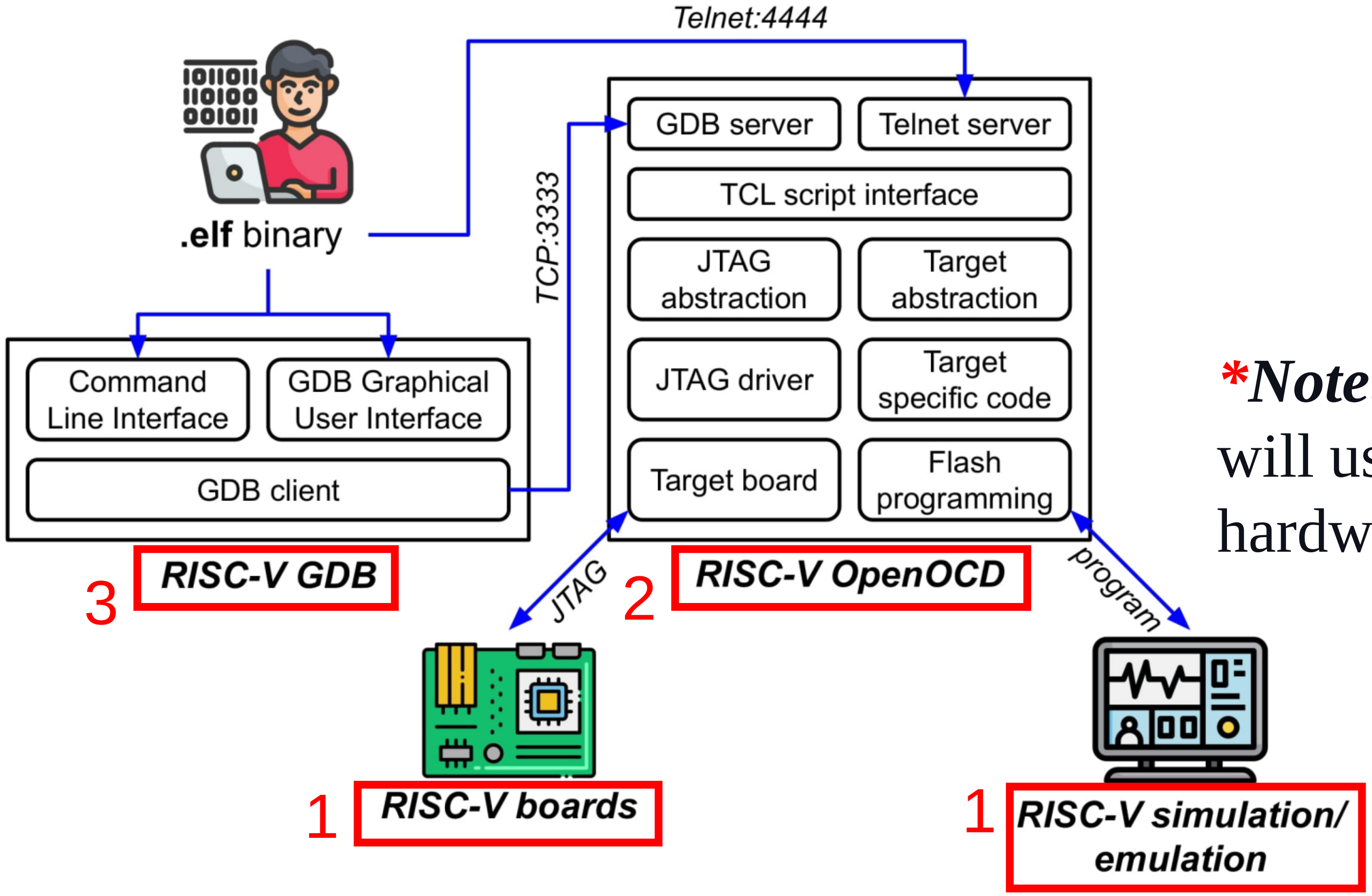


```
libtool: link: ranlib src/.libs/libopenocd.a
libtool: link: rm -fr src/.libs/libopenocd.lax src/.libs/libopenocd.lax
libtool: link: ( cd "src/.libs" && rm -f "libopenocd.la" && ln -s "../libopenocd
.la" "libopenocd.la" )
/bin/bash ./libtool  --tag=CC   --mode=link gcc -Wall -Wstrict-prototypes -Wform
at-security -Wshadow -Wextra -Wno-unused-parameter -Wbad-function-cast -Wcast-al
ign -Wredundant-decls -Wpointer-arith -Wundef -Wno-error=deprecated-declarations
 -Werror -g -O2   -o src/openocd src/main.o src/libopenocd.la  -lyaml ./jimtcl/l
ibjim.a
libtool: link: gcc -Wall -Wstrict-prototypes -Wformat-security -Wshadow -Wextra
-Wno-unused-parameter -Wbad-function-cast -Wcast-align -Wredundant-decls -Wpoint
er-arith -Wundef -Wno-error=deprecated-declarations -Werror -g -O2 -o src/openoc
d src/main.o  src/.libs/libopenocd.a -lftdi -lusb-1.0 -lm -lyaml ./jimtcl/libjim
.a
make[2]: Leaving directory '/home/thuc/vexriscv-openocd'
make[1]: Leaving directory '/home/thuc/vexriscv-openocd'
thuc@thuc-Ubuntu:~/vexriscv-openocd$
```

3. Debug using OpenOCD (2/6) Debug flow

After compiling the *(C/C++)* software, to debug, we need three things:

1. The hardware *(or the simulated/emulated)*
2. OpenOCD *(just installed earlier)*
3. GDB *(from the RISC-V toolchain)*



****Note:*** in this lecture, we will use the simulated hardware via Verilator.

3. Debug using OpenOCD (3/6) Simulated hardware

To run the simulated hardware:

Go to your VexRiscv folder:

```
$ cd VexRiscv/
```

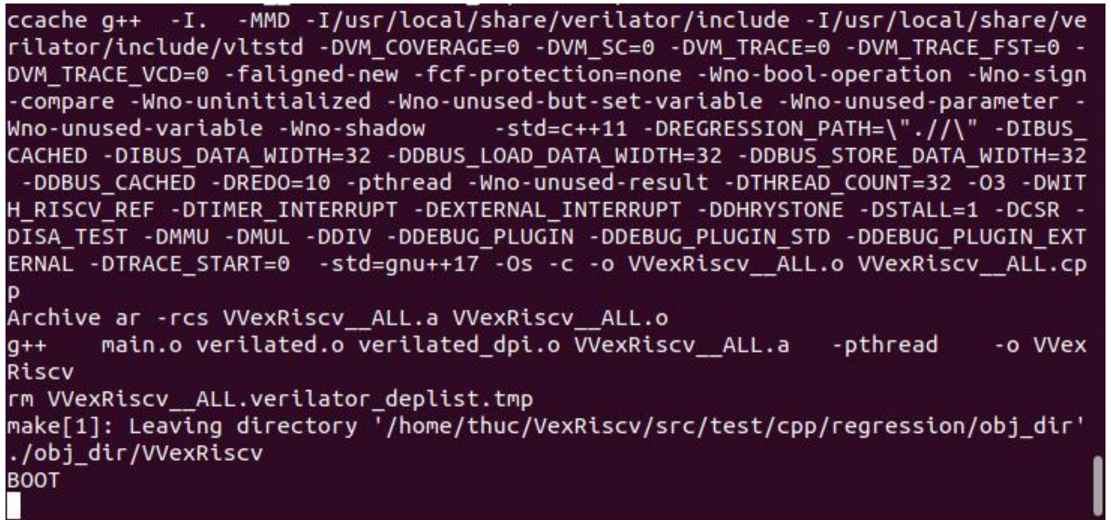
Generate GenFull:

```
$ sbt "runMain vexriscv.demo.GenFull"
```

Run Verilator:

```
$ cd src/test/cpp/regression/
$ make clean run DEBUG_PLUGIN_EXTERNAL=yes
```

The terminal after:



```
ccache g++  -I.  -MMD -I/usr/local/share/verilator/include -I/usr/local/share/ve
rilator/include/vltstd -DVM_COVERAGE=0 -DVM_SC=0 -DVM_TRACE=0 -DVM_TRACE_FST=0 -
DVM_TRACE_VCD=0 -faligned-new -fcf-protection=none -Wno-bool-operation -Wno-sign
-compare -Wno-uninitialized -Wno-unused-but-set-variable -Wno-unused-parameter -
Wno-unused-variable -Wno-shadow     -std=c++11 -DREGRESSION_PATH=\"./\/\" -DIBUS_
CACHED -DIBUS_DATA_WIDTH=32 -DDBUS_LOAD_DATA_WIDTH=32 -DDBUS_STORE_DATA_WIDTH=32
 -DDBUS_CACHED -DREDO=10 -pthread -Wno-unused-result -DTHREAD_COUNT=32 -O3 -DWIT
H_RISCV_REF -DTIMER_INTERRUPT -DEXTERNAL_INTERRUPT -DDHRYSTONE -DSTALL=1 -DCSR -
DISA_TEST -DMMU -DMUL -DDIV -DDEBUG_PLUGIN -DDEBUG_PLUGIN_STD -DDEBUG_PLUGIN_EXT
ERNAL -DTRACE_START=0  -std=gnu++17 -Os -c -o VVexRiscv__ALL.o VVexRiscv__ALL.cp
p
Archive ar -rcs VVexRiscv__ALL.a VVexRiscv__ALL.o
g++    main.o verilated.o verilated_dpi.o VVexRiscv__ALL.a    -pthread    -o VVex
Riscv
rm VVexRiscv__ALL.verilator_deplist.tmp
make[1]: Leaving directory '/home/thuc/VexRiscv/src/test/cpp/regression/obj_dir'
./obj_dir/VVexRiscv
BOOT
```

3. Debug using OpenOCD (4/6) Connect to OpenOCD

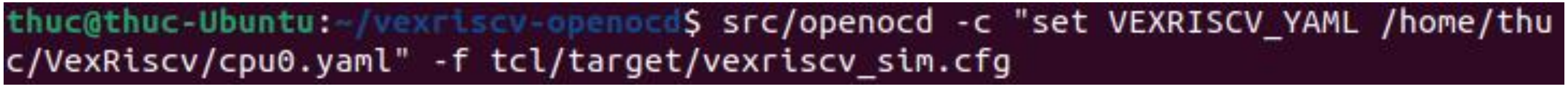
Keep the previous terminal, **open** a new terminal, then:

Go to your vexriscv_openocd folder:

```
$ cd vexriscv_openocd/

$ src/openocd -c "set VEXRISCV_YAML <cpu0.yaml PATH>" -f
tcl/target/vexriscv_sim.cfg
```

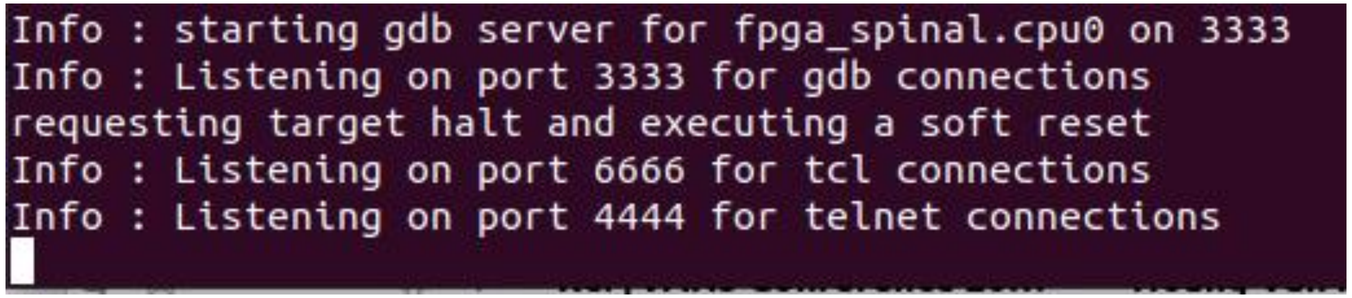
Where the ***<cpu0.yaml PATH>*** points to the file cpu0.yaml in the **VexRiscv** folder.

For example:



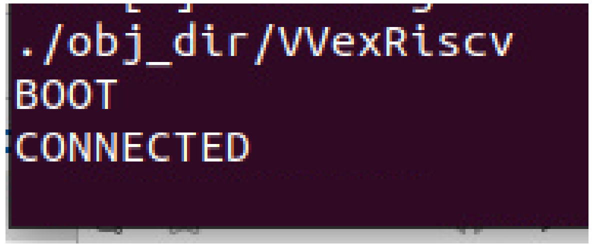
```
thuc@thuc-Ubuntu:~/vexriscv-openocd$ src/openocd -c "set VEXRISCV_YAML /home/thu
c/VexRiscv/cpu0.yaml" -f tcl/target/vexriscv_sim.cfg
```

The terminal after:



```
Info : starting gdb server for fpga_spinal.cpu0 on 3333
Info : Listening on port 3333 for gdb connections
requesting target halt and executing a soft reset
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

The previous Verilator terminal:



```
./obj_dir/VVexRiscv
BOOT
CONNECTED
```

3. Debug using OpenOCD (5/6) Run software using GDB

Keep both previous terminals, **open** a new terminal, then:

Export the RISC-V toolchain:

```
$ export PATH=/opt/riscv/bin/:$PATH
```

Go to your VexRiscv folder:

```
$ cd VexRiscv/
```

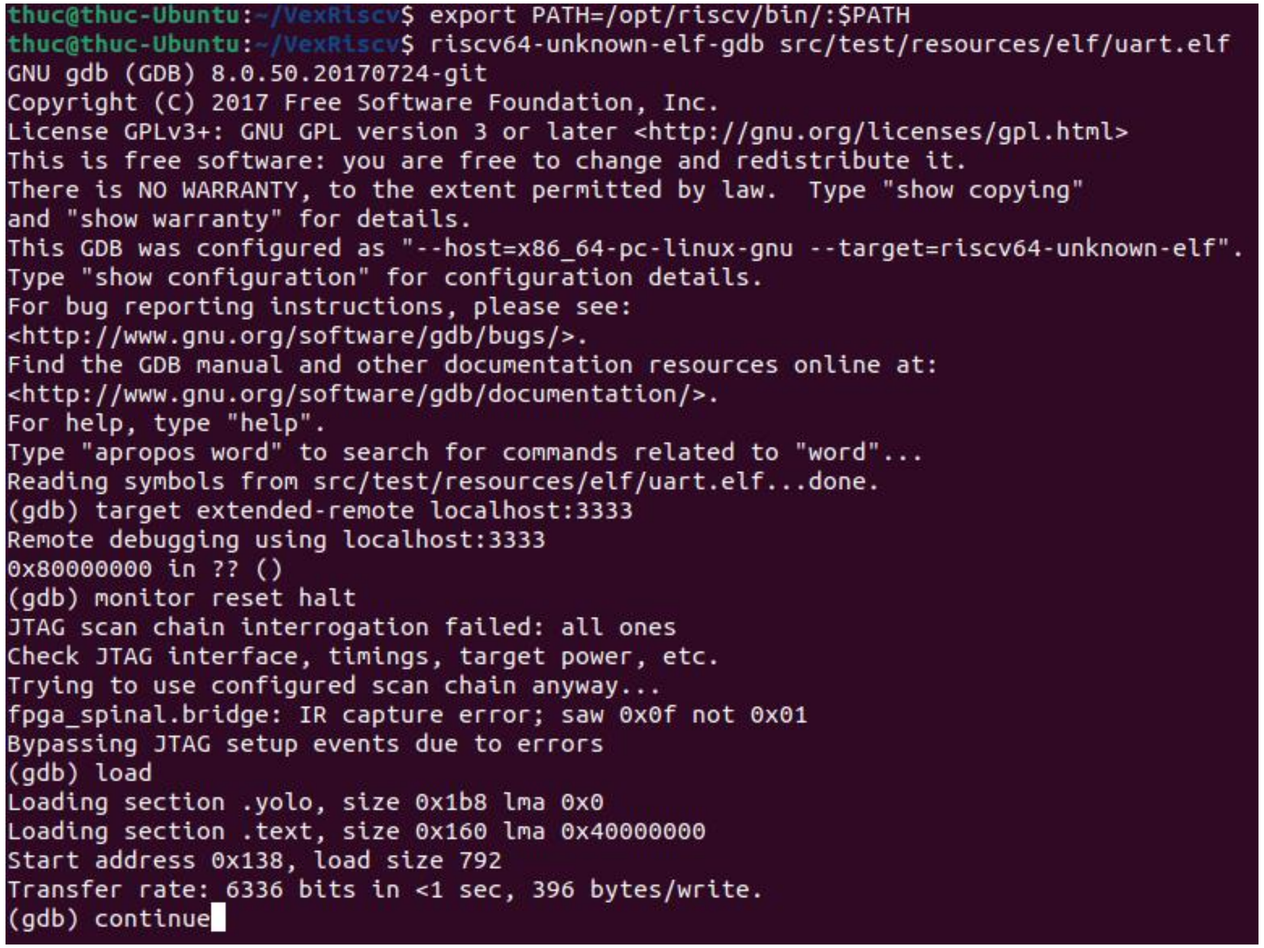
Run the software with RISC-V GDB:

```
$ riscv64-unknown-elf-gdb src/test/resources/elf/uart.elf
$ target extended-remote localhost:3333
$ monitor reset halt
$ load
$ continue
```

After this, it should prints messages to the Verilator terminal

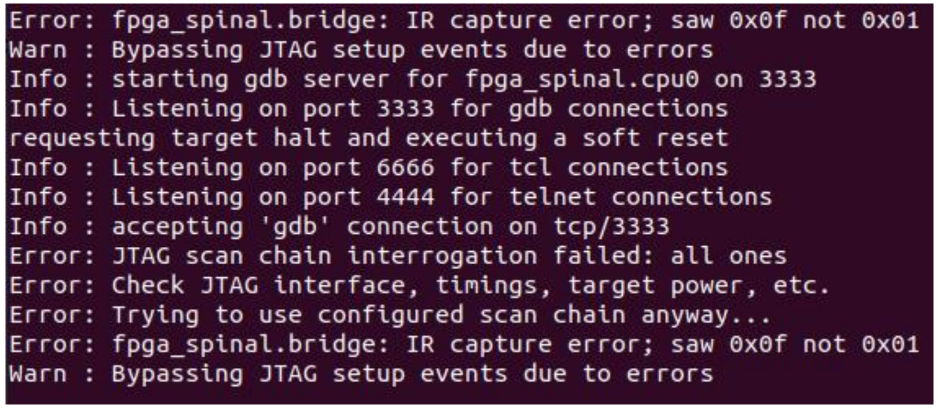
3. Debug using OpenOCD (6/6) Run software using GDB

The GDB terminal:



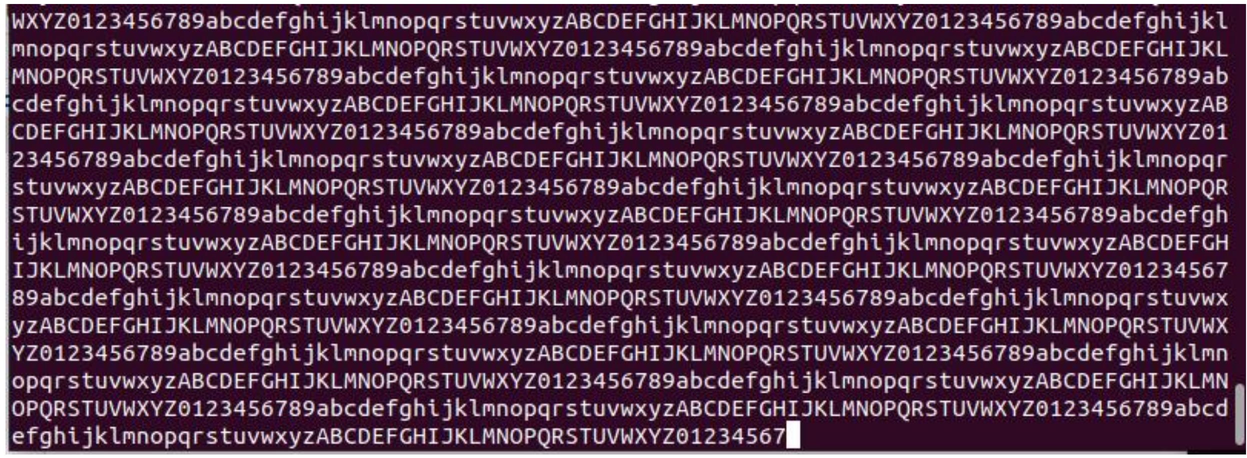
```
thuc@thuc-Ubuntu:~/VexRiscv$ export PATH=/opt/riscv/bin/:$PATH
thuc@thuc-Ubuntu:~/VexRiscv$ riscv64-unknown-elf-gdb src/test/resources/elf/uart.elf
GNU gdb (GDB) 8.0.50.20170724-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from src/test/resources/elf/uart.elf...done.
(gdb) target extended-remote localhost:3333
Remote debugging using localhost:3333
0x80000000 in ?? ()
(gdb) monitor reset halt
JTAG scan chain interrogation failed: all ones
Check JTAG interface, timings, target power, etc.
Trying to use configured scan chain anyway...
fpga_spinal.bridge: IR capture error; saw 0x0f not 0x01
Bypassing JTAG setup events due to errors
(gdb) load
Loading section .yolo, size 0x1b8 lma 0x0
Loading section .text, size 0x160 lma 0x40000000
Start address 0x138, load size 792
Transfer rate: 6336 bits in <1 sec, 396 bytes/write.
(gdb) continue
```

The OpenOCD terminal:



```
Error: fpga_spinal.bridge: IR capture error; saw 0x0f not 0x01
Warn : Bypassing JTAG setup events due to errors
Info : starting gdb server for fpga_spinal.cpu0 on 3333
Info : Listening on port 3333 for gdb connections
requesting target halt and executing a soft reset
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : accepting 'gdb' connection on tcp/3333
Error: JTAG scan chain interrogation failed: all ones
Error: Check JTAG interface, timings, target power, etc.
Error: Trying to use configured scan chain anyway...
Error: fpga_spinal.bridge: IR capture error; saw 0x0f not 0x01
Warn : Bypassing JTAG setup events due to errors
```

Printing in the Verilator terminal:



```
WXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijkl
mnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKL
MNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789ab
cdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzAB
CDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01
23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqr
stuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQR
STUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefgh
ijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGH
IJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567
89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwx
yzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWX
YZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmn
opqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
OPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcd
efghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567
```

Outline

1. Introduction
2. Git clone, make, and self-test
3. Debug using OpenOCD
4. Briey SoC and debug with Eclipse
5. Practice: make a clock using timer

4. Briey SoC and Eclipse (1/14) Software demo

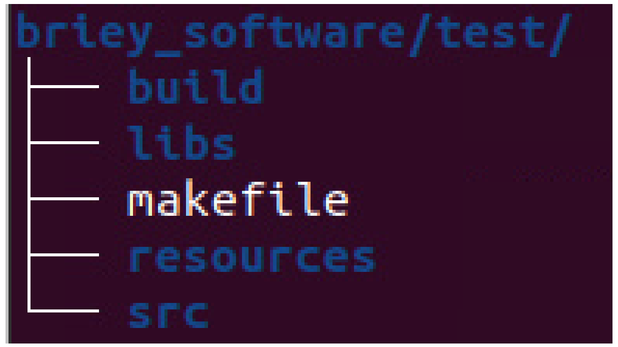
****Note:*** you can follow the guide **II.a)**, **II.b)**, and **II.c)** in:
https://thuchoang90.github.io/project/2020/07/23/VexRiscv

Now, we make and debug the **Briey** SoC.

First, we need to prepare the Briey's software demo:

From your home folder:
```
$ git clone https://github.com/thuchoang90/briey_software.git
```

There are several demos in the folder.
For this lecture, we will use the **test** folder:



```
briey_software/test/
├── build
├── libs
├── makefile
├── resources
└── src
```

4. Briey SoC and Eclipse (2/14) Make the Briey

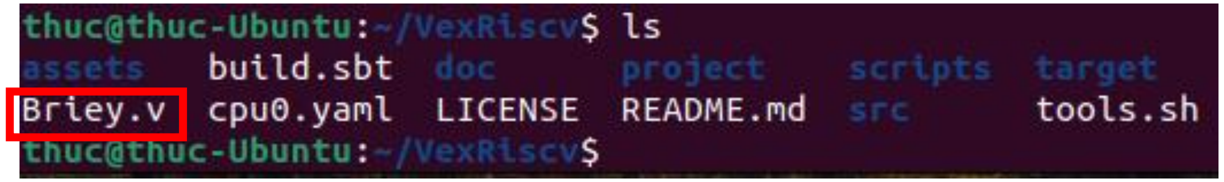
To run software on the **Briey**, similar to the previous section, we need to prepare three things: (1) the hardware *(or simulated hardware)*,
(2) OpenOCD, and
(3) GDB

First, for the hardware:

Go to your VexRiscv folder:
```
$ cd VexRiscv/
```

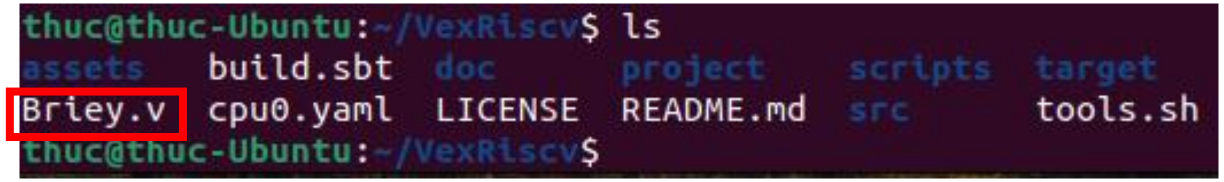
Generate the Briey SoC:
```
$ sbt "runMain vexriscv.demo.Briey"
```

After this, a Verilog file is generated, which contains all the Briey system that we just generated:



```
thuc@thuc-Ubuntu:~/VexRiscv$ ls
assets   build.sbt  doc      project    scripts  target
Briey.v  cpu0.yaml  LICENSE  README.md  src      tools.sh
thuc@thuc-Ubuntu:~/VexRiscv$
```

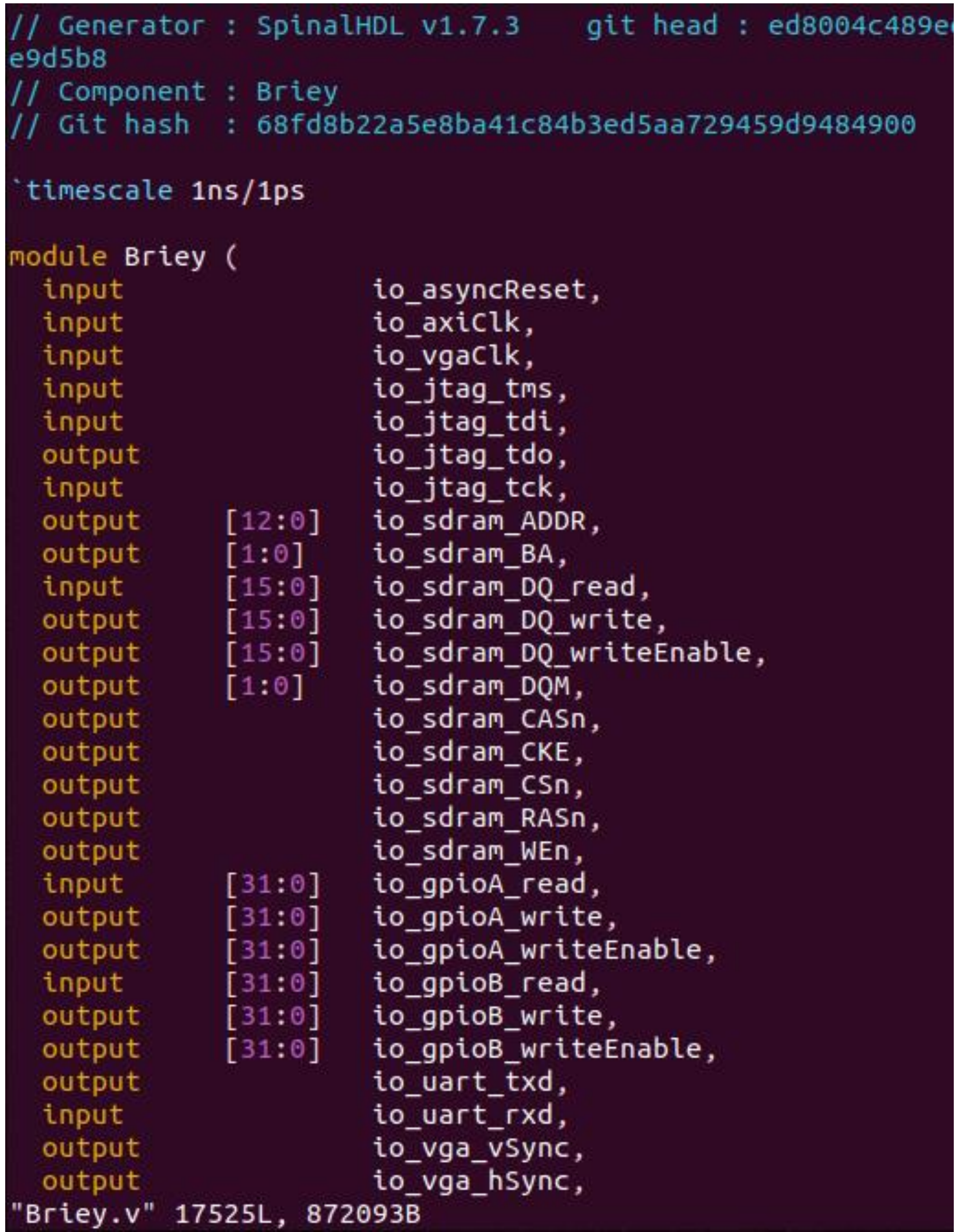
4. Briey SoC and Eclipse (3/14) Verilog file



```
thuc@thuc-Ubuntu:~/VexRiscv$ ls
assets    build.sbt  doc      project    scripts  target
Briey.v   cpu0.yaml  LICENSE  README.md  src      tools.sh
thuc@thuc-Ubuntu:~/VexRiscv$
```

This Verilog file contains the system, including the *VexRiscv core processor*, *on-chip memory*, *off-chip memory controller*, and *APB peripherals*.

This is also the file that we use for FPGA and VLSI implementations.



```
// Generator : SpinalHDL v1.7.3    git head : ed8004c489e
e9d5b8
// Component : Briey
// Git hash  : 68fd8b22a5e8ba41c84b3ed5aa729459d9484900

`timescale 1ns/1ps

module Briey (
  input                 io_asyncReset,
  input                 io_axiClk,
  input                 io_vgaClk,
  input                 io_jtag_tms,
  input                 io_jtag_tdi,
  output                io_jtag_tdo,
  input                 io_jtag_tck,
  output     [12:0]     io_sdram_ADDR,
  output     [1:0]      io_sdram_BA,
  input      [15:0]     io_sdram_DQ_read,
  output     [15:0]     io_sdram_DQ_write,
  output     [15:0]     io_sdram_DQ_writeEnable,
  output     [1:0]      io_sdram_DQM,
  output                io_sdram_CASn,
  output                io_sdram_CKE,
  output                io_sdram_CSn,
  output                io_sdram_RASn,
  output                io_sdram_WEn,
  input      [31:0]     io_gpioA_read,
  output     [31:0]     io_gpioA_write,
  output     [31:0]     io_gpioA_writeEnable,
  input      [31:0]     io_gpioB_read,
  output     [31:0]     io_gpioB_write,
  output     [31:0]     io_gpioB_writeEnable,
  output                io_uart_txd,
  input                 io_uart_rxd,
  output                io_vga_vSync,
  output                io_vga_hSync,
"Briey.v" 17525L, 872093B
```

4. Briey SoC and Eclipse (4/14) GDB → OpenOCD → Verilator

To simulate the Briey SoC via Verilator:

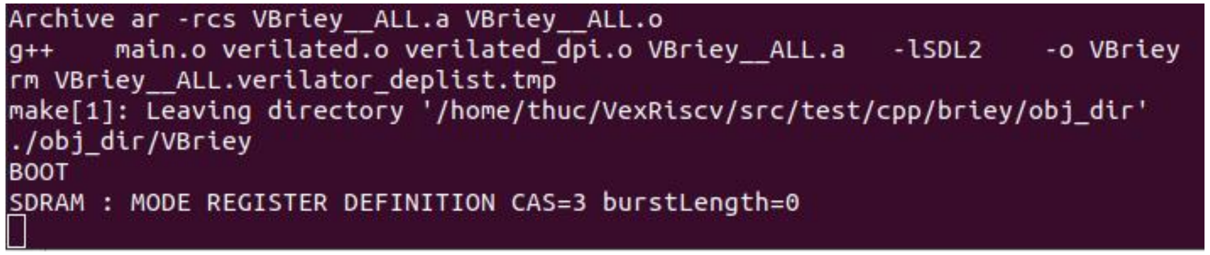
Go to your VexRiscv folder:

```
$ cd VexRiscv/
```

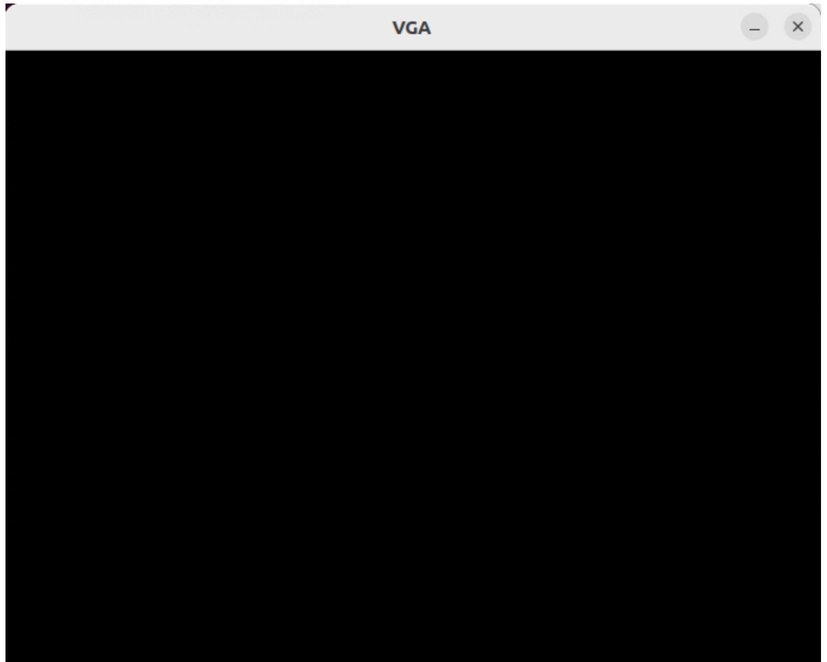
Run Verilator:

```
$ cd src/test/cpp/briey/
$ make clean run
```

The terminal after:



```
Archive ar -rcs VBriey__ALL.a VBriey__ALL.o
g++    main.o verilated.o verilated_dpi.o VBriey__ALL.a   -lSDL2    -o VBriey
rm VBriey__ALL.verilator_deplist.tmp
make[1]: Leaving directory '/home/thuc/VexRiscv/src/test/cpp/briey/obj_dir'
./obj_dir/VBriey
BOOT
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
```



Note: there will be a VGA box with a black screen popup. Don't close it. Just ignore it.

4. Briey SoC and Eclipse (5/14) GDB → OpenOCD → Verilator

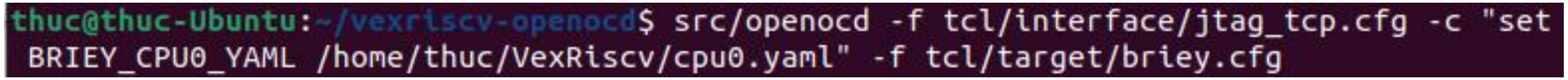
Now, for the OpenOCD, **keep** the previous Verilator terminal, **open** a new terminal, then:

Go to your vexriscv_openocd folder:

```
$ cd vexriscv_openocd/

$ src/openocd -f tcl/interface/jtag_tcp.cfg -c "set BRIEY_CPU0_YAML
<cpu0.yaml PATH>" -f tcl/target/briey.cfg
```

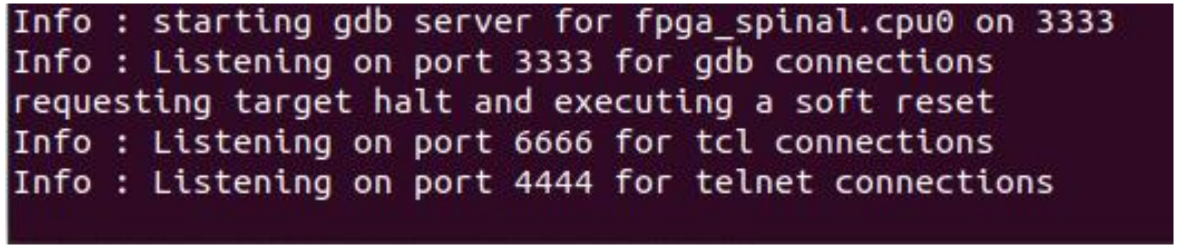
Where ***<cpu0.yaml PATH>*** points to the file cpu0.yaml in the **VexRiscv** folder.

For example:



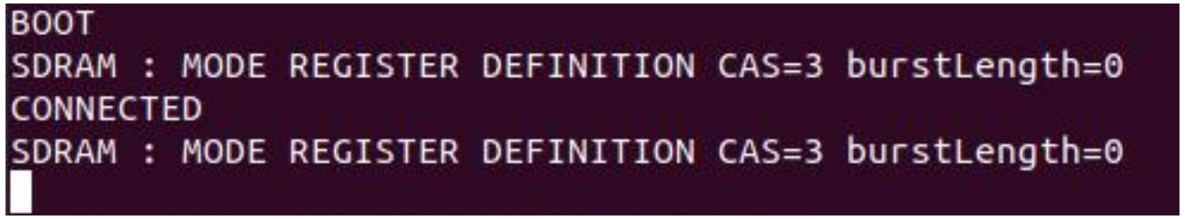
```
thuc@thuc-Ubuntu:~/vexriscv-openocd$ src/openocd -f tcl/interface/jtag_tcp.cfg -c "set
 BRIEY_CPU0_YAML /home/thuc/VexRiscv/cpu0.yaml" -f tcl/target/briey.cfg
```

The terminal after:



```
Info : starting gdb server for fpga_spinal.cpu0 on 3333
Info : Listening on port 3333 for gdb connections
requesting target halt and executing a soft reset
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

The previous Verilator terminal:



```
BOOT
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
CONNECTED
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
```

4. Briey SoC and Eclipse (6/14) GDB → OpenOCD → Verilator

Finally, to run the software, **keep** both previous terminals, **open** a new terminal, then:

Export the RISC-V toolchain:

```
$ export PATH=/opt/riscv/bin:$PATH
```

Go to your briey_software folder:

```
$ cd briey_software/
```

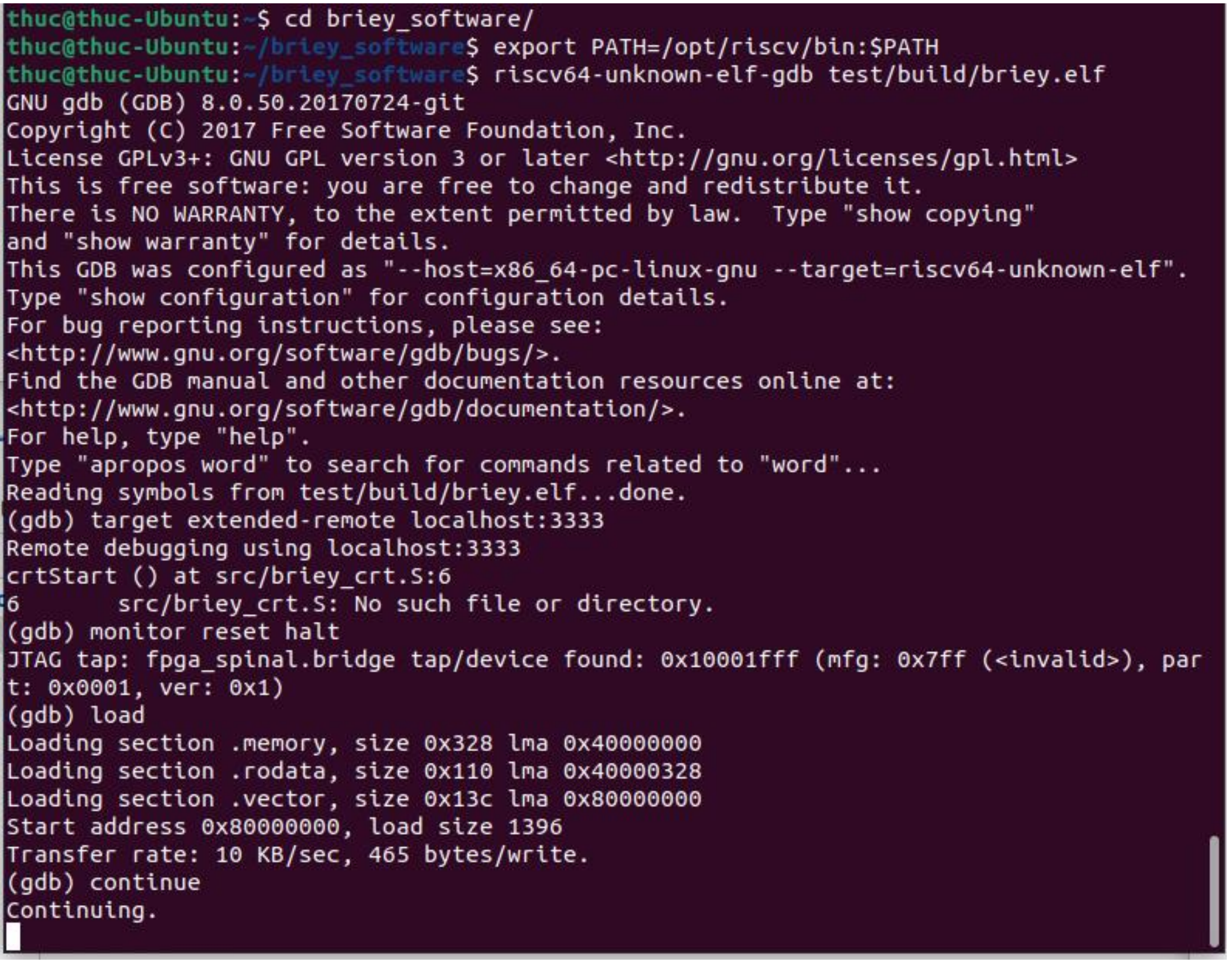
Run the software with RISC-V GDB:

```
$ riscv64-unknown-elf-gdb test/build/briey.elf
$ target extended-remote localhost:3333
$ monitor reset halt
$ load
$ continue
```

After this, it should prints messages to the Verilator terminal

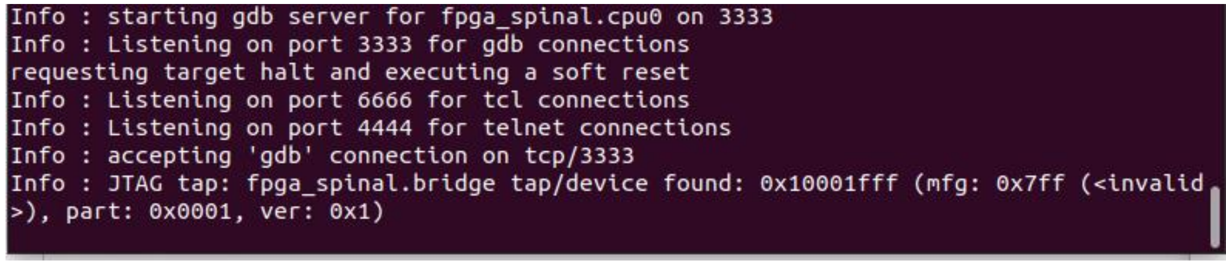
4. Briey SoC and Eclipse (7/14) GDB → OpenOCD → Verilator

The GDB terminal:



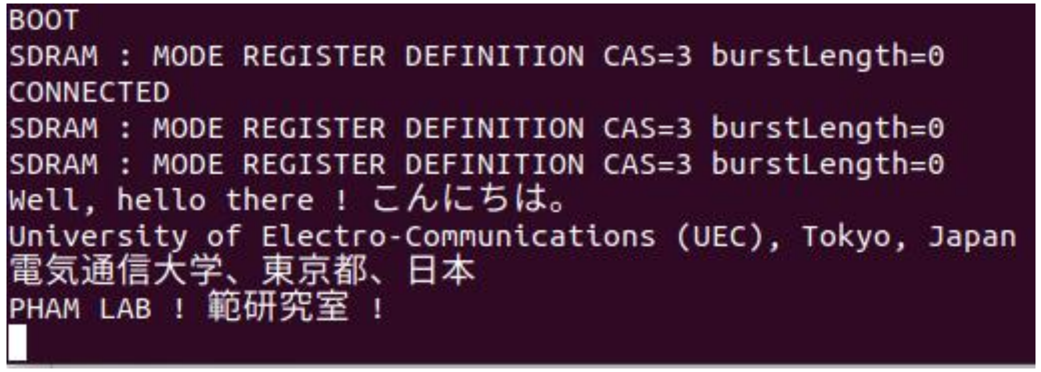
```
thuc@thuc-Ubuntu:~$ cd briey_software/
thuc@thuc-Ubuntu:~/briey_software$ export PATH=/opt/riscv/bin:$PATH
thuc@thuc-Ubuntu:~/briey_software$ riscv64-unknown-elf-gdb test/build/briey.elf
GNU gdb (GDB) 8.0.50.20170724-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from test/build/briey.elf...done.
(gdb) target extended-remote localhost:3333
Remote debugging using localhost:3333
crtStart () at src/briey_crt.S:6
6       src/briey_crt.S: No such file or directory.
(gdb) monitor reset halt
JTAG tap: fpga_spinal.bridge tap/device found: 0x10001fff (mfg: 0x7ff (<invalid>), par
t: 0x0001, ver: 0x1)
(gdb) load
Loading section .memory, size 0x328 lma 0x40000000
Loading section .rodata, size 0x110 lma 0x40000328
Loading section .vector, size 0x13c lma 0x80000000
Start address 0x80000000, load size 1396
Transfer rate: 10 KB/sec, 465 bytes/write.
(gdb) continue
Continuing.
```

The OpenOCD terminal:



```
Info : starting gdb server for fpga_spinal.cpu0 on 3333
Info : Listening on port 3333 for gdb connections
requesting target halt and executing a soft reset
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : accepting 'gdb' connection on tcp/3333
Info : JTAG tap: fpga_spinal.bridge tap/device found: 0x10001fff (mfg: 0x7ff (<invalid
>), part: 0x0001, ver: 0x1)
```

Printing in the Verilator terminal:



```
BOOT
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
CONNECTED
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
Well, hello there ! こんにちは。
University of Electro-Communications (UEC), Tokyo, Japan
電気通信大学、東京都、日本
PHAM LAB ! 範研究室 !
```

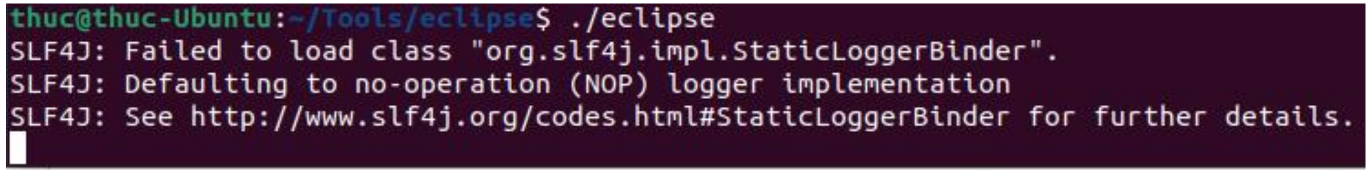
4. Briey SoC and Eclipse (8/14) Eclipse

Using **Eclipse**: Eclipse is a GUI for GDB, nothing more.

→ We still need to set up the Verilator and OpenOCD terminals like before.
This time replace the GDB terminal with the Eclipse application.

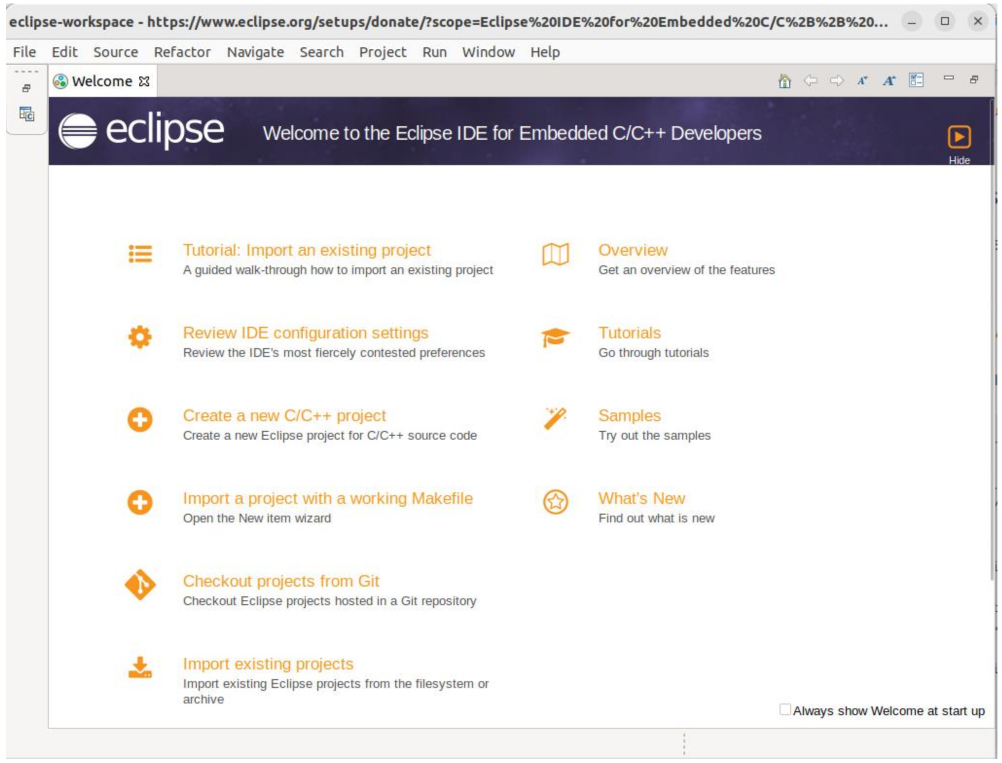
Open the Eclipse by going to the installed folder and:

```
$ ./eclipse
```



```
thuc@thuc-Ubuntu:~/Tools/eclipse$ ./eclipse
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

The GUI will show up:
(close the Welcome tab)



eclipse-workspace - https://www.eclipse.org/setups/donate/?scope=Eclipse%20IDE%20for%20Embedded%20C/C%2B%2B%20...

File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
eclipse
Hide

Welcome to the Eclipse IDE for Embedded C/C++ Developers

Tutorial: Import an existing project
A guided walk-through how to import an existing project

Overview
Get an overview of the features

Review IDE configuration settings
Review the IDE's most fiercely contested preferences

Tutorials
Go through tutorials

Create a new C/C++ project
Create a new Eclipse project for C/C++ source code

Samples
Try out the samples

Import a project with a working Makefile
Open the New item wizard

What's New
Find out what is new

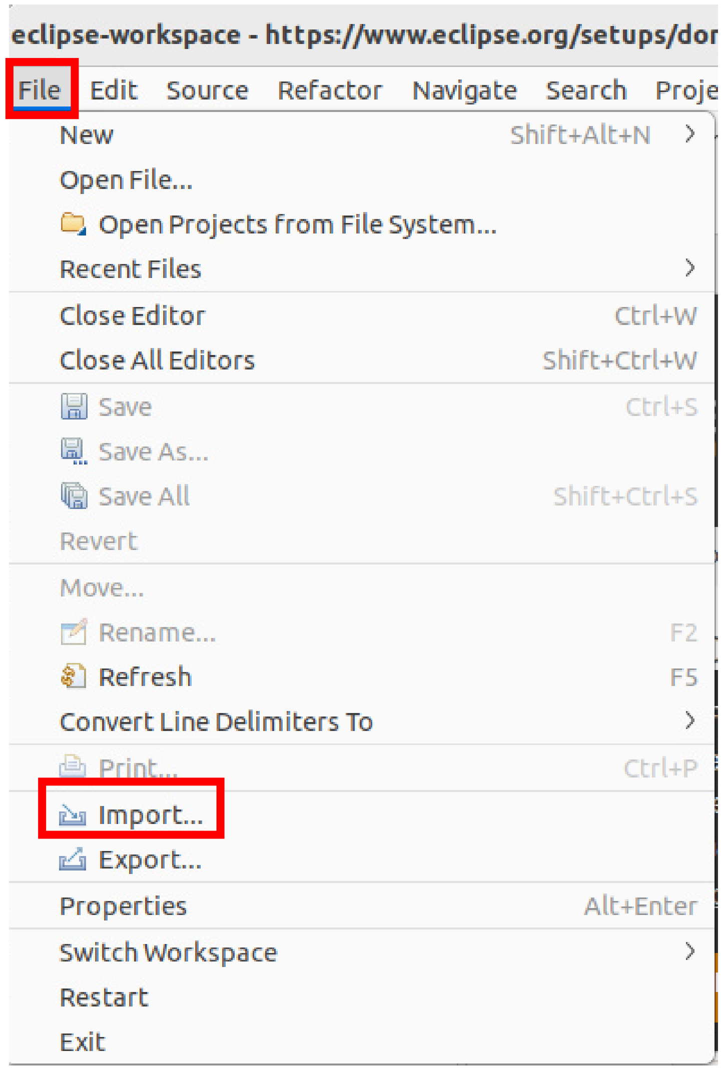
Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository

Import existing projects
Import existing Eclipse projects from the filesystem or archive

☐ Always show Welcome at start up

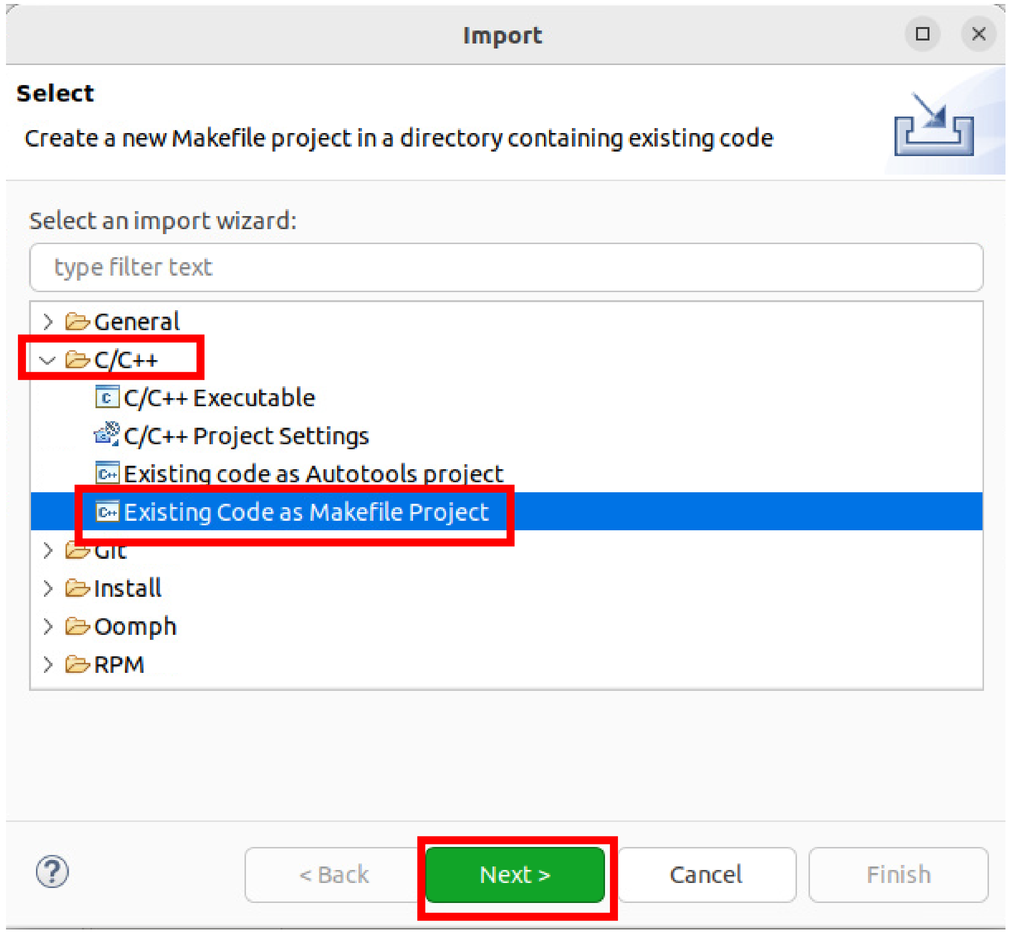
4. Briey SoC and Eclipse (9/14) Eclipse

To import the **test** project folder:



eclipse-workspace - https://www.eclipse.org/setups/dor

File → Import



Import

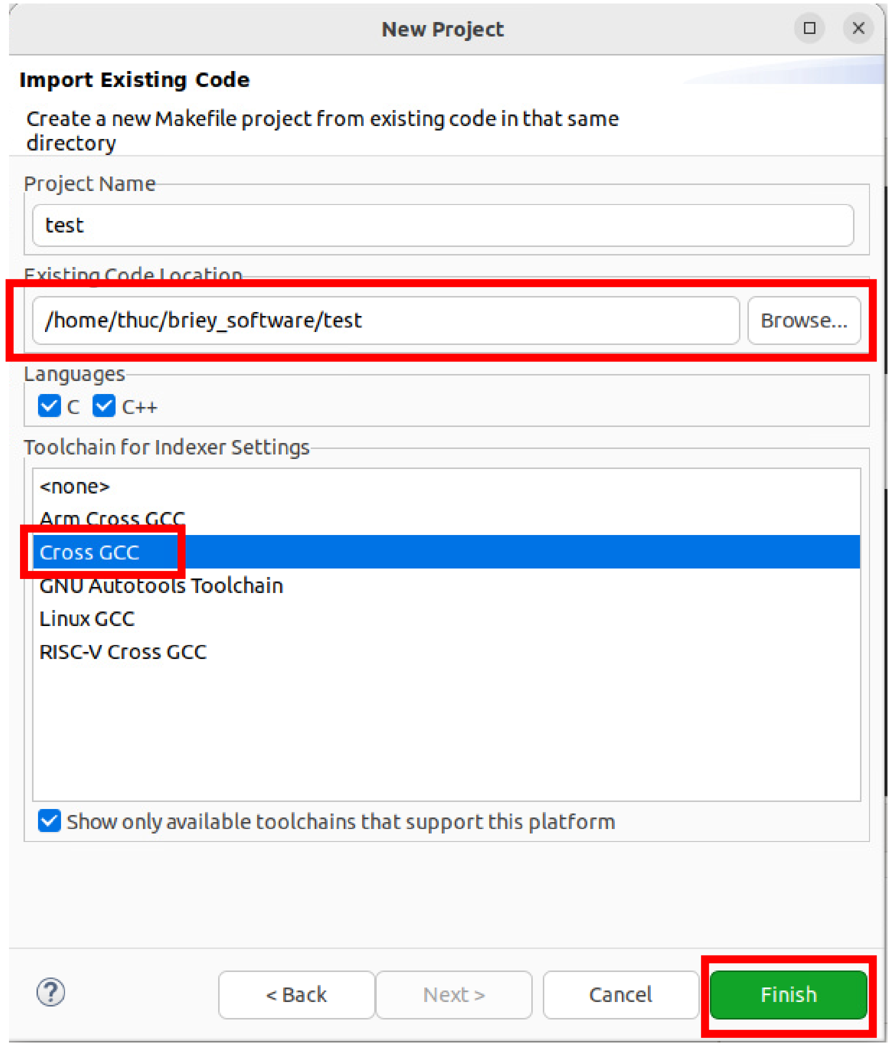
Select

Create a new Makefile project in a directory containing existing code

Select an import wizard:

C/C++ → Existing Code as Makefile Project → Next

4. Briey SoC and Eclipse (10/14) Eclipse



New Project

Import Existing Code

Create a new Makefile project from existing code in that same directory

Project Name: test

Existing Code Location: /home/thuc/briey_software/test

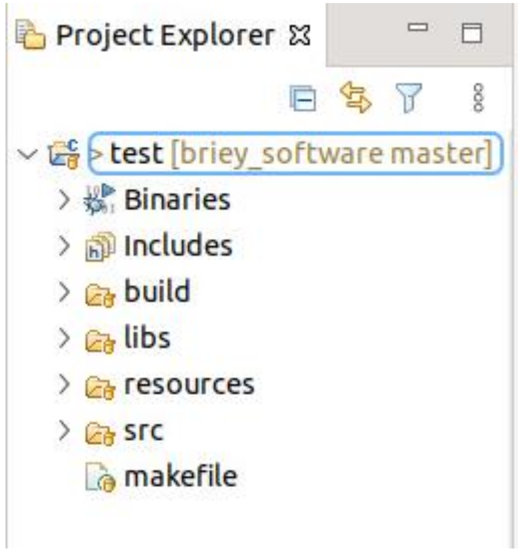
Languages: [✓ C] [✓ C++]

Toolchain for Indexer Settings:
- <none>
- Arm Cross GCC
- [✓ Cross GCC]
- GNU Autotools Toolchain
- Linux GCC
- RISC-V Cross GCC

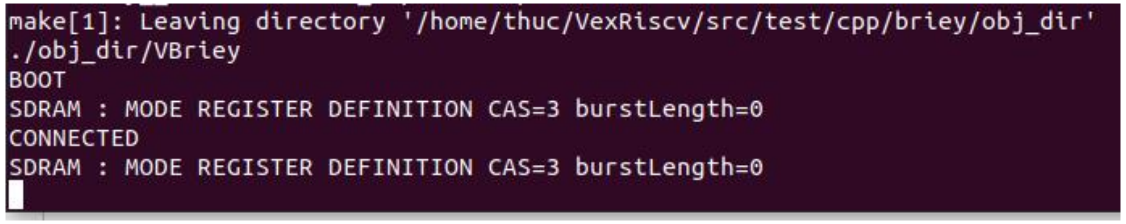
[✓] Show only available toolchains that support this platform

Browse to the **briey_software/test** → Choose the **Cross GCC** → Finish

Project **test** after imported:

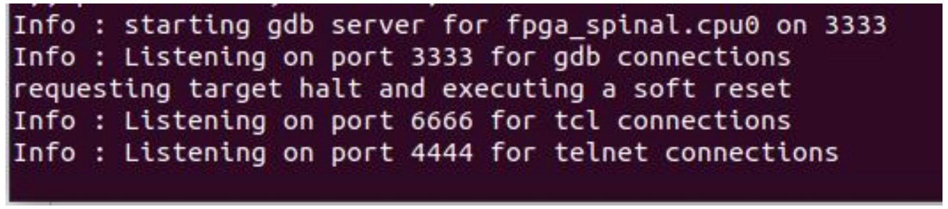


Now, like before, prepare the Verilator terminal:



```
make[1]: Leaving directory '/home/thuc/VexRiscv/src/test/cpp/briey/obj_dir'
./obj_dir/VBriey
BOOT
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
CONNECTED
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
```

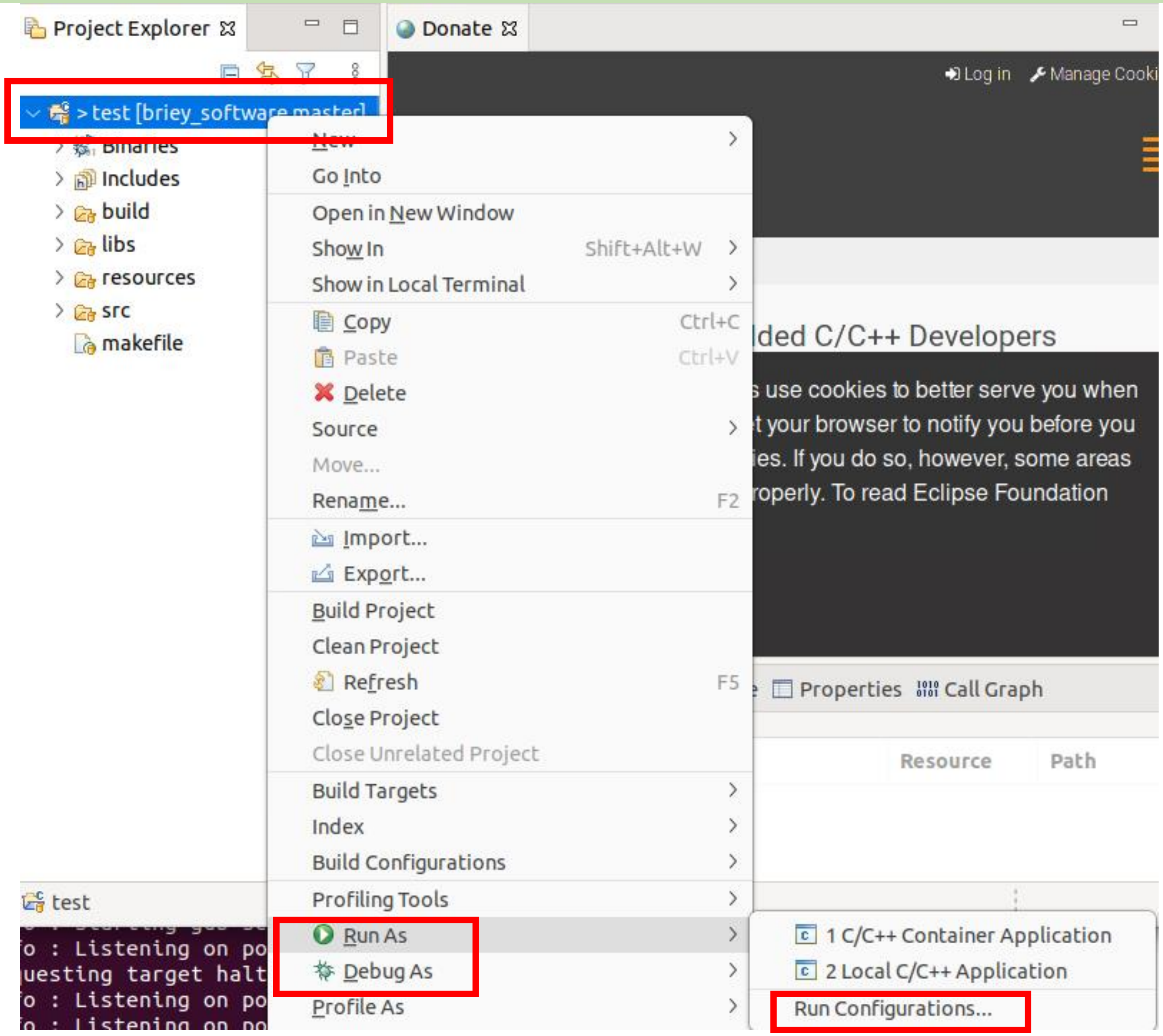
And the OpenOCD terminal:



```
Info : starting gdb server for fpga_spinal.cpu0 on 3333
Info : Listening on port 3333 for gdb connections
requesting target halt and executing a soft reset
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

4. Briey SoC and Eclipse (11/14) Eclipse

Right-click on the **test** project
→ Run as *(or Debug as)*
→ Run Configurations…



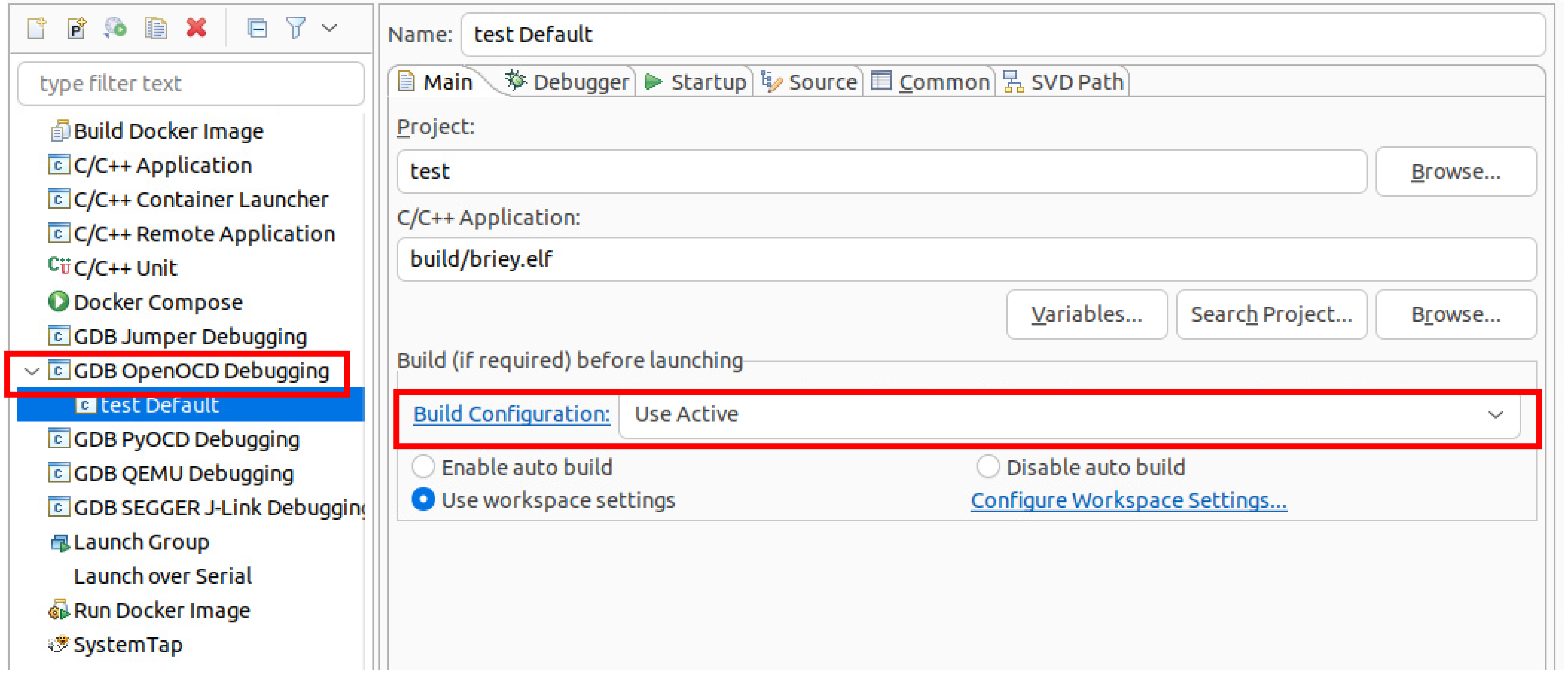
Ided C/C++ Developers

s use cookies to better serve you when
t your browser to notify you before you
ies. If you do so, however, some areas
roperly. To read Eclipse Foundation

```
o : Listening on po
uesting target halt
o : Listening on po
```

4. Briey SoC and Eclipse (12/14) Eclipse

Double-click on the **GDB OpenOCD Debugging**
→ Set the build configuration to **Use Active**



Name: test Default

Project: test

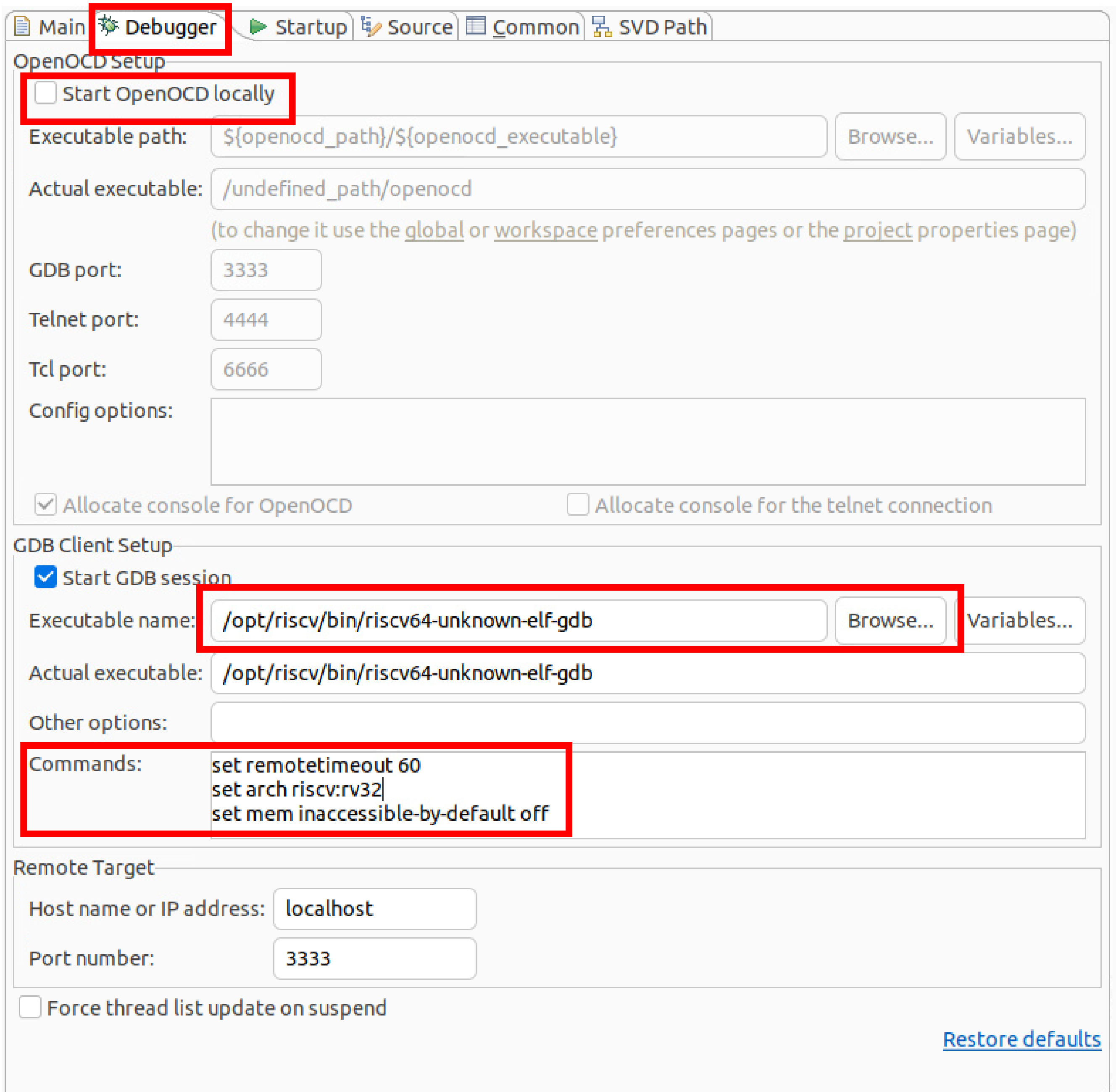
C/C++ Application: build/briey.elf

Build (if required) before launching

Build Configuration: Use Active

Enable auto build   Disable auto build
[✓ Use workspace settings]

4. Briey SoC and Eclipse (13/14) Eclipse



OpenOCD Setup

☐ Start OpenOCD locally

Executable path: ${openocd_path}/${openocd_executable} Browse... Variables...

Actual executable: /undefined_path/openocd

(to change it use the global or workspace preferences pages or the project properties page)

GDB port: 3333

Telnet port: 4444

Tcl port: 6666

Config options:

☑ Allocate console for OpenOCD ☐ Allocate console for the telnet connection

GDB Client Setup

☑ Start GDB session

Executable name: /opt/riscv/bin/riscv64-unknown-elf-gdb Browse... Variables...

Actual executable: /opt/riscv/bin/riscv64-unknown-elf-gdb

Other options:

Commands:

```
set remotetimeout 60
set arch riscv:rv32
set mem inaccessible-by-default off
```

Remote Target

Host name or IP address: localhost

Port number: 3333

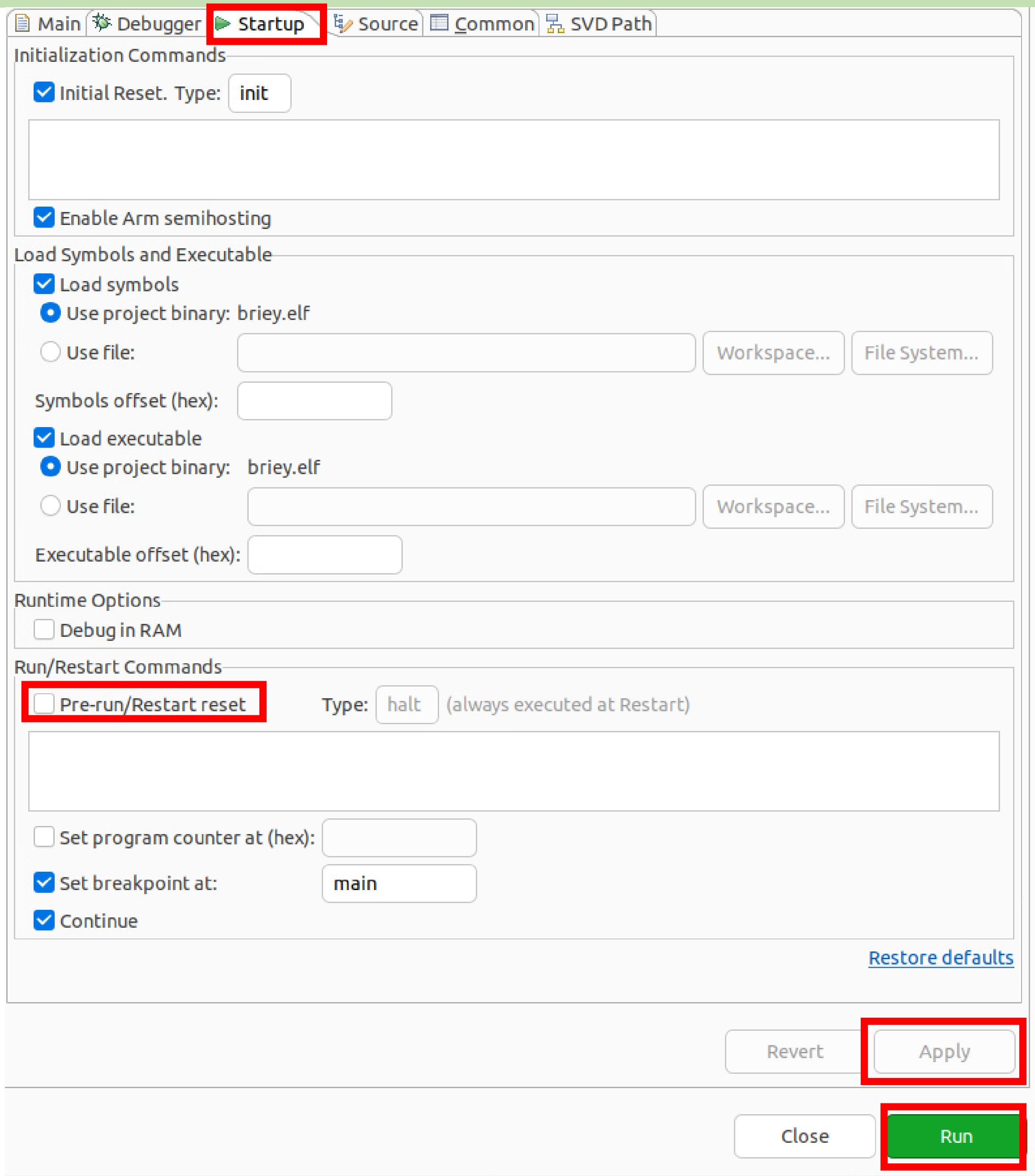
☐ Force thread list update on suspend

Restore defaults

Go the the **Debugger** tab
→ Disable the **Start OpenOCD locally**
→ Browse to the RISC-V GDB toolchain
→ Set commands:

```
set remotetimeout 60
set arch riscv:rv32
set mem inaccessible-by-default off
```

4. Briey SoC and Eclipse (14/14) Eclipse



Main | Debugger | Startup | Source | Common | SVD Path

Initialization Commands

- [✓] Initial Reset. Type: init
- [✓] Enable Arm semihosting

Load Symbols and Executable

- [✓] Load symbols
- ● Use project binary: briey.elf
- ○ Use file:
- Symbols offset (hex):
- [✓] Load executable
- ● Use project binary: briey.elf
- ○ Use file:
- Executable offset (hex):

Runtime Options

- [ ] Debug in RAM

Run/Restart Commands

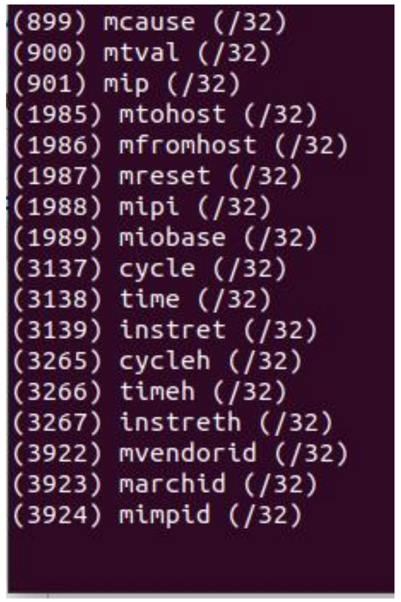
- [ ] Pre-run/Restart reset Type: halt (always executed at Restart)
- [ ] Set program counter at (hex):
- [✓] Set breakpoint at: main
- [✓] Continue

Go the the **Startup** tab

→ Disable the **Pre-run/Restart reset**

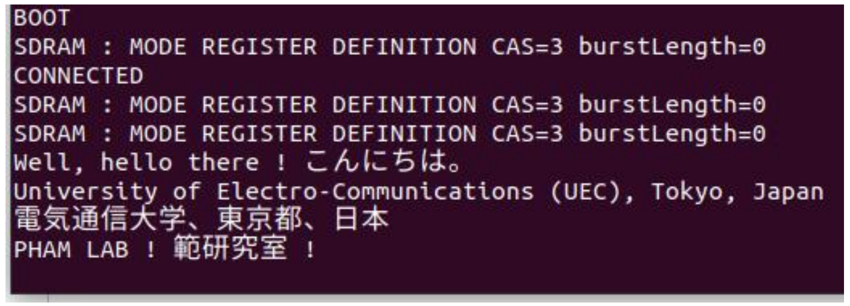
→ Apply → Run

After Run:



```
(899) mcause (/32)
(900) mtval (/32)
(901) mip (/32)
(1985) mtohost (/32)
(1986) mfromhost (/32)
(1987) mreset (/32)
(1988) mipi (/32)
(1989) miobase (/32)
(3137) cycle (/32)
(3138) time (/32)
(3139) instret (/32)
(3265) cycleh (/32)
(3266) timeh (/32)
(3267) instreth (/32)
(3922) mvendorid (/32)
(3923) marchid (/32)
(3924) mimpid (/32)
```

OpenOCD terminal



```
BOOT
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
CONNECTED
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
Well, hello there ! こんにちは。
University of Electro-Communications (UEC), Tokyo, Japan
電気通信大学、東京都、日本
PHAM LAB ! 範研究室 !
```

Verilator terminal

* Now, let's make some changes in the **test/src/briey_main.c** and re-run again.

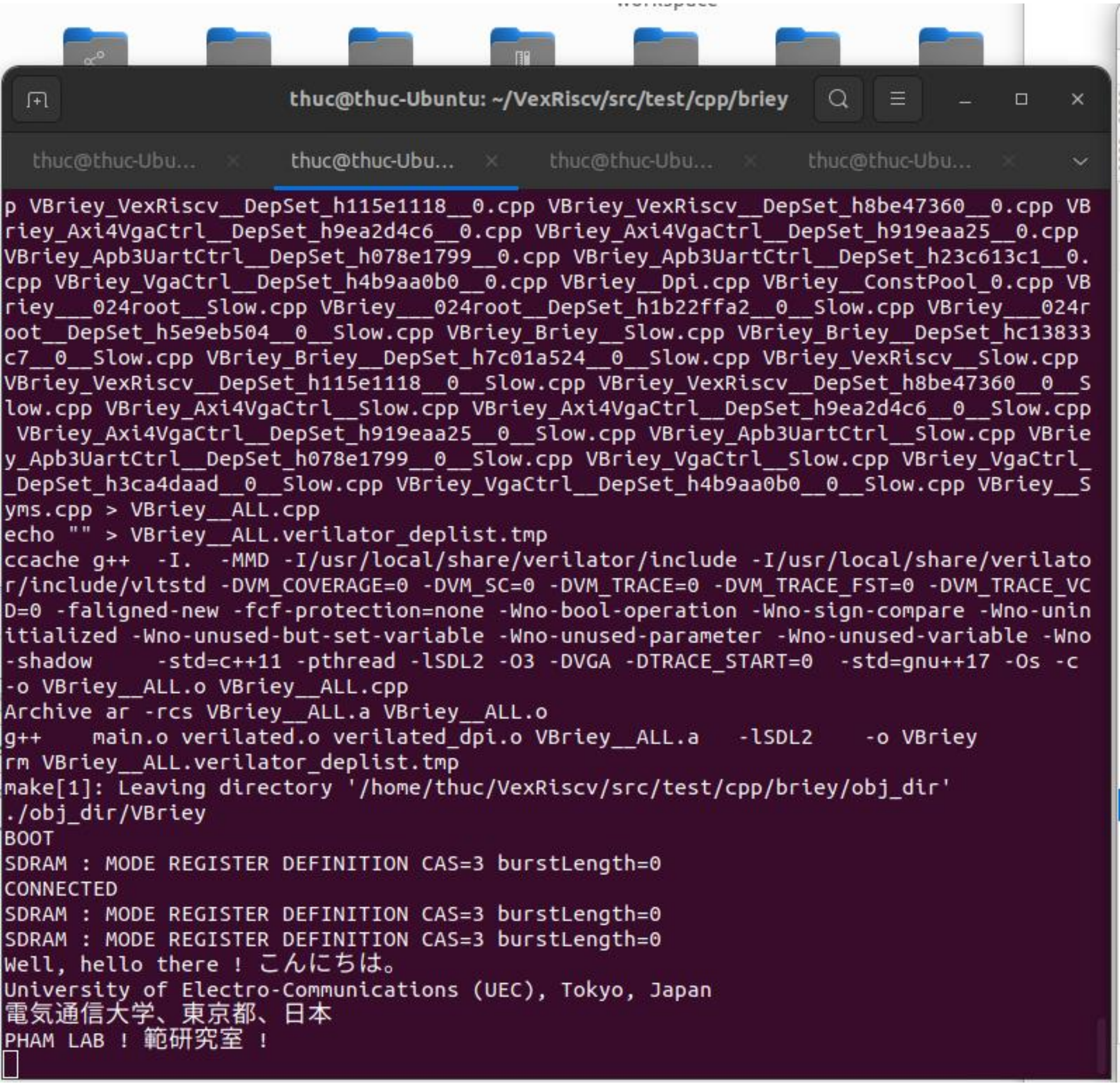
国立大学法人 電気通信大学 The University of Electro-Communications

Pham Laboratory
Integrated circuit design laboratory

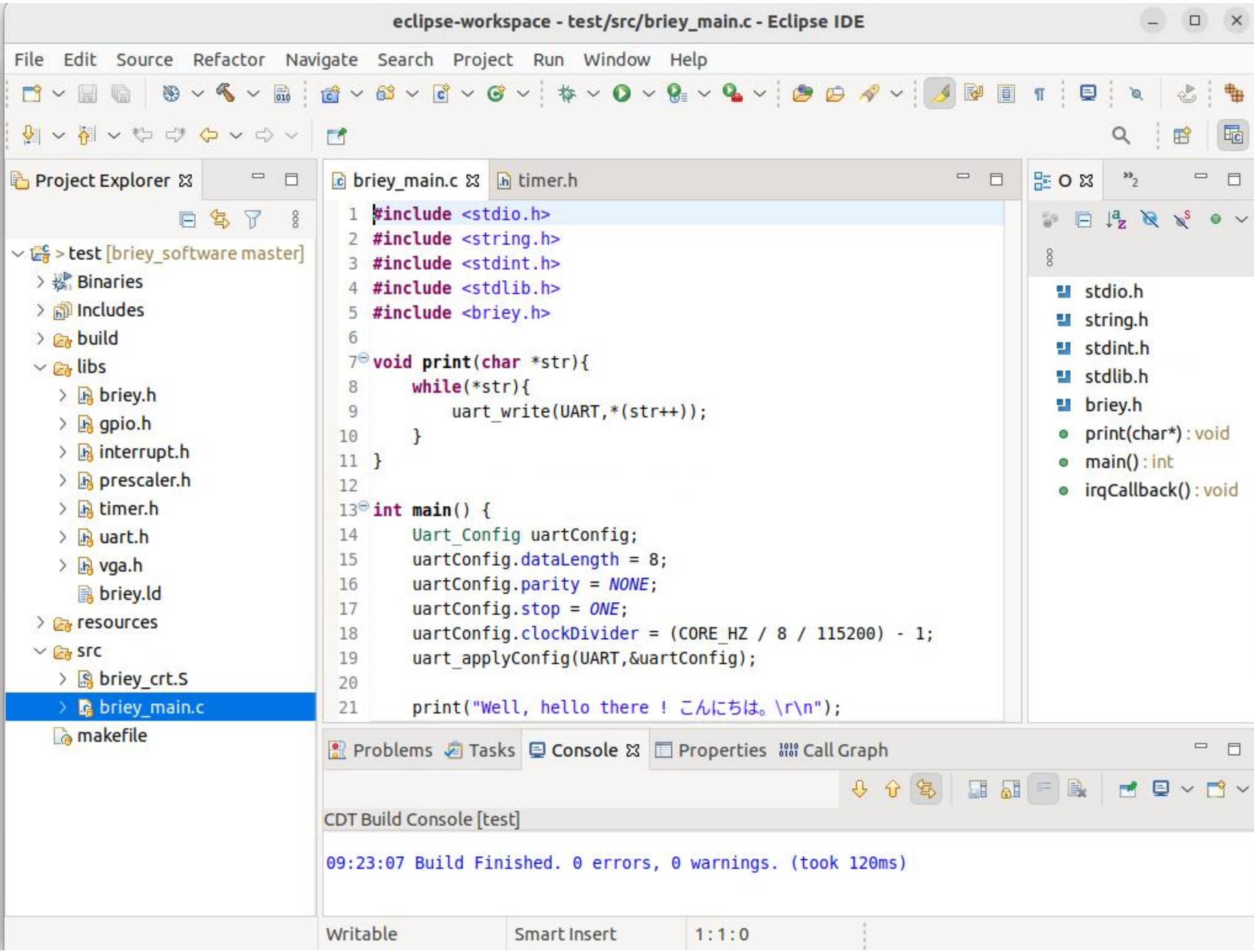
Outline

1. Introduction
2. Git clone, make, and self-test
3. Debug using OpenOCD
4. Briey SoC and debug with Eclipse
5. Practice: make a clock using timer

5. Practice: make a clock (1/2)



```
p VBriey_VexRiscv__DepSet_h115e1118__0.cpp VBriey_VexRiscv__DepSet_h8be47360__0.cpp VB
riey_Axi4VgaCtrl__DepSet_h9ea2d4c6__0.cpp VBriey_Axi4VgaCtrl__DepSet_h919eaa25__0.cpp
VBriey_Apb3UartCtrl__DepSet_h078e1799__0.cpp VBriey_Apb3UartCtrl__DepSet_h23c613c1__0.
cpp VBriey_VgaCtrl__DepSet_h4b9aa0b0__0.cpp VBriey__Dpi.cpp VBriey__ConstPool_0.cpp VB
riey___024root__Slow.cpp VBriey___024root__DepSet_h1b22ffa2__0__Slow.cpp VBriey___024r
oot__DepSet_h5e9eb504__0__Slow.cpp VBriey_Briey__Slow.cpp VBriey_Briey__DepSet_hc13833
c7__0__Slow.cpp VBriey_Briey__DepSet_h7c01a524__0__Slow.cpp VBriey_VexRiscv__Slow.cpp
VBriey_VexRiscv__DepSet_h115e1118__0__Slow.cpp VBriey_VexRiscv__DepSet_h8be47360__0__S
low.cpp VBriey_Axi4VgaCtrl__Slow.cpp VBriey_Axi4VgaCtrl__DepSet_h9ea2d4c6__0__Slow.cpp
 VBriey_Axi4VgaCtrl__DepSet_h919eaa25__0__Slow.cpp VBriey_Apb3UartCtrl__Slow.cpp VBrie
y_Apb3UartCtrl__DepSet_h078e1799__0__Slow.cpp VBriey_VgaCtrl__Slow.cpp VBriey_VgaCtrl_
_DepSet_h3ca4daad__0__Slow.cpp VBriey_VgaCtrl__DepSet_h4b9aa0b0__0__Slow.cpp VBriey__S
yms.cpp > VBriey__ALL.cpp
echo "" > VBriey__ALL.verilator_deplist.tmp
ccache g++  -I.  -MMD -I/usr/local/share/verilator/include -I/usr/local/share/verilato
r/include/vltstd -DVM_COVERAGE=0 -DVM_SC=0 -DVM_TRACE=0 -DVM_TRACE_FST=0 -DVM_TRACE_VC
D=0 -faligned-new -fcf-protection=none -Wno-bool-operation -Wno-sign-compare -Wno-unin
itialized -Wno-unused-but-set-variable -Wno-unused-parameter -Wno-unused-variable -Wno
-shadow     -std=c++11 -pthread -lSDL2 -O3 -DVGA -DTRACE_START=0  -std=gnu++17 -Os -c
-o VBriey__ALL.o VBriey__ALL.cpp
Archive ar -rcs VBriey__ALL.a VBriey__ALL.o
g++    main.o verilated.o verilated_dpi.o VBriey__ALL.a   -lSDL2    -o VBriey
rm VBriey__ALL.verilator_deplist.tmp
make[1]: Leaving directory '/home/thuc/VexRiscv/src/test/cpp/briey/obj_dir'
./obj_dir/VBriey
BOOT
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
CONNECTED
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
SDRAM : MODE REGISTER DEFINITION CAS=3 burstLength=0
Well, hello there ! こんにちは。
University of Electro-Communications (UEC), Tokyo, Japan
電気通信大学、東京都、日本
PHAM LAB ! 範研究室 !
```



```c
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
#include <briey.h>

void print(char *str){
    while(*str){
        uart_write(UART,*(str++));
    }
}

int main() {
    Uart_Config uartConfig;
    uartConfig.dataLength = 8;
    uartConfig.parity = NONE;
    uartConfig.stop = ONE;
    uartConfig.clockDivider = (CORE_HZ / 8 / 115200) - 1;
    uart_applyConfig(UART,&uartConfig);

    print("Well, hello there ! こんにちは。\r\n");
```

```
CDT Build Console [test]

09:23:07 Build Finished. 0 errors, 0 warnings. (took 120ms)
```

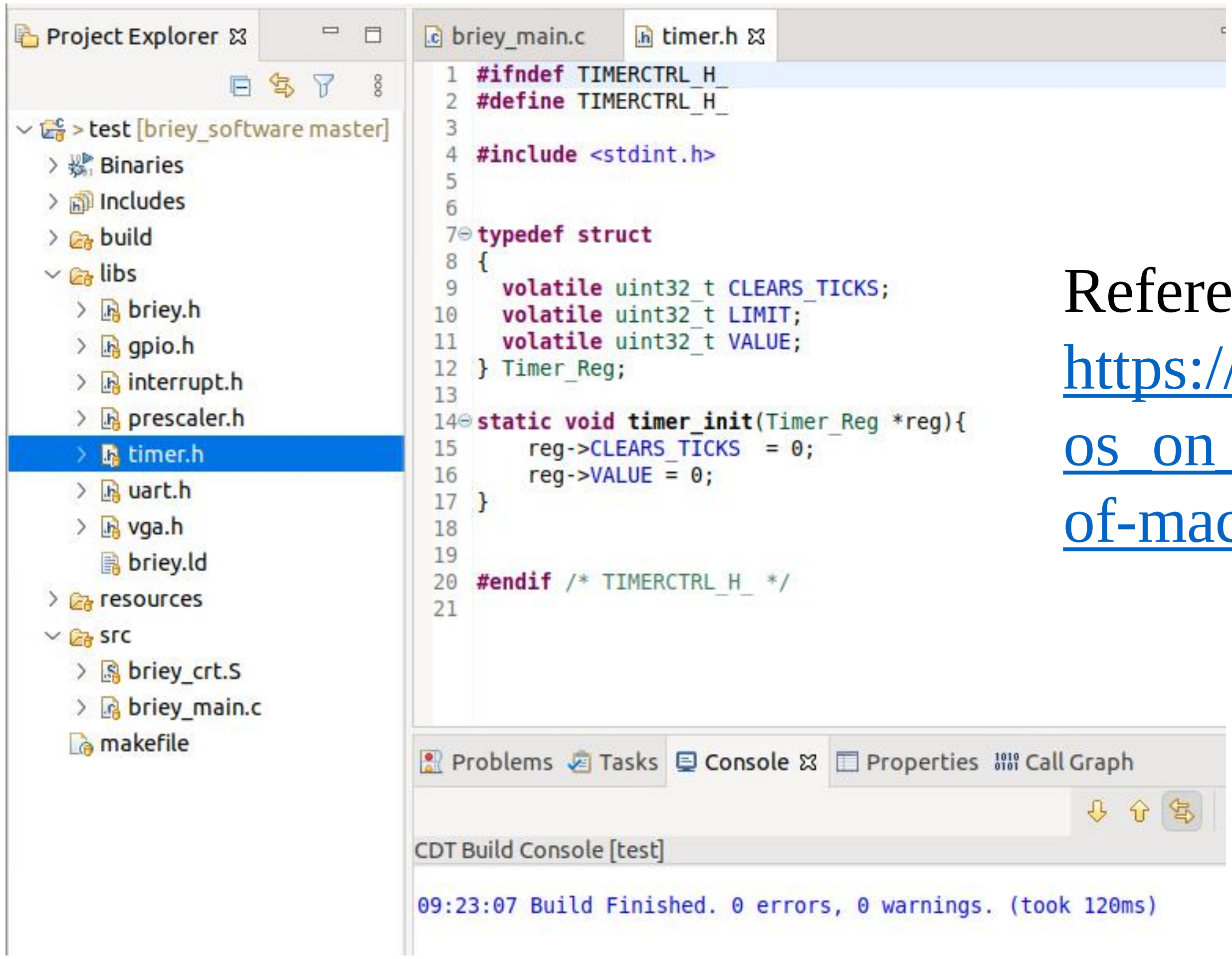
So now we can write a C/C++ code, run/debug it, and see the changes in the Verilator terminal.

5. Practice: make a clock (2/2)

For practice:

- Let's make it prints a real-time clock to the UART

Hint: use the timer peripheral to set up a 1-second tick, then adjust the print() accordingly.



```c
#ifndef TIMERCTRL_H_
#define TIMERCTRL_H_

#include <stdint.h>


typedef struct
{
  volatile uint32_t CLEARS_TICKS;
  volatile uint32_t LIMIT;
  volatile uint32_t VALUE;
} Timer_Reg;

static void timer_init(Timer_Reg *reg){
    reg->CLEARS_TICKS  = 0;
    reg->VALUE = 0;
}


#endif /* TIMERCTRL_H_ */
```

```
09:23:07 Build Finished. 0 errors, 0 warnings. (took 120ms)
```

Reference on how to use timer: https://hackmd.io/@oscarshiang/freertos_on_riscv#Use-Briey-timer-instead-of-machine-timer

国立大学法人 電気通信大学 The University of Electro-Communications

Pham Laboratory
Integrated circuit design laboratory

THANK YOU

2023/9