Module: R5: RV-fpga
Section: Installations **Task:** Tools

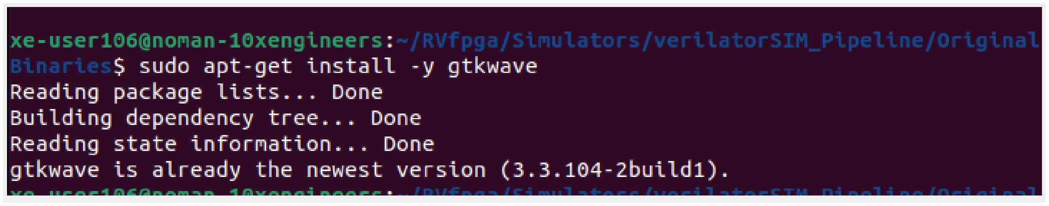
Task 1.4
RVfpga-Trace

➢ **Testing:**

■ **RVfpga-Trace**

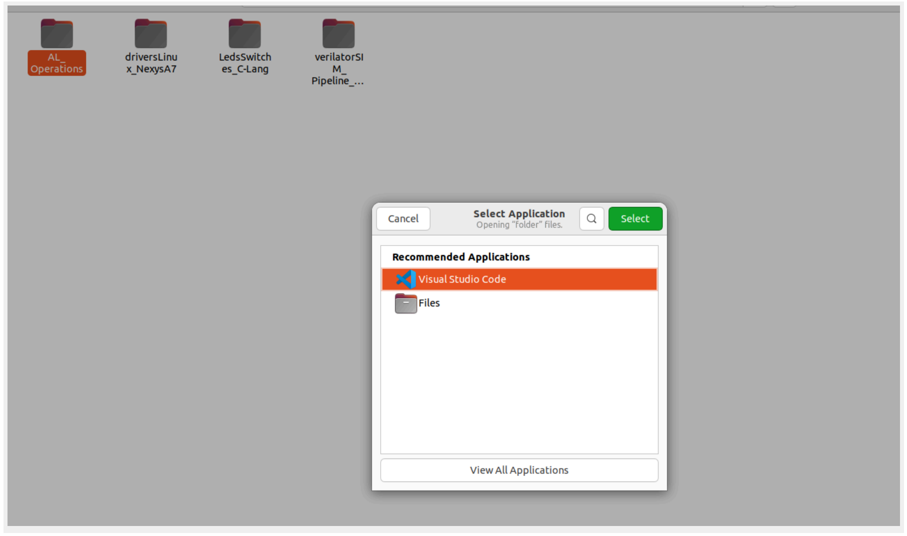
1. Install GTKwave by running the following command:

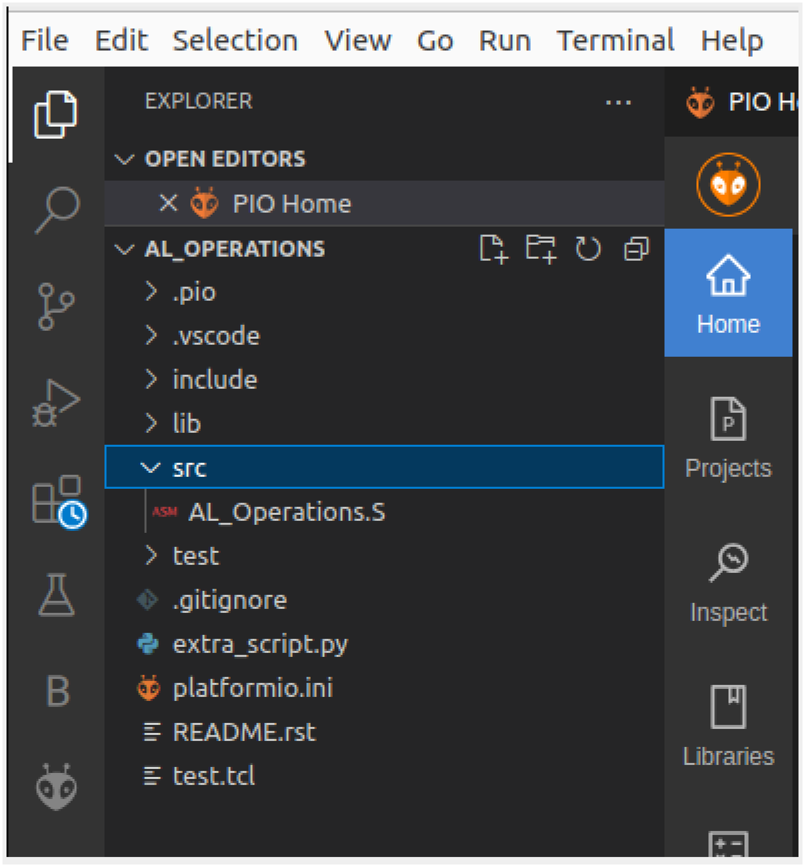
```
sudo apt-get install -y gtkwave
```



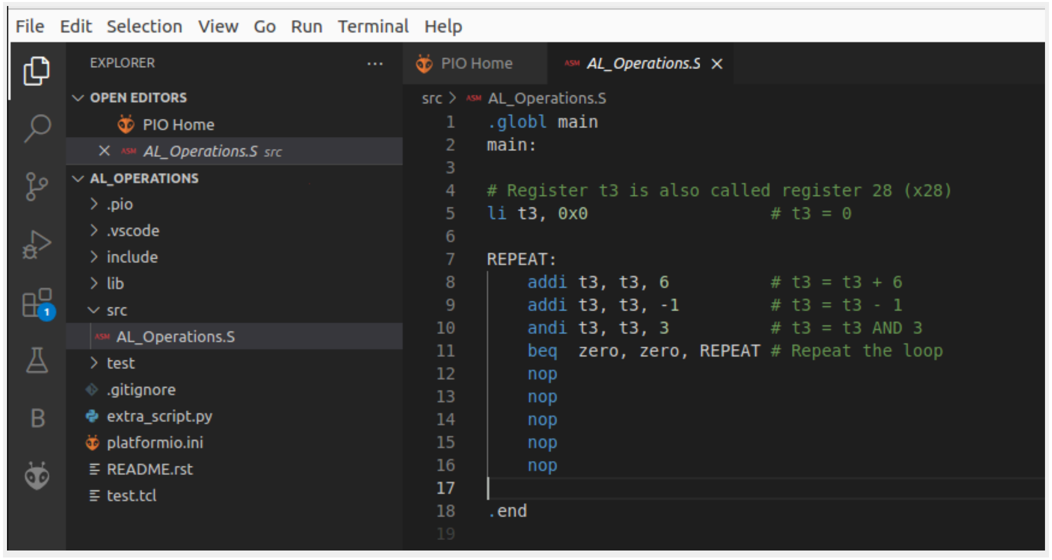
```
xe-user106@noman-10xengineers:~/RVfpga/Simulators/verilatorSIM_Pipeline/Original
Binaries$ sudo apt-get install -y gtkwave
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
gtkwave is already the newest version (3.3.104-2build1).
```

2. Open the specified folder of example program in VS Code:





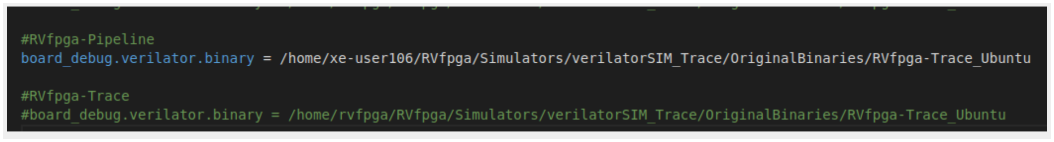
3. PlatformIO will now open this program, which includes three assembly arithmetic-logic instructions (addition, subtraction, and logical and) on the same register, t3 (also called x28), within an infinite loop. We can view the program by expanding the src folder and double-clicking on **AL_Operations.S**.



```
.globl main
main:

# Register t3 is also called register 28 (x28)
li t3, 0x0                  # t3 = 0

REPEAT:
    addi t3, t3, 6          # t3 = t3 + 6
    addi t3, t3, -1         # t3 = t3 - 1
    andi t3, t3, 3          # t3 = t3 AND 3
    beq  zero, zero, REPEAT # Repeat the loop
    nop
    nop
    nop
    nop
    nop

.end
```

4. Open file **platformio.ini**. Then, establish the path to the provided RVfpga-Trace simulator binary by editing the following line (replaced [Path-To-RVfpga] with the appropriate path in my system:

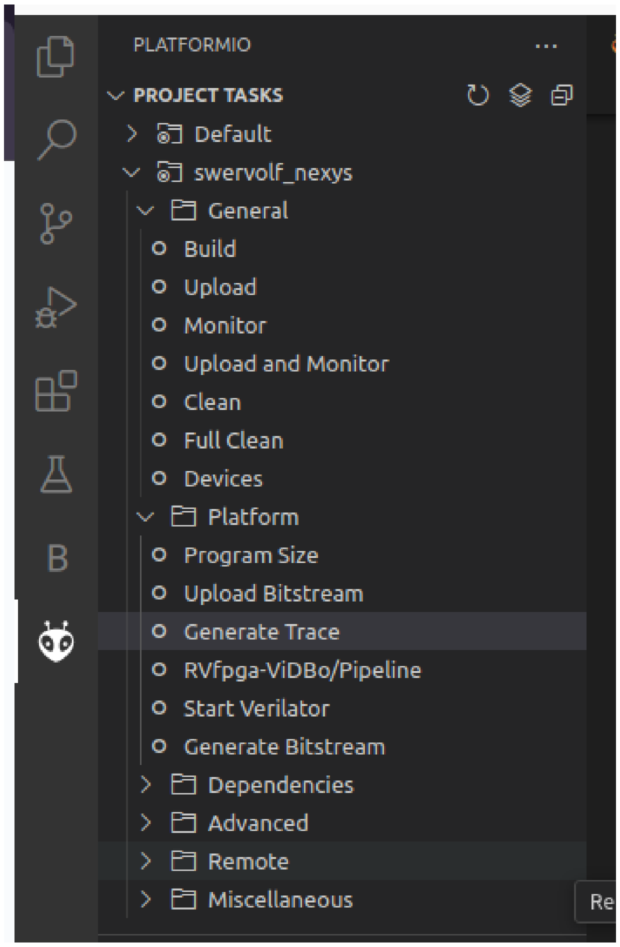
   ```
   board_debug.verilator.binary =
   /home/xe-user106/RVfpga/Simulators/verilatorSIM_Trace/Origin
   alBinaries/RVfpga-Trace_Ubuntu
   ```

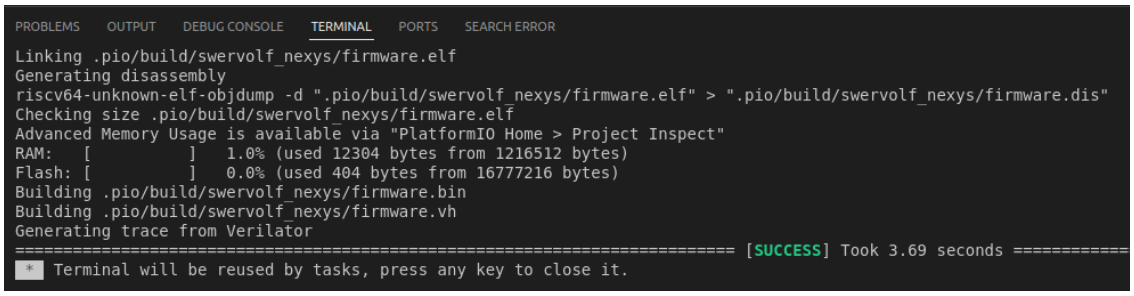
   ```
   #RVfpga-Pipeline
   board_debug.verilator.binary = /home/xe-user106/RVfpga/Simulators/verilatorSIM_Trace/OriginalBinaries/RVfpga-Trace_Ubuntu

   #RVfpga-Trace
   #board_debug.verilator.binary = /home/rvfpga/RVfpga/Simulators/verilatorSIM_Trace/OriginalBinaries/RVfpga-Trace_Ubuntu
   ```

5. Executed the RVfpga-Trace simulator from PlatformIO:
   - Click on the PlatformIO button on the left side.
   - Expand Project Tasks > env:swervolf_nexys > Platform and clicked on **Generare Trace**.
   - We have to provide execution rights to the binary by running command

     ```
     chmod +x RVfpga-Pipeline_Ubuntu
     ```



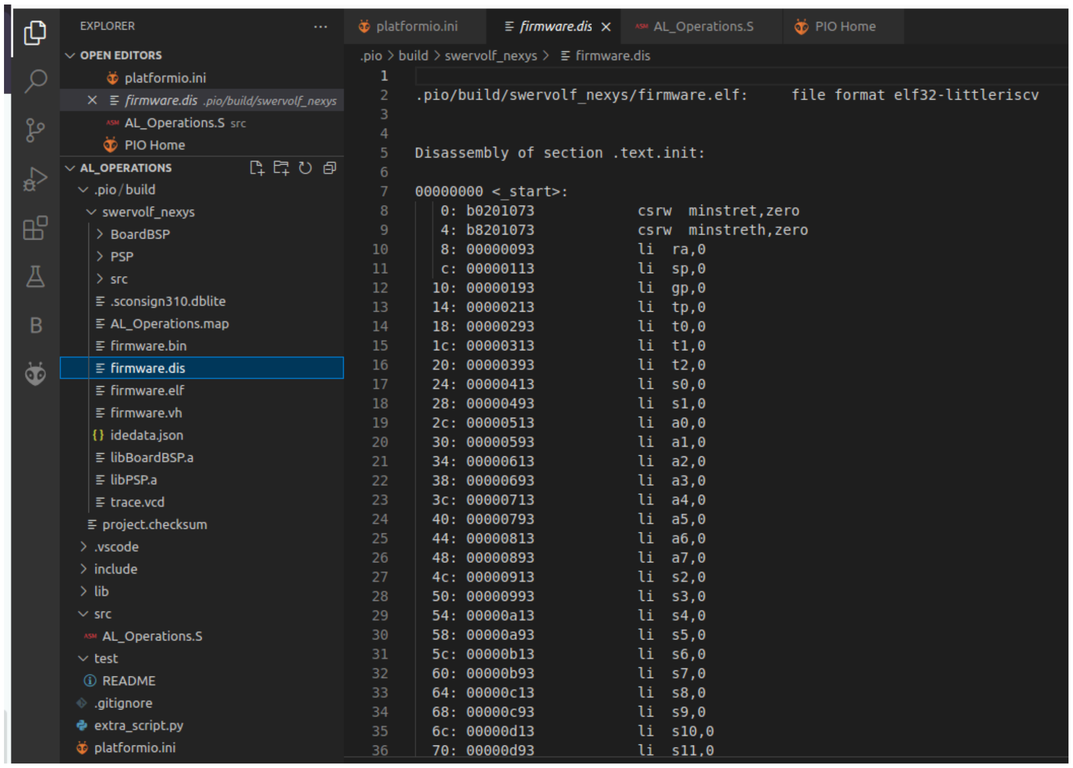
This first compiles the program and then launches the Verilator simulation of the RVfpga SoC running this program.



```
Linking .pio/build/swervolf_nexys/firmware.elf
Generating disassembly
riscv64-unknown-elf-objdump -d ".pio/build/swervolf_nexys/firmware.elf" > ".pio/build/swervolf_nexys/firmware.dis"
Checking size .pio/build/swervolf_nexys/firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM:   [          ]   1.0% (used 12304 bytes from 1216512 bytes)
Flash: [          ]   0.0% (used 404 bytes from 16777216 bytes)
Building .pio/build/swervolf_nexys/firmware.bin
Building .pio/build/swervolf_nexys/firmware.vh
Generating trace from Verilator
========================================================================== [SUCCESS] Took 3.69 seconds ============
 *  Terminal will be reused by tasks, press any key to close it.
```

6. A few seconds after the previous step, the **AL_Operations** program is compiled and file **trace.vcd** is generated inside folder **RVfpga/Chapters/01/AL_Operations/.pio/build/swervolf_nexys**.

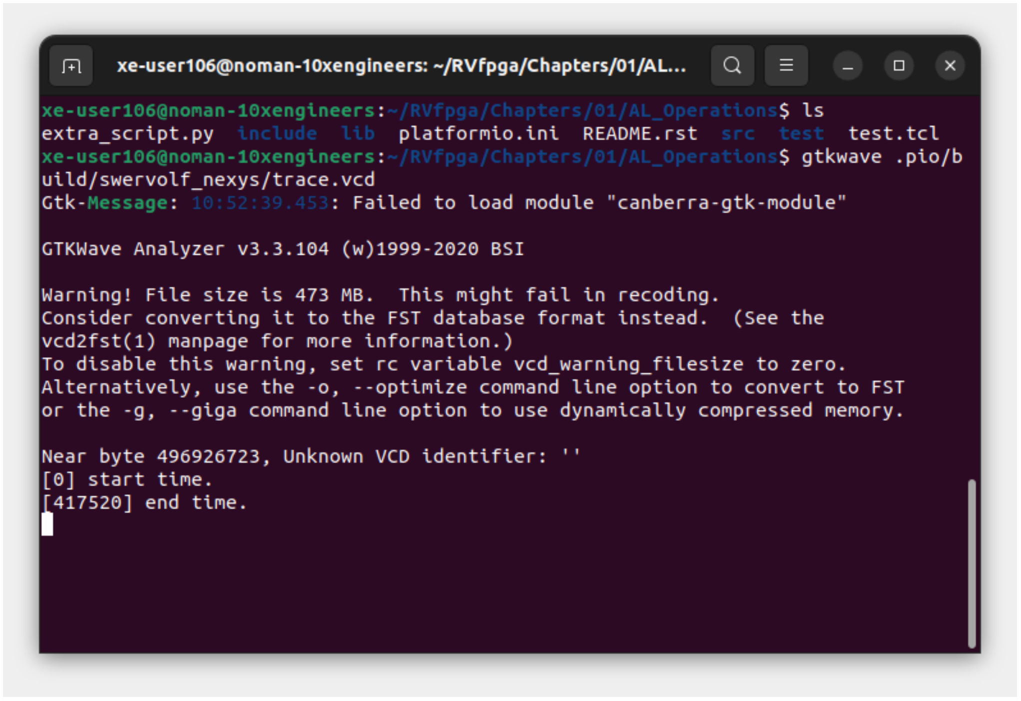
   And a disassembly program has been generated at: **RVfpga/Chapters/01/AL_Operations/.pio/build/swervolf_nexys/firmware.dis**.



```

.pio/build/swervolf_nexys/firmware.elf:     file format elf32-littleriscv


Disassembly of section .text.init:

00000000 <_start>:
   0: b0201073          csrw  minstret,zero
   4: b8201073          csrw  minstreth,zero
   8: 00000093          li  ra,0
   c: 00000113          li  sp,0
  10: 00000193          li  gp,0
  14: 00000213          li  tp,0
  18: 00000293          li  t0,0
  1c: 00000313          li  t1,0
  20: 00000393          li  t2,0
  24: 00000413          li  s0,0
  28: 00000493          li  s1,0
  2c: 00000513          li  a0,0
  30: 00000593          li  a1,0
  34: 00000613          li  a2,0
  38: 00000693          li  a3,0
  3c: 00000713          li  a4,0
  40: 00000793          li  a5,0
  44: 00000813          li  a6,0
  48: 00000893          li  a7,0
  4c: 00000913          li  s2,0
  50: 00000993          li  s3,0
  54: 00000a13          li  s4,0
  58: 00000a93          li  s5,0
  5c: 00000b13          li  s6,0
  60: 00000b93          li  s7,0
  64: 00000c13          li  s8,0
  68: 00000c93          li  s9,0
  6c: 00000d13          li  s10,0
  70: 00000d93          li  s11,0
```

7. Open the trace with GTKWave by executing the following command in a terminal:

```
gtkwave .pio/build/swervolf_nexys/trace.vcd
```

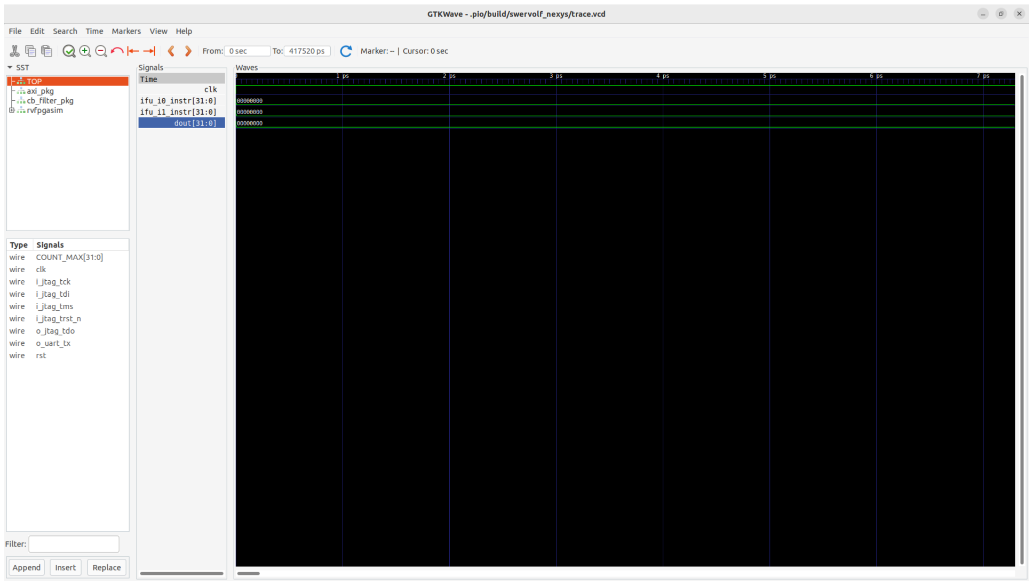


```
xe-user106@noman-10xengineers:~/RVfpga/Chapters/01/AL_Operations$ ls
extra_script.py  include  lib  platformio.ini  README.rst  src  test  test.tcl
xe-user106@noman-10xengineers:~/RVfpga/Chapters/01/AL_Operations$ gtkwave .pio/b
uild/swervolf_nexys/trace.vcd
Gtk-Message: 10:52:39.453: Failed to load module "canberra-gtk-module"

GTKWave Analyzer v3.3.104 (w)1999-2020 BSI

Warning! File size is 473 MB.  This might fail in recoding.
Consider converting it to the FST database format instead.  (See the
vcd2fst(1) manpage for more information.)
To disable this warning, set rc variable vcd_warning_filesize to zero.
Alternatively, use the -o, --optimize command line option to convert to FST
or the -g, --giga command line option to use dynamically compressed memory.

Near byte 496926723, Unknown VCD identifier: ''
[0] start time.
[417520] end time.
```

8. Now we will visualize some core internal signals in GTKWave. For that purpose, we will use file **test.tcl**, which is provided at Path-to-RVfpga/RVfpga/Chapters/01/AL_Operations.

   For using this **.tcl** file on GTKWave, we can simply click on File > Read Tcl Script File and select the Path-to-RVfpga/RVfpga/Chapters/01/AL_Operations/test.tcl file.

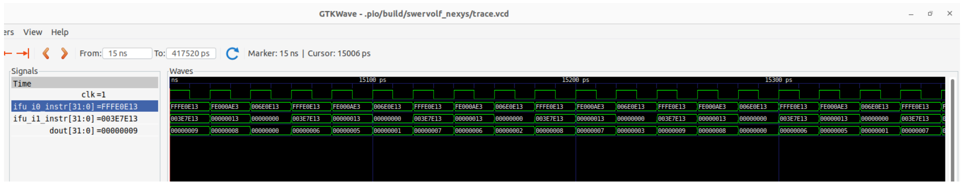
   This will add the required signals to our GTKWave for visualization.


GTKWave - .pio/build/swervolf_nexys/trace.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 417520 ps
Marker: -- | Cursor: 0 sec
SST
TOP
axi_pkg
cb_filter_pkg
rvfpgasim
Signals
Waves
Filter:
Append Insert Replace

```
Time
clk
ifu_i0_instr[31:0]
ifu_i1_instr[31:0]
dout[31:0]
```

Type	Signals
wire	COUNT_MAX[31:0]
wire	clk
wire	i_jtag_tck
wire	i_jtag_tdi
wire	i_jtag_tms
wire	i_jtag_trst_n
wire	o_jtag_tdo
wire	o_uart_tx
wire	rst

9. Once the signals are added in GTKWave. And skip the initial instructions in order to analyze the loop containing the three arithmetic-logic instructions; for example, we have selected a point in the simulation from 15ns on), in order to analyze the execution of the three arithmetic-logic instructions of a random iteration of the loop.

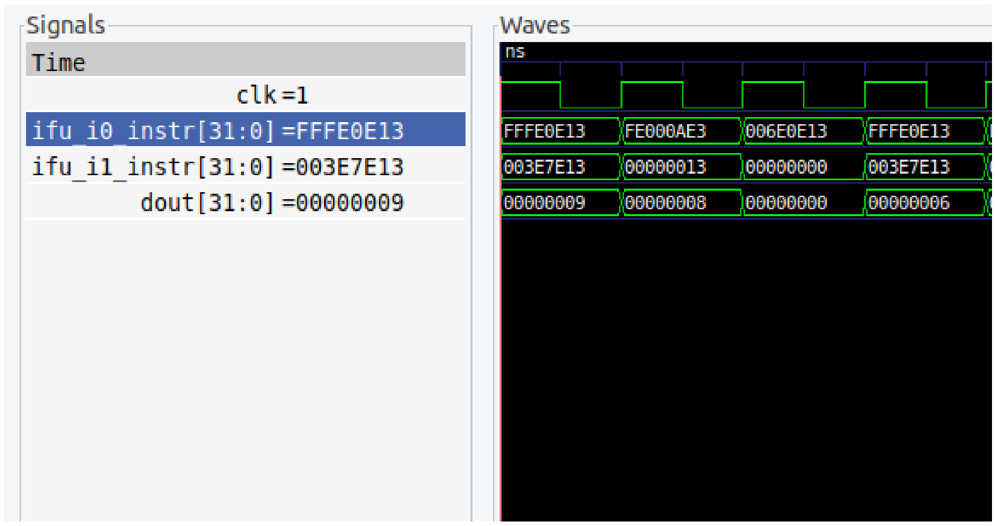

GTKWave - .pio/build/swervolf_nexys/trace.vcd
View Help
From: 15 ns To: 417520 ps
Marker: 15 ns | Cursor: 15006 ps
Signals
Waves
15100 ps
15200 ps
15300 ps

```
Time
clk=1
ifu_i0_instr[31:0]=FFFE0E13
ifu_i1_instr[31:0]=003E7E13
dout[31:0]=00000009
```

```
FFFE0E13 FE000AE3 006E0E13 FFFE0E13 FE000AE3 006E0E13 FFFE0E13 FE000AE3 006E0E13 FFFE0E13 FE000AE3 006E0E13 FFFE0E13 FE000AE3 006E0E13 FFFE0E13 FE000AE3 006E0E13 FFFE0E13
003E7E13 00000013 00000000 003E7E13 00000013 00000000 003E7E13 00000013 00000000 003E7E13 00000013 00000000 003E7E13 00000013 00000000 003E7E13 00000013 00000000 003E7E13
00000009 00000008 00000000 00000006 00000005 00000001 00000007 00000006 00000002 00000008 00000007 00000003 00000009 00000008 00000000 00000006 00000005 00000001 00000007
```

10. Note that every cycle two new instructions are fetched, one in each way of the superscalar RISC-V processor, as shown on signals **ifu_i0_instr[31:0]** and **ifu_i1_instr[31:0]**.

And the effects of the execution of the three arithmetic-logic instructions can be seen on signal **dout[31:0]**.



```
Time
clk =1                              
ifu_i0_instr[31:0] =FFFE0E13    FFFE0E13  FE000AE3  006E0E13  FFFE0E13
ifu_i1_instr[31:0] =003E7E13    003E7E13  00000013  00000000  003E7E13
dout[31:0] =00000009            00000009  00000008  00000000  00000006
```