

Using Rvfpga Practically

Luke Seed

30th May 2024

Environment

- There is no perfect environment in which to work and to develop RVfpga-based applications
 - Win
 - MacOS
 - Linux
- However, one is better than the other
 - Linux
- This can be a dual-boot Linux box OR a Linux VM
- Let's Assume you will be using Linux and that you are going to use Ubuntu 18 or 20 (either will work fine)

Ubuntu VM

- If you have not got the capacity to dual-boot a computer and install Ubuntu in a partition then you may need to go down the VM route
- VM stands for Virtual Machine and, typically, it means running a guest operating system on another host machine within an application (hypervisor)
- There are various VM applications but a good, free one is VirtualBox

*

VirtualBox will not run on M based Macs but Boot Camp does (paid for)

Requirements

- Your Linux 'box' needs at least 8Gb of RAM and should have circa 100Gb disk
- If you are using VirtualBox then these things can be set when you create your VM but to give the VM 8Gb of RAM you will need at least 16Gb of RAM in your computer
- Once you've set up your Linux box you are good to go
 - Remember – when you install Linux to set yourself up as a user with admin rights and REMEMBER the password you use (you will need it periodically)

Git Repository

- The RVfpga data has now been set up on a remote Git Repository:
- https://github.com/Microelectronic-Group-MISS-TUoS/RVFPGA_MSc_Project.git
- To set up the Rvfpga directory you need to clone the repository on your local machine
 - Open a terminal and `cd/mkdir` to a directory where you want the data to land and enter
`git clone https://github.com/Microelectronic-Group-MISS-TuOS/RVFPGA_MSc_Project.git`
 - Enter the directory
`cd RVFPGA_MSc_Project/`
 - Before you start any editing, enter
`git checkout -b MyBranchName`

(This will create a new branch and any changes you make will be associated with the branch and not affect the main repository, `main`)

Storing Changes Remotely

- Once you've edited anything in your branch you can upload it to the remote Git Repository
- This doesn't alter `main` (until we want it to) but it makes the changes available to other collaborators
- To do this, firstly, commit any changes to the local version of the repository
`git commit -a -m 'a helpful message saying what changes you made'`
- Then push the changes back to the remote repository
`git push -u MyBranchName`

Getting Started

- Follow the Rvfpnga Getting Started Guide
- This will lead you through installing
 - Visual Studio Code/Platform IO (the development environment)
 - Vivado (for Verilog code synthesis)
 - Verilator (for simulation)
 - GTKwave (for viewing simulation results)
- However, you will probably need to install a few more things
 - g++ Version 10.5 (a legacy version of gcc)
 - gedit (a good window based text editor)
- If you Google it, you will find out how to do it

Vivado does have its own simulator but it generates errors if you try to simulate Rvfpnga. Verilator, however, is more tolerant

Vivado

- To synthesise Rvfpfga for FPGA you need version Vivado 2019.2
- This is more than 3 years old now
- The preferred method for installing Vivado is the SelfExtracting Web Installer but this won't work for 2019.2
- Instead, you will need to download the Unified Installer. This is a TAR/GZIP file
 - Download it
 - Extract it using `tar -xvf <tarfile>`. This will create a new directory with a whole bunch of files in it
 - Navigate to the directory and run `xsetup`
 - Select WebPack to install and enter the credentials you set up to identify that you are a valid user

Vivado and Ubuntu

- When you try to install Vivado on Ubuntu it will report a lack of compatibility if you are using anything more recent than Ubuntu 18
- I have not noticed any issues using Ubuntu 20 but an installation on Ubuntu 22 might require some additional library installs

```
sudo apt install libtinfo5
```

```
sudo apt install libncurses5
```

- Followed by a restart

Alternatively ...

- If you are running your VM on a Win machine you can ...
- create a smaller VM
- Install Vivado on Windows
- You will need, within Virtual Box, to specify a shared directory (shared between the host OS – Windows and the guest OS – Linux)
 - This is a folder that appears in both Windows and Linux and will allow you to share data between your computer and Linux
 - Put the Rvfpga folder into the shared folder – now both OSs can access the data

Verilator

- Verilator is an application that analyses all of your Verilog code and uses it to create a bunch of C++ program files (along with some other code).
- Having done this a C compiler/linker is then used to build an executable application (Vrvfpgasim)
- Vrvfpgasim IS the simulator – when it is runs it generates the results of the simulation in a vcd file (*Value Change Dump*) – which you can then view
- The code generated by Verilator needs to be compiled using gcc version 10.x – this is a legacy version

Building the Simulation

- The building of the simulation is done in verilogSIM/ and is controlled by a Makefile
- `make` is a utility that allows you to selectively generate files based on when things have changed. It allows you to identify dependencies between files and how to generate the dependent file – this is only done if the modify date on the dependent file is earlier than the modify date on any file used to generate it
- By default, `make` looks for a file called Makefile and does what it says

The Make process

- All that Makefile in verilatorSIM/ does is to say
 - Vrvfpgasim depends on Vrvfpgasim.mk and is generated by running `make` on Vrvfpgasim.mk (using some specified options)
 - Vrvfpgasim.mk is generated by running Verilator using `swervolf_0.7.vc`
- `swervolf_0.7.vc` (Verilog Configuration) has two parts:
 - At the top are a bunch of `+incdir+` and `-CFLAGS -I` statements (these identify directories to be included by verilator – and the C compiler)
 - Below this is a long list of `.v` and `.sv` files – these are the Verilog files making up your design
 - This is, essentially, a recipe for building RVfpga

The Choice of Compiler

- The default version of GCC tools in Ubuntu will not work to compile the simulator
- Firstly, install gcc/g++ version 10 – these, by default, will be gcc-10/g++-10
- Now, if you look in Vrvfpgasim.mk two variables are used to identify the compiler and linker
 - CXX and LINK
- These are actually defined in \$(VERILATOR_ROOT)/include/verilated.mk
 - VERILATOR_ROOT is defined at the top of Vrvfpgasim.mk
- Edit this file and where you find CXX=g++ and LINK=g++, edit these to CXX=g++-10 and LINK=g++-10
- Now, when the make process runs, version 10 of the compiler will be used

Using the Built Design

- Rvfpfga is a processor system – when it operates (on the FPGA or in simulation) the processor needs to be running a program that is compiled/assembled into R-V object code and placed in the processor's RAM
- Typically, you will develop your source program (C or assembler) in VSC and then build your program:
 - Click on the PlatformIO icon
 - Click on Build in the Project Tasks menu
- Where does the resultant object code go and how is it loaded into the processors RAM?



Code Project Directory

- To build code, create a new PlatformIO project and place it in a new directory (under Rvfpga/examples/MyProj say)
- Write your code and build it
- Within Rvfpga/examples/MyProj there is a hidden directory (Ctrl-H to unhide it) called .pio/
- Navigate to Rvfpga/examples/MyProj/.pio/build/swervolf_nexys/
- Here you will find a bunch of files called firmware.x – these are your compiled object code
- Edit firmware.dis (this is a textual disassembly of your code)

Object Code

- firmware.dis will look like

Disassembly of section .text.init:

00000000 <_start>:

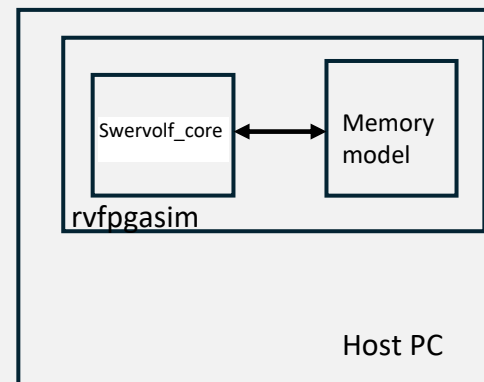
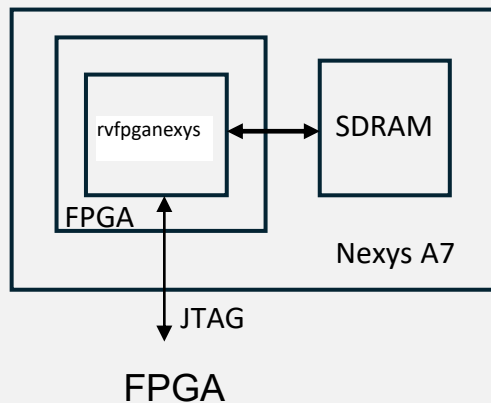
0:	b0201073	csrw	minstret,zero
4:	b8201073	csrw	minstreth,zero
8:	4081	li	ra,0
a:	4101	li	sp,0
c:	4181	li	gp,0
e:	4201	li	tp,0
10:	4281	li	t0,0

- firmware.vh will look like
 - It's a bunch of 64 bit hex numbers
 - In little-endian format

```
B8201073B0201073
4201418141014081
4401438143014281
4601458145014481
4801478147014681
```

FPGA or Simulation

- There are two top-level Verilog modules
 - Rvfpganexys for synthesis to operate on an FPGA
 - Rvfpgasim for simulation
- Why?
- The answer lies in what's inside the design and what isn't



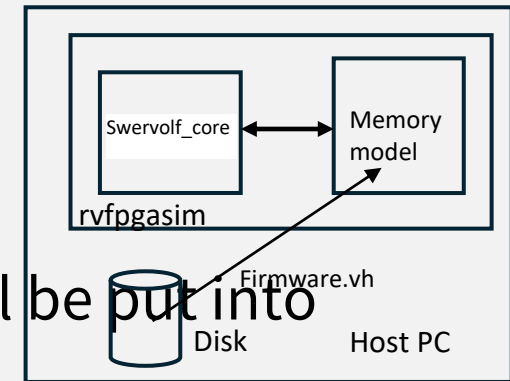
Simulation

Running Code on FPGA

- When a design is to be implemented on FPGA. The physical design will be in rvfpganexys.bit (generated by synthesizing and P&R the design in Vivado)
- Clicking on the Upload Bitstream button in in the PlatformIO menu downloads rvfpganexys.bit to the Nexys board and it begins running but there is no program to run
- Now clicking on the Upload button downloads firmware.vh to the Nexys board and loads it into the RAM at address 0
- This is done via the JTAG port
- Now resetting and releasing the R-V processor from reset will run the program that is now in memory

Running Code in Simulation

- In the top-level module for simulation there is some code that uses Verilog semi-hosting functions (these generally allow the simulation to access the host computer's file system)
- \$readmemh(file, mem) loads a file consisting of a list of hex numbers (e.g. firmware.vh) into an array of reg inside your design – the memory model
- Clicking on Generate Trace in the PlatformIO Project Tasks menu will run the simulation and the results will be put into .pio/build/swervolf_nexys/trace.vcd



Running a Simulation Manually

- Click on Generate Trace will, in most cases, be enough to run your simulation
- However, the default simulation will finish after circa 400ns and this may not be enough
- You can run your simulation for longer by doing it manually:
 - Firstly (after building your code), click on Generate Trace (this will ensure that firmware.vh exists and is up-to-date)
 - Secondly, in a Terminal, `cd` to your project directory (say Examples/MyProj)
 - Finally, enter the following command in the Terminal:

```
../../verilatorSIM/Vrvfpgasim -ram_init_file=./.pio/build/swerv_nexys/firmware.vh -vcd=1 -timeout=TIMEOUT
```
 - Where TIMEOUT is how long you want the simulation to run, in ps (i.e. if you want 1us of simulation, TIMEOUT=1000000)


Putting It All Together

- When you download Rvfpnga/ there is a default rvfpganexys.bit file
 - Until you change the Verilog you don't have to rebuild this
 - The existing one will run code
- You do have to build Vrvfpasim and once you've set the tools up it's as simple as opening a terminal, changing directory to verilatorSIM/, typing `make clean` (to clean out any previous build) and then `make` to build Vrvfpgasim
- You can build a software project in Visual Studio Code

If You are Simulating

- Have Visual Studio Code open with your project
- Have a terminal open at verilatorSIM/
- If you are editing the Verilog code, have a text editor open with the files you are editing
- When you are ready to run a simulation
 - If you've changed your C code then Build it and just Generate Trace
 - If you've changed your Verilog code then use the terminal to rebuild the simulation and then Generate Trace in Visual Studio Code
- Navigate to .pio/build/swervolf_nexys and double click on trace.vcd
 - This should open GTKwave
 - You can then navigate the design hierarchy and select signals to view
 - Don't close this when you go back and make edits – when you've re-simulated just use Menu->File->Reload Waveform to dump the new results in GTKwave
- Loop back and repeat until your design is correct

Rinse-
and-
Repeat



Linux 1.01

- Get used to using Terminal in Linux and entering text commands
 - Yes, it's 'old school' but sometimes it's the only way
- Common Linux commands
 - `cd` – change directory (`~/` is your home directory, `..` is enclosing directory, `/` is the separator)
 - `ls` - list directory contents
 - `pwd` – where am I
 - `sudo <command>` (run `<command>` with admin rights – you may be asked for your password)
 - `tar` (create or expand an archive: `tar -cvf tarfile directory/` archives the directory's contents into `tarfile`, `-cvzf` compresses it `tar -xvf tarfile` extracts the contents of `tarfile`)
 - `git` (allows you to access a remote repository `git clone repository` makes a local copy of a remote repository - `repository` will be a URI if the repository is on a remote system)
 - `apt` (allows you to install or update software e.g `sudo apt install package`)
 - `grep` (search for string/regex in files) e.g. `grep -r 'my cat' *` will list all instances of the string, "my cat" in any file inside the current directory or sub directory