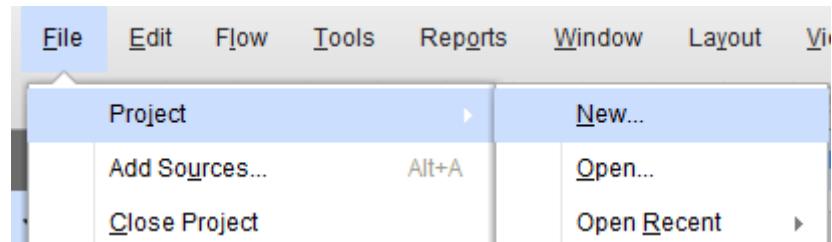


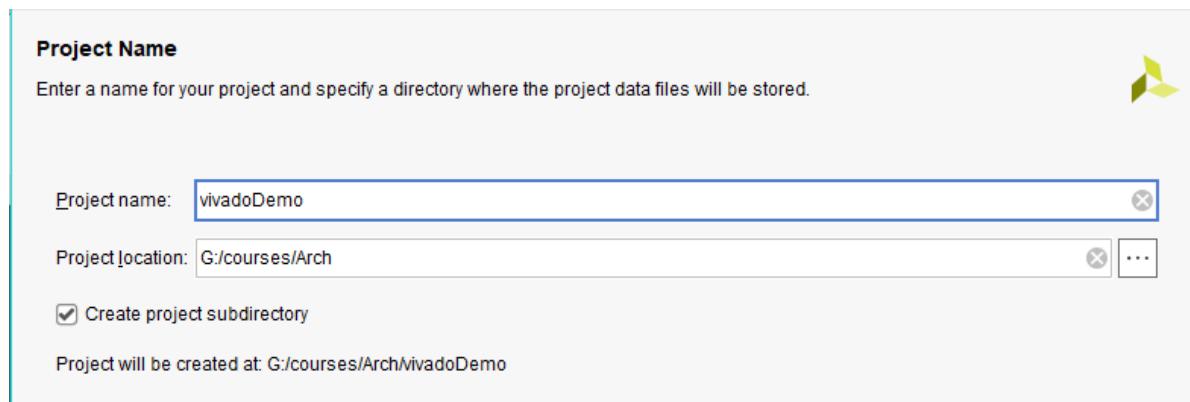
Use Vivado Step by Step

Create a project

New project

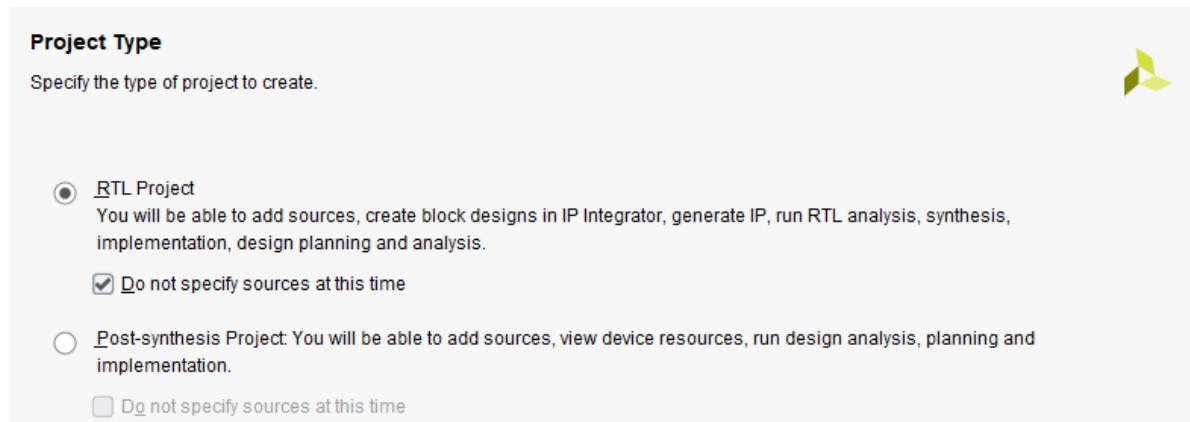


Define project name



Choose project type to be RTL project

since you need to add some provided code into the project, we recommend you add all sources after creating this project



Select Part

Important : Select the part name we wrote in the project doc, otherwise you may fail to run your CPU on FPGA. (but you can still change this part name after creating whole project, so whatever)

Default Part

Choose a default Xilinx part or board for your project.



Parts | Boards

[Reset All Filters](#)

Category:	All	Package:	All	Temperature:	All
Family:	All	Speed:	All		

Search: (1 match)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gt
xc7a35tcpg236-1	236	106	20800	41600	50	0	90	2

< >

[?](#) [Back](#) [Next](#) [Finish](#) [Cancel](#)

Add provided code and other resources

Click on the '+' button



You may see three kinds of resource type,

Here 'constraints' are for .xdc files, i.e. Basys-3-Master.xdc we provided

and 'design sources' are for .v files, i.e. cpu.v, hci.v, riscv_top.v we provided and your own .v files



Add Sources

This guides you through the process of adding and creating sources for your project

- Add or create constraints
- Add or create design sources
- Add or create simulation sources



< Back

Next >

Finish

Cancel

Add your files

You can choose add directories to add all the provided code for convenience.

But you need to add .xdc file after clicking "Add or create constraints" in the last step.

Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.



	Index	Name	Library	Location
	1	StupidCPU	xil_defaultlib	I:/

Add Files

Add Directories

Create File

Scan and add RTL include files into project

Copy sources into project

Add sources from subdirectories



< Back

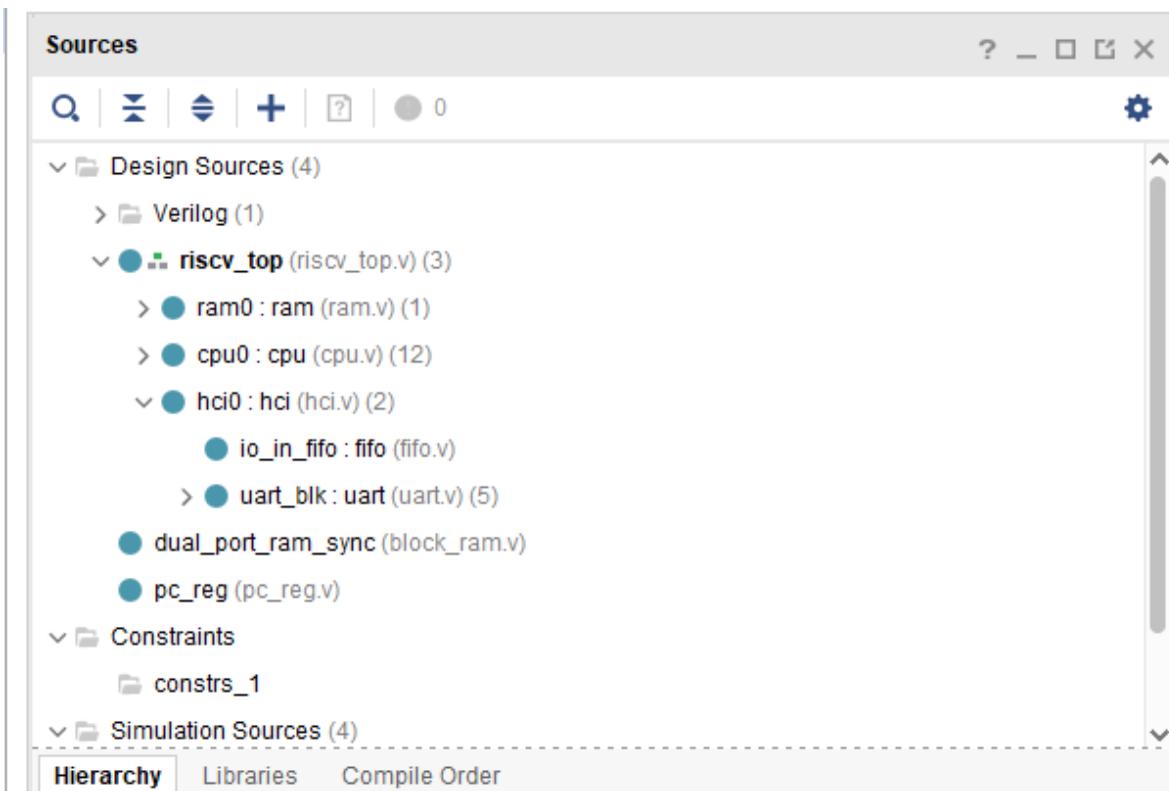
Next >

Finish

Cancel

Check sources

After adding the design files, you will see your design files organized by Vivado automatically. This also demonstrates the hierarchy of your .v files.



Run SIMULATION

We simulate the running of CPU in Vivado to check basic logic of our design.

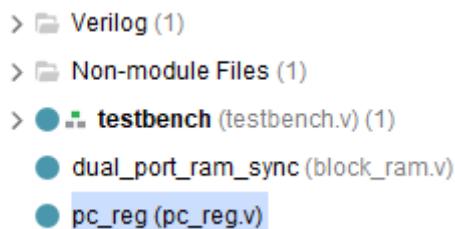
Note: Passing simulation doesn't necessarily mean your design is workable on FPGA.

Before running simulation

- You must add testbench.v into your project and enable this file to run simulation. This is the file that provides fake clk signals in simulation.

In other words, you shall find testbench.v the top of your design sources.

If you want to test your design on FPGA, you must disable 'testbench.v' since it can't be synthesised.



- You must tell Vivado where to find memory file, which is the input file for a test.

In 'block_ram.v', change the path here to the path of your 'test.data'. If there's some problem with relative path, just use absolute path.

```

Sources ? - □ ×
Design Sources (5)
> Verilog (1)
> Non-module Files (1)
✓ testbench (testbench.v) (1)
    > top : risov_top (risov_top.v) (3)
        > ram0 : ram (ram.v) (1)
        ● ram_bram : single_port_ram_sync (block_ram.v)
        > cpu0 : cpu (cpu.v) (12)
        > hd0 : hd (hd.v) (2)
    ● dual_port_ram_sync (block_ram.v)

Hierarchy Libraries Compile Order

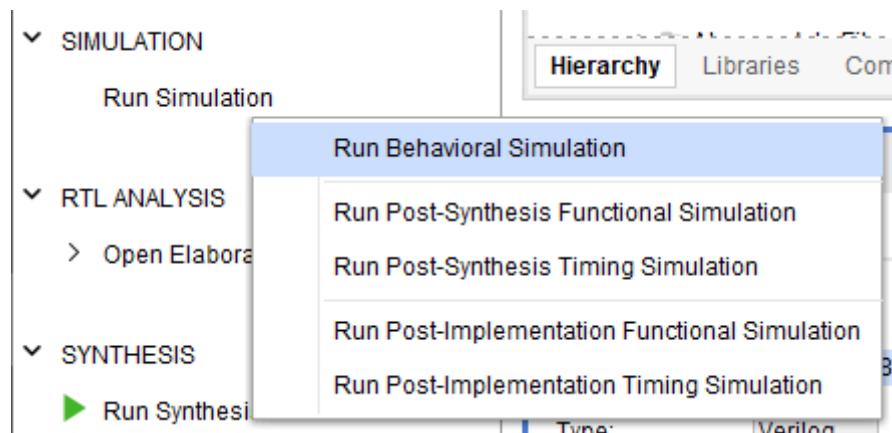
Properties ? - □ ×
block_ram.v
Enabled
Location: I/Arch_2018/src/common/block_ram
Type: Verilog
Library: xl_defaultlib
Size: 3.3 KB
Modified: Today at 19:25:11 PM

Project Summary pc_reg.v testbench.v ram.v cpu.v block_ram.v stage_if.v mem_wb.v hcl.v stage_id.v
I/Arch_2018/src/common/block_ram/block_ram.v
10: input wire [ADDR_WIDTH-1:0] addr_a;
11: input wire [DATA_WIDTH-1:0] din_a;
12: output wire [DATA_WIDTH-1:0] dout_a;
13: ;
14: 
15: reg [DATA_WIDTH-1:0] ram [2*ADDR_WIDTH-1:0];
16: reg [ADDR_WIDTH-1:0] q_addr_a;
17: 
18: always @(posedge clk)
19: begin
20:     if (vv)
21:         ram[addr_a] <= din_a;
22:     q_addr_a <= addr_a;
23: end
24: 
25: assign dout_a = ram[q_addr_a];
26: 
27: // initialize ram content (for simulation)
28: integer i;
29: initial begin
30:     for (i=0;i<2*ADDR_WIDTH;i=i+1) begin
31:         ram[i] = 0;
32:     end
33: end
34: 
35: $readmemh("I:\Arch_2018\viser-gau-toolschain\text\text.dat", ram); // add test data to vivado project or specify a valid file path
36: 
37: endmodule

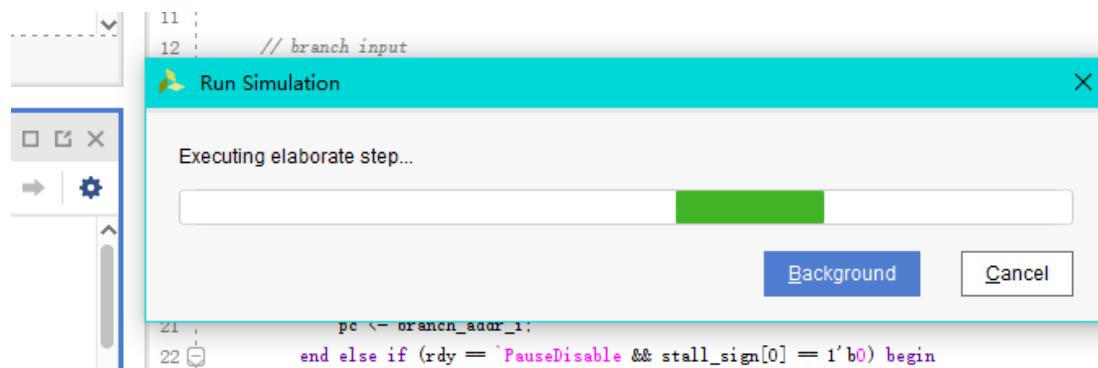
```

Start simulation

Click 'Run Simulation' and click 'Run Behavioral Simulation'

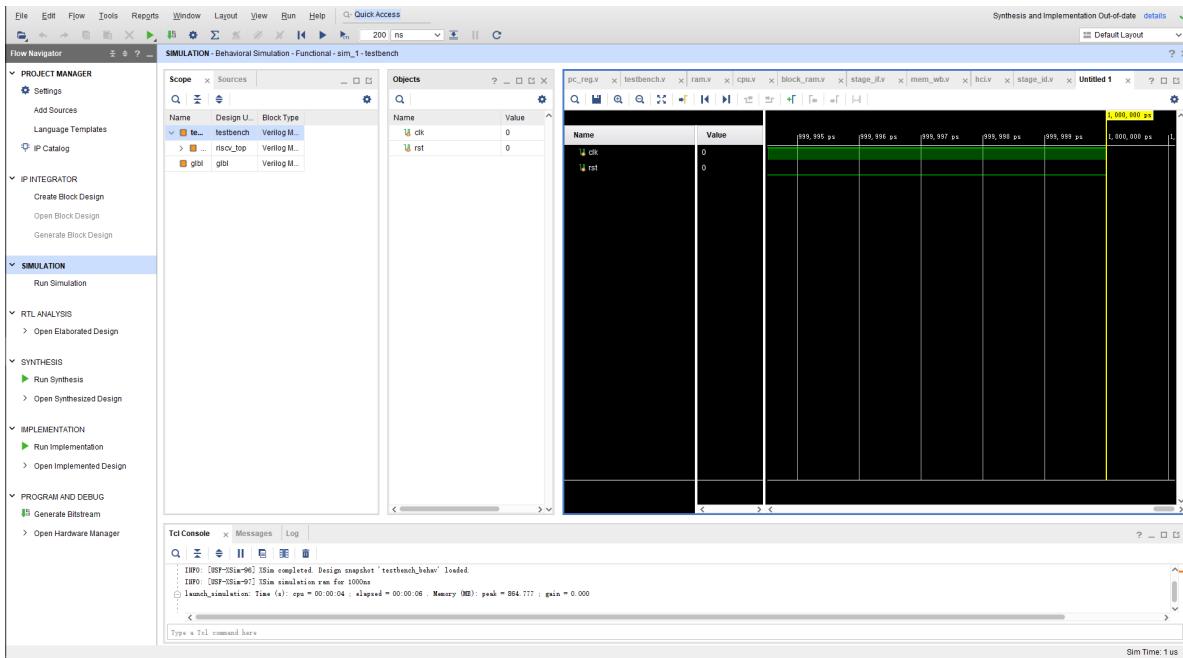


After this

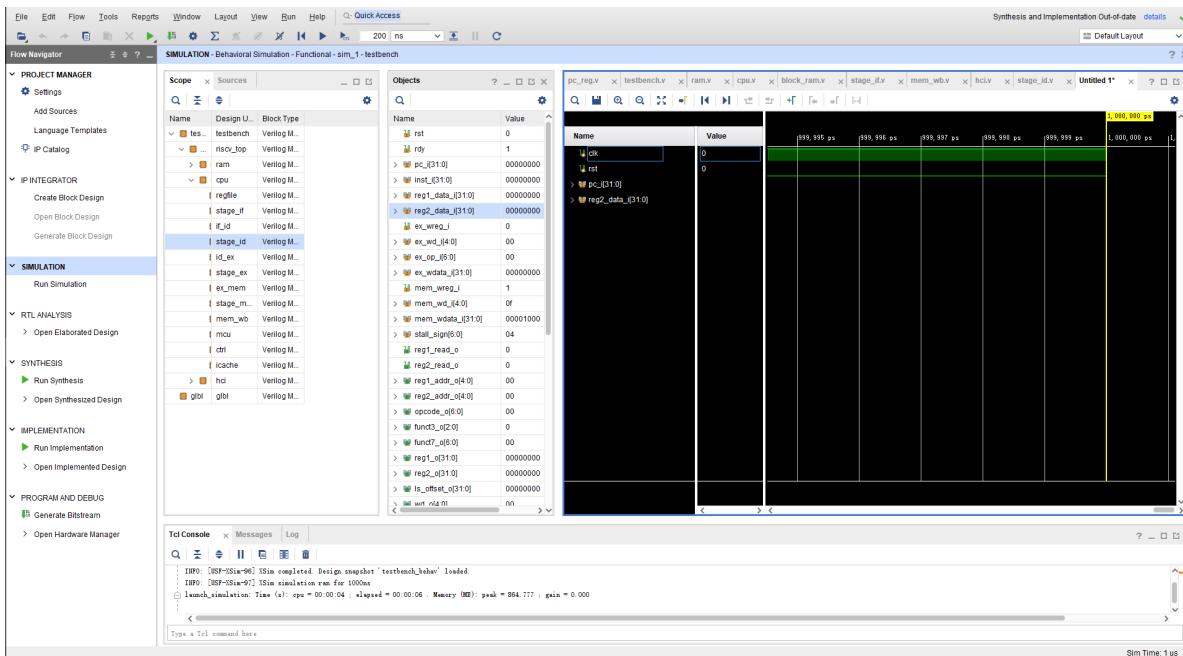


Simulation Layout

You may find Vivado change into simulation layout



In which the right side is the wave panel, and you may drag some objects into it to watch their status. Also, you can right-click on these objects and then add them into wave panel.



Run Simulation

Notice these 3 buttons.

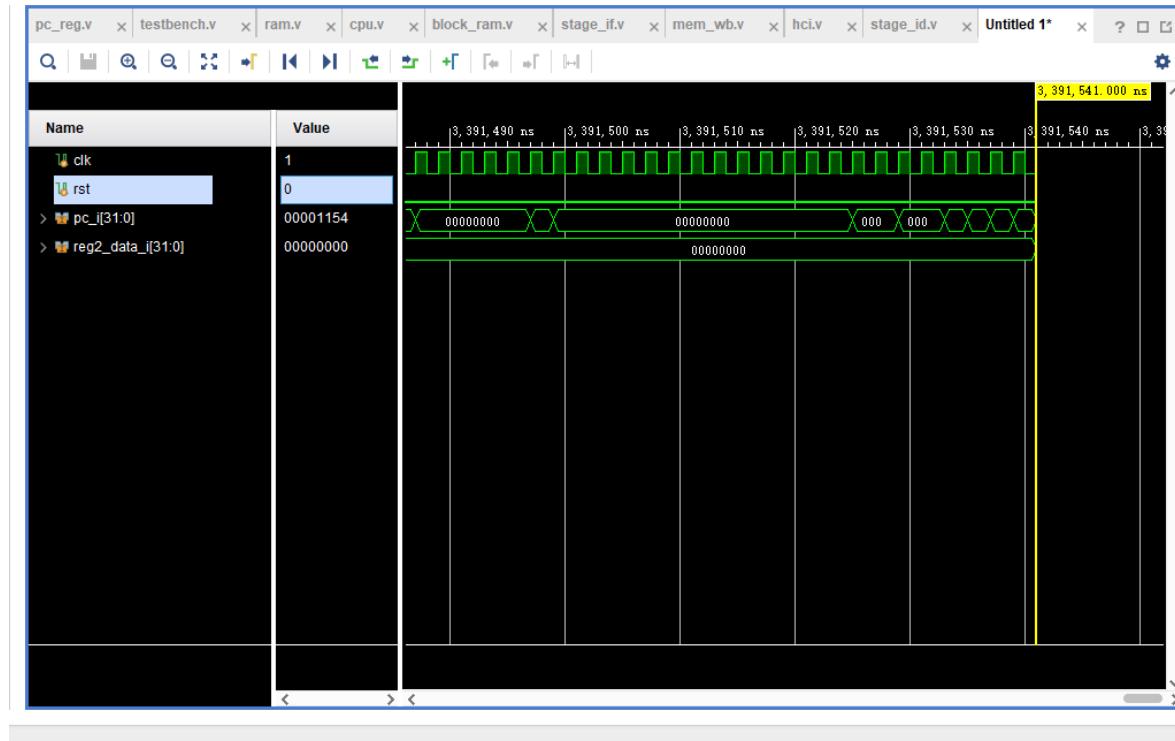


Among these 3, the first one is 'Restart', the middle one is 'Run All' and the last one is 'Run For *'.

Here 'Run All' means simulating until the program stops. As for 'Run For *', you can control the simulation time by changing the '200' in this photo, for example, to '2000' or changing 'ns' to 's'.

Running status

During simulation, the wave panel is like this



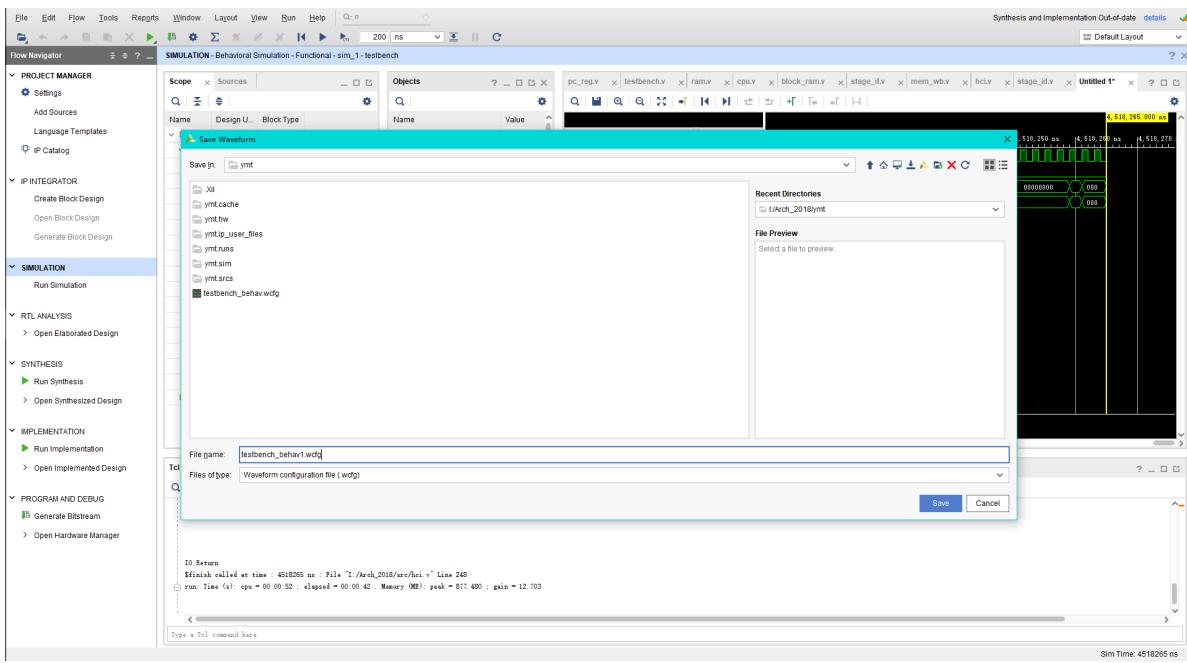
Where you can see the value(decimal by default) of some signals, and their changes.

And you can see the output in TCL console.



Save your simulation

In order to store SIMULATION status for the next use, you can just save this simulation by pressing 'CTRL+S'.



Then you will get rid of trouble of adding all signals you want to watch when you start another simulation.

Run on FPGA

Install RISCV-Toolchain

You can compile the toolchain as the project README says.

Or you can just download the compiled toolchain and extract it anywhere you feel happy to place it and modify the prefix dir in 'build_test.sh'.

```
MS108-2020 > riscv > build_test.sh
1  #!/bin/sh
2  set -e
3  prefix='/mnt/g/courses/Arch/riscv-tool-chain/riscv-tool-chain/riscv'
4
```

Install serial

You need to install following package listed in the official github page before 'make' .

dependencies

Required:

- [catkin](#) - cmake and Python based buildsystem
- [cmake](#) - buildsystem
- [Python](#) - scripting language
 - [empy](#) - Python templating library
 - [catkin_pkg](#) - Runtime Python library for catkin

You can use following script to install these dependencies.

```

# for catkin
sudo apt install python-catkin-tools catkin

# for cmake
sudo apt install cmake

# for python
# WSL has python3 within it

pip install empy catkin_pkg

```

Then:

```

cd serial
make
make install

```

You should see something like this

```

jiyi@DESKTOP-DL5456R:~/MS108-2020/serial$ make install
cd build && make install
make[1]: Entering directory '/home/jiyyi/MS108-2020/serial/build'
make[2]: Entering directory '/home/jiyyi/MS108-2020/serial/build'
make[3]: Entering directory '/home/jiyyi/MS108-2020/serial/build'
make[3]: Leaving directory '/home/jiyyi/MS108-2020/serial/build'
[ 66%] Built target serial
make[3]: Entering directory '/home/jiyyi/MS108-2020/serial/build'
make[3]: Leaving directory '/home/jiyyi/MS108-2020/serial/build'
[100%] Built target serial_example
make[2]: Leaving directory '/home/jiyyi/MS108-2020/serial/build'
Install the project...
-- Install configuration: ""
-- Installing: /tmp/usr/local/_setup_util.py
-- Installing: /tmp/usr/local/env.sh
-- Installing: /tmp/usr/local/setup.bash
-- Installing: /tmp/usr/local/setup.sh
-- Installing: /tmp/usr/local/setup.zsh
-- Installing: /tmp/usr/local/.rosinstall
-- Installing: /tmp/usr/local/lib/pkgconfig/serial.pc
-- Installing: /tmp/usr/local/share/serial/cmake/serialConfig.cmake
-- Installing: /tmp/usr/local/share/serial/cmake/serialConfig-version.cmake
-- Up-to-date: /tmp/usr/local/share/serial/package.xml
-- Installing: /tmp/usr/local/lib/libserial.so
-- Up-to-date: /tmp/usr/local/include/serial/serial.h
-- Up-to-date: /tmp/usr/local/include/serial/v8stdint.h
make[1]: Leaving directory '/home/jiyyi/MS108-2020/serial/build'

```

Modify .sh files

Give execute and read permission to all .sh files in the project folder by:

```

sudo chmod +x *.sh

```

Build controller in directory 'ctrl'

```

./build.sh

```

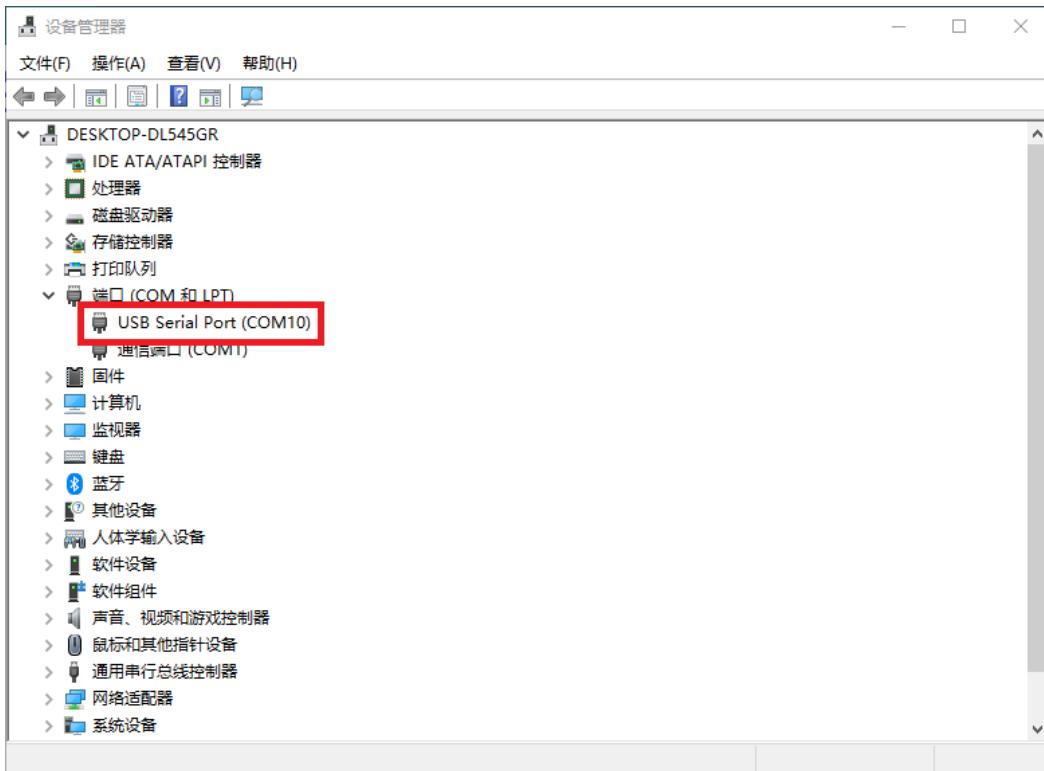
Then modify 'run_test_fpga.sh'.

You should modify the port number

```
./ctrl/build.sh  
./ctrl/run.sh ./test/test.bin ./test/test.in /dev/ttyS10 -I
```

You can see the port number in device manager on Windows:

Note: port number may change if you use different FPGA port, so you should use the port number you find in device manager after connecting FPGA with your computer



Run test by

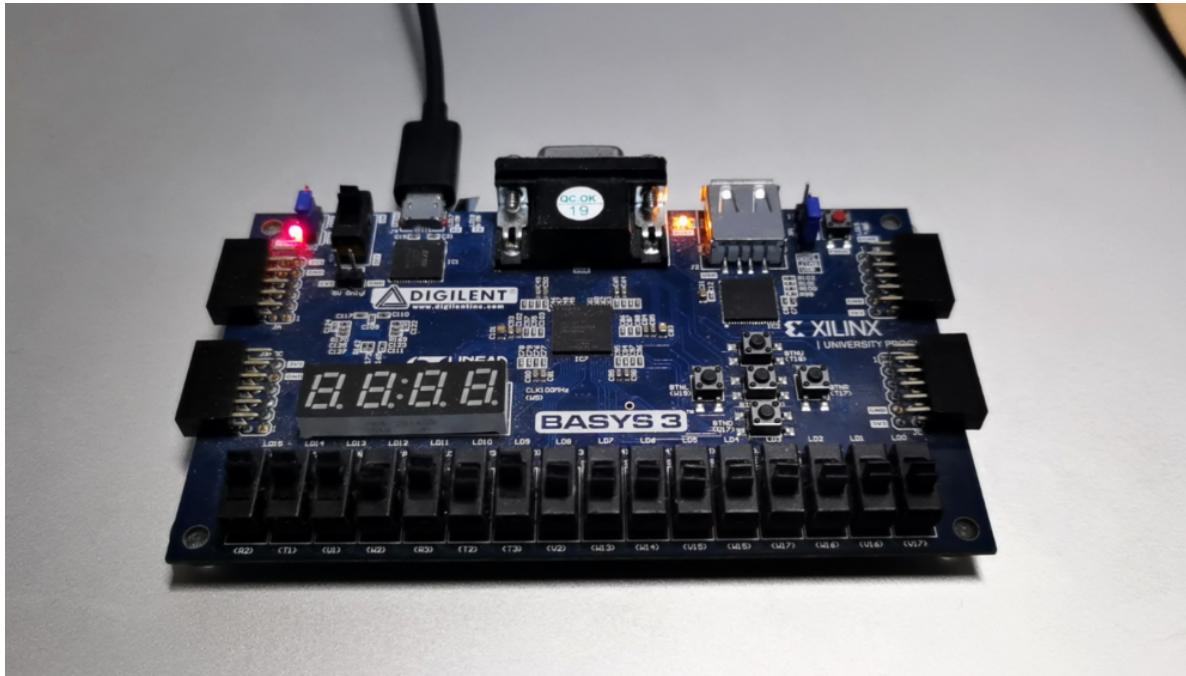
```
./run_test_fpga.sh testname
```

Connect FPGA with computer

To connect FPGA, you only need a USB-A to micro-USB cable (the same cable Kindle use).

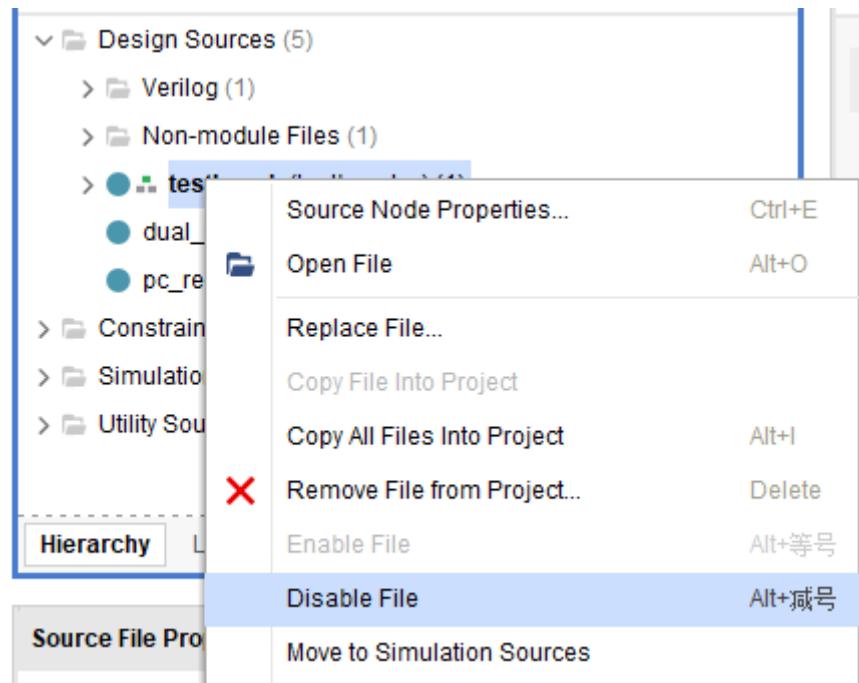


You should see power led on and a Breathing LED blinking after connecting it with your computer.

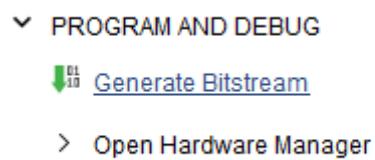


Generate Bitstream

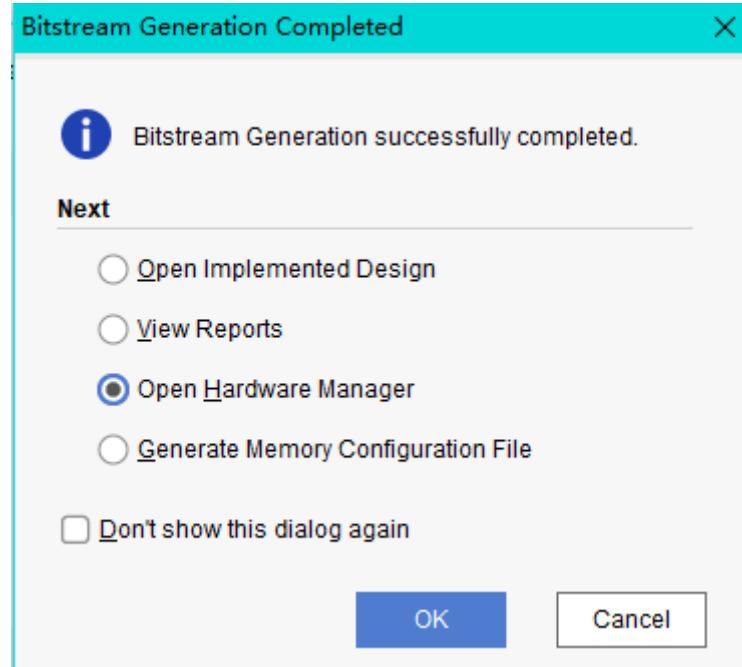
You should first disable testbench.v to run implementation.



Then generate bitstream

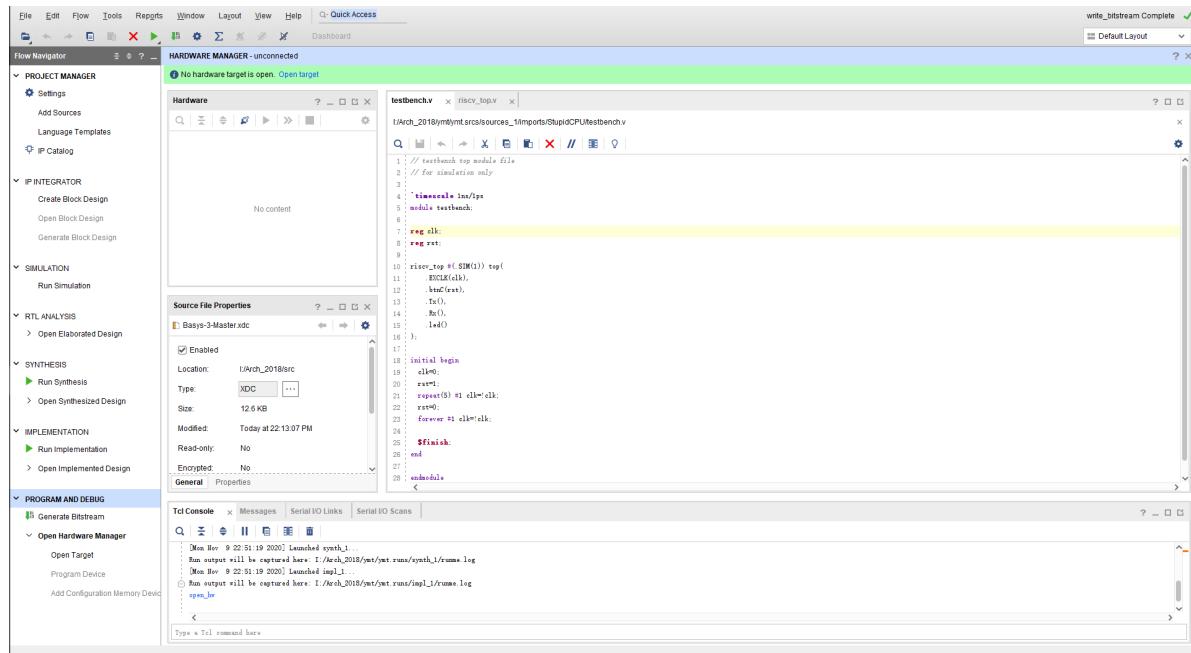


After several minutes, you will see

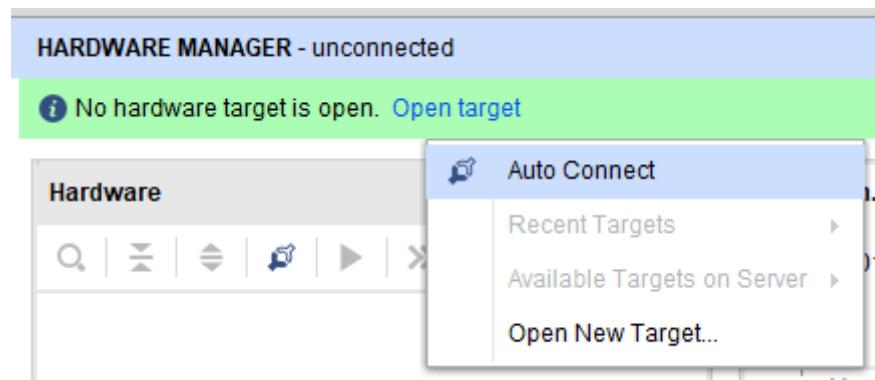


Program device

Open hardware manager, and you see hardware layout



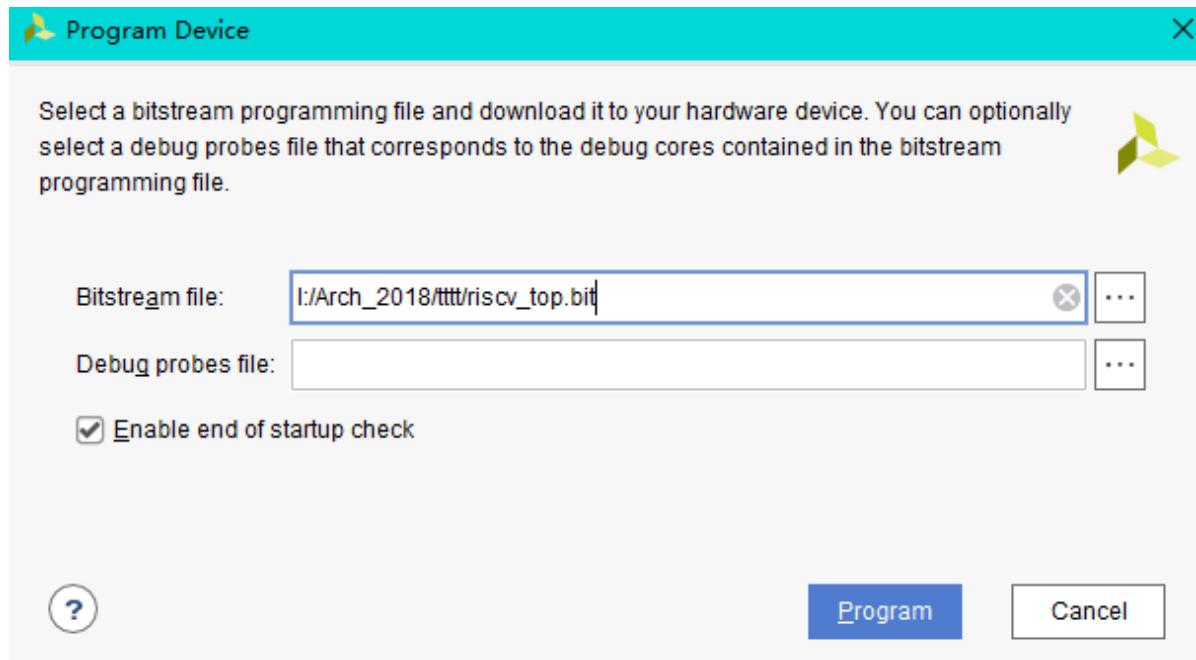
Connect Vivado with your FPGA



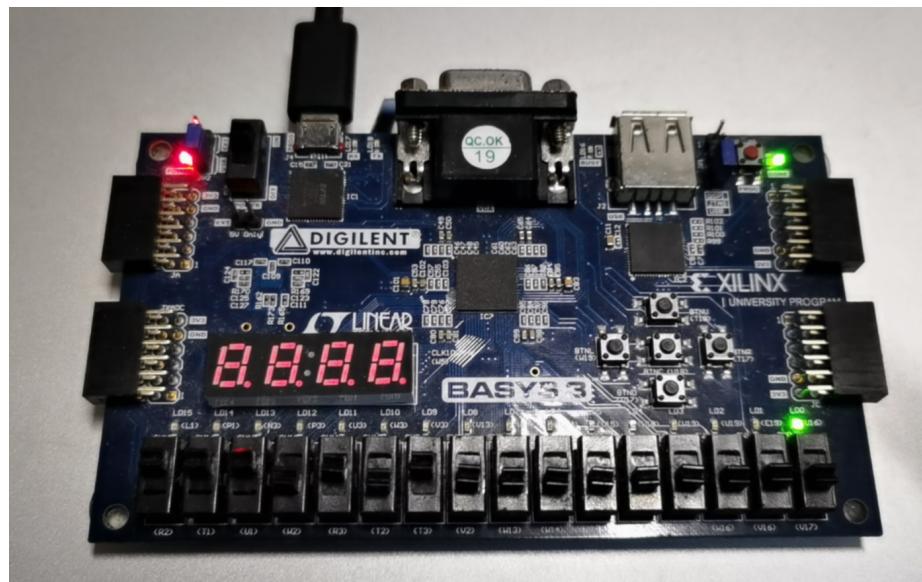
Seeing your device connected, you can program it

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/2101836993...	Open
xc7a35t_0 (1)	Not programmed
XADC (System Monitor)	

Click on program device and choose the .bit file you want to download



Program done, the breathing LED stops blinking



Then you can run test.

Run test

After programming the FPGA, run 'run_test_fpga.sh', we run pi.c as an example

```

jyi@DESKTOP-DL545GR:~/MS108-2020/riscv$ ./run_test_fpga.sh pi
RAM size: 130c
13 06 05 00 13 05 00 00 93 f6 15 00 63 84 06 00
33 05 c5 00 93 d5 15 00 13 16 16 00 e3 96 05 fe
67 80 00 00 63 40 05 06 63 c6 05 06 13 86 05 00
93 05 05 00 13 05 f0 ff 63 0c 06 02 93 06 10 00
input file not found, skipping
initialized UART port on: /dev/ttyS10
55 41 52 54
55 41 52 54
uploading RAM: 130c blks:5
blk:0 ofs:0000
blk:1 ofs:0400
blk:2 ofs:0800
blk:3 ofs:0c00
blk:4 ofs:1000
RAM uploaded
Enter r to run, q to quit, p to get cpu PC(demo)
|
```

You can see that the test program has been uploaded to FPGA and CPU is paused waiting to run, press 'r' and you will see the output in terminal.

```

jyi@DESKTOP-DL545GR:~/MS108-2020/riscv$ ./run_test_fpga.sh pi
RAM size: 130c
13 06 05 00 13 05 00 00 93 f6 15 00 63 84 06 00
33 05 c5 00 93 d5 15 00 13 16 16 00 e3 96 05 fe
67 80 00 00 63 40 05 06 63 c6 05 06 13 86 05 00
93 05 05 00 13 05 f0 ff 63 0c 06 02 93 06 10 00
input file not found, skipping
initialized UART port on: /dev/ttyS10
55 41 52 54
55 41 52 54
uploading RAM: 130c blks:5
blk:0 ofs:0000
blk:1 ofs:0400
blk:2 ofs:0800
blk:3 ofs:0c00
blk:4 ofs:1000
RAM uploaded
Enter r to run, q to quit, p to get cpu PC(demo)
r
CPU start
314159265358979323846264338327952884197169399375105820974944592307816406286208998628034825342117067982148086513282366470
9384469555822317253594081284111745028412701938521155964462294895493038196442881097566593344612847564823378678316527120
19914564856692346348610454326648213393607260249141273724587066063155881748815209209628925491715364367892590361133053054
882046652138414695194151160943305727365759591953092186117381932611793105118548074462379962749567351885752724891227938183
011949129833673362440656643860213949463952247371907021798694370277053921717629317675238467481846766945132000568127145263
56082778577134275778960917363717872146844090122495343014654958537105792796892589235420199561121290219686434418159813629
774771309960518707211349999983729780499510597317328160963185

CPU returned with running time: 1.781250
jyi@DESKTOP-DL545GR:~/MS108-2020/riscv$ |
```

Other little notes on FPGA running

- Once you programmed your CPU, you can run multiple tests. But if you directly run another test, uploading fails like this

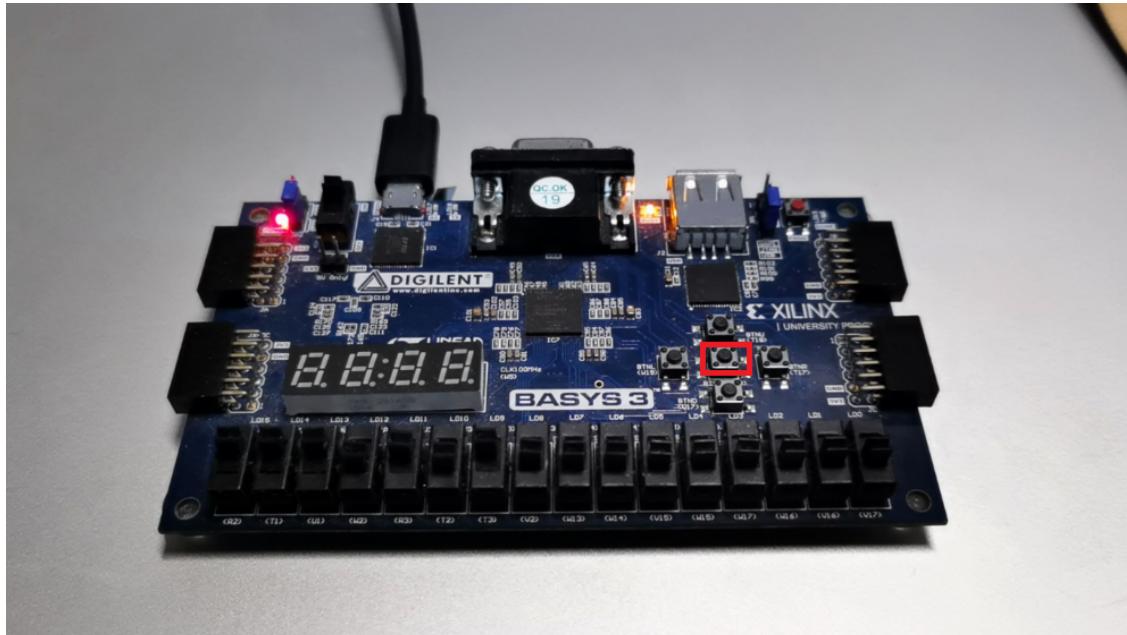
```

jiyi@DESKTOP-DL545GR:~/MS108-2020/riscv$ ./run_test_fpga.sh pi
RAM size: 130c
13 06 05 00 13 05 00 93 f6 15 00 63 84 06 00
33 05 c5 00 93 d5 15 00 13 16 16 00 e3 96 05 fe
67 80 00 00 63 40 05 06 63 c6 05 06 13 86 05 00
93 05 05 00 13 05 f0 ff 63 0c 06 02 93 06 10 00
input file not found, skipping
initialized UART port on: /dev/ttys10
55 41 52 54
55 41 52 54
uploading RAM: 130c blks:5
blk:0 ofs:0000
blk:1 ofs:0400
blk:2 ofs:0800
blk:3 ofs:0c00
blk:4 ofs:1000
RAM uploaded
Enter r to run, q to quit, p to get cpu PC(demo)
r
CPU start
314159265358979323846264338327952884197169399375105820974944592307816406286208998628034825342117067982148086513282366470
938446955582231725359408128481117450284127019385211555964462294895493038196442881097566593344612847564823378678316527120
199145648566923463486104543266482133936072602491412737245870660631558817488152092096282925491715364367892590361133053054
882046652138414695194151160943305727365759591953092186117381932611793105118548074462379962749567351885752724891227938183
011949129833673362440656643860213949463952247371907021798694370277053921717629317675238467481846766945132000568127145263
5608277857713427577896091736371787214684409091224953430146549585371057922796892589235420199561121290219686434418159813629
774771309960518702113499999983729780499510597317328160963185

CPU returned with running time: 1.843705
jiyi@DESKTOP-DL545GR:~/MS108-2020/riscv$ ./run_test_fpga.sh pi
RAM size: 130c
13 06 05 00 13 05 00 93 f6 15 00 63 84 06 00
33 05 c5 00 93 d5 15 00 13 16 16 00 e3 96 05 fe
67 80 00 00 63 40 05 06 63 c6 05 06 13 86 05 00
93 05 05 00 13 05 f0 ff 63 0c 06 02 93 06 10 00
input file not found, skipping
initialized UART port on: /dev/ttys10
55 41 52 54
00 00 00
UART assertion failed
jiyi@DESKTOP-DL545GR:~/MS108-2020/riscv$
```

Seeing this, you should press a memory reset button.

Then, you can run another test without programming the device again.



```
jyi@DESKTOP-DL545GR:~/MS108-2020/riscv$ ./run_test_fpga.sh pi
CPU returned with running time: 1.843750
jyi@DESKTOP-DL545GR:~/MS108-2020/riscv$ ./run_test_fpga.sh pi
RAM size: 130c
13 06 05 00 13 05 00 00 93 f6 15 00 63 84 06 00
33 05 c5 00 93 d5 15 00 13 16 16 00 e3 96 05 fe
67 80 00 00 63 40 05 06 63 c6 05 06 13 86 05 00
93 05 05 00 13 05 f0 ff 63 0c 06 02 93 06 10 00
input file not found, skipping
initialized UART port on: /dev/ttys10
55 41 52 54
00 00 00 00
UART assertion failed
jyi@DESKTOP-DL545GR:~/MS108-2020/riscv$ ./run_test_fpga.sh pi
RAM size: 130c
13 06 05 00 13 05 00 00 93 f6 15 00 63 84 06 00
33 05 c5 00 93 d5 15 00 13 16 16 00 e3 96 05 fe
67 80 00 00 63 40 05 06 63 c6 05 06 13 86 05 00
93 05 05 00 13 05 f0 ff 63 0c 06 02 93 06 10 00
input file not found, skipping
initialized UART port on: /dev/ttys10
55 41 52 54
55 41 52 54
uploading RAM: 130c blks:5
blk:0 ofs:0000
blk:1 ofs:0400
blk:2 ofs:0800
blk:3 ofs:0c00
blk:4 ofs:1000
RAM uploaded
Enter r to run, q to quit, p to get cpu PC(demo)
r
CPU start
314159265358979323846264338327952884197169399375105820974944592307816406286208998628034825342117067982148086513282366470
938446955582231725359408128481117450284127019385211555964462294895493038196442881097566593344612847564823378678316527120
199145648566923463486104543266482133936072602491412737245870660631558817488152092096282925491715364367892590361133053054
882046652138414695194151160943305727365759591953092186117381932611793105118548074462379962749567351885752724891227938183
011949129833673362440656643860213949463952247371907021798694370277053921717629317675238467481846766945132000568127145263
560827785771342757789609173637178721468440901224953430146549585371057922796892589235420199561121290219686434418159813629
774771309960518707211349999983729780499510597317328160963185

CPU returned with running time: 1.828125
jyi@DESKTOP-DL545GR:~/MS108-2020/riscv$
```