

Programmer's guide for running custom assembly tests on SweRV EL2 using Verilator.

1. Firstly the programmer can write any RISC V assembly with any instructions from RISC V IMC in a file with .s extension. Programmer can opt to use SweRV's predefined headers from the file defines.h by using: `#include "defines.h"` at the start of the assembly file.
2. Now if the programmer wants to make its code more readable he/she can add user defined headers by using `#define HEADER_NAME (value of header)`. For example if the header to be defined is named as STDOUT and its value is 0xd0580000 so it can be defined as `#define STDOUT 0xd0580000`. (**note**: name and value must be in same line after `#define`).
3. Two forward slashes can be used for commenting.
4. Now `.section .text` can be used before defining any additional data that is required in the test to print which usually consists of ASCII for example you can see the hello world test of SweRV which is present in RV_ROOT/testbench/asm. (RV_ROOT = the/path/to/swerv/core).
5. Now the global header of starting the code is defined as `.global _start` and from the next line after writing `_start`: programmers can begin to write the assembly code to perform any operation through core.
6. Once the file is ready it must be placed under the directory of asm which is at RV_ROOT/testbench/asm.
7. Now to run the test through Verilator you can use the Makefile for doing so run the following commands.

```
% export RV_ROOT=path/to/swerv/core  
% make -f $RV_ROOT/tools/Makefile verilator debug=1 TEST=name-of-asm-file
```

Now the test should and it will generate a sim.vcd file for debugging the waveforms.

For more details about running core in different configurations and how to install the prerequisites please visit

<https://github.com/chipsalliance/Cores-SweRV-EL2>

sample code files are provided for reference.

https://github.com/merledu/Rev-Soc/blob/master/core/testbench/asm/External_interrupt.s

https://github.com/merledu/Rev-Soc/blob/master/core/testbench/asm/spi_tb.s

https://github.com/merledu/Rev-Soc/blob/master/core/testbench/asm/timer_tb.s

Programmer's guide for running custom assembly tests on SweRV EL2 on simulators other than Verilator.

This particular documentation will guide how to run custom assembly tests on Vivado whereas all the steps will be the same but at the end the simulator can be different.

1. The programmer needs to separate the RTL design for doing so he/she must write an FPGA wrapper for SweRV core having all the synthesizable constructs of core from tb_top.sv file.
2. Secondly after completing the assembly test file and by placing it in the asm directory as did for the verilator the programmer must run the commands as did previously for verilator.
`% export RV_ROOT=path/to/swerv/core % make -f $RV_ROOT/tools/Makefile verilator debug=1 TEST=name-of-asm-file`
3. Now where the commands are executed in the same directory there will be a program.hex file generated which contains the hex converted code for the assembly file. The file will be in the format

@80000000

[illegible]

@8000010E

[illegible]

Now @80000000

Indicates the starting address for the instructions to load into the core. For running the test on Vivado after completing the design requirements one memory file is needed to be created with .mem extension and all the hex data after @80000000 and before @8000010E should be copied into the .mem file.

Which in thi case will be

```
73 10 20 B0 73 10 20 B8 B7 00 00 EE 73 90 50 30
B7 50 55 5F 93 80 50 55 73 90 00 7C B7 01 58 D0
17 02 00 00 13 02 E2 0E 83 02 02 00 23 80 51 00
05 02 E3 9B 02 FE B7 01 58 D0 93 02 F0 0F 23 80
51 00 E3 0A 00 FE 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
```

All set now just run the behavioral simulation.