**How To:**

1. **Run coremark**:

CoreMark is a benchmark program – a program that is supposed to test the performance of a CPU. CoreMark is built to minimizes the dependence of the results on compiler optimizations. As a benchmark, CoreMark allows for measurement of how changes in the RTL affect hardware performance. The score is normalized by clock speed, so it is more of a look at efficiency than raw computing power. Note that even though Coremark is compiler independent, implementing a new command that the compiler doesn’t know about will not improve the Coremark score.

* 1. Cd $MY\_PULP\_APPS/coremark
  2. Source comp\_coremark\_simtimer.sh - This is a special script because coremark is multiple programs
  3. Cd $MY\_PULP\_IRUN/
  4. Pulp\_get\_app coremark - This prepares coremark for simulation
  5. Pulp\_irun - simulate coremark
  6. the result is now presented in terms of total ticks
  7. We can now calculate the coremark score:

Explanation: total ticks is the number of clock cycles to run the coremark program once. If you take the reciprocal, then you get coremark /Hz. We normalize it to get coremark /MHz, which is the standard score

1. **Run waveform**
   1. After app is compiled and ready to run: (You can do this with the script our\_pulp\_compile)
   2. Pulp\_irun\_probe (this might take a while …)
   3. Simvision & (opens the simulator)
   4. A screenshot of a cell phone

      Description automatically generatedIn simvision: File->open Database in the browse tab there will be a folder called waves, open it.

The design is under tb/top\_i:

A screenshot of a cell phone screen with text

Description automatically generated

To save the signals and markers you add to simvision

* File->save command script
* Choose a save location

How to run the script:

* File-> source command script
* open the relevant script
* when a dialog window opens click ok

1. **Make assembly trace file**
   1. Get ready for simulation (our\_pulp\_compile)
   2. Run: pulp\_irun\_trace (must be in sim/irun directory)
   3. The trace file will appear in the irun folder:

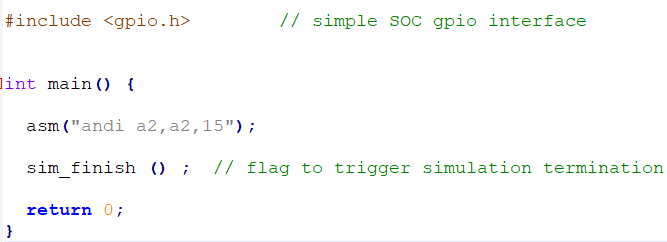
A screenshot of a computer

Description automatically generated

Notes:

* The time is the exactly the same as the simulation, but the cycles aren’t necessarily.
* You can use the time to search the waveform.
* The relevant register values in each cycle appear on the right columns.
* The mnemonic is the instruction in assembly.
* Instr column is the same as the mnemonic column written in hex.

1. After a program is compiled, you can run a waveform and trace by using the alias our\_wave\_trace <name of saved files>. The waveform will be saved in $MY\_PULP\_APPS/waves with the name you chose when running the script, and the assembly trace will be in $MY\_PULP\_APPS/trace, also with the chosen name.
2. **Other important files:**
   1. <file> .c.s is The complier output before linking.
   2. <file>.elf.read shows the disassembly of the source code but the c commands written aren’t always reliable.
3. **Run one assembly command:**
   1. We made a new folder in the apps directory called asm\_test for the assembly program which includes the c file that is called asm.c:



Then we compiled it and prepared for simulation. We then looked at the trace file and recognized the command there, it was one of the last commands. Notice that there a lot more commands there because of all the libraries and other boot commands. In the .elf.read file we found that it was disassembled in the main:

A screenshot of a social media post

Description automatically generated

The trace:

A screenshot of a cell phone

Description automatically generated

To write an assembly command in hex, the format is



The specific value of the word is just an example.

1. **Debugger**
   1. Change the file:
      1. Edit $MY\_PULP\_APPS/sw\_utils/eclipse\_gdb.cmd and change the correct elf path. For example, for bubblesort:

A screenshot of a cell phone

Description automatically generated

It’s not generic, so If you want to debug another program, you’ll need to change the path.

* 1. Run the command: Our\_pulp\_compile bubblesort
  2. Open a second terminal on the same server.
  3. Run the command: pulp\_terminal\_gdb $MY\_PULP\_APPS/bubblesort/bubblesort.elf -x $MY\_PULP\_APPS/sw\_utils/eclipse\_gdb.cmd
  4. The gdb will now be running. For reference about the gdb commands look in $MY\_PULP\_ENV/misc/gdb-refcard.pdf

1. **Add files to design**

To add a file to the design, you need to change

* 1. sim/irun/pulpino\_tb.f

//These next two files we changed for the core, we did not check if it works the same with the rest of the SoC

* 1. src/ips/riscv/verilator-model/Makefile
  2. src/ips/riscv/src\_files.yml