VirtualSoC
Simulator

MicrelLab - DEI
Università di Bologna

version 0.1

Abstract

This document is intended to guide the user in the usage of VirtualSoC.

Contacts

daniele.bortolotti@unibo.it[1]

How to run the simulator

Make sure you have set up everything in order to use the VirtualSoC Virtual Platform (see `setup.pdf`).

The VirtualSoC binary is located in `$VSOC_BIN_DIR`, the folder is added to your `PATH` enivornment variable (see `SOURCEME`) so the binary `vsoc.x` can be invoked from any directory. To get a full description of the options available, run:

```
$ vsoc.x --help
```

The main options are:

- `-cX` (where `X` $\in [1, 64]$) : sets the total number of cores (`X`) to be instantiated;

[1] please use as subject of your email : `[VirtualSoC] Support`

- `--intc=X` : sets the interconnection architecture and HW platforms.
  - single-cluster (`--intc=c`);
  - multi-cluster (NoC mesh architectures will be available in a forthcoming release.);
- `--ic=x` : switch to choose private ('`p`') or shared ('`s`') I-cache.
- `-v` : to enable waveform tracing.
- `-s` : to enable statistics output at the end of the simulation.

For a full description of the available options, run `vsoc.x -h` to see the help menu.

Applications

Cross-compilation

Prior to run a simulation you need to cross-compile (`arm-elf-gcc`) the application to be executed by the simulated ARMv6 cores. All applications are located in `$VSOC_APP_DIR` folder. To cross-compile the C source code of a given application:

```
$ cd $VSOC_APP_DIR/<target_app>

$ make clean all
```

The binary image will be installed by the toolchain (see the `Makefile`) in `$VSOC_APP_DIR/<target_app>/o-optimize/app.exe`.

Symlinks

When a platform is instantiated, before the simulation has started, VirtualSoC automatically loads a program binary for each cluster. The default name for the binary images is `TargetMem_`i`.mem`, where $i \in [1, 4]$. Once the application has been successfully compiled, the `Makefile` itself creates these symbolic links in the <`target_app`> folder. Alternatively, you can use the `vsoc_set_app` script to create the `TargetMem_`i`.mem` symbolic links in the

desired folder. Once invoked the `vsoc_set_app` script, choose the application from the menu and specify the number of links to be created (1 for the single-cluster architecture, 4 for the multi-cluster).

Examples of usage

Single Cluster

To simulate a single cluster platform with 16 cores, 16 TCDM banks and private I-cache per core, run the simulator with the following parameters:

```
$ vsoc.x -c16 --tb=16 --intc=c --ics=c
```

NOTE 1: when simulating the single-cluster architecture the option `--tc=X` is discarded, use `-cX` to specify the number of cores.

NOTE 2: since the folder `$VSOC_BIN_DIR` is added to the `PATH`, the simulator can be run from anywhere in the filesystem. However, notice that it will look for the `TargetMem_`*i*`.mem` in the folder from where it is run.

Statistics

It is possible to collect statistics since boot time or to selectively profile a portion of the application.

Boot time: to enable statistics since boot time two switches need to be set:

```
vsoc.x ... -a -s
```

where `-s` enables statistics collection and `-a` stands for autostart measuring.

Run time: two steps are required:

- enable profiling from your application (see the document `appsupport.pdf`):

```
void main()
{
  start_metric();

  //profiled code

  stop_metric();
}
```

- use the switch `-s` when invoking `vsoc.x`

At the end of the simulation the printed statistics will contain information related to:

- `DATA` (for each core)
  - `TCDM`: accesses (`TCDMacc`), cycles spent (`TCDMclk`), rate of access (`acc%`), numer of conflicting accesses (`CNFacc`), cycles spent for conflicts (`CNFclk`), percentage of conflicts (`CNFclk%`)
  - `L3 memory`: accesses (`L3acc`), cycles spent (`L3clk`), rate of access (`acc%`)
  - `HW sempahores`: accesses (`SEMacc`), cycles spent (`SEMclk`)
- `I$` (for each bank)
  - `HIT` : number of HIT
  - `MISS` : number of MISS
  - `INST` : number of instructions fetched
  - `LD/ST` : number of instructions LOAD/STORE
  - `LD/ST%` : percentage of LOAD/STORE
  - `MR%` : MISS RATE
  - `HR%` : HIT RATE
  - `LINES` : number of cache lines taken from main memory
  - `REPL` : number of replaced lines
  - `COLD` : number of cold MISS
  - `CAP` : number of capacity MISS
  - `MISSclk` : total of cycles spent for MISS
  - `HITcost` : total of cycles spent hit MISS
- `EXECUTION TIME` (for each core)
  - `EXEclk` : total number of active cycles
  - `IDLEclk` : total number of idle cycles