Dycton platform documentation

Tristan Delizy

August 23, 2018

Contents

1 Simulations Description

The Dycton platform simulates a simple low power SoC. The simulation uses MIPS32 ISA, based on the ISS from the SocLib project. The Iss behavior is cycle accurate with three stage pipeline model. The rest of the platform simulates a bus and memory using TLM (Transaction Modeling Language).The memory architecture is composed of a scratcpad memory exposing multiple memory banks directly to the processor and other memories on the bus.

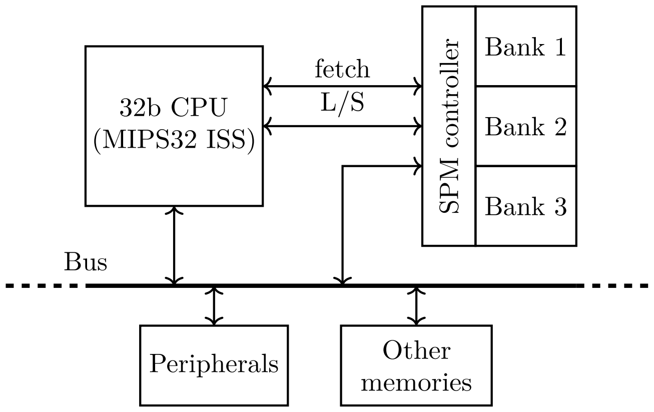

Figure 1: Overview of our simulated platform.

This document explains how to install tools required for simulation as well as the build process for the simulator and its applications. It also describe the procedure for running the experiments presented in related paper:

T. Delizy et al,Estimating the Impact of Architectural and Software Design Choices on Dynamic Allocation of Heterogeneous Memories, *Rapid System Prototyping Symposium* (2018).

2 Dycton and Toolchain Installation

Before Starting

this procedure was made and tested for Ubuntu 18.04 LTS with packages up to date (25/07/2018) start by running :

```
sudo apt-get update
sudo apt-get upgrade
```

the following packages must be installed first :

```
sudo apt-get install build-essential git
```

Dycton Sources Retrieving

Setup the git repository with:

```
git clone git@gitlab.inria.fr:citi-lab/dycton.git
```

SystemC

version used : 2.3.1 from accellera :
http://www.accellera.org/downloads/standards/systemc
installation procedure starting from the downloaded file folder (move it somewhere convenient):

```
tar xzvf systemc-2.3.1a.tgz
cd systemc-2.3.1a
mkdir obj
cd obj
../configure
make
make install
```

you need then to set the SYSTEMCROOT environment variable:

```
export SYSTEMCROOT=path/to/systemc/root/folder
```

adding path could be done in `~/.bashrc` file in order to be permanent, to take into account the modification in your terminal, execute the command:

```
source ~\.bashrc
```

Crosstool-ng Installation

You need to clone the repository and build crosstool-ng from the sources, otherwise you may not be able to build the toolchain.

Clone the sources, build them and make install:

```
git clone https://github.com/crosstool-ng/crosstool-ng
make
make install
```

Cross-compilation Toolchain

First install the required packages:

```
sudo apt-get install gcc gperf bison flex texinfo\
help2man make libncurses5-dev python-dev gawk
```

The environment variable `LD_LIB_PATH` must be unset during cross-compilation toolchain build:

```
LD_LIB_PATH_BK=$LD_LIB_PATH
unset LD_LIB_PATH
```

you can reset it after the cross-compiler build.

Now create a folder for the crosscompiler build. Then you need to get the `.config` file from the dycton repository: `repo_root/doc/crosstool-ng config file/.config`.

Copy it into the cross-compiler build folder, then open a terminal in that folder and build the toolchain with:

```
ct-ng build
```

the resulting toolchain will be located in `"~\x-tools\mipsel-unknown-elf\bin\"`, you need to add this location to your path:

```
export PATH=$PATH:~\x-tools\mipsel-unknown-elf\bin\
```

Add this line in `~/.bashrc` file, to take into account the modification in your terminal, execute the command:

```
source ~\.bashrc
```

3 Simulation Setup

Applications

The simulated platform will run an application from the list below:

jpeg: image compression / decompression (mibench suite)

jpeg2000: image compression / decompression (mediabench II video suite)

dijkstra: shortest path computation in graph (mibench suite)

parson: based on the parson library, this application parses a json file loaded in memory, select some keys and serialize the result.

h263: video compression / decompression (mediabench II video suite)

All applications are bare-metal, running with libc support (implementation from newlib).

Heap Memory Architectures

The simulation platform will have one of the following memory architecture for SPM banks dedicated to the heap. Their total size is parametrized by the maximum size used by application heap. As we study behavior of dynamic memory allocation, we consider that the code, data and stack in other "ideal" memories, inducing no delays.

0 : only one bank of fast memory

1 : two banks, fast memory bank size = 75% of application heap footprint

2 : two banks, fast memory bank size = 50% of application heap footprint

3 : two banks, fast memory bank size = 25% of application heap footprint

4 : two banks, fast memory bank size = 10% of application heap footprint

5 : two banks, fast memory bank size = 5% of application heap footprint

6 : only one bank of slow memory

-2 : virtual architecture with 2 banks of "infinite" size (large enough compared to the application needs).

Build and Run Simulator

open a terminal in the repository folder "`repository_root/src/platform_tlm/iss/`"
then run the following command:

```
make -B DY_SOFT=dummy_memset DY_ARCH=0
```

The build should run to completion with no errors. then execute the simulation:

```
./run.x
```

you should obtain as result :

```
Registering loader

        SystemC 2.3.1-Accellera --- Jul 12 2018 15:07:48
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
placement strategy : default (0)
...
>>>dycton platform init
...
================== Dummy memset test for DYCTON ==================
(TDk @ CITI Lab)


malloc return value (allocated address): 0x101005C0
writing magic number in target memory
...ok.
verifying written value
...ok.

========================== THE END ==========================
...
>>>simulation statistics
...
[wall clock time elapsed : 0h 0m 32s 915ms]
```

Build applications

Building an application just require to build project in the `repository_root/src/platform_tlm/iss` directory:

```
make -B DY_SOFT=app DY_ARCH=archi
```

with :

- `app` the folder name containing the target application in `repository_root/src/platform_tlm/software`
- `archi` the target architecture (-1 to 6).

important: When changing build arguments (changing application and/or architecture) the "`-B`" flag is required.

Application specifics: the following applications require some extra steps if changing the linked libc or when modifying the crosscompilation toolchain:

> **jpeg:** open a terminal in `"repository_root/src/platform_tlm/software/jpeg/jpeg-6a"` and build the jpeg library with `make -B`.
>
> **jpeg2000:** open a terminal in `"repository_root/src/platform_tlm/software/jpg2000"` and execute the script dedicated to crosscompile libjasper : `dycton_libjasper_build.sh`.

4 Experience Setup and Run

Requirements

In order to run the experience scripts you need to have python 2.7 installed. You also need to install cplex from IBM for placement computation (free for students and academics):
https://ibm.onthehub.com/WebStore/ProductSearchOfferingList.aspx?srch=cplex

For data processing and graph plots you need the following python packages:

```
numpy
matplotlib
python-tk
```

Configuration File

The file `config.py` contains declarations setting the variables of the experience. By modifying it you can choose which application will be run and on which memory architecture and which dynamic memory allocation strategy will be used. Executing it will print current configuration.

Memory Configuration

The latencies of the memory technologies used for heap banks can be configured in `src/platform_tlm/address_map.h` The following defines describes the latencies in cycles:

```
#define MEM_FAST_RLAT (1)
#define MEM_FAST_WLAT (3)

#define MEM_SLOW_RLAT (2)
#define MEM_SLOW_WLAT (30)
```

Experiment Preparation

In the root directory of the dycton repository run:

```
./xp_preparation.py
```

This will generate a `dycton_xp_TIMESTAMP` folder containing the different simulators, application and data to run the experiment. This folder also contains the script to tun the experiment.

Experiment Run

To run the experiment open a terminal in the experiment folder generated at previous step and run the following command:

```
./dycton_run_xp
```

To run the experiments on another machine you can just copy the experience folder on it and run the script.

Result Processing

The results are embedded in the experience folder, if the run was done on another machine you need to copy it back. To process the result and generates the graphs run the command

```
./xp_process_result.py path/to/result/folder
```

This will generates in the experiment folder a `results` folder containing the graphs in pdf and a csv file. The csv file contains informations about each execution (execution time, return code, ...)