MEET Simulator

Version 1.2.1

A Step by Step Guide

By Mostafa Bazzaz

Embedded Systems Research Lab
Sharif University of Technology

Change Log

1.2.1

- Tutorial updated

1.2

- Scripts are now compatible with Ubuntu’s ARM cross compiler
- Build script now checks for the existence of the compiler
- Some performance improvement tweaks

1.1

- Step by step guide added
- Two helper scripts added for simpler compiling and simulating

1.0

- Initial release

This document was last updated on May 22, 2019.

Contents

1. Introduction

1.1. About MEET

MEET (Microcontroller Energy Estimation Tool) is an energy profiler tool for AT91SAM7x256 microcontroller developed at embedded systems research laboratory of Sharif University of Technology, Iran (see http://esrlab.ce.sharif.ir/). The energy model used in **MEET** is the model presented in the following paper:

Mostafa Bazzaz, Mohammad Salehi, and Alireza Ejlali. ***"An accurate instruction-level energy estimation model and tool for embedded systems,"*** *IEEE Transactions on Instrumentation and Measurement, Vol. 62, No. 7, 2013, pp 1927-1934.*

This model estimates the energy consumption of CPU, SRAM, Flash memory, and memory controller. However, it does not estimate the energy consumption of other peripherals such as RS232. Therefore, **MEET** cannot estimate the energy consumption of **printf** statements or other similar functions.

The program is based on **Sim-profile** which is a part of SimpleScalar simulator suite (see http://www.simplescalar.com). **MEET** receives an ARM7TDMI compatible binary image and simulates the program at instruction level. Working with **MEET** is very similar to **Sim-profile** and anyone with enough knowledge of building and running SimpleScalar simulators should be able to run **MEET**. To simplify the process of compiling and running applications, three shell scripts are prepared: **build.sh**, **extractor.sh,** and **run.sh**. **build.sh** is a wrapper script for calling **Ubuntu's ARM cross compiler** with appropriate options. **extractor.sh** is used for analyzing the application binary and finding the address of the entry point of the application and **run.sh** is a wrapper for calling **MEET** with appropriate options.

As mentioned before, **MEET** does not estimate the energy consumption of RS232. Therefore, the simulated program cannot include any type of console output statements. The simplest way to remove all **printf** statements is to add a compiler directive to change the definition of **printf** to nothing (i.e., after **include** statements add **#define printf** this should be repeated for all console-related functions such as puts, putc, fprintf, etc).

MEET inherits the profiling ability of **Sim-profile** which can be combined with the energy estimation capability to form an energy profiler tool. **Sim-profile** can profile an application against a given metric which can be the variable holding the total energy consumption of the application. The output will be the energy consumption per instruction which can help in identifying the hotspots of the application.

It must be noted that **MEET** also inherits some of limitations of ARM version of **SimpleScalar** as well. Certain instructions of ARM ISA are not implemented in the ARM version of SimpleScalar such as **SWP**, **MSR**, **MRS**, and **SWI**. As a result, **MEET** cannot simulate applications that rely on these instructions such as **Linux** kernel.

1.2. Hardware Platform

The hardware platform simulated by **MEET** is an **AT91SAM7X256** microcontroller which is equipped with 64 KB of SRAM, 256 KB of Flash, and an ARM7TDMI processor. The internal structure of this microcontroller is shown in Figure 1. The energy estimation model includes the energy consumption of the processor core, SRAM, and Flash. The Flash memory is used for storing the code and read-only data while the SRAM is used as the runtime data memory.

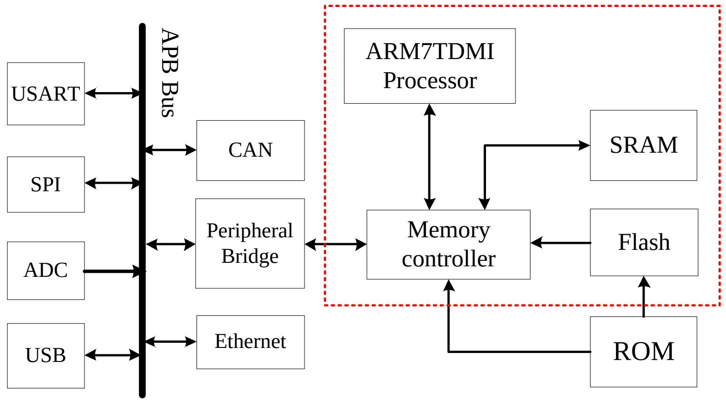


Figure 1 Internal structure of AT91SAM7X256

MEET distinguishes between Flash memory access and SRAM memory access by the target address of load/store instructions. The Flash memory is mapped to 0x100000 and SRAM memory is mapped to 0x200000. Therefore, in order to obtain accurate results, you must change the linker script according to this memory configuration. There is a sample linker script inside the package named **at91.ld**.

2. System requirement

The system requirements of **MEET** are very similar to SimpleScalar. **MEET** needs the **flex** package and it can be compiled using **GCC** compiler in **Linux** environment. Also, the provided scripts are based on Bash which must be selected as the user's shell interpreter.

Version 1.2.1 has been tested using the following configurations but it should be easy to build it using other versions as well.

- `Linux Mint 18 'Sarah' 64 bit + GCC 5.4.1`
- `Linux Mint 17.1 'Rebecca' 64 bit + GCC 4.8.4`
- `Linux Mint 17.1 'Rebecca' 64 bit + GCC 4.4.7`
- `Linux Mint 17.2 'Rafaela' 64 bit + GCC 4.8.4`

3. Installing MEET

Install the essential packages for compiling **MEET**.

- ➢ `sudo apt-get install build-essential flex`

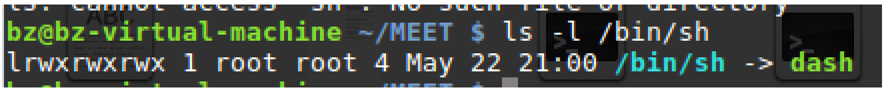
Make sure that **GCC** is installed correctly and it is included in the PATH variable.

- ➢ `gcc -v`

Make sure that **bash** is selected as the user's shell interpreter by executing the following command.

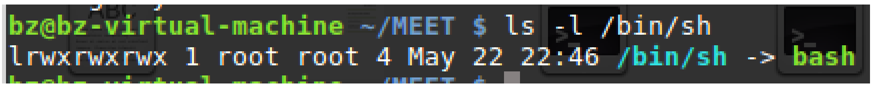
- ➢ `ls -l /bin/sh`

The following result shows that **dash** is selected instead of **bash**.



```
bz@bz-virtual-machine ~/MEET $ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 May 22 21:00 /bin/sh -> dash
```

This following shows the desired output.



```
bz@bz-virtual-machine ~/MEET $ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 May 22 22:46 /bin/sh -> bash
```

3.1. Building MEET

Download MEET-1.2.1.zip and extract the downloaded package to a folder.

- ➢ `unzip MEET-1.2.1.zip`

From here on in, we assume the folder is placed in the home folder and we refer to this new created folder as [MEETfolder]. Change the current directory to [MEETfolder].

- ➢ `cd $HOME/MEET`

Choose ARM as target configuration.

- ➢ `make config-arm`

Build the package.

- ➢ `make`

On successful build, the last line of the output should be "my work is done here...".

Validate the program by running the sample application.

- ➢ `sh validateInstall.sh`

After executing the above command, **MEET** simulates a sample application and generates the simulation results. Total energy consumption of the application is listed as “sim_total_energy 617785.6875 # total energy consumption (nJ)”. The whole simulation should take less than 5 seconds.

3.2. Installing ARM compiler

You can use Ubuntu’s default arm compiler, build your own version of GCC cross compiler, or download one of the prebuild versions from the internet. In this section, we use **Ubuntu’s default arm compiler** in package **gcc-arm-none-eabi**. It can be installed using the following command in Ubuntu:

```
➢ sudo apt-get install gcc-arm-none-eabi
```

Now, open a console and change your current directory to [MEETfolder].

```
➢ cd $HOME/MEET
```

Make sure ARM compiler is included in the PATH.

```
➢ arm-none-eabi-gcc –v
```

To test your toolchain, compile the sample application inside **tests** folder.

```
➢ sh build.sh tests/quicksort.c -o tests/quicksort –O2 -DNO_PRINT
```

build.sh wrapper calls the cross compiler with mandatory options. The user must provide input(s), output and any other options required.

After compiling the application, simulate the program using **MEET**.

```
➢ sh run.sh tests/quicksort main
```

run.sh wrapper takes two parameters: full path of executable binary and name of the starting point of the measurement. The latter enables the developer to start the energy estimation process from a specific address of the program to exclude the initialization phase of the application.

Total energy consumption of the application is reported as sim_total_energy=617785.6875 nJ.

4. Using MEET

Assuming that you have installed the required cross compiler; you can build your application using **build.sh** and simulate it using **run.sh**.

Make sure your application does not contain any print instruction (e.g., add **#define printf()** after your include statements. See sample applications inside **tests** folder).

Change your current folder to [MEETfolder] (in our example it was $HOME/MEET). This step is only required if you want to use **build.sh** and **run.sh**.

```
➢ cd $HOME/MEET
```

Compile your source code (e.g, $HOME/source/quicksort.c) using **build.sh** script

```
➢ sh build.sh $HOME/source/quicksort.c $HOME/source/quicksort
```

Simulate the program (assuming we want to estimate the energy consumption of whole program including the initialization steps).

```
➢ sh run.sh $HOME/source/quicksort main
```

4.1. Options

Executing **MEET** without any argument prints all possible options of the program. Most options are similar to that of **Sim-profile**. There are three new options which are listed in Table I. If you are using **run.sh**, you do not need to use these options directly.

Table I - New options of MEET

Option	Purpose
-initial:pc [address]	Start the execution from the specific address. If this option is not specified, the starting address of program from the binary image is used instead.
-finish:pc [address]	End the execution after reaching the specific address.
-initial:meas [address]	Start the estimation process after reaching the specified address. If this option is not specified, the estimation process will start from the beginning of the application.

4.2. Simulation statistics

Same as options, the final statistics of simulation are similar to **Sim-profile** with the exception of 7 new values. These are listed in Table II.

Table II - New simulation statistics

Option	Purpose
inst_count_after_meas	Total number of instructions executed after starting of estimation procedure.
sim_num_flash_loads	Total number of Flash read memory accesses
sim_num_sram_loads	Total number of SRAM read memory accesses
sim_total_energy	Total energy consumption of the simulation (nJ)
instruction_bus_activity	Total number of bit flips in instruction bus
instruction_bus_weight	Total number of '1' bits in instruction bus
regbank_activity	Total number of bit flip in register bank