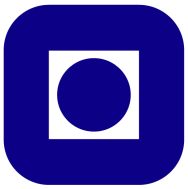

NTNU

TDT4258 - Energy Efficient Computer Systems

Exercise 3

Group 13

Mathias Ose & Øyvind Robertsen

April 24, 2014

Abstract

In this final exercise of the TDT4258 course, the group implemented a simple game on the EFM32GG development board, using the same gamepad peripheral as in the previous exercises to control the game. As in the previous exercise the game was implemented in the C language and an ARM compatible GNU toolchain was used to build and flash the development board. The difference from the previous exercise is theat for this one a Linux distro would first be flashed to the development board, and the game would be run inside this operating system. A dedicated driver was written to facilitate communication between the game and the peripheral, through the OS and the hardware GPIO.

Contents

1 Introduction **1**

2 Description & Methodology **2**
2.1 Development process 2
2.1.1 Devices 2
2.2 Project setup & toolchain 2
2.2.1 PTXdist usage 3

3 Results **4**
3.1 Program 4
3.1.1 Testing the program 4
3.2 Energy efficiency 4
3.3 Discussion 4

4 Evaluation of assigment **5**

5 Conclusion **6**

1 | Introduction

2 | Description & Methodology

This section describes the development process, use of tools, debugging techniques and details of the implementation. As listing entire implementations spread across several files for each development iteration would clutter the report and take away from the subject matter being discussed, we will only list code or formulas relevant to the task/section being described, not complete implementations. For details on the entire implementation, we refer to the source code attached to this report.

2.1 Development process

The development part of this exercise was, as with the previous ones, done at the computer lab in the IT-Vest building at NTNU (room ITV-458). The lab is equipped with workstations connected to the EFM32GG-DK3750 development board made by Energy Micro. The subject staff provided us with a framework on which to base our work, more on this in section 2.2. As per usual, we put the project under version control immediately to ease collaboration. Seeing as this is a multipart exercise, collaboration aided by version control became an important aspect of the process to a larger degree than in the previous exercises.

As mentioned above, the exercise consists of several parts. The two major ones; implementing a device driver for the gamepad connected to the board and implementing a game using said driver as an input interface. We decided upon which game to implement soon after reading the exercise requirements. This proved to be beneficial in the development of the driver, seeing as we had a clear image of how we wanted to interface with the driver.

2.1.1 Devices

The devices used in this exercise are the same as in the previous exercises, the EFM32GG-DK3570 produced by Silicon Labs, and the gamepad peripheral connected to the development board via GPIO. For this exercise, a Linux distribution called uClinux is flashed to the developement board. uCLinux is Linux distribution targeting microcontrollers.

2.2 Project setup & toolchain

The framework provided by the subject staff for this exercise has grown in complexity in comparison to the handouts for the previous exercises, but in accordance with the complexity of the exercise requirements. The development in this consisted of two largely separate parts, the driver and the game. While both were implemented in C, the driver

was made as a kernel module, and the game as a regular user space application. Developing a kernel module requires a slightly different mindset than developing general user space programs. Some of the notable differences:

- The module must implement a predefined minimal set of functions in order for the kernel to know how to utilize the module.
- The module is restricted to calling only other functions defined in the kernel. For example, functions from the C standard library may no be used.
- The module must be event based. Polling for input using an infinite loop will cause the kernel to hang.
- The module must be compiled in context of the kernel version it is meant to be used in, in much the same way we have to cross-compile our C code for use on the ARM processor on the development board.

To ease development and automate some common and tedious build tasks, the PTXdist system is used in this exercise. PTXdist is a build system for Linux, giving us a way of reproducibly building our distro from source, with any desireable modifications to both the kernel and userspace.

2.2.1 PTXdist usage

Figure 5.1 in the subject compendium [**?**, p. 48] provides a good visual overview of the build process PTXdist facilitates. In short, PTXdist allows us to define simple, Makefile like rules dictating how each independent piece of the system is to be compiled and installed, as well as a simple way of defining how to run tests. The following is an overview of usage of the PTXdist utility.

Setup

Assuming the `ptxdist` utility and a suitable toolchain has been installed on the host system, the necessary configuration is quite straight forward. All `ptxdist` commands must be run from a directory chosen to be the the root directory of the project. In our case, this directory is the `OSELAS.BSP-EnergyMicro-Gecko`-directory in the exercise framework. First of all, we specify a platform configuration file and the path to the toolchain bin-directory. Second a project specific config file is specified. See listing 2.1.

Listing 2.1: Config

```
ptxdist select configs/ptxconfig
ptxdist platform configs/platform-energymicro-efm32gg-dk3750/
   platformconfig
ptxdist toolchain <path\_to\_toolchain\_bin\_dir>
```

3 Results

3.1 Program

The program we implemented is a clone of the (at the time of writing) very popular web game 2048.

`http://en.wikipedia.org/wiki/2048_(video_game)`

3.1.1 Testing the program

After flashing the correct OS and program build to the device, the device should automatically load the gamepad driver then start the game program.

Playing the game

Use the gamepad buttons SW1 through SW4 as directional buttons.

When one directional button is pressed, all tiles will move into empty tile spaces along the axis of movements.

Two equal tiles will merge into one tile equal to the sum of it's parts.

Score

Game over

Restart game or exit game

3.2 Energy efficiency

3.3 Discussion

4 | Evaluation of assigment

5 | Conclusion