System Construction Course 2017,

**Assignment 1**

Felix Friedrich, ETH Zürich, 19.9.2017

## Introduction

At the beginning of systems programming stands the communication with hardware. Very often no diagnostic device such as a debugger connected via JTAG is accessible. Therefore the first that we need is a sign of life of the hardware we are working with. The simplest you can do is let some lights blink in order to see that your code is actually running. In this exercise we will execute just this: with a very simple program written in assembly language we will let an LED light up. In the second part of the exercise we will learn more about linking and will experience the advantages of high level (systems) programming.

Lessons to Learn

- Learn to know ARM assembly language basics, understand the limits

- Understand GPIO control on BCM 2835/6, learn to read technical manuals

- Understand basics of linking and bootloading

# 1 Sign of Life

In this part of the exercise we illuminate an LED using low level assembler code.

## Preparation

We have already prepared the boot media (SD card) for the Raspberry Pi such that it can load and boot a file called `kernel.img` from the first FAT partition. This is the standard procedure for the RPI. Parts of the bootloader image stems from `http://openelec.tv/get-openelec`.

All other resources can be found in our subversion repository located at
`https://svn.inf.ethz.ch/svn/lecturers/vorlesungen/trunk/syscon/2017/shared`
referred to as `shared` in the following. Use your ETH account credentials.

## Preparation

1. Development tools: Download the command line Oberon Compiler/Linker/Interpreter runtime for Windows (oberon.exe) or Linux (oberon) from shared/tools

2. If you run a 64-bit linux version and observe problems executing oberon such as

   ```
   Unix.Dlopen: loading library libc.so.6 failed
   ```

   then you have to install the intel architecture 32-bit libraries (aos is currently compiled only as 32-bit application). For example, in Ubuntu the following worked for me:

   ```
   sudo apt-get install ia32-libs
   ```

   or, more recently

   ```
   sudo apt-get install libc6:i386
   ```

If you get problems with using X11 in oberon, receiving this

```
Unix.Dlopen: loading library libX11.so.6 failed
```

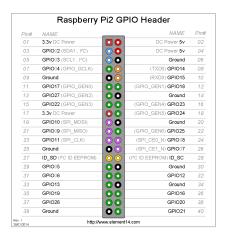you might want to install the 32-bit X11 library using

```
sudo apt-get install libx11-6:i386
```

3. If you want to see the history of commands within the Linux oberon shell, you can use the `rlwrap` tool: `rlwrap ./oberon`.

4. Download the exercise material from shared/assignments/assignment1

## Tasks

1. Connect an LED to ground and GPIO pin number 21 on the RPI2 expansion slot. The RPI2 expansion slot has the pinout displayed to the right. Note that the GPIO pin number does **not** coincide with the expansion slot pin number!



Raspberry Pi2 GPIO Header

| Pin# | NAME | | | NAME | Pin# |
|---|---|---|---|---|---|
| 01 | 3.3v DC Power | | | DC Power 5v | 02 |
| 03 | GPIO2 (SDA1 , I²C) | | | DC Power 5v | 04 |
| 05 | GPIO3 (SCL1 , I²C) | | | Ground | 06 |
| 07 | GPIO4 (GPIO_GCLK) | | | (TXD0) GPIO14 | 08 |
| 09 | Ground | | | (RXD0) GPIO15 | 10 |
| 11 | GPIO17 (GPIO_GEN0) | | | (GPIO_GEN1) GPIO18 | 12 |
| 13 | GPIO27 (GPIO_GEN2) | | | Ground | 14 |
| 15 | GPIO22 (GPIO_GEN3) | | | (GPIO_GEN4) GPIO23 | 16 |
| 17 | 3.3v DC Power | | | (GPIO_GEN5) GPIO24 | 18 |
| 19 | GPIO10 (SPI_MOSI) | | | Ground | 20 |
| 21 | GPIO09 (SPI_MISO) | | | (GPIO_GEN6) GPIO25 | 22 |
| 23 | GPIO11 (SPI_CLK) | | | (SPI_CE0_N) GPIO08 | 24 |
| 25 | Ground | | | (SPI_CE1_N) GPIO07 | 26 |
| 27 | ID_SD (I²C ID EEPROM) | | | (I²C ID EEPROM) ID_SC | 28 |
| 29 | GPIO5 | | | Ground | 30 |
| 31 | GPIO6 | | | GPIO12 | 32 |
| 33 | GPIO13 | | | Ground | 34 |
| 35 | GPIO19 | | | GPIO16 | 36 |
| 37 | GPIO26 | | | GPIO20 | 38 |
| 39 | Ground | | | GPIO21 | 40 |

Rev. 1
26/01/2014          http://www.element14.com

2. Open file RPI.MinimalLED.Mos in your favourite editor.

3. Using ARM assembler language, program the GPIO pin 21 as output pin and set the pin accordingly. You can either execute the following commands in the oberon shell

```
Compiler.Compile –p=ARM RPI.MinimalLED.Mos
StaticLinker.Link –p=RPI MinimalLED
FoxARMInstructionSet.Disassemble kernel.img
exit
```

or, equivalently, call

```
oberon execute MakeMinimalLED.txt
```

4. Compile and link the kernel. Copy `kernel.img` to the FAT partition on the SD card and insert the SD card into the (powered off) RPI.

5. Power on the RPI and check if the LED lights up upon start.

## 2   Blinkenlights

### Introduction

In this part of the exercise, we will use some high level language code in order to let the LED blink. Do not start with this second part before having solved the first part.

## Tasks

1. Open file RPI.BlinkLED.Mos in your favourite editor.

2. Read the file. On a first sight, the code should light up the LED. But it does not work (try, if you like). Why? Compile and link it. Look at the linked image and understand how things are organized. Understand what is missing in order to run the program and patch the program. Check that the LED lights up when booting.

3. Now, using high-level language features, implement a blinking LED: Compile and link the kernel. Copy `kernel.img` to the SD card and insert the SD card into the (powered off) RPI.

4. Power on the RPI and check if the LED blinks up upon start. Look at the frequency of the LED and try to estimate the rate of executed instructions per second. What is your observation? Try to explain what you find.

   (Of course, if you like, you can now also add other LED colors and play with the hardware.)

Have fun!

## Documents

- BCM2835 ARM Peripherals Technical Manual:
  shared/documents/rpi/BCM2835-ARM-Peripherals.pdf

- Lecture Slides System Construction Lecture 1 from the course-homepage
  http://lec.inf.ethz.ch/syscon

- A2 Quickstart Guide
  shared/documents/oberon/A2QuickStartGuide.pdf

More documents that are not strictly required for this exercise can be found in the repository .