Embedded and Real-time Systems

Basics of Embedded Programming

Exercise 1 – Preparing your work environment

There is an installation of the work environment for use in the unix-labs. The installation contains the following:

(i) Eclipse with CDT and OpenOCD plugin,
(ii) ARM cross-compiler [1],
(iii) OpenOCD.

To setup the work environment, you have to type the following (note the dot at the beginning!):

```
. /afs/ms/u/b/bures/ers-labs/setenv.sh
```

This will set the paths to the tools mentioned above. Please note that **this has to be executed for each opened terminal** (unless you put it to your .bashrc or .profile). Also note that you must execute Eclipse from the command line (not from the menu).

The recommended IDE (if you want to use one) for the labs is Eclipse. You may launch it by typing:

```
eclipse
```

Upon startup, Eclipse looks for a directory called 'workspace'. It keeps all projects inside. If it does not exist yet, it will create it. When Eclipse asks you for this directory either confirm or point it to the place you want to have the workspace directory.

Exercise 2 – Compiling

There is an example at GitHub https://github.com/d3scomp/stm32f4-blink. Clone it to your directory and if you are using Eclipse, import it to your workspace. To import it, choose **File->Import->General->Existing Projects into Workspace**. Select the stm32f4-*blink* folder and confirm **OK**. Check the **Copy projects into workspace** check-box and confirm **OK**. You should see a project *blink* in the left pane.

The example project is compiled via *Makefile*. You can do so by running

```
make
```

on the command line or by selecting the project (in the left pane) in Eclipse and using the context-menu (option **Build Project**) to build the project.

Exercise 3 – Flashing the Image

The compilation step creates an image for the embedded device. The image is named *blink.elf*. To upload the image to the embedded device, connect the board via USB and execute

```
make flash
```

The *Makefile* uses *OpenOCD* to connect to the board and flash it.

Note: If you are going to use your own machine you will need to make some changes to your openOCD installation, since it doesn't support the latest revision of the stm32f4 board yet. To do this you need to get the file located at

http://d3s.mff.cuni.cz/teaching/embedded_realtime_systems/files/labs/stm32f4discovery-v2.1.cfg

and put it into your openOCD installation (**openocd /share/openocd/scripts/board/**). Then change your Makefile accordingly to use this new configuration.

Exercise 4 – Connecting to the Board via JTAG

The embedded board comes with an embedded JTAG debugger. This allows you to connect to the board, inspect/modify the memory, and debug the application. You can do this from Eclipse as follows:

- Right-click on file *blink.launch*. Though a bit counter-intuitive, this loads the launch configuration to Eclipse
- Select **Debug As->blink**

Eclipse will start *OpenOCD*, *gdbserver* and show you a debug session, where you can pause the execution, place breakpoints, inspect variable, memory, etc.

Warning: Sometimes, the first start does not succeed. You will find this out in the **Console view** where the *gdb* just exits. If this happens, try again. The second run typically works.

Connect to the board this way, try to step through your program.

Exercise 5 – Using ramlog

As the very basic logging facility, the *blink* example uses a circular memory buffer as a target of all messages produced by *printf*. The buffer is located in **hardware/syscalls.c**. You can look it up there and inspect the contents of the array in the debugger.

Alternatively, you can look up the address of the *ramlog* array in **blink.map**. Then use the **Memory** view under **Window->Show View** and add the address you want to inspect.

Note that in order to see the memory dump, the application has to be paused (on **breakpoint** or via the **Pause** button in the toolbar).

Exercise 6 – Using SWO console

You may have noticed **ITM_SendChar** calls in *_write* function used for writing to *ramlog*. This call enables listening to output console using OpenOCD. There is a make target called SWO set for this purpose.

```
make swo
```

Exercise 7 – Using LEDs

Use the 4 LEDs to blink in a cycle: green, orange, red, blue (or similar).

Exercise 8 – Using the button

Hook to the button press to change the blink pattern: clockwise, counter-clockwise, all blinking together. Do it in such a way that any subsequent button presses after the first press are ignored for 50 ms.

Exercise 9 – Using peripherals

The board comprises a number of peripherals – the GPIO used to turn on/off the LEDs and read the button is just one of them. The resources you may need for understanding follows. Note that they are to be used only for reference in case you have to study a certain aspect or procedure related to the system. You are not expected to study them in full.

The board you are working on is a combination of several things:

- System on chip, which comprises various peripherals and the processor
- Embedded board, which comprises the system on chip, power module, LEDs, buttons and generally pinout of the processor signals.
- Specification of the embedded board (STM32F4-DISCOVERY) [2,3]
- Specification of the system on chip (STM32F407VG) [4,5]
- Specification of the processor and programming manual (interrupts, instructions, etc.) [6,7]

Firmware and examples (including examples for STM32F4-DISCOVERY) [8]. Link for download is on the bottom of the page. The downloaded package contains documentation to the API you should use when programming the board. Link to documentation PDF is [9].

Take a look at the reference documentation and try to see what steps are taken by the sample code to initialize the board and operate the LEDs. Try to see where this is described in the reference manuals and in the API documentation.

Exercise 10 – UART

UART is a standard way of serial communication [10]. The board provides several UART channels which can be used. You are provided *USB to Serial* cable. Connect it to the UART2 pinout on the board. This allows you to connect the board to a PC via USB and establish a 2-way communication – or at least you can show data coming from the device on a PC using a serial terminal (e.g. **moserial** or **minicom**). Write your own driver for UART (transmit from the board is enough). Then use it in the implementation of *_write* in **syscalls.c**. Once you do this, you can use *printf* in the embedded code to display messages on the PC.

Hints:

- Consult the API of firmware to see how to use it to initialize and operate the UART
- The steps you will need to perform in initialization are the following:
  - Enable GPIO module. This powers on the respective part of the chip.
    - **__HAL_RCC_GPIOA_CLK_ENABLE()** [9]
  - Create a structure to define the desired function of the correct pins and fill it with the right values

        - **GPIO_InitTypeDef** [9]
    - Initialize the pins using the structure above
        - **HAL_GPIO_Init** for **GPIOA** [9]
    - Enable UART2
        - **__HAL_RCC_USART2_CLK_ENABLE** [9]
    - Create UART handle and fill it with the proper values
        - **UART_HandleTypeDef** [9]
    - Initialize the UART handle
        - **HAL_UART_Init** [9]

Example of values for GPIO pins

Pin	GPIO_PIN_2 | GPIO_PIN_3
Mode	GPIO_MODE_AF_PP
Pull	GPIO_PULLUP
Speed	GPIO_SPEED_HIGH
Alternate	GPIO_AF7_USART2

Example of values for UART handle

Instance	USART2
BaudRate	921600
WordLength	UART_WORDLENGTH_8B
StopBits	UART_STOPBITS_1
Parity	UART_PARITY_NONE
Mode	UART_MODE_TX_RX
HwFlowCtl	UART_HWCONTROL_NONE
OverSampling	UART_OVERSAMPLING_16

When configuring the serial terminal remember to use the same values as on the board.

The pinout is as follows: black – GND, green – RX, white – TX.

Exercise 11 – Interrupts

Extend the UART driver to offer an "echo" mode. In this mode, it transmits back whatever it receives from the PC. Use UART interrupt for handling the reception.

Hints:

- You need to provide a correctly named function (to **stm32f4xx_it.h** and **stm32f4xx_it.cpp**) which will serve as the UART handler. Look into **hardware/startup_stm32f4xx.s** – the interrupt handler symbols are defined there.
- You have to configure UART to send you interrupts for events you are interested in: **__HAL_UART_ENABLE_IT** [9]
- In order to make the interrupt work it is also necessary to enable it on the interrupt controller: **HAL_NVIC_EnableIRQ** [9]
- You may also need to set the interrupt priority: **HAL_NVIC_SetPriority** [9]
- When configuring the serial terminal, make sure you switch off any handshake / control flow.

Exercise 12 – Watchdog

The board comes with a device (called watchdog), which resets the board if the watchdog does not receive a periodic notification.

Create a driver for the watchdog and configure it to expect a notification every 1s. Update the *systick* handler to provide the notification.

Test the watchdog as follows:

- Configure the priority of the button interrupt to be higher than the one of the *systick*.
- When the button is pressed, enter an endless loop within the button handler.
- The watchdog should reset the device to up to 1s after the button is pressed.

Hints:

- There are two watchdogs on the board – independent and window watchdog. It does not matter which you use. However, independent watchdog (**IWDG**) may be possibly a slightly better fit for you.

(Optional) Exercise 13 – Play with a sensor

On the board there is integrated 3-axis accelerometer. Use it and read the data it provides.

References

1. ARM cross-compiler:
   https://launchpad.net/gcc-arm-embedded/
   https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads
2. STM32F4 DISCOVERY:
   http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419?sc=internet/evalboard/product/252419.jsp
3. Discovery kit with STM32F407VG MCU, User manual:
   http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf
4. STM32F407VG:
   http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1577/LN11/PF252140
5. RM0090 Reference manual:
   http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf
6. STM32F4 Series instructions:
   http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1577
7. PM0214 Programming manual:
   http://www.st.com/web/en/resource/technical/document/programming_manual/DM00046982.pdf
8. STM32F4 cube:
   http://www.st.com/web/en/catalog/tools/PF259243
9. Description of STM32F4xx HAL drivers:
   http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00105879.pdf
10. Universal asynchronous receiver/transmitter:
    https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter
11. SHT1x click:
    http://www.mikroe.com/click/sht1x/