

Real-Time Embedded Systems, 7.5 hp

Assignment 1

Before you start the Assignment

Check the “Practical Assignments and Project Seminar” learning module on the course page on Blackboard. It includes general information about the assignments and the Lab Kit.

After you complete Assignment

Once you have completed the tasks for Assignment 1, one student in the group will submit the results via the course page on Blackboard.

The assignment submission form is found within the learning module “**Submissions**”.

Grading and Deadline

- **Grading Session:** Check the date and time in Time edit for “Grading Ass. 1”.
- **Deadline:** By the day before the grading session at **11PM**.

Objectives

- Set up and become familiar with the lab environment (e.g., Raspberry Pi (RPI) and ARM Cross Compiler).
- Gain a deep understanding of fundamental C programming concepts essential for the course.
- Develop basic skills in Raspberry Pi bare-metal programming.

Content

Assignment 1 includes two parts, beginning with preparatory work:

- Part 1 involves implementing bitwise operations in C and exploring data exchange between the RPi and a PC via serial UART communication.
- Part 2 focuses on developing firmware/kernel to control an LED through RPi GPIO.

Q&A

Supervision sessions will address questions, and students are encouraged to share relevant queries or comments on the Discussions on the course page on Blackboard. **Note that posting source code is not allowed.**

Required Equipment and Software:

- Lab kit
- Raspberry Pi OS
- Serial console software like PuTTY
- GNU Arm Embedded Toolchain
- A Windows host requires Make for Windows

NOTE: The teachers will not provide support for technical questions/problems related to alternative software tools.

References

- [Valvers Bare Metal Programming in C](http://www.valvers.com/open-software/raspberry-pi/step01-bare-metal-programming-in-cpt1)¹
- [Cambridge Baking Pi](https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/index.html)²
- [C reference](https://en.cppreference.com/w/c)³ (Recommended by Wagner)

Preparation 1: Raspberry Pi OS

The MicroSD card in the Lab Kit is blank, and installing [Raspberry Pi OS](https://www.raspberrypi.com/software)⁴ is needed because, unlike most PCs, the Raspberry Pi does not have a BIOS or UEFI. Instead, its boot process is handled by the GPU and relies on specific files on the FAT32 boot partition.

Please refer to the [installation guide](https://www.raspberrypi.com/documentation/computers/getting-started.html#installing-the-operating-system)⁵ to install Raspberry Pi OS.

- Consider the 'Install an Operating System' section.
- The most straightforward approach is to use the Raspberry Pi Imager. You can find a comprehensive guide on how to use it in the embedded YouTube video.

Note: We recommend the Raspberry Pi OS Lite version for faster installation. However, Raspberry Pi OS Desktop is also an option for Linux newcomers.

Preparation 2: C Programming

- Ensure the RPi can boot Raspberry Pi OS; otherwise, consult the installation guide.
- If necessary, familiarize yourself with writing and compiling C code.
 - While developing on another PC may be more convenient, using the RPi for code writing and testing is also an option.
 - Web-based C compilers can be utilized for coding and testing purposes.
- Copy `resetbit.c` file to the RPi or its content to a file that you will name `resetbit.c`.
 - The file resetbit.c is provided as part of Assignment 1 materials.
 - Recommended destination directories:
 - ~/ (your home directory)
 - /home/pi/ (default home for the pi user)
- Using a text or code editor, open the `resetbit.c` file.
- Try to understand what the program does. Some of the logic will be useful during this assignment.
- To compile the program, type the following command in the command line:
`gcc /PATH_TO_C_FILE/C_FILE.c -o /PATH_TO_EXECUTABLE/EXECUTABLE`
 - For example: `gcc resetbit.c -o resetbit`
- If the compilation succeeds, run the executable.

¹ <http://www.valvers.com/open-software/raspberry-pi/step01-bare-metal-programming-in-cpt1>

² <https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/index.html>

³ <https://en.cppreference.com/w/c>

⁴ <https://www.raspberrypi.com/software>

⁵ <https://www.raspberrypi.com/documentation/computers/getting-started.html#installing-the-operating-system>

Preparation 3: Bare-Metal Programming for Raspberry Pi

Cross Compiler for ARM

Bare metal programming refers to writing software that runs directly on a computer's hardware without an operating system. Unlike typical applications that rely on an OS like Linux to manage hardware and provide services, bare metal programs must handle everything themselves—like input/output, memory, and device control. This approach gives developers full control over the machine from the moment it powers on, allowing them to define exactly what the computer does. It's commonly used in embedded systems, firmware development, and low-level hardware programming.

The Raspberry Pi 3 Model B+ is a low-cost, compact computer powered by a 1.4 GHz 64-bit quad-core Arm Cortex-A53 processor. It's a widely used device across education, hobbyist, and professional communities, making it an exciting platform for bare metal programming. The RPi 3 B+ also comes equipped with useful onboard hardware like GPIO pins, networking interfaces, and multimedia capabilities, which can be leveraged in your bare metal projects.

To compile code, in most cases in C, for bare-metal execution on RPi 3, you'll need a bare-metal C compiler designed for embedded ARM chips with Cortex-R/M processors. The GNU Arm Embedded Toolchain is a suitable open-source option. Installation instructions are provided for both Windows OS and Raspberry Pi OS.

NOTE: Writing and compiling code on a different PC than the RPi would likely be more convenient for the assignments.

Choose one of the following installation options based on your selected platform for writing and compiling C code.

Option 1: Windows OS

- Download and install the [gcc-arm-embedded Arm Cross Compiler Toolchain](#)⁶.
- Download and install the [Make for Windows](#)⁷. Make is a tool that automates building programs.
 - The most straightforward approach is to download and unzip the Binaries package as a .zip file. Then, download the dependencies zip file and extract the 'libintl3.dll' and 'libiconv2.dll' files into the 'make's bin' folder or directory as described in the Make for Windows's web site.
- **NOTE:** it is important that your Windows PATH variable includes the path to gcc-arm-none-eabi and make bin folders.

Option 2: Raspberry Pi OS

- Install the gcc-arm-none-eabi GCC cross compiler using the command:
`sudo apt-get install gcc-arm-none-eabi`

⁶ <https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads>

⁷ <http://gnuwin32.sourceforge.net/packages/make.htm>

Compiling source code using the Cross compiler

For the assignments, compiling source code is facilitated by the gcc-arm-embedded Arm Cross Compiler Toolchain and the accompanying makefile.

The Makefile file takes care of all compiler options so that you only have to execute the following command in the command prompt to generate a .img file, which is pre-configured to be rteskernel.img.

```
make run
```

Changing the kernel to be loaded

Configuration files manage kernel loading and are adaptable for changing kernels as needed.

- Make sure you read about the RPi configuration files.
- A few files of the Raspberry Pi OS are stored on the MicroSD card. Some of these files contain the actual kernel to be loaded, while others are for configuration.
 - The kernel.img and kernel7.img files are kernels for the boot partition. Kernel.img is the default for Pi 1 and Pi Zero, while kernel7.img is the default for Pi 2 and Pi 3. Students will develop and name new kernels as instructed.
 - The [config.txt](https://www.raspberrypi.org/documentation/computers/config_txt.html)⁸ configuration file configures boot options, specifying the kernel loading filename. Add a line like
kernel=rteskernel.img
to indicate loading rteskernel.img instead of the default. Ensure the kernel option points to the desired kernel whenever you want to load a new kernel.
 - Every time you want to load a new kernel, you must ensure the option kernel indicates the kernel to be loaded.
- Given that the command make run succeeded and that it created a new .img file
 - Copy the new .img file into the MicroSD card.
 - Change the config.txt file in the MicroSD card by changing the value of the property kernel to the new .img file name.
- Remove the micro SD card from the computer and insert it into the RPi.
- Power on the RPi.
- The new kernel shall now execute on the RPi.

⁸ https://www.raspberrypi.org/documentation/computers/config_txt.html

Preparation 4: Bare-Metal Serial Communication

In the assignments, students will leverage a library enabling bidirectional communication between the RPi and a PC via a TTL USB serial cable. This library explores serial UART communication using RPi's serial port. Students will need the provided TTL USB serial cable⁹ and a serial console software, such as PuTTY, installed on the PC.

Connecting the TTL USB serial cable to the RPi.

Connect the **BLACK** cable (GND ground) to GPIO pin 6. Then, Connect the **WHITE** cable (TXD transmission) to GPIO pin 8. Lastly, connect the **GREEN** cable (RXD receiving) to GPIO pin 10.

DON'T connect the **red** cable (VCC power) because you will power the RPi using the power supply with the micro-USB cable. You should not have both.

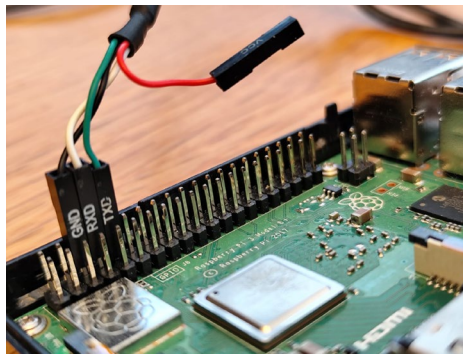


Figure 1. Black, White and Green connected to the most external pin row starting from the 3rd.

On the Pi 3 and 4 the UART is, by default, used for Bluetooth. Thus, you need to add the following lines to config.txt:

- enable_uart=1
- dtoverlay=disable-bt

To set up and test serial communication

1. Install serial console software (e.g., PuTTY) on your PC.
2. Connect the TTL USB serial cable to the PC.
 - a. A device driver might be needed for the [black](#)¹⁰ or [blue](#)¹¹ TTL cables.
3. Check which serial port was attributed to the TTL USB serial cable.
4. Configure the baud rate to 115200 and establish a serial communication session.
5. Copy the provided `helloworld.img` file to the MicroSD card.
6. Adjust the `config.txt` file's kernel property to match the `helloworld.img` kernel.
7. Insert the MicroSD card into the RPi and power it on.
8. The new kernel should execute on the RPi as intended (see Figure 1).

⁹<https://www.adafruit.com/product/954?srltid=AfmBOooFe00IjIiFv7H8IJ0SuM93KJUXy5udZVpymbnmynAwSMKH2FU9>

¹⁰https://www.silabs.com/documents/public/software/CP210x_Universal_Windows_Driver.zip

¹¹http://files.dlink.com.au/products/DGS-3630-52TC/REV_A/Software/USB-to-Serial_PL2303_Prolific_DriverInstaller_v1.12.0.zip

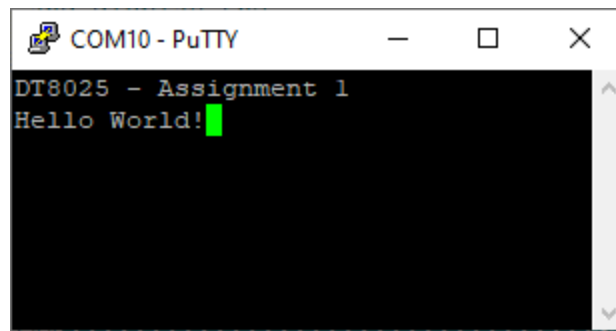


Figure 2. Expected results from helloworld.img kernel

PART 1 - Bitwise operators in C

Bitwise operators in C are commonly employed to manipulate bit sets and bit masks. The objectives of Part 1 are to learn how to utilize these operators and encapsulate them within functions. More specifically:

- Include the necessary documentation as comments in ``iregister.h``.
- Implement the functions in ``iregister.c``.
- Document **post** and **pre-conditions**.
 - Pre-conditions are the requirements or assumptions that must be true before the function is executed. Example: A pointer passed to the function must not be NULL, or a register address must be valid.
 - Post-conditions are the expected outcomes after the function has executed. Example: A register value has been successfully updated, or a bit has been cleared.
- Create a user interface that displays the results of each function.

REMEMBER

- C declarations and macro definitions belong in a header file (.h).
- Documentation for function declarations should include what the function does, its parameters, and return value.
- The actual implementation is placed in the .c files, and documentation should detail how a function accomplishes its task.

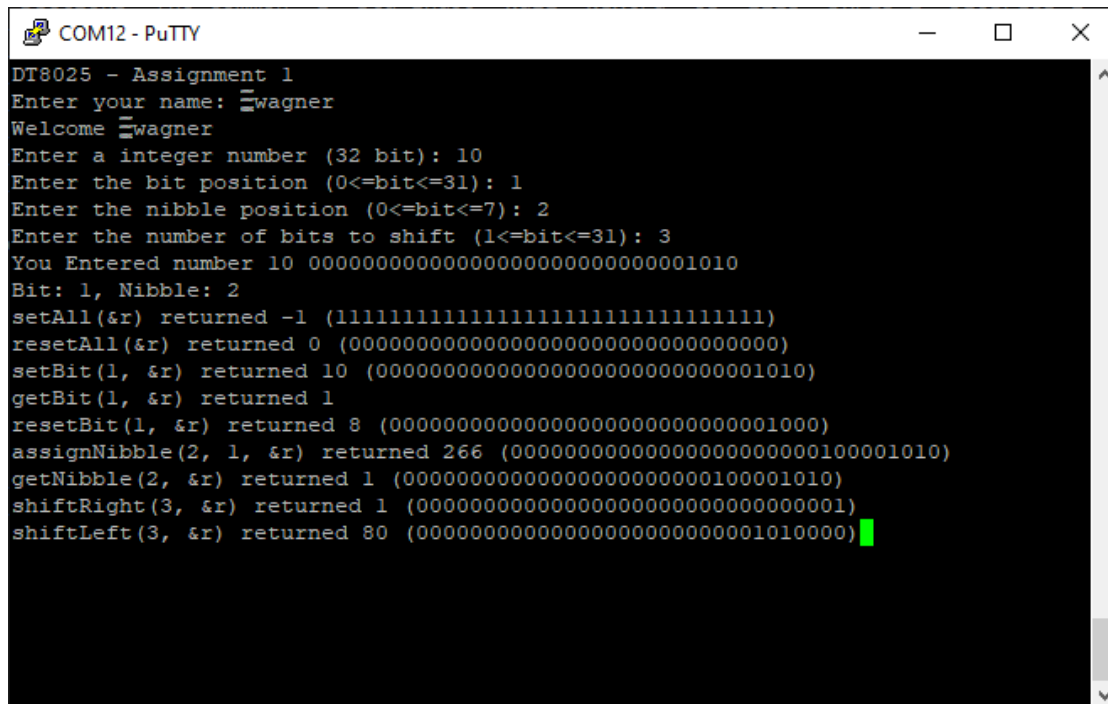
Procedure

1. Download the provided a1p1.zip file and unzip it.
2. Familiarize yourself with the ``iregister.h`` file.
 - Start by understanding the `iRegister` data structure and the function declarations that can modify and display it. Pay special attention to the "void resetBit(int, iRegister *)" function declaration in ``iregister.h`` as it serves as an example of the task's expectations.
3. Document the functions in ``iregister.h``.
 - Prior to each function declaration, follow these steps:
 - i. Include a comment block with a brief description.
 - ii. Describe all function parameters.
 - iii. Specify the return value.
 - iv. Pre-conditions
 - v. Post-conditions
 - vi. Properties: Specify at least some properties that the function should satisfy.

Obs.: Refer to the "void resetBit(int, iRegister *)" function declaration in `iregister.h` for reference.

4. Implement the functions in ``iregister.c``.
 - Consult the ``iregister.h`` file, where each function's purpose and declaration are provided.
 - Implement all functions declared in ``iregister.h`` within the `iregister.c` file.
 - The ``iregister.c`` file already contains an example implementation of the `resetBit` function.
5. Utilize the functions in ``a1p1.c``.
 - In ``a1p1.c``, use all the functions and display their results (see Figure 3. Please note that Figure 3 might not present correct numbers).
 - Allow the user to input an integer, specify the bit and nibble to manipulate, and indicate the number of bits to shift, among the other inputs (see Figure 3).

6. After completing your code, compile it and boot the RPi using the newly created kernel, i.e., `a1p1.img`.



```
COM12 - PuTTY
DT8025 - Assignment 1
Enter your name: wagner
Welcome wagner
Enter a integer number (32 bit): 10
Enter the bit position (0<=bit<=31): 1
Enter the nibble position (0<=bit<=7): 2
Enter the number of bits to shift (1<=bit<=31): 3
You Entered number 10 00000000000000000000000000001010
Bit: 1, Nibble: 2
setAll(&r) returned -1 (11111111111111111111111111111111)
resetAll(&r) returned 0 (00000000000000000000000000000000)
setBit(1, &r) returned 10 (00000000000000000000000000001010)
getBit(1, &r) returned 1
resetBit(1, &r) returned 8 (00000000000000000000000000001000)
assignNibble(2, 1, &r) returned 266 (000000000000000000000000100001010)
getNibble(2, &r) returned 1 (000000000000000000000000100001010)
shiftRight(3, &r) returned 1 (00000000000000000000000000000001)
shiftLeft(3, &r) returned 80 (00000000000000000000000001010000)
```

Figure 3. User interface with results (for illustration, thus might not be accurate).

NOTE:

- Be careful with the sign bit when implementing the shift right function.
- Be careful to use the Null terminator at the end of the array in the reg2str function.
- There are 8 nibbles in the register content. They range from 0-7 or 1-8.
- Implement the pre-and post-conditions for all functions.

NOTE: If the Lab Kit includes the TTL USB serial, it is recommended to input and output data via the serial console. The a1p1.c file hints at two functions in `uart.h` that can be used for input and output. The purpose is to learn how to use serial communication for debugging purposes.

NOTE: If the Lab Kit does not include the TTL USB serial, refer to the `resetbit.c` file provided in the Assignment 1 preparation. In this case, use the standard GCC compiler to compile the code to run with the support of the Raspberry Pi OS.

Deliverables

For Assignment 1 Part 1, one student in the group must upload (click on the title of this section) the following individual files:

- iregister.c
- iregister.h
- a1p1.c

Note:

- All students in the group are equally responsible for the submitted source code.
- The group ensures that the submitted code does not include cheating and plagiarism issues.

PART 2 - Bare-metal LED control via the Raspberry Pi GPIO

Procedure

1. Lighting a LED

- Utilizing the breadboard, a 270Ω resistor, LED, and cables provided in the Lab Kit, construct the circuit outlined in the [Raspberry Pi Projects](#)¹².
- Ensure that RPi's +3.3v and 0v pins are correctly connected to the breadboard. When you power up the RPi, the LED should illuminate.

NOTE: This step is purely for circuit testing.

2. Theory for controlling a LED via the RPi GPIO

- Any GPIO pin can control an LED.
 - i. The GPIO pin numbers can be found in [RPi's GPIO usage guide](#)¹³ or the [RPi Pinout](#)¹⁴.
- To control an LED through software, set the GPIO pin as an output by configuring the respective bit in the GPIO Function Select Register (base address 0x3F200000).

NOTE: Embedded programming often requires familiarity with the hardware architecture, including reading datasheets. In Part 2, students are advised to consult the RPi datasheet for the [BCM2837 ARM Peripheral](#)¹⁵ chip, which controls various peripherals in the RPi, including GPIO and serial interfaces. Specifically, refer to page 90 of the datasheet.

3. Controlling a LED via the RPi GPIO

Now that you are more familiar with the hardware platform, to turn the LED on and off, you will need to iteratively set the respective bits of the GPIO Output Set and GPIO Output Clear registers with some delay in between.

- Connect the breadboard to GPIO16 (pin 36) and GND (pin 39).
- Download the provided `a1p2.zip` file and extract its contents.
- Compile the code using `make`, modify the `config.txt` file accordingly, and boot the RPi.
- The LED should momentarily illuminate.

4. GPIO-controlled LED

- In `led.c`, implement the `led_blink()` function to make the LED turn on and off with a delay before and after turning it off.
 - i. Ensure comprehension of the code within the `led_on()` and `led_off()` functions, including working with GPIO and setting outputs low and high.
 - ii. Achieve blinking by setting the respective bit of the GPIO Output Set and GPIO Output Clear registers, with a delay before and after setting the Output Clear register.
 - iii. Use a 0.5 second delay and create it using the `RPI_WaitMicroSeconds` function defined in `lib/rpi-sys timer.h`.
- In `led.c`, implement the `led_toggle()` function so that it turns the LED on if it is currently off, and turns it off if it is currently on.
- In `a1p2.c`, within the `main` function, call the `led_blink()` function within an infinite loop.

¹² <https://projects.raspberrypi.org/en/projects/rpi-connect-led>

¹³ <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#gpio>

¹⁴ <https://pinout.xyz>

¹⁵ <https://github.com/raspberrypi/documentation/files/1888662/BCM2837-ARM-Peripherals.-.Revised.-.V2-1.pdf>

- Once your code is ready, compile it and boot the RPi using the newly created kernel, i.e., `a1p2.img`.
 - i. Consequently, the LED will blink every second.
 - ii. Also, check the Makefile to understand how to generate a file named `a1p2.img`.

Deliverables

For Assignment 1 Part 1, one student in the group must upload (click on the title of this section) the following individual files:

- led.c
- a1p2.c

Note:

- All students in the group are equally responsible for the submitted source code.
- The group ensures that the submitted code does not include cheating and plagiarism issues.