Lab 1

Embedded Programming

Authors:

Jonas Lussi, Alexandre Mesot, Naveen Shamsudhin and Prof. Bradley J. Nelson

Multiscale Robotics Lab, Institute of Robotics and Intelligent Systems

Date: 2021

Version: 1.2

Summary: Last week, we have built a program that reads two 16 bit hexadecimal numbers from the user, and displays the summation in binary format on the console. In this lab, we will try to bring this binary representation into our physical world using light emitting diodes (LEDs). The objective of this lab is to introduce the principles of embedded computing along with a simple architecture. This week, we will learn:

- Architecture of embedded computing
- Programming a micro-controller
- Communicating with a micro-controller
- Accessing digital input-output (IO) pins on the micro-controller
- Controlling digital IO pins of a micro-controller programatically from a PC

1.0.1 Embedded Systems

Embedded systems are computing platforms that serve dedicated functions in complex mechatronic systems, ranging from ticket-vending machines, smartphones, industrial robots, and cars to spacecrafts. Unlike personal computers (PCs), embedded computers are purpose-built for higher immunity to hardware/software failure, low power consumption and smaller form factors which have led to their ubiquity in almost all consumer, healthcare, automotive and industrial products and devices. The processing unit in an embedded system is typically a microcontroller, but more complex systems can have multi-core microprocessors, digital signal processors (DSPs), application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs) or even neuromorphic processors. Increased fault-tolerance, real-time capabilities, and scalability of these circuits have led to fly-by-wire systems in aircrafts and automation in automobiles replacing or complementing the existing mechanical and hydro-mechanical control systems. The level of sensing and automation in cars is well summarized in Fig 1.1. A typical low-end car out of today's production lines can have as many as 30-40 electronic control units (ECUs), with top-of-the-line models boasting upto 300 ECUs. Decreasing form factor and reduced power requirements of embedded systems have led to concepts of wireless-sensor networks and the Internet of Things (IoT).

Figure 1.1: Mechatronic architecture of an automobile.

1.0.2 Conceptual Architecture

We will be using two separate but equally important components during the IRM labs:

μP which in this case is the CPU of your Laptops. The operating system runs on this unit, and this gives the μP the ability to operate commercial peripheral units such as mouses, keyboards, screens, and Ethernet units. In a sense, there is the operating system between the raw hardware capabilities of the chip, such as its pins and its precise timing, and the user; making it considerably complex to access and control those capabilities precisely in a close to real-time fashion.

μC which acts as the interface to external circuits and effectively to the physical world. This unit does not have an operating system, and has the capability to execute the specific code that is dumped onto it. The absence of an operating system gives full control of the timing and an extensive access to all the pins of the chip.

In our case these chips are two distinct entities having a communication channel in between to establish any cooperation (USB serial communication). The architecture we rely on is the one represented in Fig. 1.2.

When the mouse or the keyboard is used, the activity is captured by the operating system running on the μP. Depending on the application, if a change of action is required on the hardware that is connected to the pins of the μC, this is communicated to the μC; provided that it is expecting an incoming command and knows how to understand it.

An example to this sort of use could be the following. Assume that you have one of these μC boards (like the ESP32 Feather Board we will introduce the next section 1.0.3) and you want to control the light and the blinds in your room through your computer at the press of a button. First, you should find a way to connect both the motor of the blinds and the light to the appropriate pins on the μC. Then you need an application that can understand your mouse and keyboard commands and translates those into actions that should be performed by the μC. Remember what we said in the previous paragraph. The μP can understand the peripherals, and the μC can control the pins. Since these two are two distinct entities, effectively, you should have two programs running on both and communicating with each other. One that runs on the μP should take the peripheral inputs and send these to the μC. And the one on the μC expects the communication, knows how to understand it, and executes the command in the physical world through its hardware pins.

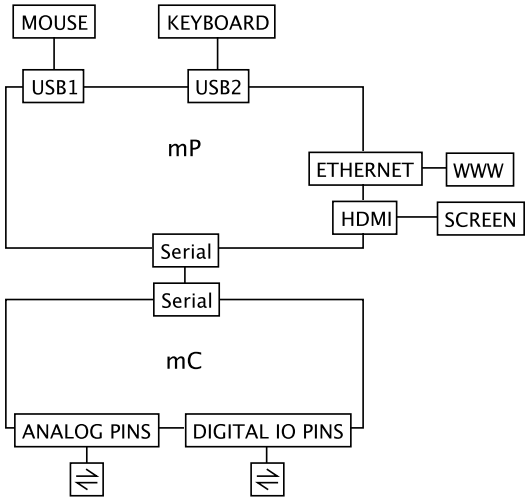

Figure 1.2: The conceptual architecture of the system.

1.0.3 Adafruit HUZZAH32 – ESP32 Feather Board

The Adafruit Feather Board (in short: Featherboard) is a powerful yet small, arduino-compatible microcontroller (μC) [1] (Fig 1.3). It featues a wide range of general purpose Input/Output pins (GPIO) and analog inputs, two analog outputs, extra peripherals like a spare UART and two cores (Fig 1.4). Using the GPIO pins more than 50 "Wings" (or shields) can be attached to the Feather Board in order to add all sorts of accessories and expand its capabilities, like an OLED screen, multiple servo/stepper motor controls, touchscreens and much more [2]. You will be using the Arduino IDE software on your computers in order to write programs and routines for the Feather Board and upload them.

1.0.3.1 Interfacing Sensors and Actuators

Various analog and digital sensors and actuators can be interfaced to the Feather Board via the general purpose input-output (GPIO) pins on the PCB. There are more than 10 GPIO pins on the PCB and each pin can be set to either INPUT or OUTPUT mode and can be controlled with both the μP and the μC. Some pins that can be used as GPIO can also be used for other purposes, like GPIO2 that can also be used as an Analogue to Digital Converter (ADC). In the lab, we will learn how to access the GPIO pins using the μC. The exact location of the pins can be seen in Figure 1.4. For more details on the pinout refer to [3].

1.0.3.2 Programming the microcontroller (μC)

The Featherboard microcontroller can be programmed using the Arduino Integrated Development Environment (IDE) which is pre-installed with the operating system of the virtual machine we provided you. The process of programming or `burning` the μC involves the transfer or download of the compiled code into the FLASH memory of the microcontroller (Fig 1.5).

[1] https://www.adafruit.com/product/3591
[2] https://www.adafruit.com/category/814
[3] Pinout of the Feather Board https://learn.adafruit.com/adafruit-huzzah32-esp32-feather/pinouts

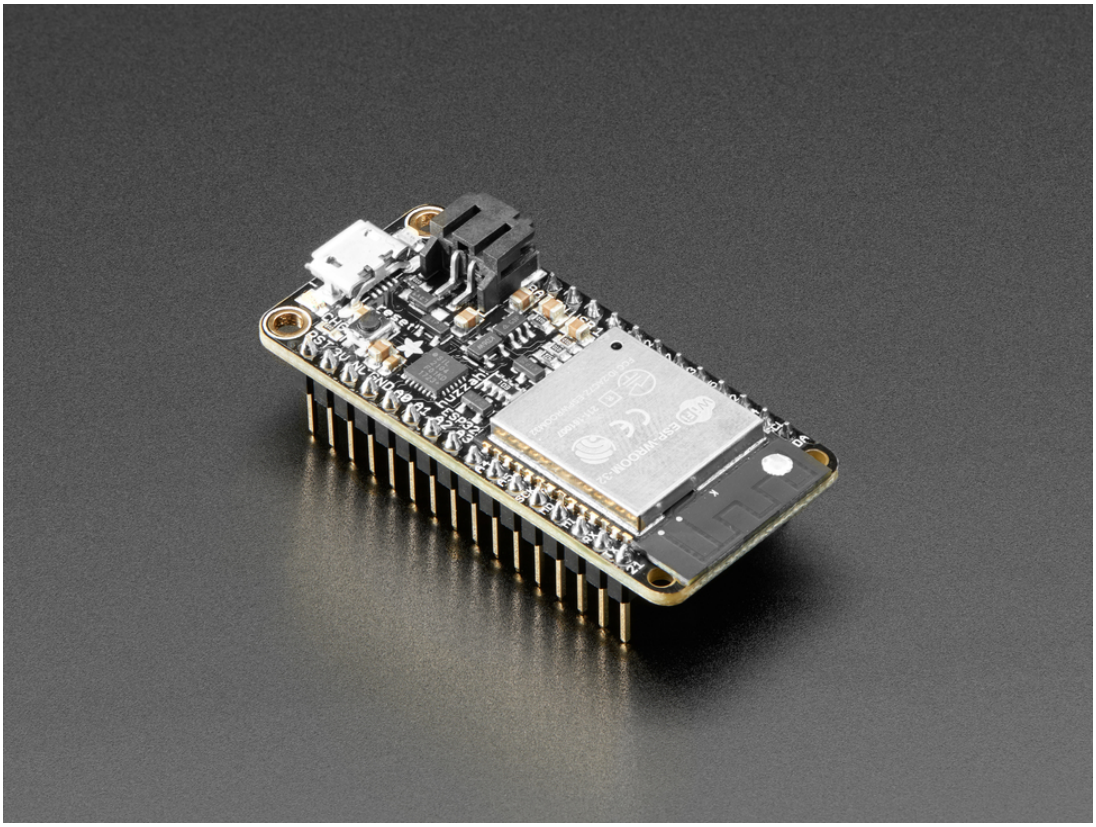

Figure 1.3: Adafruit HUZZAH32 - ESP32 Feather Board

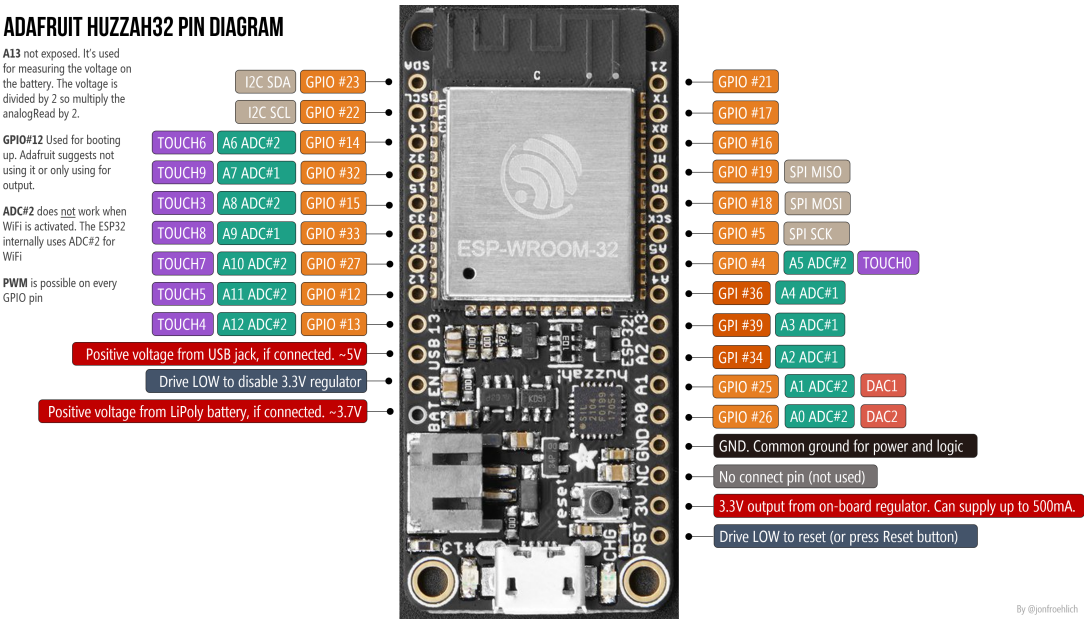

Figure 1.4: Here you can see the pinout of the Featherboard. As you can see there are a lot of GPIO pins and many can be used for multiple purposes.

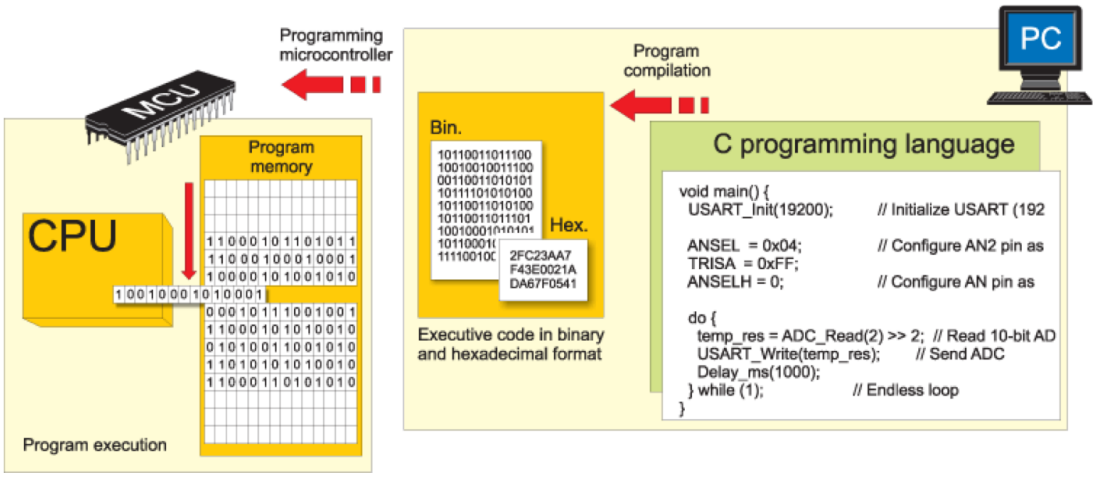

Figure 1.5: The C code is compiled into hex or binary code which is downloaded/uploaded onto the μC FLASH memory.

1.1 Prelab procedure

Note: The Prelab quiz on Moodle must be done before reporting to the lab. The submission deadlines are on Moodle. Late submissions will not be corrected. Each group member has to complete the quiz on their own, however you are allowed to work with students in your group to solve the quiz (and we absolutely recommend you to do so).

1. Read through the following documents and webpages:
   (a) Adafruit HUZZAH32 Feather board Overview:
   https://learn.adafruit.com/adafruit-huzzah32-esp32-feather/overview
   (b) Adafruit HUZZAH32 Feather board Pinout:
   https://learn.adafruit.com/adafruit-huzzah32-esp32-feather/pinouts
   (c) (Very Optional) If you want an in depth and very technical understanding of the ESP32 chip feel free to take a look at the technical reference manual:
   https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

2. On the Ubuntu VM, open the Arduino IDE and look at some code examples under "File -> Examples" in order to familiarize yourself with the IDE and the syntax.

3. Solve the Prelab Quiz on Moodle. This week there will be no coding exercise on Code Expert for the Prelab.

1.2 Lab procedure

On your lab bench, you will find a Featherboard pinned on an electronics breadboard. Please do not remove it from the breadboard. Inspect the Featherboard and see if you can identify the GPIO ports as shown in Fig 1.4. Make sure that you can identify a GND, a 3.3V, and the GPIO pins on the board because you will be using them in this lab. Connect your Laptop running the VM to the Featherboard using the microUSB to USB cable. Make sure to always keep the USB hub between your Laptop and the Featherboard. On one hand it will power the Featherboard using its dedicated power supply. On the other hand its over-current protection will protect your laptop from short-circuits that might happen if you are not careful where you plug in the cables in the pins of the Featherboard. Try to upload an example code to make sure the Arduino IDE recognizes the Feather Board and can connect to it.

Note: If you are having connection issues, make sure the VM has access to the USB port that connects the Feather board to your computer. Check the Appendix for the correct setup of the USB-Connection. Also make sure that, in the Arduino IDE, the port `ttyUSB0` is selected.

1.2.1 Circuit Preparation

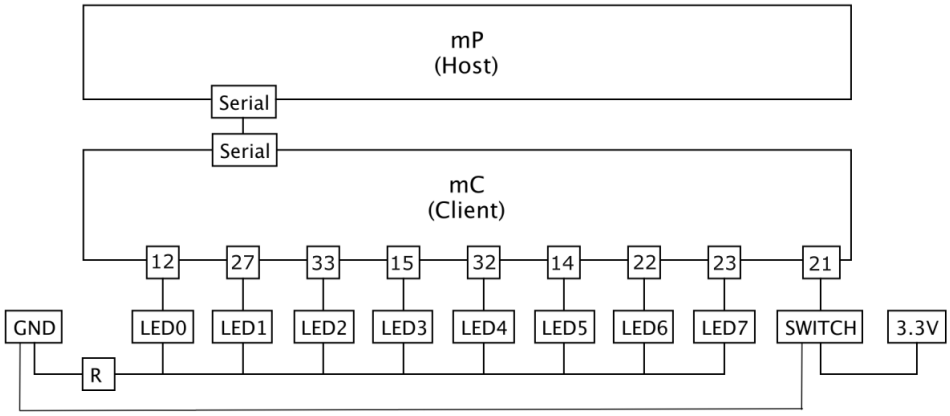

Figure 1.6: The circuit layout for this lab.

1. Put the circuit given in figure 1.6 together. On your tables, you will find a group of cables put together along with the circuit elements you will need for this lab. Once you arrange the circuit elements (LED, switch, resistor, etc.) on the breadboard, you can plug the cables into the Feather Board and the breadboard. For the LEDs, use the pins indicated in the schematic. You will find that they correspond to the order of GPIO pins on the Feather; for the switch use GPIO pin 21 on the Feather Board. Make sure your 3.3V output is **not** connected directly to the ground. This could cause the current to flow back into your laptop and damage it. **Make sure to connect the cable from the GPIO 21 pin to the central pin of the switch.**

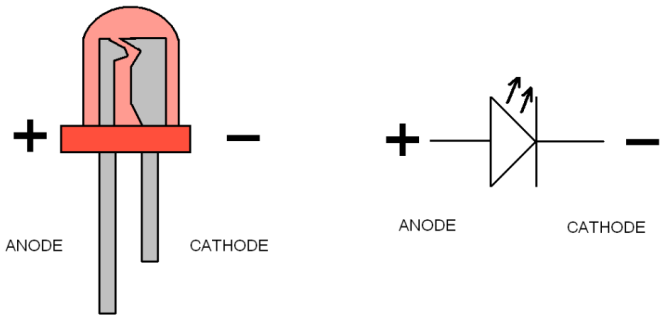

Figure 1.7: Notice the anode and cathode of the LED. For a sample LED circuit, check out the following site http://www.build-electronic-circuits.com/what-is-an-led/.

1.2.2 Digital Output

1. Extract the package Lab01.zip into the folder `irm` on the `Desktop`. Create `irm` if it is not there already. Once you extract the package, you should have two folders, `mP` and `mC`, in `Lab01`.
2. In `mC` there is the `mC.ino` that contains a part of the code to be put on the μC. Open it using the Arduino IDE.

   Hint: What you see is the basic structure of a typical code for the μC in this file. As you can see, there are two main parts to the code. The first part, function `setup()`, is for the initialization and setting up of the μC functionality to be used. The second part, function `loop()`, is for the periodic action to be carried out by the μC after the initialization. The initialization code is run as soon as the board is connected to the power. Right after the initialization, μP starts going through the loop indefinitely until it is disconnected from the power.

3. Understand which IO pins are being utilized and what is being done in `loop()`, and implement the `setup()` accordingly. You will need to use the function `pinMode()`. **(PostLab Q1)**
4. Upload the code onto the μC following the procedure explained in Section 1.0.3.2 (the code can be uploaded to the μC by clicking on the arrow in the top left corner). Observe the results on the circuit. Also, try different resistors and see what happens. Explain. **(PostLab Q2)**

   Hint: If you are working with the VM we provided you might experience issues when connecting to USB ports. If Arduino cannot find the USB port ttyUSB0 go to *Virtual Machine -> USB & Bluetooth* and select **Connect Silicon CP2104 USB to UART Bridge Controller**.

5. Notice that in the code (part C) the `delay('delay in milliseconds')` command manages the timing. Try different values for the associated variables to see the effects.

   Hint: Everytime you make a change in the code, you need upload it again on to the μC in order to see the change in action.

You should have noticed that uploading the code takes considerable amount of time, and this way of trying out different wait times is not so much fun. The next section aims to solve this issue.

1.2.3 Serial Communication on the μC

In addition to programming/flashing the μC, the serial communication link with the microprocessor μP can be used to query and debug the embedded code, to speed up code evaluation without reflashing the μC.

1. Uncomment Part A and Part B in the code.
2. This code snippet when compiled asks the μC to constantly check the serial communication link for inputs on the serial communication line. To communicate with the μC, click on Tools->Serial Monitor or press Ctrl+Shift+M.
   Now, this interface can be used to change the timing parameters online.
3. You can check how the serial communication is actually done by opening the files in the folder `/home/ubuntu/Arduino/libraries/wk7_lib`, but we do not need to modify the serial communication code for this lab.
4. Briefly explain the functionalities of Part A and Part B. **(PostLab Q3)**
5. In Part B of the code, what is the cast operation `((char*)&incoming)` doing? What is being cast to what? Why do we do need this? **(PostLab Q4)**

   Hint: Check out the appendix section about pointers.

6. How are the values that are received being interpreted? **(PostLab Q5)**

   Hint: Check out the ASCII table `http://www.ascii-code.com/`, and the last chapter of the appendix section about binary numbers.

7. How long, in bits, is the value which is expected from the serial communication? **(PostLab Q6)**
8. What is the range of values for the first and the second timing variables, `dl` and `mlt`, respectively? (Only consider ASCII characters up to the decimal value of 127) **(PostLab Q7)**

9. What should we write on the serial communication terminal in order to get `dl = 3;` and `mlt = 13;`? **(PostLab Q8)**

10. Now that you've understood how we made use of the serial communication and the permitted ranges of timing variables, `dl` and `mlt`, you should have noticed how limiting our way is. Propose an alternative way that would allow for broader ranges on these variables. **(PostLab Q9 - Bonus Point)**

1.2.4 Serial Communication on the (μP)

Now, we are going to program the second component of our application that reads user inputs. As we explained previously, this part runs on the μP. A code skeleton for establishing the serial communication is given in the folder `mP`. The `Makefile` is readily prepared for this week's task, and you should not modify it.

1. Modify `mP.c` to read two 8 bit integer from the user, and send them to the μC through the serial communication. To do that, first use the function `scanf()` to get user input. Then determine a sequence to terminate the while(1) loop. Be aware that the `scanf()` function does not accept uint8_t. Do not modify `feather_serial` **(PostLab Q10)**

2. Revise your function `print_bits()` from last week, and adapt it to 8 bits, such that it prints the binary representation of the sum of two 8 bit integers read from the user. Then send the sum to the μC through the serial communication. **(PostLab Q11)**

1.2.5 Display Binary Value on LEDs

The next task is to physically display the binary representation of the value that we receive through the serial port using the LEDs. Go back to the Arduino IDE, and modify `mC.ino` as follows.

1. Comment Part C and uncomment Part D.

2. Inside Part D, implement the code that will light up the LEDs with the binary representation of the value in variable `incoming`. **(PostLab Q12)**

3. Now we have Part C doing the LED blinking and Part D doing the binary display. Remember we put a switch in the circuit that is connected to pin 10 on the μC. Lets use this pin as an input. Depending on its value, select between Part C and Part D during the execution. Do the necessary modification on the `setup()` for this extension. Adapt the `loop()` as well to accommodate the described functionality. **(PostLab Q13)**

   Hint: Make sure that the Serial Monitor in the Arduino IDE is closed when you send your 2 values from the terminal to the Featherboard. If it is open it can interfere and prevent values from being recognized, as there are now 2 different serial monitors open. If you open the Serial Monitor to check which values have been sent to the Featherboard make sure you close it and restart your program on your laptop before you continue.

1.3 Postlab and lab report

- Record a video of your working system (**Q12** and **Q13**) and share it with your assistant on slack.
- Upload a single PDF-file with your solution to moodle. The file should contain the code you have written to solve PostLab **Q1** and **Q10 - Q13**. In particular, make sure to include the files `mC.ino` and `mP.c` and if you included your own library, also the corresponding .h and .c files of the library. Also hand in the answers to the Questions **Q2 - Q9**. Please upload your solution in time (before next lab session), late submissions will not be corrected.
- Come prepared with the Prelab procedure for the next lab.