Lab 1

Embedded Programming

Authors:

Jonas Lussi, Tarik Rifai, Naveen Shamsudhin and Prof. Bradley J. Nelson

Multiscale Robotics Lab, Institute of Robotics and Intelligent Systems

Date: 2020

Version: 1.1

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).

1.0.2 UDOO Quad

UDOO is a unique embedded computing platform offering the dual advantages of a microprocessor (μP) and a microcontroller (μC) in a compact form-factor board. A couple of UDOO's are available, such as the X86, Dual,

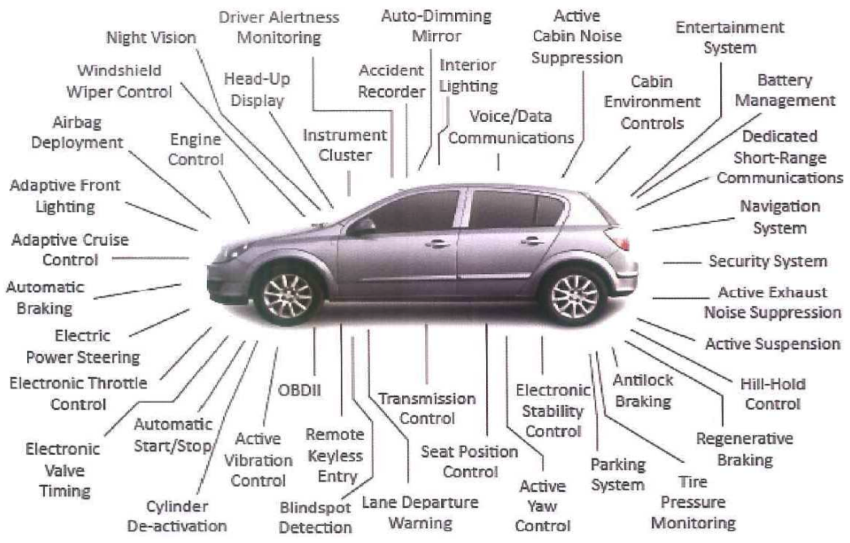


Figure 1.1: Mechatronic architecture of an automobile.

and Quad models. In this lab, we will be using the most powerful model, the Quad, which has a Freescale^{TM} i.MX6 microprocessor with four Cortex-A9 cores and an Atmel^{TM} SAM3X8E microcontroller. The quadcore μP supports a variety of Linux and Android distributions and can host realtime multi-threaded embedded applications. The μC is hardware pin-compatible with the highly popular Arduino Platform[1] enabling users to take advantage of a wide range of add-on modules (known as shields[2]) developed by the established Arduino community. In addition, the Quad model features a WIFI module, Gigabit Ethernet and SATA support (which can be used to expand the storage capacity). The UDOO Quad that we will use in the lab is installed with Linaro Ubuntu 12.04 LTS.[3]

1.0.2.1 Interfacing Sensors and Actuators

Various analog and digital sensors and actuators can be interfaced to the UDOO via custom ports or the general purpose input-output (GPIO) pins on the PCB. The custom ports include the microphone input, the audio output, USB, SATA, Ethernet, HDMI, etc as shown in the UDOO schematic (Fig 1.2). There are more than 50 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 (Fig 1.3). In the lab, we will learn how to access the GPIO pins using the μC. The exact location of the pins on the PCB can be found in the given appendix. For more details, refer to [4].

1.0.2.2 Programming the microcontroller (μC)

The Atmel SAM3x8E microcontroller can be programmed using the Arduino Integrated Development Environment (IDE) which is pre-installed with the operating system. 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 1.4).

1.0.3 Conceptual Architecture of the Board

As has been described earlier, the board consists of a:

μP which effectively acts as the CPU of the PC that we are interacting with through the peripherals. The operating system runs on this unit, and this gives the μP the ability to operate commercial peripheral units; such as

[1] http://www.arduino.cc/ The SAM3x8E microcontroller is the same as the one used in the Arduino DUE http://arduino.cc/en/Main/arduinoBoardDue

[2] Shields are boards that can be plugged on top of the Arduino PCB extending its capabilities. Different shields follow the philosophy: easy to mount and cheap to produce. For a comprehensive list of available shields, visit http://shieldlist.org

[3] http://www.linaro.org/

[4] Notes on using the GPIO pins https://elinux.org/UDOO_GPIO_Pinout

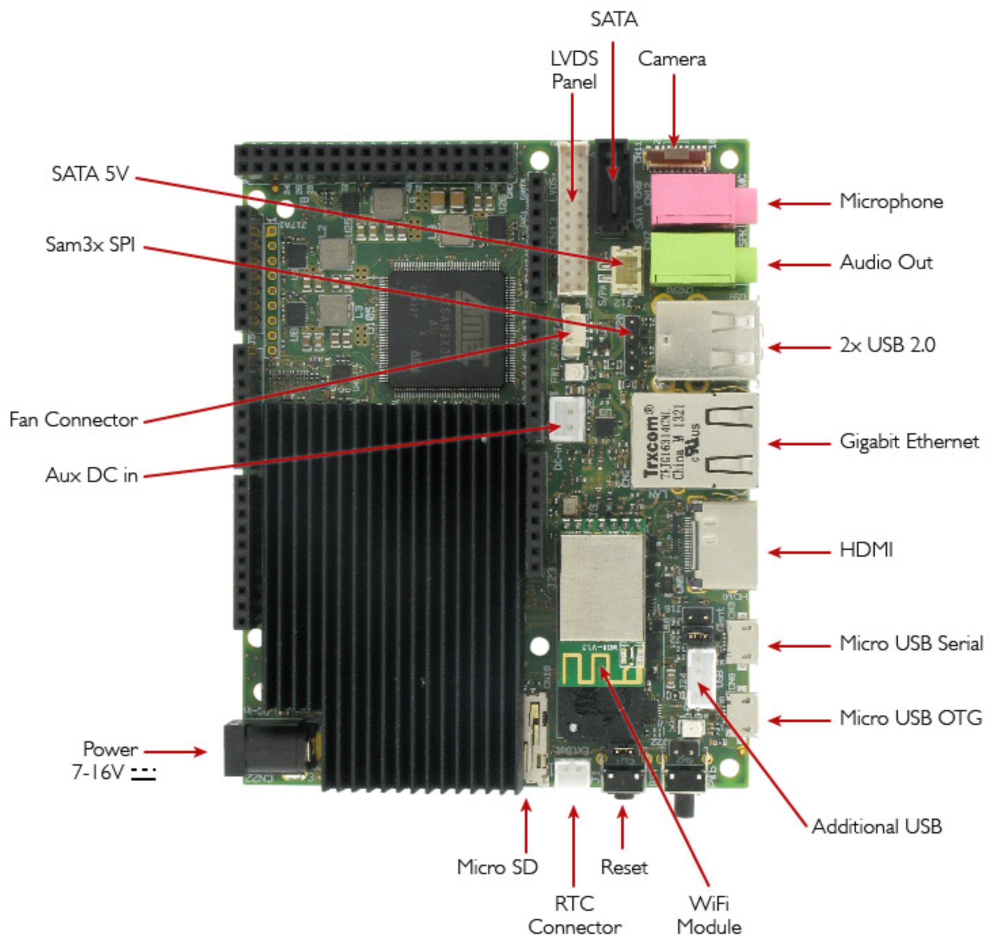


Figure 1.2

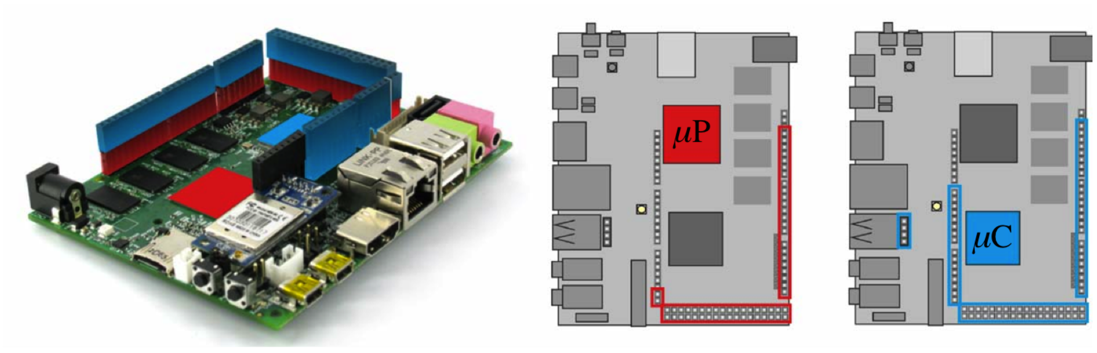
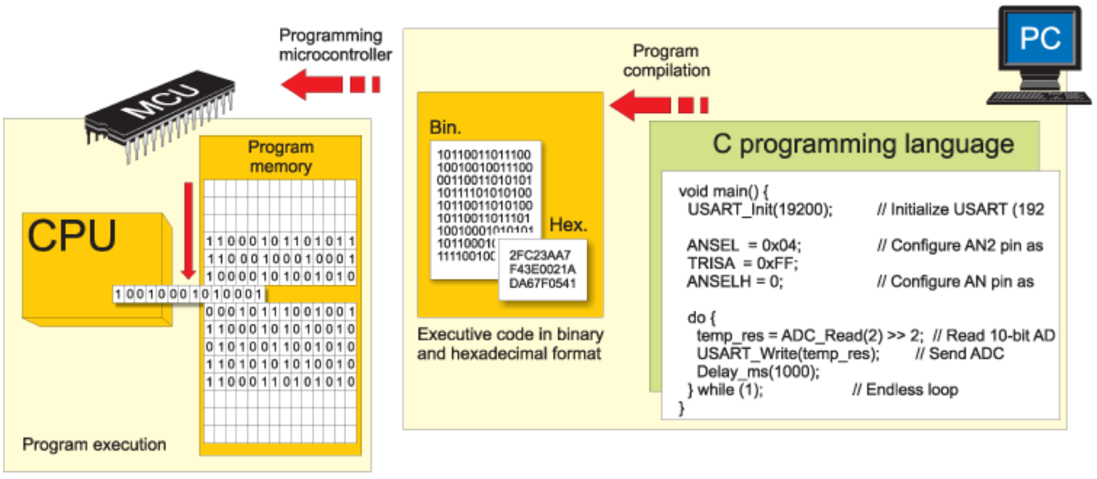


Figure 1.3: The layout of the microprocessor (μP) and microcontroller (μC) and the general purpose input-output (GPIO) pins on the UDOO PCB

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



```
void main() {
  USART_Init(19200);          // Initialize USART (192

  ANSEL  = 0x04;              // Configure AN2 pin as
  TRISA  = 0xFF;
  ANSELH = 0;                 // Configure AN pin as

  do {
    temp_res = ADC_Read(2) >> 2;  // Read 10-bit AD
    USART_Write(temp_res);        // Send ADC
    Delay_ms(1000);
  } while (1);                // Endless loop
}
```

Figure 1.4: The C code is compiled into hex or binary code which is downloaded/uploaded onto the μC FLASH memory.

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.

We can assume that these chips are two distinct entities having a communication channel in between to establish any cooperation. The architecture we rely on is the one represented in fig. 1.5. When the mouse or the keyboard is

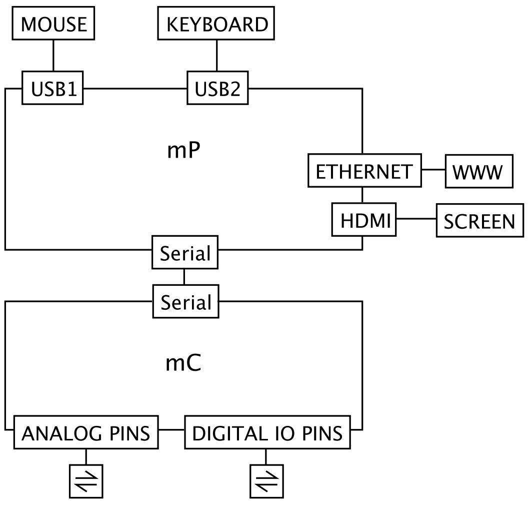


Figure 1.5: The conceptual architecture of the system.

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 a-priori.

An example to this sort of use could be the following. Assume that you have one of these UDOO boards and you want to control the light and the blinds in your room through your computer. First, you should find a way to connect both the motor on the blinds and the light to appropriate pins on the board. Then you should have an application that can understand your mouse and keyboard commands and can apply those on the hardware pins. 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.

1.1 Prelab procedure

Note: Prelab assignments must be done before reporting to the lab and must be turned in to the lab instructor at the beginning of your lab session. Additionally, you also have to upload your solution as a single PDF-file to moodle. Make sure to upload the file before your lab session starts, late submissions will not be corrected. The prelab needs to be handed in as a group. All the prelab tasks are marked with **PreLab Qx**.

1. Read the following documents:
   (a) UDOO Starting Manual: `https://www.udoo.org/docs/Getting_Started/Very_First_Start.html`
   (b) Datasheet: `http://www.seco.com/misk/UDOO_datasheet.pdf`
   (c) Communication between the two processors: `https://www.udoo.org/docs/Hardware_&_Accessories/IMX6_And_Sam3X_Communication.html`
   (d) Information on microcontroller: `https://www.udoo.org/docs/Arduino_DUE_(Sam3x)/Overview.html`
2. How many digital input/output pins does the μC on the UDOO have? **(PreLab Q1)**
3. What is the working voltage of these pins? **(PreLab Q2)**
4. Are there any limitations on the current that these pins can support? If so, what is it? **(PreLab Q3)**
5. Which pins are used for UART serial communication between the μC and the μP? **(PreLab Q4)**
6. Describe a project (in brief, 5-6 sentences) that you can imagine building with the UDOO board. What kind of inputs or measurements will it make and what kind of output is produced? (Be creative!) **(PreLab Q5)**

1.2 Lab procedure

On your lab bench, you will find the UDOO Quad with a Micro SD card inserted. Inspect the Udoo-board and see if you can identify the μC, the μP and the various I/O ports as shown in Fig 1.2 and Fig 1.3. Make sure that you can identify a GND, a 3.3V, and the digital input/output pins on the board because you will be using them in this lab.

1.2.1 Circuit Preparation

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 to the UDOO and the breadboard. For the LEDs, use pins 2 to 9; for the switch use pin 10 on the UDOO. Show your setup to one of the assistants before you connect it to the 3.3 V on the UDOO.

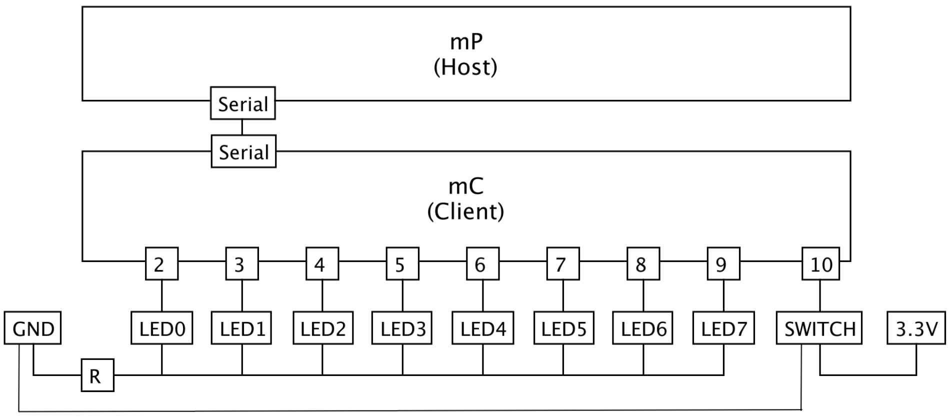


Figure 1.6: The circuit layout for this lab.

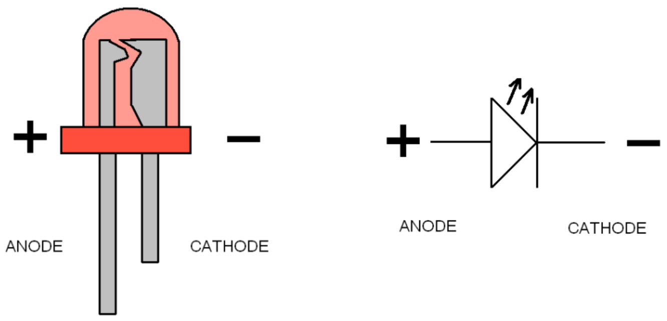


Figure 1.7: Notice the anode and cathode of the LED. For a sample LED circuit, check out the following site http://www.build-electronic-circuits.com/what-is-an-led/.

1.2.2 Digital Output

1. Extract the package Lab01.zip into 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 1.0.2.2 (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)**

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 should upload it again on to the μC.

   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.

6. Uncomment Part A and Part B in the code.

7. 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.

8. 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.

9. Briefly explain the functionalities of Part A and Part B. **(PostLab Q3)**

10. 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.

11. 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.

12. How long, in bits, is the value which is expected from the serial communication? **(PostLab Q6)**

13. What is the range of values for the first and the second timing variables, `dl` and `mlt`, respectively? **(PostLab Q7)**

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

15. 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.

16. 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 `udoo_serial` **(PostLab Q10)**

17. Revise your function `print_bits()` from the 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.

18. Comment Part C and uncomment Part D.

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

20. 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)**

1.3 Postlab and lab report

- Show your working programs to the teaching assistant.
- 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.
- Print the PDF-file and hand it in at the beginning of the next lab session.
- Come prepared with the Prelab procedure for the next lab.