ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY
KUZEY KIBRIS KAMPUSU • NORTHERN CYPRUS CAMPUS

Introduction to Microprocessors | Embedded Systems Development

EEE 347 | CNG 336

LAB MODULE #1:

INTRODUCTION TO EMBEDDED SYSTEMS DEVELOPMENT

Simple assembly programming and testing using parallel I/O interface

[15%]

Spring 22|23

Version 1.0 Ahmad Othman
Version 2.0 M Hamza Ishaq

ODTÜ METU

LAB Instructions

Labs are fun, but also essential to learn the material and to get a high grade in the course

Each laboratory exercise will get you closer step-by-step to your final semester goal of having the skills to design and verify the prototype of an IoT sub-system to support smart environment.
Overall lab score will contribute 30% to your course grade. ***You should require a minimum 70/100 from the laboratory to pass EEE-347/CNG-336 course.***

- Each of the lab Modules 1-4 will be completed in a team of two (three, when absolutely necessary to accommodate extra students).
- One report will be submitted for each team *before* the demo session.
- Report and Demonstration including the all necessarily requirements having 50% + 50% of the grading scheme for each lab module.
- If your individual demo performance (50%) reveals that you do not understand the theoretical or practical aspects of the lab, you may lose individual credit from the report (50%) portion as well.

Report (Pre-Work) Submission (50%)

You will not be permitted to perform an in-person or online demonstration of laboratory work without submitting the lab report in advance. A lab report should be submitted on the given deadline after completion of the lab exercise and before the team demonstration session. Late submission without excuse gets zero.

No "make-up" will be given in this case. This rule is strictly followed for all lab modules.

Engineering ≡ Paying attention to detail, delivering quality, and taking pride in your own work

Each group will submit one report in **.pdf file** that includes:

- **Name** of the Students+ id #, experiment, and a statement that "The content of the report represents the work completed by the submitting team only, and no material has been copied from the other source."
- **Objective** of the experiment,
- **Design details** including system hardware connectivity, code,
    - **Code** must be insert in your .pdf files, (not insert code screenshots)
- **Details of performed measurements/tests** to verify correct operation,
- **Answers** properly mentioned to the relevant questions with # in detailed as asked below.
- **Results** organized in graphical and/or tabular format,
- **Conclusions** with summary of important findings, and perspective on the benefit of the laboratory exercise.

Note: The report should be typed on A4, and delivered as one .pdf document. All fonts and pictures should be sufficiently large to be visible. If you capture images from design and simulation CAD tools, make sure the resolution is sufficiently high. Any required graphs should be plotted using Excel or similar tools. Follow a concise and direct technical report format. Late reports will not be accepted.

Lab Policy

- It is expected that each team member participates lab exercise execution, reporting and demos equally.
- Otherwise, you need to notify your lab instructors before the day of evaluation, only if you have a valid reason e.g. (exam conflict, medical etc.) accepted.
- We will follow no tolerance policy with respect to plagiarism, cheating, and any other forms of dishonesty.
- Attend your scheduled lab demo sessions on time, your lab design/simulations ready to be shared online with the lab instructor, and be ready to answer any questions.

Objectives

The purpose of this laboratory exercise is getting familiar with the development environment you will use throughout the semester in order to apply course learnings.

1.1 Introduction

In this module you will experiment with writing and executing a short assembly code segment to solve a simple problem by doing minimum computation and otherwise moving data to and from the memory, and microcontroller digital I/O ports (parallel I/O interface). You will debug and simulate your code segment before your scheduled laboratory session using **Microchip Studio** Integrated Development Environment (IDE) for AVR. Then you will use **Proteus** design suite to design and simulate your full system including the AVR microcontroller running your code, peripheral devices, and interconnects.

When you arrive to your scheduled laboratory demonstration with completed work, you should be prepared to quickly and effectively program your Board, debug any remaining software/hardware issues, and demonstrate your design. Be ready to demonstrate your code simulations, system simulations, and prototype tests to your lab instructors (will be important especially if you cannot get the hardware working), and answer any questions on your work.

1.2 PROBLEM DESCRIPTION & PRELIMINARY WORK

1.2.1 *System Functions and Interfaces*

You will design a system that receives three important parameter (temperature, moisture, water level) values using digital parallel interfaces modeled by switches (and one push-button) on the input side, and displays them using LEDs on the output side. The receiving protocol uses a 1-bit Request control (push-button) input to signal when there is a new set of data to be processed. Each request is acknowledged through a 1-bit (LED) output. The received data is also sanitized for validity, and appropriately logged to data memory space. The system is illustrated in Figure 1.1.

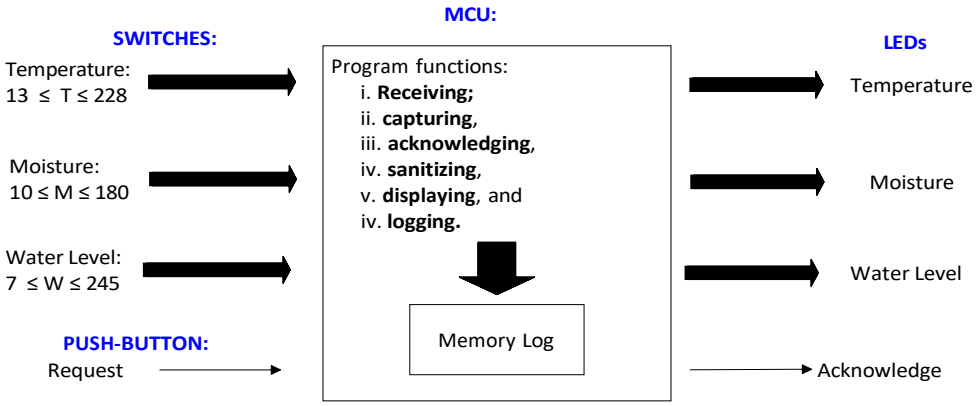

Figure 1.1. Main functions and interfaces of the data logging system

- ***Receiving***: The system will receive an unsigned 8-bit (1 byte) digital value for each of the three parameters. Since we know little about sophisticated interfaces at the moment, we will model our receiver as user-controlled switches. The input parameters should only be read if there is an assertion of active-high control signal called "Request", controlled by a push-button.
- ***Capturing***: When the "Request" is observed, each of the three parameters are read into different microcontroller registers.

- ***Acknowledging***: The system will then assert an "Acknowledge" output by turning on a single LED. This output will stay ON as long as the "Request" signal stays active. It will turn OFF when the "Request" is OFF.
- ***Sanitizing***: Each received digital parameter value will be checked against its expected range, as also documented in Figure 1.1. If the value is within the expected range for that parameter, it is preserved. If it is outside the expected range, the value is replaced by 0xFF (all 1's).
- ***Displaying***: The sanitized parameter values are then output to be displayed by different sets of LEDs.
- ***Logging***: The parameter values are then written to the next four data memory locations, starting with memory address $100. A delimiter "NUL" character (ASCII value 0) is written after each set of three entries to the memory. Once ATmega128 on-chip SRAM memory is full, the logging should continue on each data request, by overwriting the existing entries in the SRAM in round-robin fashion.

1.2.2 *Design Approach*

Proper planning and documentation are important in any design and implementation effort. Here are the steps you should follow:

a) Review ATmega128 pins and interfaces to decide how the system inputs/outputs will map to the MCU pins.

b) Decide which pins will be connected to switches, LEDs and push-buttons and how you will be connecting them. UNI-DS6 board reference and manual you have studied in the lab tutorial provides a good idea about how to connect these common parallel I/O user interface components.

c) Sketch the system layout to illustrate the signal connectivity to pins. This should provide a good idea about which ports will be used by your assembly program for different I/Os.

d) Write a pseudo-code or sketch a flowchart to indicate the main steps and decision points i.e. algorithm that you will code into your firmware. Firmware is the code you will store in the MCU program memory that will control your system operation. Figure 1.2 depicts an example piece of pseudo-code and flowchart that solves the problem of sorting three input numbers. ***You do not have to design a solution for this; it is just provided as an example.***

```
A ← 1st Number ($0000)
B ← 2nd Number ($0001)
If A < B
       A → Smallest ($0005)
       B → Largest ($0003)
Else
       B → Smallest
       A → Largest
A ← Largest
B ← 3rd Number ($0002)
If A < B
       B → Largest
       A → Middle ($0004)
       Done.
Else
       B → Middle
A ← Smallest
If A < B
       Done.
Else
       B → Smallest
       A → Middle
Done.
```

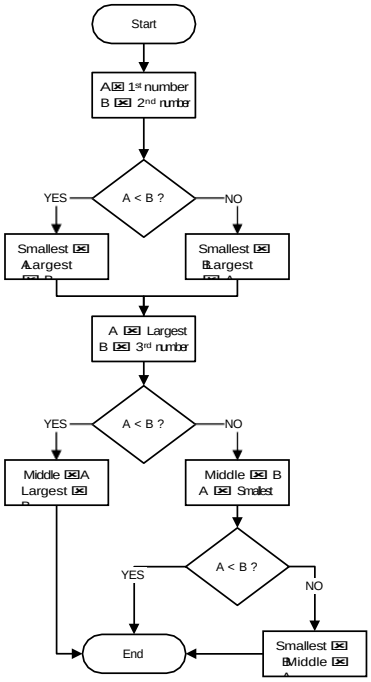

Figure 1.2. Example pseudo-code (left) and flowchart (right) to solve a sorting problem

When designing system solutions, it is a good idea to think about any special corner-cases you need to address. The more detailed and scrubbed your solution is, the fewer surprises you will have in the verification/testing phase. If any assumptions need to be made in the early design phase, you may note these down and check their validity later in verification.

1.2.3 *Implementation*

a) The following is a barebone template of an assembly program that is far from being complete. Create a text file with this source code and complete it. Add comments to explain, in at most 5 or 6 words, the function of each line. You may also add comment blocks to describe functions of different segments. Use the instruction set summary at the end of ATmega128 datasheet.

```
;
; Simple ATmega128 assembly firmware to solve the problem of receiving
; 3 unsigned 8-bit parameters using parallel ports through simple
; Request/Acknowledge protocol, sanitizing the parameters against
; predetermined expected ranges, outputting them to parallel ports,
; and logging the values to internal data memory (SRAM).
;
; Start your code with this reset vector; we will explain its meaning later:
      rjmp Start
; Code
.INCLUDE      "m128def.inc"
.EQU          ZEROS = 0x00
.EQU          ONES = 0xFF
.EQU          T_LO_LMT = 0x0D
.EQU          T_HI_LMT =0xE4

; You may want to define some other constants here for readability
; of your code…

.EQU          MEM_START = 0x100
.EQU          MEM_END = 0x10FF

.CSEG

; Start using the Program Memory at the following address
.ORG 0x0050
Start:
;
; Your code goes here
;
```

b) Use Microchip Studio to create Module1_MicrochipStudio project. Debug your code, entering input (port) pin values through the I/O window, and making sure that registers, memory locations, and output (port) pin values are correct. It is a good practice to cover different scenarios and corner cases during debug and verification. For example, ***please take a screen capture (PrtScr) of four different results and include in your report:***

i. Program starts running and data within expected range is received, but is not captured without a Request input; Acknowledge is also not asserted without the Request input,
ii. Registers showing captured data same as inputs, asserted Acknowledge in response to Request, and output data from the first Request, and corresponding logged data to memory within expected range for the first data set,
iii. Same as the previous scenario, except one of the parameter values is outside the expected range,
iv. The case when SRAM data memory is full (address $10FF has been written) and the last received set of data is logged by overwriting the first data set at the beginning of the memory (address $100).

You may for example organize the Atmel Studio windows to show relevant information in your screen captures. One organization (that may or may not work well for you) is illustrated in Figure 1.3.

Which lines in the code do not affect the machine state at all? Comment on this in your report.

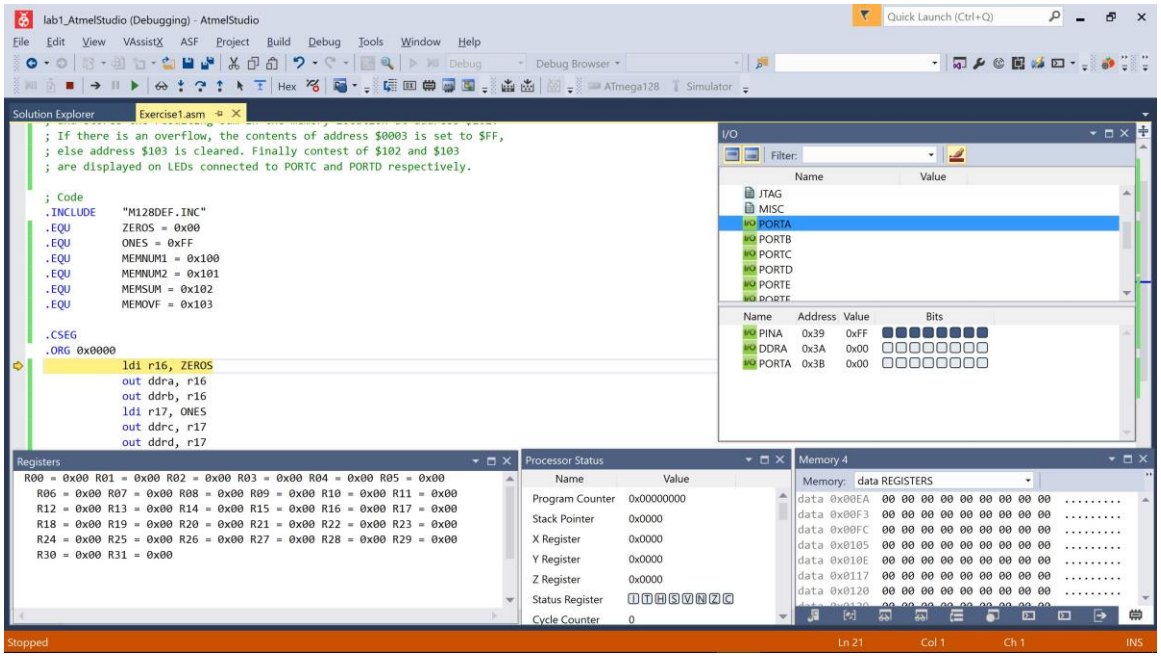

Figure 1.3. Example screen capture from Atmel/Microchip Studio

c) Use Proteus to create Module1_Proteus project. Instantiate ATMEGA128 microcontroller, and as many DIPSW_8 (8-bit dip switch), LED-BIRG (green led), 4k7 (4.7 kΩ resistor), and RX8 (pack of 8 resistors), push-button (e.g. SPST push-button in the library) components as you need in order to design the full system, and wire the ports for parallel I/O as in UNI-DS6 development board. Load your debugged code from the previous section to the microcontroller in Proteus, and run the same cases (except case iv) as in 1.2.3(b) to demonstrate that the LEDs in your system respond correctly to the inputs you provide through the switches. ***Take screen captures (PrtScr) again for the three cases to show your system design, and simulated demonstration of correct operation***. Instantiate voltage and current "probes" on the microcontroller output port wires in order to measure voltage and current levels during the simulations. What are the voltage and current levels at the output pins of the microcontroller when the signal corresponds to logic '0' and '1'? Measure the voltage drop across an LED when it is ON? What is this drop when LED is OFF?

1.2.4 What is the difference between RAM and ROM? Why is there a need for both a Flash Memory and an EEPROM?

1.2.5 What is the function of the control unit? How does the control unit get the instruction that it must execute? How does it know the number of bytes in an instruction?

1.2.6 **Submit** your answers to 1.2.2-1.2.5 as part of your lab report.

1.2.7 Be ready to demonstrate both Atmel / Microchip Studio and Proteus projects to your lab instructor during the demo session, covering different cases you may be told to simulate. You may be required to do this in the lab, especially if you have glitches or problems in your experimental demo.

- How is the microcontroller state (registers, status bits, memory, ports) affected by each line in the code?
- How does the microcontroller respond to the peripheral inputs/outputs in your design?

- How does the final system operate? Are there any assumptions? Can you think of situations that would impinge the operation of the system?

1.3 EXPERIMENTAL WORK

1.3.1 Experimental Setup

Verify to make sure your workbench has all of the following items:

- A Personal **Computer** (PC) used as a host terminal (it may be ok in some cases to use your own notebook/laptop as the host)
- **UNI-DS6** Development Board with a socketed **mikro-Board** for AVR 64-pin
- UNI-DS6 power adapter / USB Cable for UNI-DS6 mikro-Board to PC connection
- A CADET used to build external circuits (may use in some of your future labs)
- **Proteus** for simulation design

1.3.2. System Startup

Please follow the instructions below in order to start the system for your experiments:

1) Turn on the PC. On the desktop, you'll see icons for Microchip Studio, Proteus 8, and AVRFLASH.
2) Make sure that the USB cable connects the computer and the UNI-DS6 board.
3) Make sure the power adapter is connected to UNI-DS6 power plug (on the upper left side of the board). Your mikro-Board card will be powered by the board, so the "STANDALONE" jumper J1 on the mikro-Board should not be shorted. "VOLTAGE SUPPLY SELECTION" jumper J16 on the upper left should be set for 5 V operation.

1.3.3. Build and Demo (50%)

You are required to make 3 demonstrations to the lab instructor, as described below. If asked to run your experiments with new conditions, please comply with the instructions to fully demonstrate your competence of the lab material.

i) Open Microchip Studio, and enter the annotated assembly program from Preliminary Work Section to a text editor, or load it from a flash memory stick connected to the USB of the terminal. Build and run your code in debug mode (step by step) using the input conditions (vectors) provided by the TA while monitoring the microcontroller state through the windows described in the preliminary work. **When you are ready, do your first demo, showing and describing to your TA how the microcontroller state (registers, status bits, memory, ports) get affected by each line in the code.**

ii) Open Proteus, and load up your project from preliminary work. Simulate your system model using the input conditions provided by the TA.
When you are ready, do your second demo, showing and describing to your TA how the microcontroller responds to the peripheral inputs/outputs in your design. Answer any questions.

iii) Make sure the input switch jumpers on the right-hand side of UNI-DS6 board (J1-J11) are configured for pull-up configuration. Turn on the POWER SUPPLY switch on the left side of UNI-DS6 board. Open AVRFLASH. Load the .hex file generated by the build in preliminary work, and **write** to ATmega128 program memory through the programmer on AVR mikro-Board. Your AVR ports are connected to both push-buttons and DIP-switches. As long as your UNI-DS6 port configuration jumpers are set correctly, you should be able to provide your inputs to the AVR ports through either push-buttons or DIP-switches.

When you are ready, do your third demo, showing and describing to your TA how the final system operates. Remember to demonstrate the system response for different interesting scenarios.

Note: The program LED at the mikro-Board will turn on and off during the programming of the microcontroller. If you do not see any LEDs flashing after you hit the "write" button in AVRFLASH interface window, there may be a communication problem between your terminal and the AVR mikro-Board. Check with your TA to ensure the correct driver is installed for the operating system at your terminal.