ECS502U Lab: Week 10

The purpose of this lab is to practise programming using timers and interrupts. You will be using the MCU8051 IDE for this lab. On the QMPlus module page you can find 8051 assembler instruction documentation, including a brief "cheat sheet." We suggest you create a directory for all your code for ECS502U, and then a sub-directory for this week.

Once you have completed this lab sheet, combine the answers with those for week 9 into a single PDF document and submit via QMPlus.

QUESTION 1 In week 8 the exam-style questions asked to provide code snippets around timers. You may want to re-use these to complete the following tasks in the MCU8051 simulator or on the development board (where possible):

1. Provide 8051 assembly code to complement P1.0 after 128 machine cycles, using timers. Briefly describe, or provide code, how to solve this problem without timers. Then provide an implementation (with or without timers) to complement P1.0 after 1024 machine cycles.

2. Develop a program with *as few instructions as possible* that displays the number of key presses of a button (up to 255). Do so using an LED panel (of 8 LEDs) and a simple keypad/buttons (either as virtual hardware in the simulator or using the LED/switch module for the development board), and interpret the 8 LEDs as a binary string. Argue why your solution is optimal. Hint: start by describing the essential steps of this process (e.g., setting values on the LED panel), and how these translate into micro-controller instructions/actions.

3. Using the simulator's interrupt monitor (via *Virtual MCU* → *Interrupt monitor*), develop and debug a program that starts or stops timer 1 whenever the external interrupt 1 is triggered. (The interrupt can be triggered by clicking the rocket symbol in the interrupt-monitor window.)

QUESTION 2

(a) Within the 8051 micro-controller, Port 0 is used for both providing the lower half of the address bus and for the bidirectional data bus. Explain how this is possible and provide an illustration showing how you would connect an 8051 to an external 64Kbyte Random Access Memory (RAM) device including any interface logic that may be required. You should also clearly state the role of the ALE signal.

(b) Briefly explain what is meant by a bus and why it is necessary to control access to it. You are tasked with designing a means of interfacing devices via a half-duplex bus. Using an appropriate illustration explain how this can be done using buffering to avoid contention.

Practice Projects The following mini-projects are for training purposes only and do *not* need to be submitted as part of any coursework. Instead, bring your (partial) solutions with you to the lecture on Friday, where they will be discussed.

(a) *Mailboxes:* Assume you have eight external 1-byte storage devices connected (we'll call them mailboxes) to a joint address bus at port P1. The actual address bus can be at most six bits wide; P1.0 is used for "read" signalling, and P1.1 to signal "write". All mailboxes are connected to a data bus, which is connected to port P2 of the micro-controller. Write 42 to mailbox 5.

(b) *Button indicator:* Assume a button connected to P3.3, and an LED at P1.0. Toggle the LED after every seventh key press.

Optional design and implementation challenges. All of the following are ideas and suggestions of problems that are best solved using a micro-controller. Some of them may require considerable programming, whereas others are basic building blocks of just a few instructions. You may pick and choose to work on subquestions only. *None of them are required to obtain full marks on this lab sheet, but each of them may earn you bonus marks, possibly also to recover lost marks on previous/future lab sheets. It is also possible to submit answers to these after the deadline, via email.*

Any submission must be accompanied by a short description of what your implementation exactly achieves as the requirements stated below are very open. As far as time permits, we will discuss possible solutions of selected questions in the lectures.

OPTIONAL QUESTION A: Stop Watch

1. *Design* the software architecture of a stop watch. That is, you are not required to actually implement it, but instead collect the requirements to provide basic functionality, sketch the behaviour as a state machine, and identify the key building blocks.

2. *Implement* a stop watch. You may link this to your previous work of displaying numbers on a 7-segment display, or limit yourself to having an appropriate value present in a register. Furthermore your work towards Question 1 of this lab sheet will be useful.

OPTIONAL QUESTION B: Hangman

1. *Dimming LEDs.* Earlier in the lectures, the limits of human perception and the use thereof to adjust brightness of LEDs were briefly discussed. Implement brightness control using two buttons. (It will be easier to run this on the actual development board than fiddling with the virtual hardware.)

2. *Knight-Rider-style LED bar.* Search "Knight Rider intro" on youtube if you need more inspiration (e.g., `https://youtu.be/oNyXYPhnUIs`). In less fun terms, you might as well call this a "progress bar". Technically, you need to light up one or more of a bar of LEDs at different intervals, and possibly dim them.

3. *Generate/store random numbers.* In various contexts one needs a random number as part of an application. Various algorithms exist to create pseudo-random numbers (starting from an initial *seed*). In interactive systems, however, usually ample *entropy* is available so that true randomness is possible. For example, the time between key presses, or simply the number of key presses, may be used. Build a (simple) program that stores a random number in a register. For instance, you may require that the user presses a button a number of times, but refrain from debouncing the button, such that any key press can be logged one or more times.

4. *Playing octal hangman.* First, generate a random string of 24 bits (3 registers). Interpret this as a sequence of 8 octal numbers (8 three-bit values). Second, use four buttons to a) enter an octal number (using three buttons) and b) confirm the entry (fourth button). Increasingly light up more and more LEDs whenever the user's input is *not* contained in the 8 three-bit values. If the user does manage to guess all octal digits, flash all the LEDs, possibly using your Knight-Rider-style LED bar from above.

OPTIONAL QUESTION C: Infusion Pump

1. *Fixed-point arithmetic.* IEEE 754 is the industry standard to implement floating-point arithmetic. It is, however, non-trivial to implement, especially in resource-constrained environments (such as an 8-bit micro-controller). Consequently fixed-point arithmetic is often used. For this example, implement routines to perform arithmetic operations on 16-bit fixed-point numbers. You may freely choose the number of magnitude (integer) bits and the number of fractional bits.

2. *Display fixed-point numbers on a 7-segment display.* For a given fixed-point number, display it on a 4×7-segment display, ensuring that the decimal dot is placed appropriately. Observe that the required bit pattern is the bit-wise or of the digit without a decimal dot, and the single bit to light up the decimal dot.

3. *User interface of infusion pumps.* Paolo Masci and Paul Curzon have developed the PVSio tool, available online at `http://www.pvsioweb.org`. Open a project such as the "BBraunPerfusor", change to the "Simulator View", and see how current infusion pumps are controlled. You may also watch the youtube video linked from their page. Using the keypad and 7-segment displays you can mimic this behaviour by re-purposing some of the buttons as cursors. What is your behaviour upon pressing "up" when a digit is at its maximum (i.e., "9") – do you increment the next digit or just wrap around? Code to display fixed-point numbers is a valuable building block.

OPTIONAL QUESTION D: Automotive Applications

1. *Indicator and hazard lights.* Use three buttons and two (or more) LEDs with the following semantics: one button is to indicate left (left LED flashing), another to indicate right (right LED flashing), and the third to turn on hazard lights, i.e., both left- and right-hand side LEDs are flashing simultaneously. Consider modelling the intended behaviour as a state machine (on paper!) first.

2. *Window regulator.* Use three buttons and the LED panel to simulate up- and down control for an electric window, such that the LED panel reports the position of the window. A third button shall serve as emergency-stop sensor, i.e., pressing that button while holding the "up" button immediately stops the window motion.

3. *Windscreen wiper control.* Recent windscreen wipers feature multiple modes and sensors. For some ideas, watch `https://youtu.be/fCitn3TJmXo` on youtube. Use buttons to simulate sensor input as well as buttons available to the user, and use LEDs to simulate the left-right movement of the wipers.

OPTIONAL QUESTION E: Analogue Interfacing

1. *Controlling a stepper motor.* Design a circuit that connects the micro-controller with a stepper motor, as well as a keypad. The buttons shall then be able to control the direction of the motor and change the speed at which it rotates.

2. *Temperature-controlled fan.* In addition to the actuator (motor/fan), add an analogue sensor the value of which controls the speed of the motor. A practical application of such a design is a temperature-controlled fan.