EMBEDDED SOFTWARE EXERCISE

Introduction

This workbook contains practice exercises for the training course "Embedded Software". Common requirements for the exercises are described as below. However, in each particular chapter or exercise, additional requirements may be needed and detailed.

Requirements for tools

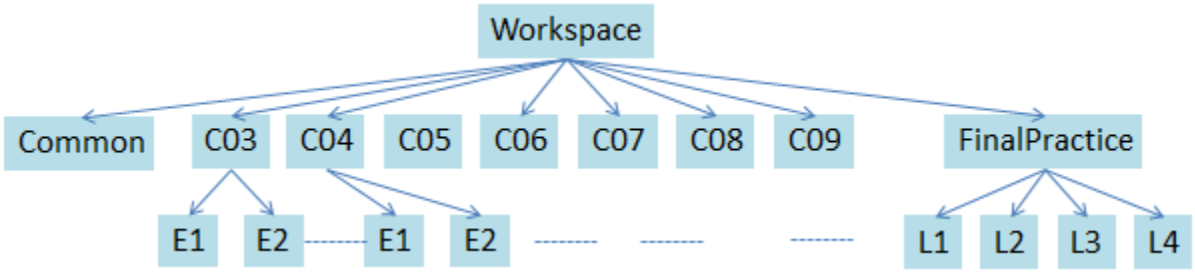
Hardware: RL78/G14 Renesas Demonstration Kit (RDK)

Software: CubeSuite+ 2.02 with CAK078R compiler

Library (see next section)

Workspace

The common workspace is organized as bellow:



Top directory (example): D:\Training\23G\Workspace

Provided library: Common

Caution: Trainees are recommended not to modify the source codes in Common. If trainees are willing to investigate this library, it is advisable to make a copy of it. If for some reason, the library was modified and trainees could not perform exercises as expected, they should contact a supporter to fix the problem.

From C03 to C09: Directories of exercises of each chapter. The number of the exercises may vary in different chapter. In each directory, there are sub-directories for each exercise.

For example, for the exercise 2 of the chapter 4, trainees must work inside "Workspace >> C04 >> E2".

FinalPractice: This directory is reserved for the final exercise of this training course. It consists of 4 levels and each of them will be done respectively in the directories L1, L2, L3, L4.

For FinalPractice, detailed descriptions will be announced only after the theory training is over.

Chapter 3. Assembly Language

There are four exercises for this chapter and they will focus on assembly language and controlling LEDs on the RL78/G14 board. It is advised that trainees should do the exercises in the order that they appear.

Exercise 1: Running sample source code

Purpose: To get familiarized with the board, assembly language and CubeSuite+ environment.

Input: See the description below**.**

Output: Instructor or supporter will check the result at trainees' desks.

Workspace: Workspace\C03\E1.

Description: Running your first assembly code by following the steps below.

- Open CubeSuite+ IDE
- Open project Exercise_3a (File >> Open… then browse to open **Exercise_3a.mtpj**)
- Read source code in **ex3a.asm** (which turns ON LED 3)
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to RL78/G14 board and run (Debug >> Download)

Your task:

- Investigate source code in **ex3a.asm**

Exercise 2: Blinking LED 3

Purpose: To write assembly codes and wait function.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C03\E2.

Description: Making a LED to blink by following the steps below.

- Open CubeSuite+ IDE
- Open project Exercise_3b (File >> Open… then browse to open **Exercise_3b.mtpj**)
- Add necessary codes in **ex3b.asm** (which will make LED 3 blinking)
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to RL78/G14 board and run (Debug >> Download)

Your task:

- Write assembly codes to configure LED 3
- Write assembly codes to switch LED 3 ON/OFF
- Write assembly codes to make delay time between ON/OFF states of LED 3

Exercise 3: Blinking multiple LEDs (1)

Purpose: To work more deeply with assembly language and to further exploit the exercise 2.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C03\E3.

Description: Making multiple LEDs (#3, 5, 7,…, 13) to be ON and OFF in reverse order (#13, 11, 9,…, 3) by following the steps below.

- Open CubeSuite+ IDE
- Open project Exercise_3c (File >> Open… then browse to open **Exercise_3c.mtpj**)
- Add necessary codes in **ex3c.asm**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to RL78/G14 board and run (Debug >> Download)

Your task:

- Write assembly codes to configure LEDs #3, 5, 7,…, 13
- Write assembly codes to make delay time between ON/OFF states of the LEDs
- Write assembly codes to switch ON the above LEDs in sequential order (#3 → #13) and OFF the LEDs in reverse order (#13 → #3)

Exercise 4: Blinking multiple LEDs (2)

Purpose: To work with assembly subroutine and to further exploit the exercise 3.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C03\E4.

Description: Making multiple LEDs (all LEDs from #3 → #14) to be ON and OFF in reverse order (#14 → #3) by following the steps below.

- Open CubeSuite+ IDE
- Open project Exercise_3d (File >> Open… then browse to open **Exercise_3d.mtpj**)
- Add necessary codes in **ex3d.asm** and **led.asm**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to RL78/G14 board and run (Debug >> Download)

Your task:

- In **led.asm**, write assembly subroutines to configure and switch LEDs. It is required that **LED be switched by passing its index to subroutine**
- In **ex3d.asm**, write assembly codes to make delay time between ON/OFF states of the LEDs
- In **ex3d.asm**, write assembly codes to switch ON the above LEDs in sequential order (#3 → #14) and OFF the LEDs in reverse order (#14 → #3)

Chapter 4. Embedded C Language

There are two exercises for this chapter. The first exercise will focus on the understanding of how a C function is generated to assembly code. The second exercise will focus on controlling the LEDs on the RL78/G14 board using embedded C language.

Exercise 1: How C function is generated to assembly code

Purpose: Understand step-by-step how a C function is generated to assembly code after compilation.

Input: See the description below**.**

Output: Write your answer in a text file and send to instructor in daily report.

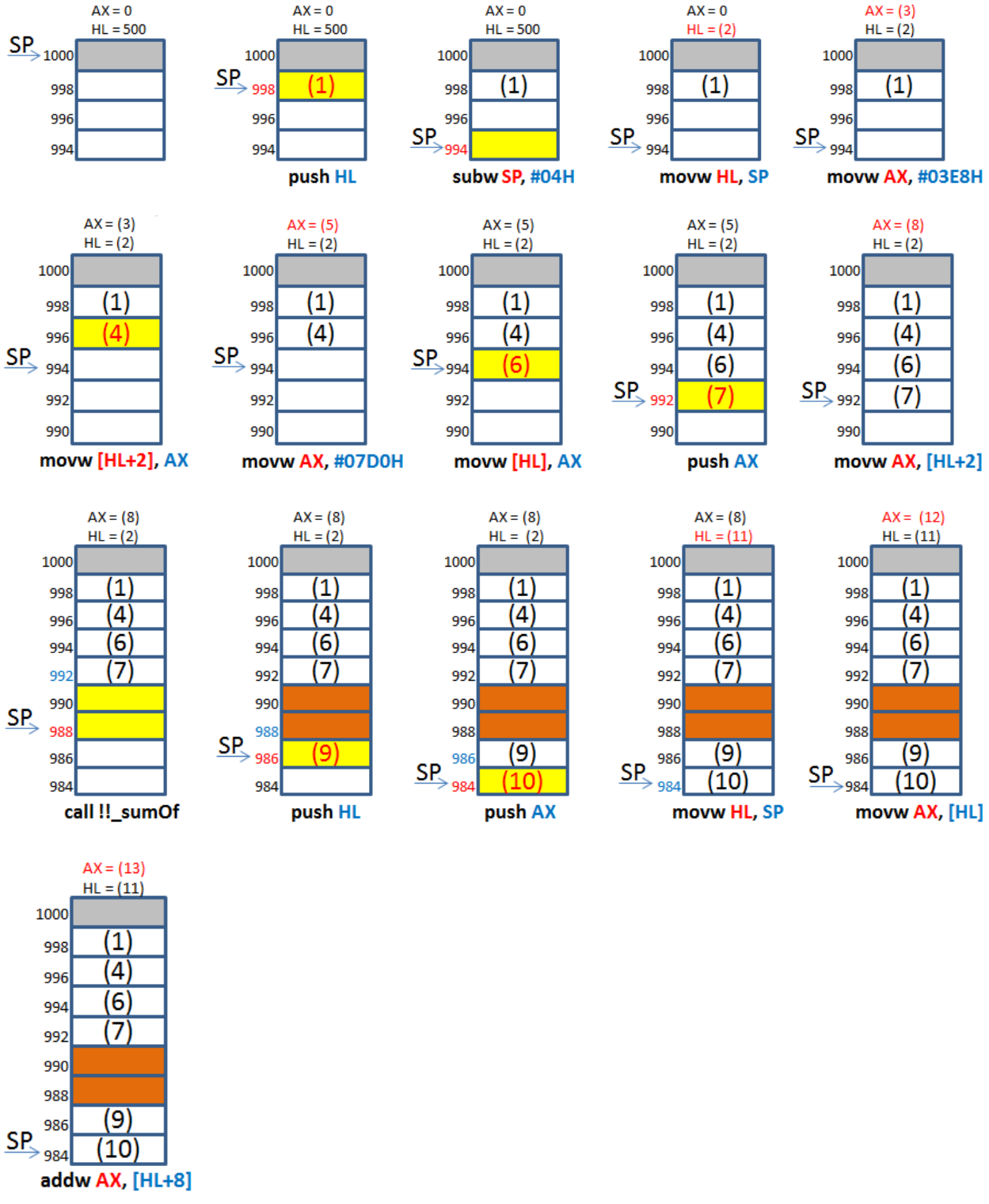
Workspace: Workspace\C04\E1.

Description: In chapter 4, we talked about a function sumOf(int a, int b). After compilation, in assembly code, we see an instruction "addw ax, [hl+8]". This exercise aims to examine the assembly instructions and understand how function arguments are passed to stacked, registers…

- Open CubeSuite+ IDE
- Open project Exercise1 (File >> Open… then browse to open **Exercise1.mtpj**)
- Open main.c and investigate it
- Build project (Build >> Build Project, or Build >> Rebuild Project)

Your task:

- After compilation, open **main.asm** inside the folder DefaultBuild (you can use a text editor to open it)
- Assuming that initially the stack pointer (SP) is pointing to the address 1000; and the value of the base register HL is HL = 500, and AX = 0
- Observe the below process and determine the value of (1) → (10) (Note: The first instruction "push HL" is at line #86 in **main.asm.**)
- Attach your answer in daily report



Operations from left to right, top to down.

Exercise 2: Controlling LEDs using embedded C program

Purpose: Understand how to use special function registers and direct address access to control the LEDs and switches on the RL78/G14 board. This exercise also gives a chance to you to write a wait() function in C.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C04\E2.

Description: This exercise involves a number of steps. You should go through it from step 1 until the end. Except the library already provided, you have to develop all functions by yourself.

- Open CubeSuite+ IDE
- Open project Exercise2 (File >> Open… then browse to open **Exercise2.mtpj**)
- Complete source codes **ports.h**, **api.h**, **api.c** and **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to RL78/G14 board and run (Debug >> Download)

Your task:

- Define all the ports with their corresponding addresses by completing the header file **ports.h** (Note: There are 12 LEDs and 3 switches.)
- Declare 3 functions in **api.h** and define them in **api.c**:
    - Function 1: Switch OFF all the LEDs
    - Function 2: Switch ON all the LEDs
    - Function 3: Make a delay (wait) of M mili-seconds
- Control the behavior of all the LEDs as follows:
    - After system reset: All the LEDs will be OFF, then wait for 1 second
    - After that, all the LEDs will be blinking with a duration of 1 second between ON/OFF states
- Alter the duration between ON/OFF states
    - If the switch 1 is pressed, the duration will be reduced to 500 mili-seconds
    - If the switch 2 is pressed, the duration will be increased to 2 seconds
    - If the switch 3 is pressed, the duration will be reset to 1 second

Important notes:

(1): If you are unable to make the LED #12 on, remember to refer to the training notes of this chapter to find the reason why.

(2) You will need to write a ***wait(int M)*** function that will "sleep" the program for M m-seconds. You will be learning how to do this accurately in chapter 6 on Timer. In this exercise, you can use ***while*** loop with **N** iterations. The value of **N** will typically depend on the clock frequency of the CPU and the number of execution cycles that is taken by one iteration of the loop.

Hint:

- In our settings, the main clock frequency is 32 MHz
- One iteration in the ***while*** loop below takes about 18 cycles (why? Try to find out more by yourself)

```
unsigned long i = M*1778;
while (i!=0) i--;
```

(3) In the last step, you should know how to enable key input, otherwise, when you press on the switches, nothing will happen. The register to control it is "key return mode register". You should refer to the hardware manual to get more information about its usage.

Chapter 5. Linking C and Assembly

There are three exercises for this chapter and they focus on calling assembly subroutine from C function and vice versa with different techniques (passing argument via register and/or stack).

Exercise 1: Linking C and assembly language (1)

Purpose:

- To practice calling an ASM subroutine from C function
- To practice passing argument via **register ONLY**
- To practice controlling return value

Input: See the description below.

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C05\E1.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **Subroutine.asm** and **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to RL78/G14 board and run (Debug >> Download)

Your task:

- Complete source code for **Subroutine.asm** and C function **main.c**
- Add comment for each line of code written by trainee
- C function **main(void)** will receive the return value of the assembly **Perimeter** subroutine
- The assembly subroutine is defined as below:

Feature	Calculate the perimeter of a circle
Prototype	unsigned long int Perimeter(unsigned int r);
Argument	unsigned int r
Return value	unsigned long int
Explanation	For now, no need to care about overload. Redefine pi = 3 instead of pi=3.14
Caution	None

Exercise 2: Linking C and Assembly language (2)

Purpose:

- To practice calling an ASM subroutine from C function
- To practice calling a C function from ASM
- To practice passing argument **via register**
- To practice controlling return value

Input: See the description below.

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C05\E2.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **Subroutine.asm** and **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Complete source code for **Subroutine.asm** and C functions **main.c** and **Sum()**
- Add comment for each line of code written by trainee
- C function **main(void)** will receive the return value of the C **Sum()** function
- The Sum() is defined in C as below:

Feature	Calculate the sum as below: S = SubSum(1) + SubSum(2) + … + SubSum(n) By using SubSum function which is defined in ASM language
Prototype	unsigned long int Sum(unsigned int n);
Argument	unsigned int n
Return value	unsigned long int
Explanation	For now, no need to care about overload. Assuming that 1 <= n <= 50
Caution	None

- ASM SubSum function is defined as below:

Feature	SubSum(n) = Square(n) + n Using Square(n) function which is defined in C language
Prototype	unsigned long int SubSum(int n);

Argument	int n
Return value	unsigned long int
Explanation	For now, no need to care about overload
Caution	None

- The C Square() function is defined as below:

Feature	Square (n) = n*n
Prototype	unsigned int Square (unsigned int n);
Argument	unsigned int n
Return value	unsigned int
Explanation	For now, no need to care about overload
Caution	None

Exercise 3: Linking C and Assembly language (3)

Purpose:

- To practice calling an ASM subroutine from C function
- To practice passing argument via **register and stack**
- To practice controlling return value

Input: See the description below.

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C05\E3.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **Subroutine.asm** and **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Complete source code for **Subroutine.asm** and C functions **main.c**
- Add comment for each line of code written by trainee
- C function **main(void)** will receive the return value of the assembly **Triangle** function subroutine
- The Triangle function is defined in assembly as below:

Feature	Calculate the perimeter of a triangle
Prototype	unsigned int Triangle(unsigned int a, unsigned int b, unsigned int c);
Argument	unsigned int a; unsigned int b; unsigned int c
Return value	unsigned int
Explanation	For now, no need to care about overload Assuming that a, b, c satisfy the conditions for the edges of a triangle
Caution	None

Chapter 6. Time Management

There are three exercises for this chapter focusing on different kinds of timer (software and hardware). The general requirement of the exercises is to activate an imaginary bomb after 10 seconds elapsed. Remember to refer carefully to hardware manual when doing these exercises.

Exercise 1: Using software wait

Purpose: Understand how to make software wait using assembly language.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C06\E1.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **SWWait.asm** and **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Complete the missing lines in:
  - **SWWait.asm** (1 instruction)
  - **main.c** (4 instructions)
- Investigate source code **SWWait.asm** and **main(void)** function
- Run the program on the board and observe the result

Exercise 2: Using 12-bit interval timer

Purpose: Understand how to configure hardware timer (12 bit interval timer) and use it to make wait function.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C06\E2.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **api.c** and **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Complete the missing lines in:
  - **api.c** (8 instructions)
  - **main.c** (2 instructions)
- Investigate source code **api.c** and **main(void)** function
- Run the program on the board and observe the result

Exercise 3: Using Timer Array Unit

Purpose: Understand how to configure hardware timer (timer array unit) and use it to make wait function.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C06\E3.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **api.c** and **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Complete the missing lines in:
  - **api.c** (6 instructions)
  - **main.c** (4 instructions)
- Investigate source code **api.c** and **main(void)** function
- Run the program on the board and observe the result

Additional tasks for all the exercises:

- Using LED to represent the count value
- Using LED to inform that the bomb has exploded

Chapter 7. I/O Control

There are three exercises for this chapter focusing on displaying information on LCD. It is advised that you do exercises in the order that they appear.

Exercise 1: Display text on LCD

Purpose: Understand how to use the Glyph API to handle the LCD module.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C07\E1.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **E07A.mtpj**)
- Complete source code **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Complete the blank lines ("…") in **main.c**
- Investigate source code of **r_main_userinit(void)** function
- Run the program on the board and observe the result

Exercise 2: Create a 100-second clock

Purpose: Understand more deeply how to use the Glyph API and timer (presented in chapter 6). Understand how to use LCD and timer at once to create sophisticated program.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

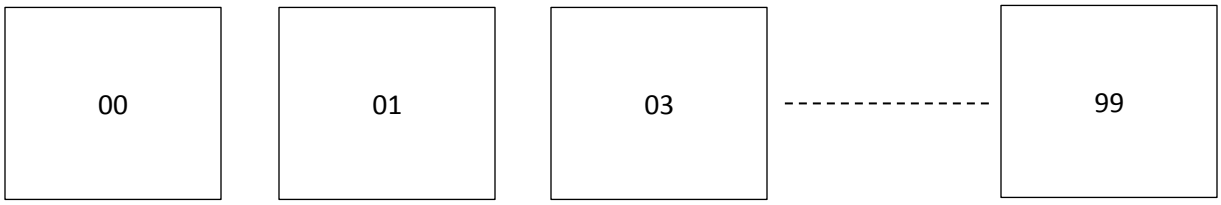
Workspace: Workspace\C07\E2.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **E07B.mtpj**)
- Complete source code **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Complete source code **main.c** to display the time (represented in second) at the center of the LCD (see the illustrations below). Requirements:
  - Use timer A0 with 50ms of cycle
  - 1 second elapsed is measured by polling the interrupt request bit of the timer
  - The initial value of the clock is set to 00. This value is increased after each second. It is displayed on the LCD repeatedly from 00 → 01 → 02 → … → 99 → 00.
- Run the program on the board and observe the result



Exercise 3: Display effect

Purpose: Understand more deeply how to use the Glyph API and timer (presented in chapter 6). Understand how to use LCD and timer at once to create sophisticated program.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C07\E3.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **E07C.mtpj**)
- Complete source code **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Complete source code **main.c** to animate "Your name" from left to right, top to down on the LCD module (see the illustrations below – Note: Replace "Your name" by your actual name).
- Case A – Requirements:
    - Upon system reset, "Your name" is positioned at the top-left corner of the LCD
    - Then, "Your name" is moving from left to right
    - When the last character of "Your name" hits the right edge of the LCD, "Your name" will be moved to next row of the LCD and restarts to move from left to right…
    - Use timer to set the interval between each two consecutive motions (e.g. 100ms)
- Case B – Requirements:
    - Similarly to Case A, except that when "Your name" hits the right edge of the LCD, it will be split into 2 rows. The characters that hit the edge are now moved to the next row

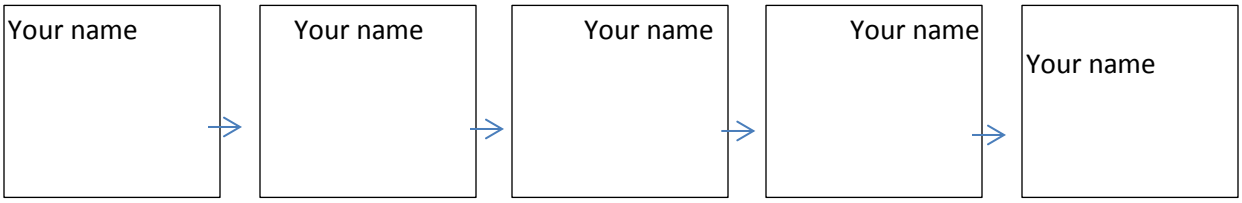


Illustration of Case A

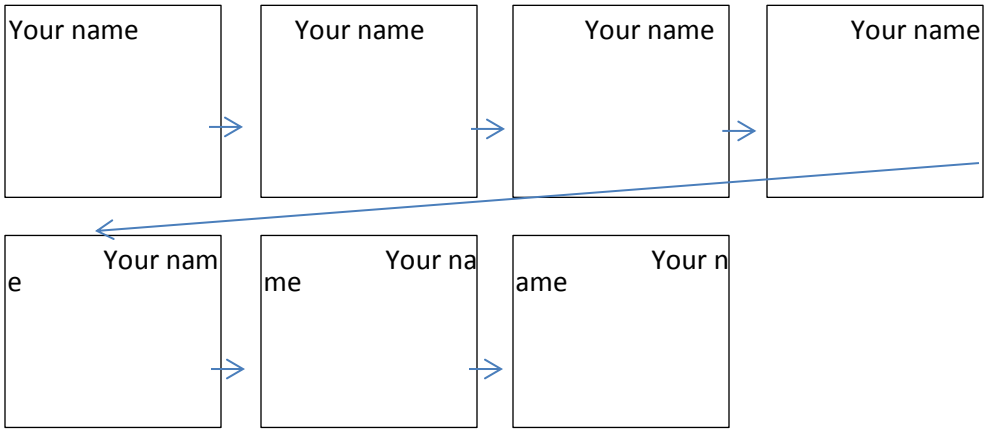


Illustration of Case B

Chapter 8. Interrupt

[T.B.D]

Chapter 9. MCU Peripherals

There are 5 exercises on Watchdog Timer, Analog-Digital converter (ADC) and Universal Asynchronous Serial Receiver Transmitter (UART).

Exercise 1: Working with WDG

Purpose: Understand how to configure and use WDG.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C09\E1.

Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)
- Create program to archive requirements.

Your task:

- Configure Watchdog timer operation as following:
  + Interval interrupt is generated when 75% of the overflow time is reached.
  + Watchdog timer window open period: 75%
  + Counter operation enabled (counting started after reset)
- Write a program to show the operation of Watchdog timer in 3 cases below:
  + Watchdog timer counter overflows.
  + Data other than activation value is written to the WDTE register.
  + Data is written to the WDTE register during a window close period.

Exercise 2: Working with ADC

Purpose: Understand how to configure and use ADC.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C09\E2.

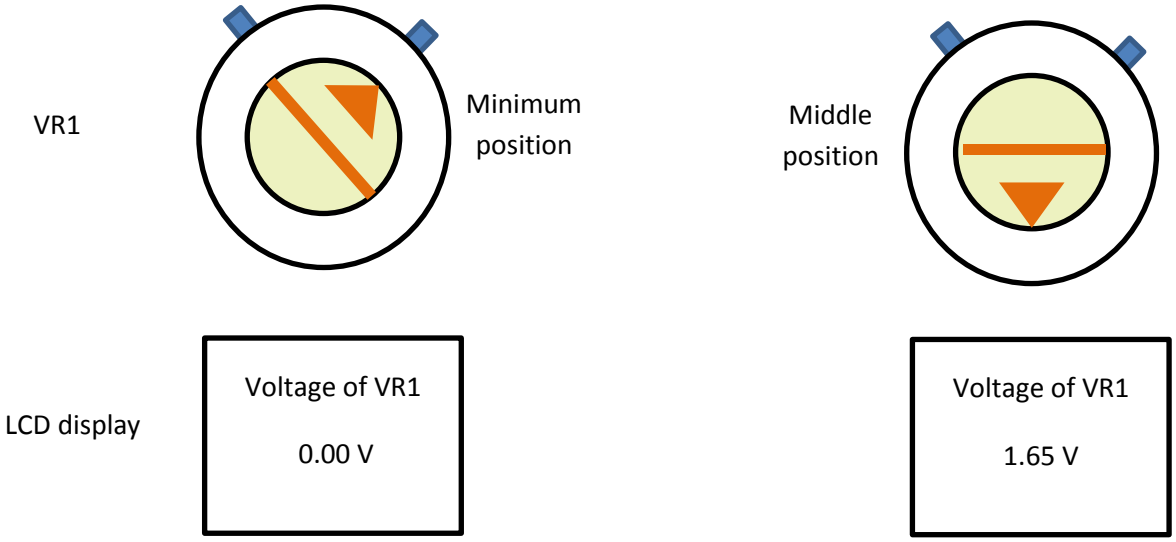
Description: Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)
- Modify sample program to archive requirements.

Your task:

- Write a program to display VR1 voltage (0 ~ 3.3V) on LCD module (see the illustration below). VR1 is connected to ANI8 pin. The voltage 0V to 3.3V is output to ANI8 when turning VR1. The reference voltage is V_{REF}=3.3V. The A-D converter will provide the results according to the resolution within 0V~3.3V. Requirements:
  - **A-D converter:**
    - Resolution: 10 bits resolution
    - A/D conversion channel selection mode: Select mode
    - A/D conversion trigger mode: Software trigger mode
    - A/D conversion mode: Sequential conversion mode
    - Conversion clock (f_{AD}): $f_{CLK}/4$
  - **LCD display:**
    - The 1st line of the LCD will show a message about measurement object.
    - Result will be shown with unit and 2 digits after decimal point.

- **Time management:**
  - The A-D converter completes sampling and gives the result every 100ms
  - Time management is based on 12-bit interval timer – 5ms cycle



Displaying measured voltage on LCD

Exercise 3: Sample program for UART

Purpose: To get familiarized with working environment for UART.

Input: See the description below**.**

Output: Instructor or supporter will check the result at trainees' desks.

Workspace: Workspace\C09\E3.

Description: You will try to run a sample program to understand the method to control UART. The program will transmit messages (characters) to computer through COM port and receive the same messages through UART to display them on the LCD module. Follow the steps below to complete this exercise.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **main.c**
- **Configure TeraTerm (see TermTerm configuration below)**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Investigate source code
- Run the program and observe the result
- Try to modify sample program and run the program in both configurations of TeraTerm

TeraTerm configuration – Case 1 (for sample program)

- Port: Check the Port name from "Device Manager" of your computer
- Baud rate: 9600 bps
- Data: 8 bits
- Parity: None
- Stop: 1 bit
- Flow control: None

TeraTerm configuration – Case 2 (optional)

- Change UART baud-rate (e.g. 19200, 38400, 57600 bps)
- Change UART data length (e.g. 7 bits, 8 bits)
- Change UART parity checking (e.g. odd, even)
- Change UART Stop bit (e.g. 1bit, 2bits)

Exercise 4: Checking UART error

Purpose: To play with different kinds of error in UART.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C09\E4.

Description: You will modify the sample program in the exercise 3 to add an error processing. When an error occurs, the program is able to show an error message on the LCD module.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Make the code to display an error message on the LCD when one of the following UART receiving errors occurs:
  - Overrun error
  - Flaming error
  - Parity error
- Stop the program when an error other than the above occurs
- Show the error message for one second on the LCD, then the program will recover automatically.

Exercise 5: Using UART communication

Purpose: To use UART communication.

Input: See the description below**.**

Output: Compress the final project and send it to instructor in daily report.

Workspace: Workspace\C09\E5.

Description: You are going to write a program to do simple math operation. Input operands and operator are received from computer through UART. The operands are from 0 to 9 and the operator is plus/subtract/multiple/divide. After calculation, target board will send result to PC through UART and display full operation to the LCD.

- Open CubeSuite+ IDE
- Open project Common (File >> Open… then browse to open **Common.mtpj**)
- Complete source codes **main.c**
- Build project (Build >> Build Project, or Build >> Rebuild Project)
- Download executable to the RL78/G14 board and run (Debug >> Download)

Your task:

- Complete the program as requirements:
  - **UART configuration:**
    - Baud rate: 19200 bps
    - Data length: 8 bits
    - Parity: None
    - Stop bit: 1 bit
  - **LCD display:**
    - Display frequency: 50Hz.
  - **Time management:**
    - Time management is based on 12-bit interval timer – 20ms/cycle
  - **Program functionality:**
    - Operand is integer comprised between 0 and 9.
    - Operator is addition (+), subtraction (-), multiplication (x) or division (÷)
    - Result can be positive or negative.
    - If operator is division, the program will ignore remainder.
  - **Program functionality (optional):**
    - Check the input data of the calculation: Whether or not operator, operands are valid. If not, notify user.
    - In case of division, provide remainder.
    - Support packaged data in transmitting – receiving.