Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics

Department of Automation and Applied Informatics
Embedded and control systems specialization

Embedded and Ambient Systems Laboratory

Laboratory guide

Microcontroller programming in assembly level

Written by: Zoltán Szabó (zoltan.szabo@aut.bme.hu)
Gergely Kardos (gergely.kardos@aut.bme.hu)
Last modified: March 10, 2014

Introduction

The main goal of the laboratory is to get familiar with the C8051 development kit and its programming in assembly language, using the Keil µVision development environment.

Description of the laboratory

Through the measurement tasks, the students get an overview and get familiar with the developer kit (C8051F040-DK) and the extension board with programming them using assembly language.

The Extension Board

An extension board has been created for the C8051F040-DK developer kit that contains numerous input and display devices to extend the functionality of the developer kit. The following peripherals can be found on the extension board (if the signal name starts with # - means the signal is low active):

Function	Enable / activate		Control signals	
	Signal	Port	Signal	Port
8 push buttons	#BTNEN	P4.4	BTN0..BTN7	P1.0..P1.7
8 switches	#SWEN	P4.5	SW0..SW7	P2.0..P2.7
8 LEDs	#LDEN	P4.3	#LD0..#LD7	P5.0..P5.7
7 segment displays (4 pieces)	#7SEN	P4.2	7S_DP	P6.7
digit selector bit 0..1	7SSEL0..7SSEL1	P4.0..P4.1	7S_A..7S_G	P6.0..P6.6
2x16 character based LCD display	#LCD_EN	P3.3	#LCD_BL LCD_RS LCD_R/#W LCD_DB0..7	P3.2 P3.0 P3.1 P7.0..P7.7
Analog potentiometer	-	-	AIN0.0	AIN0.0
Analog thermometer	-	-	AIN0.1	AIN0.1
Infrared transceiver	-	-	AIN0.3	AIN0.3
I^2C thermometer	-	-	SDA SCL I2CTEMP_CMP	P0.6 P0.7 P3.7
SPI thermometer	#CS_SPITEMP	P4.6	MISO SCK	P0.3 P0.2
SPI potentiometer	#CS_SPIPOT	P4.7	MOSI SCK	P0.4 P0.2
3 digital inputs (on extension header)	-	-	#EXT_IN0 #EXT_IN1 #EXT_IN2	P3.4 P3.5 P3.6

The internal peripherals of the 8051 microcontroller – used for this laboratory – connect to the external I/O pins through the Priority Crossbar Decoder. The Priority Crossbar Decoder is a crossbar that specifies the pin allocation for each internal

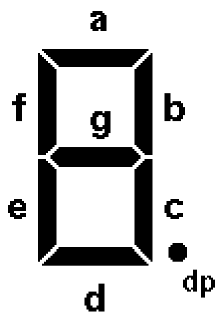
peripheral. All peripherals have a fixed priority that means enabling a peripheral with high priority will shift the peripherals with lower priority backwards. To understand the Priority Crossbar Decoder, and the pin assignments, check the configuration window of PCD in the “Configuration Wizard 2“ application’s „Peripherals->Port I/O” menu item.
A lot of peripherals can do they function without assigning IO pins for them, so it’s not required to assign a pin for example a timer peripheral.

To use the peripherals on the Extension Board, the following modules must be enabled in the Priority Crossbar Decoder: UART0, SPI0 (4 wire mode), SMBus (I^2C). (XBR0 = 0x07, XBR2=0x40). Besides of these the GPIO ports used as output, the push-pull mode must be enabled (P0MDOUT = 0x15, P3MDOUT = 0x0F, P4MDOUT = 0xFF, P5MDOUT = 0xFF, P6MDOUT = 0xFF, P7MDOUT = 0xFF).
For applying these mandatory settings always include the ***„adefs.inc”*** definition file, and call the **„A_Init”** function in the initialization part of the application.

Preparing for the laboratory

Review the schematics of the extension board! In the laboratory we will use the push buttons, the switches, the seven segment displays and the LEDs, so focus on these peripherals.

Fill the following table! The table specifies the content of the control byte for each displayable character on the 7 segment display.



	a 7S_A P6.0	b 7S_B P6.1	c 7S_C P6.2	d 7S_D P6.3	e 7S_E P6.4	f 7S_F P6.5	g 7S_G P6.6	dp 7S_DP P6.7	Hex value P6
0	X	X	X	X	X	X	-	-	3F
1									
2									
3									
4									
5									
6									
7									
8									
9									
A									

b									
C									
D									
E									
F									

Review the program code in ***„F04x_Blinky.asm”*** file – use the basics of assembly language presented on the lectures of "Microcontroller based systems". The program uses a software timer for blinking the LED on the 8051 development kit.

Laboratory tasks

Task 1. Basic GPIO usage

1.1.

*Write an application the will light up the **LD0** LED while the user holds the **BTN0** button in pressed state!*

Start the Keil µVision4 development environment, and select the *„Project->New->uVision Project"* menu item. Create a separate folder for the project on the user drive, and give a name to the project. In the next step select the Silicon Laboratories C8051F040-es microcontroller. The project wizard will ask about startup code generation, answer it with selecting the "No" option. In the Project Workspace tree, select the *„Target 1"* branch and from the right click menu, select the *„Options for Target: Target 1"* item. On the *„Debug"* tab, select the *„Use"* radio button, and set the *"Silicon Laboratories uVision driver"* in the driver combo box. At the *„Settings"*, select the *„USB Debug adapter" option* (the debugger interface must be connected to the PC). On *„Utilities"* tab at the *„Use target driver for Flash programming"* combobox select the *„Silicon Laboratories uVision driver"* option.
Open the ***„template.asm"*** file, and save it to the project folder, with an arbitrary name using the *"asm"* extension. At the last step, the file we have just saved into the project folder should be added to the project. For this, right click at the *„Source Group 1"* branch and select the *„Add files to Group Source Group 1"* menu item.

1.2.

*Change the functionality of the previous task! The **LD0** Led should light up when the user presses the **BTN0** button and it should switch off when the user presses the **BTN0** button again. Use the F0 user flag for temporary storage!*

1.3.

*Write an application that will show the state of the switches on the leds, when the user presses the **BTN0** button. If the user presses the **BTN1** button, all the leds should switch off!*

Task 2. Timer usage

2.1.

Write an application the will blink the LD0 led at 1 Hz frequency – using the Timer2 timer. (Without interrupts)

For the previous tasks the default clock settings (internal oscillator clock divided by 8 ≈ 3 MHz) were good enough, but for more accurate timing the clock settings should be modified. For this configuration use the *Configuration Wizard 2* application supplied by Silicon Laboratories. When the Configuration Wizard starts, the New Project dialog pops up, where the C8051F040 microcontroller should be selected, and in the *„Options->Code Format”* menu, select *ASM* option. Check the configuration options, and follow the changes in the different register values at the source code panel.

After successful configuration, the code made by the wizard can copied and pasted into our application code, or it can be saved in a separate file.

A generated configuration code can be seen in the following example – it will configure only the oscillator parameters of the microcontroller.

```
;------------------------------------
;-  Generated Initialization File  --
;------------------------------------


    rseg INIT

; Peripheral specific initialization functions,
; Called from the Init_Device label
Oscillator_Init:
    mov  SFRPAGE,   #CONFIG_PAGE
    mov  OSCICN,    #083h
    ret

; Initialization function for device,
; Call Init_Device from your main program
Init_Device:
    lcall Oscillator_Init
    ret
```

Setup the Timer2 timer to overflow in every 5 msec, and check the TF2 flag countinuously. (After successfully evaluating the TF2 value, don’t forget to delete it!)

Count the TF2 impulses in a register or variable and create the frequency for the blinking LED.

Declaring a variable at 0x40 address of the internal data segment:

```
               dseg AT 0x40
Cnt:           DS    1
```

The usage of the variable:

```
mov            Cnt, #0FFh
```

2.2.

Write an application - using the Timer2 timer – that will show a running light on the 8 leds! Use interrupts in the solution!

Modify the previous application to use interrupts. For handling the Timer2 interrupt, you must put the handler at the 0x002B address in the jump table.

```
                cseg  AT 0
                ljmp  Main

                org   0x002B
                ljmp  Timer2IT
                .
                .
                .
Timer2IT:       //Timer2 IT rutin
                reti
```

To enable the interrupt, the interrupt of the periprhery (ET2), and the global interrupt enable flag (EA) also should be set. Don't forget to save and restore the registers used in the interrupt handler routine!

Task 3.

Write an application that will display the state of the lower 4 switches as hexadecimal number on the leftmost 7 segment display.

Use the table prepared as homework, store it in the code memory (with DB control statement) and access these data using the MOVC instruction.

```
movc  A, @A+DPTR
```

Task 4.

Create a 4 digit hexadecimal counter for the 7 segment displays! The least significant digi should step forward in every 100 msec!

The Extension Board is capable to drive only one 7 segment display in a moment, but switching the driven digit quick enough will result as we will see all of them (persistence of vision), this usually called as time multiplexed display. For the counter use the Timer2 timer, and for the time multiplex display use the Timer3 timer – figure out the settings of both!

Related documents:

The Configuration Wizard 2 application:
http://www.silabs.com/Support%20Documents/Software/ConfigAndConfig2Install.exe.

Appendix 1. (ADefs.inc)

```
BTN             EQU   P1
BTN0            EQU   P1.0
BTN1            EQU   P1.1
BTN2            EQU   P1.2
BTN3            EQU   P1.3
BTN4            EQU   P1.4
BTN5            EQU   P1.5
BTN6            EQU   P1.6
BTN7            EQU   P1.7

SW              EQU   P2
SW0             EQU   P2.0
SW1             EQU   P2.1
SW2             EQU   P2.2
SW3             EQU   P2.3
SW4             EQU   P2.4
SW5             EQU   P2.5
SW6             EQU   P2.6
SW7             EQU   P2.7

LCD_RS          EQU   P3.0
LCD_R_nW EQU    P3.1
__LCD_BL EQU    P3.2
__LCD_EN EQU    P3.3
__EXT_IN0EQU    P3.4
__EXT_IN1EQU    P3.5
__EXT_IN2EQU    P3.6
I2TEMP_CMP      EQU   P3.7

_7SSEL0         EQU   P4.0
_7SSEL1         EQU   P4.1
__7SEN          EQU   P4.2
__LDEN          EQU   P4.3
__BTNEN         EQU   P4.4
__SWEN          EQU   P4.5
__CS_SPIPOT           EQU   P4.6
__CS_SPITEMP    EQU   P4.7

__LD            EQU   P5
__LD0           EQU   P5.0
__LD1           EQU   P5.1
__LD2           EQU   P5.2
__LD3           EQU   P5.3
__LD4           EQU   P5.4
__LD5           EQU   P5.5
__LD6           EQU   P5.6
__LD7           EQU   P5.7

_7S             EQU P6
_7S_A           EQU   P6.0
_7S_B           EQU   P6.1
_7S_C           EQU   P6.2
_7S_D           EQU   P6.3
_7S_E           EQU   P6.4
_7S_F           EQU   P6.5
_7S_G           EQU   P6.6
_7S_DP          EQU   P6.7
```

```
LCD_DB          EQU   P7
LCD_DB0         EQU   P7.0
LCD_DB1         EQU   P7.1
LCD_DB2         EQU   P7.2
LCD_DB3         EQU   P7.3
LCD_DB4         EQU   P7.4
LCD_DB5         EQU   P7.5
LCD_DB6         EQU   P7.6
LCD_DB7         EQU   P7.7

INIT SEGMENT CODE
    rseg INIT

A_Init:
    ; P0.0  -  TX0 (UART0), Push-Pull,  Digital
    ; P0.1  -  RX0 (UART0), Open-Drain, Digital
    ; P0.2  -  SCK  (SPI0), Push-Pull,  Digital
    ; P0.3  -  MISO (SPI0), Open-Drain, Digital
    ; P0.4  -  MOSI (SPI0), Push-Pull,  Digital
    ; P0.5  -  NSS  (SPI0), Open-Drain, Digital
    ; P0.6  -  SDA (SMBus), Open-Drain, Digital
    ; P0.7  -  SCL (SMBus), Open-Drain, Digital

    ; P1.0  -  Unassigned,  Open-Drain, Digital
    ; P1.1  -  Unassigned,  Open-Drain, Digital
    ; P1.2  -  Unassigned,  Open-Drain, Digital
    ; P1.3  -  Unassigned,  Open-Drain, Digital
    ; P1.4  -  Unassigned,  Open-Drain, Digital
    ; P1.5  -  Unassigned,  Open-Drain, Digital
    ; P1.6  -  Unassigned,  Open-Drain, Digital
    ; P1.7  -  Unassigned,  Open-Drain, Digital

    ; P2.0  -  Unassigned,  Open-Drain, Digital
    ; P2.1  -  Unassigned,  Open-Drain, Digital
    ; P2.2  -  Unassigned,  Open-Drain, Digital
    ; P2.3  -  Unassigned,  Open-Drain, Digital
    ; P2.4  -  Unassigned,  Open-Drain, Digital
    ; P2.5  -  Unassigned,  Open-Drain, Digital
    ; P2.6  -  Unassigned,  Open-Drain, Digital
    ; P2.7  -  Unassigned,  Open-Drain, Digital

    ; P3.0  -  Unassigned,  Push-Pull,  Digital
    ; P3.1  -  Unassigned,  Push-Pull,  Digital
    ; P3.2  -  Unassigned,  Push-Pull,  Digital
    ; P3.3  -  Unassigned,  Push-Pull,  Digital
    ; P3.4  -  Unassigned,  Open-Drain, Digital
    ; P3.5  -  Unassigned,  Open-Drain, Digital
    ; P3.6  -  Unassigned,  Open-Drain, Digital
    ; P3.7  -  Unassigned,  Open-Drain, Digital

    mov  SFRPAGE,   #CONFIG_PAGE
    mov  P0MDOUT,   #015h
    mov  P3MDOUT,   #00Fh
    mov  P4MDOUT,   #0FFh
    mov  P5MDOUT,   #0FFh
    mov  P6MDOUT,   #0FFh
    mov  P7MDOUT,   #0FFh
    mov  XBR0,      #007h
    mov  XBR2,      #040h
    ret
```

Appendix 2. (Template.asm)

```
$NOMOD51

#include <c8051f040.inc>
#include "ADefs.inc"

               cseg  AT 0
   //select the 0 address of the code segment,
               ljmp  Main
   //the command execution starts at 0 address after reset

Startup        segment CODE
               rseg  Startup
               using 0
         //select the register page 0

Main:          mov         WDTCN, #0DEh
         //disable WDT
               mov         WDTCN, #0ADh

               call  A_Init
         //Initializing ports according to the extension card

Startup:
               ljmp  Startup
END
```