
INTRODUCTION

This lab will continue our investigations into assembly language syntax in greater detail and have you write a complete code in assembly.

The objectives of this laboratory exercise would be:

- Create an assembly project in Code Composer Studio, write the entire code in assembly, compile it and download it on the MSP432 Launchpad.
- Learn how to set breakpoints and watch memory locations on the MCU using the debugging features of CCS

PART I – Assembly Project

Creating an assembly project is slightly different than creating a C project in Code Composer Studio. Follow the steps listed below to create the project:

1. Launch Code Composer Studio. Select **File → New → CCS Project** to open the ***New CCS Project*** dialog box.

2. Enter ***Lab3_Assembly*** as the Project name and select ***Empty Project*** as shown in Figure 2. Then click on **Finish** to create the project.

3. A new project called ***Lab3_Assembly*** gets created under **Project Explorer**. Right-click on the project name and select **New → Source File**.

4. **New Source File** dialog box appears. Enter the file name with the extension as shown in Figure 1 below. Then click **Finish**. A blank file without any text is created.

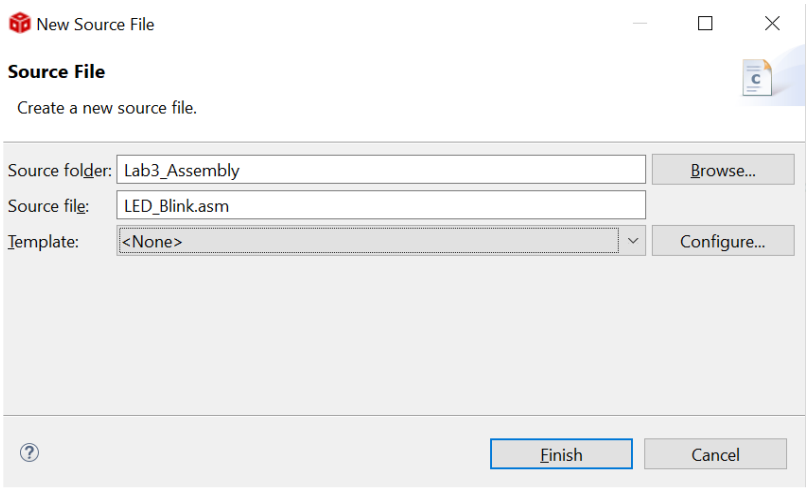

Figure 1: New assembly source file

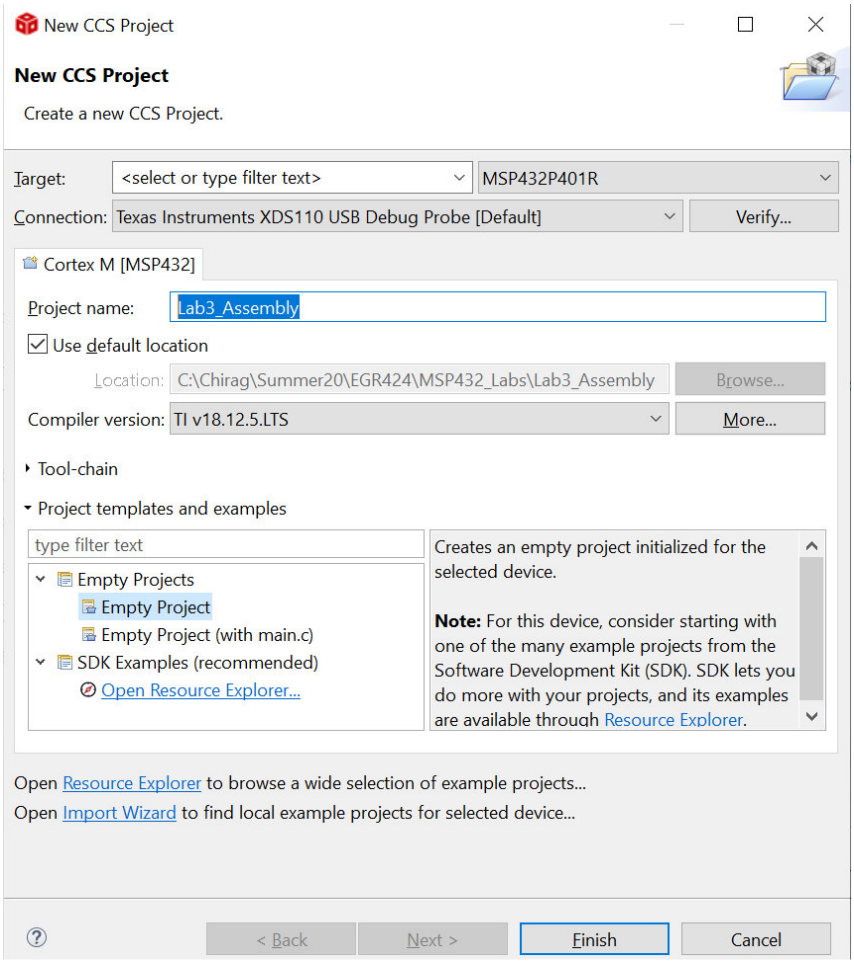

Figure 2: Creating an assembly project

An empty assembly project is created successfully.

PART II – Writing Assembly Code

In this part, you will write an assembly code that will **toggle LED1 on the MSP432 about every 0.5 seconds**.

Note:

- The comments are made using ';' as shown in the code below. Not with '/' like in case of C.
- The button used here is ACTIVE LOW which means the pin gets low whenever it is pressed.

Below is the template that you have to use for writing any assembly program.

```
.thumb
.text
.align 2
.global main

main:
; User code

.end
```

As can be seen in the template above:

- The first directive ***.thumb*** indicates that we will be using Thumb-2 mode for assembly instructions.
- The second directive ***.text*** indicates that the following lines correspond to assembly program. Hence they should be placed in the code section of MSP432 memory.
- The third directive ***.align*** defines the length of instructions to be processed in terms of bytes. If the instructions to be executed are of length 16 bits, then we should use ***.align 2*** as in our case indicating 2 bytes.
- The fourth directive ***.global*** defines a global variable main to be used throughout the program. In fact, this variable will indicate the starting address of the assembly program.
- Finally, the directive ***.end*** indicates the end of the assembly program.

1. Type the code as shown below in ***LED_Blink.asm*** file. The comments are included in the code.

Getting Started | Resource Explorer | main.c | msp432p401m_classic.h | LED_Blink.asm

```
;------- LEDs -------
;on-board LED 1 is connected to P1.0

    .thumb

    .text
    .align  2

; Addresses are available in msp432p401r_classic.h

P1IN    .field 0x40004C00,32  ; Port 1 Input
P1OUT   .field 0x40004C02,32  ; Port 1 Output
P1DIR   .field 0x40004C04,32  ; Port 1 Direction
P1REN   .field 0x40004C06,32  ; Port 1 Resistor Enable
P1DS    .field 0x40004C08,32  ; Port 1 Drive Strength
P1SEL0  .field 0x40004C0A,32  ; Port 1 Select 0
P1SEL1  .field 0x40004C0C,32  ; Port 1 Select 1

    .global main
    .thumbfunc  main

main:   .asmfunc
    BL  LED_Init        ; Branch to LED-Init()
Loop
    BL  LED_TOGGLE      ; Branch to LED_TOGGLE()

    MOV R0, #1000       ; Outer Loop
    BL  Delay           ; Branch to Delay()

    B   Loop            ; Repeat forever
    .endasmfunc         ; Indciates end of assembly function
```

```
32
33 ;------------LED_Init------------
34 ; Initialize GPIO Port P1.0 LED
35 LED_Init: .asmfunc
36     LDR  R1, P1SEL0
37     MOV  R0, #0x00          ; configure P1.0 as GPIO (SEL0 = 0)
38     STRB R0, [R1]
39
40     LDR  R1, P1SEL1
41     MOV  R0, #0x00          ; configure P1.0 as GPIO (SEL1 = 0)
42     STRB R0, [R1]
43
44     LDR  R1, P1DIR
45     MOV  R0, #0x01          ; make P1.0 as output pin
46     STRB R0, [R1]
47
48     LDR  R1, P1OUT
49     MOV  R0, #0x00          ; set LED to OFF
50     STRB R0, [R1]
51     BX   LR                 ; Return to main
52     .endasmfunc
53
54 ;------------LED_TOGGLE------------
55 LED_TOGGLE: .asmfunc
56     LDR  R1, P1OUT
57     LDRB R0, [R1]           ; Load the current value of LED
58     EOR  R0, R0, #0x01      ; Toggle
59     STRB R0, [R1]
60     BX   LR
61     .endasmfunc
62
63 ;-----------Delay_ms------------
64 ; This calculates delay
65 Delay: .asmfunc
66     MOVS R1, R0             ; Return if n = 0
67     BNE L1
68     BX  LR
69 L1
70     MOV R1, #250            ; Do inner loop 250 times
71 L2
72     SUBS R1, #1             ; Inner Loop
73     BNE L2
74     SUBS R0, #1             ; Do outer loop n times
75     BNE L1
76     BX  LR
77     .endasmfunc
78     .end
```

2. Once typed, build the project making sure it compiles without any errors. Now run the program on the MSP432 Launchpad and you should see that the on-board RED LED with flash at about every .5 seconds.

PART III – Your Turn

1. Create a new assembly file and call it ***Lab3.asm***. Type the code that will do the following:

    - When program is downloaded, all LEDs are OFF.
    - When S1 is pressed, on-board RGB LED turns BLUE
    - When S2 is pressed, on-board RGB LED turns RED
    - When S1 and S2 are pressed together, on-board RGB LED turns GREEN
    - When neither S1 nor S2 is pressed, on-board RGB LED is OFF
    - Your program sits in an infinite loop waiting for S1 or S2 to be pressed

2. Once you have typed the code then build your project. You will see that your project does not build successfully. You should see an error that says "***Symbol main is redefined***". This error is primarily due to 2 source files having main defined in them.

3. Expand ***Lab3_Assembly*** project under **Project Explorer**. Right-click ***LED_Blink.asm*** file and select **Exclude from Build** option. This will exclude the assembly file you created in Part II and will only use the assembly file you created in this part.

4. Build the project and make sure it compiles successfully. Now download the program and verify its working.

Demonstrate successful working of this part to your instructor in form of a video file.

Laboratory Deliverables

You must submit the following no later than the beginning of the next lab period.

- Report with cover page
- Brief description of your code from Part III
- Include well commented source code file ***(.asm)*** with a detailed description header section for each routine (as you were taught in your introductory C programming course) for Part III.
- Video file of your demonstration of Part III.