Application Report
SNVA822 – May 2018

LP5569 Lighting Pattern Design

ABSTRACT

This application report describes how to design lighting pattern with LP5569 Programmable lighting engine and provides examples of the fashion lighting pattern for the user.

Contents

List of Figures

List of Tables

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

This application report describes LP5559 lighting pattern design with examples for the device quick start. Most of the programs are presented with command compiler syntax. The Command compiler is described in The Control View - Source Edit Tab of Using the BOOST-LP5569EVM Evaluation Module. This application report also includes LED ring Demo (click here) sample code in both command syntax and C language. Command compiler software is available with the evaluation kit.

2 Device Overview

The LP5569 device is a programmable, easy-to-use 9-channel I2C LED driver designed to produce lighting effects for various applications. The LED driver is equipped with an internal SRAM memory for user programmed sequences and three programmable LED engines, which allow operation without processor control. Autonomous operation reduces system power consumption when the processor is put in sleep mode.

3 Lighting Pattern Design with LEDs on LP5569EVM

The following lighting pattern is realized with white LEDs on LP5569EVM.

3.1 Bouncing Effect

This design is best viewed with 9 white LEDs on the LP5569EVM. This design has 4 white LEDs on at all times. Each LED has different PWM settings to create two tracers bouncing back and forth.

```
; bouncing.src

L10:    dw      0000000000000001b
L11:    dw      0000000000000010b
L12:    dw      0000000000000100b
L13:    dw      0000000000001000b
L14:    dw      0000000000010000b
L15:    dw      0000000000001000b
L16:    dw      0000000000000100b
L17:    dw      0000000000000010b
L20:    dw      0000000100000000b
L21:    dw      0000000010000000b
L22:    dw      0000000001000000b
L23:    dw      0000000000100000b
L24:    dw      0000000000010000b
L25:    dw      0000000000100000b
L26:    dw      0000000001000000b
L27:    dw      0000000010000000b

.segment    program1
        map_start    L10        ;load the start address
        load_end     L17        ;load the end address
loop1:
        trigger      s{2}
        set_pwm      0
        map_next
        set_pwm      30
        map_next
        set_pwm      100
        map_prev
        wait         0.1        ;wait time to create effect
        branch       0, loop1
        end
.segment    program2
        map_start    L20        ;load the start address
        load_end     L27        ;load the end address
loop2:
        trigger      w{1}
        set_pwm      0
        map_next
        set_pwm      30
        map_next
        set_pwm      100
        map_prev
        branch       0, loop2
        end
.segment             program3
        end
```

3.2 Breath_white Effect

This design is best viewed with 9 white LEDs on the LP5569EVM. This design has 3 group LED breathing.

```
GRP1:       dw       0000000001001001b
GRP2:       dw       0000000010010010b
GRP3:       dw       0000000100100100b
.segment    program1        ;Begin of a segment
        map_addr    GRP1
        set_pwm     00
loop1:
        ramp        1, 100
        ramp        1, -100
        wait        0.3
        branch      0, loop1
        end
.segment    program2        ;Begin of a segment
        map_addr    GRP2
        set_pwm     00
loop2:
        ramp        1, 100
        ramp        1, -100
        wait        0.3
        branch      0, loop2
        end
.segment    program3        ;Begin of a segment
        map_addr    GRP3
        set_pwm     00
loop3:
        ramp        1, 100
        ramp        1, -100
        wait        0.3
        branch      0, loop3
        end
```

3.3 Chaser Effect

This design is best viewed with 9 white LEDs on the LP5569EVM. This design has 3 white LEDs on at all times. Each LED has different PWM settings to create a chaser (or tracer) effect.

```
; chaser.src
;
L00:    dw     0000000000000001b
L01:    dw     0000000000000010b
L02:    dw     0000000000000100b
L03:    dw     0000000000001000b
L04:    dw     0000000000010000b
L05:    dw     0000000000100000b
L06:    dw     0000000001000000b
L07:    dw     0000000010000000b
L08:    dw     0000000100000000b
L09:    dw     0000000010000000b
L10:    dw     0000000001000000b
L11:    dw     0000000000100000b
L12:    dw     0000000000010000b
L13:    dw     0000000000001000b
L14:    dw     0000000000000100b
L15:    dw     0000000000000010b
ALL:    dw     0000000111111111b

.segment    program1
        map_addr     ALL
        ramp         0.5, 150
        ramp         0.5, -150
        wait         0.2
        map_start    L00        ;load the start address
        load_end     L15        ;load the end address
loop1:
        set_pwm      0
        map_next
        set_pwm      5
        map_next
        set_pwm      50
        map_next
        set_pwm      150
        map_prev
        map_prev
        wait         0.07        ;wait time to create effect
        branch       0, loop1
        end
.segment    program2
        end
```

4 Lighting Pattern Design with LEDs on LP5569 Ring Demo

The LED ring is the latest HMI in the smart personal electronic device space and improves the user experience. More and more appliance vendors are adopting this concept in next generation products. For the appliance customer, the existing model with the existing MCU already meets system specification, but the fancy lighting pattern will exhaust the system resources and potentially cause the MCU to crash. It is requires a triple design circle to design complex lighting patterns without an engine control LED driver.

According to Figure 1 for LED board design, U1,U2,U3, and U5 drive 12 pcs RGB LED modules. The LED mapping and I2C address assignments are shown as the Table 1.

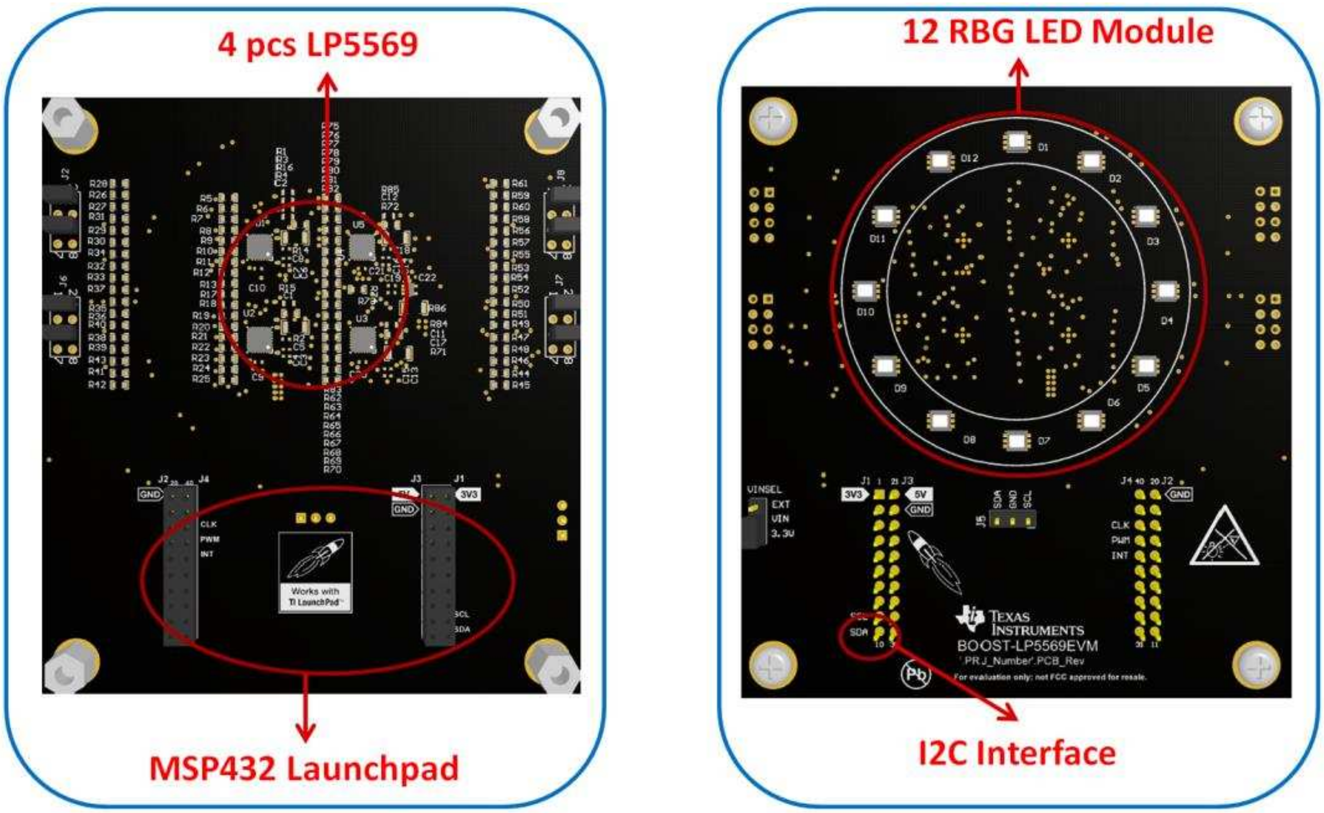

Figure 1. LED Board Design

Table 1. LED and I2C Address Assignment

Device	I2C Address	Broadcasting I2C Address	Channel Number of The LED Driver	LED
U1	0x32H	0x40H	LED0, LED3, LED6	D1-B, D1-G,D1-R
U1	0x32H	0x40H	LED1, LED4, LED7	D2-B, D2-G,D2-R
U1	0x32H	0x40H	LED2,LED5, LED8	D3-B, D3-G,D3-R
U2	0x33H	0x40H	LED0, LED3, LED6	D4-B, D4-G,D4-R
U2	0x33H	0x40H	LED1, LED4, LED7	D5-B, D5-G,D5-R
U2	0x33H	0x40H	LED2,LED5, LED8	D6-B, D6-G,D6-R
U3	0x34H	0x40H	LED0, LED3, LED6	D7-B, D7-G,D7-R
U3	0x34H	0x40H	LED1, LED4, LED7	D8-B, D8-G,D8-R
U3	0x34H	0x40H	LED2,LED5, LED8	D9-B, D9-G,D9-R
U5	0x35H	0x40H	LED0, LED3, LED6	D10-B, D10-G,D10-R
U5	0x35H	0x40H	LED1, LED4, LED7	D11-B, D11-G,D11-R
U5	0x35H	0x40H	LED2,LED5, LED8	D12-B, D12-G,D12-R

Firstly, define the LED Mapping in the beginning of the engine coding as shown below.

```
row1:        dw    0000000001001001b    ;Map B LED = D1, D4, D7 on the eval. board.
             dw    0000000010010010b    ;Map G LED = D2, D5, D8 on the eval. board.
             dw    0000000100100100b    ;Map R LED = D3, D6, D9 on the eval. board.
             dw    0000000011011011b    ;Map BG LED on the eval. board.
             dw    0000000110110110b    ;Map GR LED on the eval. board.
             dw    0000000101101101b    ;Map RB LED on the eval. board.
row7:        dw    0000000111111111b    ;Map all LEDs on the eval. board.
row8:        dw    0000000001001001b    ;Map B LED = D1,D4,D7 on the eval. board.
row9:        dw    0000000010010010b    ;Map G LED = D2,D5,D8 on the eval. board.
```

4.1 Breathing

During the breathing pattern, all LEDs fade in and out as the same color at the same rate, therefore all devices should run the same engine code below.

```
.segment        program1     ;Program for engine 1.
loop1_0:
        map_start    row1        ;Map the first LED.
        load_end     row7        ;End address of the mapping data table.
loop1:
        ramp         2, 200      ;Increase PWM 0->78% in 2 second.
        ramp         2, -255     ;Decrease PWM ->0% in 2 seconds.
        wait         0.4         ;Wait for 0.4 seconds.
        ramp         2, 200      ;Increase PWM 0->78% in 2 second.
        ramp         2, -255     ;Decrease PWM ->0% in 2 seconds.
        wait         0.4         ;Wait for 0.4 seconds.
        map_next                 ;Set the next row active in the mapping table.
        branch       6,loop1     ;Loop 6 time
        map_addr     row8
        ramp         1.5, 200    ;Increase PWM 0->78% in 1.5 second.
        wait         0.4         ;Wait for 0.4 seconds.
        wait         0.4         ;Wait for0.4 seconds.
        map_addr     row9
        ramp         3, 200      ;Increase PWM 0->78% in 3 second.
        ramp         3, -255     ;Decrease PWM ->0% in 3 seconds.
        ramp         3, 200      ;Increase PWM 0->78% in 3 second.
        ramp         3, -255     ;Decrease PWM ->0% in 3 seconds.
        map_addr     row8
        ramp         1.5, -255   ;Decrease PWM ->0% in 1.5 seconds.
```

4.2 Mono-Color Chasing

The mono-color chasing pattern needs the devices to start execution with a specific sequence delay. Each device has different code, as shown in Figure 2.

U1

```
           wait       0.4
           ld         ra, 75
loop1_2:   map_sel    7
           set_pwm    ra
           map_sel    4
           sub        ra, 20
           set_pwm    ra
           map_sel    1
           sub        ra, 20
           set_pwm    ra
           add        ra, 20
           wait       0.03
           branch     63, loop1_2
           branch     10, loop1_2
           wait       0.1
           map_addr   row7
           set_pwm    0
```

U2

```
           wait       0.4
           ld         ra, 135
loop1_2:   map_sel    7
           set_pwm    ra
           map_sel    4
           sub        ra, 20
           set_pwm    ra
           map_sel    1
           sub        ra, 20
           set_pwm    ra
           add        ra, 20
           wait       0.03
           branch     63, loop1_2
           branch     3, loop1_2
           wait       0.1
           map_addr   row7
           set_pwm    0
```

U5

```
           wait       0.4
           ld         ra, 255
loop1_2:   map_sel    7
           set_pwm    ra
           map_sel    4
           sub        ra, 20
           set_pwm    ra
           map_sel    1
           sub        ra, 20
           set_pwm    ra
           add        ra, 20
           wait       0.03
           branch     63, loop1_2
           branch     3, loop1_2
           wait       0.1
           map_addr   row7
           set_pwm    0
```

U3

```
           wait       0.4
           ld         ra, 195
loop1_2:   map_sel    7
           set_pwm    ra
           map_sel    4
           sub        ra, 20
           set_pwm    ra
           map_sel    1
           sub        ra, 20
           set_pwm    ra
           add        ra, 20
           wait       0.03
           branch     63, loop1_2
           branch     3, loop1_2
           wait       0.1
           map_addr   row7
           set_pwm    0
```

Figure 2. Mono-Color Chasing Engine Code for 4 Devices

4.3 Multi-Color Chasing

The multi-color chasing pattern requires the devices to start execution with a specific sequence delay. Each device has different code, as shown in Figure 3 and Figure 4.

U1

```
.segment    program2
loop2_3:    wait        0.08
            branch      2, loop2_3
            map_clr
            trigger     s{3}
            rst
            trigger     w{1}
            map_addr    row8
            ramp        1, 200
loop2_1:    wait        0.08
loop2_2:    map_sel     2
            set_pwm     200
            wait        0.08
            map_sel     5
            set_pwm     200
            wait        0.08
            map_sel     8
            set_pwm     200
            wait        0.08
            map_sel     2
            set_pwm     0
            wait        0.08
            map_sel     5
            set_pwm     0
            wait        0.08
            map_sel     8
            set_pwm     0
            wait        0.08
            map_sel     3
            set_pwm     200
            wait        0.08
            map_sel     6
            set_pwm     200
            wait        0.08
            map_sel     9
            set_pwm     200
            wait        0.08
            map_sel     3
            set_pwm     0
            wait        0.08
            map_sel     6
            set_pwm     0
            wait        0.08
            map_sel     9
            set_pwm     0
            wait        0.08
            branch      63, loop2_2
            map_addr    row7
            set_pwm     0
loop2_3:    wait        0.08
            branch      2, loop2_3
            map_clr
            trigger     s{3}
            rst
```

U2

```
.segment    program2
            trigger     w{1}
            map_addr    row8
            ramp        1, 200
loop2_1:    wait        0.08
            branch      3, loop2_1
loop2_2:    map_sel     2
            set_pwm     200
            wait        0.08
            map_sel     5
            set_pwm     200
            wait        0.08
            map_sel     8
            set_pwm     200
            wait        0.08
            map_sel     2
            set_pwm     0
            wait        0.08
            map_sel     5
            set_pwm     0
            wait        0.08
            map_sel     8
            set_pwm     0
            wait        0.08
            map_sel     3
            set_pwm     200
            wait        0.08
            map_sel     6
            set_pwm     200
            wait        0.08
            map_sel     9
            set_pwm     200
            wait        0.08
            map_sel     3
            set_pwm     0
            wait        0.08
            map_sel     6
            set_pwm     0
            wait        0.08
            map_sel     9
            set_pwm     0
            wait        0.08
            branch      63, loop2_2
            map_addr    row7
            set_pwm     0
            map_clr
            trigger     s{3}
            rst
```

Figure 3. Multi-Color Chasing Code for U1 and U2

U5

```
segment     program2
            trigger     w{1}
            map_addr    row8
            ramp        1, 200
loop2_1:    wait        0.08
            branch      3, loop2_1
loop2_2:    map_sel     3
            set_pwm     200
            wait        0.08
            map_sel     6
            set_pwm     200
            wait        0.08
            map_sel     9
            set_pwm     200
            wait        0.08
            map_sel     3
            set_pwm     0
            wait        0.08
            map_sel     6
            set_pwm     0
            wait        0.08
            map_sel     9
            set_pwm     0
            wait        0.08
            map_sel     2
            set_pwm     200
            wait        0.08
            map_sel     5
            set_pwm     200
            wait        0.08
            map_sel     8
            set_pwm     200
            wait        0.08
            map_sel     2
            set_pwm     0
            wait        0.08
            map_sel     5
            set_pwm     0
            wait        0.08
            map_sel     8
            set_pwm     0
            wait        0.08
            branch      63, loop2_2
            map_addr    row7
            set_pwm     0
            map_clr
            trigger     s{3}
            rst
```

U3

```
.segment    program2
            trigger     w{1}
            map_addr    row8
            ramp        1, 200
loop2_1:    wait        0.08
loop2_2:    map_sel     2
            set_pwm     200
            wait        0.08
            map_sel     5
            set_pwm     200
            wait        0.08
            map_sel     8
            set_pwm     200
            wait        0.08
            map_sel     2
            set_pwm     0
            wait        0.08
            map_sel     5
            set_pwm     0
            wait        0.08
            map_sel     8
            set_pwm     0
            wait        0.08
            map_sel     3
            set_pwm     200
            wait        0.08
            map_sel     6
            set_pwm     200
            wait        0.08
            map_sel     9
            set_pwm     200
            wait        0.08
            map_sel     3
            set_pwm     0
            wait        0.08
            map_sel     6
            set_pwm     0
            wait        0.08
            map_sel     9
            set_pwm     0
            wait        0.08
            branch      63, loop2_2
            map_addr    row7
            set_pwm     0
loop2_3:    wait        0.08
            branch      2, loop2_3
            map_clr
            trigger     s{3}
            rst
```

Figure 4. Multi-Color Chasing Code for U3 and U5

4.4 Door Open

The door open pattern needs the devices to start execution with a specific sequence delay. Each device has different code as shown in Figure 5.

U1

```
loop1_8:  wait      0.4
          map_sel   1
          set_pwm   200
          wait      0.05
          map_sel   4
          set_pwm   200
          wait      0.05
          map_sel   7
          set_pwm   200
loop1_9:  wait      0.05
          branch    6, loop1_9
          wait      0.4
          map_sel   7
          set_pwm   0
          wait      0.05
          map_sel   4
          set_pwm   0
          wait      0.05
          map_sel   1
          set_pwm   0
          wait      0.05
          branch    63, loop1_8
          map_clr
          trigger   s{2}
          trigger   w{3}
          rst
```

U2

```
loop1_8:  wait      0.4
loop1_9:  wait      0.05
          branch    2, loop1_9
          map_sel   1
          set_pwm   200
          wait      0.05
          map_sel   4
          set_pwm   200
          wait      0.05
          map_sel   7
          set_pwm   200
          wait      0.05
          wait      0.4
          map_sel   7
          set_pwm   0
          wait      0.05
          map_sel   4
          set_pwm   0
          wait      0.05
          map_sel   1
          set_pwm   0
loop1_10: wait      0.05
          branch    3, loop1_10
          branch    63, loop1_8
          map_clr
          trigger   s{2}
          trigger   w{3}
          rst
```

U5

```
loop1_8:  wait      0.4
loop1_9:  wait      0.05
          branch    2, loop1_9
          map_sel   7
          set_pwm   200
          wait      0.05
          map_sel   4
          set_pwm   200
          wait      0.05
          map_sel   1
          set_pwm   200
          wait      0.05
          wait      0.4
          map_sel   1
          set_pwm   0
          wait      0.05
          map_sel   4
          set_pwm   0
          wait      0.05
          map_sel   7
          set_pwm   0
loop1_10: wait      0.05
          branch    3, loop1_10
          branch    63, loop1_8
          map_clr
          trigger   s{2}
          trigger   w{3}
```

U3

```
loop1_8:  wait      0.4
          map_sel   7
          set_pwm   200
          wait      0.05
          map_sel   4
          set_pwm   200
          wait      0.05
          map_sel   1
          set_pwm   200
loop1_9:  wait      0.05
          branch    6, loop1_9
          wait      0.4
          map_sel   1
          set_pwm   0
          wait      0.05
          map_sel   4
          set_pwm   0
          wait      0.05
          map_sel   7
          set_pwm   0
          wait      0.05
          branch    63, loop1_8
          map_clr
          trigger   s{2}
          trigger   w{3}
          rst
```

Figure 5. Door Open

4.5 Coding Tips

- Use the branch instruction to synchronize multiple devices, as it guarantees all devices will be at the same time scale and step.
- Remember to clear the LED mapping with map_clr when using the same LED in the difference engine.
- Use the appropriate variable (ra, rb, rc, rd) for the progressive increase or decrease operation.

4.6 Uploading The Program to SRAM

The compile tool (Lasm.exe) can be downloaded from the GUI package on ti.com. The command window or GUI can be used to compile the .scr file.

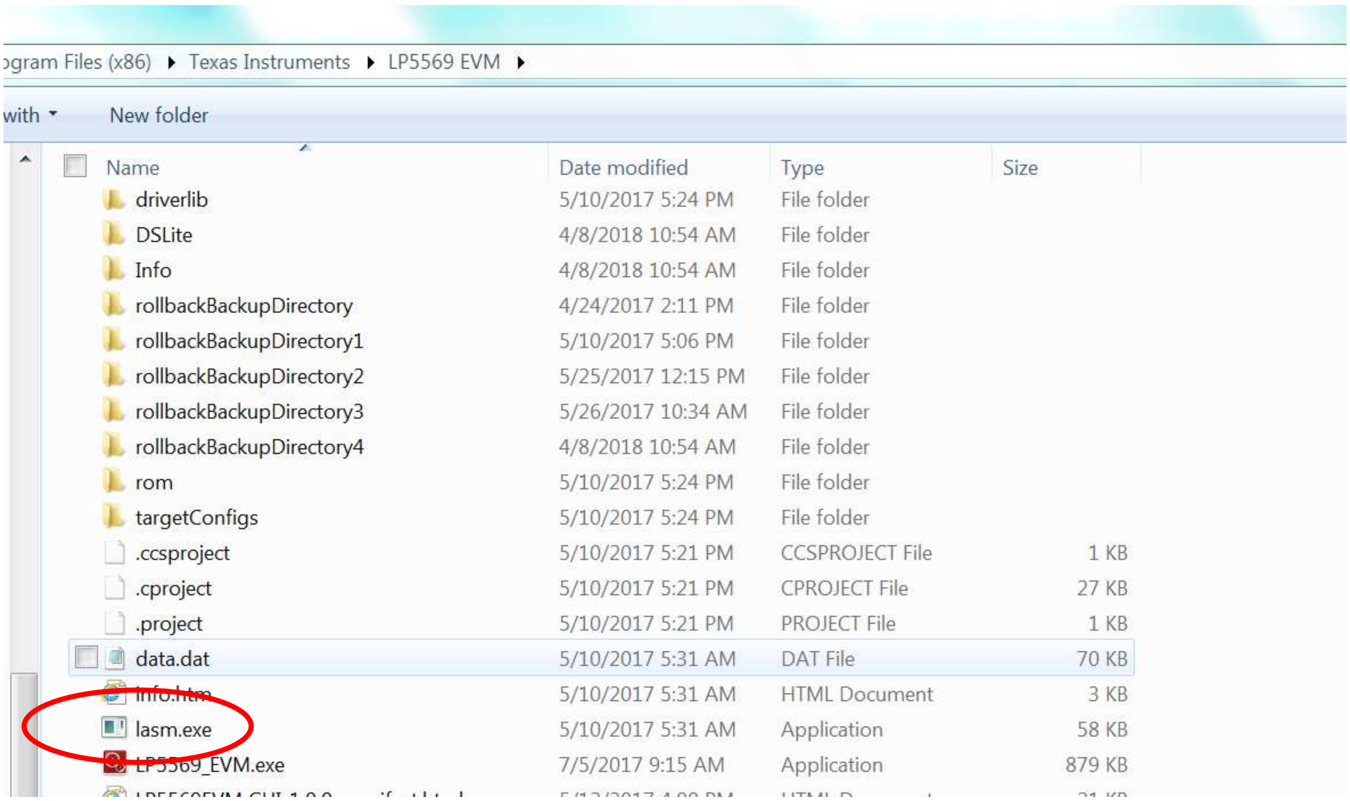

Figure 6. lasm.exe Path

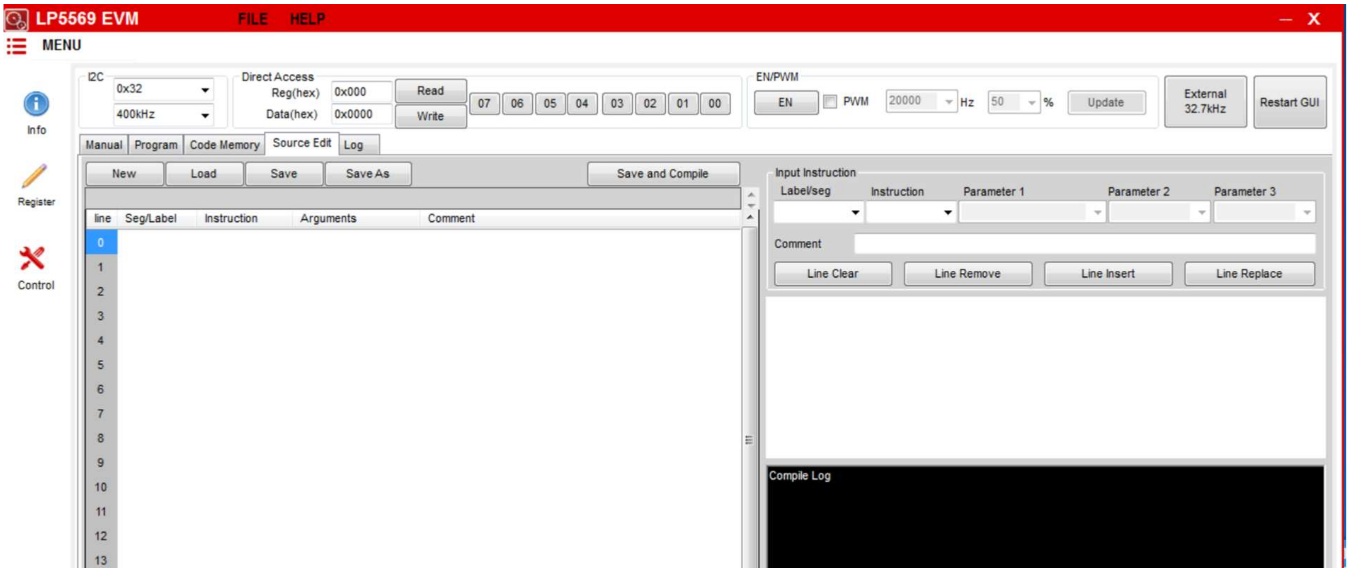

Figure 7. Compile Panel in LP5569EVM GUI

After the compiling, a .hex file will appear in the same folder as the .scr file.

```
00 49 00 92 01 24 00 DB 01 B6 01 6D 01 FF 00 49
00 92 9C 00 9C 86 28 C8 21 FF 74 00 28 C8 21 FF
74 00 9D 80 A3 02 9F 87 1E C8 74 00 74 00 9F 88
3E C8 31 FF 3E C8 31 FF 9F 87 19 FF 74 00 90 FF
9D 07 84 60 9D 04 92 14 84 60 9D 01 92 14 84 60
91 14 44 00 BF 97 A1 97 4C 00 9F 86 40 00 74 00
90 FF 9F 87 14 C8 74 00 9D 08 84 60 9D 05 92 14
84 60 9D 02 92 14 84 60 91 14 44 00 BF AB A1 AB
4C 00 9F 86 40 00 9F 87 14 C8 46 00 A4 BC 9D 02
40 C8 46 00 9D 05 40 C8 46 00 9D 08 40 C8 46 00
9D 02 40 00 46 00 9D 05 40 00 46 00 9D 08 40 00
46 00 A3 4F A8 3E 9F 86 40 00 74 00 9D 07 40 C8
46 00 9D 04 40 C8 46 00 9D 01 40 C8 46 00 A3 5D
74 00 9D 01 40 00 46 00 9D 04 40 00 46 00 9D 07
40 00 46 00 A2 54 9D 00 E0 04 E2 00 00 00 E0 80
9F 87 14 C8 4A 00 A1 83 9D 03 40 C8 4A 00 9D 06
40 C8 4A 00 9D 09 40 C8 4A 00 9D 03 40 00 4A 00
9D 06 40 00 4A 00 9D 09 40 00 4A 00 9D 02 40 C8
4A 00 9D 05 40 C8 4A 00 9D 08 40 C8 4A 00 9D 02
40 00 4A 00 9D 05 40 00 4A 00 9D 08 40 00 4A 00
A5 05 9F 86 40 00 9D 00 E0 08 00 00 E1 00 46 00
A4 01 9D 03 40 D7 46 00 9D 06 40 EB 46 00 9D 09
40 FF 46 00 9D 09 40 00 46 00 9D 06 40 00 46 00
9D 03 40 00 46 00 A4 94 A2 01 9D 00 E0 02 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
          @ 09 program1
          @ 77 program2
          @ A6 program3
```

Then, copy the hex file to the array table and upload the data to the SRAM by the below coding.

```
void load_SRAM()
{
    int i,j;
    MAP_I2C_setSlaveAddress(EUSCI_B1_BASE,0x40);                      //Device global setting
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x2F);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x48);      //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x02);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x54);      //send register data

    MAP_I2C_setSlaveAddress(EUSCI_B1_BASE, 0x32);                     //load hex to SRAM in U1
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4b);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x09);      //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4c);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x78);      //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4d);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0xa8);      //send register data
        for(j=0; j<16; j++)
        {
            MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4F); //send register address
            MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,j);    //send register data
            MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE,0x50);  //send register address
            for(i=0; i<32; i++)
             MAP_I2C_masterSendMultiByteNext(EUSCI_B1_BASE,table_32[i+j*32]);//send register data
        }

    MAP_I2C_setSlaveAddress(EUSCI_B1_BASE, 0x33);                     //load hex to SRAM in U2
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4b);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x09);      //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4c);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x7b);      //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4d);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0xaa);      //send register data
        for(j=0; j<16; j++)
        {
            MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4F); //send register address
            MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,j);    //send register data
            MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE,0x50);  //send register address
            for(i=0; i<32; i++)
             MAP_I2C_masterSendMultiByteNext(EUSCI_B1_BASE,table_33[i+j*32]);//send register data
        }

    MAP_I2C_setSlaveAddress(EUSCI_B1_BASE, 0x34);                     //load hex to SRAM in U3
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4b);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x09);      //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4c);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x7b);      //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4d);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0xab);      //send register data
        for(j=0; j<16; j++)
        {
            MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4F); //send register address
            MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,j);    //send register data
            MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE,0x50);  //send register address
            for(i=0; i<32; i++)
             MAP_I2C_masterSendMultiByteNext(EUSCI_B1_BASE,table_34[i+j*32]);//send register data
        }

    MAP_I2C_setSlaveAddress(EUSCI_B1_BASE, 0x35);                     //load hex to SRAM in U5
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4b);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x09);      //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4c);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x77);      //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4d);      //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0xa6);      //send register data
```

```
            for(j=0; j<16; j++)
            {
                MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x4F); //send register address
                MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,j);    //send register data
                MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE,0x50);  //send register address
                for(i=0; i<32; i++)
                 MAP_I2C_masterSendMultiByteNext(EUSCI_B1_BASE,table_35[i+j*32]);//send register data;
            }

        MAP_I2C_setSlaveAddress(EUSCI_B1_BASE, 0x32);                  //SET U1 as the clk out
            MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x2F);     //send register address
            MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x49);     //send register data
            MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x3d);     //send register address
            MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x08);     //send register data

    MAP_I2C_setSlaveAddress(EUSCI_B1_BASE, 0x40);                      //Run all the engine
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x02);         //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0x00);         //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x02);         //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0xa8);         //send register data
        MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE, 0x01);         //send register address
        MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,0xa8);         //send register data
    }
```

5 References

Using the BOOST-LP5569EVM Evaluation Module

LP5569 Nine-Channel I2C RGB LED Driver With Engine Control and Charge Pump