LED-ek vezérlése shift regiszterrel

MICLAB-07

Név: Stefán Kornél

Dátum: 2024. 10. 21.

Mérőhely: 7 bal

Bevezetés

Az interrupt használatának megismerése.

Ajánlott irodalom

- A házi feladatban találhatók
- Honlap: http://www.inf.u-szeged.hu/noise/Education/MicLab/

Jegyzőkönyv készítése

A jegyzőkönyvek az órán végzett munka dokumentálására szolgálnak. A letölthető minta jegyzőkönyvet kell kiegészíteni a megfelelő információkkal: név, dátum, mérőhely (pl. 3. jobb), a feladatokhoz tartozó esetleges kifejtendő válaszokkal, valamint a kódok lényeges részével.

A jegyzőkönyveket a CooSpace-en kell feltölteni, külön pdf formátumban csatolni kell a jegyzőkönyvet (a fájl neve a következő mintát kövesse: NagyJ.KissB.03.pdf), egy külön zip fájlban pedig a kódokat (*.c, *.cwg). Amennyiben probléma merül fel a beadás során, az anyagokat az oktató e-mail címére kell elküldeni, levél tárgya legyen pl. MicLab 03.

1. feladat – LED1 és LED7 bekapcsolása

Kapcsolja be a kiegészítő panelen lévő LED1 és LED7 nevű LED-eket. (Kövesse az órai ppt-t és használja a MicLab-utmutato.pdf-ben lévő kapcsolódó részeket.)

Konfigurálja az SPI-t az órai diasor segítségével. A Clock Phase és a Clock Polarity beállításaival 4 eset lehetséges az adat (MOSI) és az órajel (SCK) viszonyára. Válassza ki a Clock Phase és a Clock Polarity helyes kombinációját a shift regiszter idődiagramja és a Laboratory Practicals pdf 7.7-es ábrája alapján. Az SPI 3 vezetékes módban legyen és az órajele legyen ~1,5 MHz.

Az SPI **polling módon** történő használatával töltse fel a shift regisztert. A shift register írásához készítsen egy saját függvényt. Ne feledje, hogy a shift register csak akkor adja ki a kimeneteire a beleírt byte-ot, ha az OE_RCLK vonalon egy felfutó élt kap.

A program részekre bontott forráskódja (Config, Main.c, Interrupts.c, ha van):

Config

```
<?xml version="1.0" encoding="ASCII"?>
<device:XMLDevice xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:device="http://www.silabs.com/ss/hwconfig/document/device.ecore" name="EFM8BB10F8G-A-QSOP24"
partId="mcu.8051.efm8.bb1.efm8bb10f8g-a-qsop24" version="4.0.0" contextId="%DEFAULT%">
  <mode name="DefaultMode">
```

```
    <property object="CROSSBAR0" propertyId="xbar0.spi0.clockdata" value="Enabled"/>
    <property object="DefaultMode" propertyId="mode.diagramLocation" value="100, 100"/>
    <property object="INTERRUPT_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="INTERRUPT_0" propertyId="interrupt.interruptenable.enableallinterrupts"
value="Enabled"/>
    <property object="P0.0" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.0" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P0.2" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.2" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P0.3" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.3" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P0.4" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.4" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P1.1" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P1.1" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P1.2" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P1.2" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="PBCFG_0" propertyId="pbcfg.settings.enablecrossbar" value="Enabled"/>
    <property object="SPI_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="SPI_0" propertyId="spi.configuration.clockphase" value="Data sample on second
edge"/>
    <property object="SPI_0" propertyId="spi.configuration.enablemastermode" value="Enable"/>
    <property object="SPI_0" propertyId="spi.configuration.spimode" value="Master"/>
    <property object="SPI_0" propertyId="spi.control.slaveselectmode" value="Slave or master 3-wire
mode"/>
    <property object="SPI_0" propertyId="spi.control.spienable" value="Enabled"/>
    <property object="SPI_0" propertyId="spi.view.view" value="Advanced"/>
    <property object="WDT_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="WDT_0" propertyId="wdt.watchdogcontrol.wdtenable" value="Disable"/>
  </mode>
  <modeTransition>
    <property object="RESET &#x2192; DefaultMode" propertyId="modeTransition.source" value="RESET"/>
    <property object="RESET &#x2192; DefaultMode" propertyId="modeTransition.target"
value="DefaultMode"/>
  </modeTransition>
</device:XMLDevice>
```

Main.c

```
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <SI_EFM8BB1_Register_Enums.h>                  // SFR declarations
#include "InitDevice.h"
// $[Generated Includes]
// [Generated Includes]$

#define DispClock P0_B0
#define DispData P0_B2
#define DispOutEnable P0_B3

#define DecoderA P1_B1
#define DecoderB P1_B2
#define DecoderC P0_B4

// Displays
enum {
  LED_T1,
  LED_T2,
  LED_T3,
  LED_T4,
  LED_LEDS
};

// Leds on display
enum {
  LED_7 = 1,
  LED_1 = 2,
  LED_2 = 4,
  LED_3 = 8,
  LED_6 = 16,
  LED_5 = 32,
  LED_4 = 64,
};
```

```c
#define LEDS_TO_ACTIVE ~(LED_1 | LED_7)

//-----------------------------------------------------------------------------
// SiLabs_Startup() Routine
// ----------------------------------------------------------------------------
// This function is called immediately after reset, before the initialization
// code is run in SILABS_STARTUP.A51 (which runs before main() ). This is a
// useful place to disable the watchdog timer, which is enable by default
// and may trigger before main() in some instances.
//-----------------------------------------------------------------------------
void SiLabs_Startup (void)
{
  // $[SiLabs Startup]
  // [SiLabs Startup]$
}

// Enable selected display.
// Tudom hogy most overkill, de hasznos lehet még
void select_display(uint8_t display)
{
  switch (display) {
    case LED_T1:
      DecoderA = 0;
      DecoderB = 0;
      DecoderC = 0;

      break;
    case LED_T2:
      DecoderA = 1;
      DecoderB = 0;
      DecoderC = 0;

      break;
    case LED_T3:
      DecoderA = 0;
      DecoderB = 1;
      DecoderC = 0;

      break;
    case LED_T4:
      DecoderA = 1;
      DecoderB = 1;
      DecoderC = 0;

      break;
    case LED_LEDS:
      DecoderA = 0;
      DecoderB = 0;
      DecoderC = 1;

      break;

    default:
      break;
  }
}

void write_to_spi(uint8_t data_to_send) {
  SPI0CN0_SPIF = 0;
  SPI0DAT = data_to_send;

  // Wait until data is sent out
  while(!SPI0CN0_SPIF);

  SPI0CN0_SPIF = 1;
}

//-----------------------------------------------------------------------------
// main() Routine
// ----------------------------------------------------------------------------
int main (void)
{
  // Call hardware initialization routine
```

```
  enter_DefaultMode_from_RESET();

  select_display(LED_LEDS);

  while (1)
  {
    // $[Generated Run-time code]
    // [Generated Run-time code]$


          // disable leds
          DispOutEnable = 1;

          //write to spi
          write_to_spi(LEDS_TO_ACTIVE);

          //send signal to bit shifter.
          DispOutEnable = 0;
  }
}
```

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

- Igen
- Nem

A program működött:

- Igen
- Nem

2. feladat – Elektronikus dobókocka megvalósítása

Írjon egy olyan programot, ami a 7db LED segítségével meg tudja jeleníteni egy 6 oldalú dobókocka oldalait. Amíg a kiegészítő panelen lévő SW1 nyomógomb le van nyomva, addig léptesse a főprogramban a dobókocka oldalait. Mivel a léptetés nagy sebességgel történik, így teljesíthető a véletlenszerűség, ami egy kockadobásnak tekinthető.

A program részekre bontott forráskódja (Config, Main.c, Interrupts.c, ha van):

config

```
<?xml version="1.0" encoding="ASCII"?>
<device:XMLDevice xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:device="http://www.silabs.com/ss/hwconfig/document/device.ecore" name="EFM8BB10F8G-A-QSOP24"
partId="mcu.8051.efm8.bb1.efm8bb10f8g-a-qsop24" version="4.0.0" contextId="%DEFAULT%">
  <mode name="DefaultMode">
    <property object="CROSSBAR0" propertyId="xbar0.spi0.clockdata" value="Enabled"/>
    <property object="DefaultMode" propertyId="mode.diagramLocation" value="100, 100"/>
    <property object="INTERRUPT_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="INTERRUPT_0" propertyId="interrupt.interruptenable.enableallinterrupts"
value="Enabled"/>
    <property object="P0.0" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.0" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P0.2" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.2" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P0.3" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.3" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P0.4" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.4" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P1.1" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
```

```
    <property object="P1.1" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P1.2" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P1.2" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="PBCFG_0" propertyId="pbcfg.settings.enablecrossbar" value="Enabled"/>
    <property object="SPI_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="SPI_0" propertyId="spi.configuration.clockphase" value="Data sample on second
edge"/>
    <property object="SPI_0" propertyId="spi.configuration.enablemastermode" value="Enable"/>
    <property object="SPI_0" propertyId="spi.configuration.spimode" value="Master"/>
    <property object="SPI_0" propertyId="spi.control.slaveselectmode" value="Slave or master 3-wire
mode"/>
    <property object="SPI_0" propertyId="spi.control.spienable" value="Enabled"/>
    <property object="SPI_0" propertyId="spi.view.view" value="Advanced"/>
    <property object="WDT_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="WDT_0" propertyId="wdt.watchdogcontrol.wdtenable" value="Disable"/>
  </mode>
  <modeTransition>
    <property object="RESET &#x2192; DefaultMode" propertyId="modeTransition.source" value="RESET"/>
    <property object="RESET &#x2192; DefaultMode" propertyId="modeTransition.target"
value="DefaultMode"/>
  </modeTransition>
</device:XMLDevice>
```

Main.c

```
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <SI_EFM8BB1_Register_Enums.h>                  // SFR declarations
#include "InitDevice.h"
// $[Generated Includes]
// [Generated Includes]$

#define DispClock P0_B0
#define DispData P0_B2
#define DispOutEnable P0_B3

#define DecoderA P1_B1
#define DecoderB P1_B2
#define DecoderC P0_B4

#define SW1 P0_B1

#define DICE_SIDES 6
#define PERGES_COUNT 250

// Displays
enum
{
        LED_T1, LED_T2, LED_T3, LED_T4, LED_LEDS
};

// Leds on display
enum
{
        LED_7 = 1,
        LED_1 = 2,
        LED_2 = 4,
        LED_3 = 8,
        LED_6 = 16,
        LED_5 = 32,
        LED_4 = 64,
};

enum
{
        PRESSED, UNPRESSED,
};

uint8_t currentNumber = 0;

// szorgalmi pergés mentesítés
```

```c
uint8_t lastPressStatus = UNPRESSED;
uint8_t pressCount = 0;
uint8_t buttonStatus = UNPRESSED;

//-----------------------------------------------------------------------------
// SiLabs_Startup() Routine
// ----------------------------------------------------------------------------
// This function is called immediately after reset, before the initialization
// code is run in SILABS_STARTUP.A51 (which runs before main() ). This is a
// useful place to disable the watchdog timer, which is enable by default
// and may trigger before main() in some instances.
//-----------------------------------------------------------------------------
void SiLabs_Startup(void)
{
        // $[SiLabs Startup]
        // [SiLabs Startup]$
}

// Enable selected display.
// Tudom hogy most overkill, de hasznos lesz még kövi gyakon.
void select_display(uint8_t display)
{
        switch (display)
        {
        case LED_T1:
                DecoderA = 0;
                DecoderB = 0;
                DecoderC = 0;

                break;
        case LED_T2:
                DecoderA = 1;
                DecoderB = 0;
                DecoderC = 0;

                break;
        case LED_T3:
                DecoderA = 0;
                DecoderB = 1;
                DecoderC = 0;

                break;
        case LED_T4:
                DecoderA = 1;
                DecoderB = 1;
                DecoderC = 0;

                break;
        case LED_LEDS:
                DecoderA = 0;
                DecoderB = 0;
                DecoderC = 1;

                break;

        default:
                break;
        }

        DispOutEnable = 0;
}

void write_to_spi(uint8_t data_to_send)
{
        SPI0CN0_SPIF = 0;
        SPI0DAT = data_to_send;

        // Wait until data is sent out
        while (!SPI0CN0_SPIF)
                ;

        SPI0CN0_SPIF = 1;
}
```

```
uint8_t get_data_from_number(uint8_t num)
{
        switch (num)
        {
        case 0:

                return ~(LED_7);
        case 1:

                return ~(LED_1 | LED_6);
        case 2:

                return ~(LED_1 | LED_7 | LED_6);
        case 3:

                return ~(LED_1 | LED_4 | LED_3 | LED_6);
        case 4:

                return ~(LED_1 | LED_4 | LED_3 | LED_6 | LED_7);
        case 5:

                return ~(LED_1 | LED_2 | LED_4 | LED_3 | LED_5 | LED_6);
        default:
                return ~0;
        }
}

//-------------------------------------------------------------------------
// main() Routine
// ------------------------------------------------------------------------
int main(void)
{
        // Call hardware initialization routine
        enter_DefaultMode_from_RESET();

        // Shift register beállítása
        // Decoder beállítása
        select_display(LED_LEDS);

        // set with invalid
        write_to_spi(get_data_from_number(111));

        while (1)
        {
                // $[Generated Run-time code]
                // [Generated Run-time code]$

                //
                if (!SW1)
                {
                        if (lastPressStatus == UNPRESSED)
                        {
                                lastPressStatus = PRESSED;
                                pressCount = 0;
                        }
                        else
                        {
                                if (pressCount < PERGES_COUNT)
                                {
                                        ++pressCount;
                                }
                                else
                                {
                                        buttonStatus = PRESSED;

                                        // Hide number when pressing
                                        DispOutEnable = 1;
                                }
                        }
                }
                else
                {
                        if (lastPressStatus == PRESSED)
                        {
```

```
                        lastPressStatus = UNPRESSED;
                        pressCount = 0;
                }
                else
                {
                        if (pressCount < PERGES_COUNT)
                        {
                                ++pressCount;
                        }
                        else
                        {
                                buttonStatus = UNPRESSED;

                                // Show number when released
                                DispOutEnable = 0;
                        }
                }
        }

        if (buttonStatus == PRESSED)
        {
                //write to spit
                write_to_spi(get_data_from_number(currentNumber));

                // increment data
                currentNumber = (currentNumber + 1) % DICE_SIDES;
        }
    }
}
```

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

- Igen
- Nem

A program működött:

- Igen
- Nem

Szorgalmi feladat – LED-ek léptetése timerrel (futófény)

Valósítson meg futófényt, amely a kiegészítőpanelen található LED-ek sorszáma szerint növekvő sorrendben be-, majd kikapcsolja a LED-eket. A LED-ek léptetését egy timerrel valósítsa meg, aminek a túlcsordulási rátája 35 Hz legyen.

Valósítson meg irányváltási funkciót is, amennyiben a kiegészítő panelen található SW1-es nyomógomb lenyomásra kerül a futófény a LED-ek sorszáma szerinti csökkenő sorrendben folytatódjon. (Újbóli lenyomás esetén természetesen ismét növekvő sorrendben.). Irányváltás csak a nyomógomb felengedésére történjen!

A program részekre bontott forráskódja (Config, Main.c, Interrupts.c, ha van):

config

```
<?xml version="1.0" encoding="ASCII"?>
<device:XMLDevice xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:device="http://www.silabs.com/ss/hwconfig/document/device.ecore" name="EFM8BB10F8G-A-QSOP24"
partId="mcu.8051.efm8.bb1.efm8bb10f8g-a-qsop24" version="4.0.0" contextId="%DEFAULT%">
  <mode name="DefaultMode">
```

```
    <property object="CROSSBAR0" propertyId="xbar0.spi0.clockdata" value="Enabled"/>
    <property object="DefaultMode" propertyId="mode.diagramLocation" value="100, 100"/>
    <property object="INTERRUPT_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="INTERRUPT_0" propertyId="interrupt.interruptenable.enableallinterrupts"
value="Enabled"/>
    <property object="INTERRUPT_0" propertyId="interrupt.interruptenable.enabletimer2interrupt"
value="Enabled"/>
    <property object="P0.0" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.0" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P0.2" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.2" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P0.3" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.3" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P0.4" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P0.4" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P1.1" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P1.1" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="P1.2" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P1.2" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="PBCFG_0" propertyId="pbcfg.settings.enablecrossbar" value="Enabled"/>
    <property object="SPI_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="SPI_0" propertyId="spi.configuration.clockphase" value="Data sample on second
edge"/>
    <property object="SPI_0" propertyId="spi.configuration.enablemastermode" value="Enable"/>
    <property object="SPI_0" propertyId="spi.configuration.spimode" value="Master"/>
    <property object="SPI_0" propertyId="spi.control.slaveselectmode" value="Slave or master 3-wire
mode"/>
    <property object="SPI_0" propertyId="spi.control.spienable" value="Enabled"/>
    <property object="SPI_0" propertyId="spi.view.view" value="Advanced"/>
    <property object="TIMER01_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="TIMER16_2" propertyId="ABPeripheral.included" value="true"/>
    <property object="TIMER16_2" propertyId="timer16.control.runcontrol" value="Start"/>
    <property object="TIMER16_2" propertyId="timer16.control.timerrunningstate" value="Timer is
Running"/>
    <property object="TIMER16_2" propertyId="timer16.initandreloadvalue.targetoverflowfrequency"
value="10"/>
    <property object="TIMER16_2" propertyId="timer16.initandreloadvalue.timerreloadvalue"
value="40015"/>
    <property object="TIMER16_2" propertyId="timer16.reloadhighbyte.reloadhighbyte" value="156"/>
    <property object="TIMER16_2" propertyId="timer16.reloadlowbyte.reloadlowbyte" value="79"/>
    <property object="TIMER16_3" propertyId="ABPeripheral.included" value="true"/>
    <property object="TIMER_SETUP_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="WDT_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="WDT_0" propertyId="wdt.watchdogcontrol.wdtenable" value="Disable"/>
  </mode>
  <modeTransition>
    <property object="RESET &#x2192; DefaultMode" propertyId="modeTransition.source" value="RESET"/>
    <property object="RESET &#x2192; DefaultMode" propertyId="modeTransition.target"
value="DefaultMode"/>
  </modeTransition>
</device:XMLDevice>
```

Main.c

```c
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <SI_EFM8BB1_Register_Enums.h>                  // SFR declarations
#include "InitDevice.h"
// $[Generated Includes]
// [Generated Includes]$

#define DispClock P0_B0
#define DispData P0_B2
#define DispOutEnable P0_B3

#define DecoderA P1_B1
#define DecoderB P1_B2
#define DecoderC P0_B4

#define SW1 P0_B1
```

```
#define LED_COUNT 7

enum
{
        LED_T1, LED_T2, LED_T3, LED_T4, LED_LEDS
};

enum
{
        LED_7 = 1,
        LED_1 = 2,
        LED_2 = 4,
        LED_3 = 8,
        LED_6 = 16,
        LED_5 = 32,
        LED_4 = 64,
};

enum
{
        BUTTON_PRESSED, BUTTON_RELEASED
};

enum
{
        DIRECITON_RIGHT, DIRECTION_LEFT,
};

enum
{
        INCREMENT_HANDLED, INCREMENT_UNHANDLED,
};

uint8_t currentNumber = 0;
uint8_t currentDirection = DIRECITON_RIGHT;
uint8_t currentButtonState = BUTTON_RELEASED;
volatile incrementHandleStatus = INCREMENT_HANDLED;

//-----------------------------------------------------------------------------
// SiLabs_Startup() Routine
// ----------------------------------------------------------------------------
// This function is called immediately after reset, before the initialization
// code is run in SILABS_STARTUP.A51 (which runs before main() ). This is a
// useful place to disable the watchdog timer, which is enable by default
// and may trigger before main() in some instances.
//-----------------------------------------------------------------------------
void SiLabs_Startup(void)
{
        // $[SiLabs Startup]
        // [SiLabs Startup]$
}

// Enable selected display.
// Tudom hogy most overkill, de hasznos lesz még kövi gyakon.
void select_display(uint8_t display)
{
        switch (display)
        {
        case LED_T1:
                DecoderA = 0;
                DecoderB = 0;
                DecoderC = 0;

                break;
        case LED_T2:
                DecoderA = 1;
                DecoderB = 0;
                DecoderC = 0;

                break;
        case LED_T3:
                DecoderA = 0;
                DecoderB = 1;
                DecoderC = 0;
```

```
                break;
        case LED_T4:
                DecoderA = 1;
                DecoderB = 1;
                DecoderC = 0;

                break;
        case LED_LEDS:
                DecoderA = 0;
                DecoderB = 0;
                DecoderC = 1;

                break;

        default:
                break;
        }

        DispOutEnable = 0;
}

void write_to_spi(uint8_t data_to_send)
{
        SPI0CN0_SPIF = 0;
        SPI0DAT = data_to_send;

        // Wait until data is sent out
        while (!SPI0CN0_SPIF);

        SPI0CN0_SPIF = 0;
}

uint8_t get_data_from_number(uint8_t num)
{
        switch (num)
        {
        case 0:

                return ~LED_1;
        case 1:

                return ~LED_2;
        case 2:

                return ~LED_3;
        case 3:

                return ~LED_4;
        case 4:

                return ~LED_5;
        case 5:

                return ~LED_6;
        case 6:

                return ~LED_7;
        default:
                return 0;
        }
}

//-----------------------------------------------------------------------------
// main() Routine
// ----------------------------------------------------------------------------
int main(void)
{
        // Call hardware initialization routine
        enter_DefaultMode_from_RESET();

        select_display(LED_LEDS);

        while (1)
```

```
        {
                // $[Generated Run-time code]
                // [Generated Run-time code]$

                DispOutEnable = 1;

                //write to spit
                write_to_spi(get_data_from_number(currentNumber));

                //send signal to bit shifter.
                DispOutEnable = 0;

                if (incrementHandleStatus == INCREMENT_UNHANDLED)
                {
                        // increment data
                        if (currentDirection == DIRECITON_RIGHT)
                        {
                                currentNumber = (currentNumber + 1) % LED_COUNT;
                        }
                        else
                        {
                                if (currentNumber == 0)
                                {
                                        currentNumber = LED_COUNT;
                                }

                                currentNumber -= 1;
                        }

                        incrementHandleStatus = INCREMENT_HANDLED;
                }

                if (currentButtonState == BUTTON_PRESSED && SW1)
                {
                        currentButtonState = BUTTON_RELEASED;
                        currentDirection = currentDirection == DIRECITON_RIGHT
                                        ? DIRECTION_LEFT
                                        : DIRECITON_RIGHT;
                }

                if (currentButtonState == BUTTON_RELEASED && !SW1)
                {
                        currentButtonState = BUTTON_PRESSED;
                }
        }
}
```

Interrupts.c

```
// USER INCLUDES
#include <SI_EFM8BB1_Register_Enums.h>

enum {
  INCREMENT_HANDLED,
  INCREMENT_UNHANDLED,
};

extern volatile incrementHandleStatus;

//-----------------------------------------------------------------------------
// TIMER2_ISR
//-----------------------------------------------------------------------------
//
// TIMER2 ISR Content goes here. Remember to clear flag bits:
// TMR2CN0::TF2H (Timer # High Byte Overflow Flag)
// TMR2CN0::TF2L (Timer # Low Byte Overflow Flag)
//
//-----------------------------------------------------------------------------
SI_INTERRUPT (TIMER2_ISR, TIMER2_IRQn)
{
```

```
  TMR2CN0_TF2H = 0;

  incrementHandleStatus = INCREMENT_UNHANDLED;
}
```

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

- Igen
- Nem

A program működött:

- Igen
- Nem

Megjegyzések

A második feladatnál az volt a kérés, hogy rakjak be szoftveres pergésmentesítést. Egy pergésmentesítést alkalmaztam 8 bites túlcsordulással (ez túl kicsi valós pergések megfogására?).

A szorgalmi feladatban 35 Hz helyett kevesebbet alkalmaztam, mivel túl gyors volt az érdembeli értékeléshez.

Légyszi ne húzzatok le a `select_display` függvény miatt. Büszke vagyok rá.