

Kijelző vezérlése shift regiszterrel

MICLAB-o8

Név: Stefán Kornél

Dátum: 2024. 10. 28 18:00

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 – Tetszőleges 4 jegyű szám megjelenítése a kijelzőn

Készítsen egy programot, ami egy változóban tárolt 4 jegyű számot (1234) jelenít meg a kijelzőn.

A 0..9 számértékekhez tartozó byte-okat egy tömbben tárolja le. (Kövesse az órai ppt-t és használja a MicLab-utmutato.pdf-ben lévő kapcsolódó részeket.)

A shift register egyszerre csak az egyik digitet tudja vezérelni, ezért egy elterjedt megoldást kell használni, amiben a digiteket egymás után, váltogatni kell a dekóder IC-vel. Ha ez elegendően nagy sebességgel történik, akkor a szemünk folyamatosnak látja a megjelenítést. Mivel egyszerre csak egy szegmens aktív, ezért helyiértékekre kell bontani a 4 jegyű számot és az adott számjeggyel kell indexelni a számjegyeket tartalmazó tömböt, majd ezt egyenként beírni a shift regiszterbe. Ügyeljen a szellemképmentes megjelenítésre.

Először kiválasztjuk az első digitet, az SPI segítségével a shift regiszteren keresztül kiadjuk rá az adott szám karakterhez tartozó byte-ot, majd váltunk a következő digitre és így tovább. A digitek váltogatása között 1 ms idő teljen el (timer overflow flag figyelése polling módszerrel). A számjegyek kiírását a while függvényben ne 4-szer lemásolva oldja meg, hanem egy ciklusváltozó segítségével.

Képernyőkép a konfigurációról (Pin configuration report, használt perifériák beállításai képmetszővel):

Pin Configuration Report

Part: EFM8BB10F8G-A-QSOP24

Package: 24-pin QSOP

DefaultMode

Pin #	Pin Name	IO Mode	Signals
2	P0.2	Digital Push-Pull Output	SPI0_MOSI
3	P0.1	Digital OpenDrain I/O	SPI0_MISO
4	P0.0	Digital Push-Pull Output	SPI0_SCK
8	P2.0	Digital OpenDrain I/O	
9	P1.7	Digital OpenDrain I/O	
10	P1.6	Digital OpenDrain I/O	
11	P1.5	Digital OpenDrain I/O	
12	P2.1	Digital OpenDrain I/O	
14	P1.4	Digital OpenDrain I/O	
15	P1.3	Digital OpenDrain I/O	
16	P1.2	Digital Push-Pull Output	
17	P1.1	Digital Push-Pull Output	
18	P1.0	Digital OpenDrain I/O	
19	P0.7	Digital OpenDrain I/O	
20	P0.6	Digital OpenDrain I/O	
21	P0.5	Digital OpenDrain I/O	
22	P0.4	Digital Push-Pull Output	
23	P0.3	Digital Push-Pull Output	

Analog

☐ ADC 0

☐ Comparator 0

☐ Comparator 1

☐ Voltage Reference

Clocking

Communications

☐ SMBus 0

☒ SPI 0

☐ UART 0

Core

☐ External Interrupts

☒ Interrupts

☐ Reset Sources

Other

Power

Timers

☐ PCA

☒ Timers

☒ Watchdog Timer

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

Config

```
<?xml version="1.0" encoding="ASCII"?>
<device:XMLDevice xml: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-qso24" 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="TIMER01_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="TIMER01_0"
propertyId="timer01.timer0mode2:8bitcountertimerwithautoreload.targetoverflowfrequency" value="1000"/>
    <property object="TIMER01_0"
propertyId="timer01.timer0mode2:8bitcountertimerwithautoreload.timerreloadvalue" value="1"/>
    <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"/>
  </mode>
</device:XMLDevice>
```

```

    <property object="TIMER16_2" propertyId="timer16.initandreloadvalue.targetoverflowfrequency"
value="1000"/>
    <property object="TIMER16_2" propertyId="timer16.initandreloadvalue.timerreloadvalue"
value="65281"/>
    <property object="TIMER16_2" propertyId="timer16.reloadhighbyte.reloadhighbyte" value="255"/>
    <property object="TIMER16_2" propertyId="timer16.reloadlowbyte.reloadlowbyte" value="1"/>
    <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

```

//-----
// 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
};

#define DIGIT_COUNT 4
#define DIGITS 10

// Digits on display
enum
{
    DIGIT_0 = ~0x3f,
    DIGIT_1 = ~0x06,
    DIGIT_2 = ~0x5b,
    DIGIT_3 = ~0x4f,
    DIGIT_4 = ~0x66,
    DIGIT_5 = ~0x6D,
    DIGIT_6 = ~0x7D,
    DIGIT_7 = ~0x07,
    DIGIT_8 = ~0x7F,
    DIGIT_9 = ~0x6F,
};

uint8_t digit_values[DIGITS] = {
    DIGIT_0, DIGIT_1, DIGIT_2, DIGIT_3, DIGIT_4,
    DIGIT_5, DIGIT_6, DIGIT_7, DIGIT_8, DIGIT_9
};

// Ez egyszerre a számjegy és a kijelző.
uint8_t current_digit = LED_T1;

char digits[DIGIT_COUNT] = { 1, 2, 3, 4 };
//char digits[DIGIT_COUNT] = {5, 6, 7, 8};
//char digits[DIGIT_COUNT] = {0, 0, 9, 9};

```

```

//-----
// 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.
// T4 shifted to T1 because... reasons.
static void select_display(uint8_t display)
{
    switch (display)
    {
        case LED_T1:
            DecoderA = 1;
            DecoderB = 1;
            DecoderC = 0;

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

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

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

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

            break;
        default:
            break;
    }
}

// Ez lehetne egy valós decoder, de
// az eltárolt értékek gyorsabbak.
static uint8_t digit_decoder(char digit)
{
    if (digit >= DIGITS)
    {
        return 0;
    }

    return digit_values[digit];
}

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)
{
    uint8_t digit;

    // Call hardware initialization routine
    enter_DefaultMode_from_RESET();

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

        if (TMR2CN0_TF2H)
        {
            // disable leds
            DispOutEnable = 1;

            select_display(current_digit);

            //write to spi
            digit = digit_decoder(digits[current_digit]);
            write_to_spi(digit);

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

            current_digit = (current_digit + 1) % DIGIT_COUNT;
            TMR2CN0_TF2H = 0;
        }
    }
}

```

Interrupts.c nincs.

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

- Igen
- Nem

A program működött:

- Igen
- Nem

2. feladat – ADC kód megjelenítése a kijelzőn

Mérje folyamatosan az ADC-vel polling módban a kiegészítő panelen lévő potenciométert és a mért ADC kódot jelenítse meg a kijelzőn. Az ADC kódját a while(1) ciklusban a megfelelő helyen írja meg.

Képernyőkép a konfigurációról (Pin configuration report, használt perifériák beállításai képmetszővel):

Pin Configuration Report

Part: EFM8BB10F8G-A-QSOP24

Package: 24-pin QSOP

DefaultMode

Pin #	Pin Name	IO Mode	Signals
2	P0.2	Digital Push-Pull Output	SPI0_MOSI
3	P0.1	Digital OpenDrain I/O	SPI0_MISO
4	P0.0	Digital Push-Pull Output	SPI0_SCK
8	P2.0	Digital OpenDrain I/O	
9	P1.7	Analog I/O	ADC_IN
10	P1.6	Digital OpenDrain I/O	
11	P1.5	Digital OpenDrain I/O	
12	P2.1	Digital OpenDrain I/O	
14	P1.4	Digital OpenDrain I/O	
15	P1.3	Digital OpenDrain I/O	
16	P1.2	Digital Push-Pull Output	
17	P1.1	Digital Push-Pull Output	
18	P1.0	Digital OpenDrain I/O	
19	P0.7	Digital OpenDrain I/O	
20	P0.6	Digital OpenDrain I/O	
21	P0.5	Digital OpenDrain I/O	
22	P0.4	Digital Push-Pull Output	
23	P0.3	Digital Push-Pull Output	

DefaultMode Peripherals

Analog

☒ ADC 0☐ Comparator 0☐ Comparator 1☐ Voltage Reference

Clocking

Communications

☐ SMBus 0☒ SPI 0☐ UART 0

Core

☐ External Interrupts☒ Interrupts☐ Reset Sources

Other

Power

Timers

☐ PCA☒ Timers☒ Watchdog Timer

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.bb10f8g-a-qso24" version="4.0.0" contextId="%DEFAULT%">
  <mode name="DefaultMode">
    <property object="ADC_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="ADC_0" propertyId="adc.configuration.sarclckdivider" value="1"/>
    <property object="ADC_0" propertyId="adc.control.enableadc" value="Enabled"/>
    <property object="ADC_0" propertyId="adc.multiplexerselection.positiveinputselection"
value="ADC0.15 (P1.7)"/>
    <property object="ADC_0" propertyId="adc.view.view" value="Advanced"/>
    <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="P1.7" propertyId="ports.settings.inputmode" value="Analog"/>
    <property object="P1.7" propertyId="ports.settings.iomode" value="Analog I/O"/>
    <property object="P1.7" propertyId="ports.settings.skip" value="Skipped"/>
    <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="TIMER01_0"
propertyId="timer01.timer0mode2:8bitcountertimerwithautoreload.targetoverflowfrequency" value="1000"/>
        <property object="TIMER01_0"
propertyId="timer01.timer0mode2:8bitcountertimerwithautoreload.timerreloadvalue" value="1"/>
        <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="1000"/>
        <property object="TIMER16_2" propertyId="timer16.initandreloadvalue.timerreloadvalue"
value="65281"/>
        <property object="TIMER16_2" propertyId="timer16.reloadhighbyte.reloadhighbyte" value="255"/>
        <property object="TIMER16_2" propertyId="timer16.reloadlowbyte.reloadlowbyte" value="1"/>
        <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

```

//-----
// 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 ADC_RESOLUTION 1024.0f
#define REF_VOLTAGE_MV 3300u

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

#define DIGIT_COUNT 4
#define DIGITS 10

// Digits on display
enum
{
    DIGIT_0 = ~0x3f,
    DIGIT_1 = ~0x06,
    DIGIT_2 = ~0x5b,
    DIGIT_3 = ~0x4f,
    DIGIT_4 = ~0x66,
    DIGIT_5 = ~0x6D,
    DIGIT_6 = ~0x7D,
    DIGIT_7 = ~0x07,
    DIGIT_8 = ~0x7F,
    DIGIT_9 = ~0x6F,
};

uint8_t digit_values[DIGITS] = {

```

```

    DIGIT_0, DIGIT_1, DIGIT_2, DIGIT_3, DIGIT_4,
    DIGIT_5, DIGIT_6, DIGIT_7, DIGIT_8, DIGIT_9
};

enum
{
    PROGRAM_READING, PROGRAM_DISPLAYING,
};

// Ez egyszerre a számjegy és a kijelző.
uint8_t current_digit = LED_T1;

char digits[DIGIT_COUNT] = { 1, 2, 3, 4 };

uint8_t program_state = PROGRAM_READING;

//-----
// 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.
// T4 shifted to T1 because... reasons.
static void select_display(uint8_t display)
{
    switch (display)
    {
        case LED_T1:
            DecoderA = 1;
            DecoderB = 1;
            DecoderC = 0;

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

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

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

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

            break;
        default:
            break;
    }
}

// Ez lehetne egy valós decoder, de
// az eltárolt értékek gyorsabbak.

```



```

uint8_t digit_decoder(char digit)
{
    if (digit >= DIGITS)
    {
        return 0;
    }

    return digit_values[digit];
}

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;
}

uint16_t read_adc(void)
{
    ADC0CN0_ADINT = 0;
    ADC0CN0_ADBUSY = 1;
    while (!ADC0CN0_ADINT)
        ;

    return ADC0;
}

//-----
// main() Routine
// -----
int main(void)
{
    uint8_t digit;
    uint8_t display_counter;

    // Call hardware initialization routine
    enter_DefaultMode_from_RESET();

    select_display(LED_LEDS);

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

        if (program_state == PROGRAM_READING)
        {
            uint16_t adc_value = read_adc();
            // Simplified (no approximation) voltage divider equation
            uint16_t voltage_mv = adc_value * (REF_VOLTAGE_MV / ADC_RESOLUTION);
            char digit;
            uint8_t i;

            for (i = 1; i <= DIGIT_COUNT; i++)
            {
                digit = voltage_mv % DIGITS;
                voltage_mv = voltage_mv / DIGITS;

                digits[DIGIT_COUNT - i] = digit;
            }

            program_state = PROGRAM_DISPLAYING;
            display_counter = 0;
        }
        else if (program_state == PROGRAM_DISPLAYING)
        {
            // We show 4 digits, then on the fifth
            // step we disable the light. Anti ghosting.
            for (current_digit = 0; current_digit < 5; ++current_digit)

```

```

{
    while (1)
    {
        if (TMR2CN0_TF2H)
        {
            TMR2CN0_TF2H = 0;

            // disable leds
            DispOutEnable = 1;

            if (current_digit != 4)
            {
                select_display(current_digit);

                digit = digit_decoder(digits[current_digit]);
                write_to_spi(digit);

                DispOutEnable = 0;
            }

            break;
        }
    }

    if (++display_counter > 63) {
        program_state = PROGRAM_READING;
    }
}
}
}

```

Interrupts.c nincs.

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

- Igen
- Nem

A program működött:

- Igen
- Nem

Megjegyzések

Az én kódom nem ADC kódot, hanem millivoltot ír ki, de ez könnyen kikapcsolható (Bandi azt mondta ez oké).

A mért értéket 64 alkalommal jelenítem meg, hogy ne frissüljön olyan sűrűn az érték, hogy ne legyen leolvasható. ($mérés + 64 \cdot 4 \cdot 1ms \approx 300ms$)

Szellemkép mentesség írásbeli véde: A kód egyszer mérést végez, majd mind a négy digitet megjeleníti, amit mért. Amikor mérés történik, akkor az összes led fekete, így nem lehet szellemképes.