Kijelző vezérlése shift regiszterrel

MICLAB-08

Név: Pilter Zsófia, Vad Avar

Dátum: 2024.10.28

Mérőhely: 1 jobb és 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):

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="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="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:

```
//=========================================================
// src/feladat08_01_main.c: generated by Hardware Configurator
//
// This file will be updated when saving a document.
// leave the sections inside the "$[...]" comment tags alone
// or they will be overwritten!!
//=========================================================

//-------------------------------------------------------------------------
// 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 DECODER_A P1_B1
#define DECODER_B P1_B2
#define DECODER_C P0_B4

#define NUMBER_OF_DIGITS 10
#define NUMBER_OF_SEGMENTS 4

#define NUM_0 ~0x3f
#define NUM_1 ~0x06
#define NUM_2 ~0x5b
#define NUM_3 ~0x4f
```

```
#define NUM_4 ~0x66
#define NUM_5 ~0x6D
#define NUM_6 ~0x7D
#define NUM_7 ~0x07
#define NUM_8 ~0x7F
#define NUM_9 ~0x6F

uint8_t digits[] =
	{
		NUM_0,
		NUM_1,
		NUM_2,
		NUM_3,
		NUM_4,
		NUM_5,
		NUM_6,
		NUM_7,
		NUM_8,
		NUM_9
	};

uint8_t segments[NUMBER_OF_SEGMENTS];

void activate_display(uint8_t counter)
{
	DECODER_A = counter & (0x01 << 0);
	DECODER_B = counter & (0x01 << 1);
	DECODER_C = counter & (0x01 << 2);


}

void transmit_to_spi(uint8_t value)
{
	DISPOUTENABLE = 0;
	SPI0DAT = value;

	while (!SPI0CN0_SPIF);
	SPI0CN0_SPIF = 0;
	DISPOUTENABLE = 1;
	DISPOUTENABLE = 0;
}

void display_number(uint16_t number)
{
	segments[0] = digits[(number / 1000) % 10];
	segments[1] = digits[(number / 100) % 10];
	segments[2] = digits[(number / 10) % 10];
	segments[3] = digits[number % 10];
}

//-------------------------------------------------------------------------
// 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)
{
	// $[SiLabs Startup]
	// [SiLabs Startup]$
}
uint16_t number = 1234U;
uint8_t inc = 0;
//-----------------------------------------------------------------------------
// main() Routine
// ----------------------------------------------------------------------------
int main(void)
{
	// Call hardware initialization routine
	enter_DefaultMode_from_RESET();

	DECODER_A = 0;
	DECODER_B = 0;
	DECODER_C = 0;
	display_number(number);
	while (1)
	{
		// $[Generated Run-time code]
		// [Generated Run-time code]$


		for (inc = 0; inc < NUMBER_OF_SEGMENTS; inc++)
		{
			//DISPOUTENABLE = 0;
			activate_display(inc);

			transmit_to_spi(0xFF);
			transmit_to_spi(segments[inc]);

			while(!TMR2CN0_TF2H);
			TMR2CN0_TF2H = 0;

			//DISPOUTENABLE = 1;

		}
	}
}
```

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):

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="ADC_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="ADC_0" propertyId="adc.configuration.gaincontrol" value="1x
gain"/>
    <property object="ADC_0" propertyId="adc.configuration.sarclockdivider"
value="0"/>
    <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="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="VREF_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="VREF_0" propertyId="vref.hidden.voltagereferenceselect"
value="VDD pin"/>
    <property object="VREF_0"
propertyId="vref.voltagereferencecontrol.selectvoltagereference"
value="Unregulated VDD"/>
    <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:

```
//=========================================================
// src/feladat08_01_main.c: generated by Hardware Configurator
//
// This file will be updated when saving a document.
// leave the sections inside the "$[...]" comment tags alone
// or they will be overwritten!!
//=========================================================

//-------------------------------------------------------------------------------
// 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 ADC_MAX 1024.0f
#define VDD 3300.0F
```

```
#define DECODER_A P1_B1
#define DECODER_B P1_B2
#define DECODER_C P0_B4

#define NUMBER_OF_DIGITS 10
#define NUMBER_OF_SEGMENTS 4

#define NUM_0 ~0x3f
#define NUM_1 ~0x06
#define NUM_2 ~0x5b
#define NUM_3 ~0x4f
#define NUM_4 ~0x66
#define NUM_5 ~0x6D
#define NUM_6 ~0x7D
#define NUM_7 ~0x07
#define NUM_8 ~0x7F
#define NUM_9 ~0x6F

uint8_t digits[] =
	{
		NUM_0,
		NUM_1,
		NUM_2,
		NUM_3,
		NUM_4,
		NUM_5,
		NUM_6,
		NUM_7,
		NUM_8,
		NUM_9
	};

uint8_t segments[NUMBER_OF_SEGMENTS];

void activate_display(uint8_t counter)
{
	DECODER_A = counter & (0x01 << 0);
	DECODER_B = counter & (0x01 << 1);
	DECODER_C = counter & (0x01 << 2);


}

void transmit_to_spi(uint8_t value)
{
	DISPOUTENABLE = 0;
	SPI0DAT = value;

	while (!SPI0CN0_SPIF);
	SPI0CN0_SPIF = 0;
	DISPOUTENABLE = 1;
	DISPOUTENABLE = 0;
}

void display_number(uint16_t number)
{
	segments[0] = digits[(number / 1000) % 10];
	segments[1] = digits[(number / 100) % 10];
	segments[2] = digits[(number / 10) % 10];
```

```
	segments[3] = digits[number % 10];
}

uint16_t ADCconv(void)
{
  uint16_t adcData;
  ADC0CN0_ADINT = 0;
  ADC0CN0_ADBUSY = 1;
  // clear flag
  // start conversion
  while (!ADC0CN0_ADINT);  // wait for end of conversion
  adcData = ADC0;
  return adcData;
}


//-----------------------------------------------------------------------------
// 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)
{
	// $[SiLabs Startup]
	// [SiLabs Startup]$
}

uint8_t inc = 0;
uint16_t adcValue = 0;
//-----------------------------------------------------------------------------
// main() Routine
// ----------------------------------------------------------------------------
int main(void)
{
	// Call hardware initialization routine
	enter_DefaultMode_from_RESET();


	DECODER_A = 0;
	DECODER_B = 0;
	DECODER_C = 0;

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

		adcValue = ADCconv();
		display_number(adcValue);

		for (inc = 0; inc < NUMBER_OF_SEGMENTS; inc++)
		{
			//DISPOUTENABLE = 0;
			activate_display(inc);

			transmit_to_spi(0xFF);
```

```
                    transmit_to_spi(segments[inc]);

                    //DISPOUTENABLE = 1;

                }
        }
}
```

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

- Igen
- Nem

A program működött:

- Igen
- Nem

Megjegyzések