ADC (Ellenállásmérés, hőmérsékletmérés termisztorral)

MICLAB-06

Név: Stefán Kornél

Dátum: 2024. 10. 14. 18:00

Mérőhely: 7 bal

Bevezetés

Az interrupt használatának megismerése.

Ajánlott irodalom

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.o3.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 o3.

1. feladat – Ellenállás mérése feszültségosztóban

Mérje meg a kiadott feszültségosztó kimenő feszültségét, majd a feszültségosztó képletének felhasználásával határozza meg a GND felőli ellenállás értékét és jelenítse meg az Expressions ablakban. A referencia ellenállás (R_o) értéke 10 k Ω , a feszültségosztó bemenetére kötött feszültség 3,3 V.

Az ADC-t interrupt módban használja, 100 Hz mintavételi rátával, a Vref legyen a VDD (3,3 V). Ügyeljen rá, hogy az ADC belső órajele a lehető legnagyobb frekvenciájú legyen, de maximum 12,50 MHz. A megszakításkezelő függvényben csak az A/D konverter adatának változóba mentése történjen. Az A/D konvertert 10 bites módban használja és legyen balra igazítva. Ügyeljen a megfelelő változóvédelemre.

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

Config

```
<property object="ADC 0" propertyId="adc.accumulatorconfiguration.accumulatorshiftandjustify"</pre>
value="Left justified"/>
   cproperty object="ADC_0" propertyId="adc.multiplexerselection.positiveinputselection"
value="ADC0.11 (P1.3)"/>
   cproperty object="ADC 0" propertyId="adc.view.view" value="Advanced"/>
   propertyId="interrupt.extendedinterruptenable1.enableadc0conversioncompleteinterrupt" value="Enabled"/>
   value="Enabled"/>
   <property object="P1.4" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
<property object="P1.4" propertyId="ports.settings.outputmode" value="Push-pull"/>
   cproperty object="TIMER01_0" propertyId="ABPeripheral.included" value="true"/>
cproperty object="TIMER16_2" propertyId="ABPeripheral.included" value="true"/>
cproperty object="TIMER16_2" propertyId="timer16.control.runcontrol" value="Start"/>
cproperty object="TIMER16_2" propertyId="timer16.control.runcontrol" value="Start"/>
   cyproperty object="TIMER16_2" propertyId="timer16.control.timerrunningstate" value="Timer is
Running"/>
   <property object="TIMER16_2" propertyId="timer16.initandreloadvalue.targetoverflowfrequency"</pre>
value="100"/>
   value="62984"/>
   <property object="TIMER16_2" propertyId="timer16.reloadhighbyte.reloadhighbyte" value="246"/>
<property object="TIMER16_2" propertyId="timer16.reloadlowbyte.reloadlowbyte" value="8"/>
<property object="TIMER16_3" propertyId="ABPeripheral.included" value="true"/>
   cproperty object="TIMER_SETUP_0" propertyId="ABPeripheral.included" value="true"/>
   cproperty object="VREF_0" propertyId="ABPeripheral.included" value="true"/>
cproperty object="VREF_0" propertyId="vref.hidden.voltagereferenceselect" value="VDD pin"/>
   value="Unregulated VDD"/>
   <property object="WDT_0" propertyId="ABPeripheral.included" value="true"/>
   cproperty object="WDT_0" propertyId="wdt.watchdogcontrol.wdtenable" value="Disable"/>
 </mode>
   value="DefaultMode"/>
 </modeTransition>
</device:XMLDevice>
Main
// Includes
//-----
#include <SI_EFM8BB1_Register_Enums.h>
                                              // SFR declarations
#include "InitDevice.h"
// $[Generated Includes]
// [Generated Includes]$
//-----
// 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)
 // $[SiLabs Startup]
 // [SiLabs Startup]$
```

```
}
#define RESOLUTION 65536.0f
#define REFERENCE_VOLTAGE_MV 3300u
#define RESISTOR_BASE 10000u
volatile uint16_t adc_value = 0;
// main() Routine
int main (void)
  // Call hardware initialization routine
  enter_DefaultMode_from_RESET ();
  // V_in
  while (1)
     uint16_t local_adc_value;
     uint16_t thermistor_resistance;
     // save local <a href="mailto:adc">adc</a> value, can be interrupted after.
     IE\_EA = 0;
     local_adc_value = adc_value;
     IE\_EA = 1;
     thermistor resistance = (RESISTOR_BASE / ((RESOLUTION / adc_value) - 1));
 }
}
Interrupts
// src/Interrupts.c: generated by Hardware Configurator
// This file will be regenerated when saving a document.
// leave the sections inside the "\{[\ldots]" comment tags alone
// or they will be overwritten!
// USER INCLUDES
#include <SI_EFM8BB1_Register_Enums.h>
#define CLEAR_FLAG(FLAG) FLAG = 0
extern volatile uint16_t adc_value;
//-----
// ADC0EOC ISR Content goes here. Remember to clear flag bits:
// ADCOCNO::ADINT (Conversion Complete Interrupt Flag)
SI_INTERRUPT (ADC0EOC_ISR, ADC0EOC_IRQn)
 CLEAR_FLAG(ADC0CN0_ADINT);
 adc_value = ADC0;
```

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

Igen

• Nem

A program működött:

- Igen
- Nem

2. feladat – Hőmérséklet mérése termisztorral

Cserélje ki a feszültségosztóban a GND felőli ellenállást a kiadott termisztorra. Módosítsa az előző programot úgy, hogy a termisztor mért ellenállásának felhasználásával a program meg tudja határozni a hőmérsékletet a termisztor egyenletének felhasználásával. A kapott hőmérsékletet váltsa át °C-ba.

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"</pre>
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">
  value="Left justified"/>
  <property object="ADC_0" propertyId="adc.configuration.gaincontrol" value="1x gain"/>
<property object="ADC_0" propertyId="adc.configuration.sarclockdivider" value="1"/>
  value="ADC0.11 (P1.3)"/>
  cproperty object="ADC_0" propertyId="adc.view.view" value="Advanced"/>
  <property object="DefaultMode" propertyId="mode.diagramLocation" value="100, 100"/>
  propertyId="interrupt.extendedinterruptenable1.enableadc0conversioncompleteinterrupt" value="Enabled"/>
  <property object="INTERRUPT_0" propertyId="interrupt.interruptenable.enableallinterrupts"</pre>
value="Enabled"/>
  cproperty object="P1.3" propertyId="ports.settings.inputmode" value="Analog"/>
  cproperty object="TIMER01_0" propertyId="ABPeripheral.included" value="true"/>
  Running"/>
  value="100"/>
  cproperty object="TIMER16_2" propertyId="timer16.initandreloadvalue.timerreloadvalue"
value="62984"/>
  cproperty object="TIMER16_2" propertyId="timer16.reloadhighbyte.reloadhighbyte" value="246"/>
  cyproperty object="TIMER_SETUP_0" propertyId="ABPeripheral.included" value="true"/>
  value="Unregulated VDD"/>
  cproperty object="WDT_0" propertyId="ABPeripheral.included" value="true"/>
  cproperty object="WDT_0" propertyId="wdt.watchdogcontrol.wdtenable" value="Disable"/>
 </mode>
 <modeTransition>
  value="DefaultMode"/>
 </modeTransition>
```

Main

```
// Includes
//----
#include "InitDevice.h"
#include "math.h"
// $[Generated Includes]
// [Generated Includes]$
// 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)
 // $[SiLabs Startup]
 // [SiLabs Startup]$
#define RESOLUTION 65536.0f
#define REFERENCE_VOLTAGE_MV 3300u
#define RESISTOR_BASE 10000u
#define THERMISTOR 25 10000.0f
#define THERMISTOR_B_25_85 3977.0f
#define KELVIN_OFFSET 273.15f
#define THERMISTOR_T_25 (25u + KELVIN_OFFSET)
volatile uint16_t adc_value = 0;
// main() Routine
// -----
int main (void)
  // Call hardware initialization routine
  enter_DefaultMode_from_RESET ();
  while (1)
  {
      // save local adc value, can be interrupted after.
     uint16_t local_adc_value;
      uint16_t thermistor_resistance;
     float temperature_log;
     float temperature;
     IE EA = 0;
      local_adc_value = adc_value;
     IE\_EA = 1;
      thermistor_resistance = (RESISTOR_BASE / ((RESOLUTION / adc_value) - 1));
      temperature_log = log(thermistor_resistance / THERMISTOR_25);
      temperature = 1 / ((1 / THERMISTOR_T_25) + (1 / THERMISTOR_B_25_85) * temperature_log) -
KELVIN_OFFSET;
 }
}
```

Interrupts

```
//========// src/Interrupts.c: generated by Hardware Configurator
```

Képernyőkép a mért hőmérsékletről:

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

- <u>Igen</u>
- Nem

A program működött:

- <u>Igen</u>
- Nem

Szorgalmi feladat – Mérési zaj csökkentése átlagolással

Számolja ki 100 mérés átlagát, majd abból számoljon hőmérsékletet. Ezzel a módszerrel a mérési zaj csökkenthető, ami megfigyelhető, ha összevetünk két egymást követő mérést.

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

Config

```
value="ADC0.11 (P1.3)"/>
    cproperty object="ADC_0" propertyId="adc.view.view" value="Advanced"/>
    cproperty object="DefaultMode" propertyId="mode.diagramLocation" value="100, 100"/>
    propertyId="interrupt.extendedinterruptenable1.enableadc0conversioncompleteinterrupt" value="Enabled"/>
    <property object="INTERRUPT 0" propertyId="interrupt.interruptenable.enableallinterrupts"</pre>
value="Enabled"/>
    cending />
cproperty object="P1.3" propertyId="ports.settings.inputmode" value="Analog"/>
cproperty object="P1.3" propertyId="ports.settings.iomode" value="Analog I/O"/>
cproperty object="P1.3" propertyId="ports.settings.skip" value="Skipped"/>
cproperty object="P1.4" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
cproperty object="P1.4" propertyId="ports.settings.outputmode" value="Push-pull"/>
cproperty object="P1.6" propertyId="pbcfg.settings.enablecrossbar" value="Enabled"/>
    cproperty object="TIMER01_0" propertyId="ABPeripheral.included" value="true"/>
cproperty object="TIMER16_2" propertyId="ABPeripheral.included" value="true"/>
cproperty object="TIMER16_2" propertyId="timer16.control.runcontrol" value="Start"/>
cproperty object="TIMER16_2" propertyId="timer16.control.timerrunningstate" value="Timer is
Running"/>
    cyroperty object="TIMER16_2" propertyId="timer16.initandreloadvalue.targetoverflowfrequency"
value="100"/>
    cproperty object="TIMER16_2" propertyId="timer16.initandreloadvalue.timerreloadvalue"
value="62984"/>
    <property object="TIMER16_2" propertyId="timer16.reloadhighbyte.reloadhighbyte" value="246"/>
    <property object="TIMER_SETUP_0" propertyId="ABPeripheral.included" value="true"/>
    value="Unregulated VDD"/>
    <property object="WDT_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="WDT_0" propertyId="wdt.watchdogcontrol.wdtenable" value="Disable"/>
  </mode>
    value="DefaultMode"/>
  </modeTransition>
</device:XMLDevice>
Main
// Includes
#include <SI_EFM8BB1_Register_Enums.h>
#include "InitDevice.h"
                                                       // SFR declarations
#include "math.h"
// $[Generated Includes]
// [Generated Includes]$
//-----
// 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)
{
 // $[SiLabs Startup]
 // [SiLabs Startup]$
#define RESOLUTION 65536.0f
#define REFERENCE_VOLTAGE_MV 3300u
```

#define RESISTOR_BASE 10000u
#define THERMISTOR_25 10000.0f

```
#define THERMISTOR_B_25_85 3977.0f
#define KELVIN OFFSET 273.15f
#define THERMISTOR_T_25 (25u + KELVIN_OFFSET)
#define MEASUREMENTS 100
#define MEASUREMENTSF 100.0f
  UNREAD.
  READ
};
enum {
 UNCONVERTED,
  CONVERTED,
volatile uint16_t adc_value = 0;
volatile uint8_t adc_values_index = 0;
volatile uint8_t adc_value_read = READ;
uint8_t adc_value_converted = CONVERTED;
float adc_values;
// main() Routine
int main (void)
  // Call hardware initialization routine
  enter_DefaultMode_from_RESET ();
  while (1)
  {
      uint16_t local_adc_value;
      uint16_t thermistor_resistance;
      float Temperature_log;
      float temperature;
      if (adc_value_read == UNREAD)
      {
          IE_EA = 0;
          local_adc_value = adc_value;
          IE\_EA = 1;
          adc_value_read = READ;
          adc_values += adc_value / MEASUREMENTSF;
      }
      if (adc_values_index == 1)
          adc_value_converted = UNCONVERTED;
      if (adc_values_index == 0 && adc_value_converted == UNCONVERTED)
          adc_value_converted = CONVERTED;
          thermistor_resistance = (RESISTOR_BASE / ((RESOLUTION / adc_values) - 1));
          temperature_log = log(thermistor_resistance / THERMISTOR_25);
          temperature = 1 / ((1 / THERMISTOR_T_25) + (1 / THERMISTOR_B_25_85) * temperature_log) -
KELVIN_OFFSET;
          adc_values = 0;
      }
 }
Interrupts
```

```
// src/Interrupts.c: generated by Hardware Configurator
\ensuremath{//} This file will be regenerated when saving a document.
```

```
// leave the sections inside the "$[...]" comment tags alone
// or they will be overwritten!
// USER INCLUDES
#include <SI_EFM8BB1_Register_Enums.h>
#define CLEAR_FLAG(FLAG) FLAG = 0
#define MEASUREMENTS 100
extern volatile uint16_t adc_value;
extern volatile uint8_t adc_values_index;
extern volatile uint8_t adc_value_read;
enum {
 UNREAD,
 READ
};
//-----
// ADCOEOC ISR Content goes here. Remember to clear flag bits:
// ADC0CN0::ADINT (Conversion Complete Interrupt Flag)
SI_INTERRUPT (ADC0EOC_ISR, ADC0EOC_IRQn)
  CLEAR_FLAG(ADC0CN0_ADINT);
  adc_value = ADC0;
 adc_values_index = (adc_values_index + 1) % MEASUREMENTS;
 adc_value_read = UNREAD;
```

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

- <u>Igen</u>
- Nem

A program működött:

- <u>Igen</u>
- Nem

Megjegyzések