**Angelica Semenec**

**012179180**

**Lab 6**

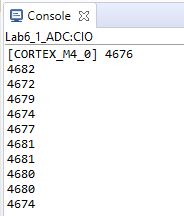
**ADC and temperature sensor on MSP432 MCU**

**Purpose**

The purpose of this lab is to become familiarized with the analog-to-digital converter (ADC) and the temperature sensor mechanism on the MSP432.

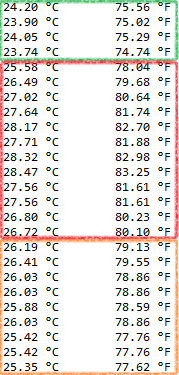
**Exercise 1**

|  |
| --- |
| **main.c** |
| void delay\_ms(uint32\_t count);  uint32\_t t0, t1, cycles;  int main(void)  {  .  .  .  /\* Set up Timer32 \*/  MAP\_Timer32\_initModule(TIMER32\_0\_BASE, TIMER32\_PRESCALER\_1, TIMER32\_32BIT,  TIMER32\_FREE\_RUN\_MODE);  MAP\_Timer32\_startTimer(TIMER32\_0\_BASE, 0);  while(true)  {  // Trigger conversion with software  adc\_done = false;  ADC14\_toggleConversionTrigger();  while (!adc\_done)  {  }  uint32\_t adc\_value = ADC14\_getResult(ADC\_MEM0);  delay\_ms(500);  printf("%u\n", adc\_value);  }  }  .  .  .  /\* Delay number of milliseconds \*/  void delay\_ms(uint32\_t count)  {  t0 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //Get current count  cycles = count \* (MAP\_CS\_getMCLK()/1000); //Calculate number of cycles  do{  t1 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE);  }while((t0 - t1) < cycles); //Check cycle count  return;  } |



**Exercise 2**

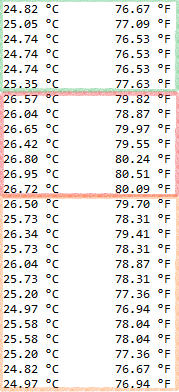
|  |
| --- |
| **main.c** |
| uint32\_t\* CAL\_ADC\_25T30 = (uint32\_t\*)0x002010EC;  uint32\_t\* CAL\_ADC\_25T85 = (uint32\_t\*)0x002010F0;  float temp\_C, temp\_F;  int main(void)  {  .  .  .  float t30 = (float)(\*CAL\_ADC\_25T30);  float t85 = (float)(\*CAL\_ADC\_25T85);  while(true)  {  .  .  .  temp\_C = ((float)adc\_value - t30) \* (85 - 30) / (t85 - t30) + 30;  temp\_F = temp\_C \* 9 / 5 + 32;  printf("%.2f %cC\t%.2f %cF\n", temp\_C, 0xb0, temp\_F, 0xb0);  }  } |



The console output displayed in the image shows the temperatures of the MCU in Farenheit and Celsius displayed at 500ms intervals. The data within the **green box** shows the temperature of the chip with no interaction. In the **red box**, the data points show a sudden increase in temperature after the chip had been touched. In the **orange box**, the temperature is slowly decreasing after the chip is no longer being touched. It doesn’t drop immediately back down to its earlier temperature before interaction (shown in the green box), but takes time to disperse the added heat.

**Exercise 3**

|  |
| --- |
| **main.c** |
| uint32\_t\* CAL\_ADC\_25T30 = (uint32\_t\*)0x002010EC;  uint32\_t\* CAL\_ADC\_25T85 = (uint32\_t\*)0x002010F0;  int temp\_Cx100, temp\_Fx100;  int main(void)  {  .  .  int t30 = (int)(\*CAL\_ADC\_25T30);  int t85 = (int)(\*CAL\_ADC\_25T85);  while(true)  {  .  .  temp\_Cx100 = ((int)adc\_value - t30) \* (85 - 30) \* 100 / (t85 - t30) + 3000;  temp\_Fx100 = temp\_Cx100 \* 9 / 5 + 3200;  printf("%d.%02d %cC\t%d.%02d %cF\n", temp\_Cx100/100, temp\_Cx100%100, 0xb0, temp\_Fx100/100, temp\_Fx100%100, 0xb0);  }  } |

To calculate the temperature without using floating-point arithmetic, integer variables are utilized. To gain a precision of 2 decimal places, the numerator of each summed value is multiplied by 100 (essentially shifting left by 2 decimal places). To display the correct value, the whole number portion is calculated by dividing the resulting temperature value *temp\_Cx100* and *temp\_Fx100* by 100. To get the decimal number portion, the temperature values *temp\_Cx100* and *temp\_Fx100* undergo modulus 100 to calculate the remainder. To accurately express single-digit values as a two-digit value decimal (i.e. 4 represented as 04), the decimal number portion within the *printf()* statement is formatted to 2 digits. The *printf()* statement also formats the output so that it displays both the whole number portion and the decimal number portion together to create a visual representation of the temperature to a precision of 2 decimal places. 

The console output shows the temperature of the MCU taken at 500ms intervals. The **green box** denotes the temperature prior to being touched. The **red box** denotes the temperature while being touched. And the **orange box** denotes the temperature after the touch is removed and the temperature slowly drops back down to its prior temperature.

**Appendix**

**Exercise 1**

|  |
| --- |
| **main.c** |
| // DriverLib Includes  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  // Standard Includes  #include <stdint.h>  #include <stdio.h>  #include <string.h>  volatile bool adc\_done;  void delay\_ms(uint32\_t count);  uint32\_t t0, t1, cycles;  int main(void)  {  // Halt WDT  WDT\_A\_holdTimer();  // Set reference voltage to 2.5 V and enable temperature sensor  REF\_A\_enableReferenceVoltage();  REF\_A\_enableTempSensor();  REF\_A\_setReferenceVoltage(REF\_A\_VREF2\_5V);  // Initializing ADC (MCLK/1/1) with temperature sensor routed  ADC14\_enableModule();  ADC14\_initModule(ADC\_CLOCKSOURCE\_MCLK, ADC\_PREDIVIDER\_1, ADC\_DIVIDER\_1, ADC\_TEMPSENSEMAP);  // Configure ADC Memory for temperature sensor data  ADC14\_configureSingleSampleMode(ADC\_MEM0, false);  ADC14\_configureConversionMemory(ADC\_MEM0, ADC\_VREFPOS\_INTBUF\_VREFNEG\_VSS, ADC\_INPUT\_A22, false);  // Configure the sample/hold time  ADC14\_setSampleHoldTime(ADC\_PULSE\_WIDTH\_192, ADC\_PULSE\_WIDTH\_192);  // Enable sample timer in manual iteration mode and interrupts  ADC14\_enableSampleTimer(ADC\_MANUAL\_ITERATION);  // Enable conversion  ADC14\_enableConversion();  // Enabling Interrupts  ADC14\_enableInterrupt(ADC\_INT0);  Interrupt\_enableInterrupt(INT\_ADC14);  Interrupt\_enableMaster();  /\* Set up Timer32 \*/  MAP\_Timer32\_initModule(TIMER32\_0\_BASE, TIMER32\_PRESCALER\_1, TIMER32\_32BIT,  TIMER32\_FREE\_RUN\_MODE);  MAP\_Timer32\_startTimer(TIMER32\_0\_BASE, 0);  while(true)  {  // Trigger conversion with software  adc\_done = false;  ADC14\_toggleConversionTrigger();  while (!adc\_done)  {  }  uint32\_t adc\_value = ADC14\_getResult(ADC\_MEM0);  delay\_ms(500);  printf("%u\n", adc\_value);  }  }  // This interrupt happens every time a conversion has completed  void ADC14\_IRQHandler(void)  {  uint64\_t status;  status = ADC14\_getEnabledInterruptStatus();  ADC14\_clearInterruptFlag(status);  if(status & ADC\_INT0)  {  adc\_done = true;  }  }  /\* Delay number of milliseconds \*/  void delay\_ms(uint32\_t count)  {  t0 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //Get current count  cycles = count \* (MAP\_CS\_getMCLK()/1000); //Calculate number of cycles  do{  t1 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE);  }while((t0 - t1) < cycles); //Check cycle count  return;  } |

**Exercise 2**

|  |
| --- |
| **main.c** |
| // DriverLib Includes  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  // Standard Includes  #include <stdint.h>  #include <stdio.h>  #include <string.h>  volatile bool adc\_done;  void delay\_ms(uint32\_t count);  uint32\_t t0, t1, cycles;  uint32\_t\* CAL\_ADC\_25T30 = (uint32\_t\*)0x002010EC;  uint32\_t\* CAL\_ADC\_25T85 = (uint32\_t\*)0x002010F0;  float temp\_C, temp\_F;  int main(void)  {  // Halt WDT  WDT\_A\_holdTimer();  // Set reference voltage to 2.5 V and enable temperature sensor  REF\_A\_enableReferenceVoltage();  REF\_A\_enableTempSensor();  REF\_A\_setReferenceVoltage(REF\_A\_VREF2\_5V);  // Initializing ADC (MCLK/1/1) with temperature sensor routed  ADC14\_enableModule();  ADC14\_initModule(ADC\_CLOCKSOURCE\_MCLK, ADC\_PREDIVIDER\_1, ADC\_DIVIDER\_1, ADC\_TEMPSENSEMAP);  // Configure ADC Memory for temperature sensor data  ADC14\_configureSingleSampleMode(ADC\_MEM0, false);  ADC14\_configureConversionMemory(ADC\_MEM0, ADC\_VREFPOS\_INTBUF\_VREFNEG\_VSS, ADC\_INPUT\_A22, false);  // Configure the sample/hold time  ADC14\_setSampleHoldTime(ADC\_PULSE\_WIDTH\_192, ADC\_PULSE\_WIDTH\_192);  // Enable sample timer in manual iteration mode and interrupts  ADC14\_enableSampleTimer(ADC\_MANUAL\_ITERATION);  // Enable conversion  ADC14\_enableConversion();  // Enabling Interrupts  ADC14\_enableInterrupt(ADC\_INT0);  Interrupt\_enableInterrupt(INT\_ADC14);  Interrupt\_enableMaster();  /\* Set up Timer32 \*/  MAP\_Timer32\_initModule(TIMER32\_0\_BASE, TIMER32\_PRESCALER\_1, TIMER32\_32BIT,  TIMER32\_FREE\_RUN\_MODE);  MAP\_Timer32\_startTimer(TIMER32\_0\_BASE, 0);  float t30 = (float)(\*CAL\_ADC\_25T30);  float t85 = (float)(\*CAL\_ADC\_25T85);  while(true)  {  // Trigger conversion with software  adc\_done = false;  ADC14\_toggleConversionTrigger();  while (!adc\_done)  {  }  uint32\_t adc\_value = ADC14\_getResult(ADC\_MEM0);  delay\_ms(500);  temp\_C = ((float)adc\_value - t30) \* (85 - 30) / (t85 - t30) + 30;  temp\_F = temp\_C \* 9 / 5 + 32;  printf("%.2f %cC\t%.2f %cF\n", temp\_C, 0xb0, temp\_F, 0xb0);  }  }  // This interrupt happens every time a conversion has completed  void ADC14\_IRQHandler(void)  {  uint64\_t status;  status = ADC14\_getEnabledInterruptStatus();  ADC14\_clearInterruptFlag(status);  if(status & ADC\_INT0)  {  adc\_done = true;  }  }  /\* Delay number of milliseconds \*/  void delay\_ms(uint32\_t count)  {  t0 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //Get current count  cycles = count \* (MAP\_CS\_getMCLK()/1000); //Calculate number of cycles  do{  t1 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE);  }while((t0 - t1) < cycles); //Check cycle count  return;  } |

**Exercise 3**

|  |
| --- |
| **main.c** |
| // DriverLib Includes  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  // Standard Includes  #include <stdint.h>  #include <stdio.h>  #include <string.h>  volatile bool adc\_done;  void delay\_ms(uint32\_t count);  uint32\_t t0, t1, cycles;  uint32\_t\* CAL\_ADC\_25T30 = (uint32\_t\*)0x002010EC;  uint32\_t\* CAL\_ADC\_25T85 = (uint32\_t\*)0x002010F0;  int temp\_Cx100, temp\_Fx100;  int main(void)  {  // Halt WDT  WDT\_A\_holdTimer();  // Set reference voltage to 2.5 V and enable temperature sensor  REF\_A\_enableReferenceVoltage();  REF\_A\_enableTempSensor();  REF\_A\_setReferenceVoltage(REF\_A\_VREF2\_5V);  // Initializing ADC (MCLK/1/1) with temperature sensor routed  ADC14\_enableModule();  ADC14\_initModule(ADC\_CLOCKSOURCE\_MCLK, ADC\_PREDIVIDER\_1, ADC\_DIVIDER\_1, ADC\_TEMPSENSEMAP);  // Configure ADC Memory for temperature sensor data  ADC14\_configureSingleSampleMode(ADC\_MEM0, false);  ADC14\_configureConversionMemory(ADC\_MEM0, ADC\_VREFPOS\_INTBUF\_VREFNEG\_VSS, ADC\_INPUT\_A22, false);  // Configure the sample/hold time  ADC14\_setSampleHoldTime(ADC\_PULSE\_WIDTH\_192, ADC\_PULSE\_WIDTH\_192);  // Enable sample timer in manual iteration mode and interrupts  ADC14\_enableSampleTimer(ADC\_MANUAL\_ITERATION);  // Enable conversion  ADC14\_enableConversion();  // Enabling Interrupts  ADC14\_enableInterrupt(ADC\_INT0);  Interrupt\_enableInterrupt(INT\_ADC14);  Interrupt\_enableMaster();  /\* Set up Timer32 \*/  MAP\_Timer32\_initModule(TIMER32\_0\_BASE, TIMER32\_PRESCALER\_1, TIMER32\_32BIT,  TIMER32\_FREE\_RUN\_MODE);  MAP\_Timer32\_startTimer(TIMER32\_0\_BASE, 0);  int t30 = (int)(\*CAL\_ADC\_25T30);  int t85 = (int)(\*CAL\_ADC\_25T85);  while(true)  {  // Trigger conversion with software  adc\_done = false;  ADC14\_toggleConversionTrigger();  while (!adc\_done)  {  }  uint32\_t adc\_value = ADC14\_getResult(ADC\_MEM0);  delay\_ms(500);  temp\_Cx100 = ((int)adc\_value - t30) \* (85 - 30) \* 100 / (t85 - t30) + 3000;  temp\_Fx100 = temp\_Cx100 \* 9 / 5 + 3200;  printf("%d.%02d %cC\t%d.%02d %cF\n", temp\_Cx100/100, temp\_Cx100%100, 0xb0, temp\_Fx100/100, temp\_Fx100%100, 0xb0);  }  }  // This interrupt happens every time a conversion has completed  void ADC14\_IRQHandler(void)  {  uint64\_t status;  status = ADC14\_getEnabledInterruptStatus();  ADC14\_clearInterruptFlag(status);  if(status & ADC\_INT0)  {  adc\_done = true;  }  }  /\* Delay number of milliseconds \*/  void delay\_ms(uint32\_t count)  {  t0 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //Get current count  cycles = count \* (MAP\_CS\_getMCLK()/1000); //Calculate number of cycles  do{  t1 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE);  }while((t0 - t1) < cycles); //Check cycle count  return;  } |