**Task 0:**

Execute the supplied code, no submission required.

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/adc.h"

**#define** TARGET\_IS\_BLIZZARD\_RB1

**#include** "driverlib/rom.h"

**int** **main**(**void**)

{

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 64);

ROM\_ADCSequenceConfigure(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 0, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 1, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 2, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

ROM\_ADCSequenceEnable(ADC0\_BASE, 1);

**while**(1)

{

ROM\_ADCIntClear(ADC0\_BASE, 1);

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1);

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false))

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value);

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

}

}

// timer1 int

**void** **Timer1IntHandler**(**void**)

{}

**Task 1:**

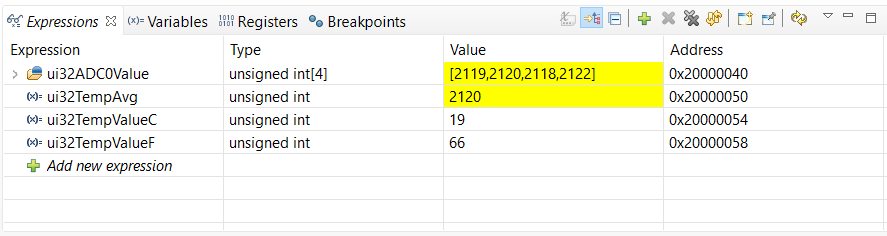
Change the ADC Sequencer to SS3. Turn on the LED at PF2 if the temperature is greater

that 72 degF. Use internal temperature sensor for all SS2 sequence. Display the temperature in

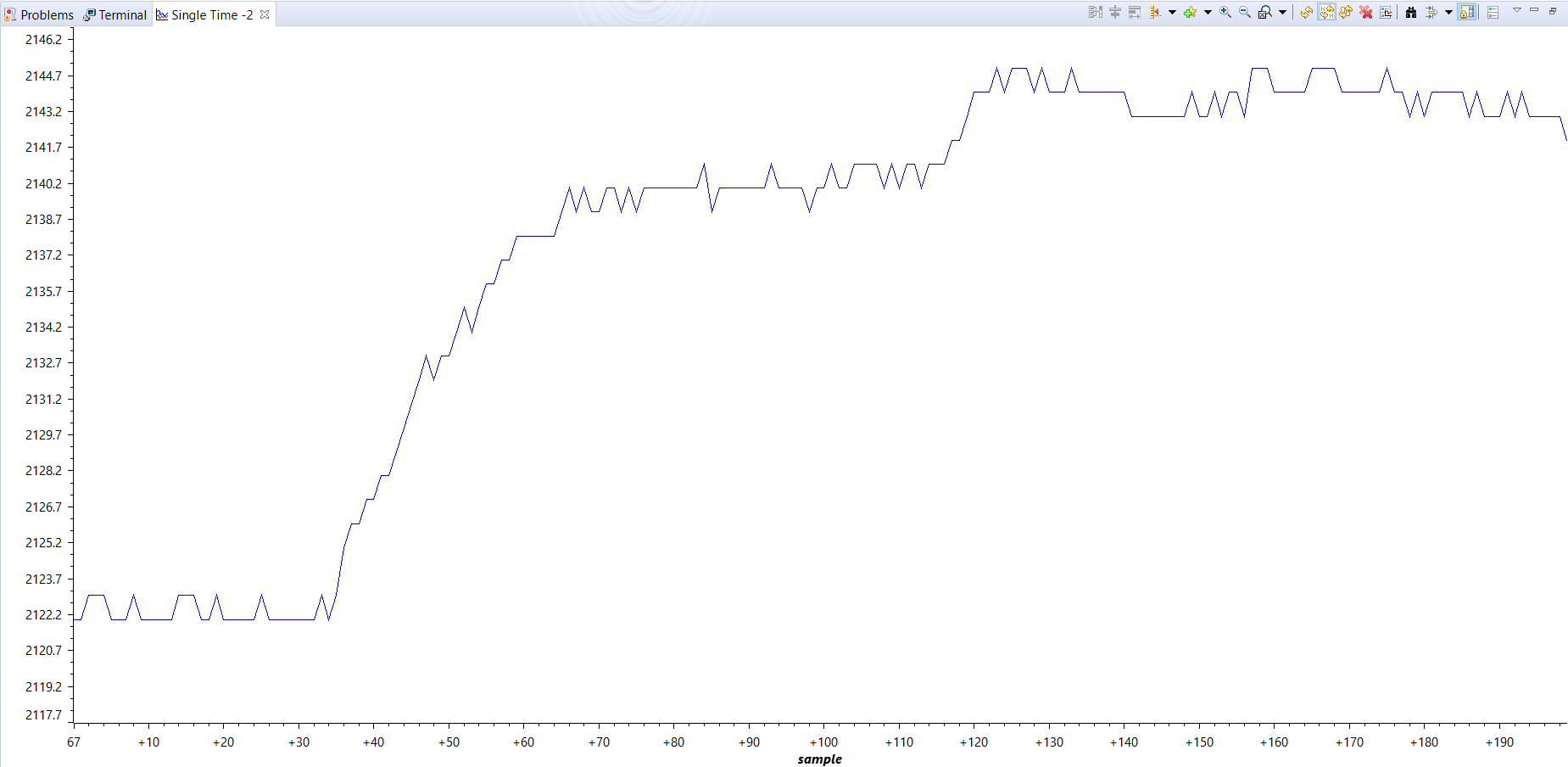
the built-in graph tool.

**For the purpose of this demonstration, I set the LED trigger value to anything greater than 66F. This is so that the LED will turn on.**

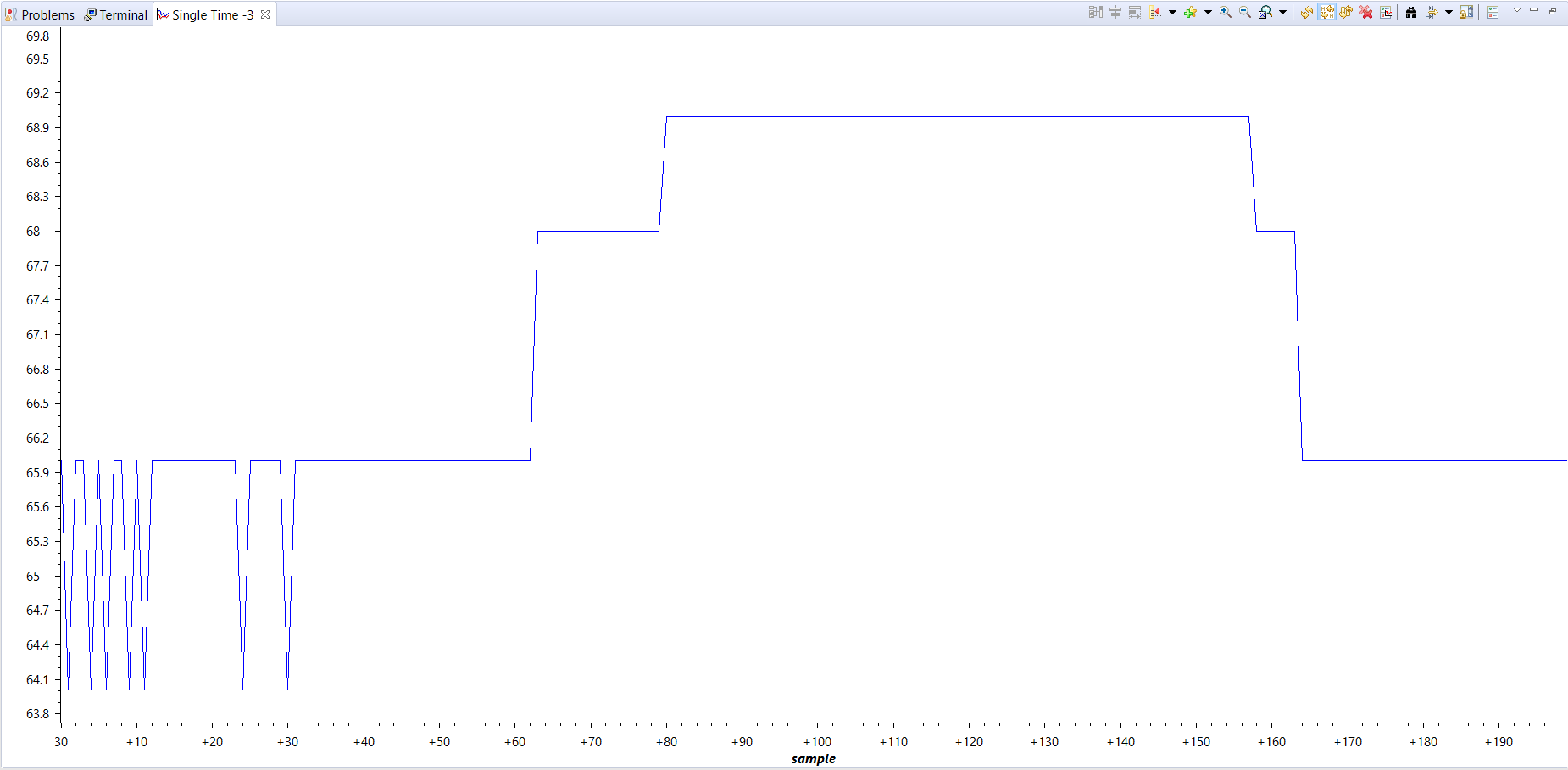
The expressions table when no heat is applied to the CPU:



The graph of the ADC average values polled:



The graph of the ADC values for Fahrenheit polled, notice the spike above 66F, this is when heat is applied to the CPU:



**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/adc.h"

**#define** TARGET\_IS\_BLIZZARD\_RB1

**#include** "driverlib/rom.h"

**int** **main**(**void**)

{

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// ==================================================

// Configure system clock

// CLK = 40MHz = (400MHz PLL / (5 \* 2))

// --------------------------------------------------

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

// enable periphery for GPIO\_F

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);

// GPIO output enable for LEDs

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);

// enable ADC0, set oversampling to 64

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 64);

// configure ADC sequencer to 2 and assign the ADC value types

ROM\_ADCSequenceConfigure(ADC0\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 1, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

ROM\_ADCSequenceEnable(ADC0\_BASE, 2);

**while**(1)

{

// clear ADC interrupt and set processor trigger to sequence 2

ROM\_ADCIntClear(ADC0\_BASE, 2);

ROM\_ADCProcessorTrigger(ADC0\_BASE, 2);

// while busy, keep looping

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 2, false))

{}

// grab ADC data and compute the avg value and temps for F and C

ROM\_ADCSequenceDataGet(ADC0\_BASE, 2, ui32ADC0Value);

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

// turn blue LED if tempF is > 66, else keep it off

**if**(ui32TempValueF > 66)

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0x04);

**else**

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0x00);

}

}

// timer1 int

**void** **Timer1IntHandler**(**void**)

{

}

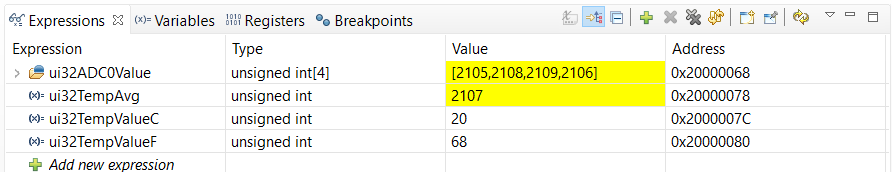
**Task 2:**

Introduce hardware averaging to 32. Using the timer TIMER1A conduct an ADC conversion

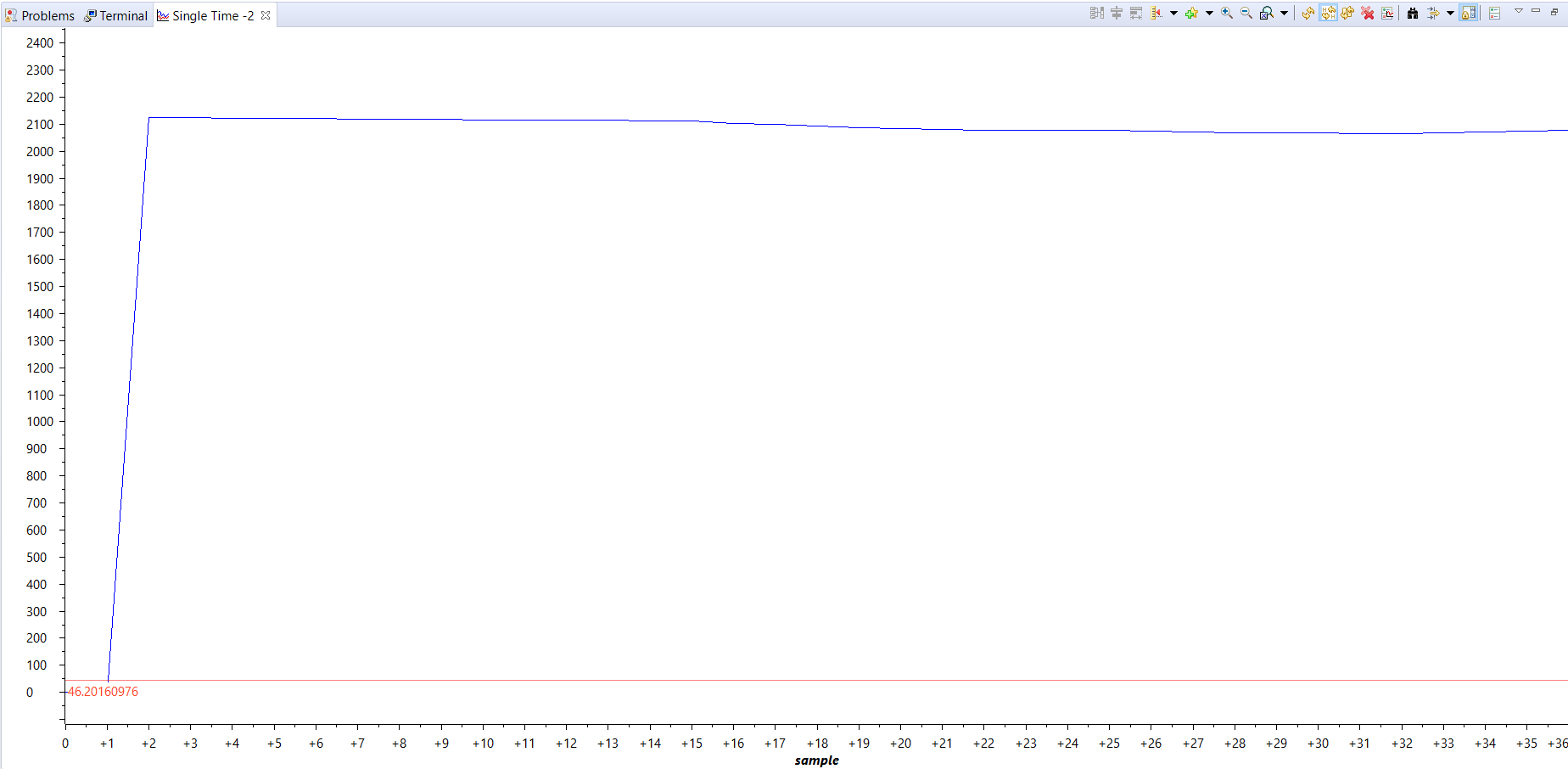
on overflow every 0.5 sec. Use the Timer1A interrupt. Display the temperature in the

built-in graph tool.

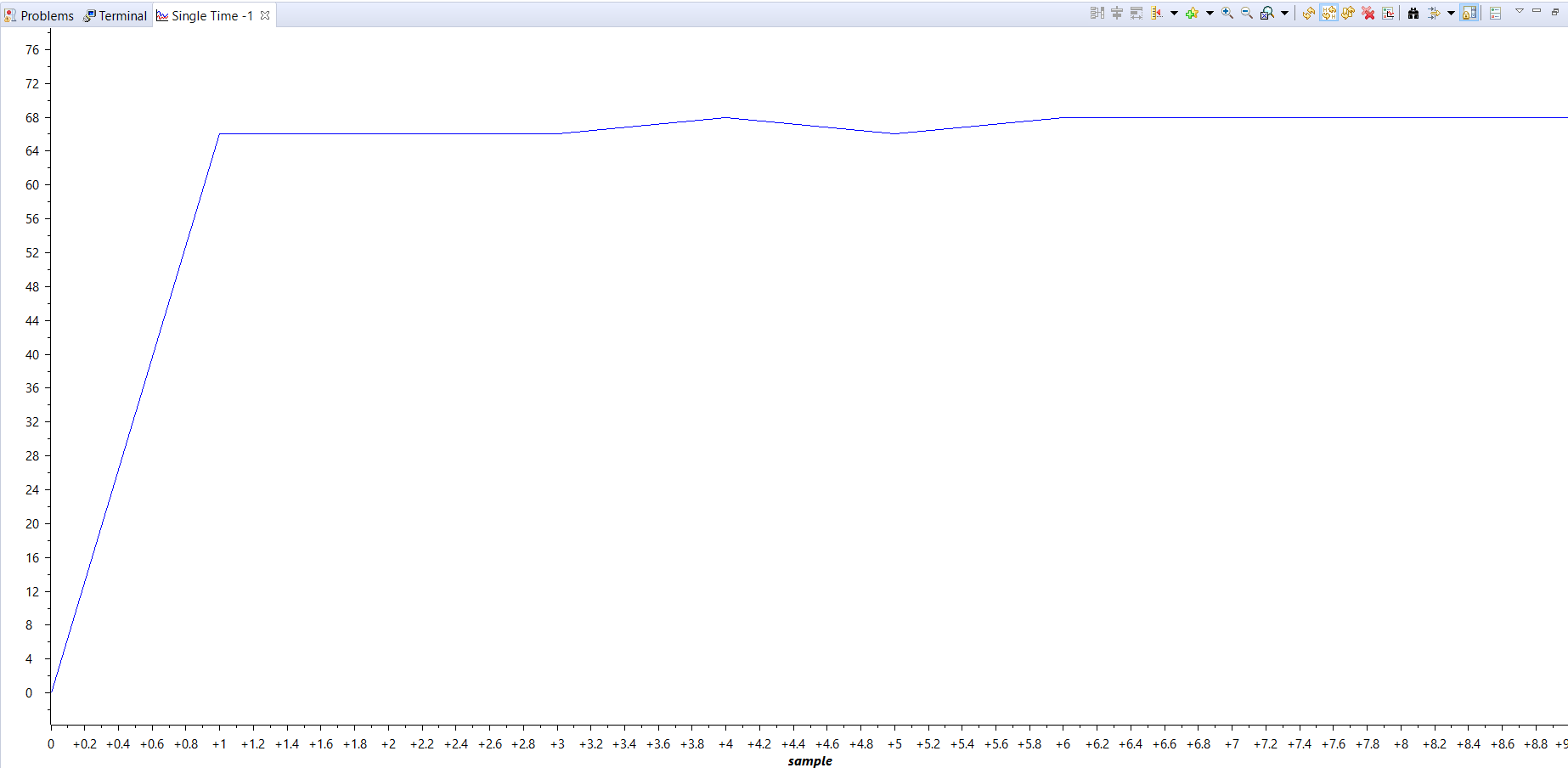
The expressions shown updating in the table, these are all updated every 0.5s:



The graph of the ADC average values polled every 0.5s:



The graph of the ADC values for Fahrenheit polled every 0.5s:



**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/adc.h"

**#define** TARGET\_IS\_BLIZZARD\_RB1

**#include** "driverlib/rom.h"

// new header files for timer interrupt and gpio

**#include** "inc/tm4c123gh6pm.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/timer.h"

uint32\_t ui32Period;

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**int** **main**(**void**)

{

// ==================================================

// Configure system clock

// CLK = 40MHz = (400MHz PLL / (5 \* 2))

// --------------------------------------------------

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_XTAL\_16MHZ|SYSCTL\_OSC\_MAIN);

// enable periphery for GPIO\_F

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

// GPIO output enable for LEDs

ROM\_GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);

// enable ADC0, set oversampling to 32

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 32);

// configure ADC sequencer to 2 and assign the ADC value types

ROM\_ADCSequenceConfigure(ADC0\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 1, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

ROM\_ADCSequenceEnable(ADC0\_BASE, 2);

// enable peripheral TIMER1

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1);

// assign a periodic configuration that overflows every 0.5s

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, **SysCtlClockGet**() \* 0.5);

// enable TIMER1

**TimerEnable**(TIMER1\_BASE, TIMER\_A);

// enable interrupt for TIMER1

**IntEnable**(INT\_TIMER1A);

**TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

// enable master for interrupts

**IntMasterEnable**();

**while**(1)

{}

}

// timer1, default LED blinking protocol

**void** **Timer1IntHandler**(**void**)

{

// Clear the timer interrupt

**TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

// clear ADC interrupt and set processor trigger to sequence 2

**ADCIntClear**(ADC0\_BASE, 2);

**ADCProcessorTrigger**(ADC0\_BASE, 2);

// while busy, keep looping

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 2, false))

{}

// grab ADC data and compute the avg value and temps for F and C

**ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

// turn blue LED if tempF is > 66, else keep it off

**if**(ui32TempValueF > 66)

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0x04);

**else**

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0x00);

}