Phillip Sortomme
Github root directory:

Date Submitted: 10/16/2018

Task 01:

Modified Code:

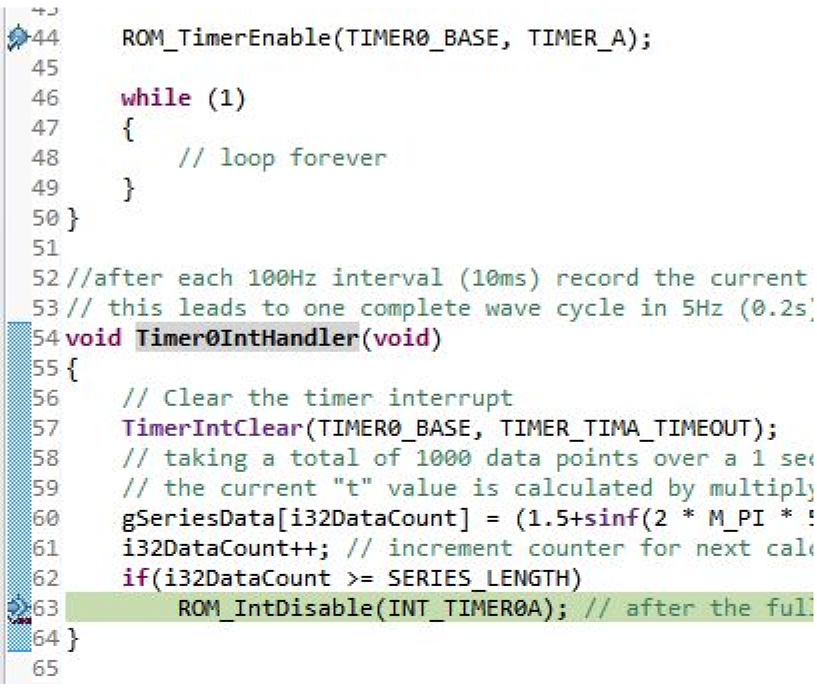
```
#include <stdint.h>
#include <stdbool.h>
#include <math.h>          // contains the sinf function
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/fpu.h"  // header file that defines floating numbers
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"

#ifndef M_PI // define the value of pi if not already done so
#define M_PI                  3.14159265358979323846
#endif

#define SERIES_LENGTH 100   // used to determine how many data points will be recorded
float gSeriesData[SERIES_LENGTH]; // a floating point array of size SERIES_LENGTH used to store sin wave points
int32_t i32DataCount = 0;   // counter for the sin function calculation

int main(void)
{
   float fRadians;

   ROM_FPULazyStackingEnable();    // enable lazy stacking. this allows the CPU to reserve space on the stack for the FPU state
   ROM_FPUEnable();               // enable the floating point unit

   ROM_SysCtlClockSet(
         SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ
               | SYSCTL_OSC_MAIN);
   //  since a full sin wave is 2Pi, divide by the number of steps taken, which in this case is 100, so fRadians = 0.062790519
   fRadians = ((2 * M_PI) / SERIES_LENGTH);

   while (i32DataCount < SERIES_LENGTH) // loop until all data points are recorded (in this case 100 times)
   {
      gSeriesData[i32DataCount] = sinf(fRadians * i32DataCount); // input the result of the sin function at each increment into the corresponding
array location
      i32DataCount++; // increment counter for next calculation
   }

   while (1)
   {
      // loop forever
   }
}
```

Task 02:

Modified Code:

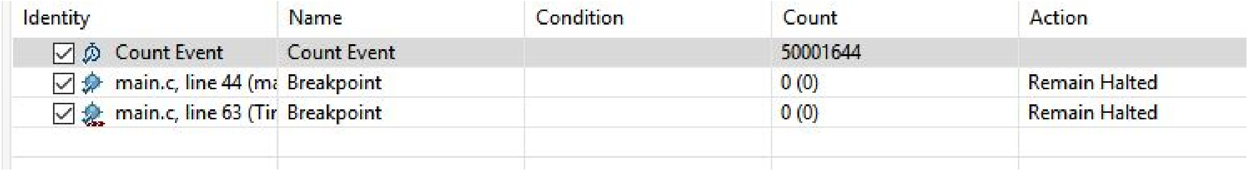
```
#include <stdint.h>
#include <stdbool.h>
#include <math.h>          // contains the sinf function
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/fpu.h"  // header file that defines floating numbers
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
```

```
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "inc/tm4c123gh6pm.h"

#ifndef M_PI // define the value of pi if not already done so
#define M_PI                    3.14159265358979323846
#endif

#define SERIES_LENGTH 1000   // used to determine how many data points will be recorded
volatile float gSeriesData[SERIES_LENGTH]; // a floating point array of size SERIES_LENGTH used to store sin wave points
volatile int32_t i32DataCount = 0;   // counter for the sin function calculation

int main(void)
{
    ROM_FPULazyStackingEnable(); // enable lazy stacking. this allows the CPU to reserve space on the stack for the FPU state
    ROM_FPUEnable();                // enable the floating point unit

    ROM_SysCtlClockSet(
    SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);

// timer0 is used to create a 100Hz clock for the purpose of recording the sin function calculation every 10ms.
// this gives a complete wave cycle in 0.2s, or every 5Hz.
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    ROM_TimerConfigure(TIMER0_BASE, TIMER_CFG_A_PERIODIC);

    // set timer0 to 100Hz.
    ROM_TimerLoadSet(TIMER0_BASE, TIMER_A,
                    ((ROM_SysCtlClockGet() / 10) / 100) - 1);

    ROM_IntEnable(INT_TIMER0A);
    ROM_TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    ROM_IntMasterEnable();
    ROM_TimerEnable(TIMER0_BASE, TIMER_A);

    while (1)
    {
        // loop forever
    }
}

//after each 100Hz interval (10ms) record the current step calculation.
// this leads to one complete wave cycle in 5Hz (0.2s).
void Timer0IntHandler(void)
{
    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    // taking a total of 1000 data points over a 1 second period equates to 20 points per cycle for a total of 5 cycles.
    // the current "t" value is calculated by multiplying the datacount by 1ms (0.001)
    gSeriesData[i32DataCount] = (1.5
            + sinf(2 * M_PI * 50 * (i32DataCount * .001))
            + 0.5 * cosf(2 * M_PI * 200 * (i32DataCount * .001)));
    i32DataCount++; // increment counter for next calculation
    if (i32DataCount >= SERIES_LENGTH)
        ROM_IntDisable(INT_TIMER0A); // after the full 1s waveform is recorded disable the timer.
}
```



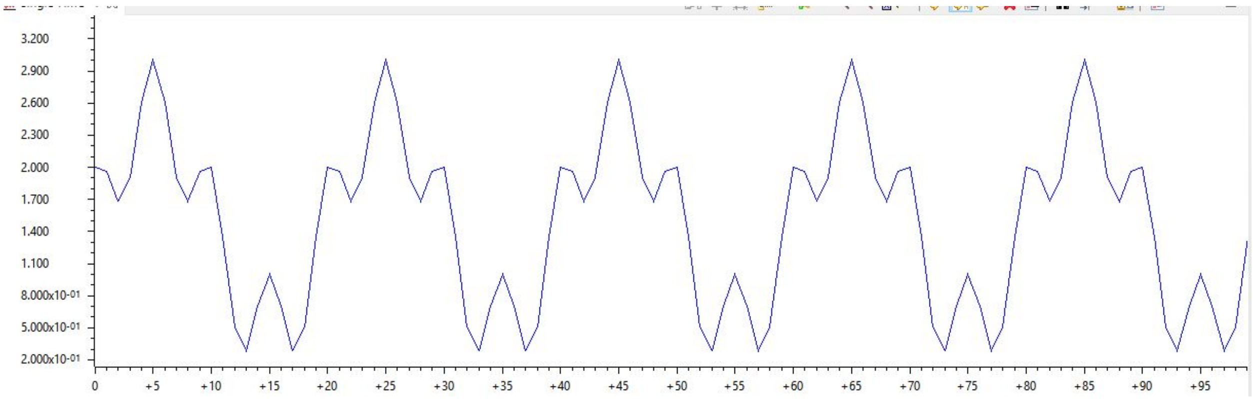
```
44     ROM_TimerEnable(TIMER0_BASE, TIMER_A);
45
46     while (1)
47     {
48         // loop forever
49     }
50 }
51
52 //after each 100Hz interval (10ms) record the current
53 // this leads to one complete wave cycle in 5Hz (0.2s
54 void Timer0IntHandler(void)
55 {
56     // Clear the timer interrupt
57     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
58     // taking a total of 1000 data points over a 1 se
59     // the current "t" value is calculated by multiply
60     gSeriesData[i32DataCount] = (1.5+sinf(2 * M_PI * 5
61     i32DataCount++; // increment counter for next cal
62     if(i32DataCount >= SERIES_LENGTH)
63         ROM_IntDisable(INT_TIMER0A); // after the full
64 }
65
```

between these two breakpoints the total number of clock cycles is shown below



Identity	Name	Condition	Count	Action
☑ Count Event	Count Event		50001644	
☑ main.c, line 44 (ma	Breakpoint		0 (0)	Remain Halted
☑ main.c, line 63 (Tir	Breakpoint		0 (0)	Remain Halted

output waveform is as follows


3.200
2.900
2.600
2.300
2.000
1.700
1.400
1.100
8.000x10-01
5.000x10-01
2.000x10-01
0
+5
+10
+15
+20
+25
+30
+35
+40
+45
+50
+55
+60
+65
+70
+75
+80
+85
+90
+95