香港中文大學
The Chinese University of Hong Kong

CENG2400 Embedded System Design

Lab 06: Analog-Digital Converter

Kezhi LI

kzli24@cse.cuhk.edu.hk

Outline

- **Step 1:** Learning the conversion between analog signal and digital signal
- **Step 2:** Learning how to enable ADC in Tiva and how to use temperature sensory in ADC
- **Step 3:** Doing your assignment (upload your code and video on blackboard before next lab)

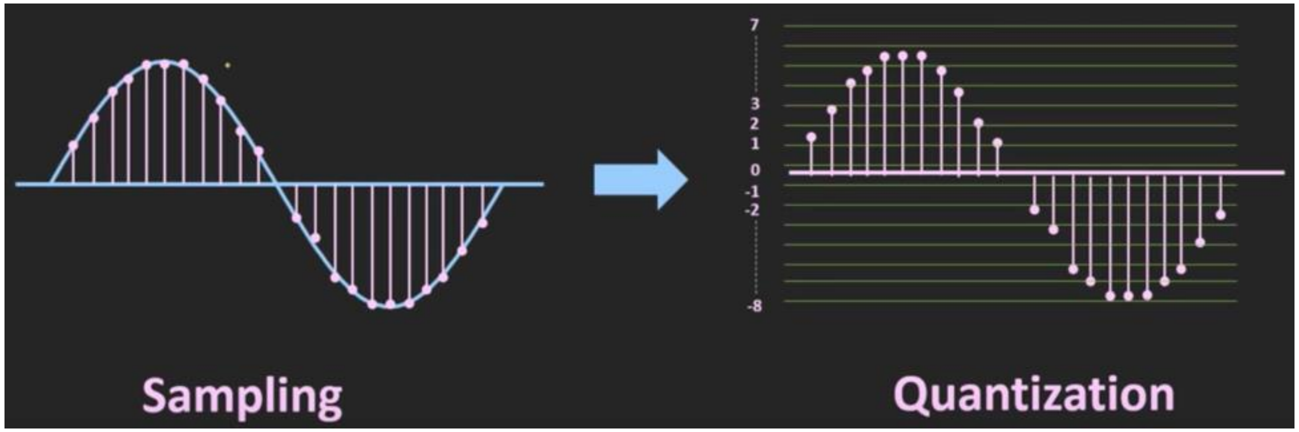
Step 1: Learning ADC

Why we need ADC?

- Analog signal is continuous and susceptible to noise, which is difficult to process.
- Digital signal is discrete and robust, which is easier than analog signal to process.

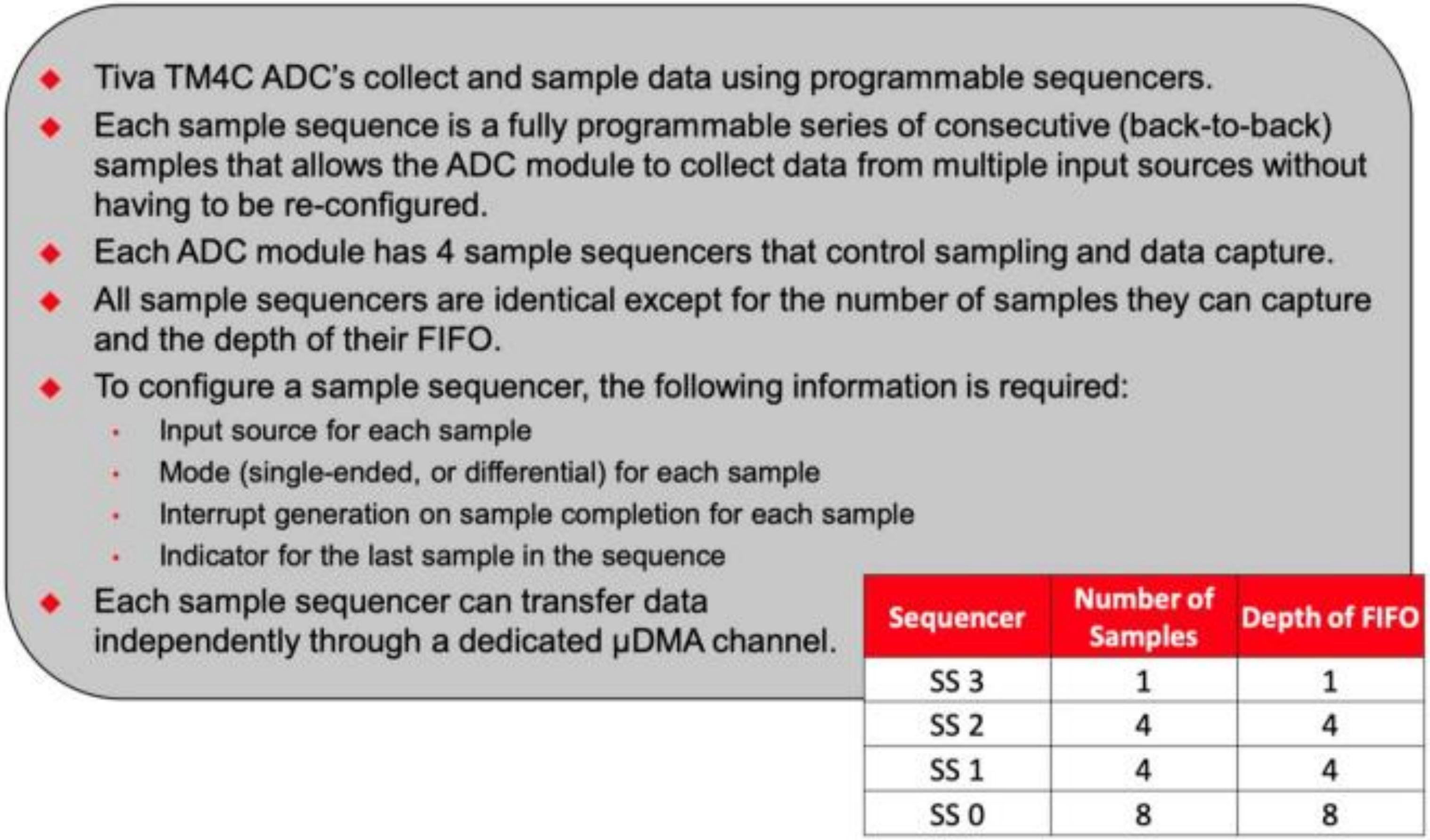
How to do ADC?

- By sampling some points from the analog signal and encoding the values into discrete integers.



Step 2: Learning ADC for Tiva

Tiva features two ADC modules (ADC0 and ADC1) that can be used to convert continuous analog voltages to discrete digital values.



- Tiva TM4C ADC's collect and sample data using programmable sequencers.
- Each sample sequence is a fully programmable series of consecutive (back-to-back) samples that allows the ADC module to collect data from multiple input sources without having to be re-configured.
- Each ADC module has 4 sample sequencers that control sampling and data capture.
- All sample sequencers are identical except for the number of samples they can capture and the depth of their FIFO.
- To configure a sample sequencer, the following information is required:
  - Input source for each sample
  - Mode (single-ended, or differential) for each sample
  - Interrupt generation on sample completion for each sample
  - Indicator for the last sample in the sequence
- Each sample sequencer can transfer data independently through a dedicated μDMA channel.

Sequencer	Number of Samples	Depth of FIFO
SS 3	1	1
SS 2	4	4
SS 1	4	4
SS 0	8	8

Step 2: Learning ADC for Tiva

1. Include header files

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

2. Enable ADC0

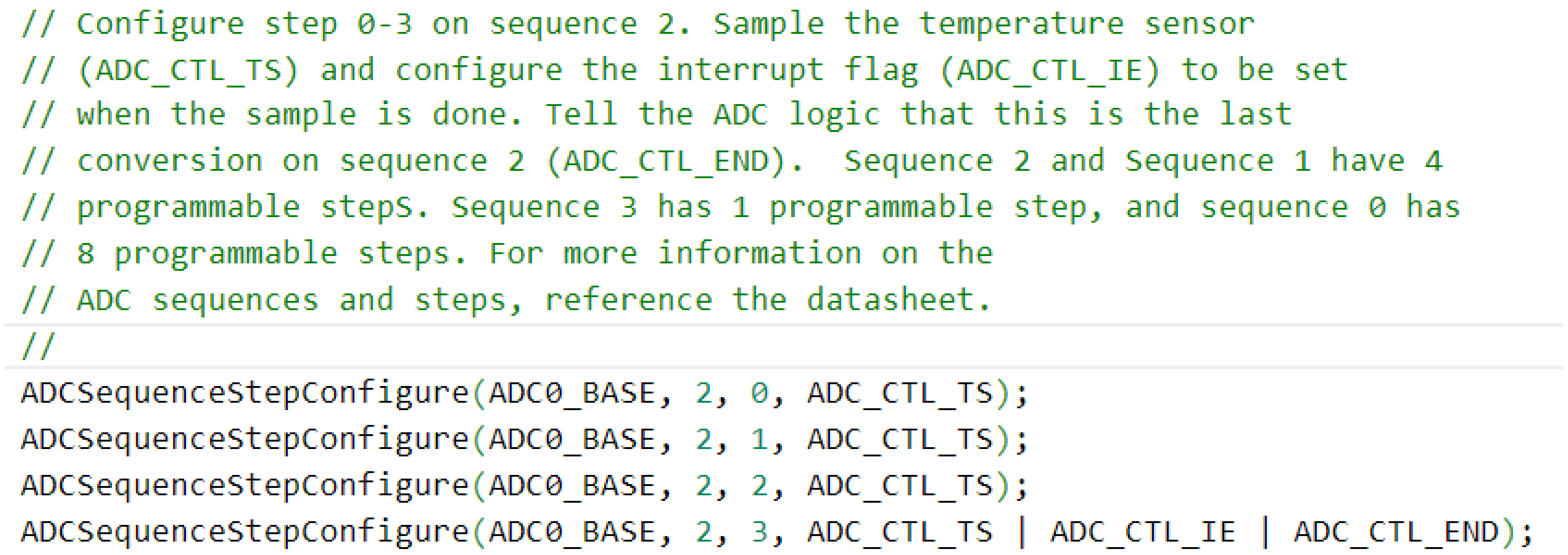
```
//
// The ADC0 peripheral must be enabled for use.
//
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
```

3. Enable sample sequence and process trigger

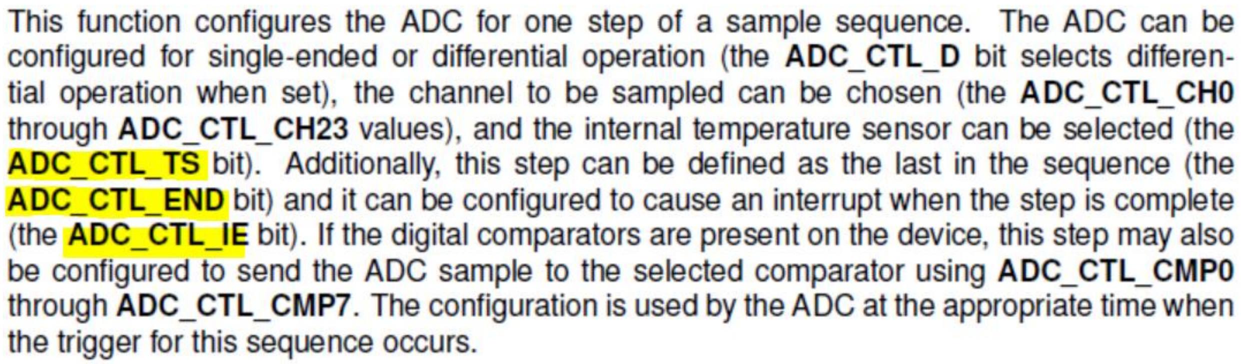
```
//
// Enable sample sequence 2 with a processor signal trigger. Sequence 2
// will do 4 sampleS when the processor sends a singal to start the
// conversion. Each ADC module has 4 programmable sequences, sequence 0
// to sequence 3. This example is arbitrarily using sequence 2.
//
ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
```

Step 2: Learning ADC for Tiva

4. Configure each sequence step



```
// Configure step 0-3 on sequence 2. Sample the temperature sensor
// (ADC_CTL_TS) and configure the interrupt flag (ADC_CTL_IE) to be set
// when the sample is done. Tell the ADC logic that this is the last
// conversion on sequence 2 (ADC_CTL_END).  Sequence 2 and Sequence 1 have 4
// programmable stepS. Sequence 3 has 1 programmable step, and sequence 0 has
// 8 programmable steps. For more information on the
// ADC sequences and steps, reference the datasheet.
//
ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
```



This function configures the ADC for one step of a sample sequence. The ADC can be configured for single-ended or differential operation (the **ADC_CTL_D** bit selects differential operation when set), the channel to be sampled can be chosen (the **ADC_CTL_CH0** through **ADC_CTL_CH23** values), and the internal temperature sensor can be selected (the **ADC_CTL_TS** bit). Additionally, this step can be defined as the last in the sequence (the **ADC_CTL_END** bit) and it can be configured to cause an interrupt when the step is complete (the **ADC_CTL_IE** bit). If the digital comparators are present on the device, this step may also be configured to send the ADC sample to the selected comparator using **ADC_CTL_CMP0** through **ADC_CTL_CMP7**. The configuration is used by the ADC at the appropriate time when the trigger for this sequence occurs.

Step 2: Learning ADC for Tiva

5. Enable ADC sequence

```
//
// Since sample sequence 2 is now configured, it must be enabled.
//
ADCSequenceEnable(ADC0_BASE, 2);
```

6. Enable interrupt handler

```
//
// Register the interrupt handler and enable ADC interrupt
//
ADCIntRegister(ADC0_BASE, 2, ADC0IntHandler);
IntEnable(INT_ADC0SS2);
ADCIntEnable(ADC0_BASE, 2);
```

7. Define interrupt handler

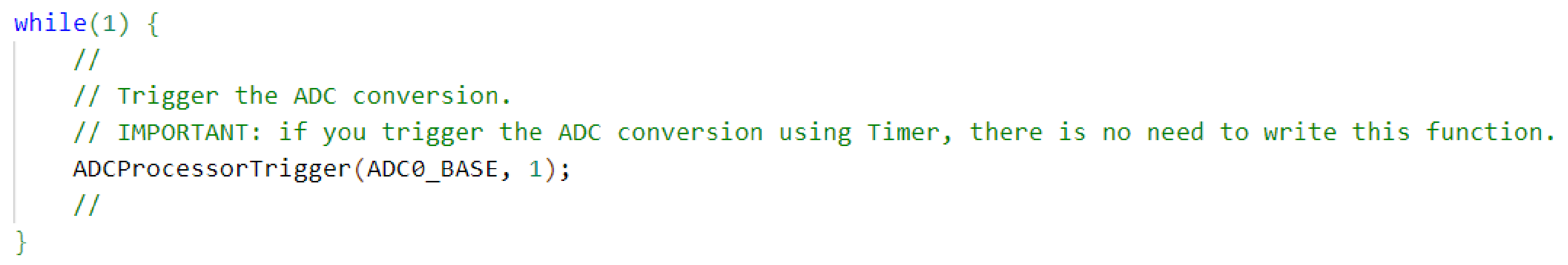
```
void ADC0IntHandler(void) {
```

8. Get data from ADC

```
//
// Get the ADC data
//
ADCSequenceDataGet(ADC0_BASE, 2, ui32ADC0Value);
```

Step 2: Learning ADC for Tiva

9. Trigger the conversion



```
while(1) {
    //
    // Trigger the ADC conversion.
    // IMPORTANT: if you trigger the ADC conversion using Timer, there is no need to write this function.
    ADCProcessorTrigger(ADC0_BASE, 1);
    //
}
```

TODO: How to trigger the conversion using timer (hardware) instead of processor (software)?

- **Software Trigger**: requires the software to write a value to a specific ADC control register to start the conversion.

- **Hardware Trigger**: requires a hardware signal to be asserted by a circuit or peripheral (e.g., timer).

Step 2: Learning ADC for Tiva

HINT: Remember what we have done in Lab3

1. Configure a Timer and enable Timer trigger

```
SysCtlPeripheralEnable(…);
TimerConfigure(…);
TimerLoadSet(…);

TimerControlTrigger(…, …, true);

TimerEnable(…, …);
```

2. Change processor trigger mode to timer trigger

```
ADCSequenceConfigure(ADC0_BASE,
1, ADC_TRIGGER_PROCESSOR, 0);
```

➡

```
ADCSequenceConfigure(ADC0_BASE,
1, ADC_TRIGGER_TIMER, 0);
```

Step 2: Learning temperature sensor

Since we get four samples from sequence 2, we need to average them!

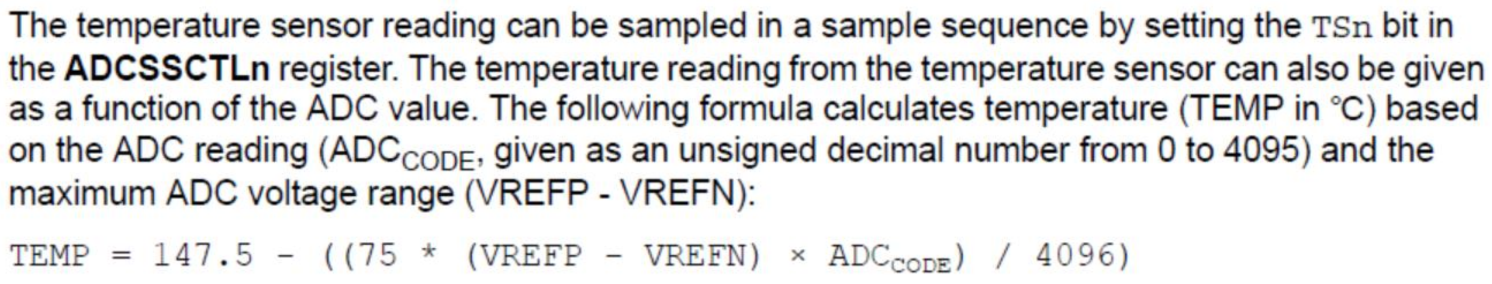
```
ui32VolAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] +
ui32ADC0Value[2] + ui32ADC0Value[3] + 2) / 4;
```

Why “+2” when averaging?

Since the four values are integer, the divide result is also a integer in “c”. However, we want to round off the result to the closest integer. For example, if dividend is 9, the result should be 2.25 and round off to 2. (9+2) / 4 = 2. If the dividend is 11, the result should be 2.75 and round off to 3. (11+2) / 4 = 3.

Step 2: Learning temperature sensor

Actually, the value we have measured is the "voltage value". There is a equation to calculate the temperature from the voltage. ADC_CODE = 3.3.



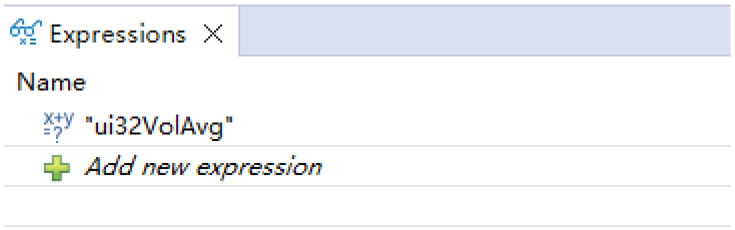
The temperature sensor reading can be sampled in a sample sequence by setting the `TSn` bit in the **ADCSSCTLn** register. The temperature reading from the temperature sensor can also be given as a function of the ADC value. The following formula calculates temperature (TEMP in °C) based on the ADC reading (ADC_{CODE}, given as an unsigned decimal number from 0 to 4095) and the maximum ADC voltage range (VREFP - VREFN):

$$TEMP = 147.5 - ((75 * (VREFP - VREFN) \times ADC_{CODE}) / 4096)$$

Based on the equation, what you get is the Celsius temperature. To get the Fahrenheit temperature, apply another transformation. (Search online if needed)

Step 2: How to visualize the results?

1. Build and debug the program.
2. Select “View” -> “Expressions”.
3. Add new expression to supervise.



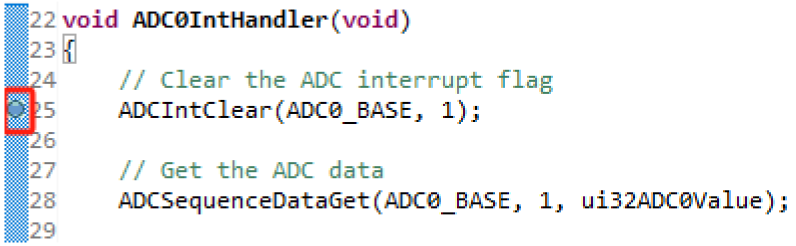
Expressions

Name

"ui32VolAvg"

Add new expression

4. Add a break point in the ADC handler.

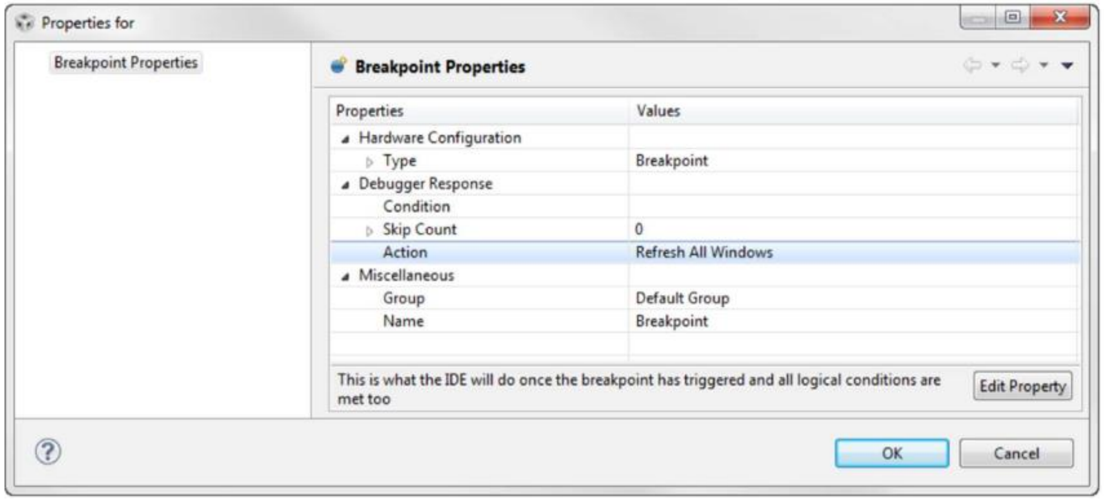


```
void ADC0IntHandler(void)
{
    // Clear the ADC interrupt flag
    ADCIntClear(ADC0_BASE, 1);

    // Get the ADC data
    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
```

Step 2: How to visualize the results?

5. Right click the break point and choose "Breakpoint Properties". Change the action to "Refresh All Windows".



Breakpoint Properties

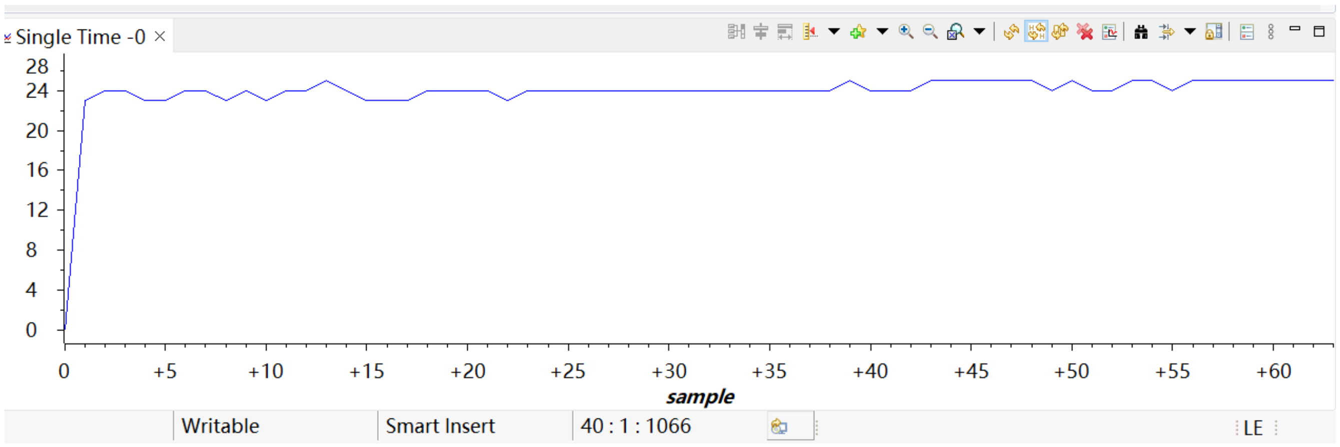
Properties	Values
Hardware Configuration	
Type	Breakpoint
Debugger Response	
Condition	
Skip Count	0
Action	Refresh All Windows
Miscellaneous	
Group	Default Group
Name	Breakpoint

This is what the IDE will do once the breakpoint has triggered and all logical conditions are met too

6. Right click the supervised value and choose "graph" to visualize its value in a graph.

Step 2: How to visualize the results?

The expected visualization should be like this:



Step 3: Assignments

Assignment 1: Carefully read the provided code and make visualization of “ui32VolAvg”.

Assignment 2: Based on the equation, calculate “ui32TempValueC” and “ui32TempValueF”, and make visualization of them.

Assignment 3: Change the processor trigger to timer trigger based on the instruction.

Q&A

Thanks for listening!

Q & A