Lecture 18

Project Prep

Agenda for the class:

Prep for the Project

Doing Real-World Math with the MCU

Q-Format Numbers

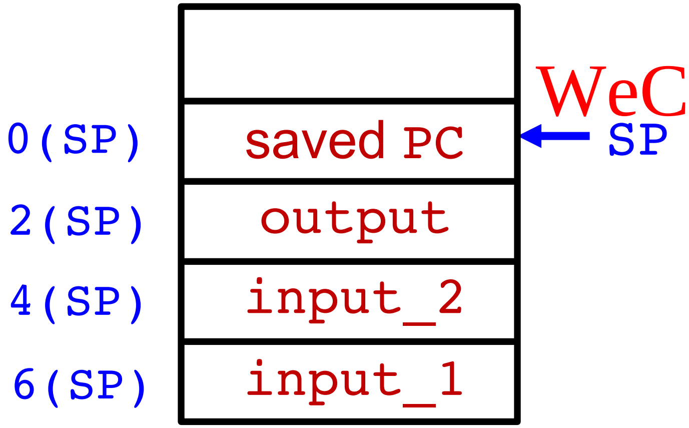
CCS Tools: Load Memory & Graph

In class coding demo

Last Time: Stack Frames

The subroutine contract specifies the structure of the **stack frame** the subroutine will see when it is first called

e.g., a stack frame with two input values and one output value



0(SP) saved PC ← SP

2(SP) output

4(SP) input_2

6(SP) input_1

- Caller pushes `input_1`, then `input_2`
- With the subroutine call `PC` is placed onto the stack

Subroutine

- reads `input_1` and `input_2`
- computes and writes `output` into the stack frame

`ret` removes `PC` from stack

Caller

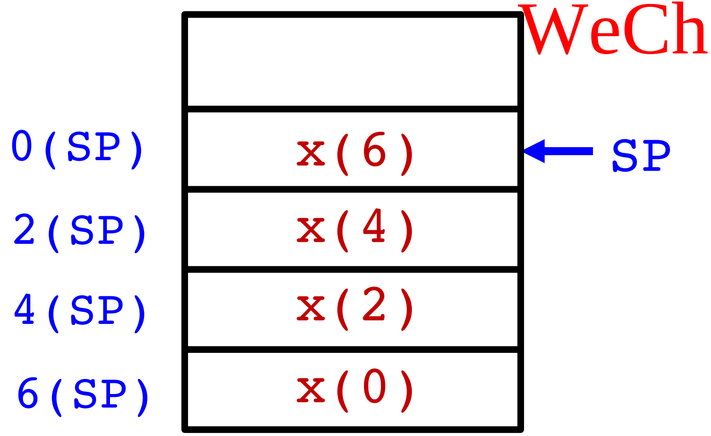
- reads `output` from stack frame
- cleans up the rest of the stack

Temporary Variables on Stack

We can also create a similar structure in stack without calling a subroutine

⇒ **Dynamic data allocation**

e.g., copy of an array with 4 elements `x = {x(0), x(2), x(4), x(6)}`



```
        clr.w    R4
rep:    push     x(R4)
        incd.w   R4
        cmp.w    #8, R4
        jne      rep

        ...

        add.w   #8,   SP
```

release memory on stack after use!!

Project

(Tentative) Task: You will be given some information buried in noise and you will recover it

Will use sin(x) and cos(x) as carriers of information

- Find a way of dealing with *fractional* numbers in code and CCS
- Load data into memory: sin(x), cos(x), data points will be provided in a file
- Plot the graph of sin(x), cos(x), data points
- Project the data points to sin(x) and cos(x) to find the buried information

You will write two subroutines:

- Inner product
- Signed multiplication

These can use the subroutines we have already developed: e.g., x_times_y

Fixed-Point Rational Numbers

There is only so much math we can do with integers only

We cannot even properly divide by a power of two: 27/4 = 6.75

We can write this number in binary with integer and fractional part

```
  110.11 = 4 + 2 + 1/2 + 1/4 = 6.75
 1010.10 = 10.5
    0.01 = 0.25
   11.00 = 3.00
```

Note: Demo only. Never find decimal values like this!

All these numbers have **2 fractional bits** after the radix point

⇒ Fixed-point representation with two fractional bits

⇒ Called **Q2** format by Texas Instruments with **Q Value 2**

This is only how we interpret a binary sequence, HW does not care !!

Q Format

What does Q Value mean?

Q Value alone is incomplete, we have to consider the number of bits too

A 16-bit signed number with Q value 2 is well defined:

$\texttt{0x002B} = \underbrace{\texttt{0000 0000 0010 10}}_{\textbf{14 integer bits}}\underbrace{\texttt{11}}_{\textbf{2 fractional bits}}$

$= (10.75)_{10}$

imagine a radix point here

signed Q(14.2) number

An 8-bit **signed** number with Q value 7

signed Q(1.7) number

$\texttt{0xF2} = \texttt{1}\underbrace{\texttt{111 0010}}_{\textbf{7 fractional bits}}$

radix point here

Value in decimal ? Complicated!

Q Format to Decimal Conversion

Good news: It is easy if you do it the easy way

Better news: You will make CCS do it for you (most of the time)

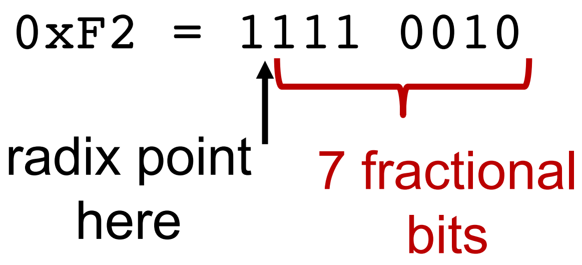
What is the easy way?

Do not attempt to add up place values – too complicated with (–)ve numbers

Instead, divide the integer value of the number by the **correct power of two**

i.e., the Q value

An 8-bit signed number with **Q value 7**



`0xF2 = 1111 0010`

radix point here

7 fractional bits

Integer value of `0xF2` is `-14`

Shifting radix point 7 positions to the left means dividing the number by **2^7 = 128**

⇒ decimal value of `0xF2` in Q(1.7) format is `-14/128 = -0.0546875`

Q Format ↔ Decimal

All you need is a Hex ↔ Decimal convertor and a calculator

Q value 0

`0xD1C8 = (-11832)`$_{10}$

signed!

In Q(16.0) the decimal value is -11832 / 2^0

Q(14.2) -11832 / 2^2

Q(8.8) -11832 / 2^8

Q(1.15) -11832 / 2^{15}

Q value 15

If we want to convert from decimal to Q format

We have to watch the range of numbers!!

e.g., Q(1.15) can represent only numbers in the range **[– 1, 1)**

open

Once you fix the range, multiply by the **correct power of 2**

5.245 in Q(9.7): 5. 245 x 2^7 = 671.36 approx. `0x029F`

Q value 7

Q Format Arithmetic

This is only how we interpret a binary sequence, HW does not care !!

Addition and subtraction are easy

- Let the HW do `add` and `sub`
- Just make sure to only add and subtract numbers in the same Q format

corresponds to aligning the radix point when adding and subtracting

Multiplication is easy too but watch the format change

- Let the HW multiply
- Always watch for overflow – Q format or not!
- **The Q values are added!!**

same as decimal point placement in decimal multiplication

Multiplying a Q(1.**7**) number with a Q(2.**5**) number ⇒ Q(4.**12**) number

$x / 2^7$ $y / 2^5$ $xy / 2^{12}$

Changing between Q Formats

Decreasing the Q value without changing the encoded value

e.g., from Q(1.7) to Q(4.4) – Q value decreases from 7 to 4

```
0.1000000 = 0x80  => 0000.1000 = 0x04
```

both numbers encode decimal value 0.5

Shift the radix point 3 positions to the **right** = **divide** by $2^{7-4} = 2^3$

Increasing the Q Value

e.g., from Q(8.0) to Q(4.4) – Q value increases from 0 to 4

```
00000001. = 0x01  => 0001.0000 = 0x10
```

both numbers encode decimal value 1.0

Shift the radix point 4 positions to the **left** = **multiply** by 2^4

Motivation

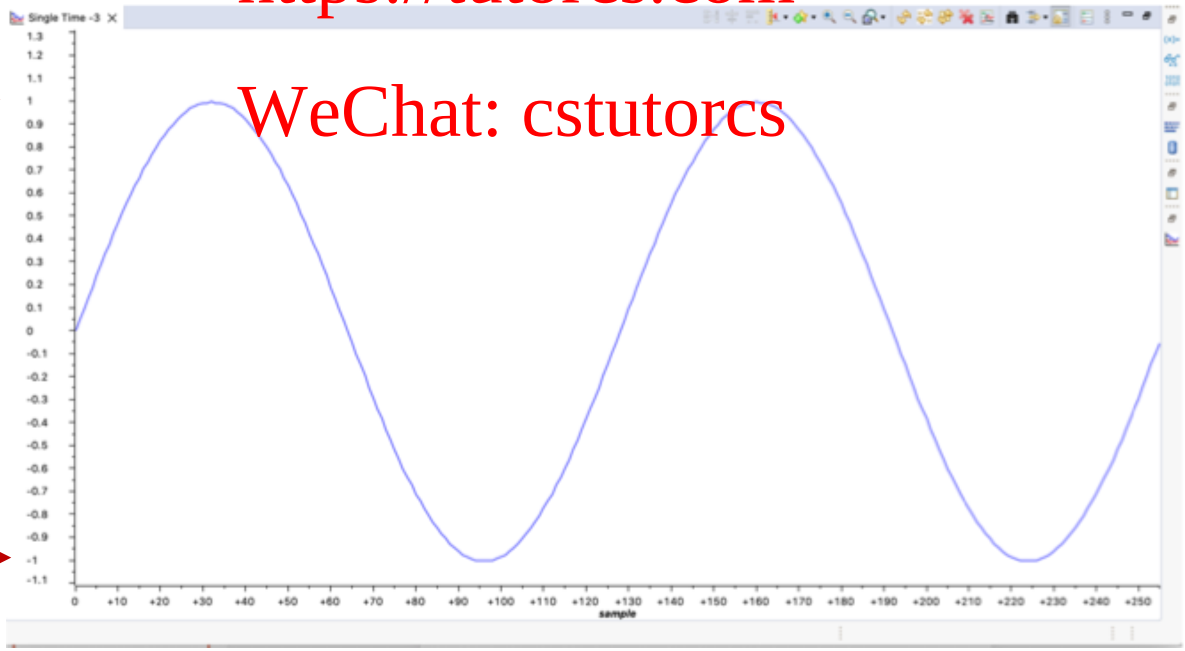
Why do we need all this?

Because there is so much more to numbers than integers

e.g., sin(x) produces numbers in the range [– 1, 1]

without fractions there is no sine, only a square wave!

For the project we will use sin(x) – and more



1.0 →

– 1.0 →

properly!

Importing Data into the MCU

CCS enables importing external data into the RAM/FRAM of the MCU

Several file formats are supported, we will use TI Data format

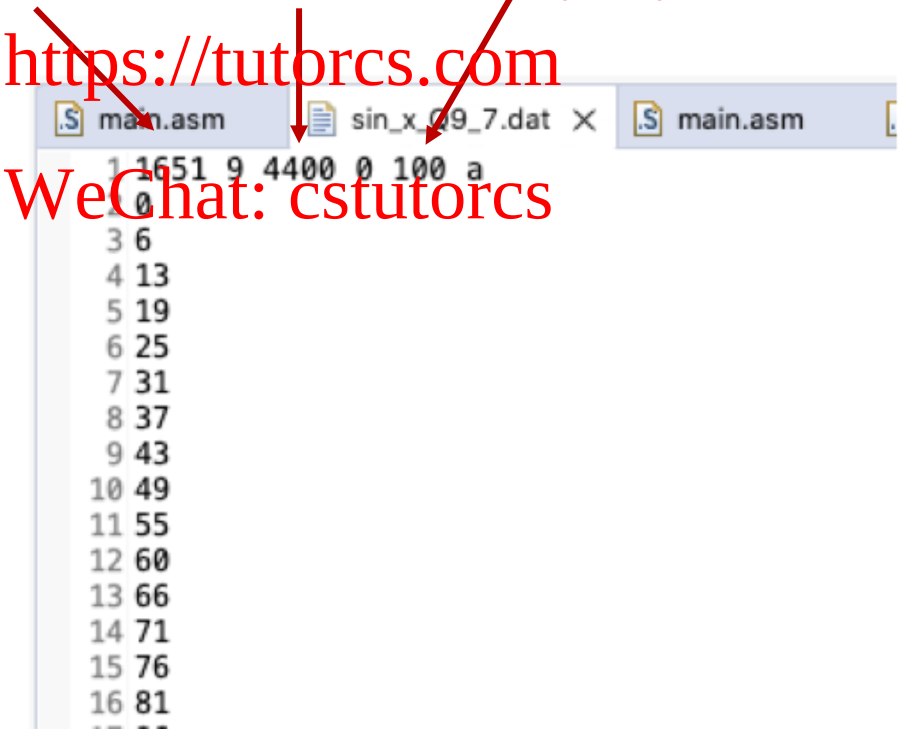
magic number

starting address (hex)

length in words (hex)

header line

one data sample per line

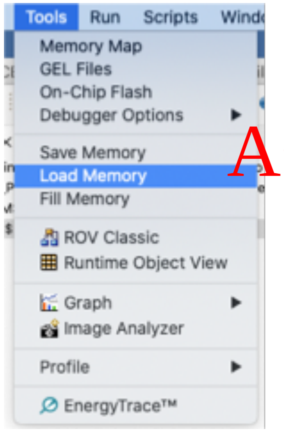


```
1651 9 4400 0 100 a
0
6
13
19
25
31
37
43
49
55
60
66
71
76
81
```

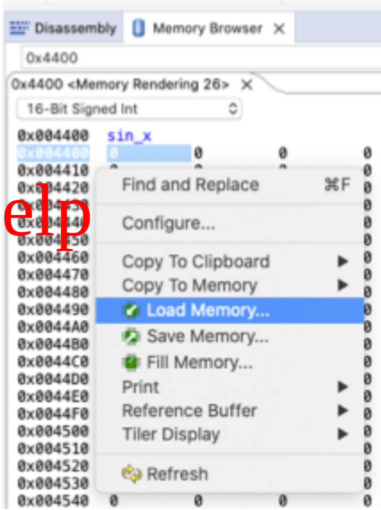
Importing Data into the MCU

You can import data **only during an active debug session**

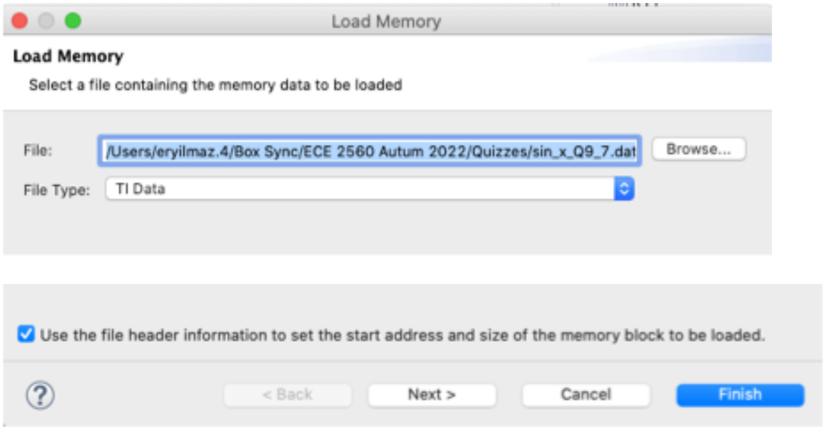
Find “Load Memory” under the Tools Tab



Alternatively, right click inside the memory browser and select ”Load Memory”



Browse for the file you want to upload in the “Load Memory” dialogue


Load Memory

Load Memory

Select a file containing the memory data to be loaded

File: /Users/eryilmaz.4/Box Sync/ECE 2560 Autum 2022/Quizzes/sin_x_Q9_7.dat

File Type: TI Data

☑ Use the file header information to set the start address and size of the memory block to be loaded.

Make sure to check the box at the bottom: Use the file header information…

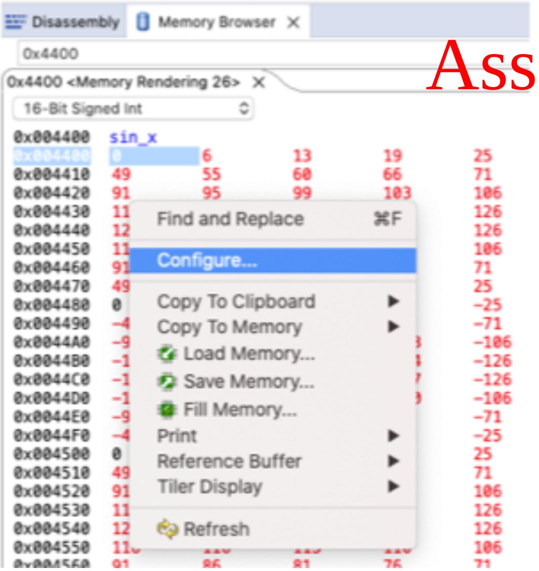
To confirm starting address and length if necessary hit “Next”

Hit “Finish”

Configuring the Q Format

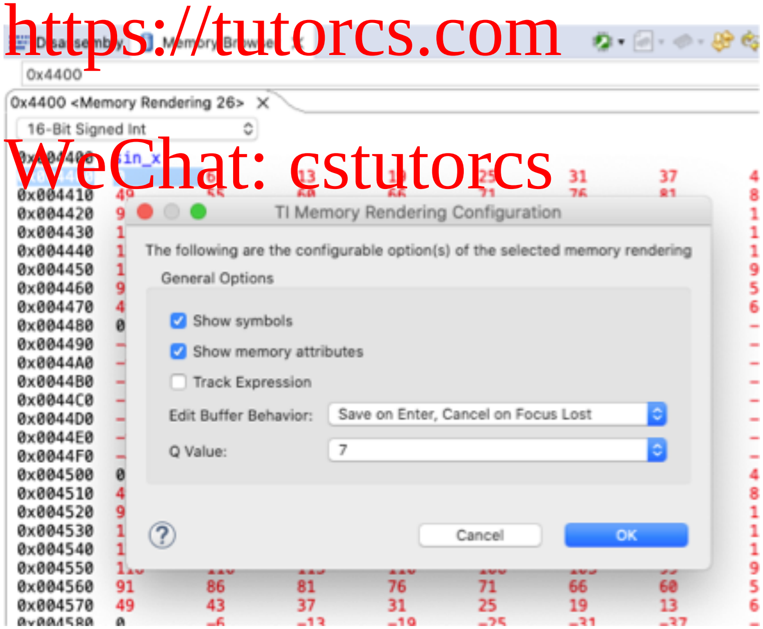
You will see the values in the file populate the MCU memory

Choose “16-bit Signed Integer” and right click on the memory browser to access the “Configure” option



```
0x004400  sin_x
0x004400  0      6      13     19     25
0x004410  49     55     60     66     71
0x004420  91     95     99     103    106
0x004430  11                          126
0x004440  12                          126
0x004450  11                          106
0x004460  91                          71
0x004470  49                          25
0x004480  0                           -25
0x004490  -4                          -71
0x0044A0  -9                          -106
0x0044B0  -1                          -126
0x0044C0  -1                          -126
0x0044D0  -1                          -106
0x0044E0  -9                          -71
0x0044F0  -4                          -25
0x004500  0                           25
0x004510  49                          71
0x004520  91                          106
0x004530  11                          126
0x004540  12                          126
0x004550  11                          106
```

Select the Q value that is associated with the data



TI Memory Rendering Configuration

The following are the configurable option(s) of the selected memory rendering

General Options

☑ Show symbols

☑ Show memory attributes

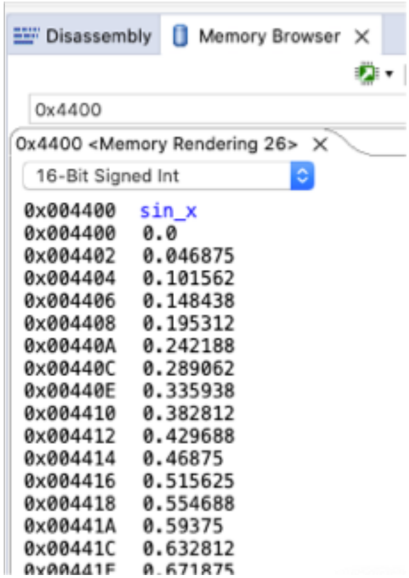
□ Track Expression

Edit Buffer Behavior: Save on Enter, Cancel on Focus Lost

Q Value: 7

```
0x004560  91     86     81     76     71     66     60
0x004570  49     43     37     31     25     19     13
```

CCS will display fractional values

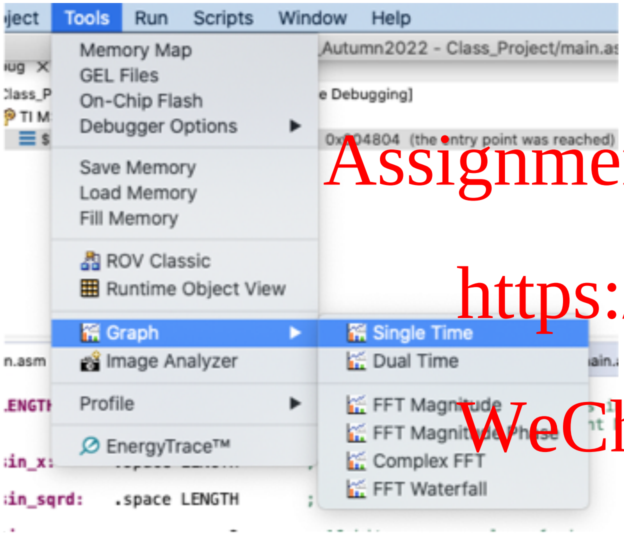


```
0x004400  sin_x
0x004400  0.0
0x004402  0.046875
0x004404  0.101562
0x004406  0.148438
0x004408  0.195312
0x00440A  0.242188
0x00440C  0.289062
0x00440E  0.335938
0x004410  0.382812
0x004412  0.429688
0x004414  0.46875
0x004416  0.515625
0x004418  0.554688
0x00441A  0.59375
0x00441C  0.632812
0x00441E  0.671875
```

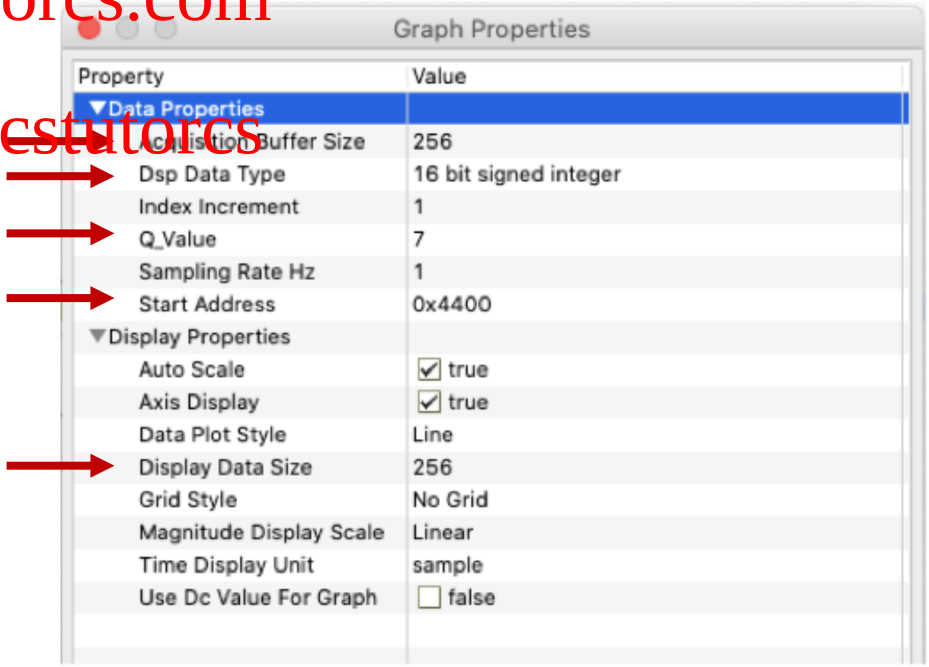
Graphing

You can graph **only during an active debug session**

Find “Graph >> Single Time” under the Tools Tab



Populate the marked fields in the “Graph Properties” dialogue including the right Q Value



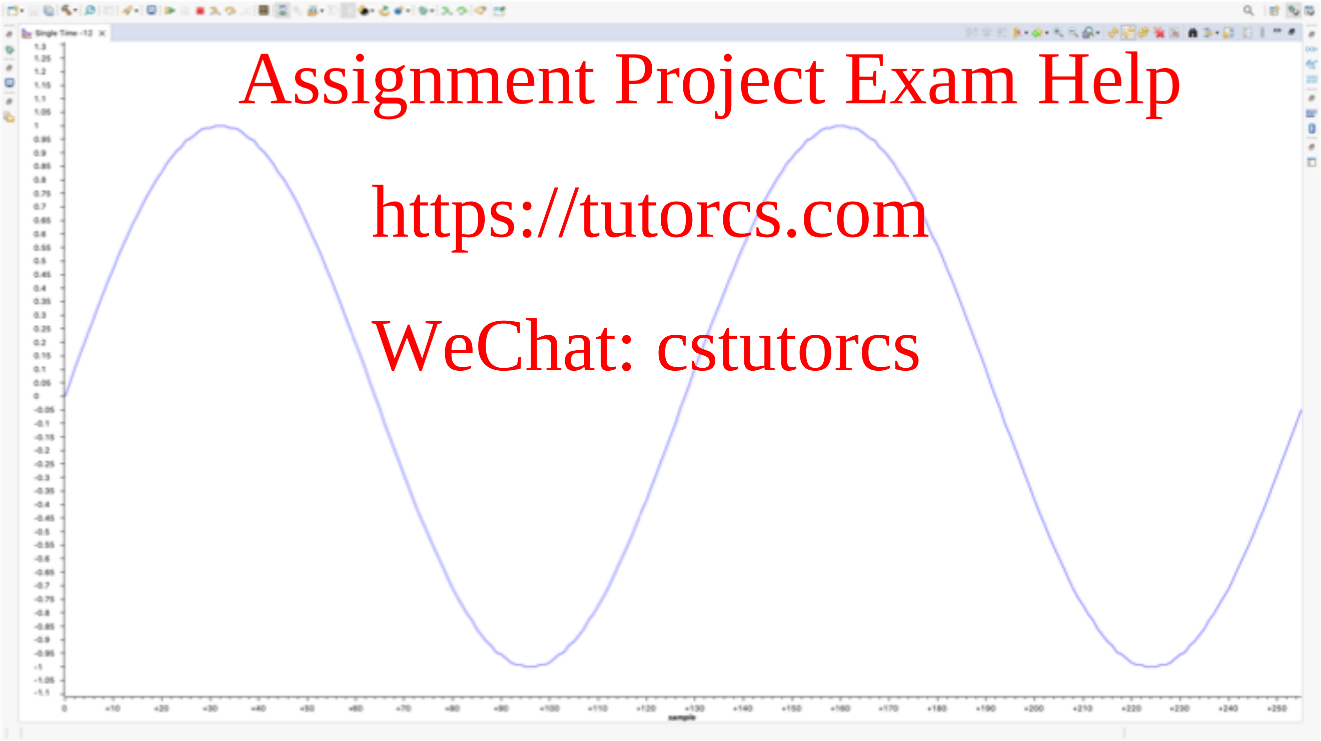
Property	Value
▼Data Properties	
Acquisition Buffer Size	256
Dsp Data Type	16 bit signed integer
Index Increment	1
Q_Value	7
Sampling Rate Hz	1
Start Address	0x4400
▼Display Properties	
Auto Scale	☑ true
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	256
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	□ false

Double check that all fields are filled with the correct parameters and hit “OK”

Graphing

CCS will graph the values in the specified memory locations

Double check axes and values, if they are not correct make necessary adjustments in the “Graph Properties” dialogue


Single Time - 12
1.3
1.25
1.2
1.15
1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
-0.4
-0.45
-0.5
-0.55
-0.6
-0.65
-0.7
-0.75
-0.8
-0.85
-0.9
-0.95
-1
-1.05
-1.1
0
+10
+20
+30
+40
+50
+60
+70
+80
+90
+100
+110
+120
+130
+140
+150
+160
+170
+180
+190
+200
+210
+220
+230
+240
+250
sample