ADC Programming

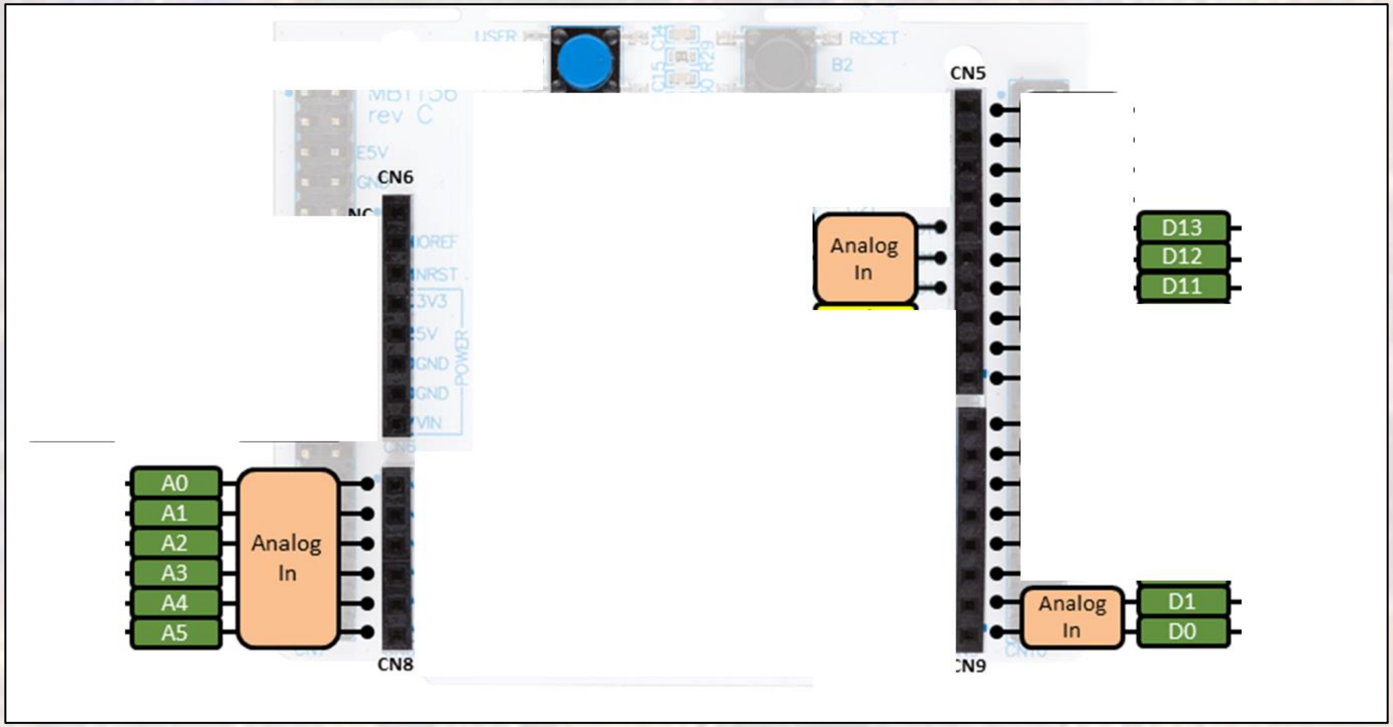
Last updated 5/14/21

ADC Programming

- ADC Resolution
  - Nucleo-L476RG has three 12Bit ADCs
    - mbed only supports 1 ADC
    - mbed scales ALL measurement (see class functions)

ADC Programming

- ADC Connections
  - Nucleo-L476RG has 11 analog inputs assigned to the Arduino header
  - Nucleo-L476RG has 4 additional analog inputs assigned to the Morpho header (not shown)

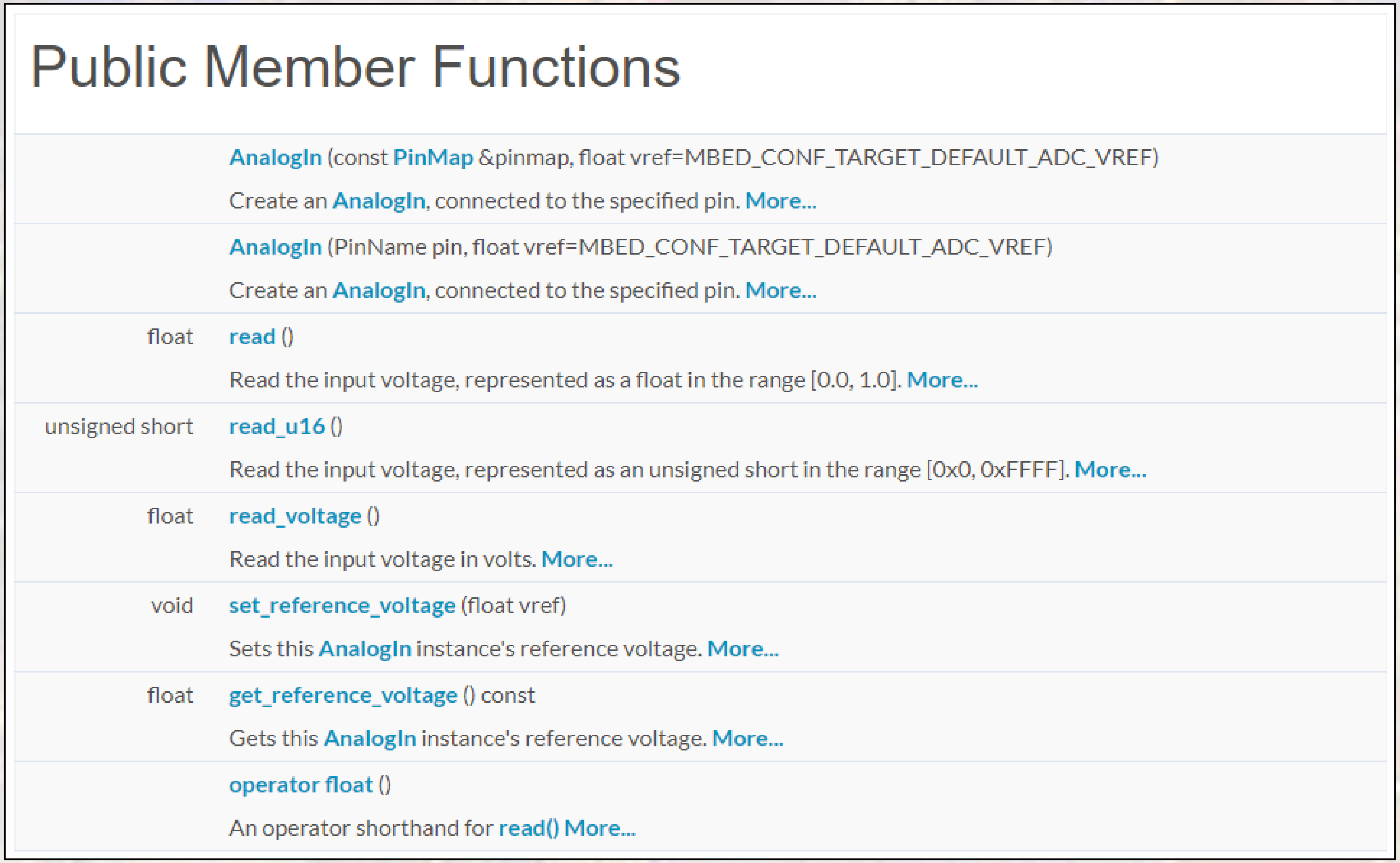


ADC Programming

- ADC Reference Voltage
  - Nucleo-L476RG uses 3.3V as the default Vref for the ADC
  - Note:
    - This value is not the same as the “reference_voltage” used in the AnalogIn class functions
    - The class functions use “reference voltage” as a scaling factor
    - The “reference_voltage” can be set to anything you want – it ONLY impacts the read_voltage function scaling factor, not the actual Vref value

ADC Programming

- AnalogIn Class
  - `#include <AnalogIn.h>` - already included with mbed.h

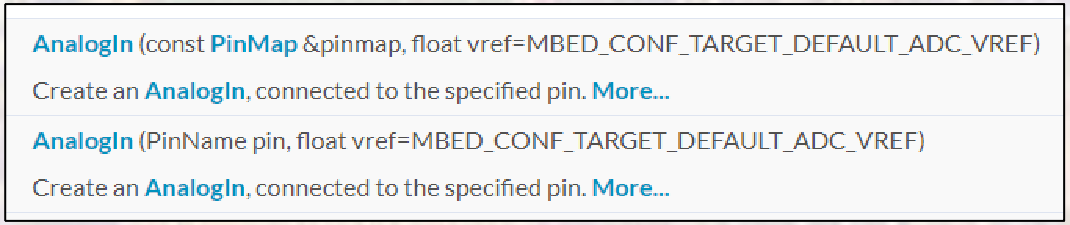


Public Member Functions

	AnalogIn (const PinMap &pinmap, float vref=MBED_CONF_TARGET_DEFAULT_ADC_VREF)
	Create an AnalogIn, connected to the specified pin. More...
	AnalogIn (PinName pin, float vref=MBED_CONF_TARGET_DEFAULT_ADC_VREF)
	Create an AnalogIn, connected to the specified pin. More...
float	read ()
	Read the input voltage, represented as a float in the range [0.0, 1.0]. More...
unsigned short	read_u16 ()
	Read the input voltage, represented as an unsigned short in the range [0x0, 0xFFFF]. More...
float	read_voltage ()
	Read the input voltage in volts. More...
void	set_reference_voltage (float vref)
	Sets this AnalogIn instance's reference voltage. More...
float	get_reference_voltage () const
	Gets this AnalogIn instance's reference voltage. More...
	operator float ()
	An operator shorthand for read() More...

ADC Programming

- Constructors

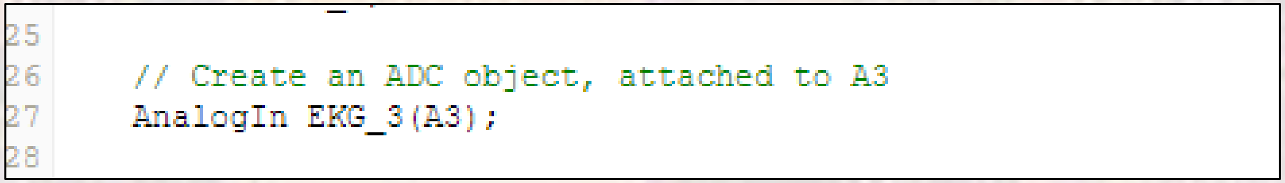


AnalogIn (const PinMap &pinmap, float vref=MBED_CONF_TARGET_DEFAULT_ADC_VREF)

Create an AnalogIn, connected to the specified pin. More...

AnalogIn (PinName pin, float vref=MBED_CONF_TARGET_DEFAULT_ADC_VREF)

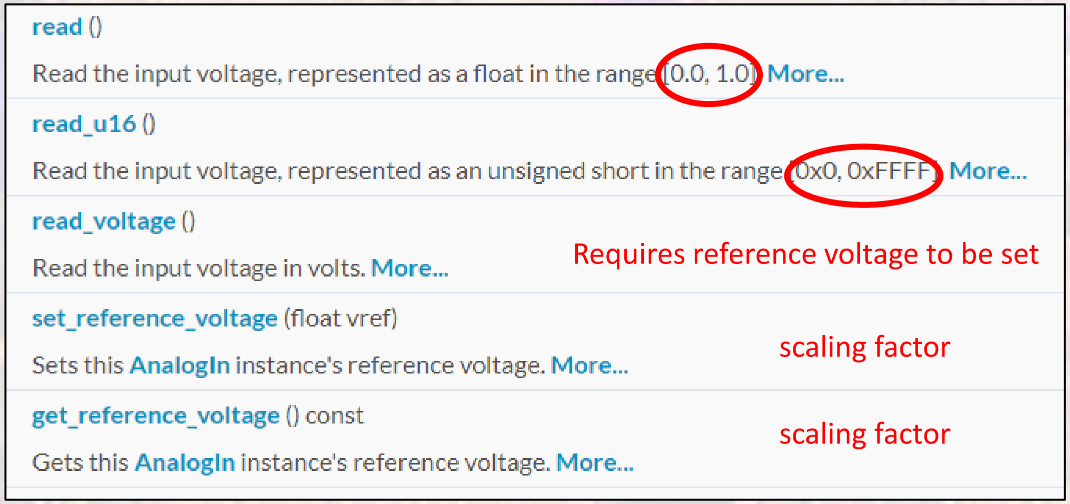
Create an AnalogIn, connected to the specified pin. More...



```

    // Create an ADC object, attached to A3
    AnalogIn EKG_3(A3);

```

ADC Programming

- Member Functions (Methods)



read ()

Read the input voltage, represented as a float in the range [0.0, 1.0] More...

read_u16 ()

Read the input voltage, represented as an unsigned short in the range [0x0, 0xFFFF] More...

read_voltage ()

Read the input voltage in volts. More...

set_reference_voltage (float vref)

Sets this AnalogIn instance's reference voltage. More...

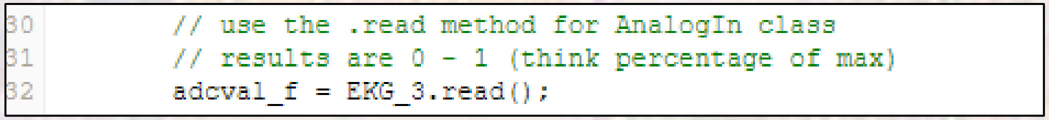
get_reference_voltage () const

Gets this AnalogIn instance's reference voltage. More...

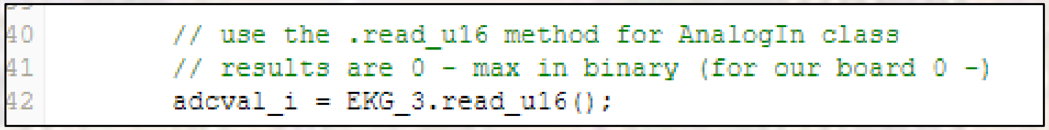
Requires reference voltage to be set

scaling factor

scaling factor



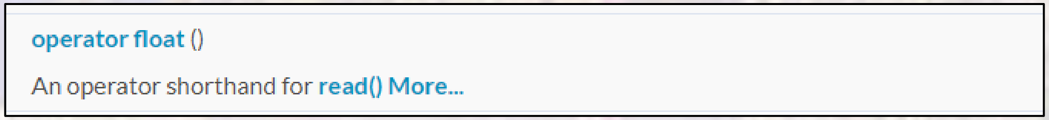
```
        // use the .read method for AnalogIn class
        // results are 0 - 1 (think percentage of max)
        adcval_f = EKG_3.read();
```



```
        // use the .read_u16 method for AnalogIn class
        // results are 0 - max in binary (for our board 0 -)
        adcval_i = EKG_3.read_u16();
```

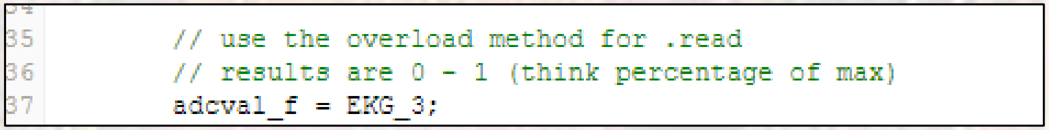
ADC Programming

- Operator Overloads



operator float ()

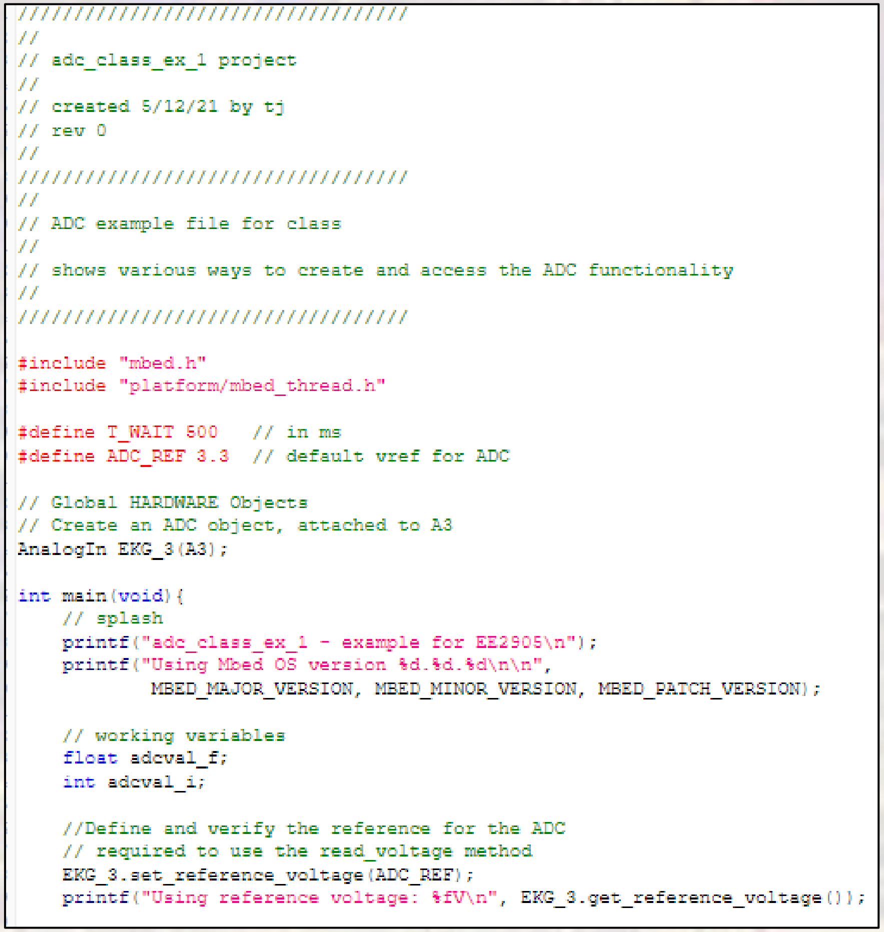
An operator shorthand for read() More...



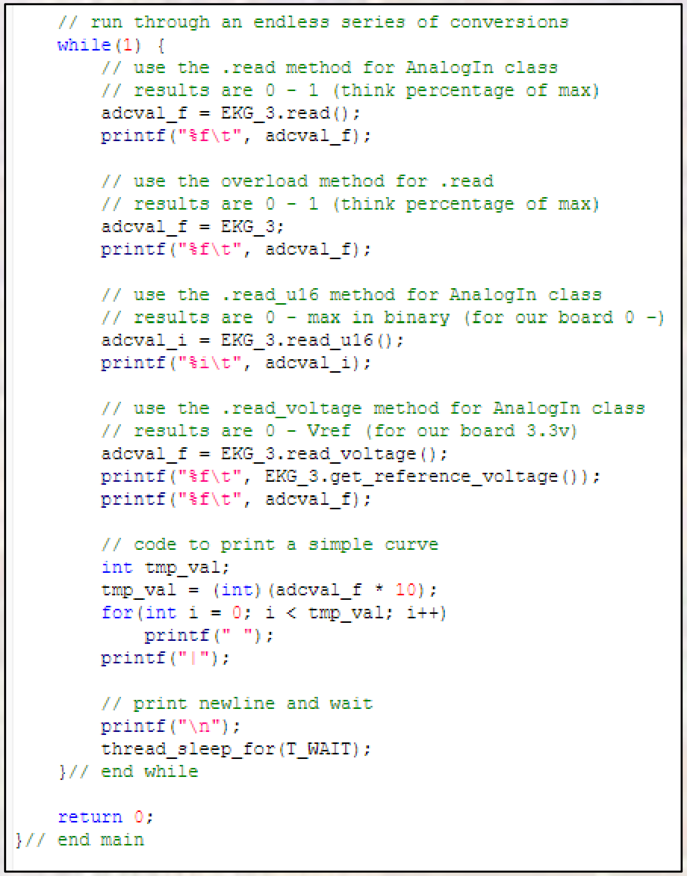
```
        // use the overload method for .read
        // results are 0 - 1 (think percentage of max)
        adcval_f = EKG_3;
```

ADC Programming

- Simple example 1
  - Running conversions using different methods



```
///////////////////////////////////
//
// adc_class_ex_1 project
//
// created 5/12/21 by tj
// rev 0
//
///////////////////////////////////
//
// ADC example file for class
//
// shows various ways to create and access the ADC functionality
//
///////////////////////////////////

#include "mbed.h"
#include "platform/mbed_thread.h"

#define T_WAIT 500   // in ms
#define ADC_REF 3.3  // default vref for ADC

// Global HARDWARE Objects
// Create an ADC object, attached to A3
AnalogIn EKG_3(A3);

int main(void){
    // splash
    printf("adc_class_ex_1 - example for EE2905\n");
    printf("Using Mbed OS version %d.%d.%d\n\n",
            MBED_MAJOR_VERSION, MBED_MINOR_VERSION, MBED_PATCH_VERSION);

    // working variables
    float adcval_f;
    int adcval_i;

    //Define and verify the reference for the ADC
    // required to use the read_voltage method
    EKG_3.set_reference_voltage(ADC_REF);
    printf("Using reference voltage: %fV\n", EKG_3.get_reference_voltage());
```

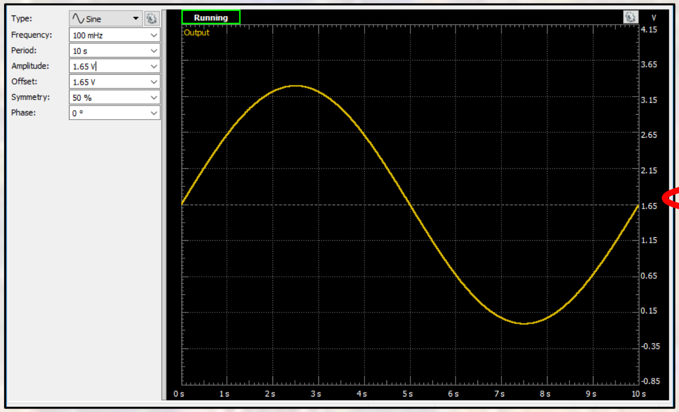


```
    // run through an endless series of conversions
    while(1) {
        // use the .read method for AnalogIn class
        // results are 0 - 1 (think percentage of max)
        adcval_f = EKG_3.read();
        printf("%f\t", adcval_f);

        // use the overload method for .read
        // results are 0 - 1 (think percentage of max)
        adcval_f = EKG_3;
        printf("%f\t", adcval_f);

        // use the .read_u16 method for AnalogIn class
        // results are 0 - max in binary (for our board 0 -)
        adcval_i = EKG_3.read_u16();
        printf("%i\t", adcval_i);

        // use the .read_voltage method for AnalogIn class
        // results are 0 - Vref (for our board 3.3v)
        adcval_f = EKG_3.read_voltage();
        printf("%f\t", EKG_3.get_reference_voltage());
        printf("%f\t", adcval_f);

        // code to print a simple curve
        int tmp_val;
        tmp_val = (int)(adcval_f * 10);
        for(int i = 0; i < tmp_val; i++)
            printf(" ");
        printf("|");

        // print newline and wait
        printf("\n");
        thread_sleep_for(T_WAIT);
    }// end while

    return 0;
}// end main
```

ADC Programming

- Simple example 1 - results

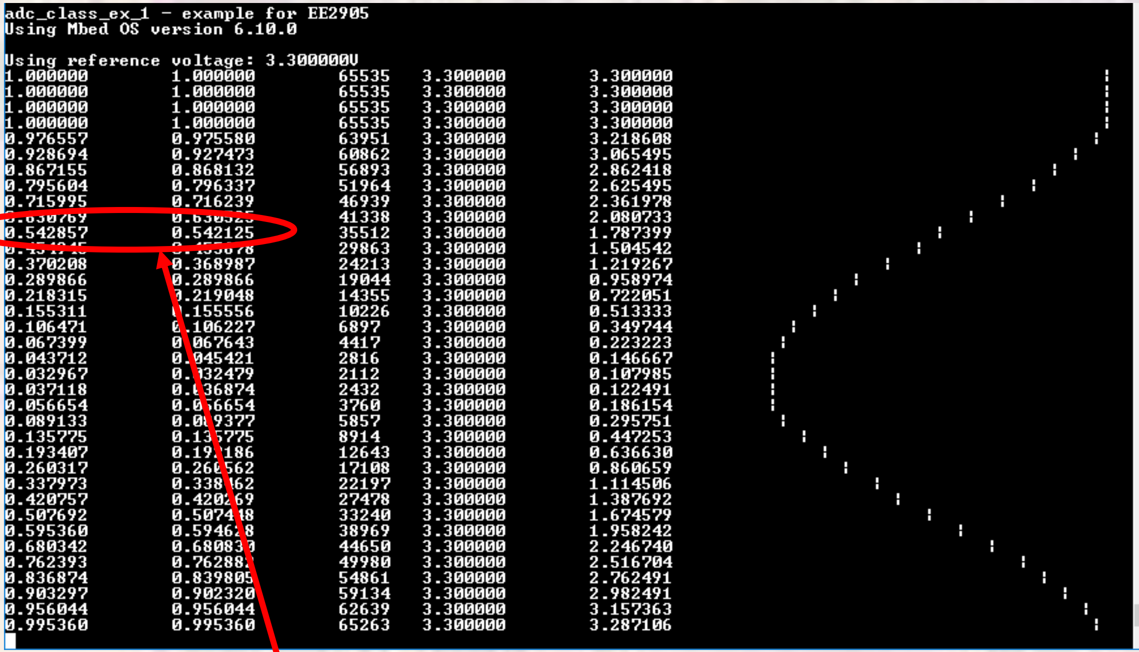
0.1Hz 0-3.3V sine wave input



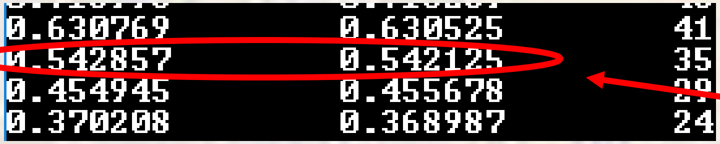
Type: Sine
Frequency: 100 mHz
Period: 10 s
Amplitude: 1.65 V
Offset: 1.65 V
Symmetry: 50 %
Phase: 0 °

output

.read overload .read_u16 Vref .read_voltage plot



```
adc_class_ex_1 - example for EE2905
Using Mbed OS version 6.10.0

Using reference voltage: 3.300000V
1.000000        1.000000        65535   3.300000        3.300000
1.000000        1.000000        65535   3.300000        3.300000
1.000000        1.000000        65535   3.300000        3.300000
1.000000        1.000000        65535   3.300000        3.300000
0.976557        0.975580        63951   3.300000        3.218608
0.928694        0.927473        60862   3.300000        3.065495
0.867155        0.868132        56893   3.300000        2.862418
0.795604        0.796337        51964   3.300000        2.625495
0.715995        0.716239        46939   3.300000        2.361978
0.630769        0.630525        41338   3.300000        2.080733
0.542857        0.542125        35512   3.300000        1.787399
0.454945        0.455678        29863   3.300000        1.504542
0.370208        0.368987        24213   3.300000        1.219267
0.289866        0.289866        19044   3.300000        0.958974
0.218315        0.219048        14355   3.300000        0.722051
0.155311        0.155556        10226   3.300000        0.513333
0.106471        0.106227        6897    3.300000        0.349744
0.067399        0.067643        4417    3.300000        0.223223
0.043712        0.045421        2816    3.300000        0.146667
0.032967        0.032479        2112    3.300000        0.107985
0.037118        0.036874        2432    3.300000        0.122491
0.056654        0.056654        3760    3.300000        0.186154
0.089133        0.089377        5857    3.300000        0.295751
0.135775        0.135775        8914    3.300000        0.447253
0.193407        0.192186        12643   3.300000        0.636630
0.260317        0.260562        17108   3.300000        0.860659
0.337973        0.338462        22197   3.300000        1.114506
0.420757        0.420269        27478   3.300000        1.387692
0.507692        0.507448        33240   3.300000        1.674579
0.595360        0.594628        38969   3.300000        1.958242
0.680342        0.680830        44650   3.300000        2.246740
0.762393        0.762882        49980   3.300000        2.516704
0.836874        0.839805        54861   3.300000        2.762491
0.903297        0.902320        59134   3.300000        2.982491
0.956044        0.956044        62639   3.300000        3.157363
0.995360        0.995360        65263   3.300000        3.287106
```

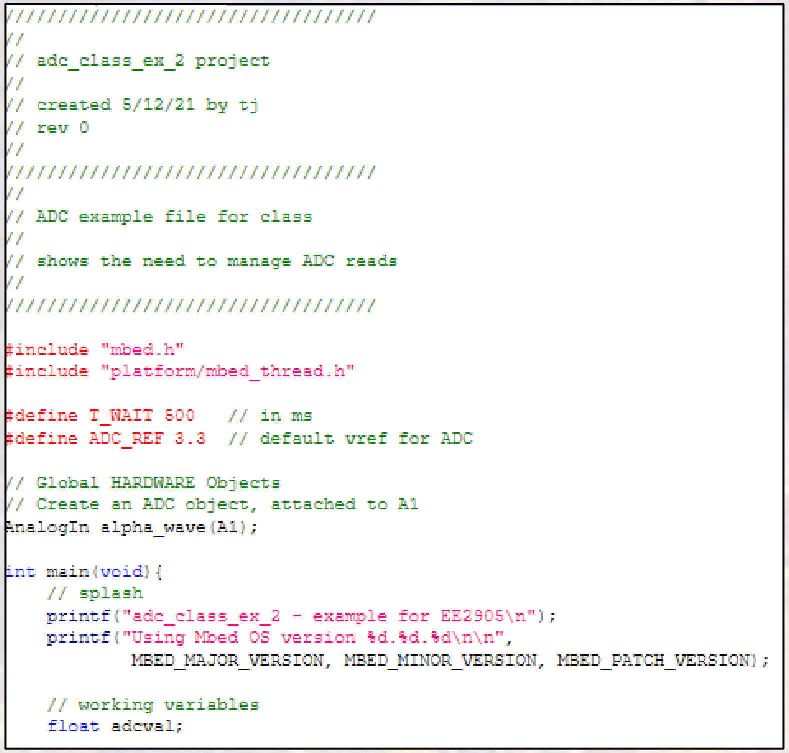


```
0.630769        0.630525        41
0.542857        0.542125        35
0.454945        0.455678        29
0.370208        0.368987        24
```

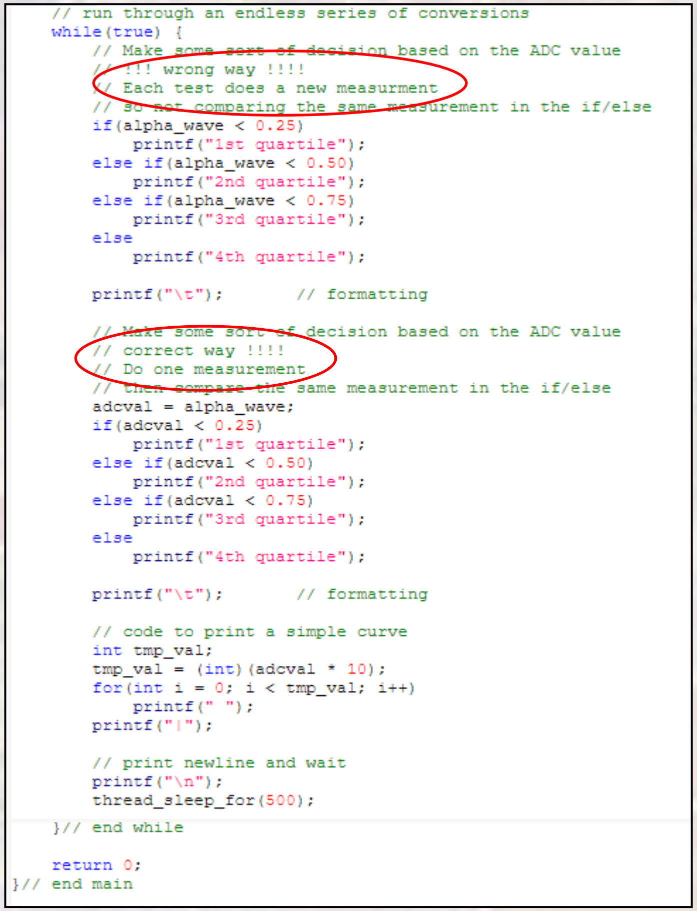
Note – different conversion values – WHY?

ADC Programming

- Simple example 2
  - Decisions based on ADC value
    - Wrong way and right way



```
///////////////////////////////////
//
// adc_class_ex_2 project
//
// created 5/12/21 by tj
// rev 0
//
///////////////////////////////////
//
// ADC example file for class
//
// shows the need to manage ADC reads
//
///////////////////////////////////

#include "mbed.h"
#include "platform/mbed_thread.h"

#define T_WAIT 500   // in ms
#define ADC_REF 3.3  // default vref for ADC

// Global HARDWARE Objects
// Create an ADC object, attached to A1
AnalogIn alpha_wave(A1);

int main(void){
    // splash
    printf("adc_class_ex_2 - example for EE2905\n");
    printf("Using Mbed OS version %d.%d.%d\n\n",
            MBED_MAJOR_VERSION, MBED_MINOR_VERSION, MBED_PATCH_VERSION);

    // working variables
    float adcval;
```

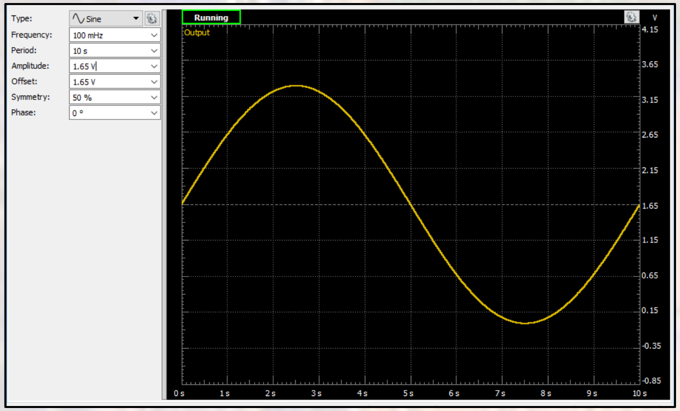


```
    // run through an endless series of conversions
    while(true) {
        // Make some sort of decision based on the ADC value
        // !!! wrong way !!!!
        // Each test does a new measurment
        // so not comparing the same measurement in the if/else
        if(alpha_wave < 0.25)
            printf("1st quartile");
        else if(alpha_wave < 0.50)
            printf("2nd quartile");
        else if(alpha_wave < 0.75)
            printf("3rd quartile");
        else
            printf("4th quartile");

        printf("\t");        // formatting

        // Make some sort of decision based on the ADC value
        // correct way !!!!
        // Do one measurement
        // then compare the same measurement in the if/else
        adcval = alpha_wave;
        if(adcval < 0.25)
            printf("1st quartile");
        else if(adcval < 0.50)
            printf("2nd quartile");
        else if(adcval < 0.75)
            printf("3rd quartile");
        else
            printf("4th quartile");

        printf("\t");        // formatting

        // code to print a simple curve
        int tmp_val;
        tmp_val = (int)(adcval * 10);
        for(int i = 0; i < tmp_val; i++)
            printf(" ");
        printf("|");

        // print newline and wait
        printf("\n");
        thread_sleep_for(500);
    }// end while

    return 0;
}// end main
```

ADC Programming

- Simple example 2 - results

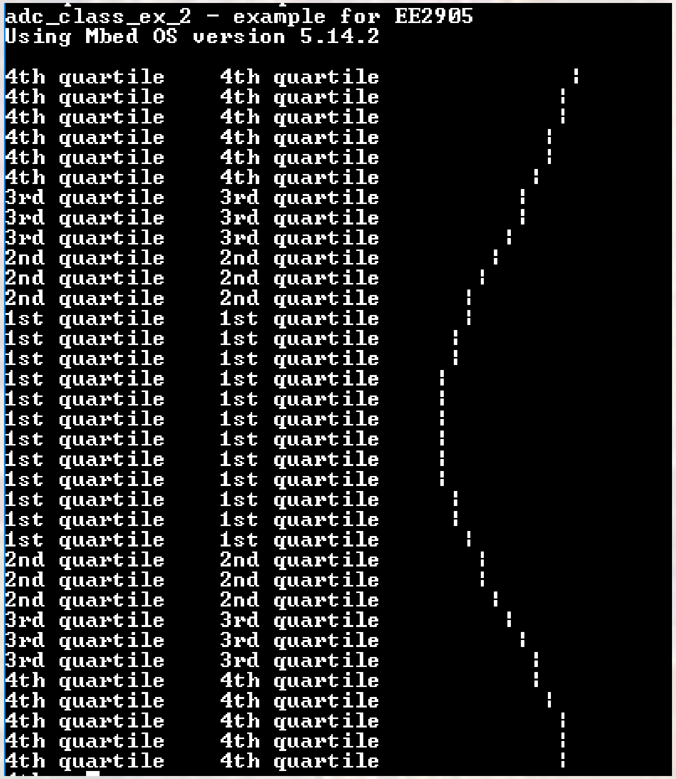
0.1Hz 0-3.3V sine wave input



Type: Sine
Frequency: 100 mHz
Period: 10 s
Amplitude: 1.65 V
Offset: 1.65 V
Symmetry: 50 %
Phase: 0 °

output

Up to 4 reads 1 read



```
adc_class_ex_2 - example for EE2905
Using Mbed OS version 5.14.2

4th quartile    4th quartile               ¦
4th quartile    4th quartile              ¦
4th quartile    4th quartile              ¦
4th quartile    4th quartile             ¦
4th quartile    4th quartile             ¦
4th quartile    4th quartile            ¦
3rd quartile    3rd quartile           ¦
3rd quartile    3rd quartile           ¦
3rd quartile    3rd quartile          ¦
2nd quartile    2nd quartile         ¦
2nd quartile    2nd quartile        ¦
2nd quartile    2nd quartile       ¦
1st quartile    1st quartile       ¦
1st quartile    1st quartile      ¦
1st quartile    1st quartile      ¦
1st quartile    1st quartile     ¦
1st quartile    1st quartile     ¦
1st quartile    1st quartile     ¦
1st quartile    1st quartile     ¦
1st quartile    1st quartile     ¦
1st quartile    1st quartile     ¦
1st quartile    1st quartile      ¦
1st quartile    1st quartile      ¦
1st quartile    1st quartile       ¦
2nd quartile    2nd quartile        ¦
2nd quartile    2nd quartile        ¦
2nd quartile    2nd quartile         ¦
3rd quartile    3rd quartile          ¦
3rd quartile    3rd quartile           ¦
3rd quartile    3rd quartile            ¦
4th quartile    4th quartile            ¦
4th quartile    4th quartile             ¦
4th quartile    4th quartile              ¦
4th quartile    4th quartile              ¦
4th quartile    4th quartile              ¦
```

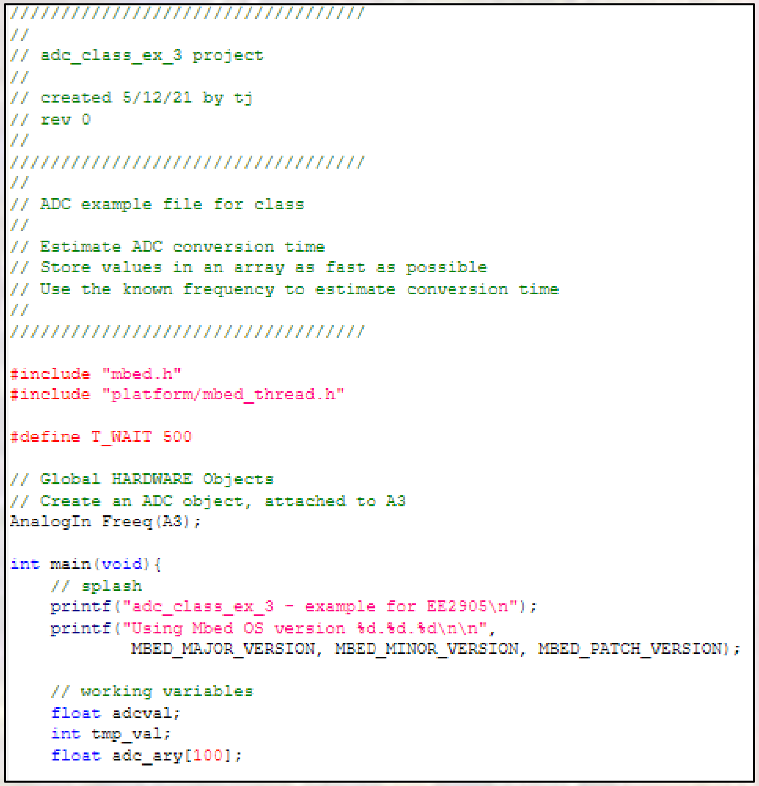
ADC Programming

- Simple example 3
  - Estimate the conversion time for the ADC
    - Use a known frequency input signal
    - Run the ADC as fast as possible
      - Do as little as possible between conversions
        - Store into an array
    - Print out the array and count how many conversions are done in a single period

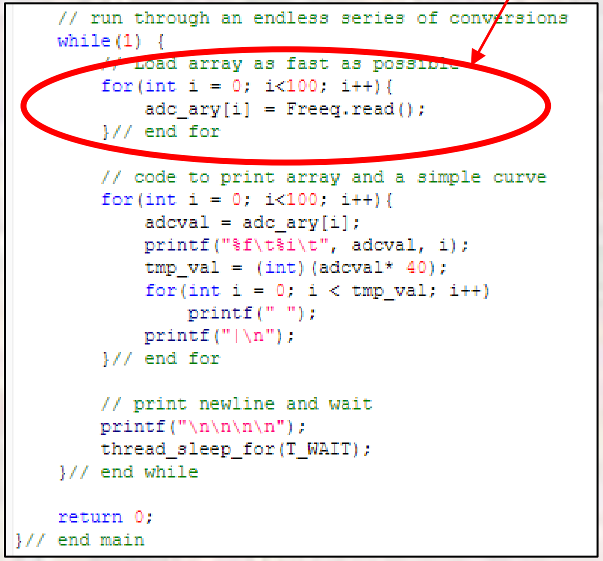
T/#conversions → time for 1 conversion

ADC Programming

- Simple example 3
  - Estimate the conversion time for the ADC



```
///////////////////////////////////
//
// adc_class_ex_3 project
//
// created 5/12/21 by tj
// rev 0
//
///////////////////////////////////
//
// ADC example file for class
//
// Estimate ADC conversion time
// Store values in an array as fast as possible
// Use the known frequency to estimate conversion time
//
///////////////////////////////////

#include "mbed.h"
#include "platform/mbed_thread.h"

#define T_WAIT 500

// Global HARDWARE Objects
// Create an ADC object, attached to A3
AnalogIn Freeq(A3);

int main(void){
    // splash
    printf("adc_class_ex_3 - example for EE2905\n");
    printf("Using Mbed OS version %d.%d.%d\n\n",
            MBED_MAJOR_VERSION, MBED_MINOR_VERSION, MBED_PATCH_VERSION);

    // working variables
    float adcval;
    int tmp_val;
    float adc_ary[100];
```



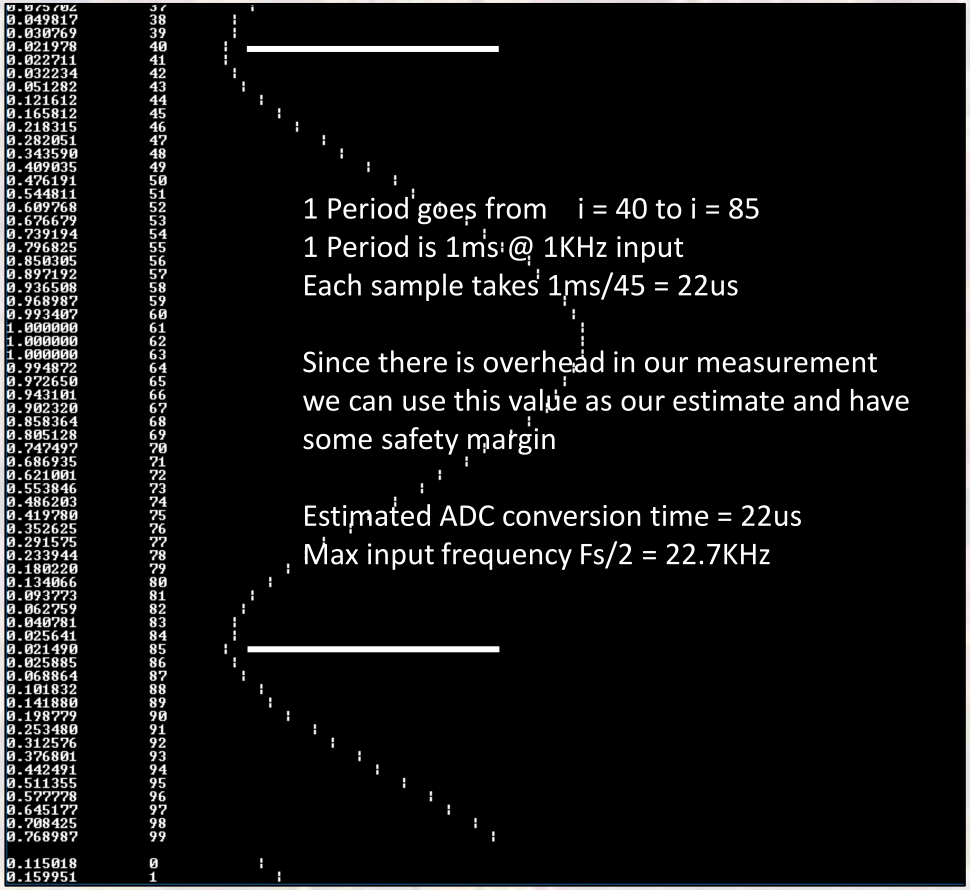
Minimal work done

```
    // run through an endless series of conversions
    while(1) {
        // load array as fast as possible
        for(int i = 0; i<100; i++){
            adc_ary[i] = Freeq.read();
        }// end for

        // code to print array and a simple curve
        for(int i = 0; i<100; i++){
            adcval = adc_ary[i];
            printf("%f\t%i\t", adcval, i);
            tmp_val = (int)(adcval* 40);
            for(int i = 0; i < tmp_val; i++)
                printf(" ");
            printf("|\n");
        }// end for

        // print newline and wait
        printf("\n\n\n\n");
        thread_sleep_for(T_WAIT);
    }// end while

    return 0;
}// end main
```

ADC Programming

- Simple example 3 - results

1KHz 0-3.3V sine wave input



```
0.075702     37
0.049817     38
0.030769     39
0.021978     40
0.022711     41
0.032234     42
0.051282     43
0.121612     44
0.165812     45
0.218315     46
0.282051     47
0.343590     48
0.409035     49
0.476191     50
0.544811     51
0.609768     52
0.676679     53
0.739194     54
0.796825     55
0.850305     56
0.897192     57
0.936508     58
0.968987     59
0.993407     60
1.000000     61
1.000000     62
1.000000     63
0.994872     64
0.972650     65
0.943101     66
0.902320     67
0.858364     68
0.805128     69
0.747497     70
0.686935     71
0.621001     72
0.553846     73
0.486203     74
0.419780     75
0.352625     76
0.291575     77
0.233944     78
0.180220     79
0.134066     80
0.093773     81
0.062759     82
0.040781     83
0.025641     84
0.021490     85
0.025885     86
0.068864     87
0.101832     88
0.141880     89
0.198779     90
0.253480     91
0.312576     92
0.376801     93
0.442491     94
0.511355     95
0.577778     96
0.645177     97
0.708425     98
0.768987     99

0.115018     0
0.159951     1
```

1 Period goes from i = 40 to i = 85
1 Period is 1ms @ 1KHz input
Each sample takes 1ms/45 = 22us

Since there is overhead in our measurement we can use this value as our estimate and have some safety margin

Estimated ADC conversion time = 22us
Max input frequency Fs/2 = 22.7KHz

The actual ADC is capable of running much faster than this (5Msps)

Our Nucleo/mbed implementation has set some default parameters that lead to this result