Performing 8 MSPS ADC Code through M483

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This sample code uses M483KGCAE2A to perform 8 MSPS ADC by EADC0 and EADC1 sampling alternately.
BSP Version	M480 BSP CMSIS V3.05.001
Hardware	NuMaker-M483KG Ver1.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Overview

The M483KGCAE2A has two 5MSPS ADC — EADC0 and EADC1. The sample rate can be doubled if EADC0 and EADC1 do conversion just alternately as Figure 1-1.

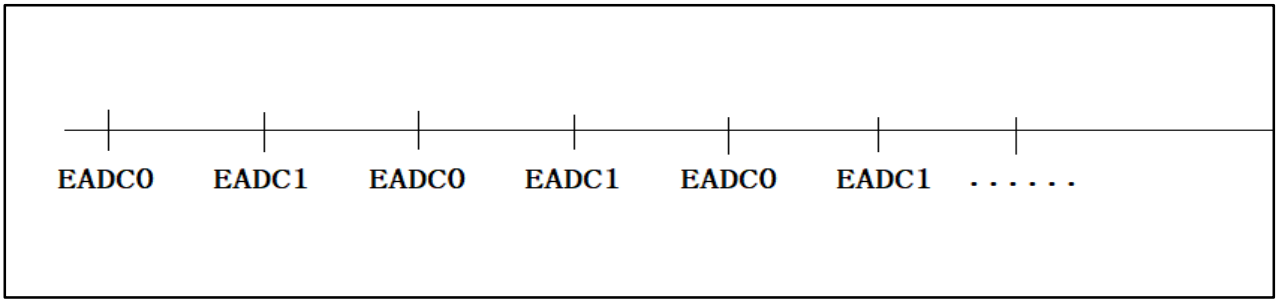


Figure 1-1 EADC0 and EADC1 Alternately Convert

This sample code performs 8 MSPS ADC with M483KGCAE2A. 8 MHz trigger signal is from 1 MHz PWM that triggers evenly 8 times within one period. Such high rate data can only be transferred by PDMA. For that reason, EADC0, EADC1, PWM and PDMA is configured working co-operation in this example.

This example can be used for ultrasonic water meter application as Figure 1-2. 1 MHz Ultrasonic is emitted prior to echo data gathering, and then calculates the accurate time of echo.

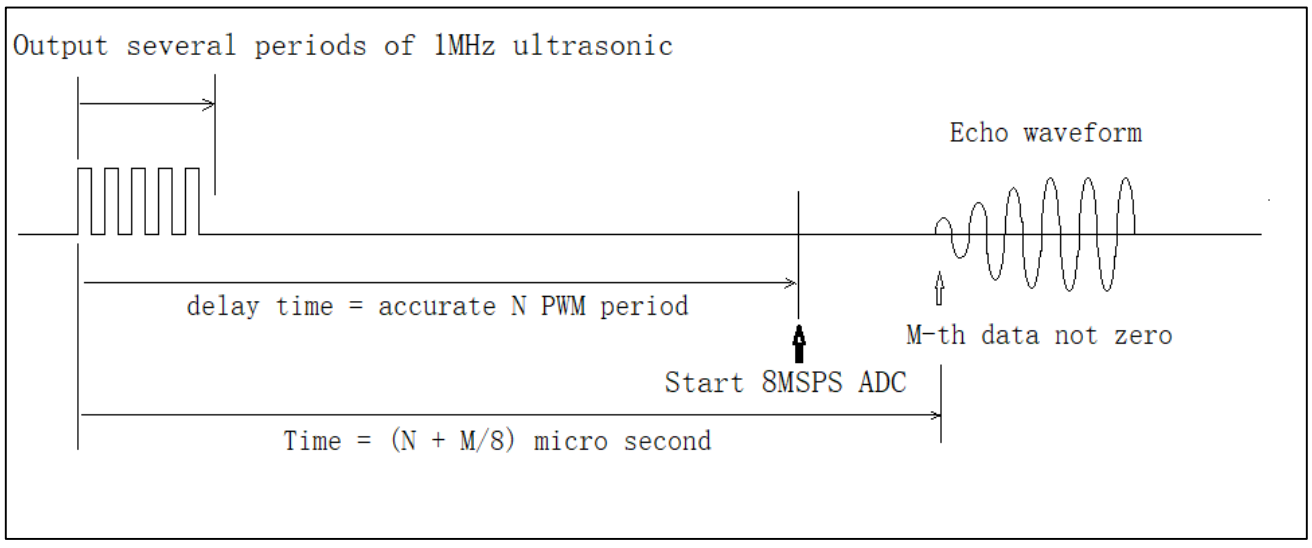


Figure 1-2 Relationship of Time

1.1 Demo Result

Figure 1-3 shows the PWM waveforms measured from PC12_EPWM1_CH0 and PC11_EPWM1_CH1, which are used to drive the ultrasonic transducer.

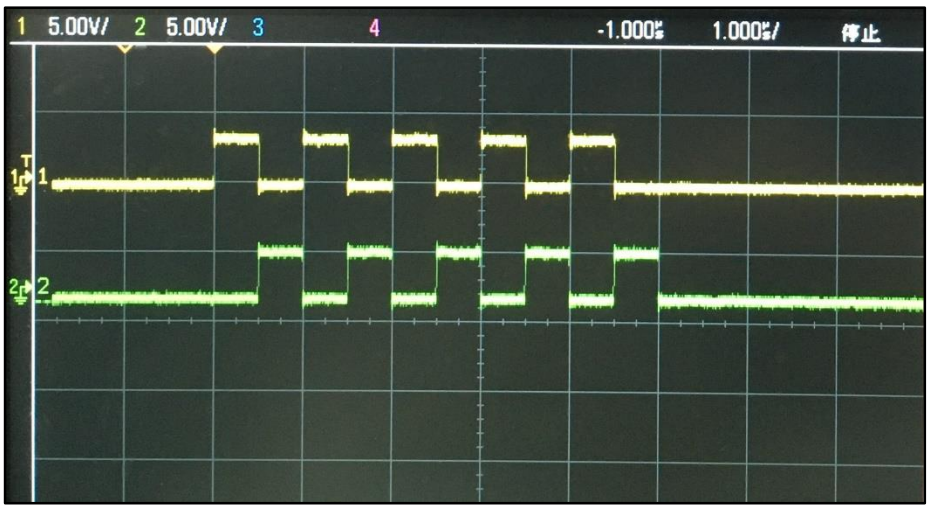


Figure 1-3 Output Just 5 Periods PWM Waveform

After 1000 PWM periods, just equal to1000 ms, EADC0 and EADC1 start to convert and gather 200*4 data. So, as shown in Figure 1-4, gathering data is finished after 1200 ms.

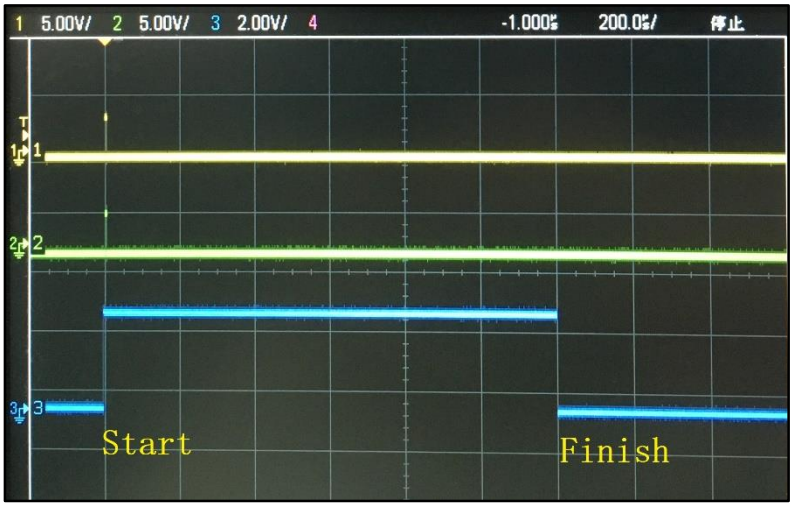


Figure 1-4 Finish After 1200 Micro Seconds

Figure 1-5 shows the 1 MHz signal generated from the waveform generator, which is the input of PB15_EADC0_CH15_EADC1_CH15 while this example is being tested.



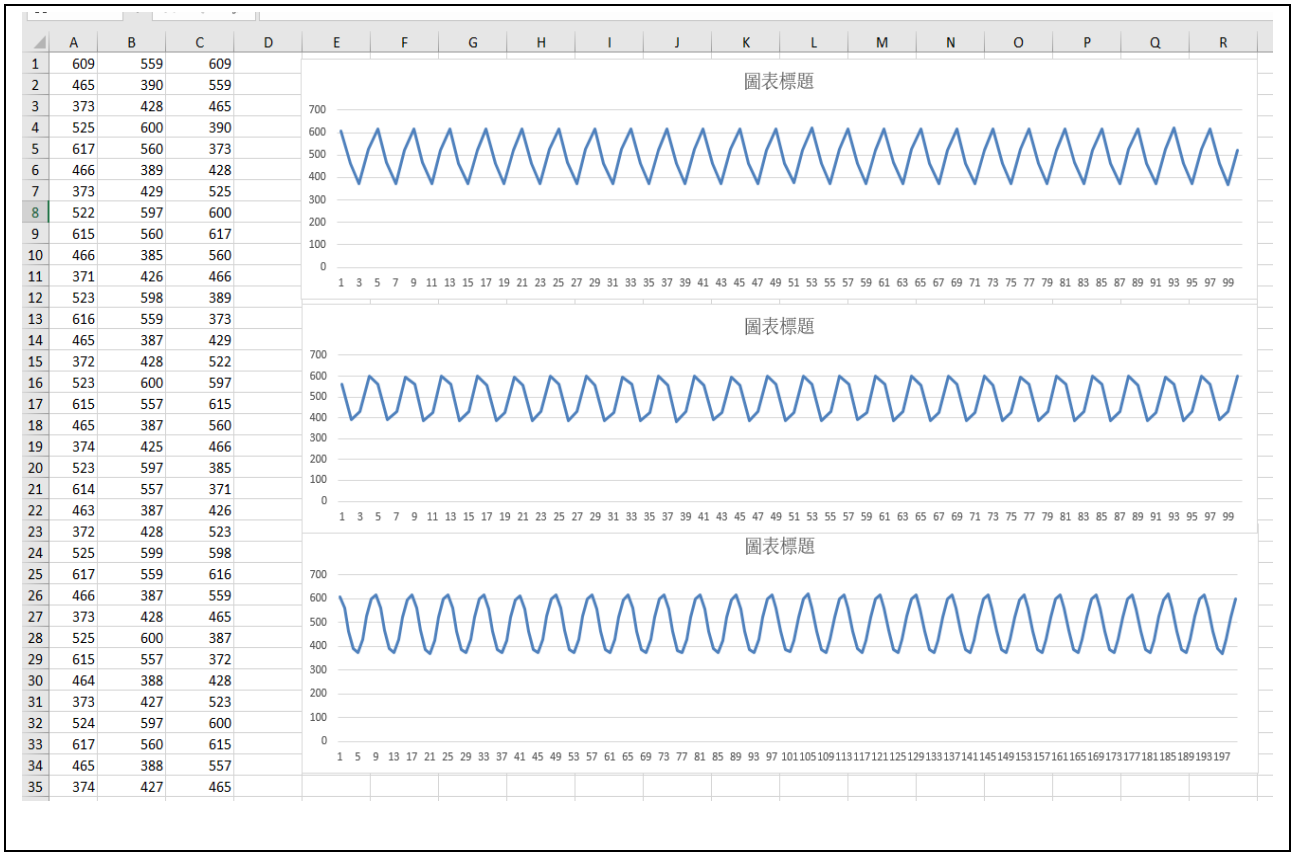
Figure 1-5 1MHz Sine Waveform

Figure 1-6 shows data from UART tools on PC. M483 UART0 prints only 100 bytes from EADC0, and then prints 100 bytes from EADC1. The third group data is from EADC0 and EADC1 in turn alternately.

```
 609 465 373 525 617 466 373 522 615 466 371 523 616 465 372 523 615 465 374 523 614 463 372 525
617 466 373 525 615 464 373 524 617 465 374 523 618 465 372 524 617 463 372 524 616 464 374 523 615
465 377 523 619 465 373 524 616 465 373 523 618 466 373 525 616 465 372 523 615 463 372 523 615 465
373 524 616 465 373 522 615 466 372 523 615 464 374 524 617 465 372 525 619 465 374 525 618 463 368
523
 559 390 428 600 560 389 429 597 560 385 426 598 559 387 428 600 557 387 425 597 557 387 428 599
559 387 428 600 557 388 427 597 560 388 427 601 558 383 428 599 558 389 426 597 557 387 429 599 560
386 423 598 559 388 428 598 559 389 426 599 559 388 429 598 558 387 426 599 562 387 426 599 558 387
427 597 559 388 428 598 559 388 429 598 559 388 428 599 558 387 425 597 559 387 428 599 559 389 428
599
 609 559 465 390 373 428 525 600 617 560 466 389 373 429 522 597 615 560 466 385 371 426 523 598
616 559 465 387 372 428 523 600 615 557 465 387 374 425 523 597 614 557 463 387 372 428 525 599 617
559 466 387 373 428 525 600 615 557 464 388 373 427 524 597 617 560 465 388 374 427 523 601 618 558
465 383 372 428 524 599 617 558 463 389 372 426 524 597 616 557 464 387 374 429 523 599 615 560 465
386 377 423 523 598 619 559 465 388 373 428 524 598 616 559 465 389 373 426 523 599 618 559 466 388
373 429 525 598 616 558 465 387 372 426 523 599 615 562 463 387 372 426 523 599 615 558 465 387 373
427 524 597 616 559 465 388 373 428 522 598 615 559 466 388 372 429 523 598 615 559 464 388 374 428
524 599 617 558 465 387 372 425 525 597 619 559 465 387 374 428 525 599 618 559 463 389 368 428 523
599
```

Figure 1-6 Data On COM Tool

Figure 1-7 shows 3 waveforms integrated by 3 groups of data, including 4 MSPS, 4 MSPS and 8 MSPS respectively.



	A	B	C	D
1	609	559	609	
2	465	390	559	
3	373	428	465	
4	525	600	390	
5	617	560	373	
6	466	389	428	
7	373	429	525	
8	522	597	600	
9	615	560	617	
10	466	385	560	
11	371	426	466	
12	523	598	389	
13	616	559	373	
14	465	387	429	
15	372	428	522	
16	523	600	597	
17	615	557	615	
18	465	387	560	
19	374	425	466	
20	523	597	385	
21	614	557	371	
22	463	387	426	
23	372	428	523	
24	525	599	598	
25	617	559	616	
26	466	387	559	
27	373	428	465	
28	525	600	387	
29	615	557	372	
30	464	388	428	
31	373	427	523	
32	524	597	600	
33	617	560	615	
34	465	388	557	
35	374	427	465	

Figure 1-7 Waveforms of 3 Groups Data

2 Code Description

Because one PWM channel can trigger only one point, 8-channel PWM must be configured and worked synchronously. They trigger ADC at respective point tick. Figure 2-1 shows time sequence.

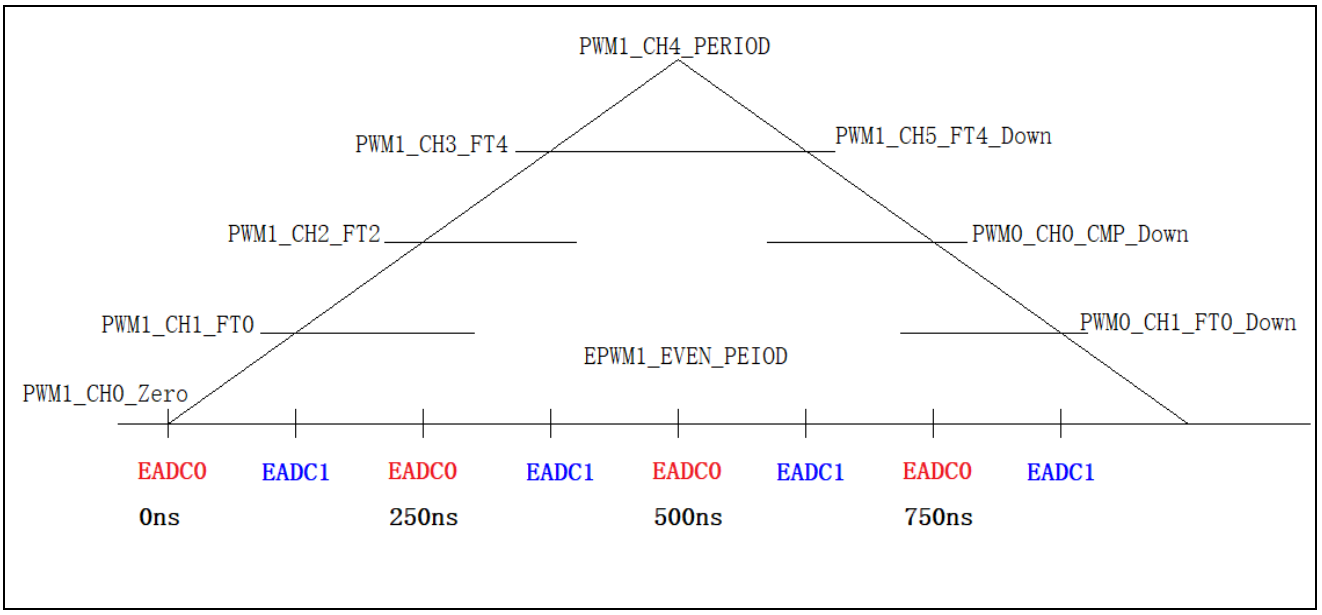


Figure 2-1 Time Sequence of EAD0 and EADC1

After N periods of PWM, EPWM1P1 interrupt occurs and PWM starts to trigger ADC.

Time has gone 100 ns before executing the first sentence of interrupt program. And it will take about 50 ns executing one sentence. However, the first trigger point is at 125 ns tick. In order to ensure the time of data is accurate, rear 4 trigger point is configured in advance.

Due to above reason, the time of first data must be added 0.5 ms. So, if the M-th data is not 0, the delay time of echo is calculated from formula:

Time = (N+0.5+M/8) ms

```
void EPWM1P1_IRQHandler(void)
{
    int8_t const Sample_Add = 3 ;
      // 100ns has gone
    EADC->SCTL[0]  = (Sample_Add << 24) | EADC_EPWM1TG4_TRIGGER | 15 ;   // takes 50ns
every sentence
    EADC1->SCTL[0] = (Sample_Add << 24) | EADC_EPWM1TG5_TRIGGER | 15 ;

    EADC->SCTL[1]  = (Sample_Add << 24) | EADC_EPWM0TG0_TRIGGER | 15 ;
    EADC1->SCTL[1] = (Sample_Add << 24) | EADC_EPWM0TG1_TRIGGER | 15 ;
                                                                  // (100+200)ns has gone
    EADC->SCTL[2]  = (Sample_Add << 24) | EADC_EPWM1TG0_TRIGGER | 15 ;
    EADC1->SCTL[2] = (Sample_Add << 24) | EADC_EPWM1TG1_TRIGGER | 15 ;
                                                                     // 400ns has gone
    //  PH4 ^= 1;                                                  // Only for test
    EPWM1->AINTEN  = 0 ;      //  Disabling Interrupt cannot be the last sentence
    EADC->SCTL[3]  = (Sample_Add << 24) | EADC_EPWM1TG2_TRIGGER | 15 ;
    EADC1->SCTL[3] = (Sample_Add << 24) | EADC_EPWM1TG3_TRIGGER | 15 ;
}
```

EPWM initialization code is as follows.

```
// One PWM can trigger one time every period. So configures 8 PWM channels to trigger 8
times in one period
void EPWM_Initial(void)
{
    int32_t Cnt ;

    EPWM_ConfigOutputChannel(EPWM1, 0, 1000000, 50);  // Configure 8ch PWM output 1MHz,
duty=50%
    EPWM_ConfigOutputChannel(EPWM1, 1, 1000000, 50);
    EPWM_ConfigOutputChannel(EPWM1, 2, 1000000, 50);
    EPWM_ConfigOutputChannel(EPWM1, 3, 1000000, 50);
    EPWM_ConfigOutputChannel(EPWM1, 4, 1000000, 50);
    EPWM_ConfigOutputChannel(EPWM1, 5, 1000000, 50);

    EPWM_ConfigOutputChannel(EPWM0, 0, 1000000, 50);
    EPWM_ConfigOutputChannel(EPWM0, 1, 1000000, 50);

    EPWM1->CTL0   = 0 ;
    EPWM1->CTL1   = 0x0AAA ;                          // up-down count, center-align
    EPWM1->WGCTL0 = 0x05550555 ;            // output low at Zero and center at first
    EPWM1->WGCTL1 = 0 ;
    // Now PWM all output low
    EPWM0->CTL0   = 0 ;
    EPWM0->CTL1   = 0x0AAA ;
    EPWM0->WGCTL0 = 0x05550555 ;
    EPWM0->WGCTL1 = 0 ;
    // Half period value can be divided by 4. So a period can be divided 8 stages.
    Cnt = 24 * 4 * 1 ;                          // Period =96 · Frequency=192M/(96*2)=1M
    EPWM_SET_CNR(EPWM1, 0, Cnt);
    EPWM_SET_CNR(EPWM1, 1, Cnt);
    EPWM_SET_CNR(EPWM1, 2, Cnt);
    EPWM_SET_CNR(EPWM1, 3, Cnt);
    EPWM_SET_CNR(EPWM1, 4, Cnt);
    EPWM_SET_CNR(EPWM1, 5, Cnt);
    EPWM_SET_CNR(EPWM0, 0, Cnt);
    EPWM_SET_CNR(EPWM0, 1, Cnt);

    EPWM1->FTCMPDAT[0] = Cnt / 4 ;                // Configure tick triggering EADC0/1
    EPWM1->FTCMPDAT[1] = Cnt / 2 ;
    EPWM1->FTCMPDAT[2] = Cnt * 3 / 4 ;

    EPWM0->FTCMPDAT[0] = Cnt / 4 ;
    EPWM_SET_CMR(EPWM0, 0, Cnt / 2);

    EPWM1->EADCTS0 = ((0x80 +  EPWM_TRG_ADC_CH_4_FREE_CMP_UP) << 24)  //configure PWM
channel triggering ADC
                   | ((0x80 + EPWM_TRG_ADC_CH_2_FREE_CMP_UP) << 16)
                   | ((0x80 + EPWM_TRG_ADC_CH_0_FREE_CMP_UP) << 8)
                   | (0x80 +  EPWM_TRG_ADC_EVEN_ZERO) ;

    EPWM1->EADCTS1 = ((0x80 + EPWM_TRG_ADC_CH_4_FREE_CMP_DOWN) << 8)
                   | (0x80 + EPWM_TRG_ADC_EVEN_PERIOD) ;

    EPWM0->EADCTS0 = ((0x80 + EPWM_TRG_ADC_CH_0_FREE_CMP_DOWN) << 8)
                   | (0x80 + EPWM_TRG_ADC_EVEN_COMPARE_DOWN) ;

    EPWM_ENABLE_TIMER_SYNC(EPWM1, 0x3F, EPWM_SSCTL_SSRC_EPWM1);   // 8PWM channel active
synchronous
    EPWM_ENABLE_TIMER_SYNC(EPWM0, 0x03, EPWM_SSCTL_SSRC_EPWM1);

    EPWM_EnableOutput(EPWM1, 0x3F);
    EPWM_EnableOutput(EPWM0, 0x03);                   // Enable output of PWM module
```

```
    NVIC_SetPriority(EPWM1P1_IRQn, 1) ;                     // Enable NVIC of PWM1_CH1
    NVIC_ClearPendingIRQ(EPWM1P1_IRQn);
    NVIC_EnableIRQ(EPWM1P1_IRQn);                           // enable PWM1_CH1 NVIC

    NVIC_SetPriority(EPWM1P0_IRQn, 2) ;
    NVIC_ClearPendingIRQ(EPWM1P0_IRQn);
    NVIC_EnableIRQ(EPWM1P0_IRQn);
}
```

After PWM outputs configured period waveform, EPWM1P0 interrupt occurs and all PWM channels output low. Ultrasonic transducer stops working.

```
void EPWM1P0_IRQHandler(void)
{
    EPWM1->AINTEN ^= 1 ;              // Disabling interrupt cannot be put at last
    EPWM1->WGCTL0 = 0x05550555 ;      // Zero&center all output low
}
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M480_BSP_CMSIS_V3.05.001
- IDE version
  - Keil uVersion 5.26

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M483KG V1.1
- Pin Connect

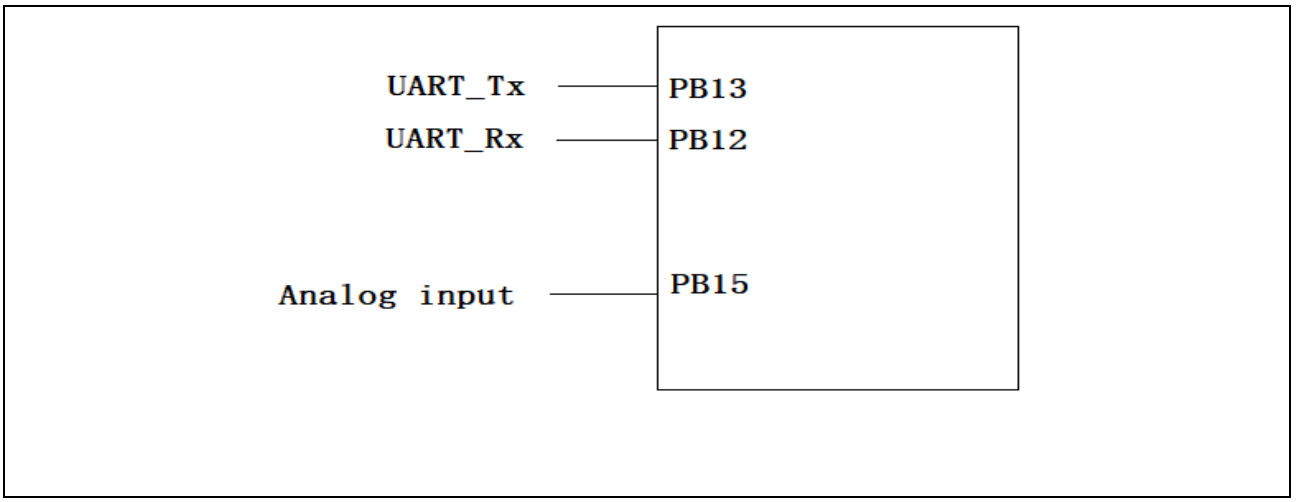


Figure 3- 1 Pin Connect

4 Directory Information

The directory structure is shown below.

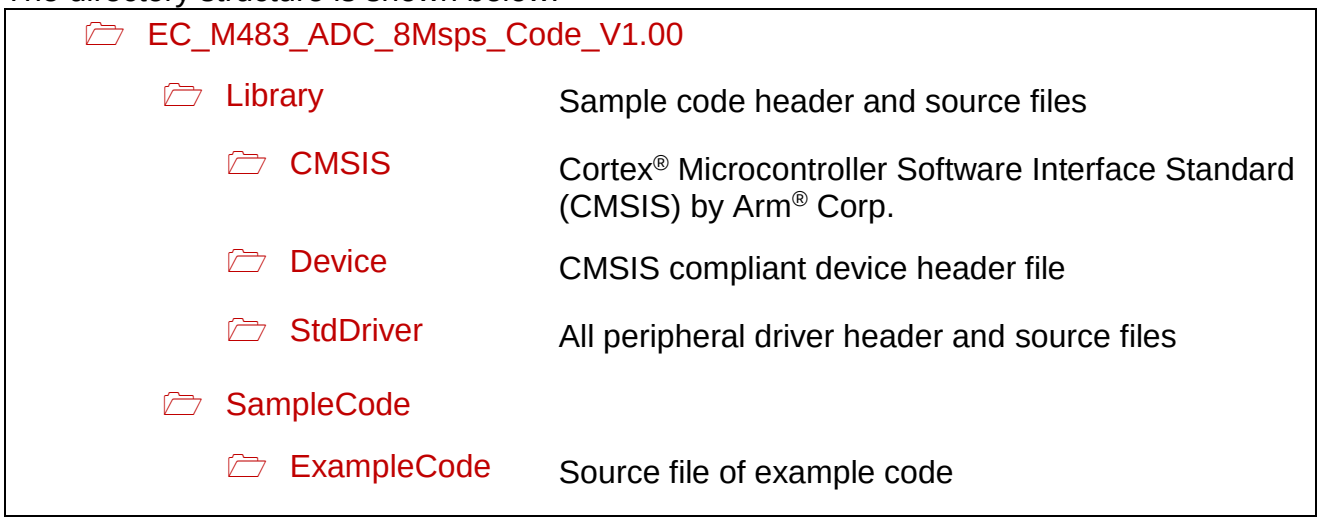


Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M483_ADC_8Msps.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2021.09.01	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.