Understanding and Managing Jitter in an Application

AN5823

Overview

The purpose of this document is to provide an understanding of clock jitter and its impact on applications that utilize PWM signals in control loops. It also provides information on how to understand where jitter comes from, how to measure it, and how to design an application with jitter in mind.

Table of Contents

Overview 1

1. Introduction 3
   - 1.1. What is Jitter? 3
   - 1.2. Setting Expectations 3

2. Understanding Jitter 4
   - 2.1. Period Jitter 4
   - 2.2. Cycle-to-Cycle Jitter 5
   - 2.3. Time Interval Error (TIE) 6

3. Jitter Metrics 8
   - 3.1. Min/Max 8
   - 3.2. Standard Deviation (σ) 8

4. Types of Jitter 10
   - 4.1. Random Jitter 10
   - 4.2. Deterministic Jitter 10

5. Measurement Techniques 12
   - 5.1. Measurement Steps 12

6. Mitigating Jitter 13
   - 6.1. Clock Source Jitter 13
   - 6.2. Phase-Locked Loop (PLL) 13
   - 6.3. Power supply 13
   - 6.4. Circuit Adjustments 13
   - 6.5. Components 13
   - 6.6. Other Factors 14

7. Example Clocking Configuration for Best Performance 18

8. Device-Specific Examples 19
   - 8.1. dsPIC33E 19
   - 8.2. dsPIC33C 22
   - 8.3. dsPIC33A 25

9. Conclusion 33

10. Revision History 34

Microchip Information 35
- Trademarks 35
- Legal Notice 35
- Microchip Devices Code Protection Feature 35

1. Introduction

1.1 What is Jitter?

Jitter is defined as a timing deviation from a reference signal. Jitter is created by small deviations in the timing or threshold of electronic circuits. These deviations can be exacerbated by environmental and system factors, including supply noise and signal integrity. The types, components and quantification of jitter are discussed in the following sections. As the noise in a system increases, the amount of corresponding jitter can become detrimental to some applications.

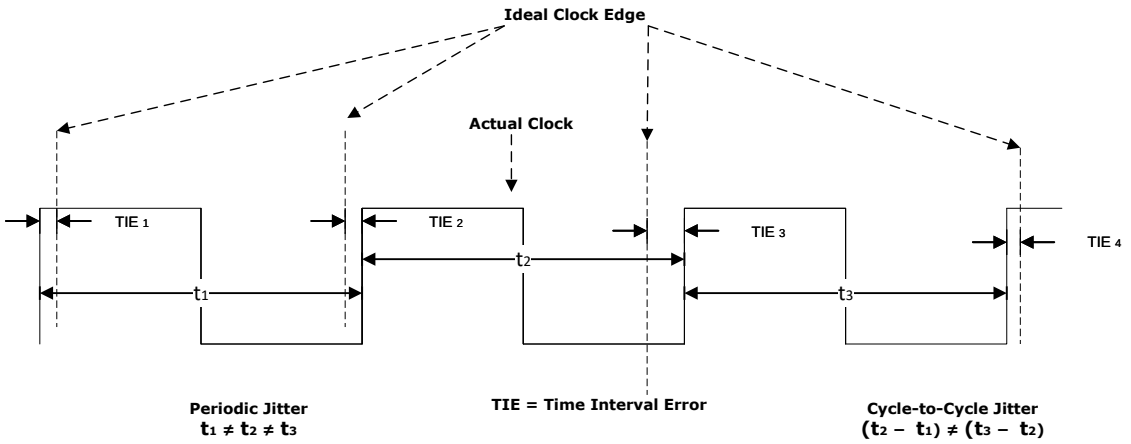
1.2 Setting Expectations

It is important to understand that jitter can never be zero, but it can be mitigated and reduced. Some products offer edge resolution that may be smaller than the amount of system jitter. This may or may not present problems depending on the type of application. Control loops that utilize averaging or other filters can be less susceptible to jitter-induced errors. Cycle-by-cycle controlled systems can tolerate jitter but may incur ripple with non-uniform distributions. Some of the test conditions in which resolution or jitter are specified may be in an ideal environment; in a real application, transients are present from power switching and will raise the noise floor and resulting system jitter.

2. Understanding Jitter

Jitter can be observed and measured in different ways, such as period jitter, cycle-to-cycle jitter and long-term jitter. These methods are all related and can provide another layer of understanding of exactly what components are present and how they can affect an application. The following figure shows the clock source jitter metrics.

Figure 2-1. Clock Source Jitter Metrics

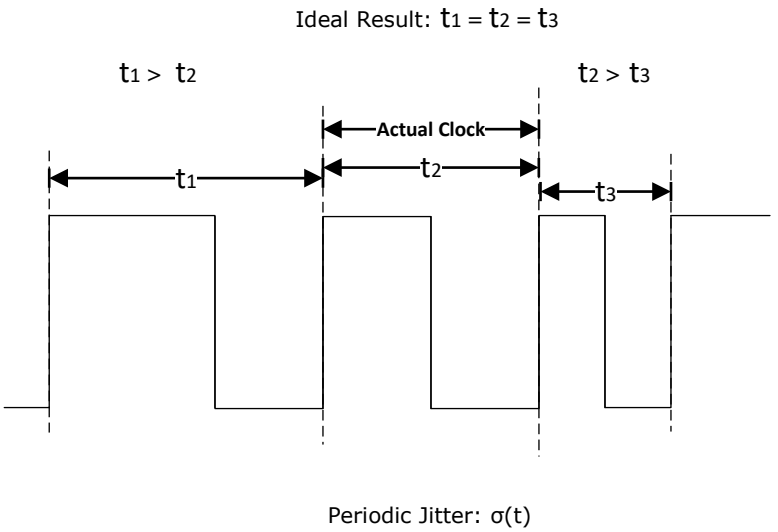


2.1 Period Jitter

Period jitter occurs when any single clock period does not match the expected clock value. When jitter affects a timing signal, the period could be shifted in either direction, meaning the period could be either larger or smaller than the ideal. With jitter, the period will change every single cycle. In Figure 2-2, t1 is a longer-than-expected pulse, while t2 is the exact desired clock frequency, and t3 is a shorter-than-expected pulse. In an ideal application, all three pulses are the same length.

The following figure shows the period jitter:

Figure 2-2. Period Jitter



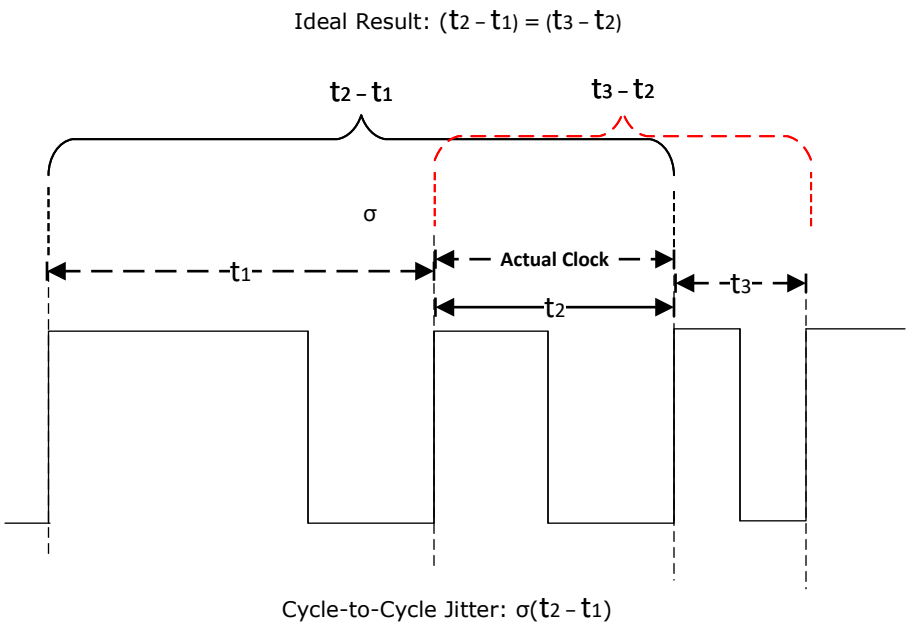
2.2 Cycle-to-Cycle Jitter

Cycle-to-cycle jitter is the period difference between neighboring clock cycles. This is the relationship between different cycles of period jitter, as represented in Figure 2-3 and described by the following equation:

Equation 2-1.

$$cycle\text{-}to\text{-}cycle\ jitter = \sigma\ (t2 - t1)$$

Figure 2-3. Cycle-to-Cycle Jitter

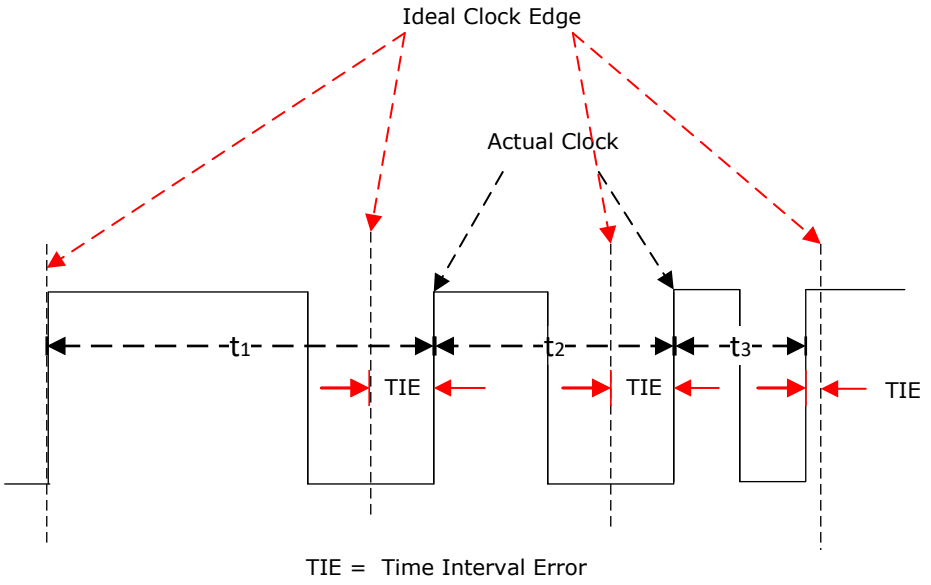


2.3 Time Interval Error (TIE)

The resulting difference from one time measurement to another is called the Time Interval Error (TIE). In Figure 2-4, none of the three clock pulses occur at the expected time. This is because t1's period was longer than the expected frequency, shifting the entire waveform by a set amount of time. This time error is TIE and can have a positive or negative value. For example, t1 to t2 and t2 to t3 both have a positive TIE, but since t3 has a shorter than normal period (negative TIE), the total TIE is averaged out. TIE can also have different time scales in relation to the clock. Slow TIE can manifest as many cycles with small errors and take a long time to accumulate a signification error in either direction. The envelope timing and the magnitude of TIE are independent and can affect system performance in different ways. With random jitter, these shifts will still occur but remain manageable.

The following figure shows the time interval error.

Figure 2-4. Time Interval Error



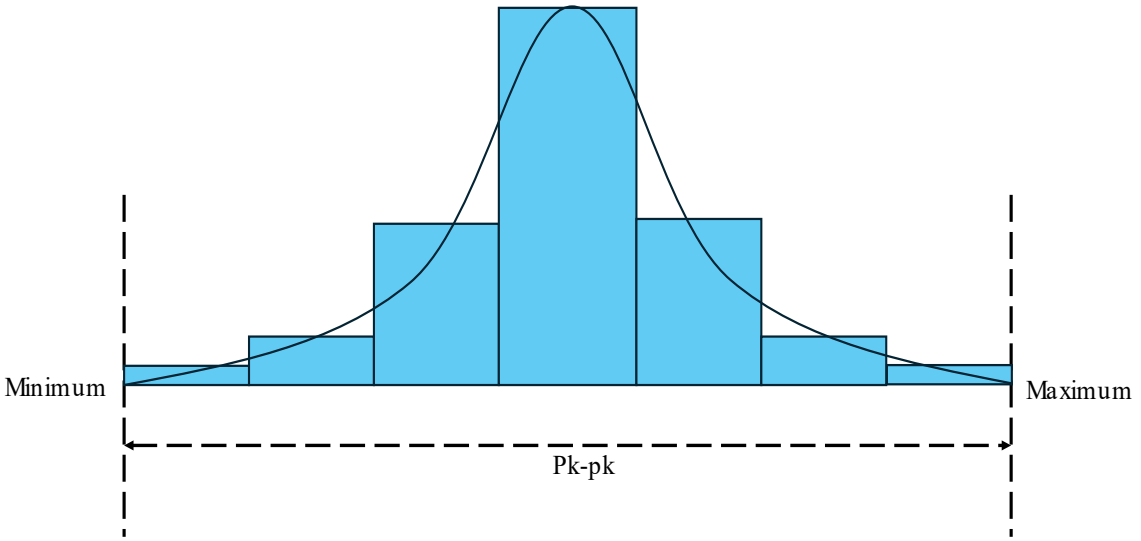
3. Jitter Metrics

There are several metrics for measuring clock timing jitter. Jitter data is often represented as a histogram, where x is the timing difference from the ideal clock period, and y is the number of times a period value is measured.

3.1 Min/Max

The minimum and maximum in a histogram are the two furthest points on either side, with the minimum being the lowest period measurement and the maximum being the highest. The peak-to-peak measurement is the difference between the min and max. These values are taken into account when considering the full distribution of jitter in an application, but the peak-to-peak value will theoretically grow infinitely as the number of counts in a histogram increases (due to random unbounded jitter) and should not be the main factor in evaluating jitter. Instead, evaluating the mean and standard deviation of the histogram is a more accurate form of jitter analysis. The following figure shows the minimum and maximum in a histogram.

Figure 3-1. Min/Max Histogram

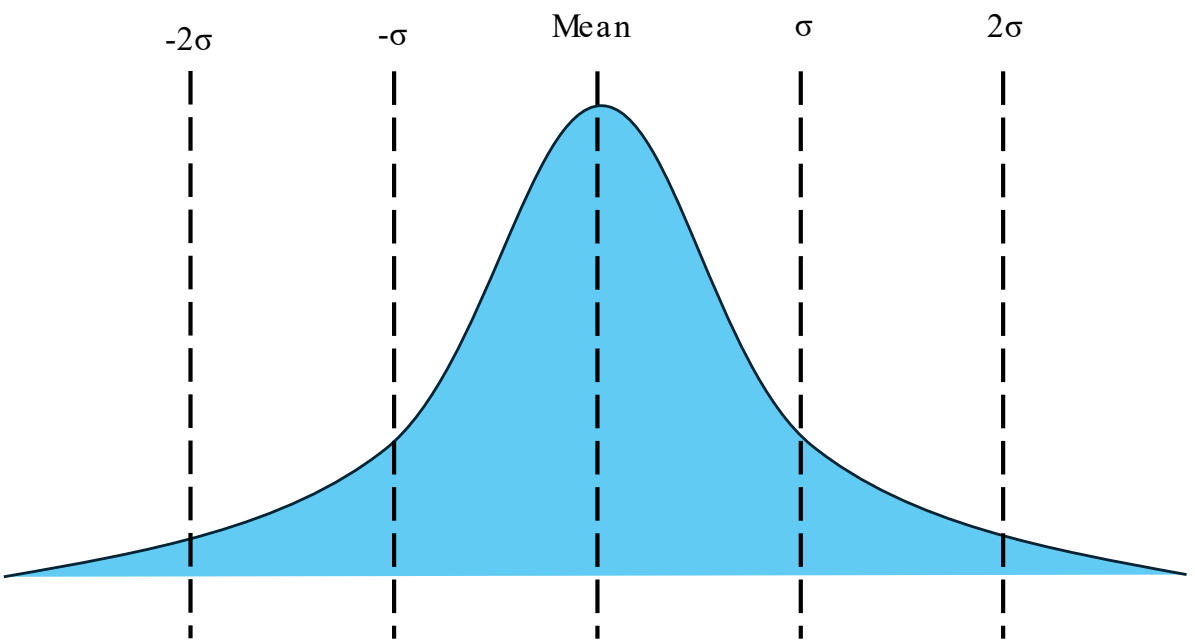


3.2 Standard Deviation (σ)

Standard deviation is a statistical measure that quantifies the amount of variation or dispersion in a set of data values. If the calculated standard deviation is low, it means the majority of data points are close to the mean, and a higher standard deviation means higher dispersion. In a normal distribution where the data appears Gaussian, approximately 68% of data falls within the bounds of one standard deviation, or the space between -1σ and 1σ. Standard deviation is derived from the mean of the data; these two values are the most important metrics when analyzing jitter.

The following figure shows the standard deviation and mean.

Figure 3-2. Standard Deviation and Mean

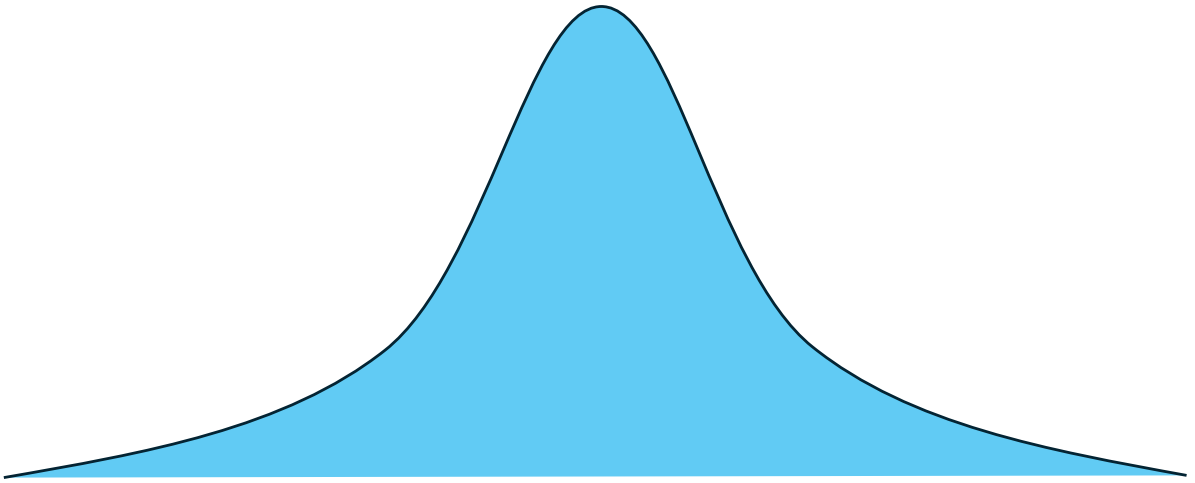


4. Types of Jitter

4.1 Random Jitter

Jitter falls under two classifications: random and deterministic. In many cases, it can be caused by a noisy power supply, poor hardware design, thermal noise, or cross-talk. Jitter appears in every system as a deviation in the period of a signal. In an ideal case where only random jitter is present, the histogram of clock periods will appear Gaussian, or as a bell curve, as shown in the following figure.

Figure 4-1. Random Jitter



A Gaussian distribution measurement result indicates that the jitter is “random” and the cause cannot be determined or eliminated. Random jitter is unbounded, meaning the tails of the Gaussian distribution extend to infinity, but the probability of large deviations decreases exponentially.

4.2 Deterministic Jitter

In a system with both random and deterministic jitter, the peak of the curve may be shifted to either side, or there may even be multiple peaks.

Figure 4-2. Deterministic Jitter - Shifted Peak

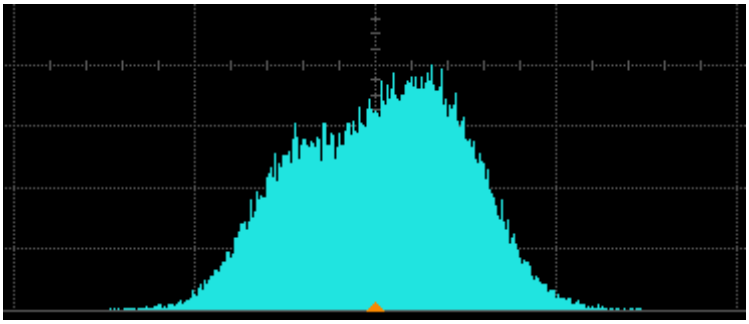
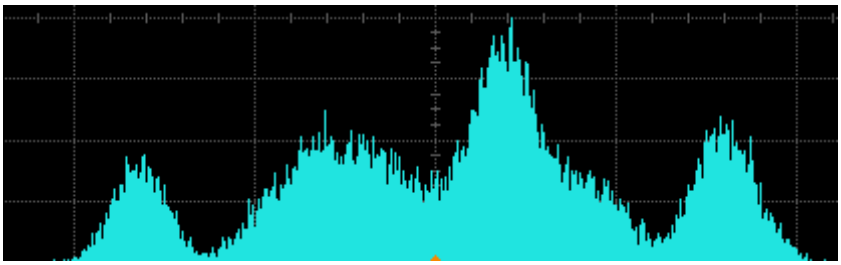
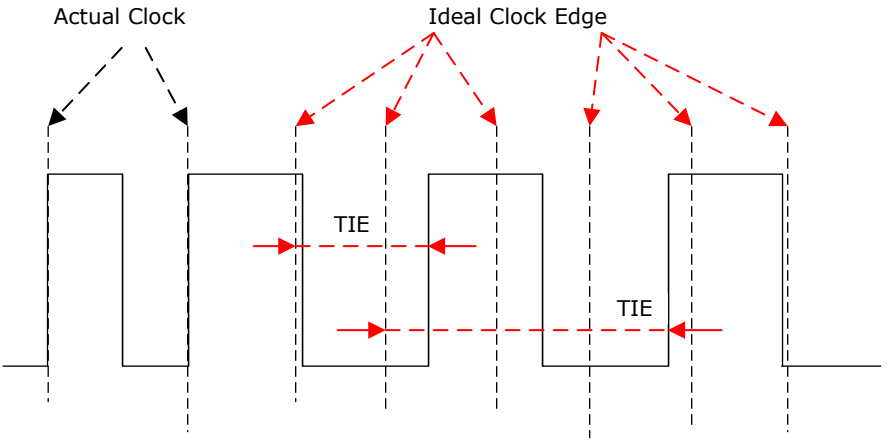


Figure 4-3. Deterministic Jitter - Multiple Peaks



The preceding figures are representative of how deterministic jitter may appear in an application. Instead of a single Gaussian peak, there appear to be several combined. This shows the deterministic behavior of something in the application repeatedly affecting the cycle time. In Figure 4-2, the peak of the curve is shifted to the right side of the histogram, meaning that the mean clock period is larger than the expected clock period. This is an example of long-term jitter, where the period of a clock cycle errors incrementally in one direction over time. In these cases, the TIE can be much larger because the period of the clock cycle is consistently larger than desired. Figure 4-4 shows how deterministic jitter can affect TIE in the long term. Notice that the TIE increases because two consecutive clock pulses are larger than expected.

Figure 4-4. Long Term Jitter



5. Measurement Techniques

Measurement techniques and conditions can have a large impact on results. The equipment used needs to have the required capabilities for the signal speeds and edge rates. Care should be taken with probe ground lead length to minimize inaccurate amplitudes. Some equipment may have built-in clock or jitter analysis tools or software. It may be necessary to perform several tests to identify and quantify system jitter. Measurements can be taken along the circuit path to help identify the sources. Some typical test points are:

- Clock source, if external to the microcontroller
- Microcontroller internal clock source brought out to a pin
- PLL output brought out to a pin
- PWM output pins
- Gate driver outputs

Given that the microcontroller and other circuitry can add to system noise, it can be helpful to isolate noise sources and compare before and after measurements. For example, halting communication lines or other PWM signals can help evaluate their impact on the signal of interest.

5.1 Measurement Steps

Perform the following steps to measure jitter:

1. Set up the Device Under Test (DUT), including power and configuration parameters.
2. Connect the oscilloscope probe to the pin designated as the reference output. It is important not to use the oscilloscope in averaging mode because it can mask the results.
3. Configure oscilloscope as needed for the type of jitter to be measured. Recording adjacent periods allows the cycle-to-cycle jitter to be calculated, as well as the period jitter and TIE.
4. Repeat step 3 for a minimum of 10,000 counts. A large sample size will help guarantee any long-term jitter is identified.
5. Create a histogram from the test results and calculate the mean and standard deviation. The histogram will show the number of occurrences of each measured period.
6. Using this data, determine if any deterministic jitter can be identified and if the level of jitter is acceptable for the application.
7. If deterministic jitter is seen, such as multiple peaks displayed on the histogram, consider using a spectrum analyzer to identify which frequency is causing the additional peaks.
8. Follow the instructions below to mitigate the deterministic jitter. If a spectrum analyzer was used and any additional frequency ranges beyond the desired timing signal were identified, focus on the steps in the following section that would best address the issue.

6. Mitigating Jitter

Knowing now that different components and types of jitter can become troublesome problems in a system, it's time to consider ways to reduce deterministic jitter in a system. The random component cannot be removed. In the previous sections, the causes of jitter were discussed. Some methods used to reduce jitter are:

- Source input clock
- PLL filtering
- Power supply noise reduction
- PCB and system design

6.1 Clock Source Jitter

While an external oscillator is not necessary, it has been found that using a high-speed MEMS or a crystal oscillator circuit as the reference clock can reduce device jitter compared to an internal source. Using a higher frequency input clock allows better clocking accuracy when used with a dsPIC® PLL. This is due to the VCO frequency correction happening every few clock cycles, meaning an external clock frequency higher than the default 8MHz FRC speed will allow the VCO correction to occur more often. It is important to test the system in order to understand if additional clock accuracy is necessary for the application, or if the level of jitter in the system is acceptable.

6.2 Phase-Locked Loop (PLL)

It is common for a PLL to be used in PWM applications to achieve the required speeds and resolutions. A PLL can either improve or degrade jitter performance depending on its design and filtering. The PLL's performance can be dependent on its operational parameters, including feedback multipliers and VCO frequency. Configuring the PLL's pre or post dividers to optimize the VCO frequency can help reduce deterministic jitter.

6.3 Power supply

Capturing the histogram of jitter with multiple power supply sources can help identify sources of jitter in the system. For example, consider the difference between the results when using an on-board power supply and a laboratory power supply. If, after testing, the results show that the jitter is much higher when using the designed power circuit on the PCB, then some circuit adjustments will need to be made.

6.4 Circuit Adjustments

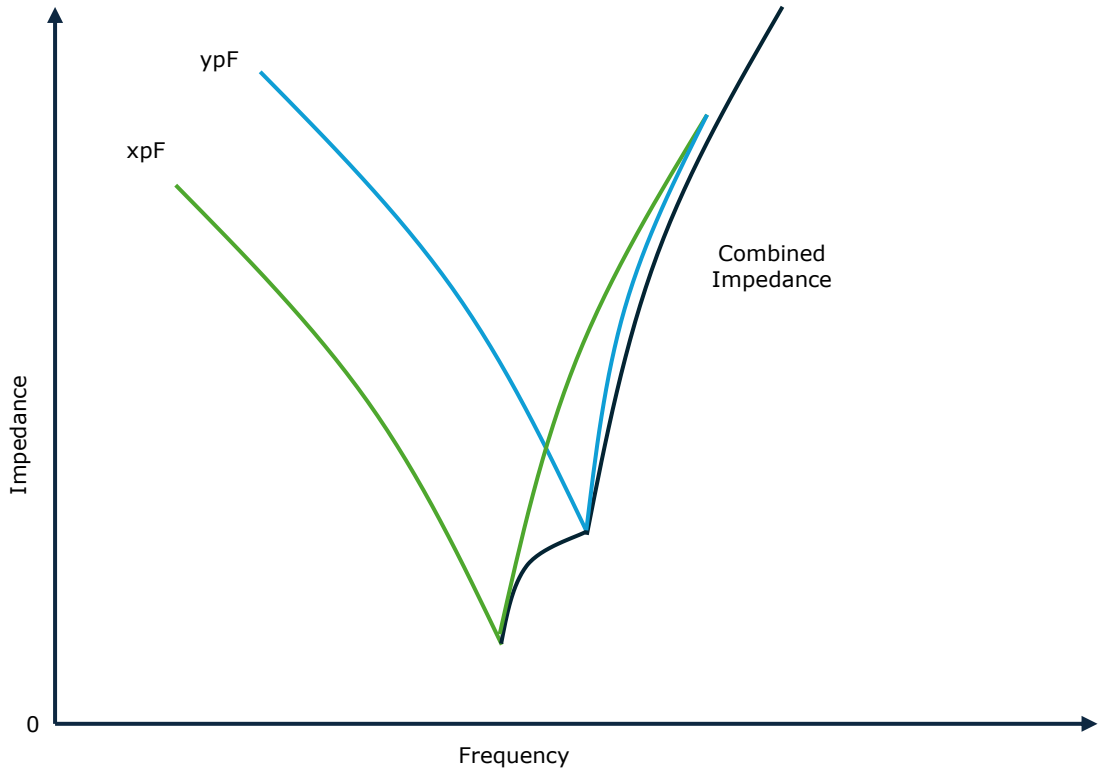
Starting with the design of the power traces, consider their lengths. The longer a trace takes to reach its destination, the more susceptible to noise it becomes. The quickest way to resolve this would be to use a PCB with a minimum of 4 layers, with an independent layer for both VDD and VSS. In this layout, the ground path is no longer than the length of the net plus the via to ground. To limit the length of traces even further, ensure components are placed close together. This is especially important with the bypass capacitors next to the DUT. Additionally, when choosing pins from a device, two high-speed signals should not be used from adjacent pins unless they are meant to be used together. For example, using I2C right next to PWM will introduce noise across the two pins, increasing jitter and reducing the resolution of the PWM. This noise will increase if the I2C and PWM are using different clock sources, as asynchronous signals will inject more errors than synchronous ones.

6.5 Components

Component selection can have an impact on system noise. Bypass capacitor values should be chosen with the lowest impedance at the desired operating frequency. This means understanding which capacitors will act as the best filter on the power supply when the pins are toggling. Adding these additional bypass capacitors may go beyond the recommended capacitor layout on a device's

data sheet and would also take up space on a PCB, so make sure to consult a capacitor's data sheet and select only the necessary values based on the application frequency. The following figure shows a common graph found in these data sheets.

Figure 6-1. Impedance and Frequency



6.6 Other Factors

Other variables, such as temperature, may affect jitter results in a device. The following example uses the dsPIC33AK128MC106 GP DIM on a Curiosity Platform Board with a PWM output at 100kHz. As seen in the two diagrams, an increase in temperature can greatly affect the output of a PWM. As temperature increases, so too does the deterministic jitter, which causes more infrequent cycle times.

Figure 6-2. PWM 100kHz Room Temperature

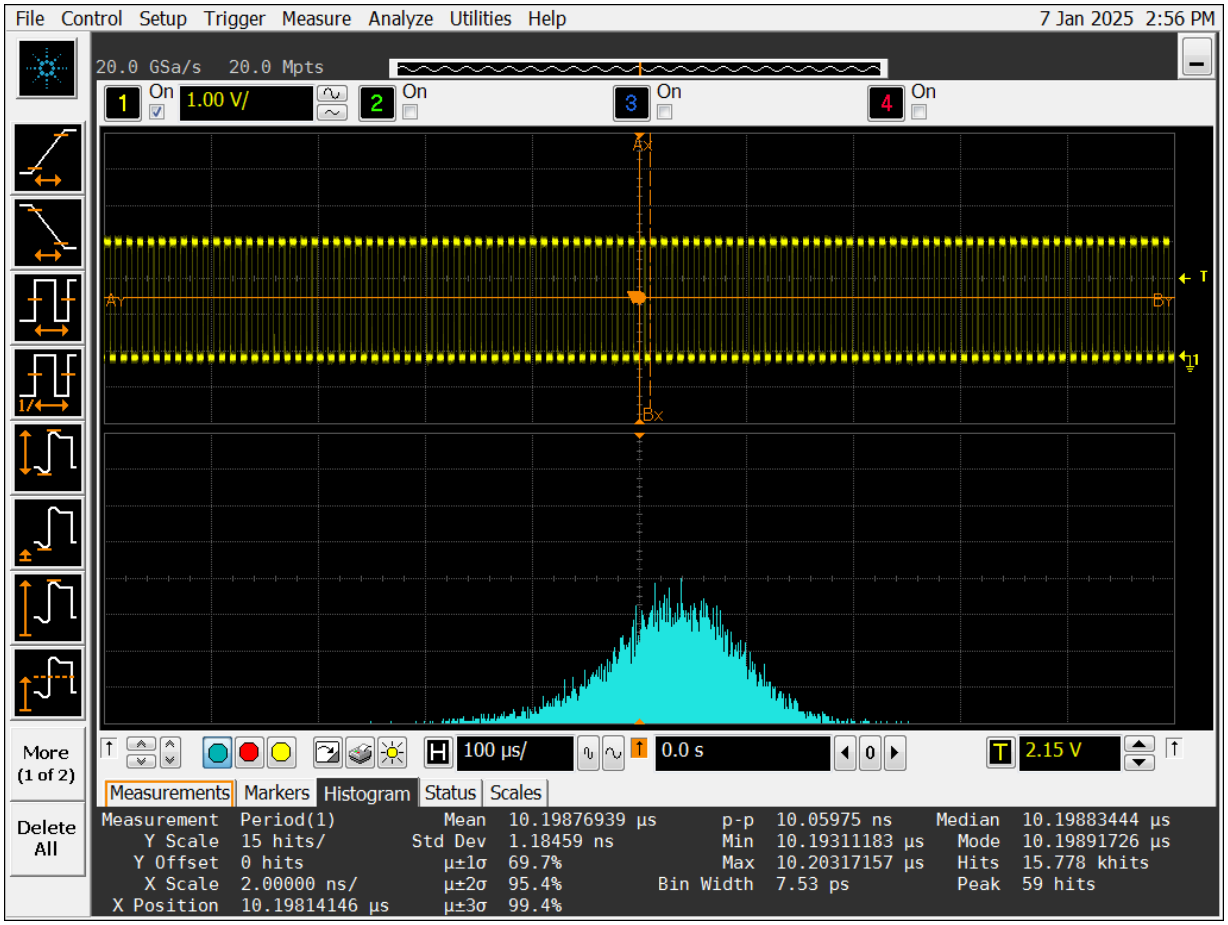
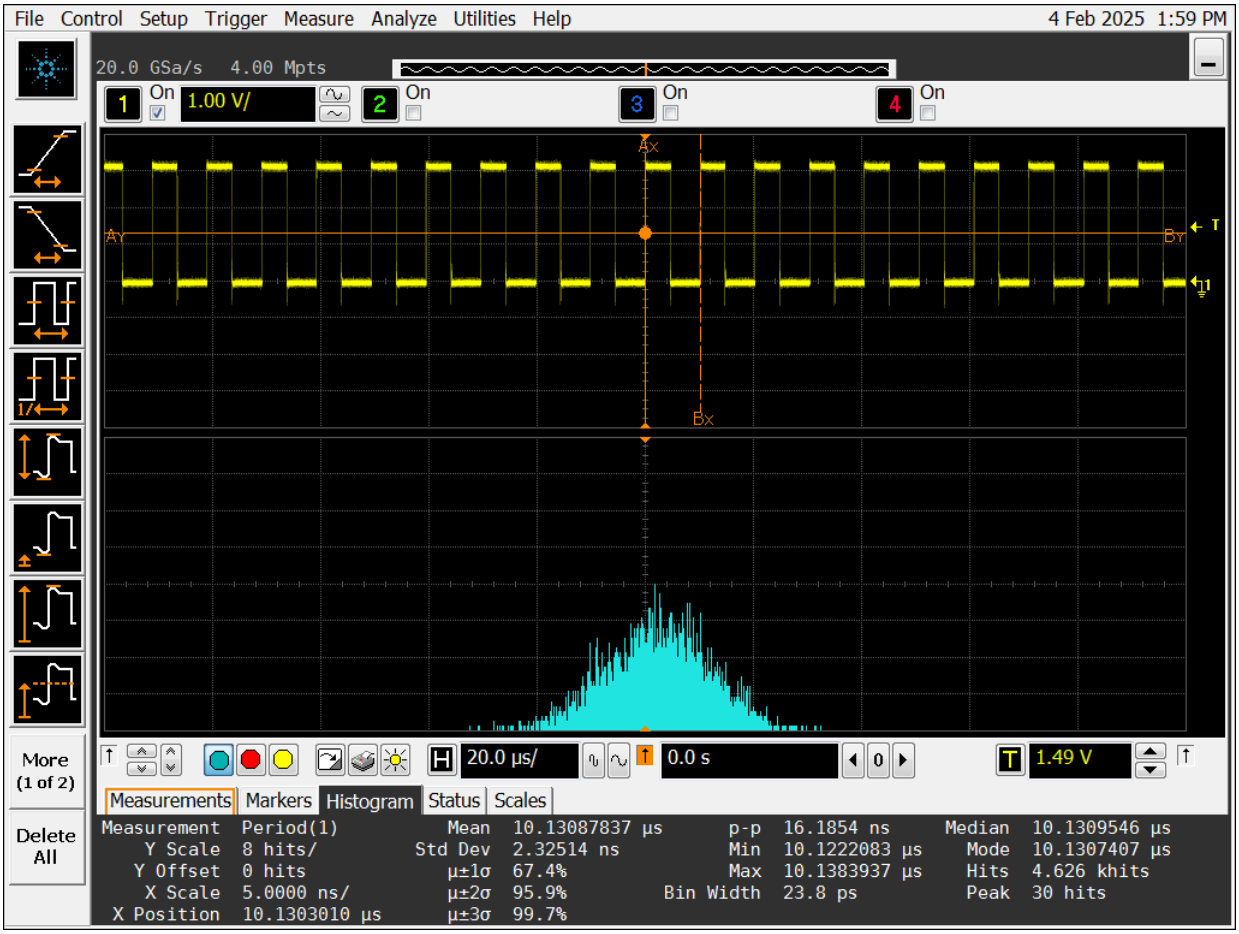
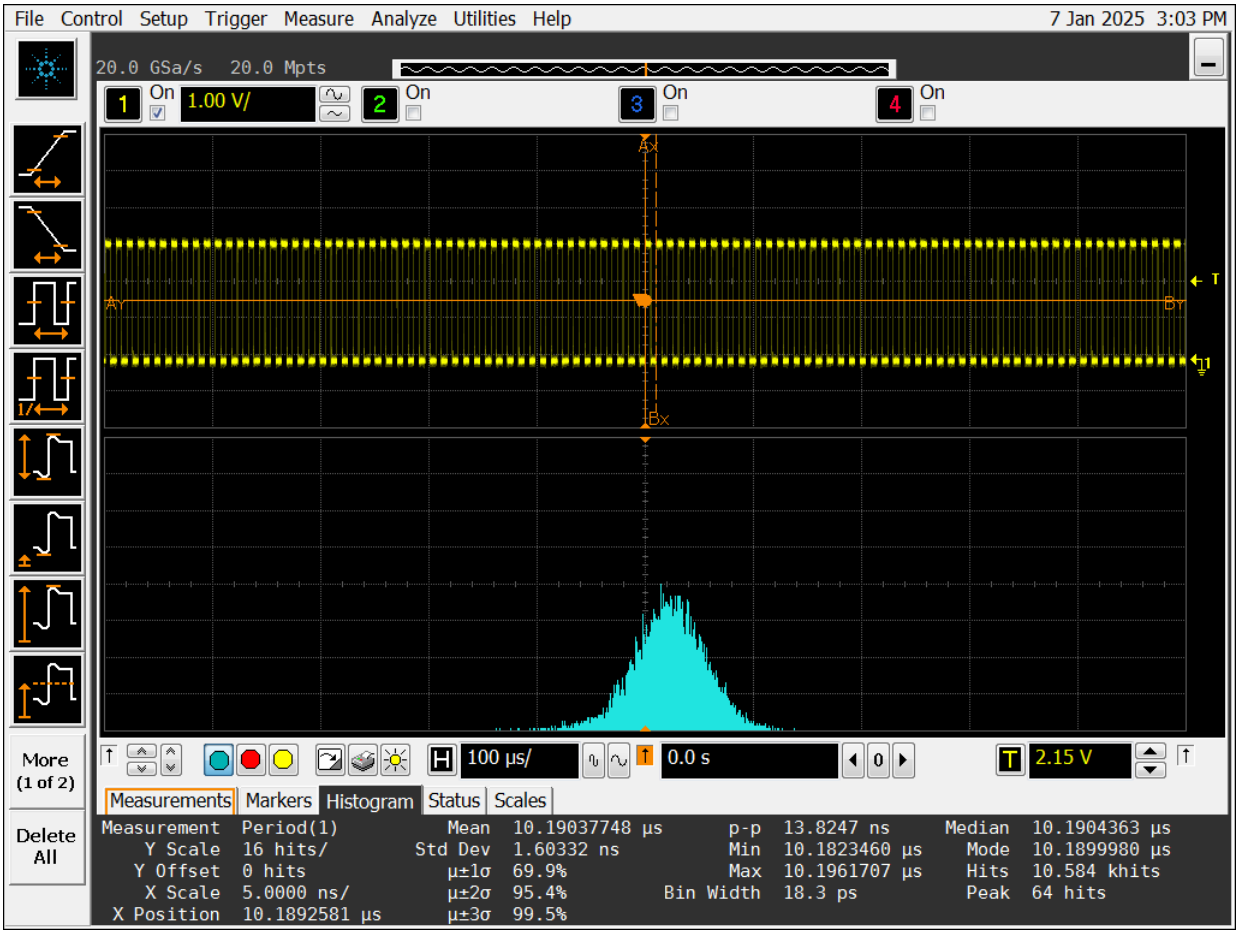


Figure 6-3. PWM 100kHz High Temperature



A decrease from room temperature can also affect the PWM output.

Figure 6-4. PWM 100kHz Low Temperature



Beyond temperature, jitter can also vary from part to part and can change over time. This can be due to the manufacturing process, component degradation and power supply stability. The most effective way to ensure longevity in devices and prevent jitter from increasing over time is to maintain a controlled environment during device operation. A power supply with less noise will inject less stress on the system over time, and protecting the device from high temperatures will help slow the aging process.

7. Example Clocking Configuration for Best Performance

The input clock source and PLL configuration can have a significant impact on PWM output jitter. This section provides an example of how to maximize performance in a dsPIC PWM application. The equations used throughout this example are based on the following equation:

Equation 7-1.

$$F_{PLLI}\ x\ \left(\frac{PLLFBDIV}{PLLPRE\ x\ POSTDIV1\ x\ POSTDIV2}\right) = F_{PLLo}$$

A PLL circuit utilizing a smaller feedback divide value (PLLFBDIV) can minimize jitter. A smaller feedback divider value results in a shorter period of the filter loop and a faster response to correct for deviations in the input signal. In this example, the minimum VCO frequency (as specified in the electrical characteristics of the device data sheet) is 400HMz. Using the PLL to get an output clock frequency of 400MHz can be achieved in a variety of ways. If an 8MHz FRC is used, the PLL can be set up using the following equation:

Equation 7-2.

$$8\ x\ (\frac{100}{1\ x\ 2\ x\ 1}) = 400$$

With an 8MHz FRC, the feedback divider is 100, meaning there is a gain multiplier of 100.

Further jitter reduction can be achieved with a source clock of high accuracy and higher frequency. If the PLL reference clock frequency is increased from 8MHz to a 25MHz external clock, either a MEMS or a crystal oscillator, the equation yields a smaller feedback divider value of 32.

Equation 7-3.

$$25\ x\ (\frac{32}{1\ x\ 2\ x\ 1}) = 400$$

By increasing the frequency of the reference clock and adjusting the PLL feedback divider accordingly, PLL performance is optimized. This drastically decreases the potential jitter on a given signal. In this example, two methods of mitigating jitter are included, but additional methods presented earlier should be considered to reduce deterministic jitter and achieve the best reference clock for a PWM.

To test jitter on a dsPIC33A part, use the dsPIC33AK128MC106 GP DIM and the Curiosity Platform Development Board. The following example code will use mikroBUS header A's pin 16 to ouput a PWM signal. Each example uses a different clock source against which jitter can be measured. The first example uses an 8MHz internal FRC oscillator from the dsPIC, and the PWM is clocked from that to output ~100kHz. The second example uses that same FRC in addition to a PLL that sets the output clock signal to 200MHz. The PWM signal dividers are adjusted accordingly to still output 100kHz. In both cases, an oscilloscope can be used to measure the jitter on pin 16 of mikroBUS interface A.

8. Device-Specific Examples

An example of the test setup and results is provided for each of the following device families: dsPIC33E, dsPIC33C and dsPIC33A. Each test provides both FRC and FRC+PLL as clock source options, so FRC results can be used as a point of comparison for any other results. Jitter can be tested on both the clock REFO peripheral and a PWM output. These setups use development boards that are available for customer purchase so that they can be replicated. Additionally, the duty cycle and period can be easily modified for more specific tests such as duty cycle-duty cycle and TIE.

8.1 dsPIC33E

The dsPIC33E test uses a version of the Digital Power Development Board with a corresponding dsPIC33EP128GS806 DP PIM. PWM output 1 is configured in code, and the module uses either FRC or FRC+PLL as its clock source, depending on what is defined at the start of the code. In both cases, the PWM outputs 100 kHz for jitter measurement and comparison of the two results.

```
// <editor-fold defaultstate="collapsed" desc="Config Bits">
// FICD
#pragma config ICS = PGD1               // ICD Communication Channel Select bits (Communicate
on PGEC1 and PGED1)
//#pragma config JTAGEN = OFF           // JTAG Enable bit (JTAG is disabled)

// FPOR
#pragma config ALTI2C1 = OFF            // Alternate I2C1 pins (I2C1 mapped to SDA1/SCL1 pins)
//#pragma config ALTI2C2 = OFF          // Alternate I2C2 pins (I2C2 mapped to SDA2/SCL2 pins)
#pragma config WDTWIN = WIN25           // Watchdog Window Select bits (WDT Window is 25% of
WDT period)

// FWDT
#pragma config WDTPOST = PS32768        // Watchdog Timer Postscaler bits (1:32,768)
#pragma config WDTPRE = PR128           // Watchdog Timer Prescaler bit (1:128)
#pragma config PLLKEN = ON              // PLL Lock Enable bit (Clock switch to PLL source
will wait until the PLL lock signal is valid.)
#pragma config WINDIS = OFF             // Watchdog Timer Window Enable bit (Watchdog Timer
in Non-Window mode)
#pragma config WDTEN = OFF            // Watchdog Timer Enable bit (Watchdog timer always
enabled)

// FOSC
#pragma config POSCMD = NONE            // Primary Oscillator Mode Select bits (Primary
Oscillator disabled)
#pragma config OSCIOFNC = ON            // OSC2 Pin Function bit (OSC2 is clock output)
#pragma config IOL1WAY = ON             // Peripheral pin select configuration (Allow only
one reconfiguration)
#pragma config FCKSM = CSECMD           // Clock Switching Mode bits (Clock switching is
enabled,Fail-safe Clock Monitor is disabled)

// FOSCSEL
#pragma config FNOSC = FRC              // Oscillator Source Selection (Internal Fast RC
(FRC))
#pragma config PWMLOCK = OFF            // PWM Lock Enable bit (Certain PWM registers may
only be written after key sequence)
#pragma config IESO = ON                // Two-speed Oscillator Start-up Enable bit (Start up
device with FRC, then switch to user-selected oscillator source)

// FGS
//#pragma config GWRP = OFF             // General Segment Write-Protect bit (General Segment
may be written)
//#pragma config GCP = OFF              // General Segment Code-Protect bit (General Segment
Code protect is Disabled)
// </editor-fold>

#include<xc.h>

#define use_PLL
//#define use_FRC

int main(void) {

    _TRISA4 = 0;
    _TRISC13 = 0;
#ifdef use_FRC
```

```
    PTPER = 4000000 / 6840;                // Period setting for use with FRC
#endif

#ifdef use_PLL
    //PTPER = FCY/(PWM frequency * prescale)
    PTPER = 40000000 / 68400;                // Period setting for use with FRC + PLL
    CLKDIVbits.PLLPRE = 0;
    PLLFBD = 40;
    CLKDIVbits.PLLPOST = 0b01;
    CLKDIVbits.PLLPRE = 0;
    // initiate clock switch to FRC with PLL (NOSC=1)
    __builtin_write_OSCCONH(0x01);      // NOSC=1 -> set new OSC
    __builtin_write_OSCCONL(OSCCONL | 0x01); // OSWEN=1 -> request clock switch
    while(OSCCONbits.OSWEN != 0);       // wait for clock switch
    while(OSCCONbits.LOCK != 1);        // wait for PLL lock
#endif
    //REFO setup
    _RP52R = 0b110001;                  // REFO PPS
    _RODIV = 1;                         //REFO Divider
    _REFOMD = 0;                        //Reference module is enabled
    _ROON = 1;                          //Reference out is on
    _ROSEL = 0;                         //Reference out is system clock

    PTCONbits.PTEN = 0;                 //PWM Module is disabled during setup
    PWMCON1bits.ITB = 0;                //PTPER register provides time base ofr PWM1
    PTCONbits.PTSIDL = 0;               // PWM time base operates in a Free Running mode
    PTCON2bits.PCLKDIV = 0;             // PWM time base input clock period is TCY (1:1
prescale)

    // Configure PWM1
    IOCON1bits.PENH = 1;                //PWMx module controls the PWMxH pin
    IOCON1bits.PENL = 0;                //GPIO module controls the PWMxL pin
    IOCON1bits.PMOD = 3;                // PWM1 in Independent mode
    FCLCON1bits.FLTMOD = 0b11;          //Fault mode disabled

    PDC1 = PTPER / 2; // 50% of PTPER
    PTCONbits.PTEN = 1;                 //Enable PWM after all other settings are configured

    while(1){
        Nop();
    }
    return 0;
}
```

8.1.1 dsPIC33E Results

Below are the results from the jitter testing on the dsPIC33EP128GS806 DP PIM.

Figure 8-1. dsPIC33EP128GS806 PWM Jitter Results with FRC

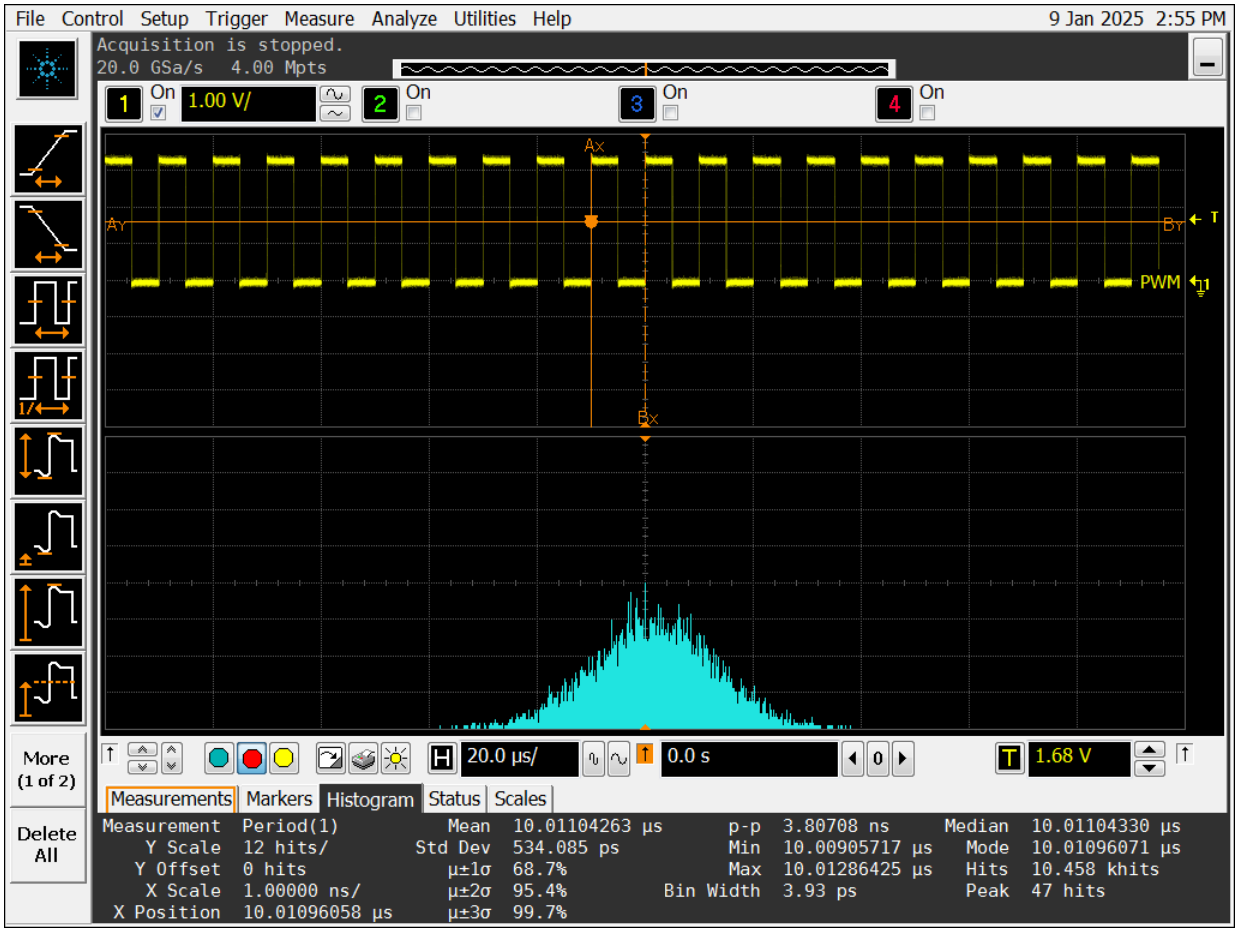
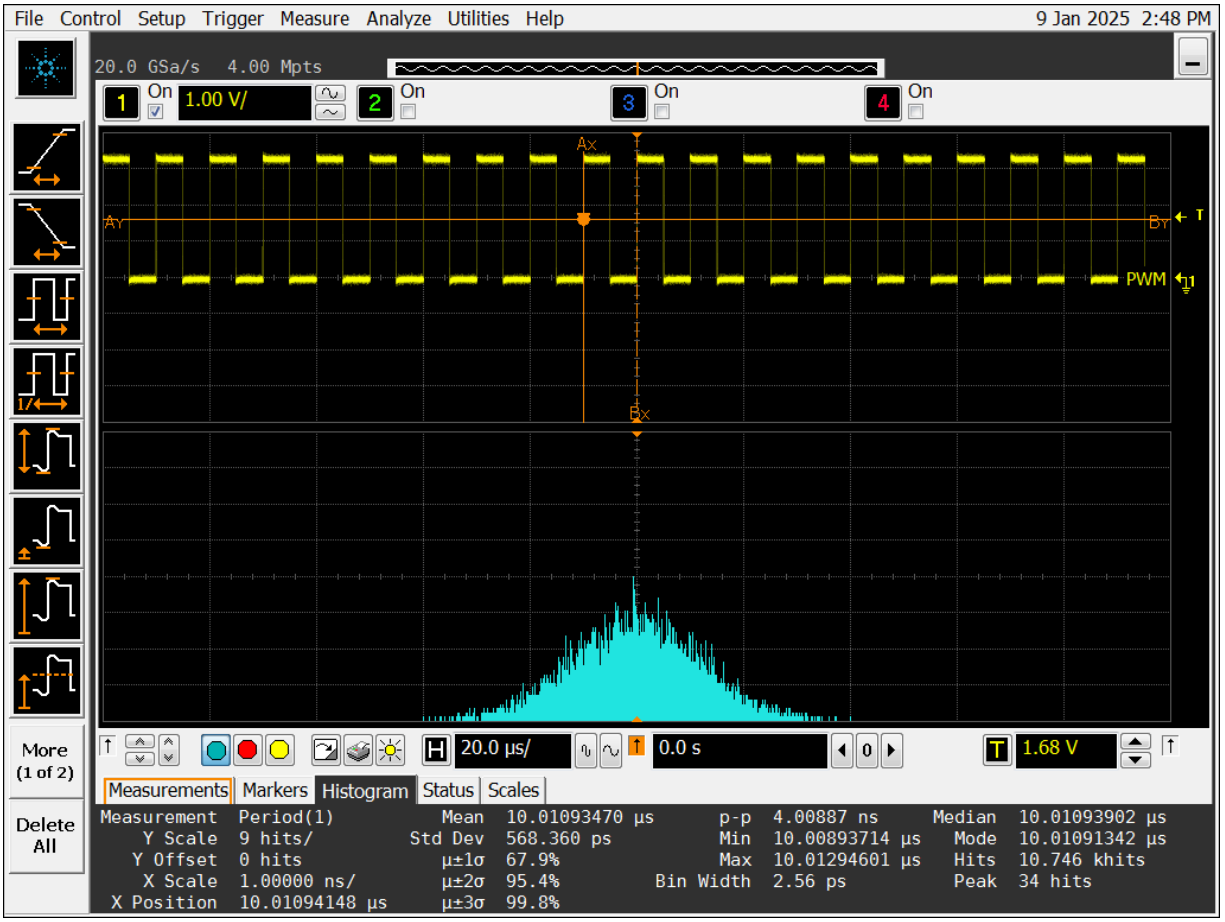


Figure 8-2. dsPIC33EP128GS806 PWM Jitter Results with FRC+PLL



8.2 dsPIC33C

The dsPIC33C test uses the dsPIC33C Touch CAN LIN Curiosity Development Board with an on-board dsPIC33CK1024MP710. The provided example code allows the testing of both a PWM and the clock reference output (REFO) peripheral. Both outputs can be tested using either FRC or PLL, depending on which is defined at the start of the code.

```
// <editor-fold defaultstate="collapsed" desc="Config Bits">
// FOSCSE
#pragma config FNOSC = FRCDIVN          // Oscillator Source Selection (Internal Fast RC
(FRC) Oscillator with postscaler)
#pragma config IESO = ON                // Two-speed Oscillator Start-up Enable bit (Start up
device with FRC, then switch to user-selected oscillator source)

// FOSC
#pragma config POSCMD = NONE            // Primary Oscillator Mode Select bits (Primary
Oscillator disabled)
#pragma config OSCIOFNC = OFF           // OSC2 Pin Function bit (OSC2 is clock output)
#pragma config FCKSM = CSECMD           // Clock Switching Mode bits (Clock switching is
enabled,Fail-safe Clock Monitor is disabled)
#pragma config PLLKEN = ON              // PLL Lock Status Control (PLL lock signal will be
used to disable PLL clock output if lock is lost)
#pragma config XTCFG = G3               // XT Config (24-32 MHz crystals)
#pragma config XTBST = ENABLE           // XT Boost (Boost the kick-start)

// FICD
#pragma config ICS = PGD1               // ICD Communication Channel Select bits (Communicate
on PGC1 and PGD1)
#pragma config JTAGEN = OFF             // JTAG Enable bit (JTAG is disabled)
#pragma config NOBTSWP = DISABLED       // BOOTSWP instruction disable bit (BOOTSWP
instruction is disabled)
// </editor-fold>
```

```
#include "xc.h"
#include <libpic30.h>
//#define use_PLL
#define use_FRC
int main(void) {
    _RP52R = 14;                              // pin 80 on the device. RC4

#ifdef use_PLL
    CLKDIVbits.PLLPRE   = 1;                  // N1=1
    PLLFBDbits.PLLFBDIV = 125;                // M=125 PLLFBD
    PLLDIVbits.POST1DIV = 5;                  // N2=5 PLLDIV
    PLLDIVbits.POST2DIV = 2;                  // N3=1

    __builtin_write_OSCCONH(0x01);            // NOSC=1 -> set new OSC
    __builtin_write_OSCCONL(OSCCONL | 0x01);// OSWEN=1 -> request clock switch
    while(OSCCONbits.OSWEN != 0);             // wait for clock switch
    while(OSCCONbits.LOCK != 1);              // wait for PLL lock
    PG7PER = 0x200;                            //Set period with FRC+PLL
    PG7DC = 0x100;                             //Set duty cycle with FRC+PLL
#endif

#ifdef use_FRC
    PG7PER = 0x50;                            //Set period with FRC clock source
    PG7DC = 0x25;                             //Set duty cycle with FRC clock source
#endif

    _ANSELE0 = 0;                             //Digital function on E0
    _ROOUT = 1;                               //Reference Clock Output Enable
    _ROSEL = 0;                               //Reference Source is System Clock
    _RODIV = 0;                               //No division of base clock value
    _ROEN = 1;                                //Enable Reference Clock
    _TRISD1 = 0;                              //D1 set digital output

    PCLKCONbits.MCLKSEL = 0b00;               //0 is FOSC  and 2 is PLL divider output
    PG7CONHbits.TRGMOD = 1;                   //re-triggerable mode
    PG7CONLbits.CLKSEL = 0b01;                //uses clock set by MCLKSEL

    PG7IOCONHbits.PMOD = 1;                   //PWM operates in independent mode
    PG7IOCONHbits.PENH = 1;                   //PWM controls the output of the pin
    PG7CONLbits.ON = 1;                       //Enable PWM 7

    while(1){
        Nop();
    }
    return 0;
}
```

8.2.1 dsPIC33C Results

Below is a comparison of the histogram for the PWM output being clocked by an 8MHz FRC vs FRC+PLL on a dsPIC33C Touch CAN LIN Curiosity Board using the provided test code.

Figure 8-3. dsPIC33CK1024MP710 PWM Jitter Results with FRC

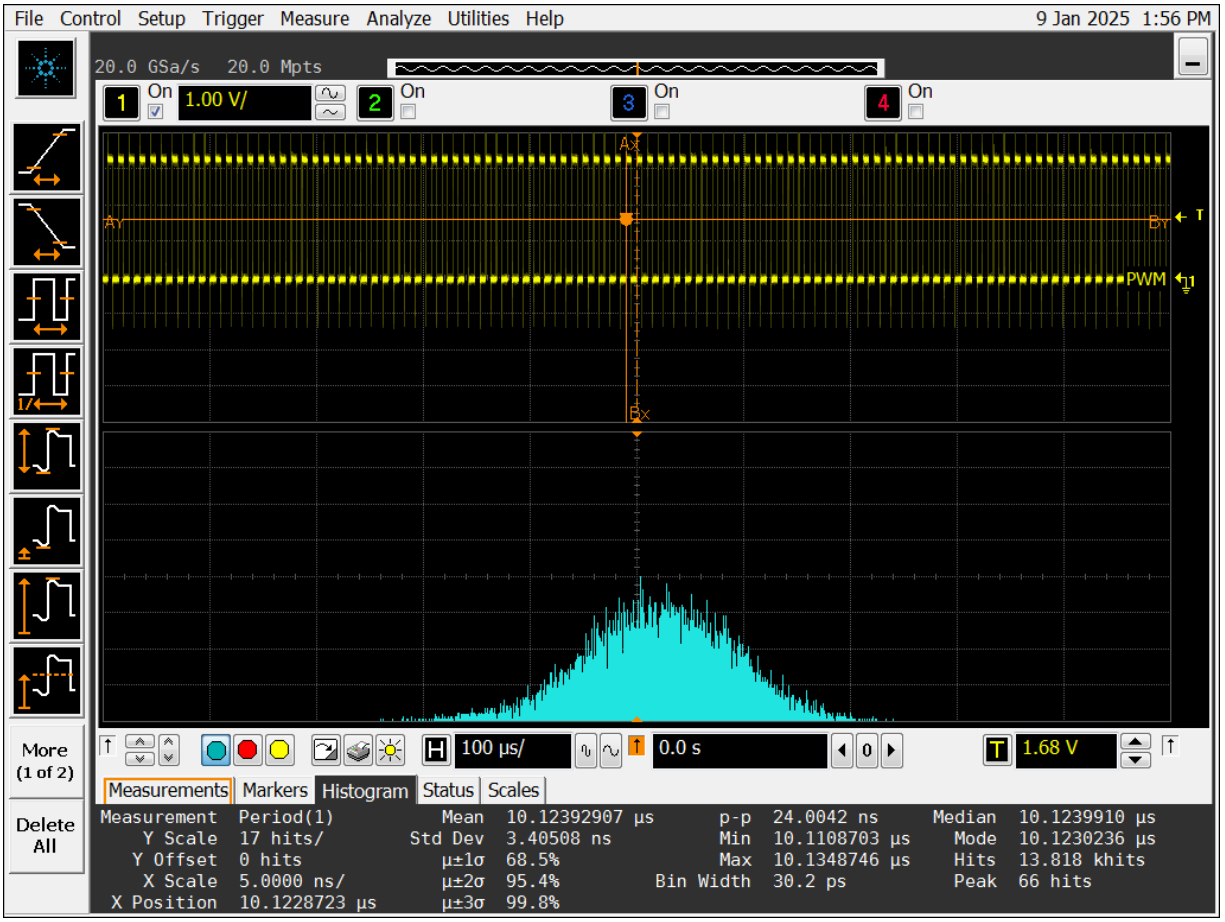
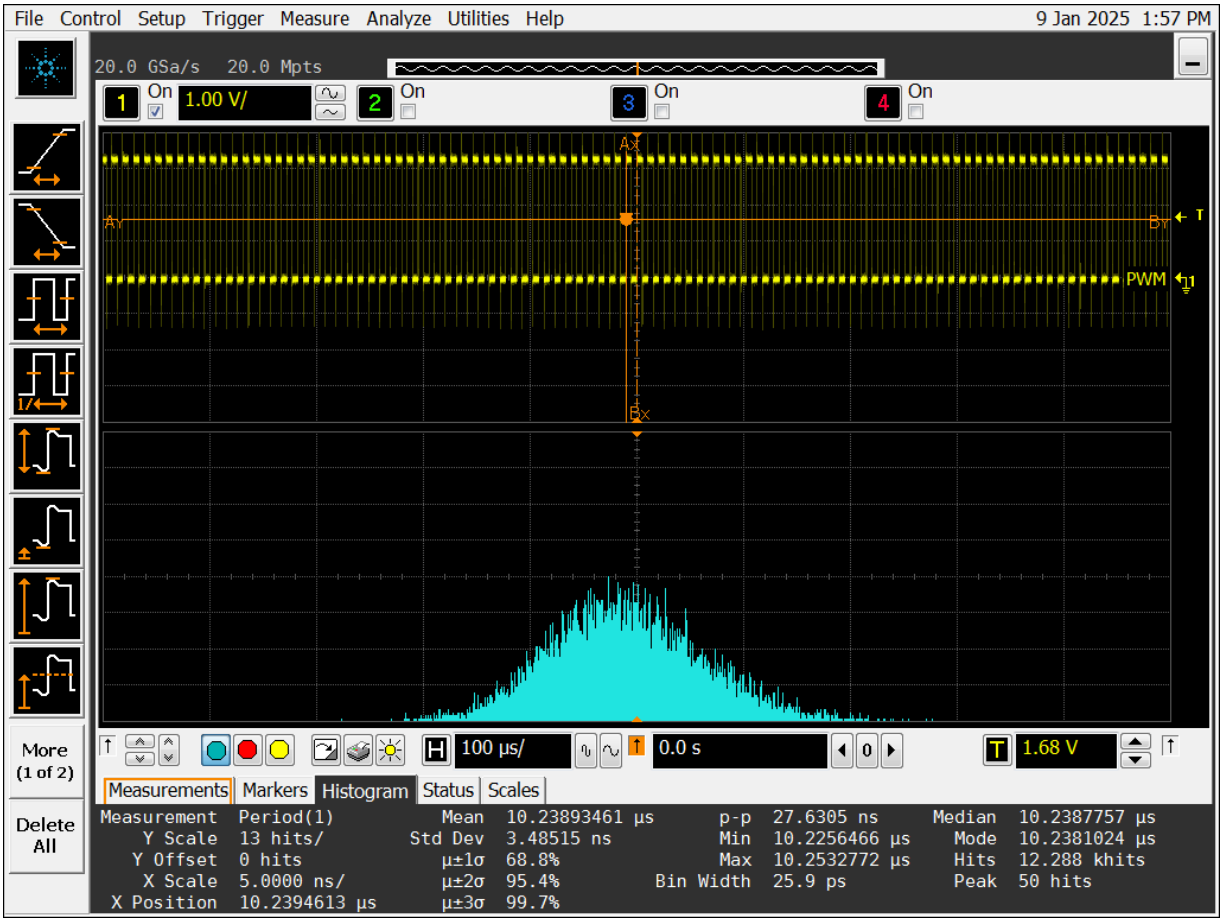


Figure 8-4. dsPIC33CK1024MP710 PWM Jitter Results with FRC+PLL



8.3 dsPIC33A

To test jitter on a dsPIC33A part, use the dsPIC33AK128MC106 GP DIM and the Curiosity Platform Development Board.

The example code provides several clock setups to test jitter. The examples include the PWM being clocked by FRC, FRC+PLL, 8 MHz MEMS and MEMS+PLL. The PWM signal dividers are adjusted accordingly to output 100 kHz in all cases, and an oscilloscope can be used to measure the output.

```
#include "xc.h"
#include <stdio.h>
#define use_FRC
//#define use_PLL
//#define use_MEMS
//#define use_MEMS_PLL


void pwm_Init();
void clock_PWM_at_FRC();
void clock_PWM_at_MEMS();
void clock_PWM_at_400MHz_from_PLL2_Fout();
void clock_PWM_at_400MHz_EC_PLL2_Fout();

void clock_PWM_at_400MHz_from_PLL2_Fout() {
    PLL2CONbits.ON = 1; //Enable PLL generator 2, if not already enabled
    //Select FRC as PLL2's clock source
    PLL2CONbits.NOSC = 1;
    //Request PLL2 clock switch
    PLL2CONbits.OSWEN = 1;
    //Wait for PLL2 clock switch to complete
```

```
    while(PLL2CONbits.OSWEN);

    //Set up PLL2 dividers to output 200MHz
    PLL2DIVbits.PLLPRE = 1; //Reference input will be 8MHz, no division
    PLL2DIVbits.PLLFBDIV = 200; //Fvco = 8MHz * 200 = 1600MHz
    PLL2DIVbits.POSTDIV1 = 4; //Divide Fcvo by 4
    PLL2DIVbits.POSTDIV2 = 1; //Fpllo = Fvco / 4 / 1 = 400 MHz

    //The PLLSWEN bit controls changes to the PLL feedback divider.
    //Request PLL2 feedback divider switch
    PLL2CONbits.PLLSWEN = 1;
    //Wait for PLL2 feedback divider switch to complete
    while(PLL2CONbits.PLLSWEN);

    //The FOUTSWEN bit controls changes to the PLL output dividers.
    //Request PLL2 output divider switch
    PLL2CONbits.FOUTSWEN = 1;
    //Wait for PLL2 output divider switch to complete
    while(PLL2CONbits.FOUTSWEN);

    //Enable CLKGEN5
    CLK5CONbits.ON = 1;
    //Reset CLKGEN5 fractional divider for 1:1 ratio
    CLK5DIVbits.INTDIV = 0;
    CLK5DIVbits.FRACDIV = 0;
    //Request CLKGEN5 fractional divider switch
    CLK5CONbits.DIVSWEN = 1;
    //Wait for CLKGEN5 fractional divider switch to complete
    while(CLK5CONbits.DIVSWEN);

    //Set PLL2 Fout as new CLKGEN5 clock source
    CLK5CONbits.NOSC = 6;
    //Request CLKGEN5 clock switch
    CLK5CONbits.OSWEN = 1;
    //Wait for CLKGEN5 clock switch to complete
    while (CLK5CONbits.OSWEN);

    //Select CLKGEN5 as PWM master clock source
    PCLKCONbits.MCLKSEL = 1;
}

void clock_PWM_at_400MHz_EC_PLL2_Fout() {
    _POSCMD = 0b00;
    PLL2CONbits.ON = 1; //Enable PLL generator 2, if not already enabled
    //Select FRC as PLL2's clock source
    PLL2CONbits.NOSC = 3;
    //Request PLL2 clock switch
    PLL2CONbits.OSWEN = 1;
    //Wait for PLL2 clock switch to complete
    while(PLL2CONbits.OSWEN);

    //Set up PLL2 dividers to output 200MHz
    PLL2DIVbits.PLLPRE = 1; //Reference input will be 8MHz, no division
    PLL2DIVbits.PLLFBDIV = 200; //Fvco = 8MHz * 200 = 1600MHz
    PLL2DIVbits.POSTDIV1 = 4; //Divide Fcvo by 4
    PLL2DIVbits.POSTDIV2 = 1; //Fpllo = Fvco / 4 / 1 = 400 MHz

    //The PLLSWEN bit controls changes to the PLL feedback divider.
    //Request PLL2 feedback divider switch
    PLL2CONbits.PLLSWEN = 1;
    //Wait for PLL2 feedback divider switch to complete
    while(PLL2CONbits.PLLSWEN);

    //The FOUTSWEN bit controls changes to the PLL output dividers.
    //Request PLL2 output divider switch
    PLL2CONbits.FOUTSWEN = 1;
    //Wait for PLL2 output divider switch to complete
    while(PLL2CONbits.FOUTSWEN);

    //Enable CLKGEN5
    CLK5CONbits.ON = 1;
    //Reset CLKGEN5 fractional divider for 1:1 ratio
    CLK5DIVbits.INTDIV = 0;
    CLK5DIVbits.FRACDIV = 0;
    //Request CLKGEN5 fractional divider switch
    CLK5CONbits.DIVSWEN = 1;
    //Wait for CLKGEN5 fractional divider switch to complete
    while(CLK5CONbits.DIVSWEN);
```

```
    //Set PLL2 Fout as new CLKGEN5 clock source
    CLK5CONbits.NOSC = 6;
    //Request CLKGEN5 clock switch
    CLK5CONbits.OSWEN = 1;
    //Wait for CLKGEN5 clock switch to complete
    while (CLK5CONbits.OSWEN);

    //Select CLKGEN5 as PWM master clock source
    PCLKCONbits.MCLKSEL = 1;
}

void clock_PWM_at_FRC(){
    //Enable CLKGEN5
    CLK5CONbits.ON = 1;
    //Reset CLKGEN5 fractional divider for 1:1 ratio
    CLK5DIVbits.INTDIV = 0;
    CLK5DIVbits.FRACDIV = 0;
    //Request CLKGEN5 fractional divider switch
    CLK5CONbits.DIVSWEN = 1;
    //Wait for CLKGEN5 fractional divider switch to complete
    while(CLK5CONbits.DIVSWEN);
    //Set FRC as PWM clock source
    CLK5CONbits.NOSC = 1;
    //Request CLKGEN5 clock switch
    CLK5CONbits.OSWEN = 1;
    //Wait for CLKGEN5 clock switch to complete
    while (CLK5CONbits.OSWEN);
    //Select CLKGEN5 as PWM master clock source
    PCLKCONbits.MCLKSEL = 1;
}

void clock_PWM_at_MEMS(){
    _POSCMD = 0b00;
        //Enable CLKGEN5
    CLK5CONbits.ON = 1;
    //Reset CLKGEN5 fractional divider for 1:1 ratio
    CLK5DIVbits.INTDIV = 0;
    CLK5DIVbits.FRACDIV = 0;
    //Request CLKGEN5 fractional divider switch
    CLK5CONbits.DIVSWEN = 1;
    //Wait for CLKGEN5 fractional divider switch to complete
    while(CLK5CONbits.DIVSWEN);
    //Set FRC as PWM clock source
    CLK5CONbits.NOSC = 3;
    //Request CLKGEN5 clock switch
    CLK5CONbits.OSWEN = 1;
    //Wait for CLKGEN5 clock switch to complete
    while (CLK5CONbits.OSWEN);
    //Select CLKGEN5 as PWM master clock source
    PCLKCONbits.MCLKSEL = 1;
}

void pwm_Init(){
    //PG1PER = 0x10000;//100kHz with PLL
    //PG1DC = 0x8000;//100kHz with PLL


    PG1CONbits.UPDMOD = 0b000;  //PWM buffer update mode is at start of next PWM cycle if
UPDREQ = 1
    PG1CONbits.TRGMOD = 0b01;  //PWM generator 1 operates in single trigger mode
    PG1CONbits.SOCS = 0b0000;    //Start of cycle is local EOC

    PG1CONbits.ON = 0;          //PWM Generator 1 is disabled (do not start yet)
    PG1CONbits.TRGCNT = 1;      //PWM Generator 1 produces 1 PWM cycle when triggered
    PG1CONbits.CLKSEL = 0b01;   //PWM Generator 1 uses PWM Master Clock, undivided and
unscaled
    PG1CONbits.MODSEL = 0b000;  //PWM Generator 1 operates in Independent Edge PWM mode

    PG1IOCONbits.PMOD = 0b01; //PWM Generator 1 Output Mode is Independent Mode
    PG1IOCONbits.PENH = 1;    //PWM Generator 1 controls the PWM1H output pin
    PG1IOCONbits.PENL = 1;    //PWM Generator 1 controls the PWM1L output pin
    PG1CONbits.ON = 1;
}

int main(void) {

#ifdef use_FRC
```

```
    clock_PWM_at_FRC();
    PG1PER = 0x500;//100kHz with FRC
    PG1DC = 0x250;//100kHz with FRC
#endif

#ifdef use_PLL
    clock_PWM_at_400MHz_from_PLL2_Fout();
    PG1PER = 0xFB11;//100kHz with PLL
    PG1DC = 0x7D81;//100kHz with PLL
#endif

#ifdef use_MEMS
    clock_PWM_at_MEMS();
    PG1PER = 0x500;//100kHz with MEMS
    PG1DC = 0x250;//100kHz with MEMS
#endif

#ifdef use_MEMS_PLL
    clock_PWM_at_400MHz_EC_PLL2_Fout();
    PG1PER = 0xFB11;//100kHz with PLL
    PG1DC = 0x7D81;//100kHz with PLL
#endif

    pwm_Init();

    while(1){
        Nop();
    }
}
```

8.3.1 dsPIC33A Results

Below are the test results for jitter on the PWM output when being clocked by an 8 MHz FRC or clocked by FRC+PLL at 400 MHz. The results from the FRC+PLL test show a more defined center point as well as a lower standard deviation (1.2 ns compared to the FRC test's 1.7 ns).

Figure 8-5. dsPIC33AK128MC106 PWM Jitter Results with FRC

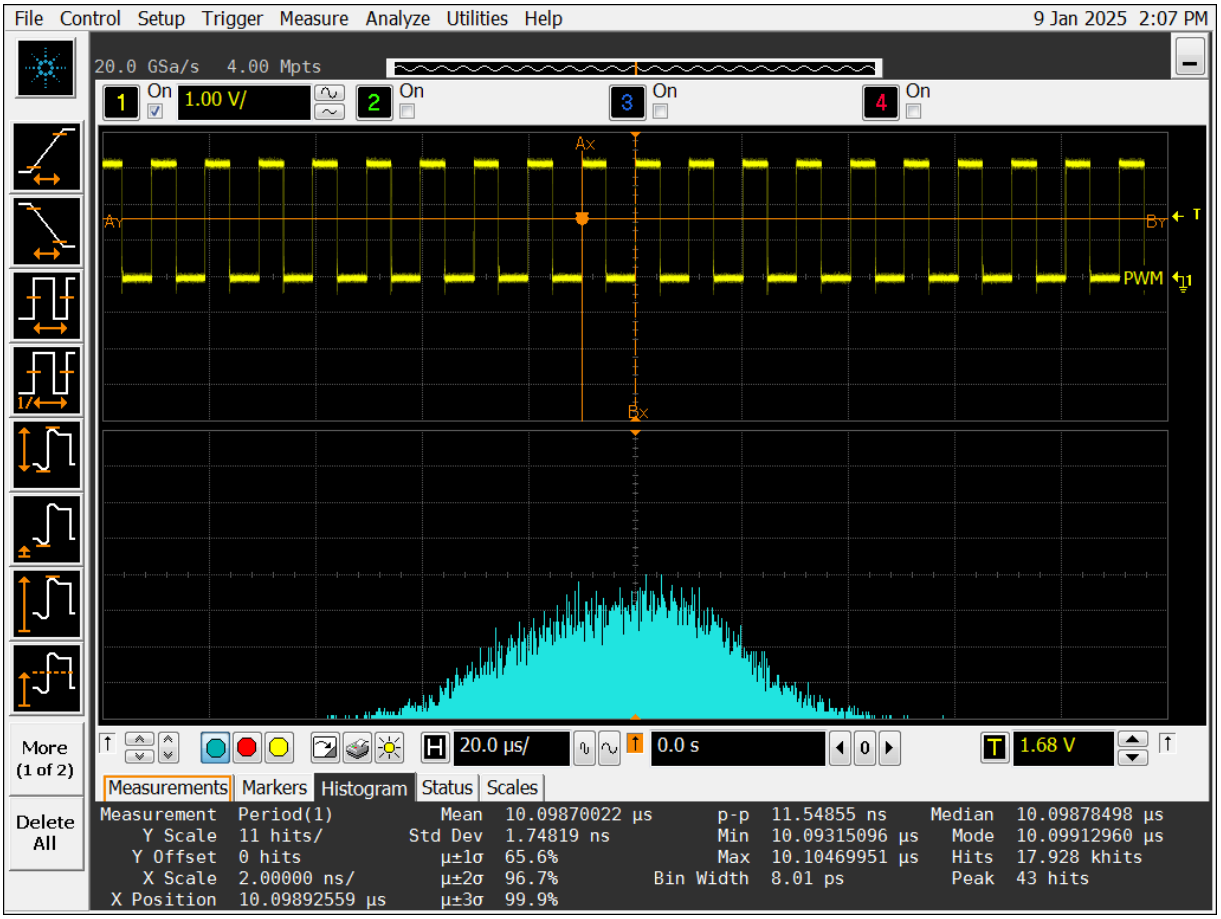


Figure 8-6. dsPIC33AK128MC106 PWM Jitter Results with FRC+PLL

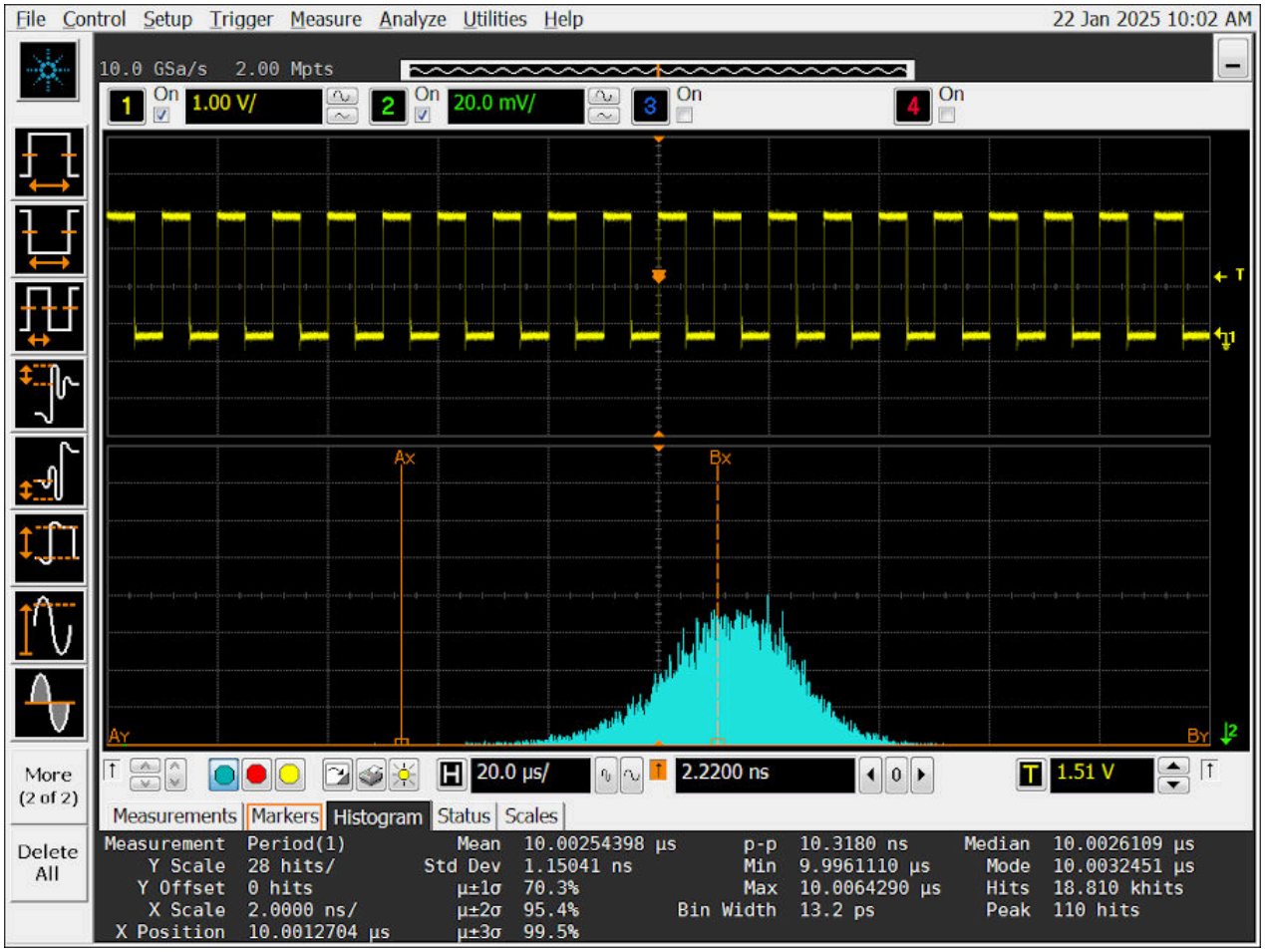


Figure 8-7. dsPIC33AK128MC106 PWM Jitter Results with 8 MHz MEMS

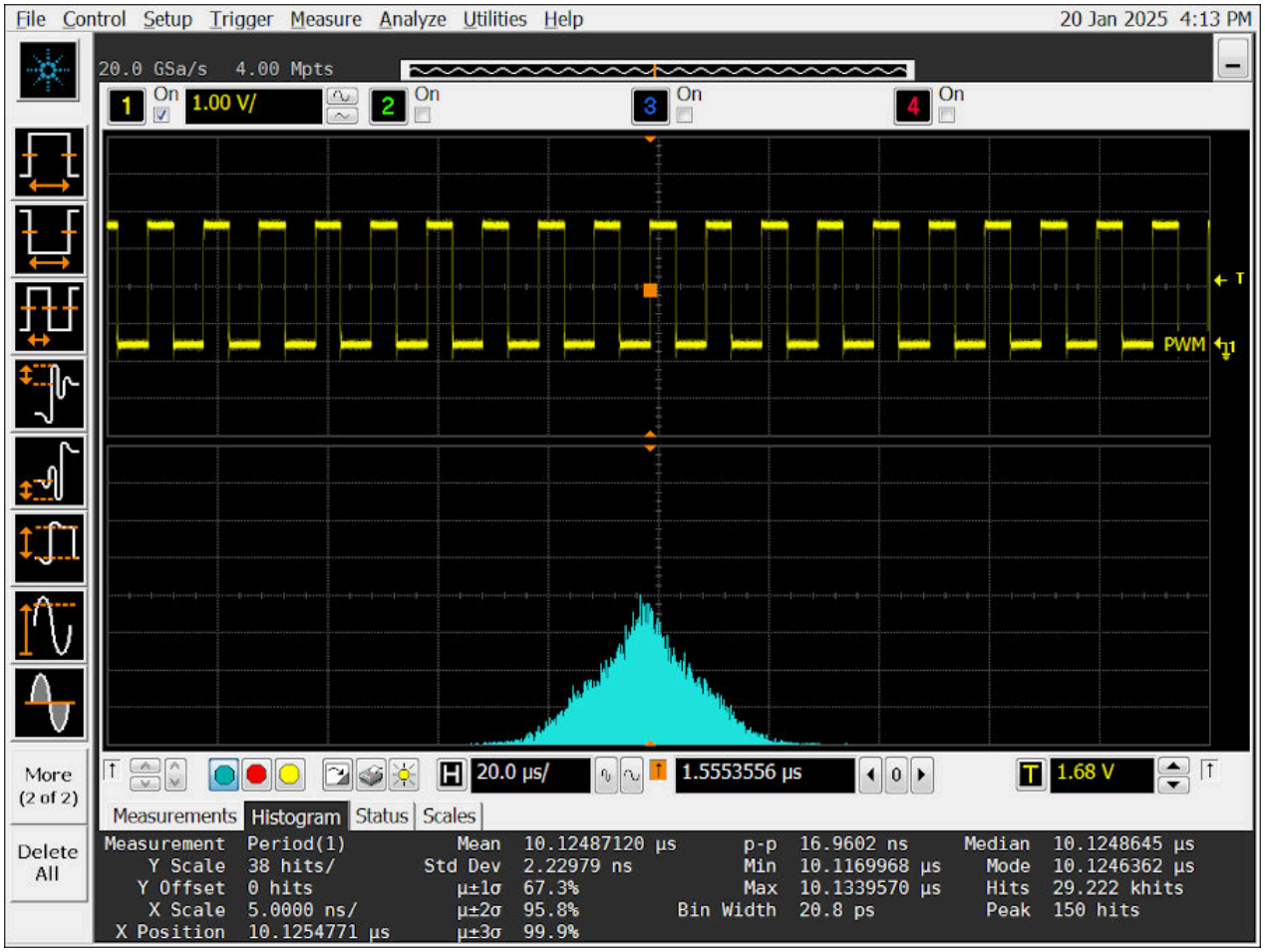
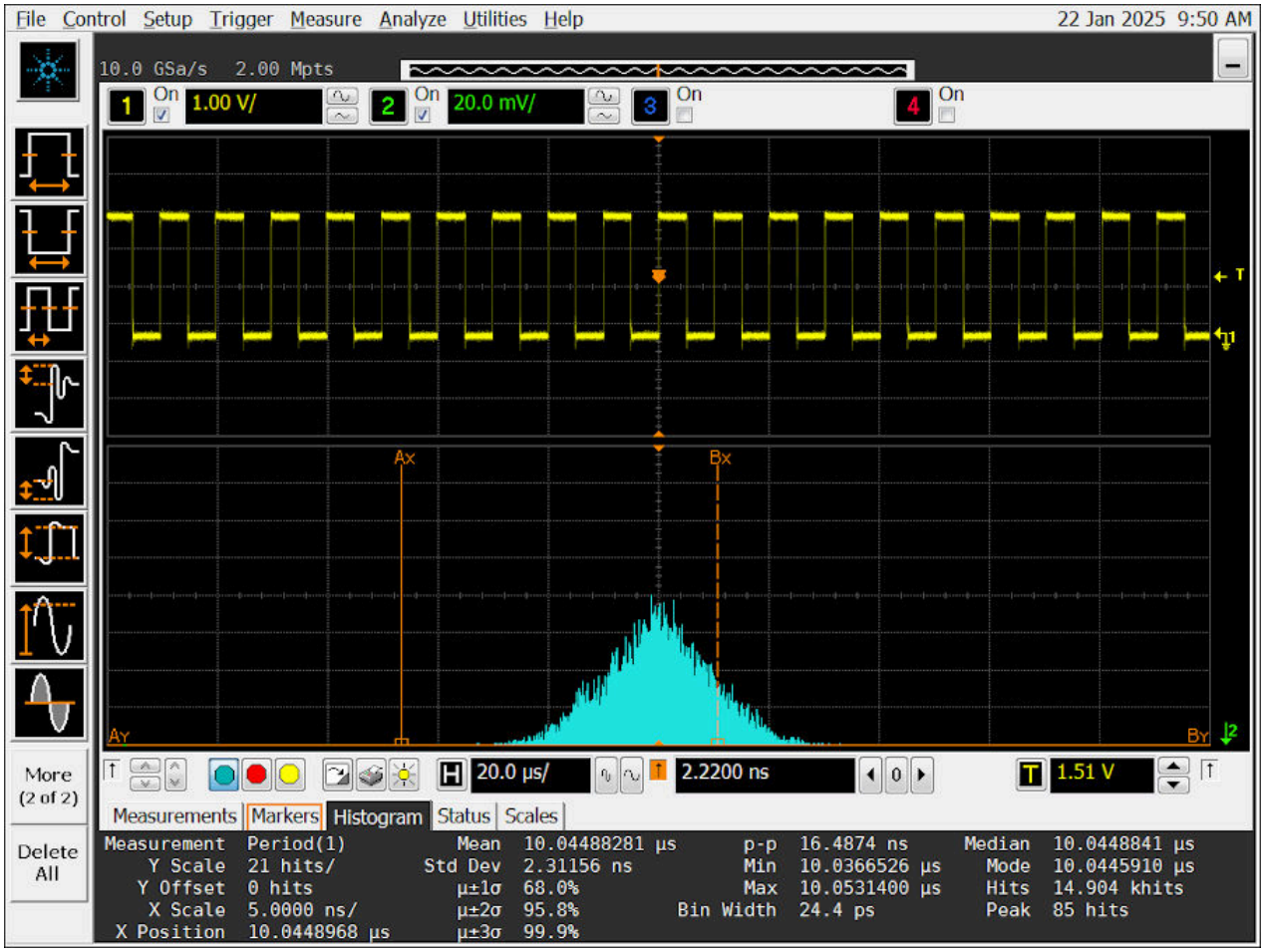


Figure 8-8. dsPIC33AK128MC106 PWM Jitter Results with MEMS + PLL



9. Conclusion

This document sets expectations for jitter in an application and covers how to understand, measure, and reduce jitter when possible. While random jitter will be present in any system, using the tools provided in this document, deterministic jitter can be understood and managed. Using these tools can improve applications, which in turn will produce better results from a device and the surrounding system.

10. Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision	Date	Description
A	3/2025	Initial revision

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-0650-2

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.