了解和管理应用中的抖动

AN5823

概述

本文档旨在帮助了解时钟抖动及其对 PWM 控制环应用的影响，并介绍了如何识别抖动源、如何测量抖动以及如何在设计应用时兼顾抖动问题。

目录

1. 简介

1.1. 什么是抖动？

抖动的定义是与参考信号的时序偏差。抖动是由电子电路中较小的时序偏差或阈值偏差引起。这些偏差可能因环境和系统因素（包括电源噪声和信号完整性）而加剧。以下章节将讨论抖动的类型、分量和量化。随着系统中噪声的增加，相应的抖动量可能会对某些应用产生不利影响。

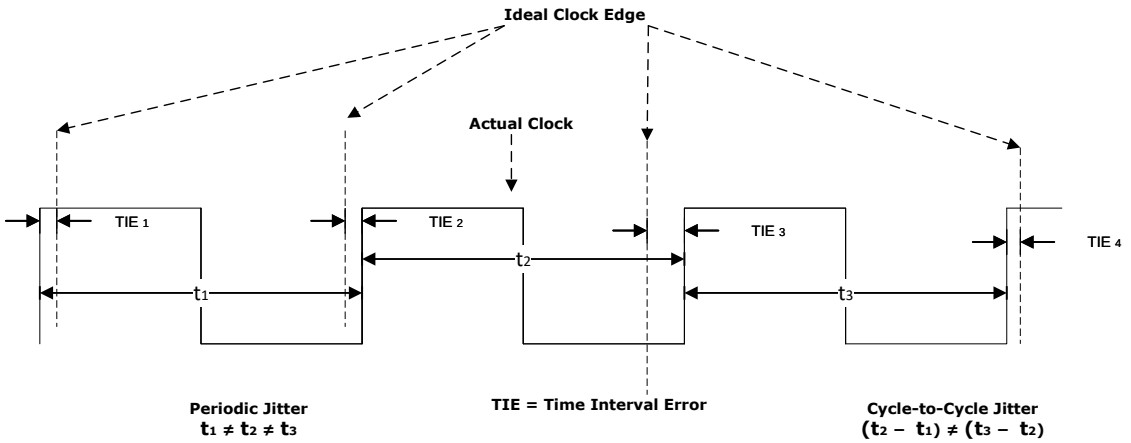
1.2. 设定预期

需要明确的是，抖动不可能完全消除，只能缓解和减少。某些产品的边沿分辨率可能小于系统抖动量，但具体是否会产生问题取决于应用类型。采用平均滤波器或其他滤波器的控制环通常更加不易受抖动引起的误差影响。采用逐周期控制的系统尽管可以容忍抖动，但可能引发非分布均匀的纹波问题。分辨率或抖动参数有时是在理想的测试环境下测得；但在实际应用中，电源开关时会出现瞬变，进而导致基底噪声增加并引发系统抖动。

2. 了解抖动

抖动可通过多种方式进行观测，例如周期抖动、周期间抖动和长期抖动。这些测量方式具有内在关联性，有助于从另一个层面了解抖动的分量及其如何对应用产生影响。下图显示了时钟源抖动指标。

图 2-1. 时钟源抖动指标

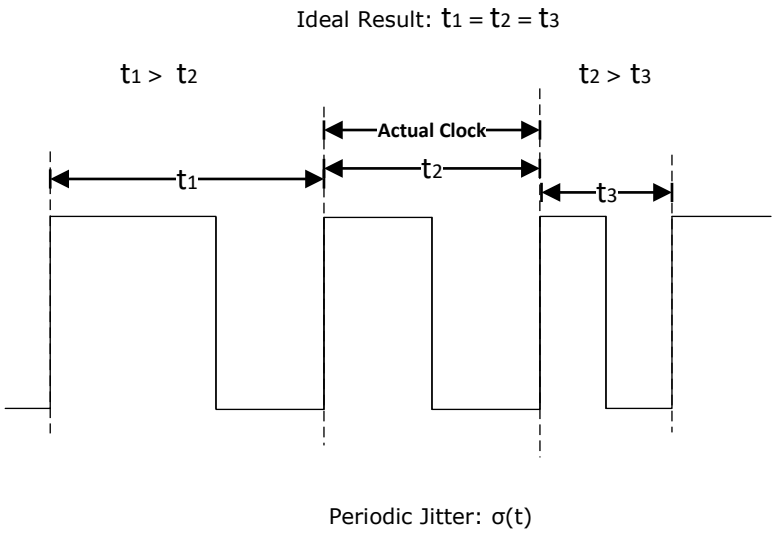


2.1. 周期抖动

周期抖动是指任一时钟周期值偏离预期时钟值的现象。当抖动对时序信号产生影响时，周期值可能会向任一方向偏移，这意味着周期值可能大于或小于理想值。由于存在抖动，每个周期的值都会发生变化。在图 2-2 中，t1 是比预期长的脉冲，t2 符合所需的时钟频率，t3 是比预期短的脉冲。在理想应用中，这三个脉冲的长度完全相同。

下图显示了周期抖动：

图 2-2. 周期抖动



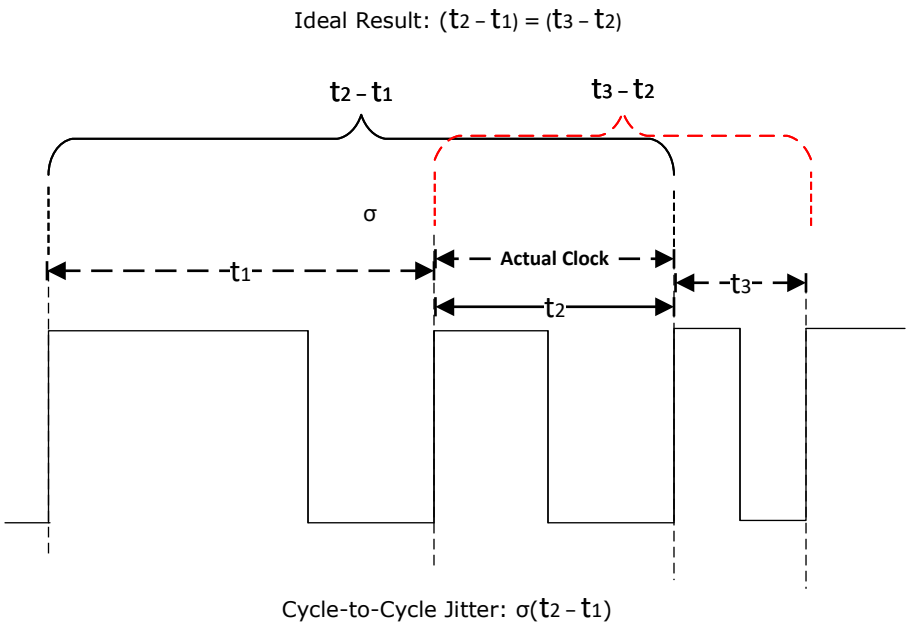
2.2. 周期间抖动

周期间抖动是指相邻时钟周期之间的周期差值，即不同周期的周期抖动之间的关系，如图 2-3 所示，其计算公式如下：

公式 2-1.

$$周期间抖动 = \sigma\ (t2 - t1)$$

图 2-3. 周期间抖动

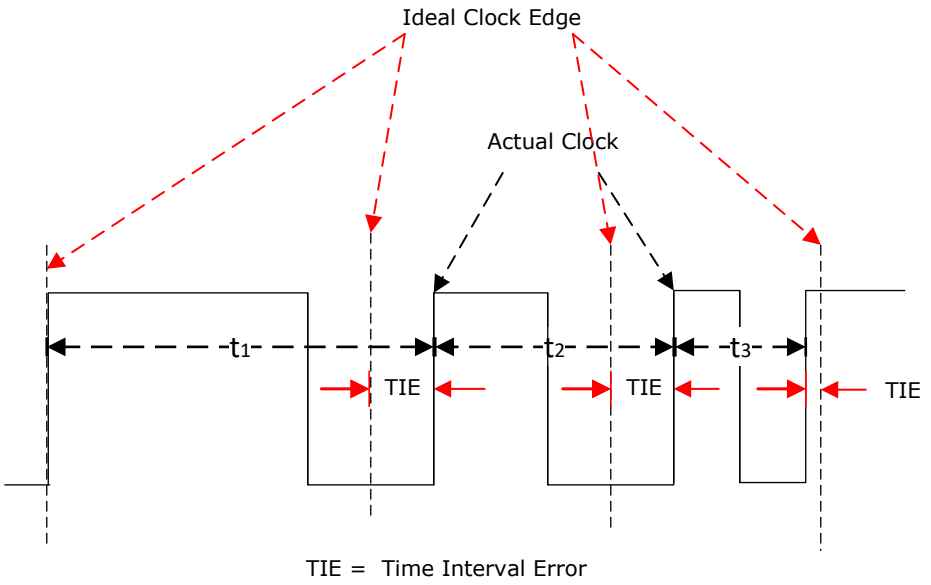


2.3. 时间间隔误差（TIE）

时间间隔误差（Time Interval Error，TIE）是指两次时间测量之差。在图 2-4 中，三个时钟脉冲均未按照预期时间出现。这是因为 t1 的周期比预期长，导致整个波形的时间发生了一定量的偏移。该时间误差即为 TIE，既可以是正值也可以是负值。例如，从 t1 到 t2 和从 t2 到 t3 的 TIE 均为正值，但由于 t3 的周期比正常时间短（TIE 为负值），因此总 TIE 会被平均掉。此外，TIE 的时间尺度还因具体时钟而异。慢速 TIE 可表现为许多带有较小误差的周期，需要经过较长的时间才能在任一方向上累积出明显的误差。TIE 的包络时序与幅值彼此独立，两者会以不同的方式影响系统性能。由于存在随机抖动，因此仍会发生上述偏移，但都处于可控制范围内。

下图显示了时间间隔误差。

图 2-4. 时间间隔误差



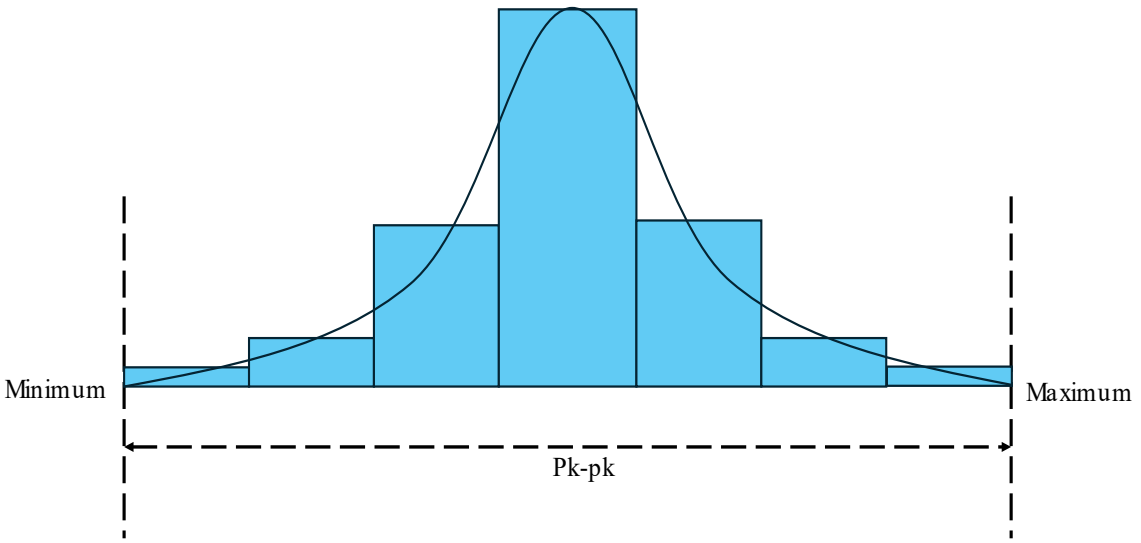
3. 抖动指标

时钟时序抖动有多个测量指标。抖动数据通常采用直方图形式呈现，其中 x 轴表示与理想时钟周期的时序差，y 轴表示周期值的测量次数。

3.1. 最小值/最大值

直方图中的最小值和最大值是位于两端距离最远的两个点，其中最小值是最小周期测量值，最大值是最大周期测量值。峰-峰值测量值是最大值与最小值之差。在分析应用中的整体抖动分布时，需要将这些值考虑在内。但理论上，随着直方图中计数的增加（由于存在随机无界抖动），峰-峰值会无限增大，因此不宜作为评估抖动的主要因素。相比之下，基于直方图的平均值与标准差进行抖动分析更为精确。下图显示了直方图中的最小值和最大值。

图 3-1. 直方图中的最小值/最大值

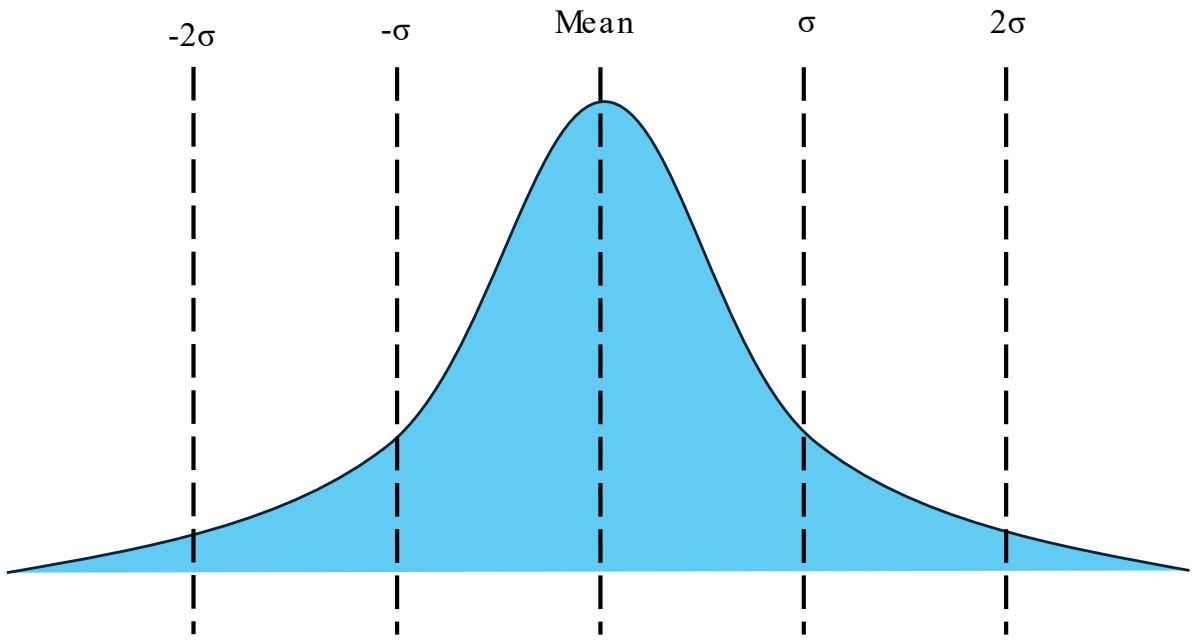


3.2. 标准差（σ）

标准差是一种统计测量，用于量化一组数据值的变化量或离散度。如果计算出的标准差较低，则意味着大部分数据点接近平均值，而标准差较高则意味着离散度较高。在数据呈现高斯分布的正态分布中，约 68% 的数据落在一个标准差的范围（即，-1σ与 1σ之间的空间）内。标准差基于数据平均值计算得出，这两个值是分析抖动时最重要的指标。

下图显示了标准差和平均值。

图 3-2. 标准差和平均值

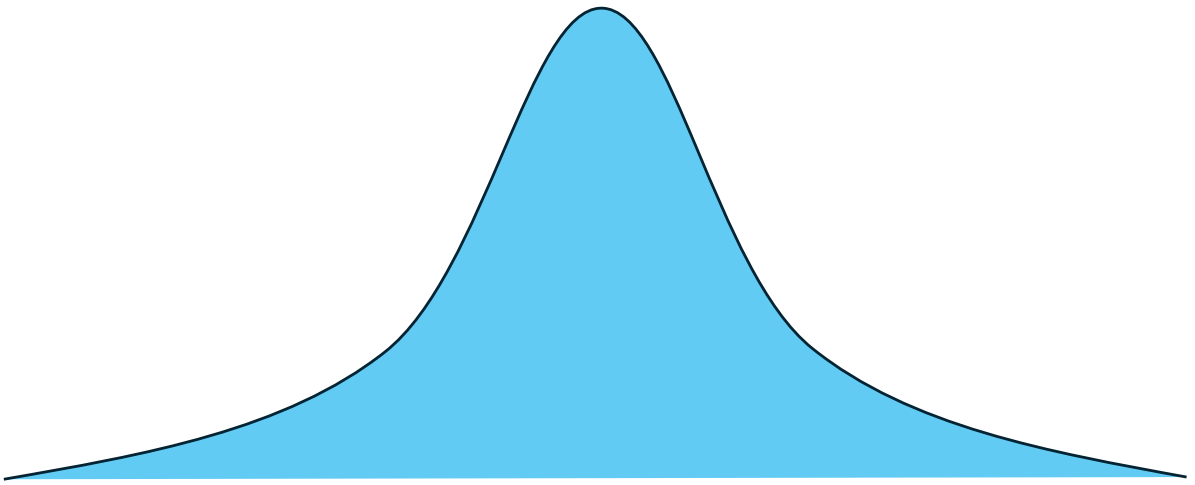


4. 抖动的类型

4.1. 随机抖动

抖动分为两类：随机抖动和确定性抖动。在许多情况下，抖动可能是由电源噪声、硬件设计缺陷、热噪声或串扰引起。所有系统中的抖动都表现为信号周期出现偏差。在仅存在随机抖动的理想情况下，时钟周期的直方图将呈现高斯分布（即，形成钟形曲线），如下图所示。

图 4-1. 随机抖动



高斯分布测量结果表明抖动是“随机的”，既无法识别抖动源也无法消除。随机抖动是无界的，这意味着高斯分布的尾部延伸至无穷大，但出现较大偏差的概率呈指数下降。

4.2. 确定性抖动

在同时存在随机抖动和确定性抖动的系统中，曲线的峰值可能会向任一侧偏移，甚至可能出现多个峰值。

图 4-2. 确定性抖动——峰值偏移

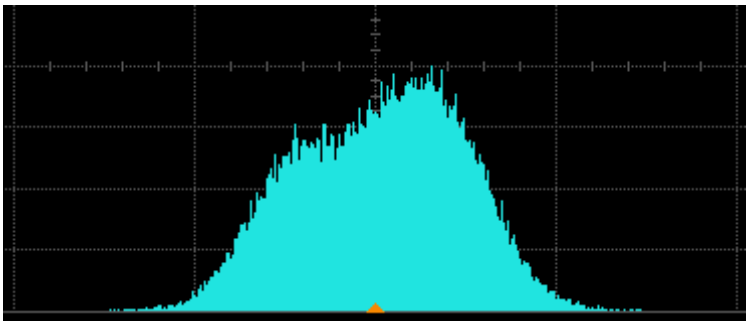
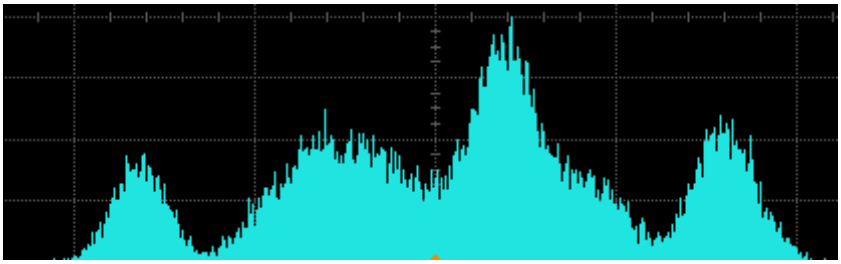
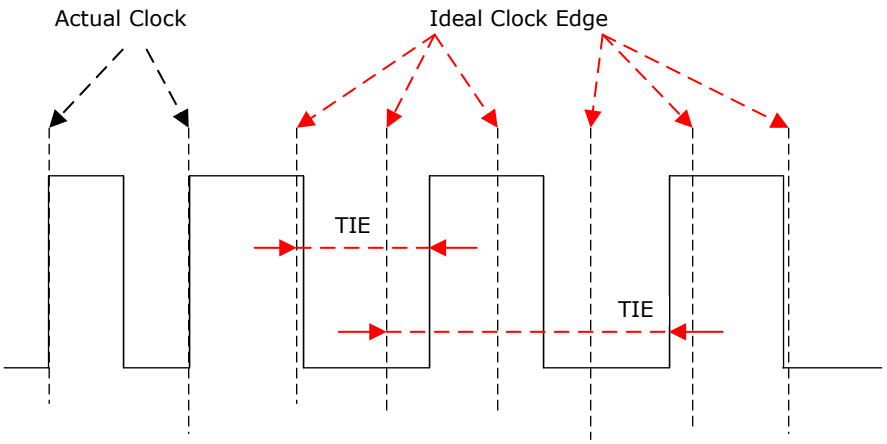


图 4-3. 确定性抖动——多个峰值



上面两张图展示了确定性抖动在应用中的可能表现形式。有时不是单一高斯峰，而是多个高斯峰的组合。这说明应用中有某种确定性行为在反复影响周期时间。在图 4-2 中，曲线的峰值偏移至直方图的右侧，这意味着平均时钟周期大于预期时钟周期。这是长期抖动的一个示例，即时钟周期随时间推移沿单一方向持续累积偏差。在这种情况下，TIE 可能会非常大，因为时钟周期值始终大于所需的周期值。 图 4-4 展示了确定性抖动如何对 TIE 产生长期影响。请注意，TIE 会增大是由于两个连续的时钟脉冲均比预期长。

图 4-4. 长期抖动



5. 测量技术

测量技术与测试条件会对结果产生很大影响。所用设备需具备与信号速度和边沿速率相匹配的测量能力。应注意探头接地引线长度，以尽量减少不准确的幅值。部分设备可能内置时钟/抖动分析工具或软件。可能需要执行多项测试来识别和量化系统抖动。沿着电路路径进行测量有助于识别抖动源。下面列出了一些典型的测试点：

- 时钟源（如果位于单片机外部）
- 连接至某个引脚的单片机内部时钟源
- 连接至某个引脚的锁相环（Phase-Locked Loop，PLL）输出
- PWM 输出引脚
- 栅极驱动器输出

鉴于单片机及其他电路可能引入系统噪声，隔离噪声源并比较测量前后的变化会有所帮助。例如，暂停通信线路或其他 PWM 信号有助于评估其对目标信号的影响。

5.1. 测量步骤

按照下列步骤测量抖动：

1. 设置被测器件（Device Under Test，DUT），包括电源和配置参数。
2. 将示波器探头连接到被指定为参考输出的引脚。切勿使用示波器的平均模式，否则可能会掩盖结果。
3. 根据要测量的抖动类型要求配置示波器。记录相邻周期可以计算周期间抖动以及周期抖动和 TIE。
4. 重复执行步骤 3，至少 10000 次。大量样本将有助于确保识别任何长期抖动。
5. 根据测试结果创建直方图并计算平均值和标准差。直方图将显示每个周期的测量次数。
6. 使用这些数据确定是否可以识别任何确定性抖动以及抖动水平是否在应用的可接受范围内。
7. 如果发现确定性抖动（例如，直方图上显示多个峰值），可考虑使用频谱分析仪来识别哪个频率产生了额外的峰值。
8. 按照下述说明来缓解确定性抖动。如果使用频谱分析仪识别到超出目标时序信号的任何其他频率范围，请重点关注下一章中可以很好地解决该问题的步骤。

6. 缓解抖动

在明确不同的抖动分量和类型对系统的潜在影响后，需想办法减少系统中的确定性抖动。随机分量无法消除。在前面的各节中，我们讨论了抖动成因。下面列出了一些减少抖动的方法：

- 源输入时钟
- PLL 滤波
- 电源降噪
- PCB 和系统设计

6.1. 时钟源抖动

尽管并非必须使用外部振荡器，但实践表明：与内部时钟源相比，使用高速 MEMS 或晶振电路作为参考时钟可以减少器件抖动。此外，将更高频率的输入时钟与 dsPIC® PLL 配合使用时，还可以实现更高的时钟精度。这是因为每隔几个时钟周期就会执行一次 VCO 频率校正，这意味着采用高于默认 8 MHz FRC 速度的外部时钟频率将能够更频繁地进行 VCO 校正。必须对系统进行测试以明确应用是否需要更高的时钟精度，或者系统中的抖动水平是否在可接受范围内。

6.2. 锁相环（PLL）

在 PWM 应用中，通常使用 PLL 来实现所需的速度和分辨率。PLL 可根据其设计和滤波方式来改善或减少抖动。PLL 的性能取决于其工作参数，其中包括反馈倍频比和 VCO 频率。通过配置 PLL 的预分频比或后分频比来优化 VCO 频率有助于减少确定性抖动。

6.3. 电源

捕捉多个电源的抖动直方图有助于识别系统中的抖动源。例如，可以考虑使用板上电源与实验室电源进行对比测试。如果测试结果表明使用 PCB 上设计的电源电路时抖动要高得多，则需要对电路进行一些调整。

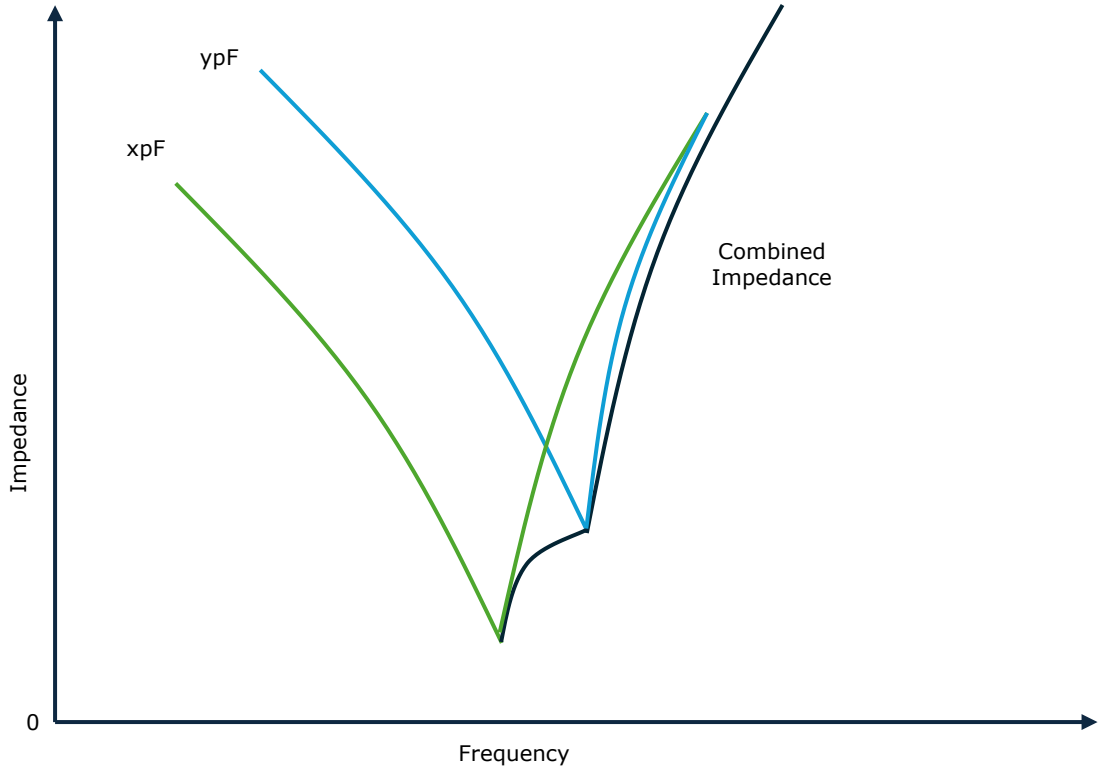
6.4. 电路调整

从电源走线的设计开始，比如走线的长度。连接至目标所需的走线越长，就越容易受到噪声的影响。解决这一问题最高效的方法是采用至少 4 层的 PCB，其中 VDD 与 VSS 应作为独立层进行设计。这种布线可确保接地路径的长度不会超过网络走线长度与过孔接地走线长度的总和。为了进一步限制走线的长度，应确保各元件彼此靠近放置，尤其是 DUT 旁边的旁路电容。此外，为两个高速信号选择器件引脚时，切勿选择相邻的引脚，除非这两个引脚需要一起使用。例如，在 PWM 旁边使用 I2C 会在两个引脚之间引入噪声，进而增加抖动并降低 PWM 分辨率。如果 I2C 与 PWM 使用不同的时钟源，则噪声会进一步增加，因为异步信号会比同步信号注入更多的误差。

6.5. 元件

元件选型可能会对系统噪声产生影响。选择旁路电容值时应确保其在所需工作频率下呈现最低阻抗。这意味着，需要知道哪些电容最适合在引脚翻转状态时用作电源的滤波器。添加这些额外的旁路电容可能会超出器件数据手册中建议的电容配置，并且还会占用 PCB 上的空间，因此请务必查阅电容的数据手册并根据应用频率仅选择需要的值。下图显示了这些数据手册中常见的曲线图。

图 6-1. 阻抗与频率之间的关系



6.6. 其他因素

其他变量（如温度）也可能会影响器件的抖动结果。下面以 Curiosity 平台开发板上的 dsPIC33AK128MC106 GP DIM 为例进行说明，其 PWM 输出频率为 100 kHz。从下面的两张图中可以看出，温度升高会极大地影响 PWM 的输出。随着温度的升高，确定性抖动也会增加，进而导致周期时间延长。

图 6-2. PWM 100 kHz（室温）

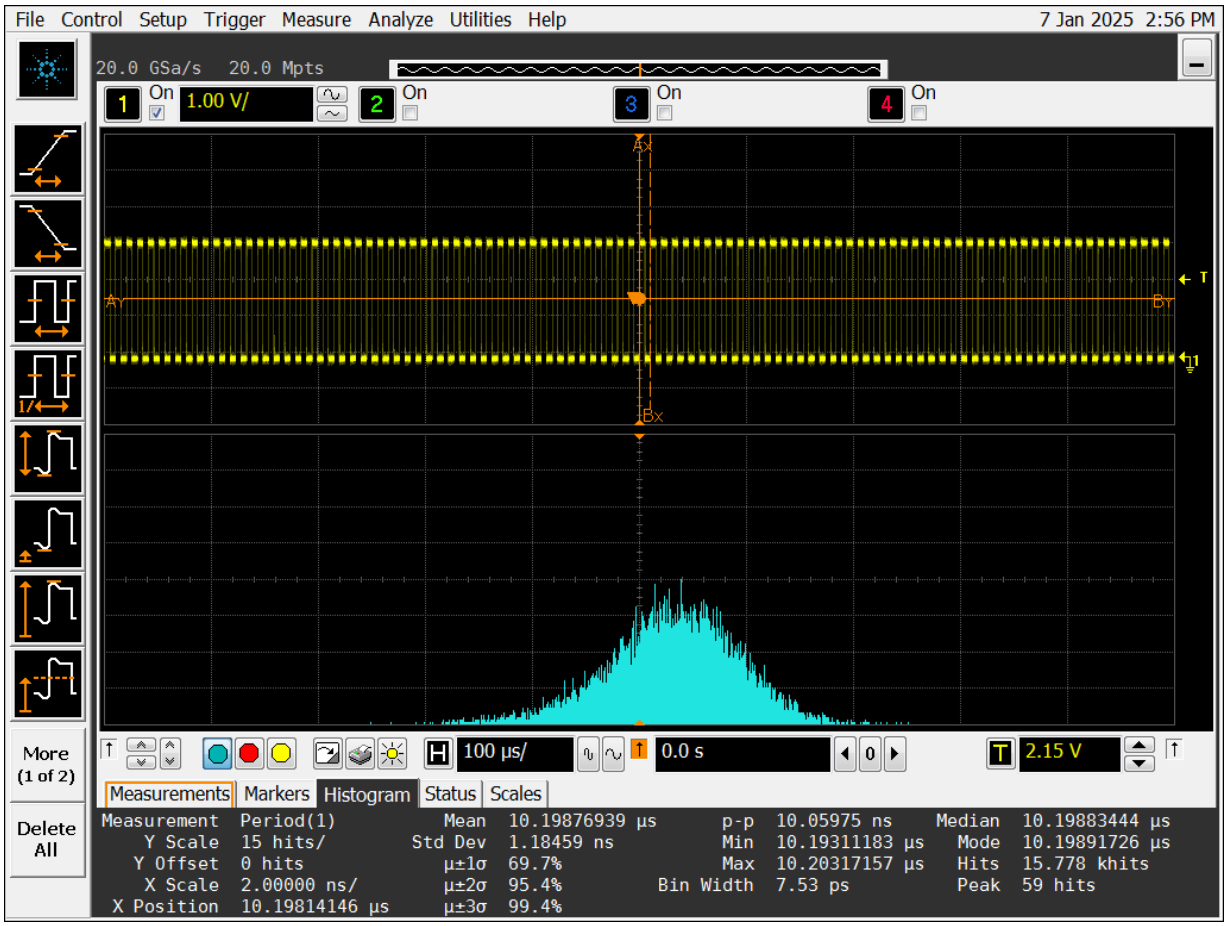
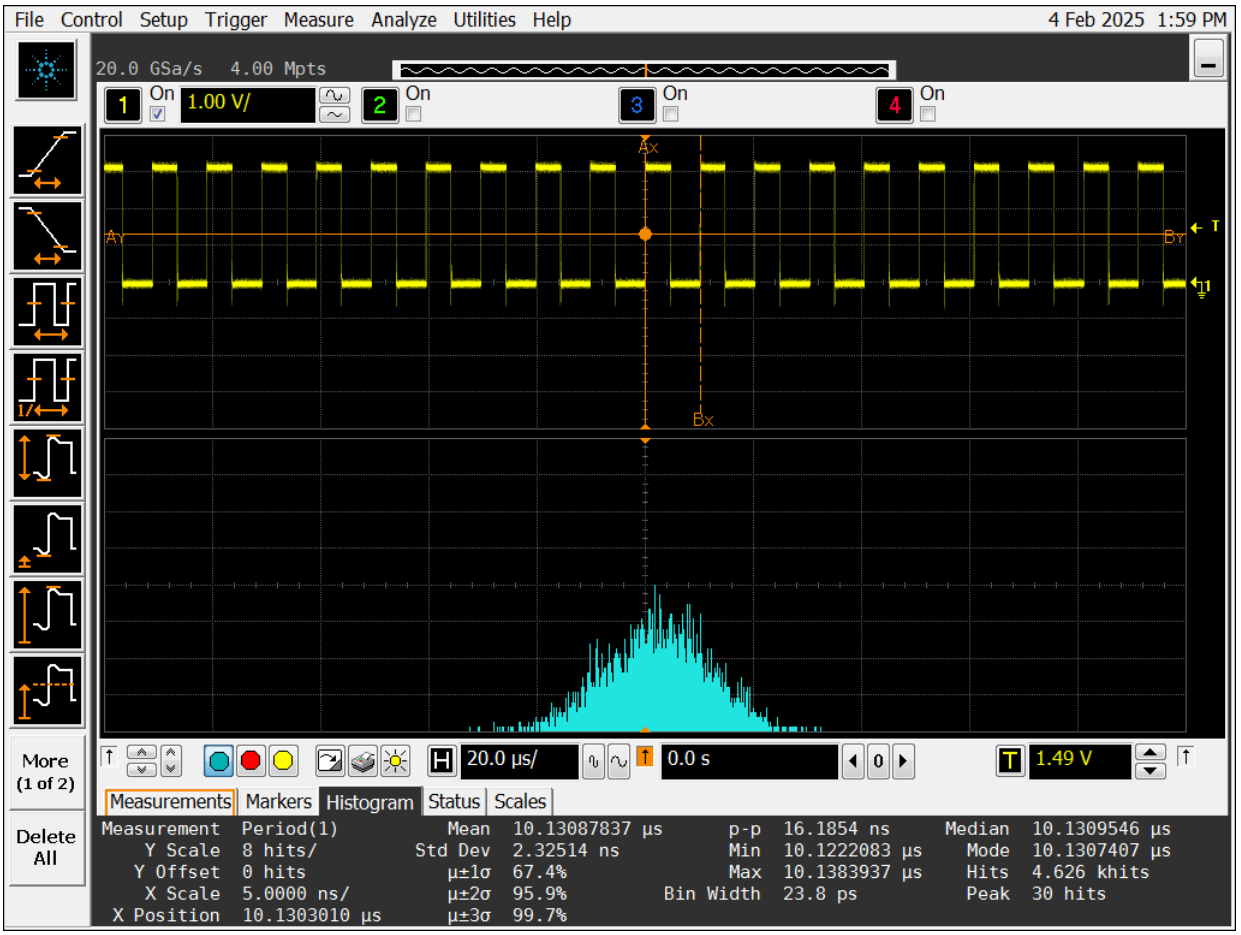
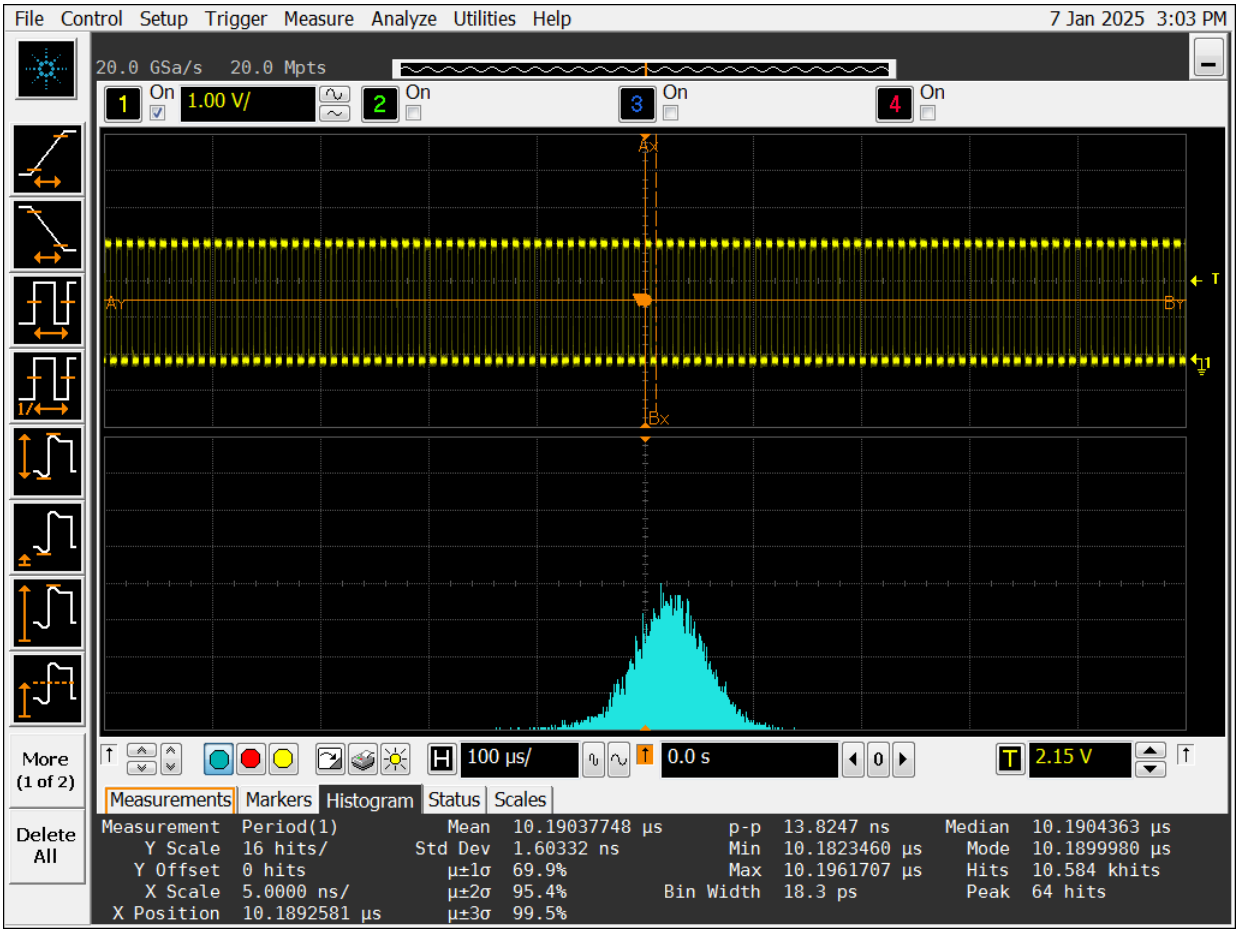


图 6-3. PWM 100 kHz（高温）



此外，温度降至室温以下也可能会影响 PWM 输出。

图 6-4. PWM 100 kHz（低温）



除了温度之外，抖动还可能因具体器件而异，并且可能随着时间而发生变化。这可能是由于制造工艺、元件退化和电源稳定性造成的。为确保器件寿命并防止抖动随时间增加，最有效的方法是保证器件在受控环境下工作。随着时间的推移，噪声越低的电源给系统带来的压力越小，并且保护器件免受高温影响将有助于减缓老化过程。

7. 最佳性能的时钟配置示例

输入时钟源和 PLL 配置可能对 PWM 输出抖动产生很大影响。本章举例说明了如何在 dsPIC PWM 应用中实现最佳性能。本例中使用的所有公式均基于以下公式：

公式 7-1.

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

采用较小反馈分频值（PLLFBDIV）的 PLL 电路可以最大限度地减少抖动。反馈分频比值越小，滤波器环路的周期越短，对输入信号偏差的校正响应就越快。在本例中，最小 VCO 频率（如器件数据手册的电气特性中规定）为 400 MHz。可以通过多种方式使用 PLL 获得 400 MHz 的输出时钟频率。如果使用 8 MHz FRC，则可以使用以下公式设置 PLL：

公式 7-2.

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

对于 8 MHz FRC，反馈分频比值为 100，这意味着增益倍数为 100。

使用更高频率的高精度源时钟可以进一步减少抖动。如果 PLL 参考时钟频率从 8 MHz 增加至 25 MHz 外部时钟（MEMS 或晶振），则该公式会得出较小的反馈分频比值 32。

公式 7-3.

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

通过增大参考时钟的频率并相应地调整 PLL 反馈分频比，可以优化 PLL 性能。这样会极大地减少给定信号的潜在抖动。本例中使用了两种缓解抖动的方法，但应结合前文提到的其他方法来减少确定性抖动，从而为 PWM 提供最佳参考时钟。

若要测试 dsPIC33A 器件上的抖动，需使用 dsPIC33AK128MC106 GP DIM 和 Curiosity 平台开发板。以下示例代码将使用 mikroBUS™插座 A 的引脚 16 来输出 PWM 信号。每个示例都使用不同的时钟源来测量抖动。第一个示例使用 dsPIC 内置的 8 MHz FRC 振荡器，PWM 基于该时钟输出 100 kHz 左右的频率。第二个示例同样使用该 FRC，另外还使用 PLL 将输出时钟信号设置为 200 MHz。PWM 信号分频器经过相应的调整后仍输出 100 kHz。在这两种情况下，都可以使用示波器测量 mikroBUS 接口 A 的引脚 16 上的抖动。

8. 不同器件系列示例

下面分别以 dsPIC33E、dsPIC33C 和 dsPIC33A 器件系列为例提供了测试设置和结果。每个测试都提供 FRC 和 FRC+PLL 两种时钟源选项，因此可使用 FRC 结果作为参考点与其他结果进行比较。可以在时钟 REFO 外设和 PWM 输出上测试抖动。这些测试设置所采用的开发板均支持用户自主采购，便于用户复现测试设置。此外，还可轻松修改占空比和周期以进行更具体的测试，例如占空比关联性（duty cycle-duty cycle）和 TIE。

8.1. dsPIC33E

dsPIC33E 测试使用带有相应 dsPIC33EP128GS806 DP PIM 的数字电源开发板版本。PWM 输出 1 在代码中配置，模块使用 FRC 或 FRC+PLL 作为其时钟源，具体取决于代码开头的定义。无论使用哪种时钟源，PWM 输出均为 100 kHz，用于抖动测量和两个结果的比较。

```
// <editor-fold defaultstate="collapsed" desc="Config Bits">
// FICD
#pragma config ICS = PGD1                   // ICD 通信通道选择位（通过 PGEC1 和 PGED1 进行通信）
//#pragma config JTAGEN = OFF               // JTAG 使能位（禁止 JTAG）

// FPOR
#pragma config ALTI2C1 = OFF                // 备用 I2C1 引脚（I2C1 映射到 SDA1/SCL1 引脚）
//#pragma config ALTI2C2 = OFF              // 备用 I2C2 引脚（I2C2 映射到 SDA2/SCL2 引脚）
#pragma config WDTWIN = WIN25               // 看门狗窗口选择位（WDT 窗口为 WDT 周期的 25%）

// FWDT
#pragma config WDTPOST = PS32768            // 看门狗定时器后分频比位（1:32,768）
#pragma config WDTPRE = PR128               // 看门狗定时器预分频比位（1:128）
#pragma config PLLKEN = ON                  // PLL 锁定使能位（时钟切换至 PLL 源需等待 PLL 锁定信号生效。）
#pragma config WINDIS = OFF                 // 看门狗定时器窗口使能位（看门狗定时器处于非窗口模式）
#pragma config WDTEN = OFF                 // 看门狗定时器使能位（始终使能看门狗定时器）

// FOSC
#pragma config POSCMD = NONE                // 主振荡器模式选择位（禁止主振荡器）
#pragma config OSCIOFNC = ON                // OSC2 引脚功能位（OSC2 为时钟输出）
#pragma config IOL1WAY = ON                 // 外设引脚选择配置（只允许重新配置一次）
#pragma config FCKSM = CSECMD               // 时钟切换模式位（使能时钟切换，禁止故障保护时钟监视器）

// FOSCSEL
#pragma config FNOSC = FRC                  // 振荡器源选择（内部快速 RC（FRC））
#pragma config PWMLOCK = OFF                // PWM 锁定使能位（只能在密钥序列后写入某些 PWM 寄存器）
#pragma config IESO = ON                    // 双速振荡器启动使能位（使用 FRC 启动器件，然后切换至用户选择的
振荡器源）

// FGS
//#pragma config GWRP = OFF                 // 通用段写保护位（可写入通用段）
//#pragma config GCP = OFF                  // 通用段代码保护位（禁止通用段代码保护）
// </editor-fold>

#include<xc.h>

#define use_PLL
//#define use_FRC

int main(void) {

    _TRISA4 = 0;
    _TRISC13 = 0;
#ifdef use_FRC
    PTPER = 4000000 / 6840;                 // 使用 FRC 时的周期设置
#endif

#ifdef use_PLL
    //PTPER = FCY/(PWM 频率 * 预分频比)
    PTPER = 40000000 / 68400;                 // 使用 FRC+PLL 时的周期设置
    CLKDIVbits.PLLPRE = 0;
    PLLFBD = 40;
    CLKDIVbits.PLLPOST = 0b01;
    CLKDIVbits.PLLPRE = 0;
    // 初始时钟切换至 FRC+PLL（NOSC = 1）
```

```
    __builtin_write_OSCCONH(0x01);       // NOSC = 1 -> 设置新的振荡器源
    __builtin_write_OSCCONL(OSCCONL | 0x01); // OSWEN = 1 -> 请求时钟切换
    while(OSCCONbits.OSWEN != 0);        // 等待时钟切换
    while(OSCCONbits.LOCK != 1);         // 等待 PLL 锁定
#endif
    //REFO 设置
    _RP52R = 0b110001;                   // REFO PPS
    _RODIV = 1;                          // REFO 分频比
    _REFOMD = 0;                         // 使能参考时钟模块
    _ROON = 1;                           // 参考时钟输出开启
    _ROSEL = 0;                          // 参考时钟输出为系统时钟

    PTCONbits.PTEN = 0;                  // 在设置期间禁止 PWM 模块
    PWMCON1bits.ITB = 0;                 // PTPER 寄存器提供 PWM1 的时基
    PTCONbits.PTSIDL = 0;                // PWM 时基处于自由运行模式
    PTCON2bits.PCLKDIV = 0;              // PWM 时基输入时钟周期为 TCY（1:1 预分频比）

    // 配置 PWM1
    IOCON1bits.PENH = 1;                 // PWMx 模块控制 PWMxH 引脚
    IOCON1bits.PENL = 0;                 // GPIO 模块控制 PWMxL 引脚
    IOCON1bits.PMOD = 3;                 // PWM1 处于独立模式
    FCLCON1bits.FLTMOD = 0b11;           // 禁止故障模式

    PDC1 = PTPER / 2; // PTPER 的 50%
    PTCONbits.PTEN = 1;                  // 在配置完所有其他设置后使能 PWM

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

8.1.1. dsPIC33E 结果

下面给出了 dsPIC33EP128GS806 DP PIM 的抖动测试结果。

图 8-1. dsPIC33EP128GS806 PWM 抖动结果（使用 FRC）

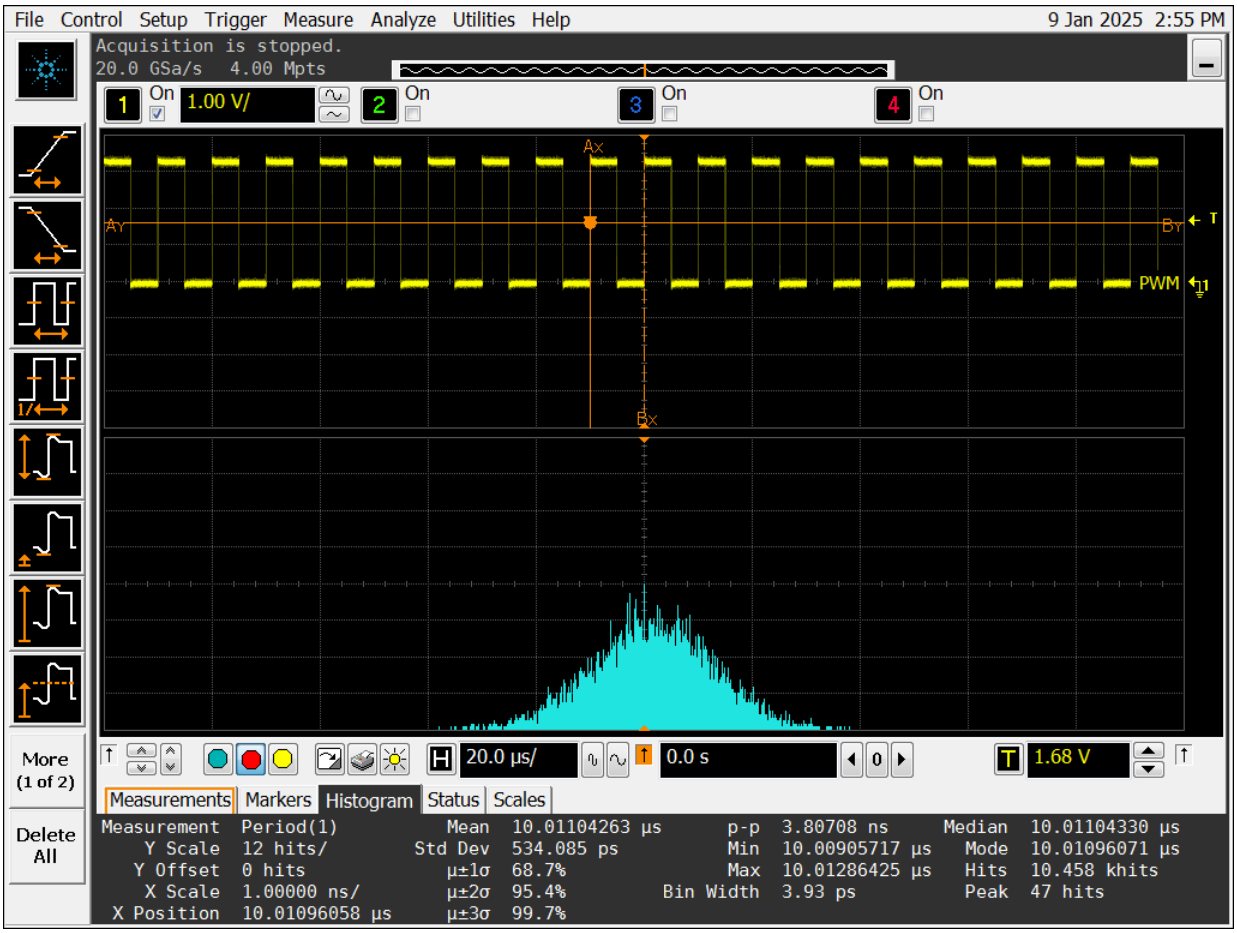
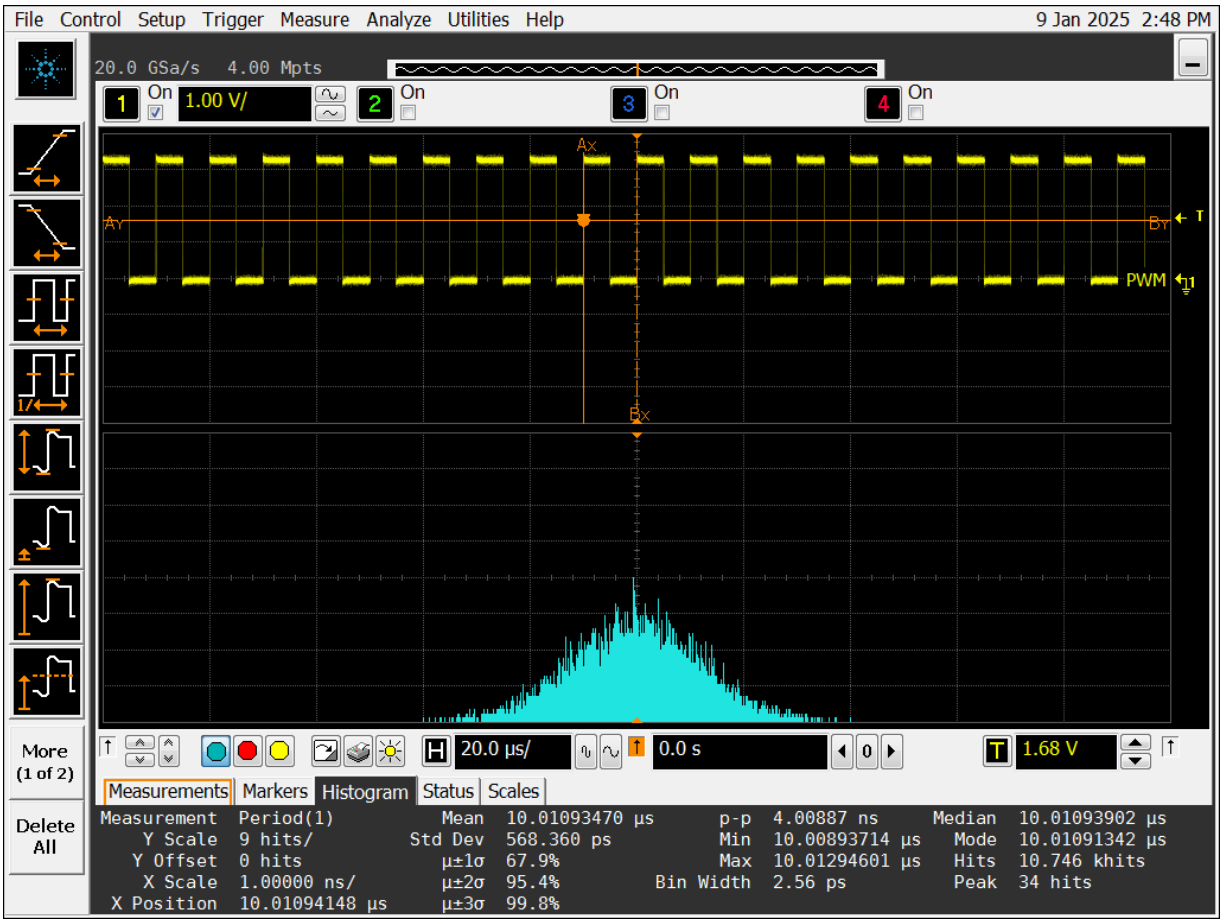


图 8-2. dsPIC33EP128GS806 PWM 抖动结果（使用 FRC+PLL）



8.2. dsPIC33C

dsPIC33C 测试使用带 dsPIC33CK1024MP710 的 dsPIC33C Touch CAN LIN Curiosity 开发板。提供的示例代码可用于测试 PWM 和时钟参考输出（REFO）外设。这两个输出均可使用 FRC 或 PLL 进行测试，具体取决于代码开头的定义。

```
// <editor-fold defaultstate="collapsed" desc="Config Bits">
// FOSCSE
#pragma config FNOSC = FRCDIVN          // 振荡器源选择（带后分频器的内部快速 RC（FRC）振荡器）
#pragma config IESO = ON                // 双速振荡器启动使能位（使用 FRC 启动器件，然后切换至用户选择的振荡器源）

// FOSC
#pragma config POSCMD = NONE            // 主振荡器模式选择位（禁止主振荡器）
#pragma config OSCIOFNC = OFF           // OSC2 引脚功能位（OSC2 为时钟输出）
#pragma config FCKSM = CSECMD           // 时钟切换模式位（使能时钟切换，禁止故障保护时钟监视器）
#pragma config PLLKEN = ON              // PLL 锁定状态控制（如果锁定丢失，PLL 锁定信号将用于禁止 PLL 时钟输出）
#pragma config XTCFG = G3               // XT 配置（24 MHz-32 MHz 晶振）
#pragma config XTBST = ENABLE           // XT 加速（加快启动）

// FICD
#pragma config ICS = PGD1               // ICD 通信通道选择位（通过 PGC1 和 PGD1 进行通信）
#pragma config JTAGEN = OFF             // JTAG 使能位（禁止 JTAG）
#pragma config NOBTSWP = DISABLED       // BOOTSWP 指令禁止位（禁止 BOOTSWP 指令）
// </editor-fold>

#include "xc.h"
#include <libpic30.h>
//#define use_PLL
#define use_FRC
```

```
int main(void) {
    _RP52R = 14;                                // 器件上的引脚 80（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 -> 设置新的振荡器源
    __builtin_write_OSCCONL(OSCCONL | 0x01);// OSWEN = 1 -> 请求时钟切换
    while(OSCCONbits.OSWEN != 0);               // 等待时钟切换
    while(OSCCONbits.LOCK != 1);                // 等待 PLL 锁定
    PG7PER = 0x200;                              // 设置 FRC+PLL 的周期
    PG7DC = 0x100;                               // 设置 FRC+PLL 的占空比
#endif

#ifdef use_FRC
    PG7PER = 0x50;                              // 设置 FRC 时钟源的周期
    PG7DC = 0x25;                               // 设置 FRC 时钟源的占空比
#endif

    _ANSELE0 = 0;                               // E0 上的数字功能
    _ROOUT = 1;                                 // 参考时钟输出使能
    _ROSEL = 0;                                 // 参考源为系统时钟
    _RODIV = 0;                                 // 基本时钟不分频
    _ROEN = 1;                                  // 使能参考时钟
    _TRISD1 = 0;                                // D1 设为数字输出

    PCLKCONbits.MCLKSEL = 0b00;                 // 0 为 FOSC，2 为 PLL 分频器输出
    PG7CONHbits.TRGMOD = 1;                     //可重新触发模式
    PG7CONLbits.CLKSEL = 0b01;                  // 使用通过 MCLKSEL 设置的时钟

    PG7IOCONHbits.PMOD = 1;                     // PWM 处于独立模式
    PG7IOCONHbits.PENH = 1;                     // PWM 控制引脚的输出
    PG7CONLbits.ON = 1;                         // 使能 PWM 7

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

8.2.1. dsPIC33C 结果

下面分别给出了使用 8 MHz FRC 和 FRC+PLL 作为时钟时，在 dsPIC33C Touch CAN LIN Curiosity 开发板上使用提供的测试代码得出的 PWM 输出直方图。

图 8-3. dsPIC33CK1024MP710 PWM 抖动结果（使用 FRC）

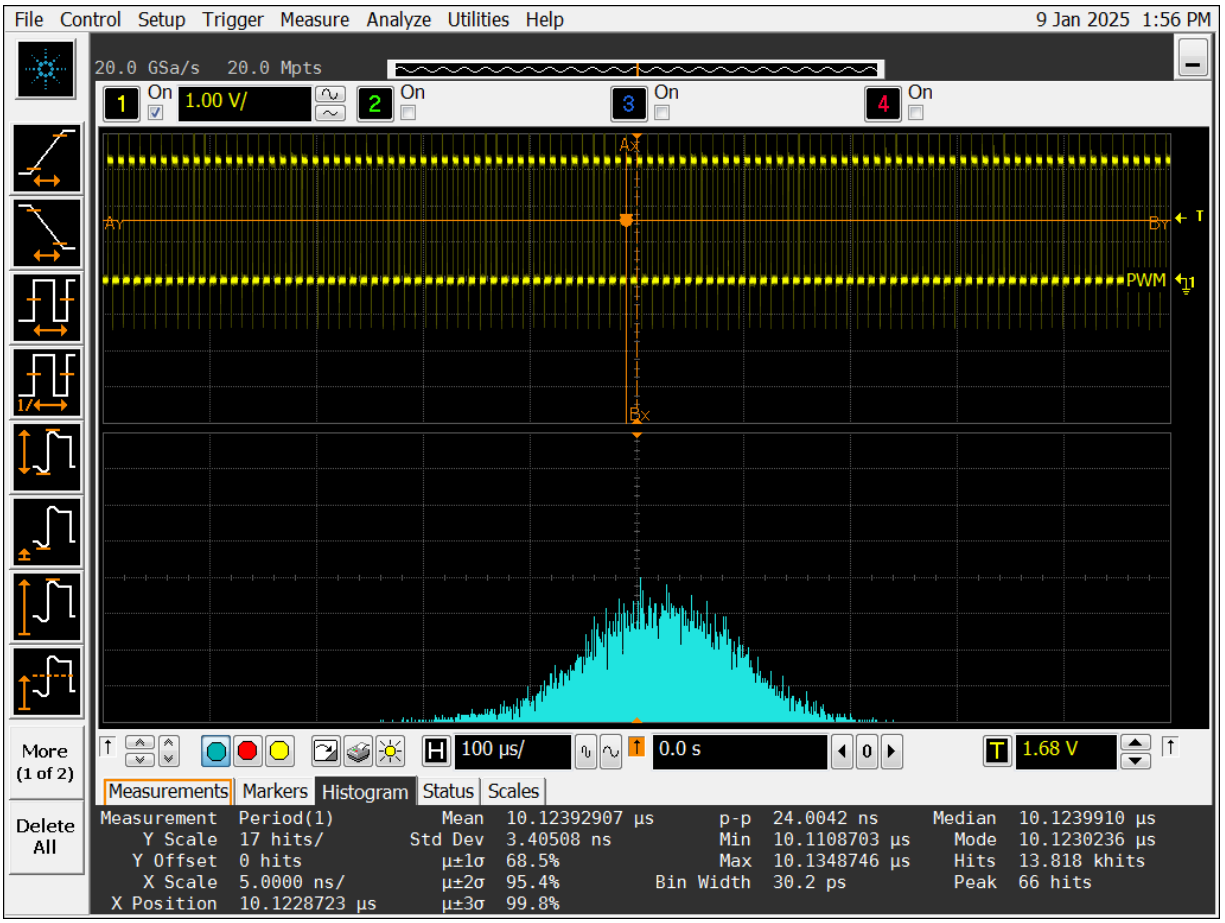
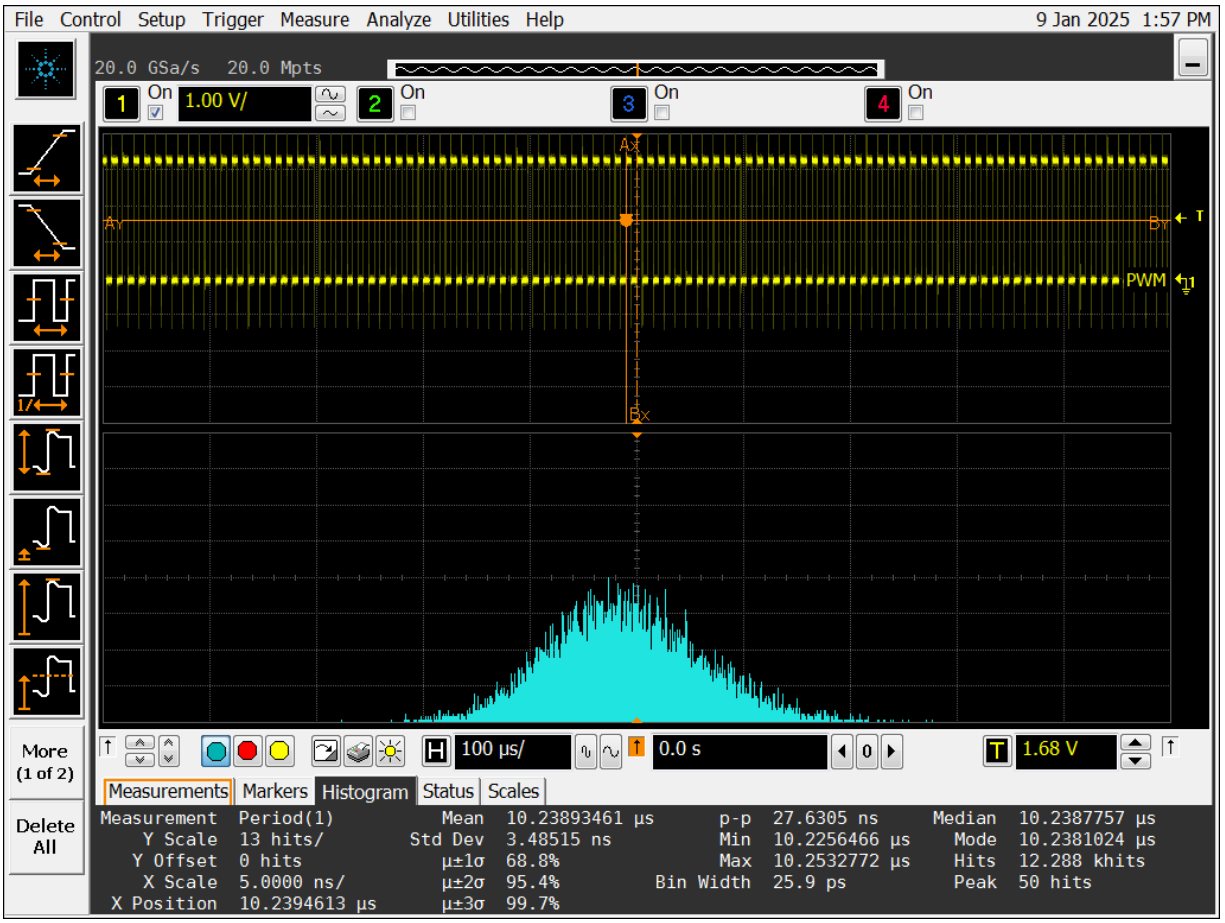


图 8-4. dsPIC33CK1024MP710 PWM 抖动结果（使用 FRC+PLL）



8.3. dsPIC33A

若要测试 dsPIC33A 器件上的抖动，需使用 dsPIC33AK128MC106 GP DIM 和 Curiosity 平台开发板。

示例代码中提供了多种时钟设置来测试抖动，其中包括 FRC、FRC+PLL、8 MHz MEMS 和 MEMS+PLL。PWM 信号分频器在所有情况下都进行相应的调整以输出 100 kHz，并且可以使用示波器测量输出。

```
#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; // 使能 PLL 发生器 2（若未使能）
    // 选择 FRC 作为 PLL2 的时钟源
    PLL2CONbits.NOSC = 1;
    // 请求 PLL2 时钟切换
    PLL2CONbits.OSWEN = 1;
    // 等待 PLL2 时钟切换完成
    while(PLL2CONbits.OSWEN);
```

```
    // 设置 PLL2 分频比以输出 200 MHz
    PLL2DIVbits.PLLPRE = 1; // 参考时钟输入将为 8 MHz，不分频
    PLL2DIVbits.PLLFBDIV = 200; // Fvco = 8 MHz * 200 = 1600 MHz
    PLL2DIVbits.POSTDIV1 = 4; // 将 Fcvo 4 分频
    PLL2DIVbits.POSTDIV2 = 1; // Fpllo = Fvco / 4 / 1 = 400 MHz

    // PLLSWEN 位用于控制切换至 PLL 反馈分频比。
    // 请求 PLL2 反馈分频比切换
    PLL2CONbits.PLLSWEN = 1;
    // 等待 PLL2 反馈分频比切换完成
    while(PLL2CONbits.PLLSWEN);

    // FOUTSWEN 位用于控制切换至 PLL 输出分频比。
    // 请求 PLL2 输出分频比切换
    PLL2CONbits.FOUTSWEN = 1;
    // 等待 PLL2 输出分频比切换完成
    while(PLL2CONbits.FOUTSWEN);

    // 使能 CLKGEN5
    CLK5CONbits.ON = 1;
    // 将 CLKGEN5 小数分频比复位为 1:1
    CLK5DIVbits.INTDIV = 0;
    CLK5DIVbits.FRACDIV = 0;
    // 请求 CLKGEN5 小数分频比切换
    CLK5CONbits.DIVSWEN = 1;
    // 等待 CLKGEN5 小数分频比切换完成
    while(CLK5CONbits.DIVSWEN);

    // 将 PLL2  Fout 设置为新的 CLKGEN5 时钟源
    CLK5CONbits.NOSC = 6;
    // 请求 CLKGEN5 时钟切换
    CLK5CONbits.OSWEN = 1;
    // 等待 CLKGEN5 时钟切换完成
    while (CLK5CONbits.OSWEN);

    // 选择 CLKGEN5 作为 PWM 主时钟源
    PCLKCONbits.MCLKSEL = 1;
}

void clock_PWM_at_400MHz_EC_PLL2_Fout() {
    _POSCMD = 0b00;
    PLL2CONbits.ON = 1; // 使能 PLL 发生器 2（若未使能）
    // 选择 FRC 作为 PLL2 的时钟源
    PLL2CONbits.NOSC = 3;
    // 请求 PLL2 时钟切换
    PLL2CONbits.OSWEN = 1;
    // 等待 PLL2 时钟切换完成
    while(PLL2CONbits.OSWEN);

    // 设置 PLL2 分频比以输出 200 MHz
    PLL2DIVbits.PLLPRE = 1; // 参考时钟输入将为 8 MHz，不分频
    PLL2DIVbits.PLLFBDIV = 200; // Fvco = 8 MHz * 200 = 1600 MHz
    PLL2DIVbits.POSTDIV1 = 4; // 将 Fcvo 4 分频
    PLL2DIVbits.POSTDIV2 = 1; // Fpllo = Fvco / 4 / 1 = 400 MHz

    //PLLSWEN 位用于控制切换至 PLL 反馈分频比。
    // 请求 PLL2 反馈分频比切换
    PLL2CONbits.PLLSWEN = 1;
    // 等待 PLL2 反馈分频比切换完成
    while(PLL2CONbits.PLLSWEN);

    // FOUTSWEN 位用于控制切换至 PLL 输出分频比。
    // 请求 PLL2 输出分频比切换
    PLL2CONbits.FOUTSWEN = 1;
    // 等待 PLL2 输出分频比切换完成
    while(PLL2CONbits.FOUTSWEN);

    // 使能 CLKGEN5
    CLK5CONbits.ON = 1;
    // 将 CLKGEN5 小数分频比复位为 1:1
    CLK5DIVbits.INTDIV = 0;
    CLK5DIVbits.FRACDIV = 0;
    // 请求 CLKGEN5 小数分频比切换
```

```
    CLK5CONbits.DIVSWEN = 1;
    // 等待 CLKGEN5 小数分频比切换完成
    while(CLK5CONbits.DIVSWEN);

    // 将 PLL2 Fout 设置为新的 CLKGEN5 时钟源
    CLK5CONbits.NOSC = 6;
    // 请求 CLKGEN5 时钟切换
    CLK5CONbits.OSWEN = 1;
    // 等待 CLKGEN5 时钟切换完成
    while (CLK5CONbits.OSWEN);

    // 选择 CLKGEN5 作为 PWM 主时钟源
    PCLKCONbits.MCLKSEL = 1;
}

void clock_PWM_at_FRC(){
    // 使能 CLKGEN5
    CLK5CONbits.ON = 1;
    // 将 CLKGEN5 小数分频比复位为 1:1
    CLK5DIVbits.INTDIV = 0;
    CLK5DIVbits.FRACDIV = 0;
    // 请求 CLKGEN5 小数分频比切换
    CLK5CONbits.DIVSWEN = 1;
    // 等待 CLKGEN5 小数分频比切换完成
    while(CLK5CONbits.DIVSWEN);
    // 将 FRC 设置为 PWM 时钟源
    CLK5CONbits.NOSC = 1;
    // 请求 CLKGEN5 时钟切换
    CLK5CONbits.OSWEN = 1;
    // 等待 CLKGEN5 时钟切换完成
    while (CLK5CONbits.OSWEN);
    // 选择 CLKGEN5 作为 PWM 主时钟源
    PCLKCONbits.MCLKSEL = 1;
}

void clock_PWM_at_MEMS(){
    _POSCMD = 0b00;
        // 使能 CLKGEN5
    CLK5CONbits.ON = 1;
    // 将 CLKGEN5 小数分频比复位为 1:1
    CLK5DIVbits.INTDIV = 0;
    CLK5DIVbits.FRACDIV = 0;
    // 请求 CLKGEN5 小数分频比切换
    CLK5CONbits.DIVSWEN = 1;
    // 等待 CLKGEN5 小数分频比切换完成
    while(CLK5CONbits.DIVSWEN);
    // 将 FRC 设置为 PWM 时钟源
    CLK5CONbits.NOSC = 3;
    // 请求 CLKGEN5 时钟切换
    CLK5CONbits.OSWEN = 1;
    // 等待 CLKGEN5 时钟切换完成
    while (CLK5CONbits.OSWEN);
    // 选择 CLKGEN5 作为 PWM 主时钟源
    PCLKCONbits.MCLKSEL = 1;
}

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


    PG1CONbits.UPDMOD = 0b000;  // 如果 UPDREQ = 1，则在下一个 PWM 周期开始时进入 PWM 缓冲区更新模式
    PG1CONbits.TRGMOD = 0b01;  // PWM 发生器 1 处于单触发模式
    PG1CONbits.SOCS = 0b0000;    // 周期的开头为本地 EOC

    PG1CONbits.ON = 0;           // 禁止 PWM 发生器 1（尚未启动）
    PG1CONbits.TRGCNT = 1;       // PWM 发生器 1 被触发时会产生 1 个 PWM 周期
    PG1CONbits.CLKSEL = 0b01;    // PWM 发生器 1 使用 PWM 主时钟，既不分频也不调整
    PG1CONbits.MODSEL = 0b000;  // PWM 发生器 1 处于独立边沿 PWM 模式

    PG1IOCONbits.PMOD = 0b01; // PWM 发生器 1 输出模式为独立模式
    PG1IOCONbits.PENH = 1;     // PWM 发生器 1 控制 PWM1H 输出引脚
    PG1IOCONbits.PENL = 1;     // PWM 发生器 1 控制 PWM1L 输出引脚
```

```
    PG1CONbits.ON = 1;
}

int main(void) {

#ifdef use_FRC
    clock_PWM_at_FRC();
    PG1PER = 0x500;// 100 kHz, 使用 FRC
    PG1DC = 0x250;// 100 kHz, 使用 FRC
#endif

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

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

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

    pwm_Init();

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

8.3.1. dsPIC33A 结果

下面分别给出了使用 8 MHz FRC 或 400 MHz FRC+PLL 作为时钟时 PWM 输出的抖动测试结果。FRC+PLL 测试结果呈现出了更明确的中心点和更低的标准偏差（1.2 ns，而 FRC 测试结果为 1.7 ns）。

图 8-5. dsPIC33AK128MC106 PWM 抖动结果（使用 FRC）

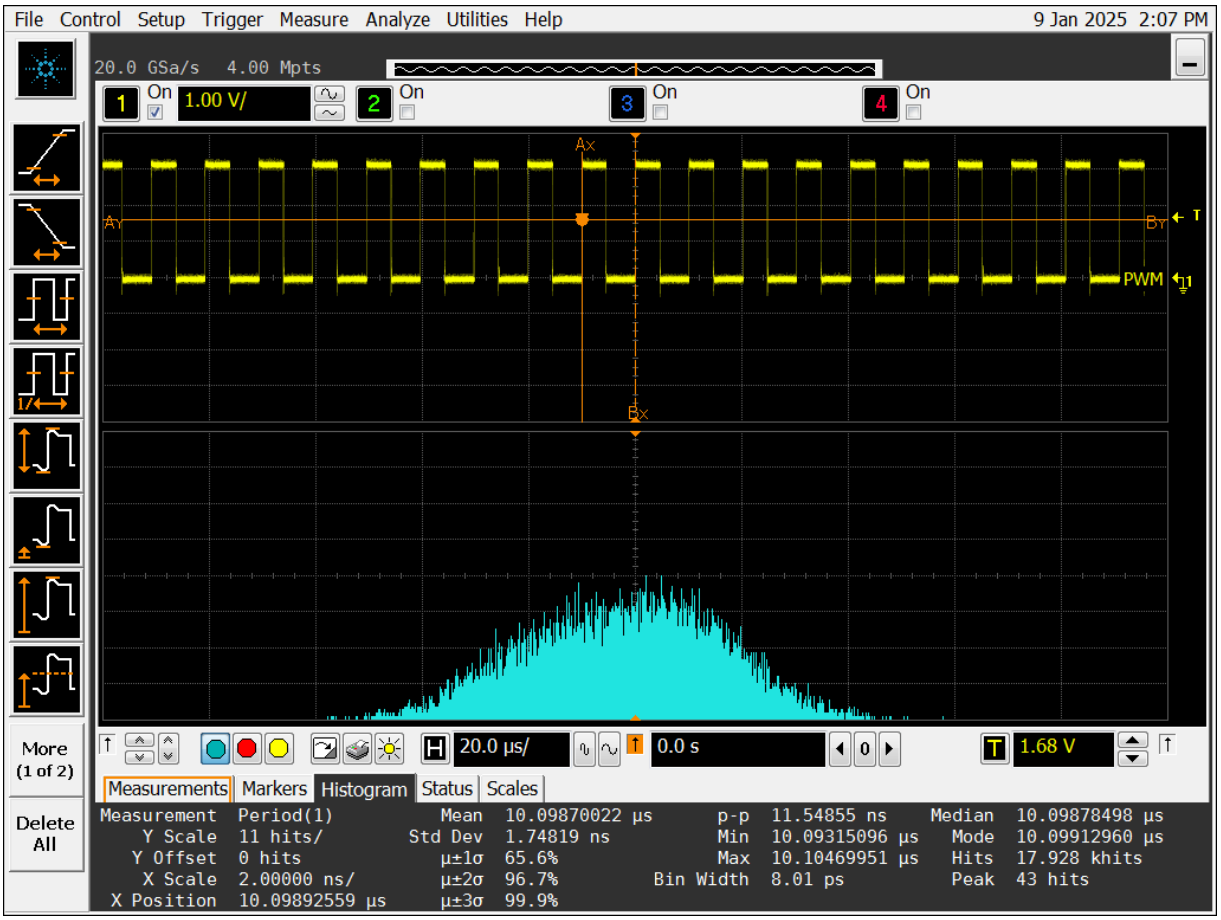


图 8-6. dsPIC33AK128MC106 PWM 抖动结果（使用 FRC+PLL）

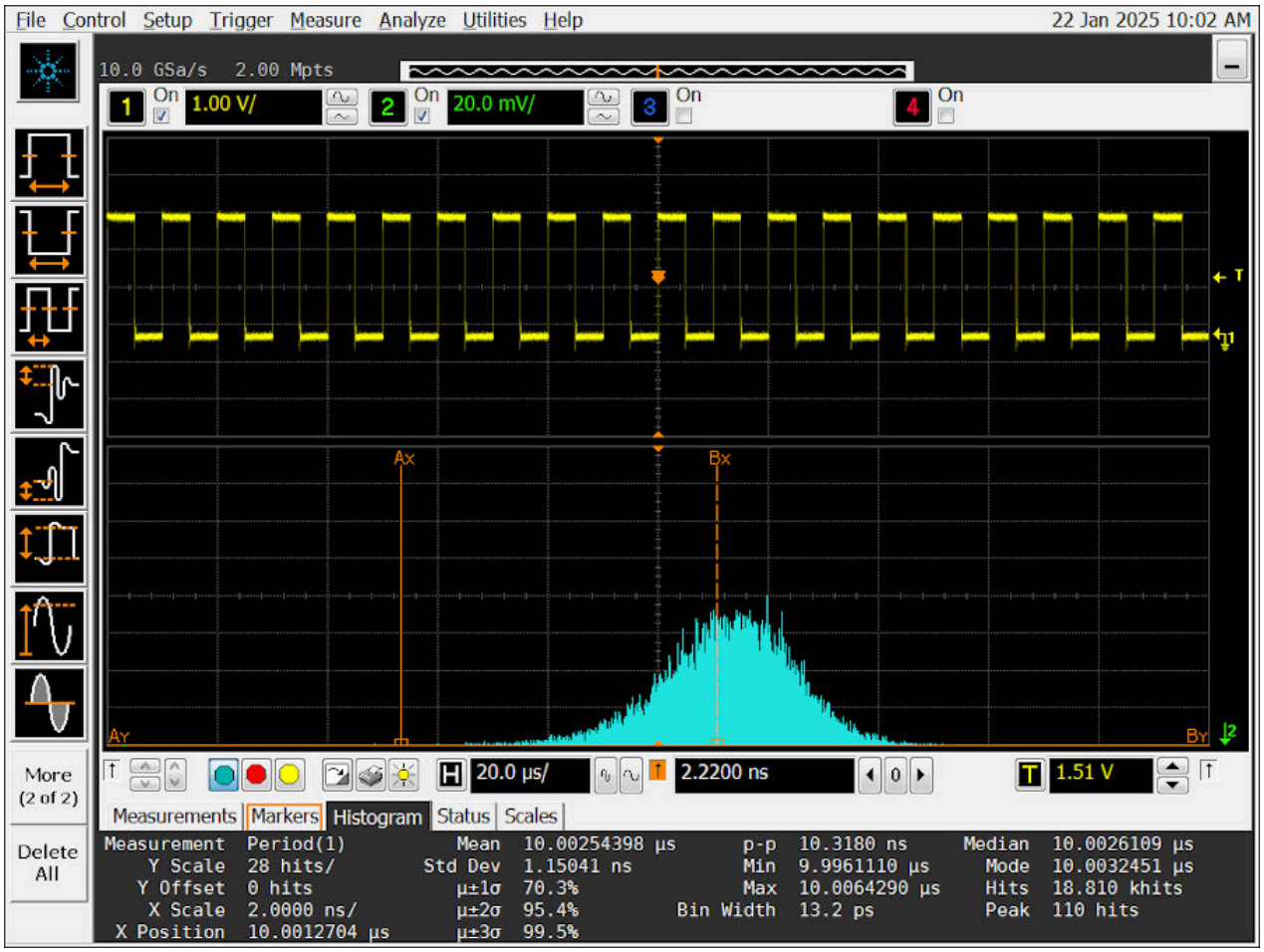


图 8-7. dsPIC33AK128MC106 PWM 抖动结果（使用 8 MHz MEMS）

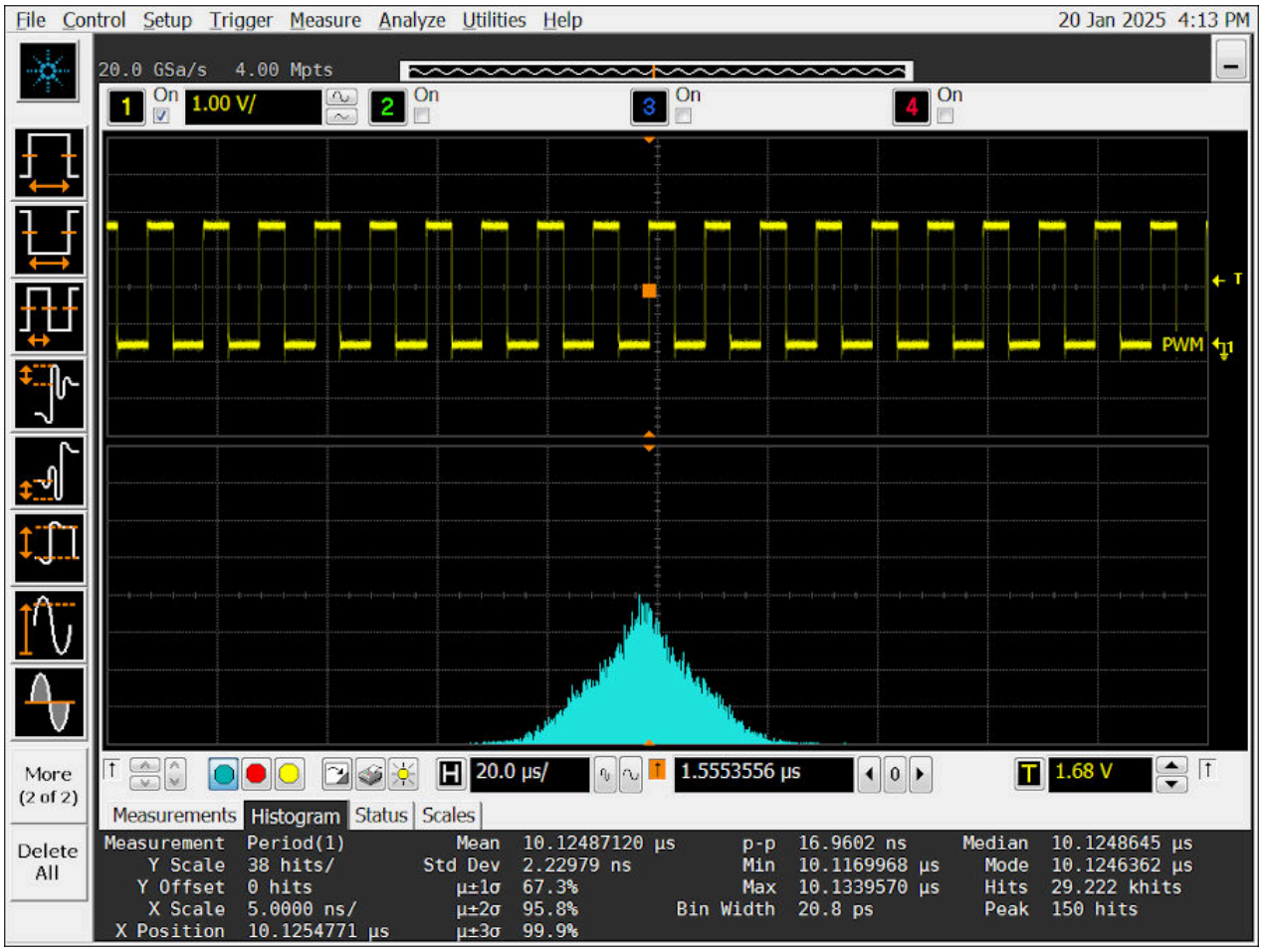
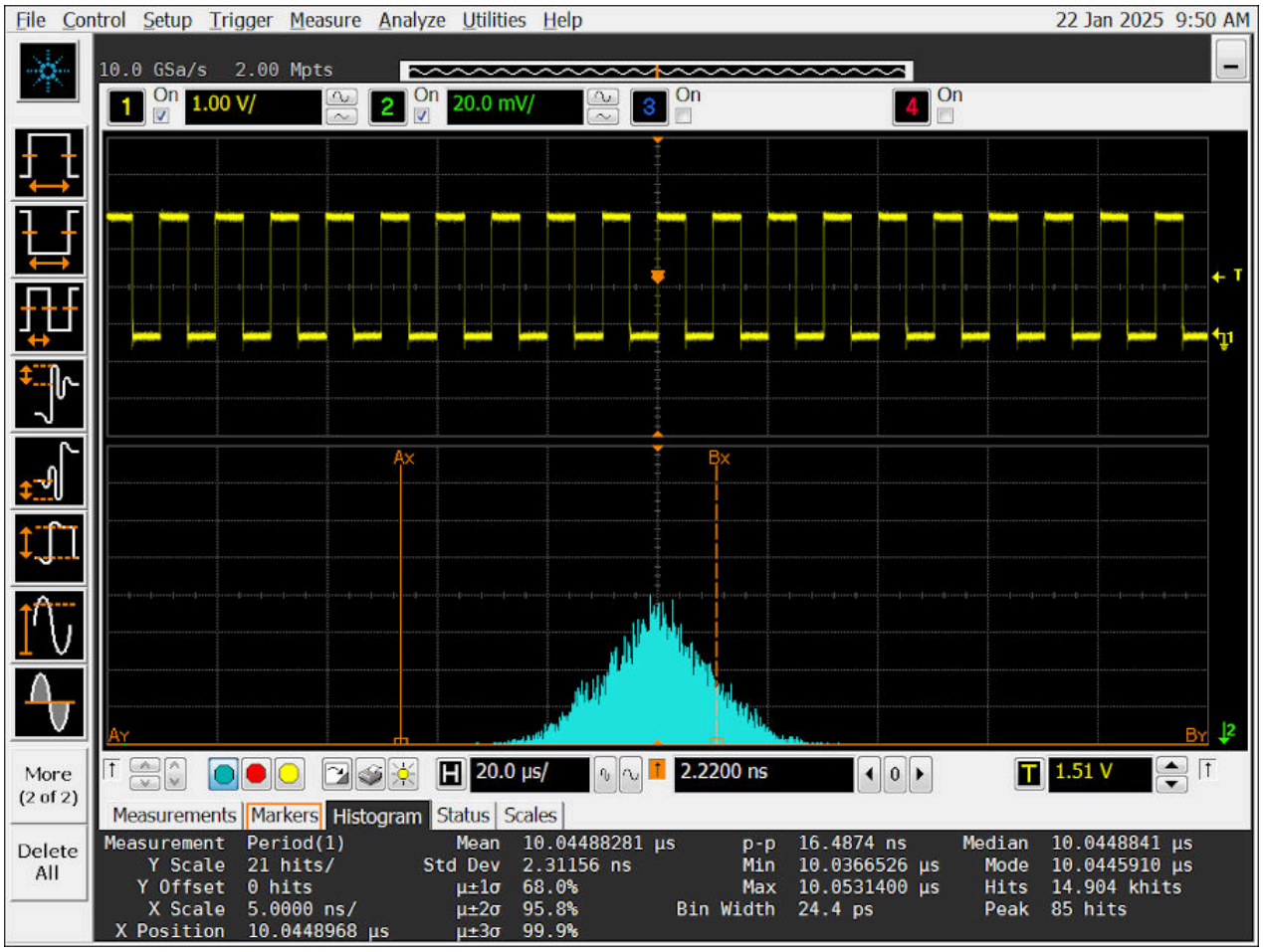


图 8-8. dsPIC33AK128MC106 PWM 抖动结果（使用 MEMS + PLL）



9. 结论

本文档就应用中的抖动设定了预期，并介绍了如何了解、测量和尽可能减少抖动。尽管任何系统中都会存在随机抖动，但可以使用本文档中提供的工具来了解并管理确定性抖动。这些工具有助于改进应用，从而能从器件和周边系统获得更好的结果。

10. 版本历史

版本历史部分汇总了文档中所做的更改，并按照出版物版本从新到旧的顺序列出。

版本	日期	说明
A	2025 年 3 月	初始版本

Microchip 信息

商标

“Microchip”名称和徽标、“M”徽标及其他名称、徽标和品牌均为 Microchip Technology Incorporated 或其关联公司和/或子公司在美国和/或其他国家/地区的注册商标和未注册商标（“Microchip 商标”）。有关 Microchip 商标的信息，可访问 https://www.microchip.com/en-us/about/legal-information/microchip-trademarks。

ISBN： 979-8-3371-1013-4

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物及其提供的信息仅适用于 Microchip 产品，包括设计、测试以及将 Microchip 产品集成到您的应用中。以其他任何方式使用这些信息都将被视为违反条款。本出版物中的器件应用信息仅为您提供便利，将来可能会发生更新。您须自行确保应用符合您的规范。如需额外的支持，请联系当地的 Microchip 销售办事处，或访问 www.microchip.com/en-us/support/design-help/client-support-services。

Microchip“按原样”提供这些信息。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或间接的损失、损害或任何类型的开销，Microchip 概不承担任何责任，即使 Microchip 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，Microchip 在任何情况下所承担的全部责任均不超出您为获得这些信息向 Microchip 直接支付的金额（如有）。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。除非另外声明，在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

Microchip 器件代码保护功能

请注意以下有关 Microchip 产品代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用且符合工作规范的情况下，Microchip 系列产品非常安全。
- Microchip 注重并积极保护其知识产权。严禁任何试图破坏 Microchip 产品代码保护功能的行为，这种行为可能会违反《数字千年版权法案》（Digital Millennium Copyright Act）。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。