注意 : この日本語版文書は参考資料としてご利用ください。
最新情報は必ずオリジナルの英語版をご参照願います。

アプリケーションでのジッタを理解して対処する方法

AN5823

概要

本書はクロックジッタに関する知識を提供すると共に、制御ループ内でPWM信号を使うアプリケーションに対するクロックジッタの影響について解説します。また、ジッタ発生源の識別方法、ジッタの計測方法、ジッタに配慮したアプリケーションの設計方法も記載しています。

目次

1. はじめに

1.1 ジッタとは

ジッタは基準信号からのタイミングの逸脱として定義されます。ジッタは電子回路のタイミングまたはしきい値の微小偏差よって生じます。これらの偏差は環境またはシステム要因(電源ノイズ、信号の不完全性等)によって増加します。本書ではジッタのタイプと成分およびジッタの定量化について説明します。システム内のノイズが増加するにつれてジッタが大きくなり、アプリケーションに対して有害となる可能性があります。

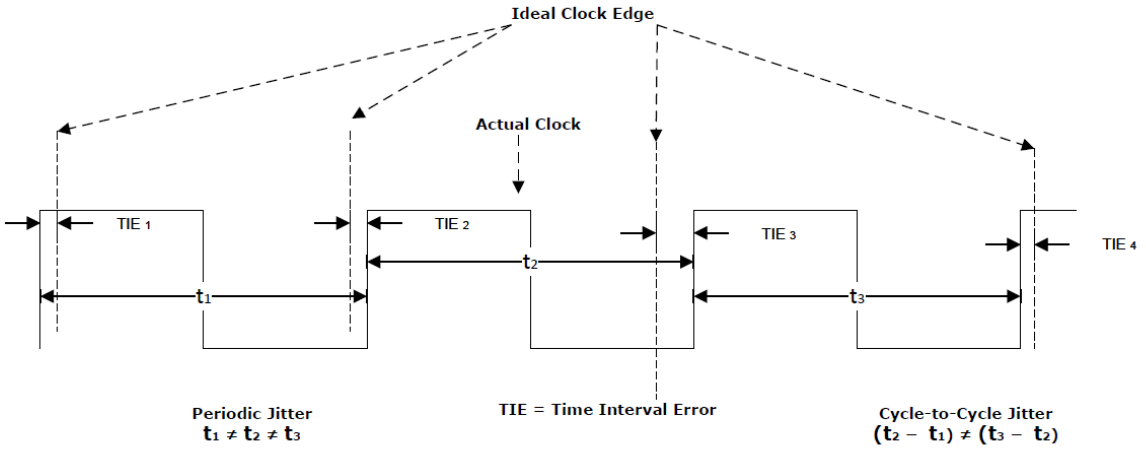
1.2 ジッタに関して予期される事

ジッタは軽減できても完全に除去する事はできないと理解する事が重要です。製品によっては、エッジ分解能がシステムジッタよりも小さい場合があります。この場合、アプリケーションのタイプによっては問題が生じる可能性があります。制御ループで平均化等のフィルタを使う事により、ジッタに起因する誤差の影響を受けにくくできます。サイクルごとに制御を行うシステムはジッタを許容できますが、ヒストグラムを取ると不均一に分布したリップルが生じる可能性があります。分解能またはジッタの仕様値を決定する一部の試験は理想的な環境で行われますが、実際の環境では電源スイッチングから過渡状態が生じてノイズフロアが上昇し、結果としてシステムジッタが生じます。

2. ジッタの原理

周期ジッタ、Cycle-to-Cycleジッタ、長期/長周期ジッタ等、タイプの異なるジッタは異なる方法で観察および計測できます。これらの計測方法は全て関連しており、どのジッタ成分が存在し、それらの成分がどのようにアプリケーションに影響を及ぼすのか、といったジッタに関する理解をさらに深めるために役立ちます。図2-1に、クロック源ジッタの測定指標を示します。

図2-1. クロック源ジッタの測定指標

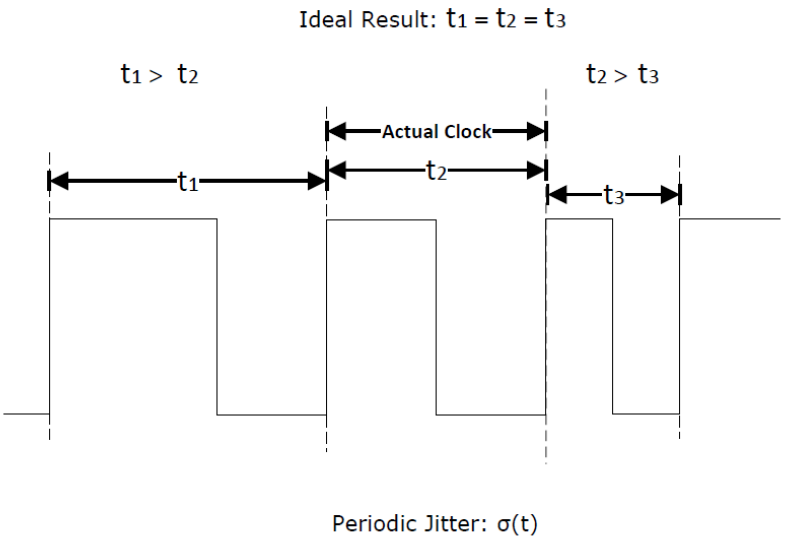


2.1 周期ジッタ

周期ジッタは、単一クロック周期が期待されるクロック周期と一致しない場合に発生します。ジッタがタイミング信号に影響を及ぼすと信号エッジが進み側または遅れ側にシフトするため、周期は理想よりも短くなったり長くなったりします。ジッタにより周期はサイクルごとに変化します。図2-2では、t1は期待されるパルスより長く、t2は正確に期待通りのクロック周期を持ち、t3は期待されるパルスよりも短くなっています。理想的なアプリケーションでは、これらのパルスは全て同じ長さになります。

図2-2に周期ジッタを示します。

図2-2. 周期ジッタ



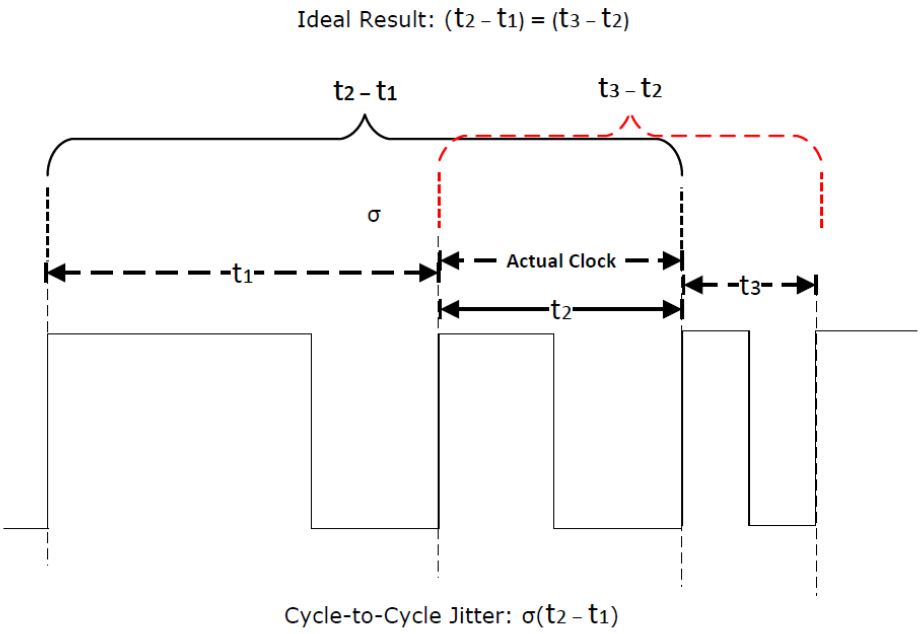
2.2 Cycle-to-Cycleジッタ

図2-3に示す通り、Cycle-to-Cycleジッタは隣接する2つのクロックサイクル間の周期差として、式2-1により表されます。

式2-1.

Cycle-to-Cycle ジッタ = σ (t2 − t1)

図2-3. Cycle-to-Cycleジッタ

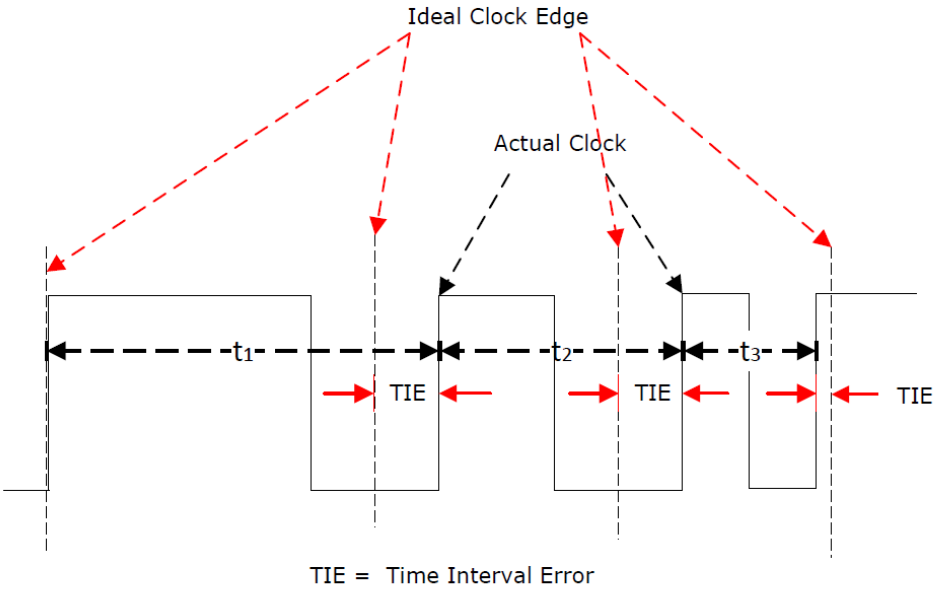


2.3 タイム インターバル エラー(TIE)

図2-4に示す通り、実際のクロックエッジと期待される(理想的な)クロックエッジのタイミング差をタイム インターバル エラー(TIE)と呼びます。この図では、3つのクロックパルス エッジの全てが期待されるタイミングで発生していません。これは、t1の周期が期待される周期よりも長いために後続の波形全体がシフトするためです。TIEは正または負の値を取ります。この図では、t1とt2の間およびt2とt3の間のTIEはどちらも正の値ですが、t3は理想タイミングよりも早く終了するためTIEは負の値です。このようにTIEは平均化によって相殺される場合があります。TIEはクロックと異なるタイムスケールを持つ場合もあります。TIEのタイムスケールがクロックよりも低速である場合、小さな誤差を持つ多数のサイクルによって長期的に巨大な誤差が正または負方向に累積される可能性があります。エンベロープ タイミング(タイミング エラー自体の長期/長周期的な変動: ワンダーとも言われる)とTIEの大きさは無関係であり、それらは異なる方法でシステム性能に影響を及ぼします。ランダムジッタでもこれらのシフトは生じますが、対処は可能です。

図2-4に、タイム インターバル エラーを示します。

図2-4. タイム インターバル エラー(TIE)



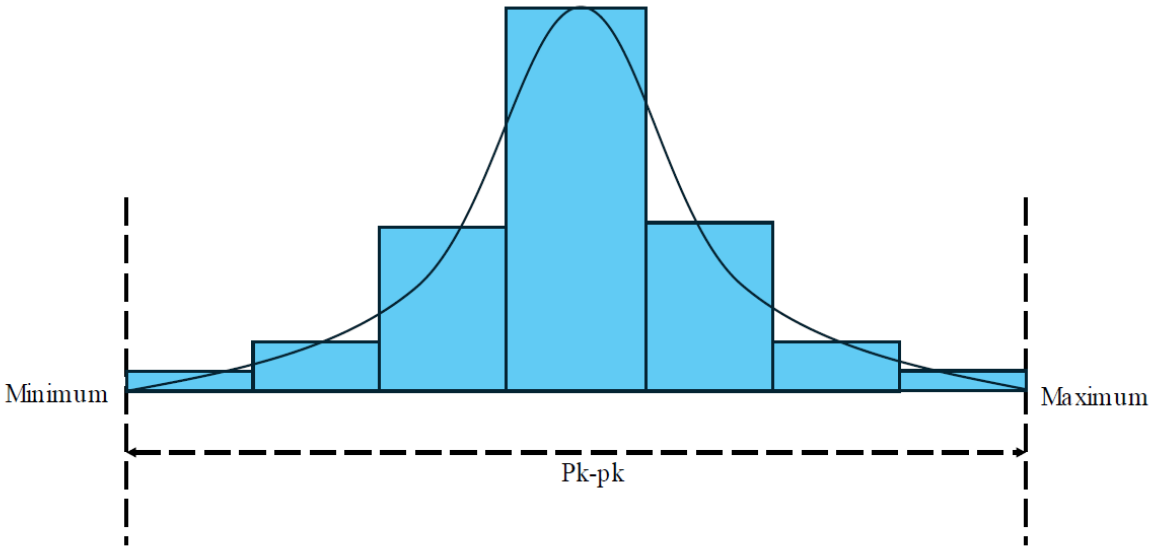
3. ジッタの測定指標

クロック タイミング ジッタを評価するために各種の測定指標が使われます。多くの場合、ジッタの計測データはヒストグラムとして示されます。ヒストグラムのx軸は理想クロック周期からのタイミング差を表し、y軸は各周期値が計測された回数を表します。

3.1 最小値/最大値

ヒストグラムのx軸の最小値は計測された最小の周期であり、最大値は計測された最大の周期です。ピークツーピーク値はヒストグラムの最小値と最大値の差として求まります。この値はアプリケーション内のジッタの全分布を考慮する場合に使われますが、ヒストグラムのサンプル数が増加するにつれてピークツーピーク値は理論的に限界が無く(クロック停止、喪失等)、ジッタの評価における主要指標とはなりません。一方、ヒストグラムの平均値と標準偏差は、ジッタ解析においてより正確な指標となります。図3-1に、ヒストグラムの最小値と最大値を示します。

図3-1. ヒストグラムの最小値と最大値

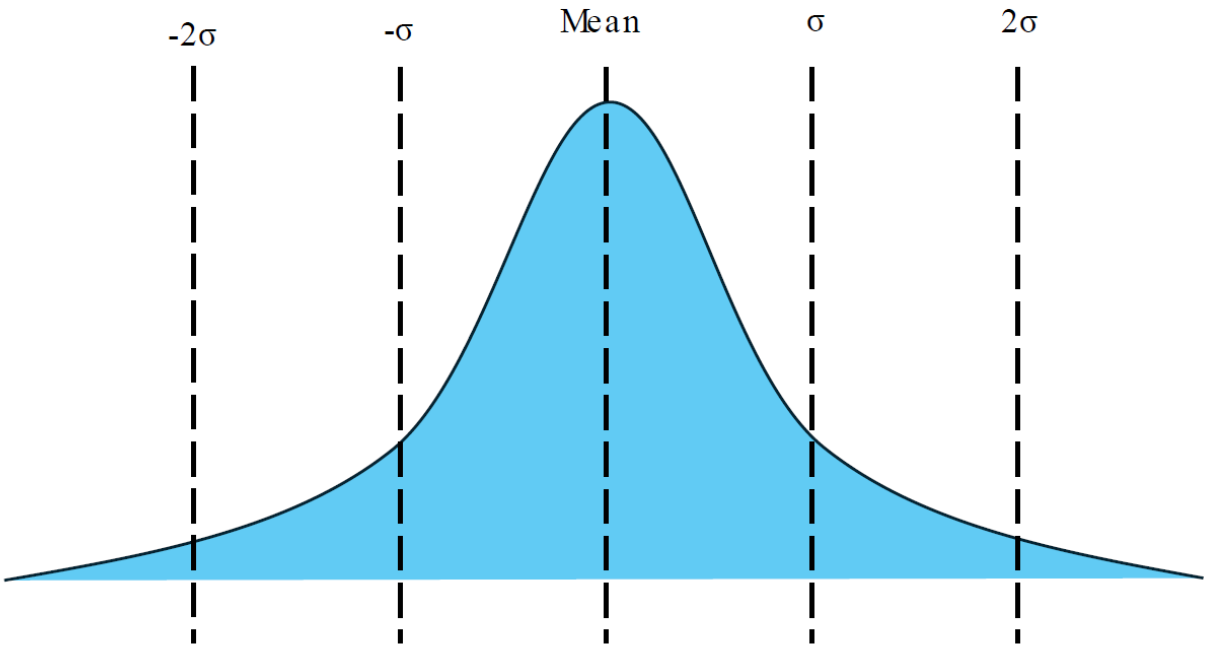


3.2 標準偏差 (σ)

標準偏差は、一連のデータ値の変動またはばらつきの大きさを定量化する統計的指標です。多くのデータが平均値の近くにまとまって分布していると標準偏差は小さくなり、データが大きくばらついていると標準偏差は大きくなります。正規分布(ガウス分布)では、データの約68%が-1σ と+1σの間に入ります。標準偏差はデータの平均値から算出されます。平均値と標準偏差はジッタ解析における最重要指標です。

図3-2に、標準偏差と平均値を示します。

図3-2. 標準偏差と平均値

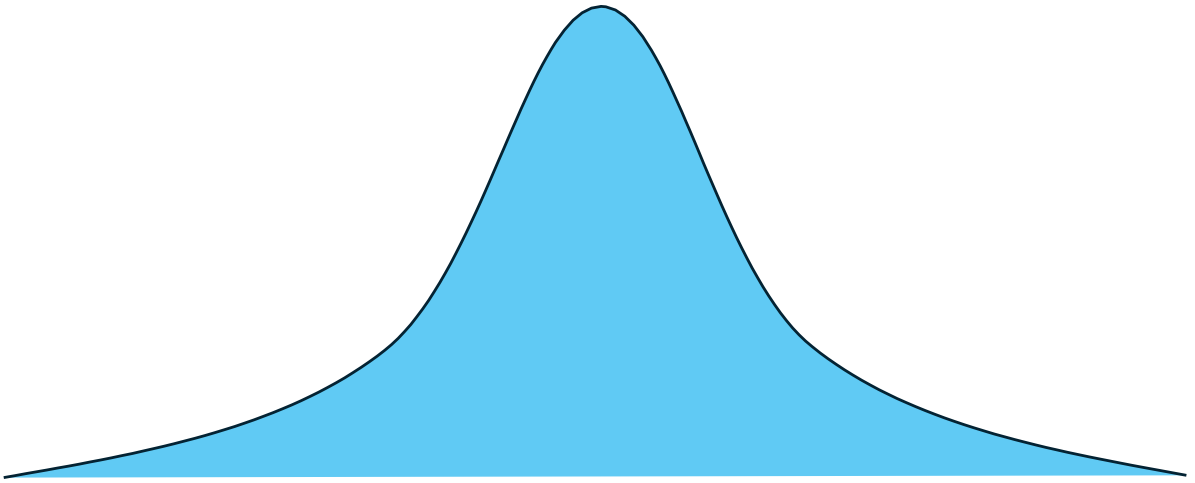


4. ジッタのタイプ

4.1 ランダムジッタ

ジッタはランダムジッタと確定的ジッタの2種類に分類されます。多くの場合、ジッタはノイズの多い電源、不適切なハードウェア設計、熱雑音、クロストークによって生じます。全てのシステムにおいて、ジッタは信号周期の偏差として現れます。理想的な状態ではランダムジッタのみが存在し、クロック周期のヒストグラムはガウス分布(または釣鐘型曲線)を示します(図4-1参照)。

図4-1. ランダムジッタ



計測結果がガウス分布を示す場合、ジッタは「ランダム」であり、その原因を特定または除去する事はできません。ランダムジッタは無限界(ガウス分布の裾が無限に拡がる事を意味する)ですが、大きな偏差の頻度は急激に減少します。

4.2 確定的ジッタ

システムにランダムジッタと確定的ジッタの両方が存在する場合、分布曲線のピークがどちらかの方向へシフトし、複数のピークが存在する場合すらあります。

図4-2. 確定的ジッタ - ピークがシフトしたケース

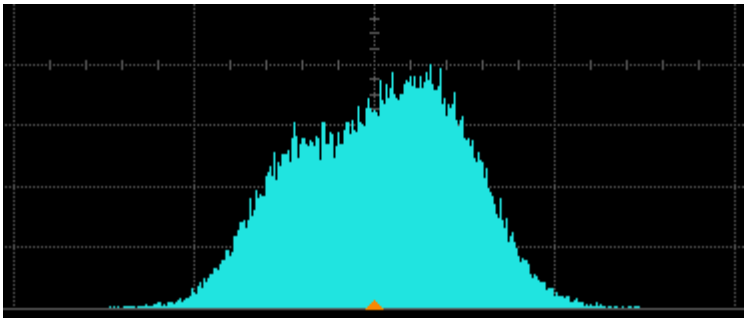
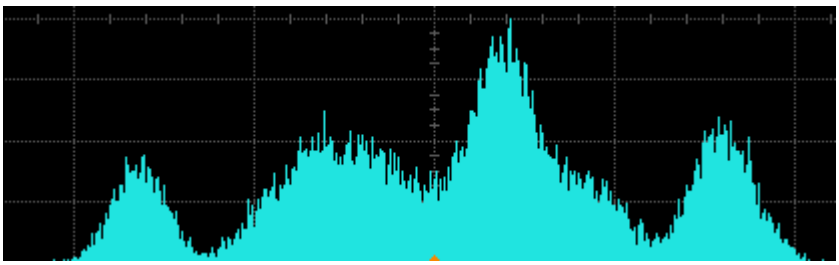
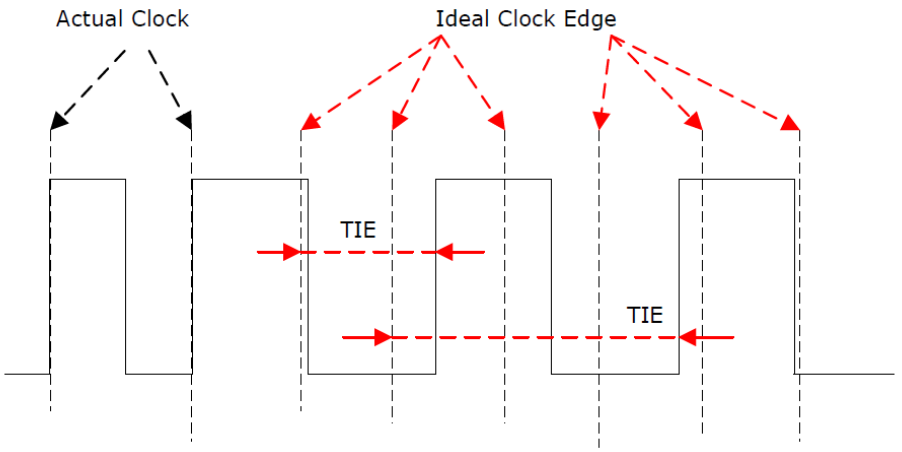


図4-3. 確定的ジッタ - ピークが複数存在するケース



これらの図は、アプリケーションで確定的ジッタが観測される場合の代表例です。単一ピークを持つガウス分布とは異なり、複数の分布曲線が組み合わさったように見えます。このような曲線は、アプリケーション内でサイクル時間に繰り返し影響を及ぼす確定的な要因の存在を示しています。図 4-2では、曲線のピークがヒストグラムの右方へシフトしており、平均クロック周期が期待されるクロック周期より大きい事を示しています。これは、クロックサイクルの周期が時間の経過とともに単調に増加した場合に見られる長期ジッタの一例です。このような場合、クロックサイクルの周期は期待される周期よりも一貫して長くなるため、TIEヒストグラム ピークは非常に大きくなる可能性があります。図4-4に、確定的ジッタが長期間のTIEに影響を及ぼす様子を示します。2つの連続したクロックパルスの周期が期待値よりも長いためTIEが増加している事に注意してください。

図4-4. 長期ジッタ



5. 計測技術

計測技術と計測条件は計測結果に大きな影響を及ぼします。使用する計測器には、信号速度とエッジレートに見合った性能が必要です。計測誤差を最小化するため、プローブのグランド線の長さに配慮する必要があります。一部の計測器はクロックまたはジッタ解析ツール(ソフトウェア)を備えています。システムジッタ要因を特定および定量化するために複数回の試験が必要になる場合があります。ジッタの発生源を特定するには、回路パスに沿って計測するのが効果的です。代表的な計測点として以下が挙げられます。

- クロック源(マイクロコントローラ外部のクロック源を使用する場合)
- マイクロコントローラ内部クロック源の出力ピン
- PLL出力ピン
- PWM出力ピン
- ゲートドライバ出力

複数のノイズ源(マイクロコントローラとその他の回路)からシステムへノイズが追加される可能性がある場合、それらのノイズ源を分離して分離前の計測結果と比較するのが効果的です。例えば、通信ラインまたはその他のPWM信号を停止する事で、興味がある信号に対するそれらの影響を評価する事ができます。

5.1 計測手順

ジッタの計測手順は以下の通りです。

1. 被試験デバイス(DUT)を準備します(電源、設定パラメータを含む)。
2. 参照出力として割り当てられたピンにオシロスコープ プローブを接続します。オシロスコープを平均化モードで使ってはいけません(平均化によって問題が覆い隠されてしまう可能性があるため)。
3. 計測するジッタのタイプに応じてオシロスコープを設定します。連続する周期を記録する事によりCycle-to-Cycleジッタを計算できる他、周期ジッタとTIEも計測できます。
4. ステップ3を10,000回以上繰り返します。サンプル数を増やす事により、どのような長期/長周期ジッタも計測できます。
5. 試験結果からヒストグラムを生成し、平均値と標準偏差を計算します。ヒストグラムは、計測された各周期の発生回数を示します。
6. このデータにより、確定的ジッタが存在するかどうかと、ジッタのレベルがアプリケーションに対して許容可能かどうかを判断します。
7. 確定的ジッタが観測された場合(例: 複数のピークがヒストグラムに現れた場合)、スペクトル アナライザを使って追加ピークを生じさせている周波数を特定できます。
8. 確定的ジッタの軽減法は次章で説明します。スペクトル アナライザを使って必要なタイミング信号とは異なる追加の周波数レンジが見つかった場合、次章の手順に従う事で効果的に問題を解消できます。

6. ジッタの軽減法

システム内で問題となりそうなジッタの成分とタイプが判明したら、システム内の確定的ジッタを低減する方法を検討します。ランダム成分は除去できません。前章ではジッタの原因について説明しました。ジッタの低減法として以下が挙げられます。

- クロック入力ソースの選択
- 位相ロックループ(PLL)のフィルタ処理
- 電源ノイズの低減
- プリント基板(PCB)とシステム設計の最適化

6.1 クロック源ジッタ

外部オシレータが不要な場合でも、内部クロック源の代わりに高速なMEMSまたは水晶振動子回路を参照クロックとして使う事により、デバイスジッタを低減できる事が知られています。より高速な入力クロックをdsPIC® PLLと一緒に使う事により、クロック精度を上げる事ができます。なぜなら、VCO周波数補正は数クロックサイクルごとに発生しますが、既定値の8 MHz FRCよりも高速な外部クロックを使う事により、VCO補正をより頻繁に発生させる事ができるからです。システムを試験する事により、アプリケーションにとってクロック精度の改善が必要かどうか(またはシステム内のジッタのレベルが許容可能かどうか)を判定する事が重要です。

6.2 位相ロックループ(PLL)

PWMアプリケーションでは、必要な速度と分解能を達成するためにPLLが一般的に使われます。ジッタ性能はPLLの設計とフィルタ処理に応じて向上もすれば低下もします。PLLの性能は、その動作パラメータ(分周比設定、VCO周波数等)によって決まります。PLLの前置または後置分周器を設定してVCO周波数を最適化する事で、確定的ジッタを効果的に低減できる可能性があります。

6.3 電源

複数の異なる電源を使ってジッタのヒストグラムを生成する事により、システム内のジッタの原因を特定できる可能性があります。例えば、内蔵電源と実験用外部電源を使って結果を比較します。PCB上の内蔵電源の使用時にジッタが非常に大きくなる場合、電源回路の調整が必要かもしれません。

6.4 回路の調整

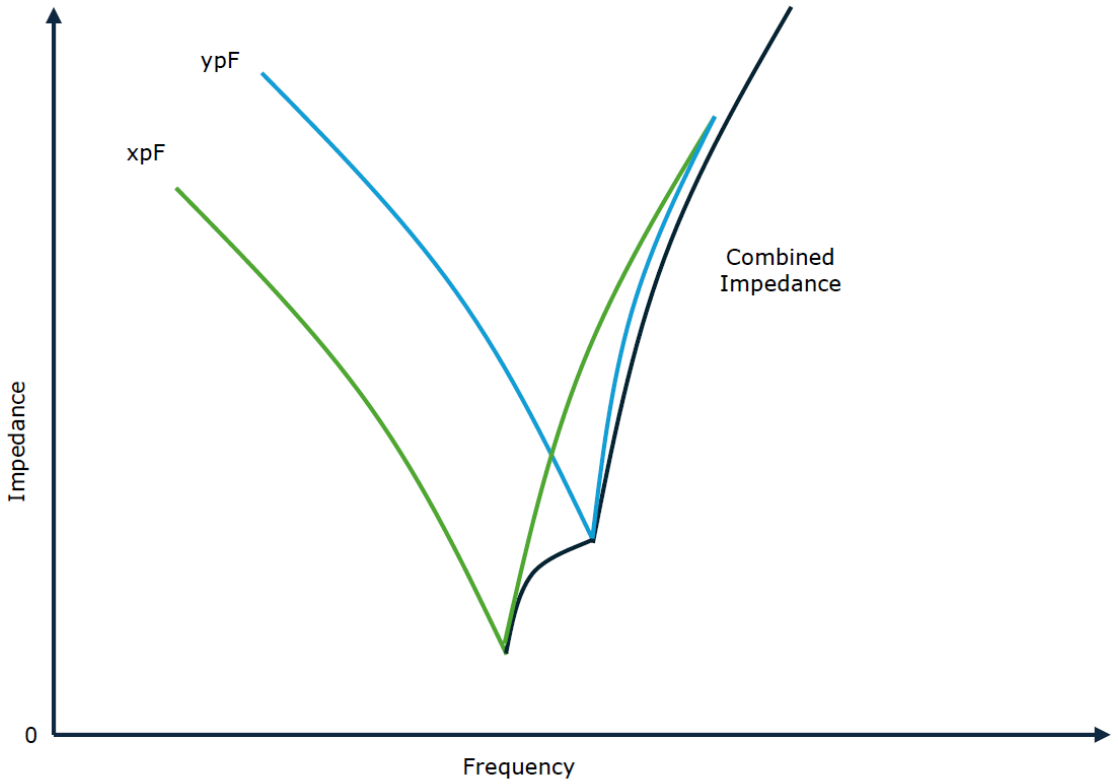
電源トレースを設計する際は、それらの長さに注意する必要があります。トレースが長いほどノイズの影響を受けやすくなります。これに対する最も効果的な対策は、4層以上のPCBを使用する(VDDとVSSに別々の層を使用する)事です。ベタ層から端子まで最短で配線できるため、誘導ノイズの影響を受け難くなります。さらにトレース長を制限するため、部品を互いに近付けて配置します。これは特にDUT(被試験デバイス)に隣接するバイパス コンデンサで重要となります。加えて、デバイスから2つの高速信号を取り出す場合、隣接するピンを使ってはいけません(ただし、それらがペアで意味を持つ場合を除く)。例えば、PWM信号ピンに隣接するピンからI2C信号を取り出すと、2本のピンを横断してノイズが導入され、ジッタの増加とPWM分解能の低下を招きます。I2CとPWMが異なるクロック源を使う場合、これらの信号は互いに非同期となるため、同期系よりエラー解析難度が上がります。

6.5 部品

部品の選定はシステムノイズに大きな影響を及ぼします。バイパス コンデンサの値は、目標の動作周波数でインピーダンスが最低となるよう選定します。すなわち、ピンのトグル時に電源に対して最適なフィルタとして動作するようにコンデンサを選ぶ必要があります。これらのバイパス コンデンサを追加すると、デバイスのデータシートに記載されている推奨コンデンサ レイアウトに適合しなくなる可能性があり、PCB上のスペースも使います。従って、コンデンサのデータシートを参照し、アプリケーション周波数に応じた必要最小限の値を選定します。

図6-1に、これらのデータシートに記載される一般的なグラフを示します。

図6-1. 周波数に対するインピーダンス特性



6.6 その他の要因

温度を含むその他の要因もデバイス内のジッタ計測値に影響を及ぼす可能性があります。以下の例では、Curiosityプラットフォーム ボード上でdsPIC33AK128MC106 GP DIMを100 kHzのPWM出力で使います。図6-2と図6-3が示す通り、温度の上昇はPWMの出力に大きく影響を及ぼします。温度が上昇すると確定的ジッタが増加するため、周期がより不規則になります。

図6-2. 室温でのPWM 100 kHz

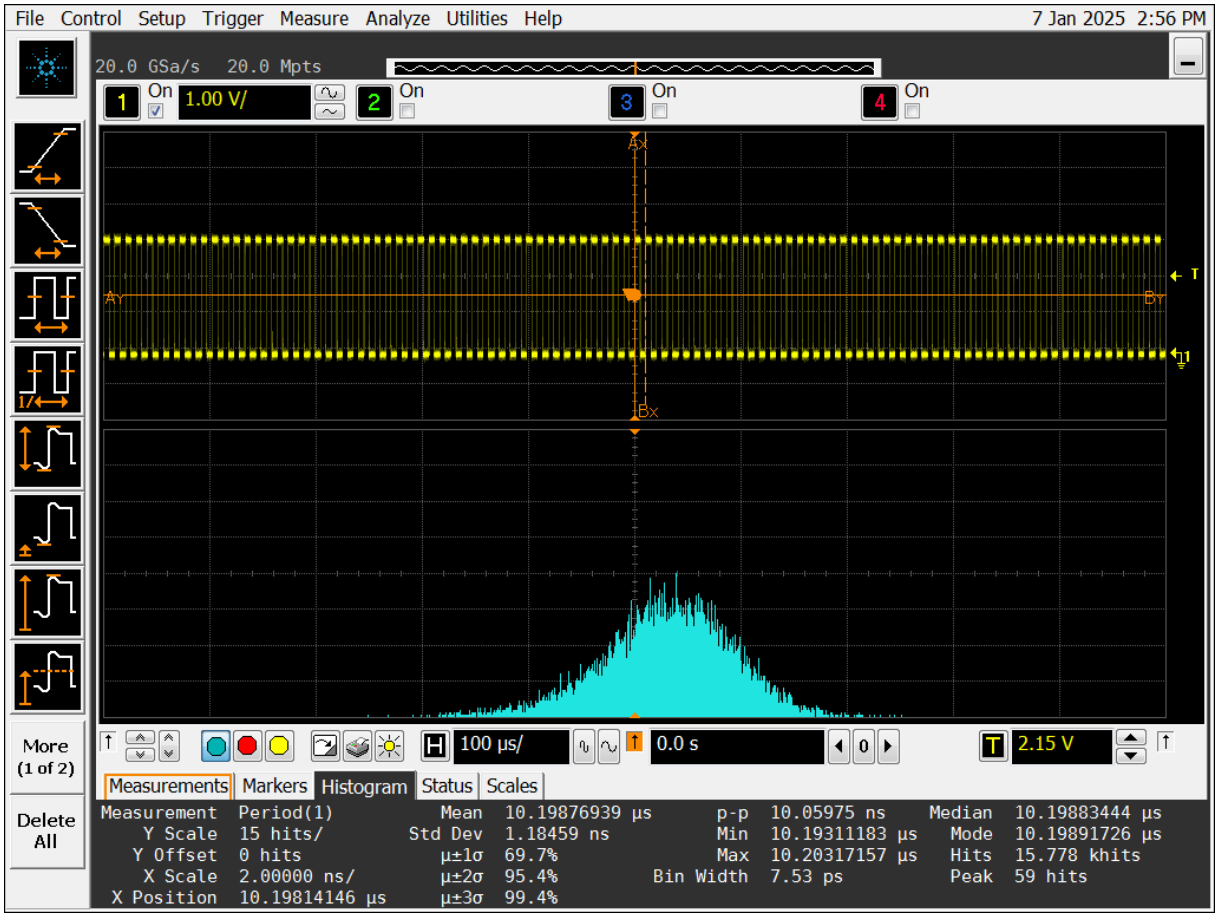
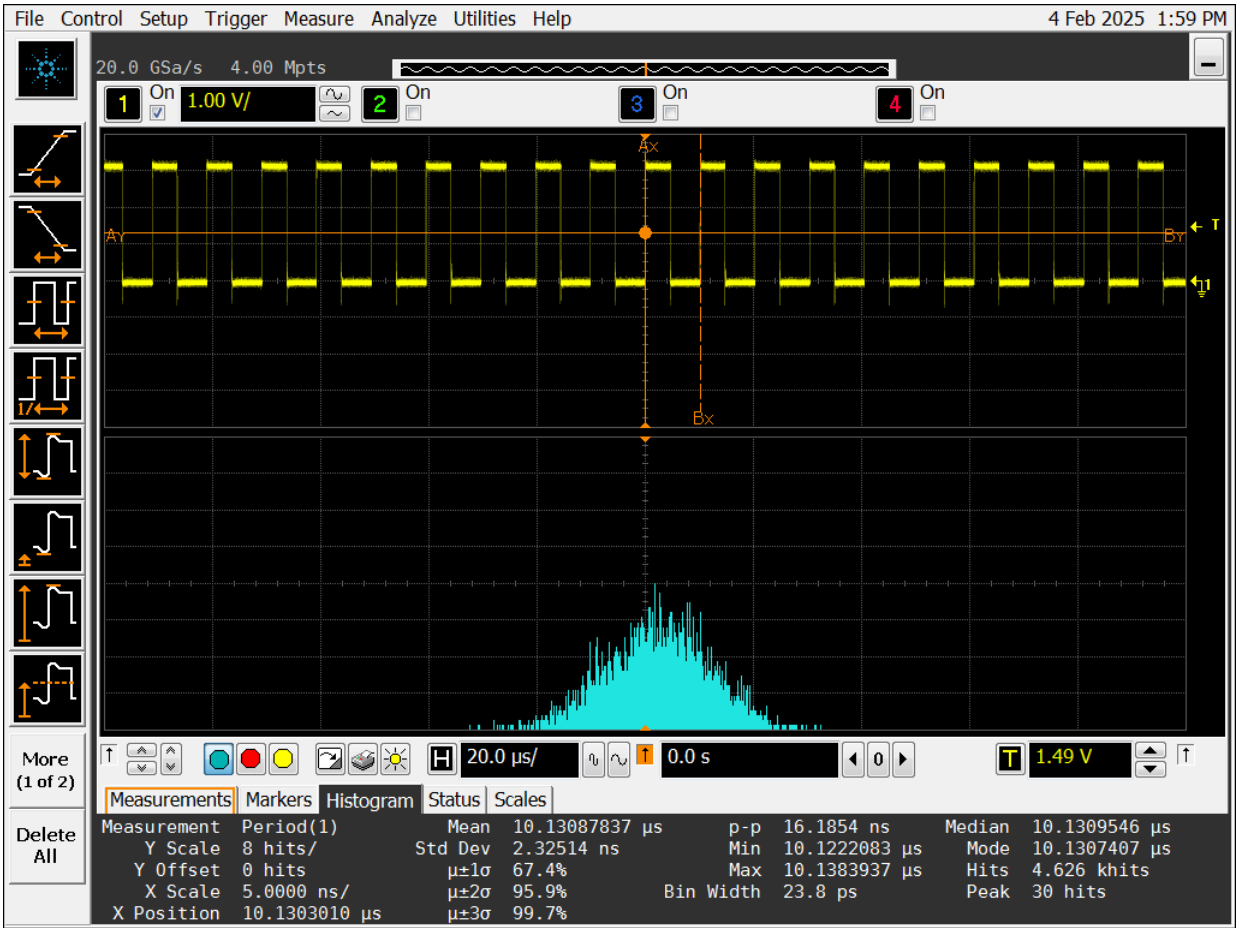
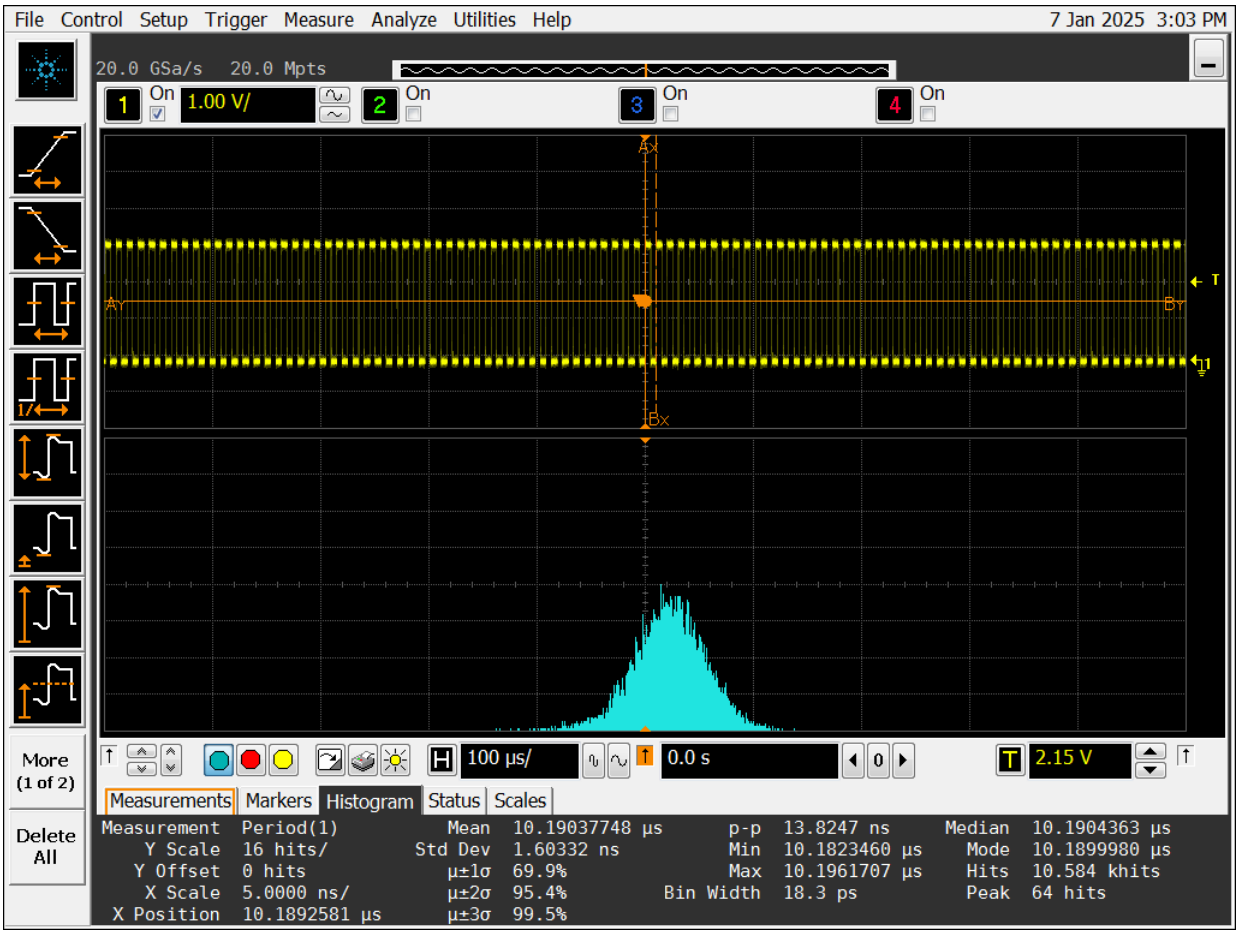


図6-3. 高温でのPWM 100 kHz



温度が室温より低下した場合にもPWM出力に影響が現れます。

図6-4. 低温でのPWM 100kHz



温度の影響の他に、ジッタはデバイスごとにばらつき、経時的にも変化します。これは製造プロセス、部品劣化、電源安定性により生じます。デバイスの長寿命を保証し、経時的なジッタの増加を防ぐ最も効果的な方法は、デバイス動作中の環境を一定に制御する事です。低ノイズ電源によりシステムへの経時的なストレスの注入を低減し、デバイスを高温から保護する事により、経時劣化の進行を遅らせる事ができます。

7. 性能最適化のためのクロック構成例

入力クロック源とPLL設定は、PWM出力ジッタに非常に大きな影響を及ぼします。以下では、dsPIC PWMアプリケーションにおける性能最適化の例を示します。この例で使用する計算式は式7-1に基づきます。

式7-1.

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

PLL回路のVCO出力分周値(PLLFBDIV)が小さいほどジッタは小さくなります。この値を小さくするとフィルタループの周期が短くなり、入力信号の偏差補正の応答速度が向上します。この例では、最小VCO周波数は400 MHzです(デバイス データシートの電気的特性内の仕様値に従う)。400 MHzの出力クロック周波数は各種方法でPLLを使って達成できます。8 MHz FRCを使う場合、PLLは式7-2を使って設定できます。

式7-2.

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

8 MHz FRCをVCO出力分周比=100で使う場合、逓倍比は100です。

より高精度かつ高速な源クロックを使うとジッタをさらに低減できます。PLL参照クロック周波数を8 MHzから25 MHz(MEMSまたは水晶振動子を使った外部クロック)に変更すると、VCO出力分周比は32に低減できます(式7-3参照)。

式7-3.

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

参照クロックの周波数を高くし、それに応じてPLL-VCO出力分周比を調整する事により、PLL性能は向上します。これにより、与えられた信号に対する潜在的ジッタは劇的に減少します。この例には2通りのジッタ軽減策が含まれますが、既述のその他の方法も、確定的ジッタの低減とPWM向けに最善の参照クロックを達成するために考慮すべきです。

8. デバイス別のジッタ試験の例

例として3つのデバイスファミリ(dsPIC33E、dsPIC33C、dsPIC33A)向けに行った試験の設定と結果を以下に記載します。各デバイスに対し、クロック源オプションとしてFRCとFRC+PLLの両方で試験を行い、それらの結果を比較します。ジッタは、REFO(クロック参照出力)モジュールとPWM出力の両方で計測できます。これらの試験はお客様が購入可能な開発ボードを使うため、お客様自身による再試験が可能です。デューティサイクルと周期は容易に変更できるため、より詳細な試験(サイクル間デューティ比変動、TIE等)が可能です。

8.1 dsPIC33E

dsPIC33Eの試験には、デジタル電源開発ボードのいずれかのバージョンと、対応するdsPIC33EP128GS806 DP PIMを使います。PWM出力1はコード内で設定され、PWMモジュールはFRCまたはFRC+PLLをクロック源として使います(コードの冒頭部で定義)。どちらの場合もPWMはジッタ計測用に100 kHzを出力し、2つの結果を比較します。

```
// <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での結果

図8-1と図8-2に、dsPIC33EP128GS806 DP PIM上でのジッタ試験の結果を示します。

図8-1. dsPIC33EP128GS806 PWMジッタ計測結果(FRC)

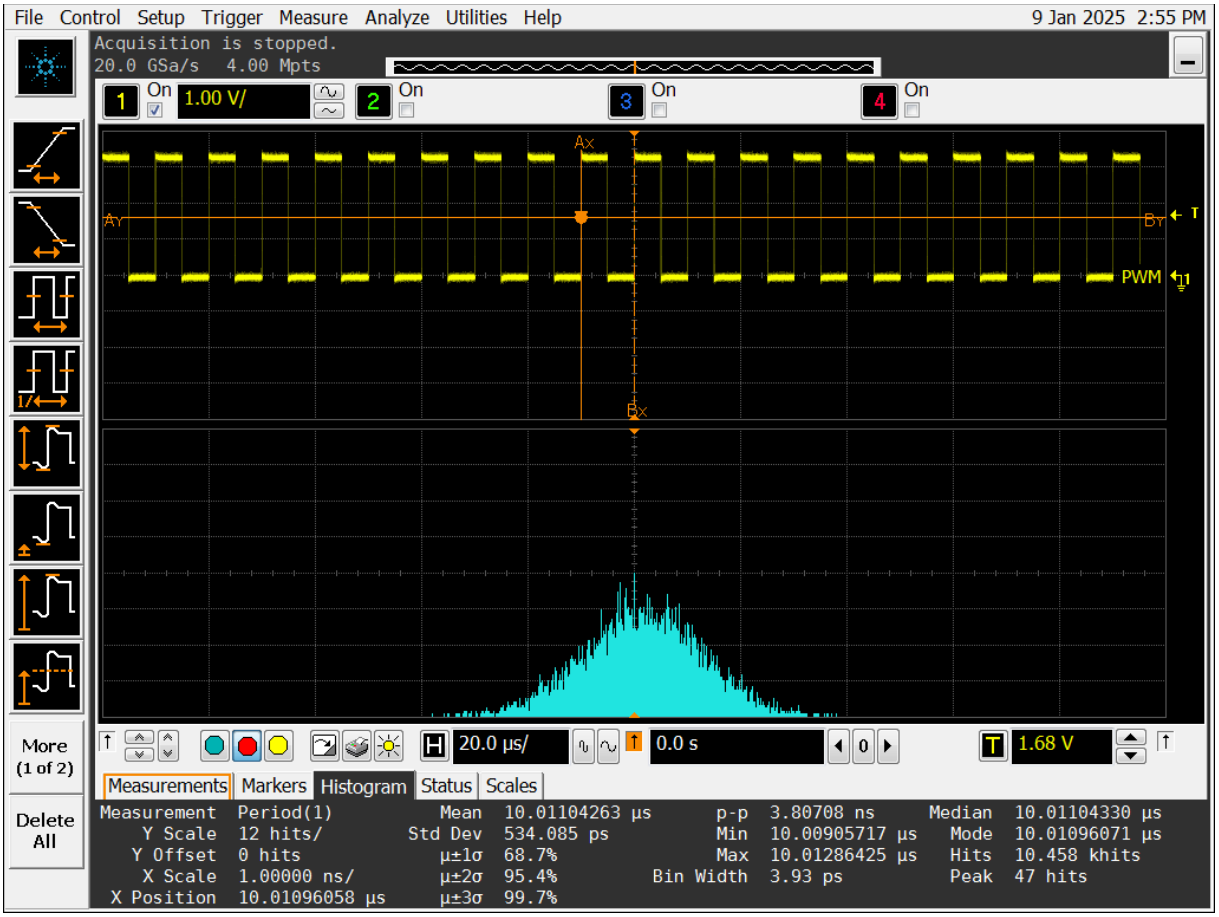
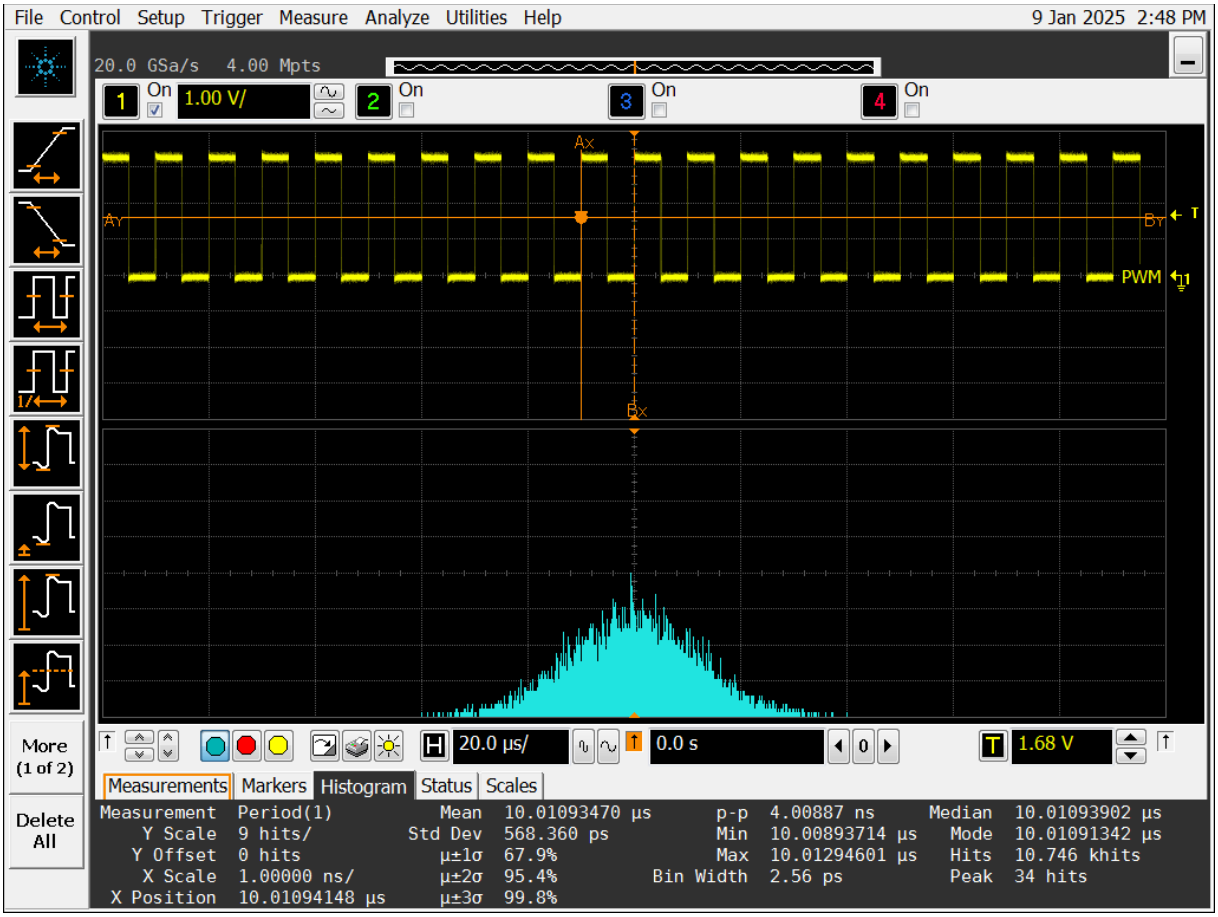


図8-2. dsPIC33EP128GS806 PWMジッタ計測結果(FRC+PLL)



8.2 dsPIC33C

dsPIC33Cの試験にはdsPIC33CK1024MP710を実装したdsPIC33C Touch CAN LIN Curiosity開発ボードを使います。下記のサンプルコードにより、PWM出力とクロック参照出力(REFO)モジュールの両方での試験が可能です。どちらの出力も、FRCまたはFRC+PLLを使って試験できます(コードの冒頭部で定義)。

```
// <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での結果

図8-3と図8-4に、dsPIC33C Touch CAN LIN Curiosityボード上で8 MHz FRCとFRC+PLLをクロック源として使った場合にPWM出力で計測されたヒストグラムを示します。

図8-3. dsPIC33CK1024MP710 PWMジッタ計測結果(FRC)

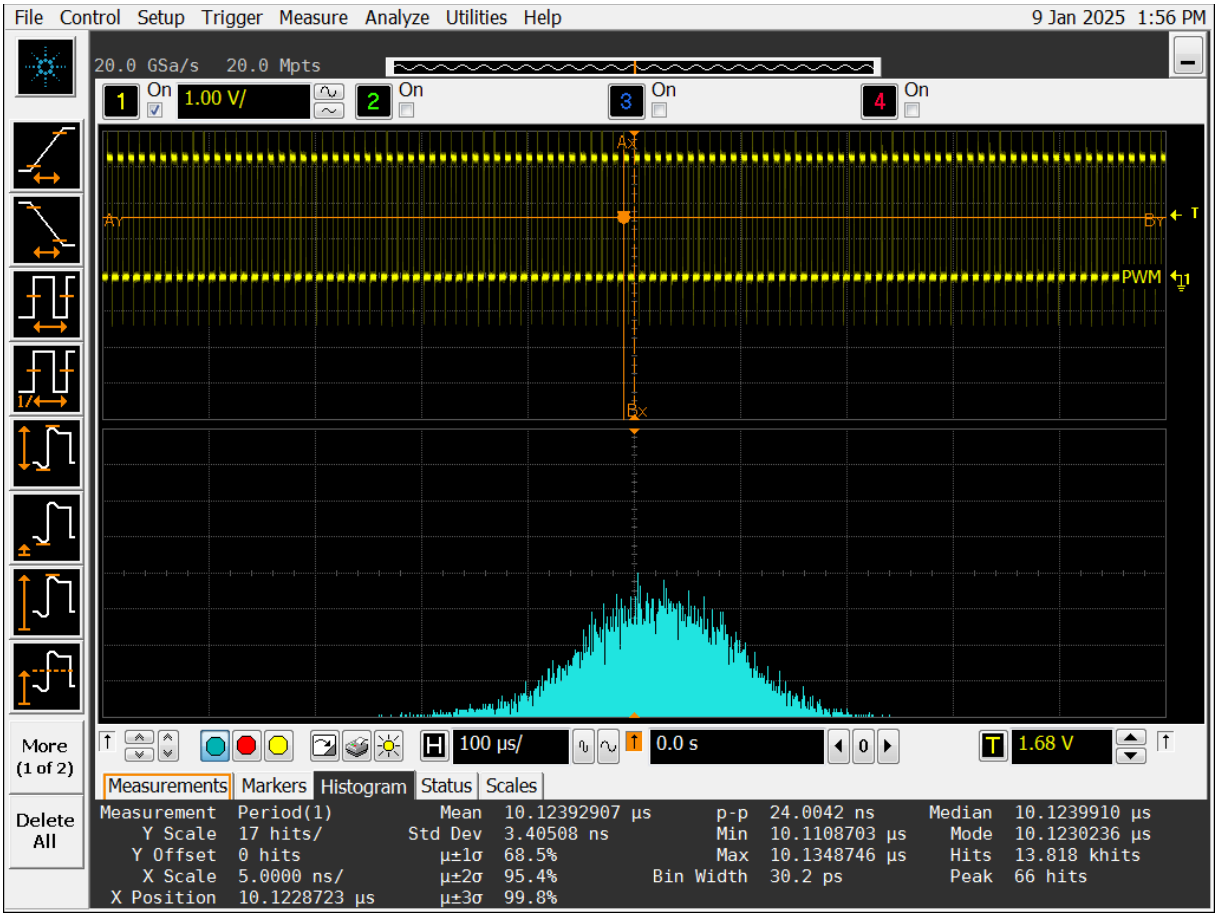
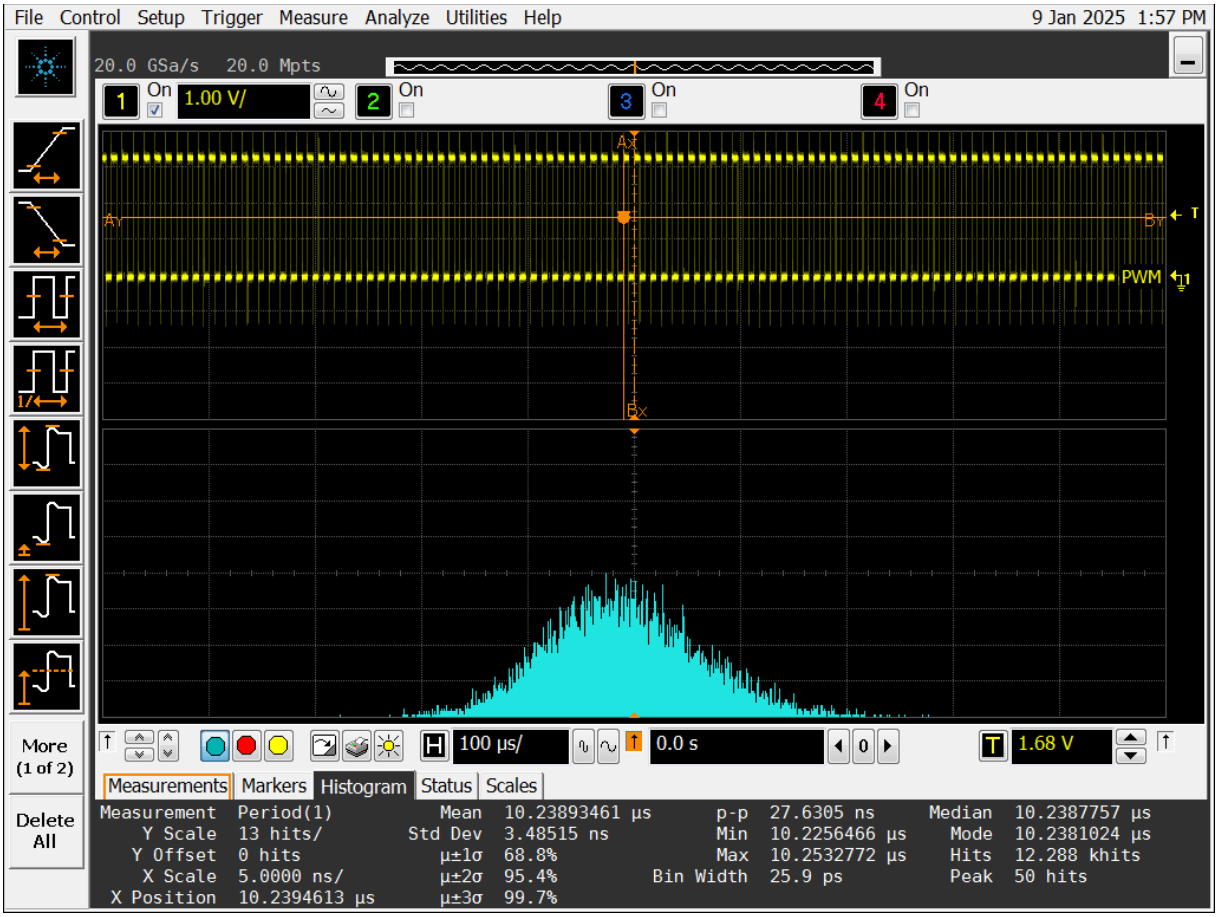


図8-4. dsPIC33CK1024MP710 PWMジッタ計測結果(FRC+PLL)



8.3 dsPIC33A

dsPIC33Aデバイス上でジッタを試験するため、dsPIC33AK128MC106 GP DIMとCuriosityプラットフォーム開発ボードを使います。

下記のサンプルコードはジッタ試験用の各種クロック設定を提供します。この例にはPWMのクロック源オプションとしてFRC、FRC+PLL、8 MHz MEMS、MEMS+PLLが含まれます。PWM信号の分周器は全てのクロック源で出力が100 kHzとなるよう調整され、オシロスコープを使って出力を計測できます。

```
#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での結果

図8-5と図8-6に、クロック源として8 MHz FRCおよびFRC+PLL(400 MHz)を使った場合にPWM出力で計測されたジッタを示します。FRC+PLLの結果(図8-6)では、分布曲線の中心ピークがより明確に表れ、標準偏差が小さくなっています(FRCが1.7 nsであるのに対しFRC+PLLは1.2 ns)。

図8-5. dsPIC33AK128MC106 PWMジッタ計測結果(FRC)

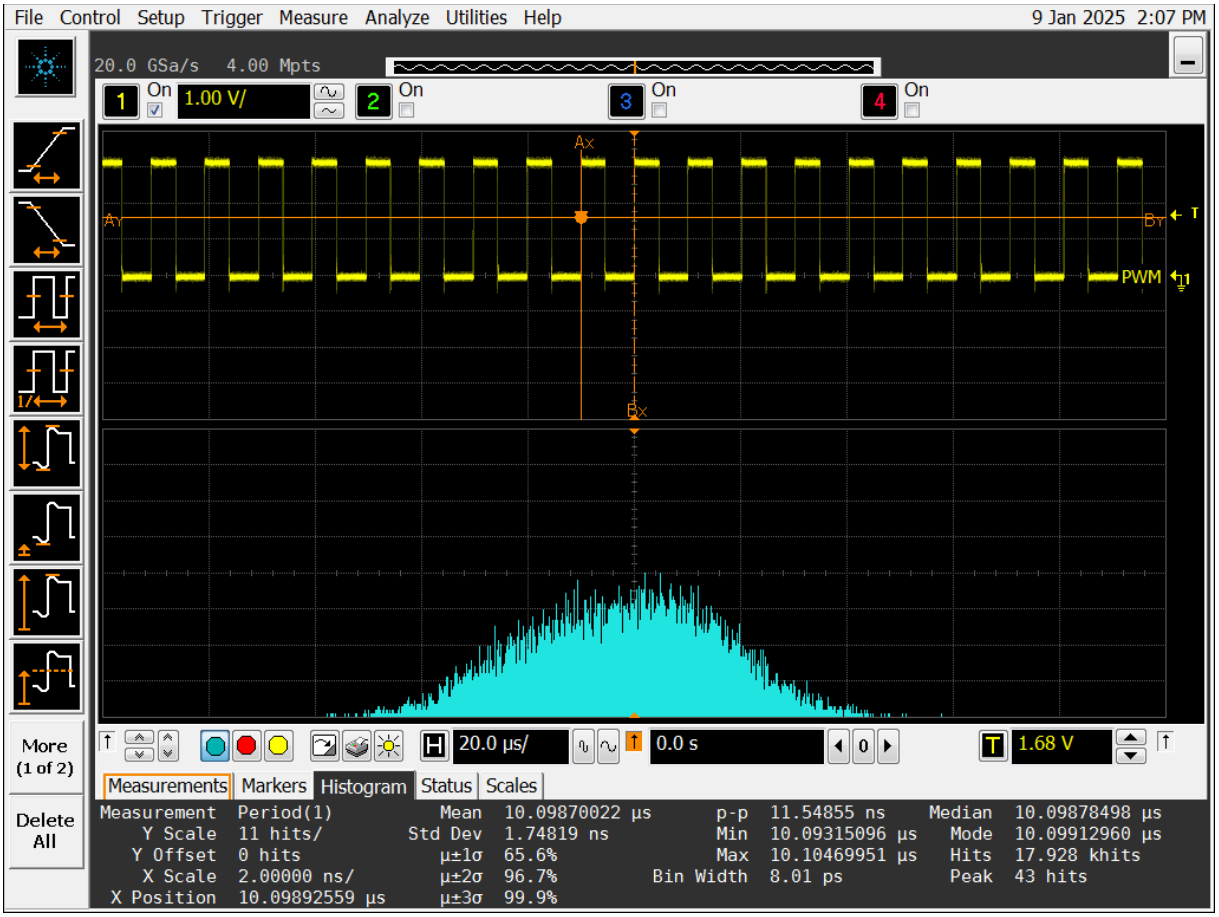


図8-6. dsPIC33AK128MC106 PWMジッタ計測結果(FRC+PLL)

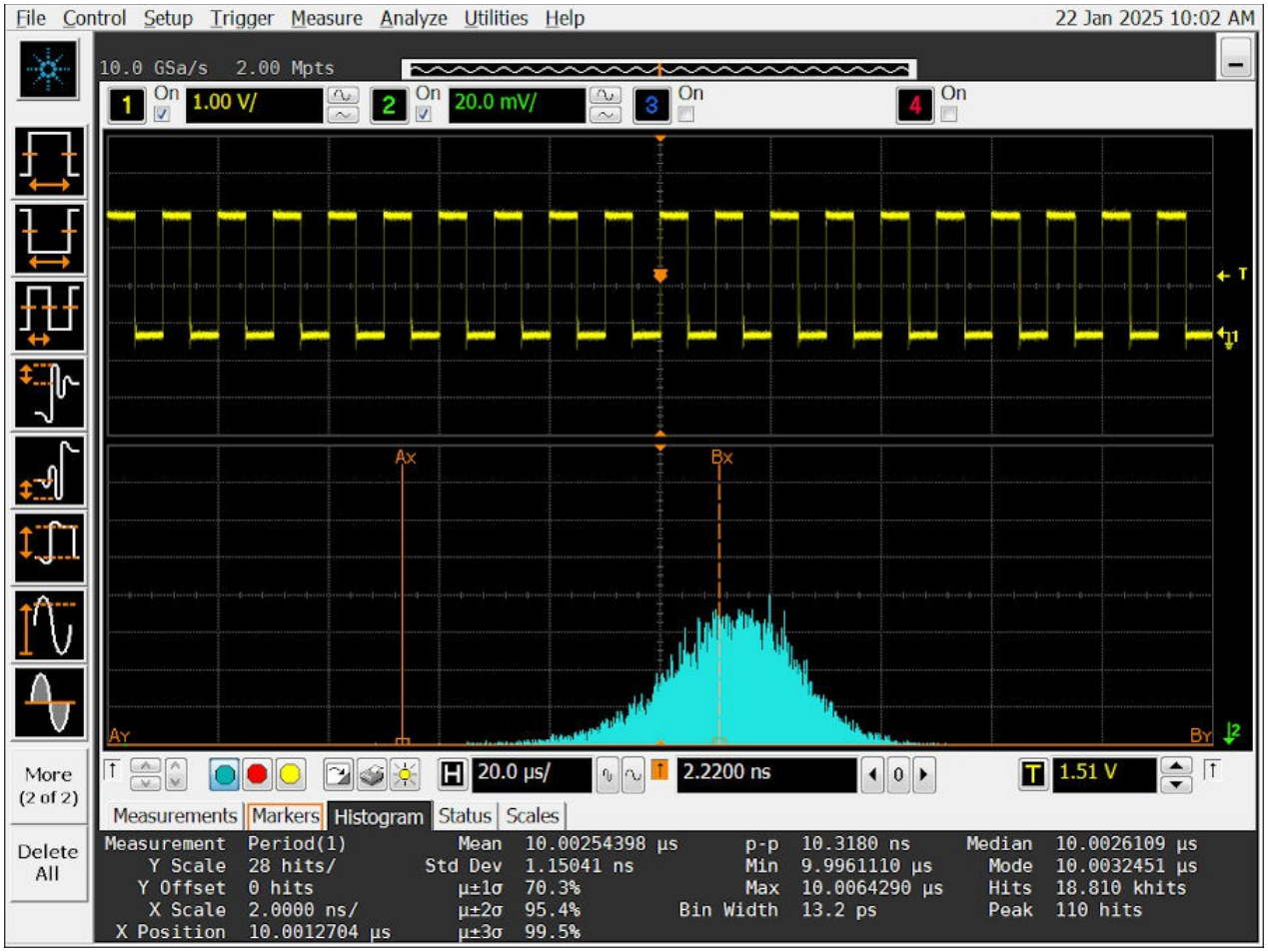


図8-7. dsPIC33AK128MC106 PWMジッタ計測結果(8 MHz MEMS)

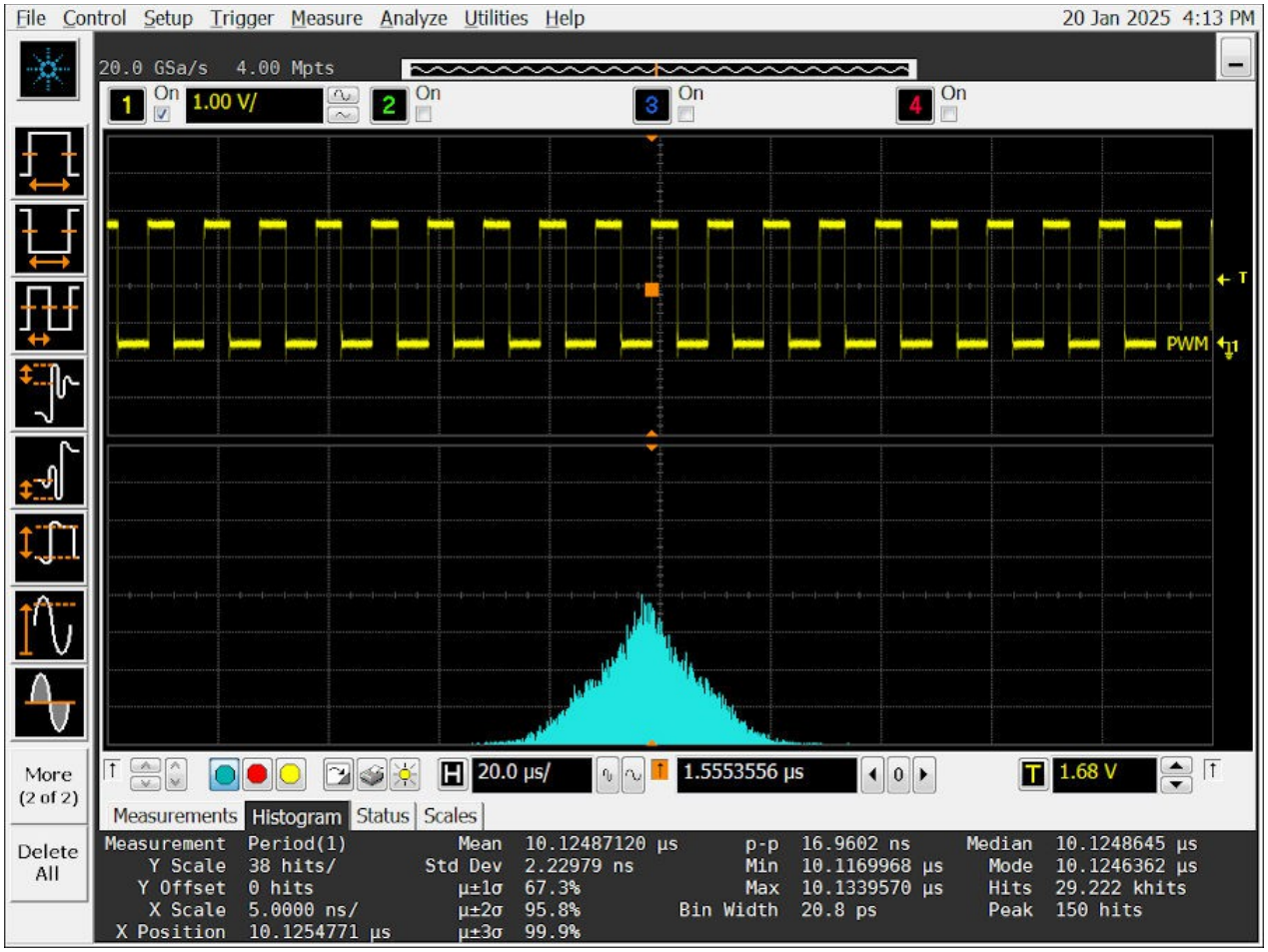
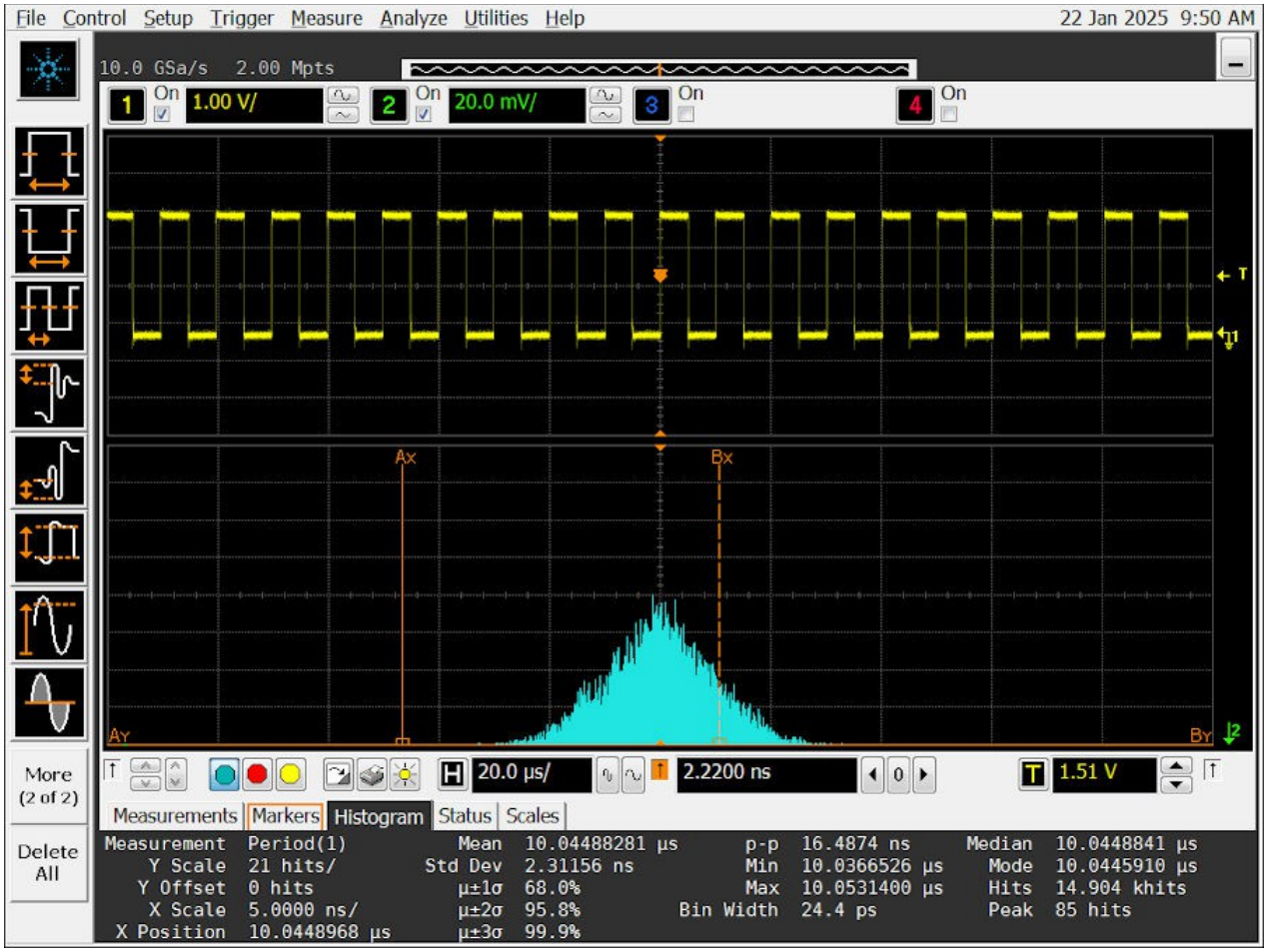


図8-8. dsPIC33AK128MC106 PWMジッタ計測結果(MEMS + PLL)



9. まとめ

本書では、アプリケーション内のジッタ期待値を明示し、ジッタの発生要因を理解して計測および低減するための方法を説明しました。ランダムジッタは全てのシステムに存在しますが、確定的ジッタは本書に記載したツールを使って解析および対処できます。これらのツールを使ってアプリケーションを改善する事で、デバイスおよび周辺システムからより良い結果が得られます。

10. 改訂履歴

本書に適用された変更の履歴は以下の通りです。最新版から順にリビジョンごとに記載します。

リビジョン	日付	変更内容
A	2025年3月	本書は初版です。

Microchip社の情報

商標

ISBN: 979-8-3371-2521-3

法律上の注意点

本書および本書に記載されている情報は、Microchip社製品を設計、テスト、お客様のアプリケーションと統合する目的を含め、Microchip社製品に対してのみ使用する事ができます。それ以外の方法でこの情報を使用する事はこれらの条項に違反します。デバイス アプリケーションの情報は、ユーザの便宜のためにのみ提供されるものであり、更新によって変更となる事があります。お客様のアプリケーションが仕様を満たす事を保証する責任は、お客様にあります。その他のサポートについては、弊社または代理店にお問い合わせになるか、www.microchip.com/en-us/support/design-help/ client-support-services をご覧ください。

Microchip社は本書の情報を「現状のまま」で提供しています。Microchip社は明示的、暗黙的、書面、口頭、法定のいずれであるかを問わず、本書に記載されている情報に関して、非侵害性、商品性、特定目的への適合性の暗黙的保証、または状態、品質、性能に関する保証をはじめとするいかなる類の表明も保証も行いません。

いかなる場合もMicrochip社は、本情報またはその使用に関連する間接的、特殊的、懲罰的、偶発、的または必然的損失、損害、費用、経費のいかんにかかわらず、またMicrochip社がそのような損害が生じる可能性について報告を受けていた場合あるいは損害が予測可能であった場合でも、一切の責任を負いません。法律で認められる最大限の範囲を適用しようとも、本情報またはその使用に関連する一切の申し立てに対するMicrochip社の責任限度額は、使用者が当該情報に関連してMicrochip社に直接支払った額を超えません。法律で認められる最大限の範囲を適用しようとも、本情報またはその使用に関連する一切の申し立てに対するMicrochip社の責任限度額は、使用者が当該情報に関連してMicrochip社に直接支払った額を超えません。

Microchip社の明示的な書面による承認なしに、生命維持装置あるいは生命安全用途にMicrochip社の製品を使用する事は全て購入者のリスクとし、また購入者はこれによって発生したあらゆる損害、クレーム、訴訟、費用に関して、Microchip社は擁護され、免責され、損害をうけない事に同意するものとします。特に明記しない場合、暗黙的あるいは明示的を問わず、Microchip社が知的財産権を保有しているライセンスは一切譲渡されません。

Microchip社のデバイスコード保護機能

Microchip社製品のコード保護機能について以下の点にご注意ください。

- Microchip社製品は、該当するMicrochip社データシートに記載の仕様を満たしています。
- Microchip社では、通常の条件ならびに仕様に従って使った場合、Microchip社製品のセキュリティ レベルは、現在市場に流通している同種製品の中でも最も高度であると考えています。
- Microchip社はその知的財産権を重視し、積極的に保護しています。Microchip社製品のコード保護機能の侵害は固く禁じられており、デジタル ミレニアム著作権法に違反します。
- Microchip社を含む全ての半導体メーカーで、自社のコードのセキュリティを完全に保証できる企業はありません。コード保護機能とは、Microchip社が製品を「解読不能」として保証するものではありません。コード保護機能は常に進歩しています。Microchip社では、常に製品のコード保護機能の改善に取り組んでいます。