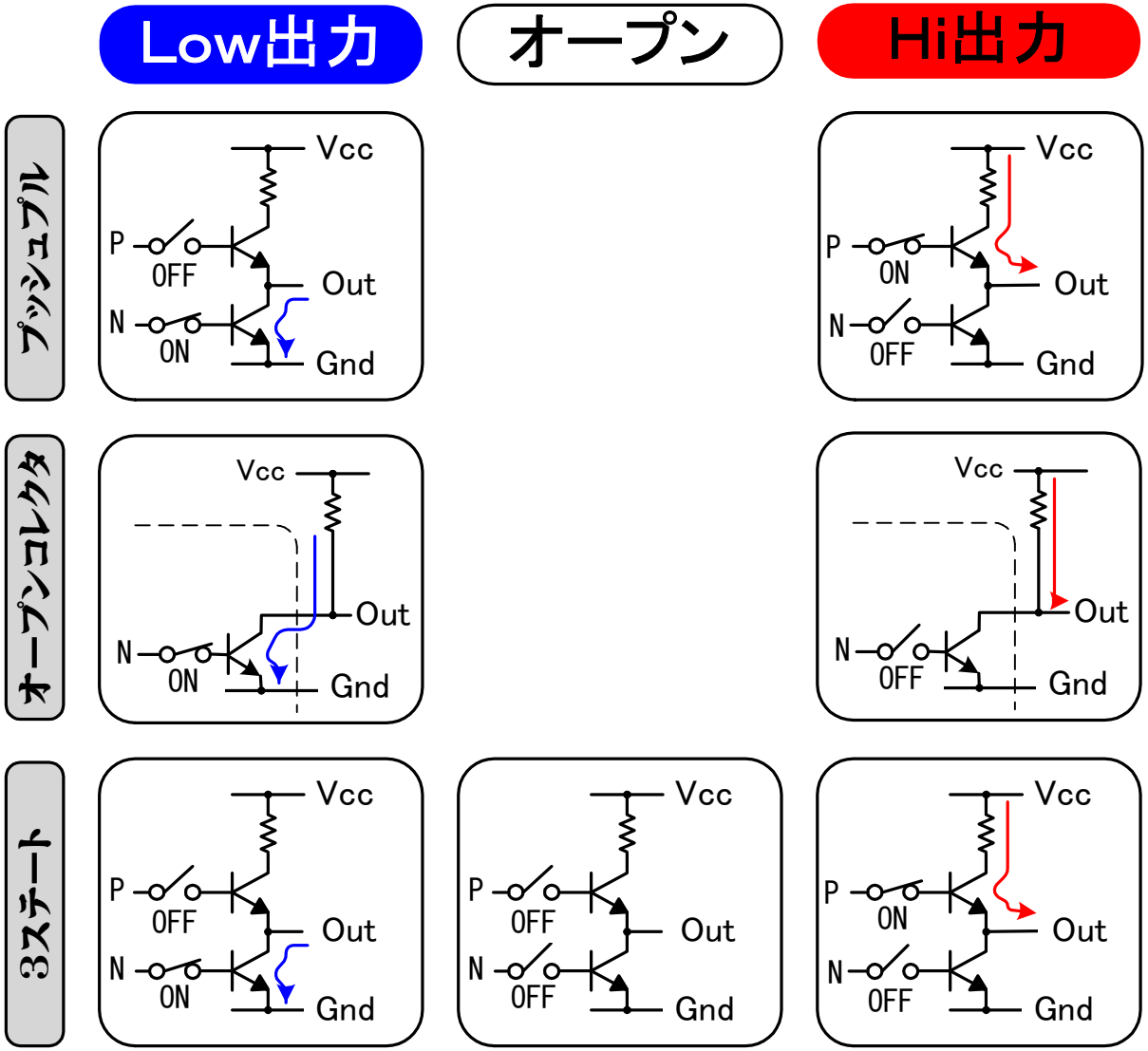
出力端子の種類

TTLゲートＩＣの出力端子の種類：

① 出力段が プッシュプル動作で
Hiにも、Lowにも
引っ張るタイプ。

② オープンコレクタタイプ、通常
外部抵抗で、PullUpするタイプ。
ワイヤードOR接続が可能。

③ ３ステート出力、Hi出力、
Low出力、Open状態の
３つの状態を持ちます。



入力端子の種類

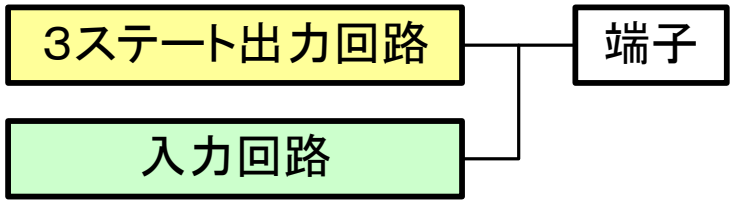
TTLゲートICの入力端子は、スレシホールドレベルが、1.4Vぐらいです。入力端子をオープンにすると、Hi入力状態になります。　これに対し、マイコンも含めたC-MOS素子の入力端子は、電源電圧の半分の電圧が、スレシホールドレベルになります。

C-MOS素子は、入力インピーダンスが非常に高く、オープンにすると ノイズを拾い好ましくありません。　よって、未使用端子は、抵抗で PullUpするか、GNDに落とす事が望ましいです。　マイコンの場合 未使用端子は、出力端子に設定しておく方法も 考えられます。

マイコンも含めた C-MOS系の素子は、昔から静電気に弱いと言われてます。　遥か昔の C-MOS ゲートよりは、耐性を改善してある様ですが 特に冬場、扱いには気を付けて下さい。

腕にリストバンドを付けてアースに接続しておくのが効果的です。　あと マイコンには、一部の端子を アナログ入力端子として設定し、A/D変換、または、アナログコンパレータとして使えるものもあります。 A/D変換器は、通常 10bitか12bitぐらいが 多いです。A/D変換器そのものは１個で、その前段にサンプルホルダーがあり、さらにその前に アナログマルチプレクサ（切替え器）が付いており 入力端子を切り替えながらA/D変換します。 精度を要求する場合、アナログの基準電圧を用意する必要もあります。　A/D変換に関しては、また別の機会に詳しく説明します。

マイコンの入出力端子



マイコンの入出力端子は、３ステートの機能を持ちます。 そして この出力回路と入力回路は、入出力端子のところで接続されています。　出力端子として設定すると、Hiか Lowを出力します。　入力端子として設定すると出力端子は、オープン状態となります。

　よって、外からの信号を取り込めます。

Pinのところで、出力と入力が接続されているので、出力端子として設定している端子を、読み込むと出力端子が出している信号を取り込む事になります。

　マイコンの端子は 入出力の設定は当然ありますがマイコンの種類により端子を PullUpする機能や駆動力の強弱設定、用途によりオープンドレイン駆動に切り替える設定が、マイコンの種類により ありま す。

今回の基礎プログラミング

R8C/M120Aマイコンを使って、今回行うプログラミングですが、割込み処理やアセンブラは 使わず Cプログラムだけで、シンプルに行います。
そして実験回路で動作を確認します。

最初、右の ①～⑤ まで一気にやってしまうつもりでしたが、制作時間が、ややかかり過ぎるのと見る人も疲れるので小分けにする事にしました。　実験基板は、②～⑤の実験 も 見据えて作りました。

＊＊＊　今回、実装する機能　＊＊＊

---

① クロックを 高速オンチップオシレーター（20MHz）に変更する。

★ 次回、行う予定 ★

---

② キー入力のチャタリングキャンセル

③ キーが押された瞬間のエッジ検出

④ 一定のパルス幅を出すワンショットパルスの機能を実現する。

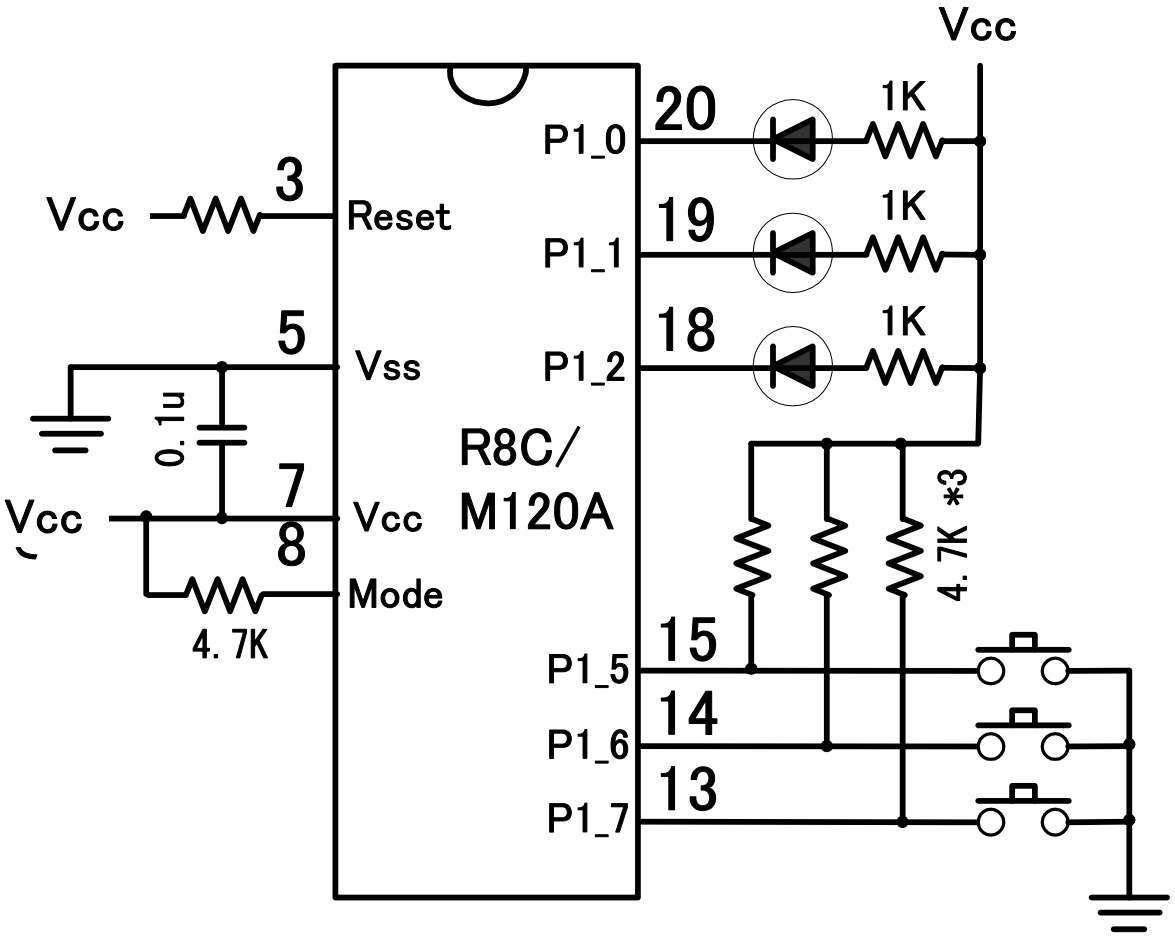
⑤ 複数のキーの多重押しの際に、先着優先機能を付ける。

今回の実験回路

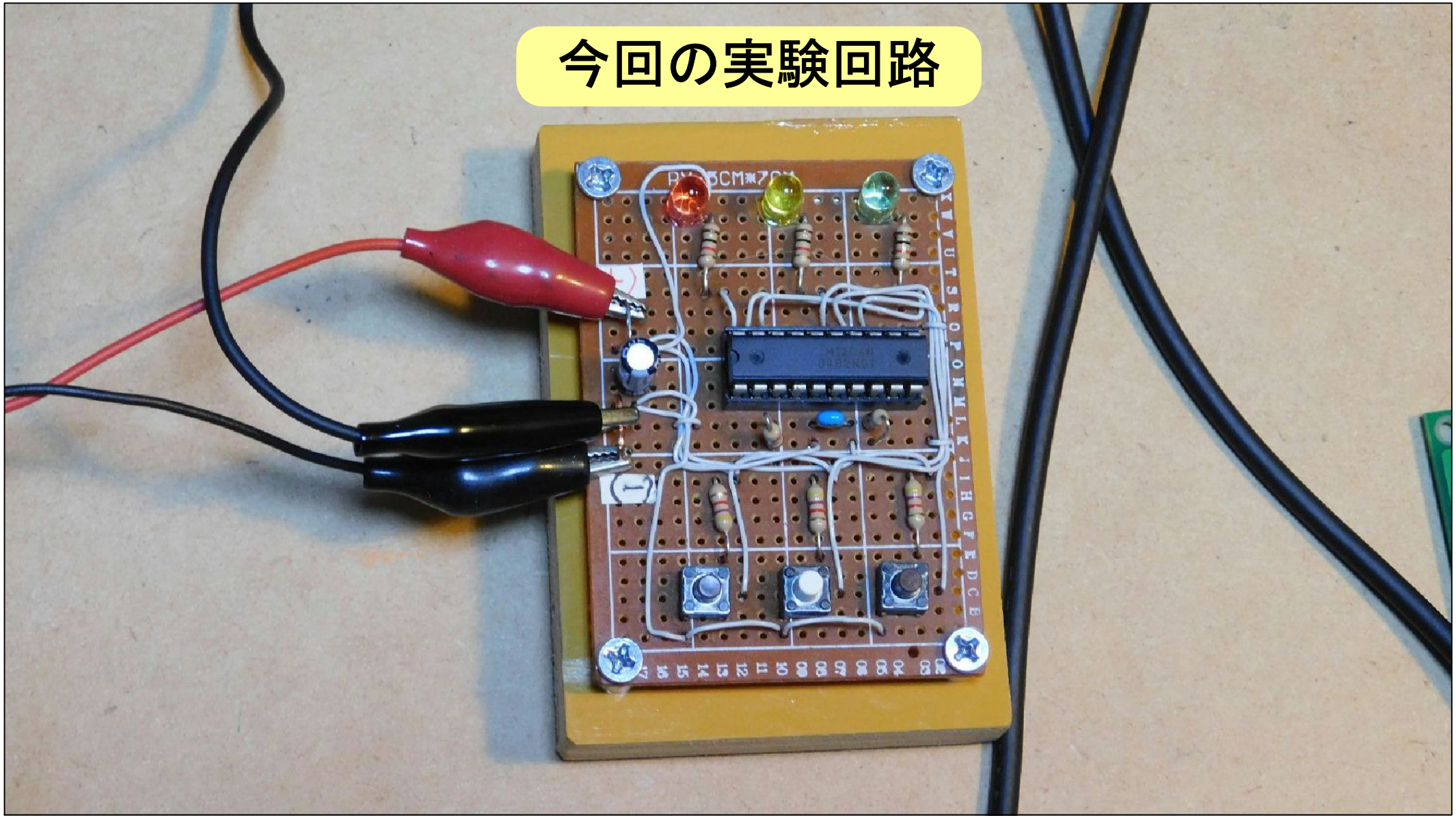
① マイコンには、R8C/M120Aを使用します。最低限必要な配線は、
3 Pinが 4.7K～10Kの抵抗を入れて電源に接続します。
5 Pinが GND
7 Pinが 電源(5V または 3.3V)
5Pin と 7Pin の近くに跨ぐ形で
0.1uFの積セラを入れる。
8 Pinが 4.7K～10Kの抵抗を入れて電源に接続します。

② 入力には、３つの PullUpされた押しボタンスイッチを接続します。

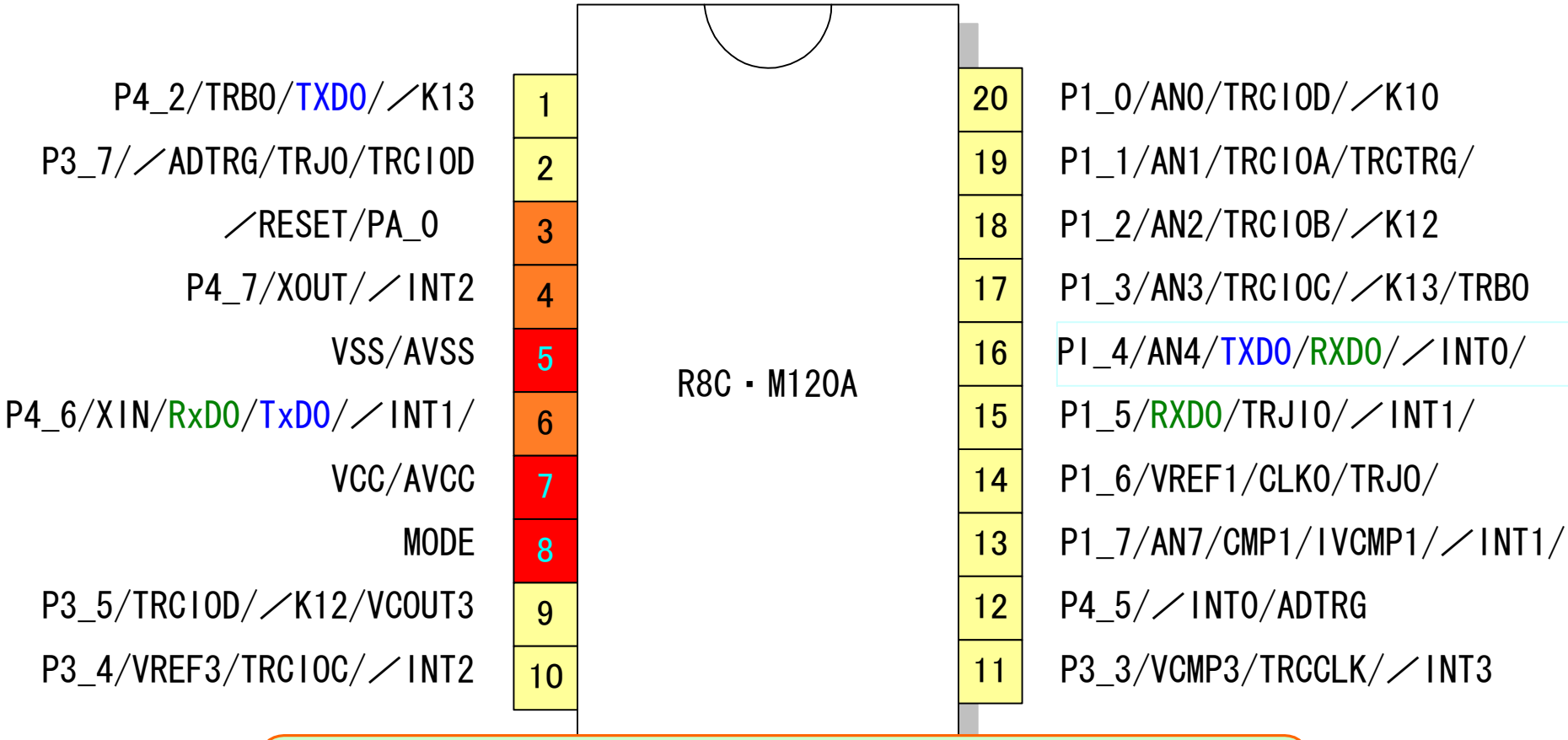
③ 出力には、３つの LED＋電流制限抵抗を接続します。



④ 電源と GND間に 10～33uFの電解コンを入れる。


今回の実験回路

R8C/M120A CPUの Pinアサイン



赤のPin（ 5, 7, 8 ）は、他の用途では使えない。
柿色のPin（ 3, 4, 6 ）は、通常は、RESET、XOUT、XIN に使用するが、
Pinが 足りない時は、条件付きで 別の用途にも使える。

R8C/M120Aのポートレジスタ

Port.1のレジスタは、b7～b0まで全て、P1_7～P1_0にアサインされる。 **水色は、A/D入力と重なる Pin。** 当然、A/D入力として使用するPinはI/O Pinとしては 使えない。

Port.3のレジスタは、b7, b5, b4, b3 の 4bitが、P3_7, P3_5, P3_4, P3_3 に アサインされる。

Port.4のレジスタは、通常 b5, b2 の 2bitが、P4_5, P4_2 にアサインされる。
水晶発振子を使用しないのであれば、
b7(P4_7)、B6(P4_6)も、使える。

Port.Aの 3Pin端子を RESET信号入力として使わない場合、B0(PA_0)として 使える。

Port.1							
b7	b6	b5	b4	b3	b2	b1	b0
P1_7	P1_6	P1_5	P1_4	P1_3	P1_2	P1_1	P1_0
13	14	15	16	17	18	19	20

Port.3							
b7	b6	b5	b4	b3	b2	b1	b0
P3_7		P3_5	P3_4	P3_3			
2		9	10	11			

Port.4							
b7	b6	b5	b4	b3	b2	b1	b0
P4_7	P4_6	P4_5			P4_2		
4	6	12			1		

Port.A							
b7	b6	b5	b4	b3	b2	b1	b0
							PA_0
							3

クロックを 高速オンチップオシレーター (20MHz)に変更するプログラム

```
//*********************************************
//**   高速内部オシレータ 20MHzに切り替え     **
//*********************************************
void sel_int_osc_20m( void )
{
      asm( "fclr I" );      // 割り込み処理 禁止
      min_wait( 1000 );     // ★ 若干の時間待ち

      prc0 = 1;             // Protect bit 0を 解除する
      ococr = 0x01;         // 高速オシレータ発振開始
      min_wait( 1000 );     // ★ 若干の時間待ち

      sckcr = 0x40;         // 高速オンチップ OSC選択
      ckstpr = 0x80;        // システムクロック高速選択
      prc0 = 0;             // プロテクト有効化
      min_wait( 100 );      // ★ 若干の時間待ち
}
```

```
//*********************************
//**   若干の時間待ち              **
//** ------------------------- **
//**   n : 時間待ちループ回数     **
//*********************************
void min_wait( int n )
{
      int   i;

      for( i=0; i<n; i++ );
}
```

プログラムソース名
simple_1.c

```
#include "sfr_r8m12a.h"

#define  LED_G  p1_2  // 緑 LEDポート
#define  LED_Y  p1_1  // 黄 LEDポート
#define  LED_R  p1_0  // 赤 LEDポート


void initproc( void )
{
      sel_int_osc_20m();  // 内部OSC 20MHz
      LED_G = 1;     // 緑LED 消灯
      LED_Y = 1;     // 黄LED 消灯
      LED_R = 1;     // 赤LED 消灯

      pd1 = 0x1f;     // Port1入出力設定
      pd3 = 0xb8;     // Port3入出力設定
      pd4 = 0x64;     // Port4入出力設定
}
```

初期化処理とメイン関数 プログラム

```
void main(void)
{
    int    i;

    initproc();    // 初期化処理
    while( 1 )
    {
        LED_G = 1;          // 緑LED 消灯
        LED_R = 0;          // 赤LED 点灯
        for( i=0; i<5000; i++ );  // Wait
        LED_R = 1;          // 赤LED 消灯
        LED_Y = 0;          // 黄LED 点灯
        for( i=0; i<5000; i++ );  // Wait
        LED_Y = 1;          // 黄LED 消灯
        LED_G = 0;          // 緑LED 点灯
        for( i=0; i<5000; i++ );  // Wait
     }
}
```

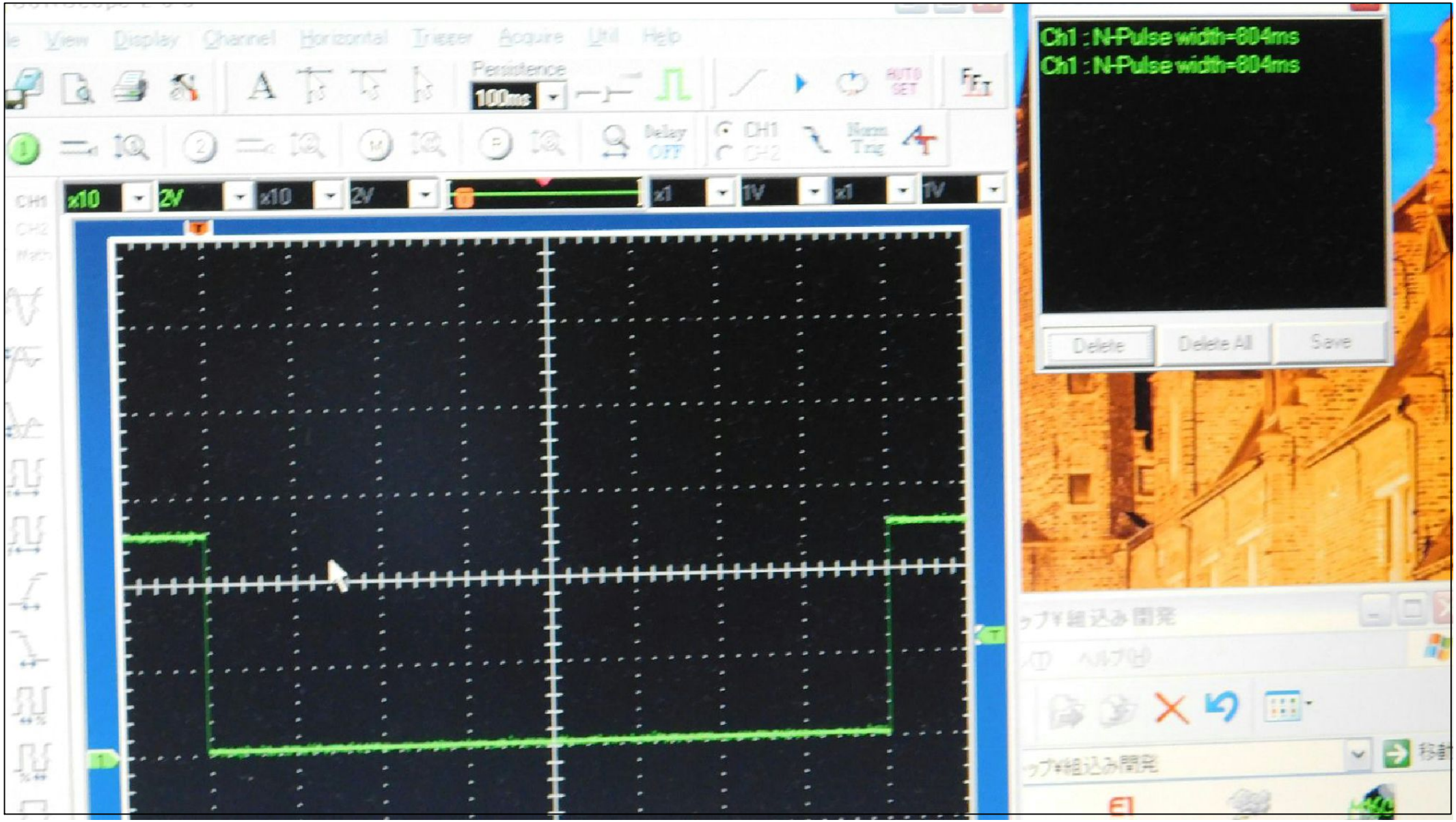
CPUクロックを高速に変更する実験の確認方法

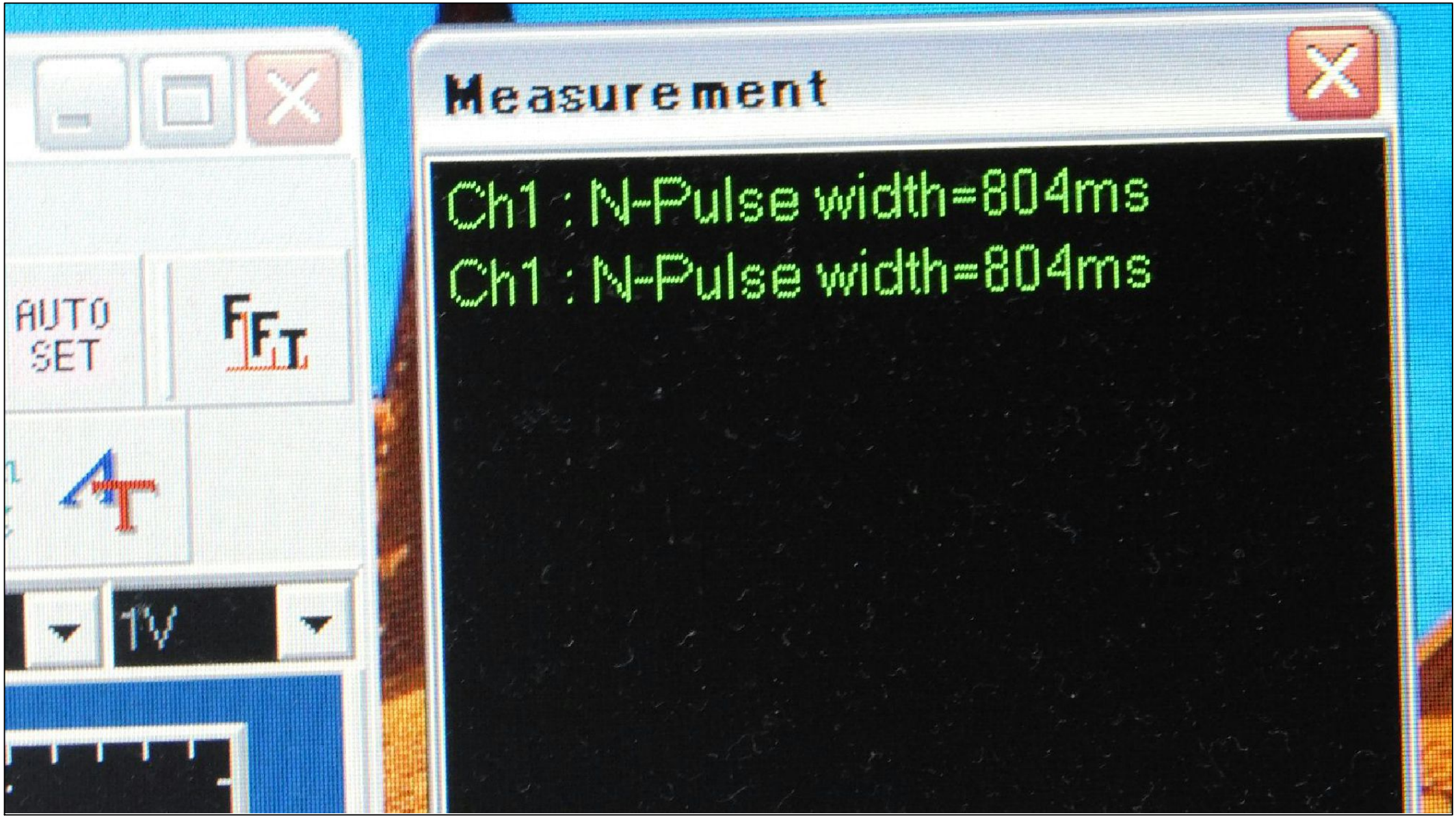
最初、初期化の sel_int_osc_20m()を、コメント化して、プログラムを作っておきます。

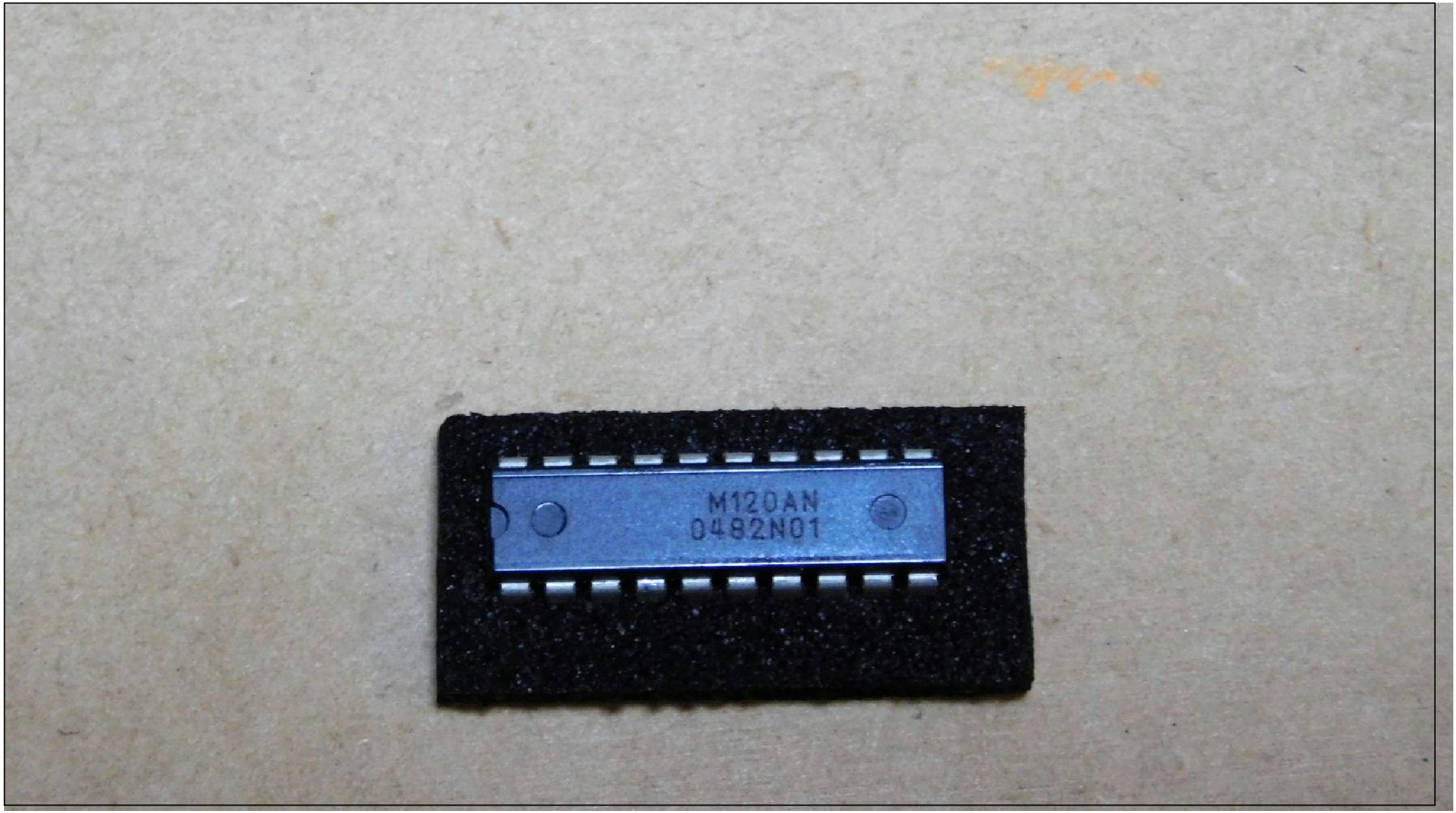
電源オン直後の低速クロックで、LEDを
赤、黄、緑と順次点灯させます。
（ ゆっくり点灯が、切り替わります。）
各LEDの点灯が切り替わる間に、for文による空ループが、5000回まわります。

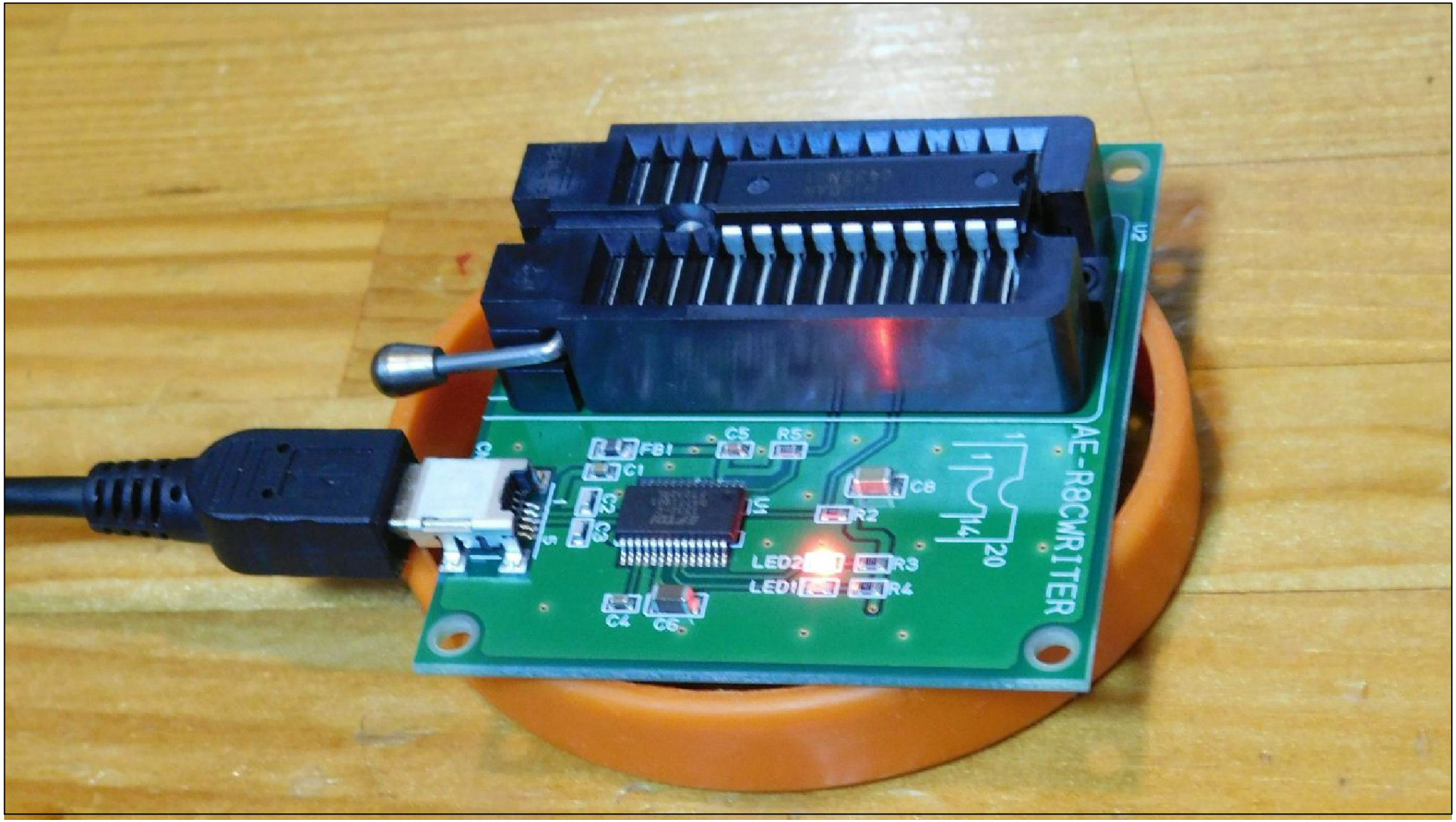
１個のLEDが点灯している時間をオシロの
リードアウト機能で計ります。

次に、初期化の sel_int_osc_20m()を、生かして高速クロックで動作させます。
当然、コンパイルして、MOTファイルをマイコンに書き込みます。
（ 今度は、早すぎて赤、黄、緑が 全て点灯しているように見えます。）
先ほどと同じく１個のLEDが、点灯している時間をオシロの リードアウト機能で計ります。
これで、何倍 早くなったか分かります。


Persistence
100ms
x10
2V
x10
2V
x1
1V
x1
1V
Ch1 : N-Pulse width=804ms
Ch1 : N-Pulse width=804ms
Delete
Delete All
Save


Measurement
Ch1 : N-Pulse width=804ms
Ch1 : N-Pulse width=804ms
AUTO SET
FFT
TV


M120AN
0482N01


AE-R8CWRITER
U2
FB1
C5
R5
C1
C8
U1
R2
C2
C3
LED2
R3
LED1
R4
C4
C6
1
14
20