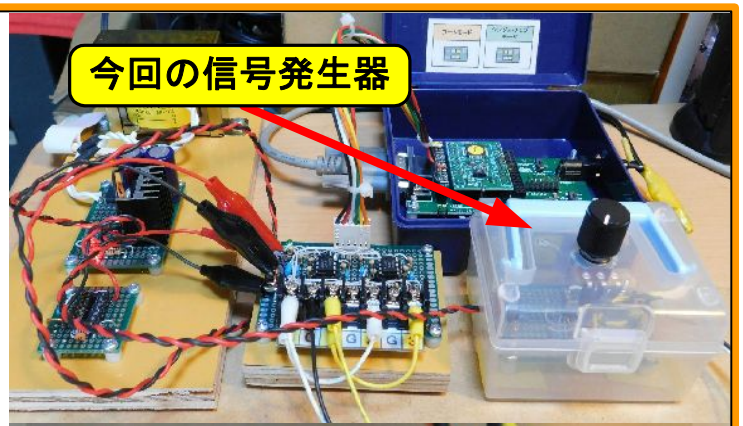
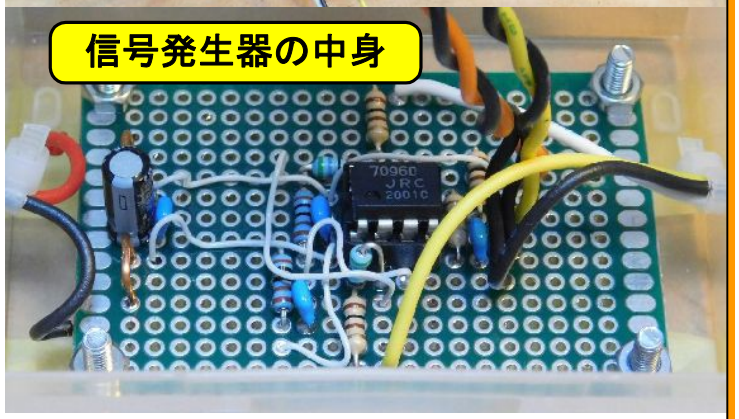
テスト用　手動アナログ信号発生器

今回は、最初に手動アナログ信号発生器を、用意する事にしました。
通常のファンクションジェネレータとは、異なる形で ボリュームを回す事でテスト信号を発生させようと考えました。 で、1チャネルだけの出力だけでは 複数入力のA/D入力において単調になるので 電源 5Vで、中点の 2.5Vを、中心に極性を入れ替えた逆相信号も 用意する事にしました。

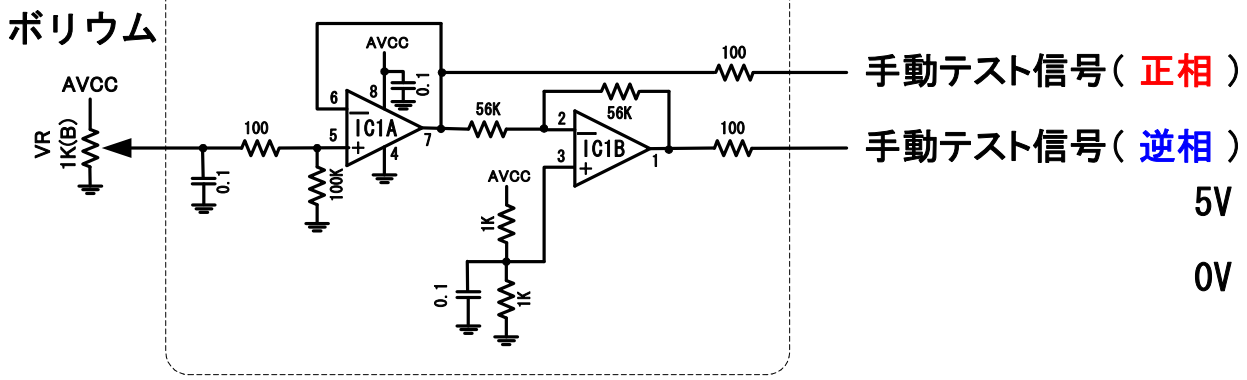
① 正相出力が、5Vのとき、逆相出力は 0V
② 正相出力が、0Vのとき、逆相出力は 5V
③ 正相出力が、2.5Vのとき、逆相出力も 2.5V に、なります。

使用する部品の関係で 精度は いまいちです。

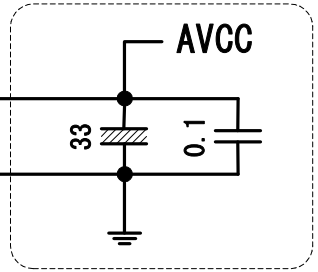




手動テスト信号 生成回路



電源 引き回し



RX220 A/D変換処理ソフト作成

RX220の 12bit A/Dコンバーターは、以下の 3つの動作モードを持っています。

[1] シングルスキャンモード
[2] 連続スキャンモード
[3] グループスキャンモード

シングルスキャンモードは、一番シンプルで、A/Dスタートを行うと 1回 A/D入力のスキャンを行います。今回は、シングルスキャンモードを使用します。

RX220に おいては 1回のA/D変換ではなく、1回のA/Dスキャンというような表現をしてあります。

私は、最初 単純に 1チャネル1回のA/D変換をするのは、どうアクセスするのかと思っていたので、そこに考え方の違いがあったようです。

どういう事かというと、今まで私は、1回のA/D変換において、① 入力チャネルの選択、② A/D変換スタート、③ A/D変換終了確認、④ A/D変換後の量子化データ取り出し というシーケンスをイメージしていたのです。 つまり、1チャネル毎にソフトで、上記①～④のアクセスを行うと思っていたのです。

ところが、RXシリーズでは、A/Dコンバータ周辺回路が進化しており、例えば、AN000～AN003の4チャネルをサンプリングする場合は、ADANSA レジスタに 0x000F を 設定すれば、ハード側で、4チャネル連続で A/D変換処理を行います。そしてADCSRレジスタのADSTビットに 1 を 書き込む事で、A/D変換が開始します。
ADCSR. ADSTビットが 1の時は A/D変換中です。
ADCSR. ADSTビットが、0 になったら、4チャネル全てがA/D変換終了してます。

ちなみに AN000単独で A/D変換を行う場合は、ADANSAレジスタに 0x0001 を設定します。
要は、ADANSAレジスタの b0 が AN000 に対応、b1 が AN001 に対応、b2 が AN002 に対応、b3 が AN003 に対応しています。 b15まで同様に対応しています。
よって、AN000～AN003の4チャネルの場合、ADANSAレジスタに 0x000Fを設定する事になります。

A/Dコンバータは、1つですが 変換データを書き込むレジスタはチャネル毎に独立して対応し 16個分かれています。 よって、複数チャネルA/Dスキャンして も、データを 上書きして消失する心配は ないです。

A/Dコンバータ アクセス処理ソース

```
void  init_adc( void );  // 12bit A/Dコンバータ初期化 処理 4ch シングルスキャンモード
void  start_adc( void );  // A/Dコンバータ シングルスキャン スタート
void  get_ad_scan( short  ad[] );    // A/D変換データ 4ch分 取り出し
```

（ init_adc関数は、ちょっと長いの で次のページに示します。）

```
void  start_adc( void )
{
    S12AD.ADCSR.BIT.ADST = 0x1;              // AD変換スタート
}

void  get_ad_scan( short  ad[] )
{
     while( S12AD.ADCSR.BIT.ADST != 0 );       // A/D変換終了まで 待ち

     ad[0] = S12AD.ADDR0;            // A/D変換データ ch.0 取り出し
     ad[1] = S12AD.ADDR1;            // A/D変換データ ch.1 取り出し
     ad[2] = S12AD.ADDR2;            // A/D変換データ ch.2 取り出し
     ad[3] = S12AD.ADDR3;            // A/D変換データ ch.3 取り出し
}
```

```
#define    ADC_STATE  20          // A/D変換時間の単位
void init_adc( void )
{
    // Port設定
    PORT4.PMR.BYTE = 0x0F;                    // Port40～43を周辺機能とする
    PORT4.PDR.BYTE = 0x00;                    // 入力端子

    // S12ADモジュールを 有効化
    SYSTEM.PRCR.WORD = 0xA503;               // クロックソース選択の保護解除
    MSTP(S12AD) = 0;                          // S12ADを有効化
    SYSTEM.PRCR.WORD = 0xA500;               // クロックソース選択の保護

     // A/D 変換設定
    S12AD.ADCSR.BIT.ADCS = 0;                    // シングルスキャンモード
    S12AD.ADANSA.WORD = 0x000F;               // AN000～AN003 を 有効にする

    S12AD.ADSSTR0 = ADC_STATE;                // AN000 : 20 ステート ( 暫定値 )
    S12AD.ADSSTR1 = ADC_STATE;                // AN001 : 20 ステート ( 暫定値 )
    S12AD.ADSSTR2 = ADC_STATE;                // AN002 : 20 ステート ( 暫定値 )
    S12AD.ADSSTR3 = ADC_STATE;                // AN003 : 20 ステート ( 暫定値 )
}
```

A/Dアクセス処理、呼び出し側

```
void main(void)
{
      short ad[4];

      init_proc();          // 初期化処理   init_proc関数の内容は次のページに示します。
      while( 1 )
      {
            set_timer_1m1( 50 );              // 50ms時間待ち
            while( get_timer_1m1() > 0 );

            PORTH.PODR.BIT.B0 = 1;            // LED 点灯
            start_adc();                      // A/Dコンバータ シングルスキャン スタート
            get_ad_scan( ad );                // A/D変換データ 4ch分 取り出し
            send_addat( ad, 1 );              // A/Dデータ 送信
            PORTH.PODR.BIT.B0 = 0;            // LED 消灯
      }
}
            // 通信に関わるややこしい部分は、取り外しました。
```

```
void init_proc()          // 全体の 初期化処理
{
     disable_irq();                  // CPU I Flag = 0  割り込みを禁止する

//   CPU クロック初期化
// ---------------------------------------------------------------
     setup_main_clk_20m();           // メインクロック 20MHz 切り替え
//   setup_hoco_clk_32m();           // HOCOクロック 32MHz 切り替え

//  I/Oポート 初期化
// ---------------------------------------------------------------
     PORTH.PODR.BYTE = 0x00;         // ポートH 出力データ初期値 = 0
     PORTH.PDR.BYTE  = 0xFF;         // ポートH 各ビットの入出力指定( 1=出力 )

     setup_interval_timer();         // インターバルタイマー起動
     setup_sci1( 9, 0 );             // SCI1 オープン処理 ( 57600 b/s )
     init_adc();                     // 12bit A/Dコンバータ初期化 処理

     enable_irq();                   // CPU I Flag = 1  割り込みを許可する
}
```