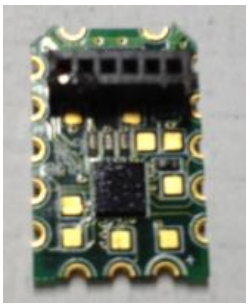
# YRPBRL78G11 のターゲット・デバイス部の準備

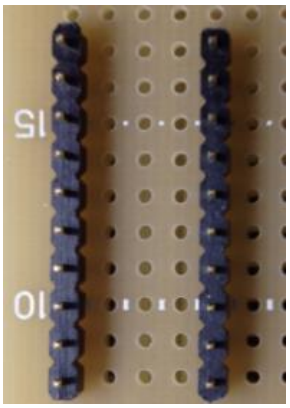


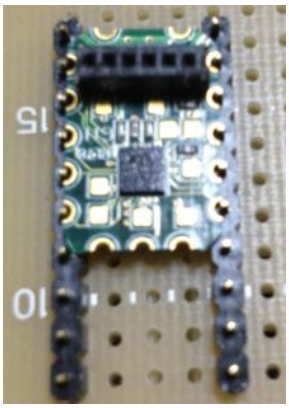
YRPBRL78G11 のターゲット・デバイス部はすごく小さいので，このままでは取り扱うのに不便です。ブレッドボードで使えるように DIP に変換しようと思います。

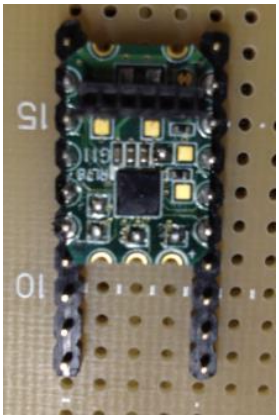
右の写真では，上がバッテリーボードと切り離した方です。その後の作業の関係で，角を削っています。この後，下側も同様に角を削ります。（ここでは，25pin のものを使っています。）

## １．DIP への変換方法

DIP の端子としては，秋月電子の細ピンヘッダー（PHA-1x40SG 等）を 10pin に切ったものを 2 つ準備して，ユニバーサル基板に 2.54mm で 4 つ分の間隔（400mil）に並べます。そこに，下の真ん中の写真のようにボードをセットします。下の右の写真では，分かりにくいかもしれませんが，使用する 5 つの PAD（TP1～TP5）に半田を盛っています（使用しない PAD は半田を盛っていませんので，金色のままです。金色の PAD は 4 つしか見えなくなっています）。





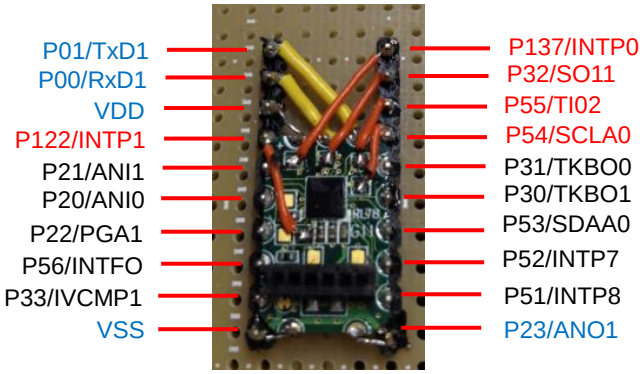


## ２．　引き出す信号

25 ピンの RL78/G11（R5F1058A）のデバイス部分で，DIP に対応した信号が 10 本で，それ以外に残りの 2 辺に 5 本が出ています。ボード上に TP1～TP9 として PAD が 9 つあります。

TP9（EVDD）は，ボード上のショート PAD で VDD と接続されているので，使用する必要はありません。TP8（P1235/RESET#）と TP7（P40/TOOL0）については，デバッグで使用するのを優先にするので，PAD は使用しません。残った 6 つの PAD の中から 5 つを信号として外部に引き出すと，20 ピンになります。そこで，X1 発振回路以外の兼用機能がなく，入力専用の TP6（P121）は，使用しないことにします。

ここまでの検討結果から，DIP で外部に引き出す信号を次のように決めます。ここで，信号名が赤字なのはボード上面の PAD から引き出した信号で，青字なのは，DIP 以外の上下の辺から引き出した信号です。




24pin（EVDD の有無が相違点のようです）でも同じ信号配置にできるようなので，これを共通の信号配置にします。さすがに，20pin の SSOP を DIP に変換したもの（北斗電子の変換ボード：HSBRL78G11-20-PT）とは同じ信号配置にはできません。20pin の場合には，この変換ボードと E2OB 部は電源関係，デバッグ関係（TOOL0 と RESET#）で接続するだけなので，それほど気にする必要はありません。

この状態にする最初の手順は，4 隅のカットですが，その際には削り過ぎないようにしてください（基板を光にかざすと，内層の電源やグランドがどこまであるかが分かるはずです）。もし，自信がなければ，ボードの隅と干渉する細ピンヘッダーのピンの方を抜きます（その代わり，細ピンヘッダーは 10pin ではなく 12pin にすることになります）。

最初は，ボードを固定するために，1 ページ中ほどの写真で示したものをそのままブレッドボードに刺して固定します。その後，2 辺の信号部分を細ピンヘッダーに半田付けします。その後，残った辺の信号を接続します。最後は，ボード上の PAD 部を P122，P54，P55，P32，P137 の順に接続します。PAD 部の半田付けは，焦らず，それでもできるだけ素早くやってください。

## ３．　YRPBRL78G11 を使う準備作業（その１）

それでは，このページの最初の写真のような変換ボード（？）ができたので，その確認を行います。接続した信号は基本的にポートだけなので，E2OB 部と接続すると，デバッガは動作するはずです。デバッガが正常に起動できない場合には，電源関係をショートさせた可能性があります。ボードの 4 隅を中心に拡大鏡を使って，確認してください。（私は，今回の作業で太陽電機産業（goot）社製の ST-93 という作業台を使いました。直径 90mm で 3 倍の拡大鏡が付いています。）

問題は，今回の配線が正しくできたかをどうやってチェックするかです。この後の 24pin のもののチェックと合わせて，ブレッドボードを使って LED を並べてみました。

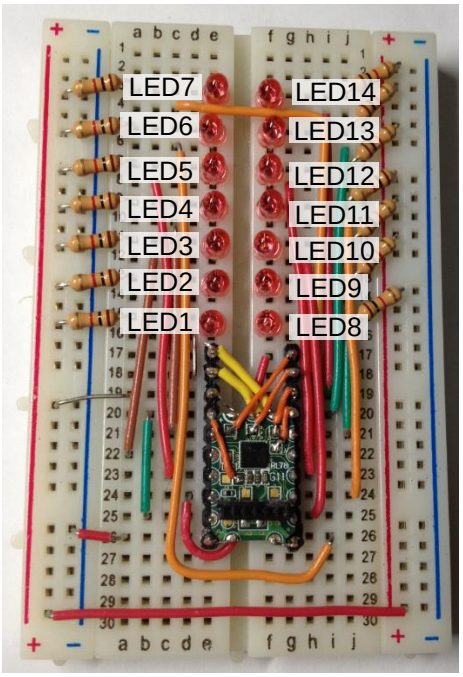
LED を出力ポートの分（16 個）並べてもよかったのですが，それだと，2 本の入力専用ポートが確認できないので，14 個の LED を並べて，2 本の入力専用ポートは残り 2 本の出力ポートに接続することにしました。

もちろん，14 本の信号についても 7 本を出力，7 本を入力に設定して，出力と入力を接続する方が簡単ではあります。しかし，見ていて面白くないので，ここは 14 個の LED を並べてみました。

そこで作成したのが右の写真に示す回路（ボード）です。左側の 7 個と右側の 7 個の合計 14 個の LED を並べてみました。

信号の引き回しは，できるだけクロスしないようにしたかったのですが，左側に電源とグランドがあるためと，最初にポートの順に配置するようなプログラムを作ってしまっていたので，3 本ほど配線がクロスしてしまいました。一番上と下のオレンジの線がクロスしている信号です。残り 1 本はボードの下をくぐっている赤い線です。

ここらは，チェックが完了したら，修正するつもりです。



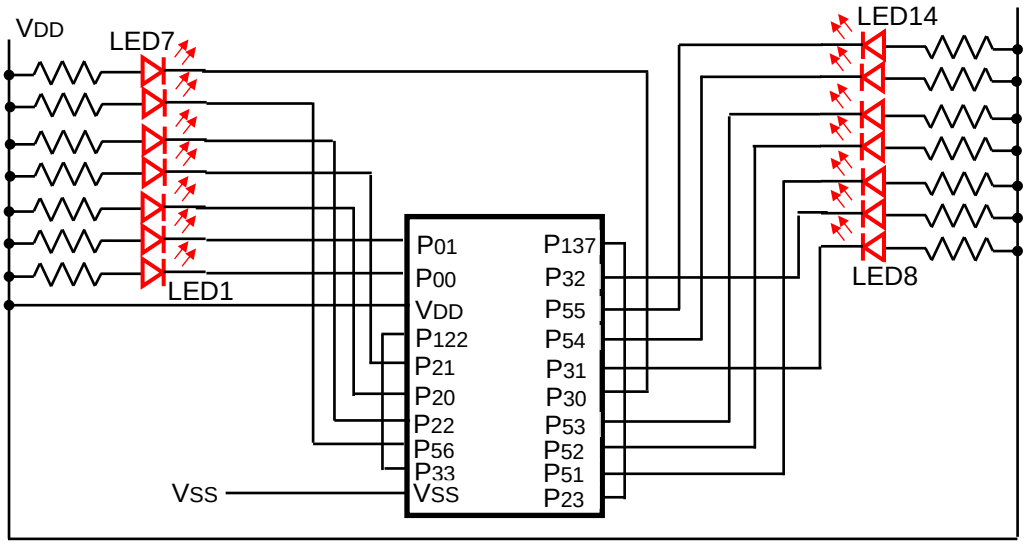

最終的な結線を以下に示します。

| 信号名 | 接続先 | 信号名 | 接続先 |
|---|---|---|---|
| P01/TxD1 | LED2 | P137/INTP0 | P23/ANO1 |
| P00/RxD1 | LED1 | P32/SO11 | LED9 |
| VDD | 電源 | P55/TI02 | LED14 |
| P122/INTP1 | P33/IVCMP1 | P54/SCLA0 | LED13 |
| P21/ANI1 | LED4 | P31/TKBO0 | LED8 |
| P20/ANI0 | LED3 | P30/TKBO1 | LED7 |
| P22/PGA1 | LED5 | P53/SDAA0 | LED12 |
| P56/INTFO | LED6 | P52/INTP7 | LED11 |
| P33/IVCMP1 | P122/INTP1 | P51/INTP8 | LED10 |
| VSS | グランド | P23/ANO1 | P137/INTP0 |

入力専用ポートの確認は，最初は E2OB の「実行中のメモリ・アクセス」を使って確認したのですが，LED と画面の両方を眺めるのは非効率的なので，最終的には，LED7 については，

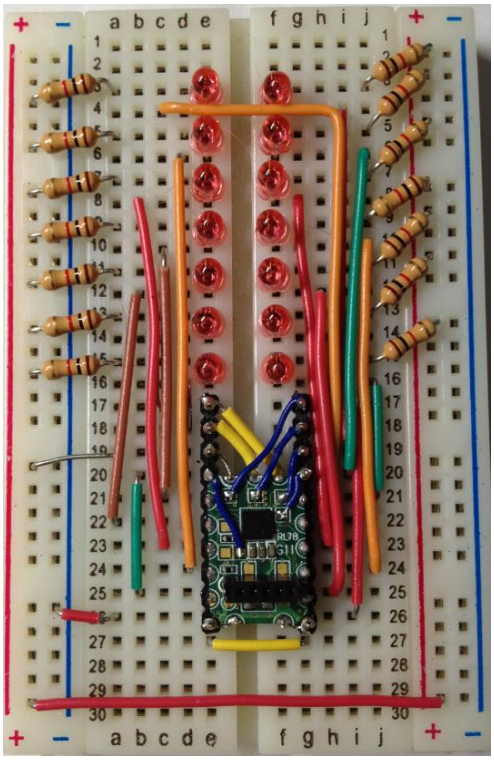
データを一旦 P33 に出力し，接続されている P122 から入力します。入力たその値を P30 から出力しています。同様に，LED14 も P23 に一旦出力したデータを P137 から入力して，P55 に出力しています。

この状態の回路図（ボードの動作確認回路）を以下に示します。




次の写真が，最終的な回路を組み上げたものです。LED7 をドライブする信号だけが，左右をまたがっています。ターゲット・ボードの直ぐ下の黄色い配線は単にボードをセットする位置を示すためのものです。

右の写真では，いつも使っているダイナミック点灯時のように，LED の電流制限抵抗は 1k Ω を使用してしまいました。RL78 の P2x 端子は A/D コンバータ機能を最優先にしたためか，他のポートに比べてドライブ能力が 0.4mA [注]と小さくなっています。しかし，RL78/G11 の公開されている出力特性（IOLvs VOL）のデータ（RL78/G11 のページのドキュメントの中に「特性データ」として公開されている[注]）の P20 端子のページを見ると，それ以上流せるようです[注]。実際に他のポートと同じように LED は点灯していました。

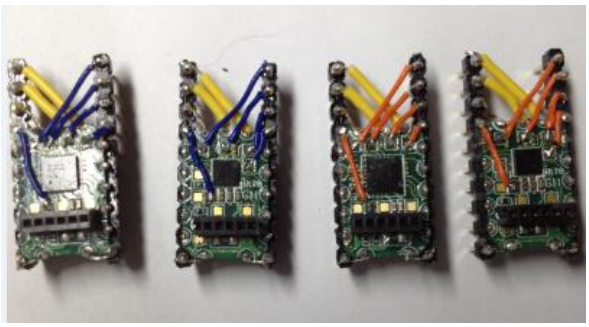


**注：詳細は PAGE8 を参照してください。**

ただし，それでは，規格オーバとなるので，最終的に抵抗値を 3.3kΩに変更しました。

今回は，スタティック点灯で，しかも単なる確認用なので，P2x 端子の規格内の電流でも十分確認できるので大丈夫です。

また，ターゲット・ボードの配線の色が 2 ページの写真と異なるのは，ボードが異なるからです。最初に，端子の間隔が通常の DIP の間隔（300MIL）と同じと思いこんでいて，その間隔で作成しようとしたせいで，何枚か失敗しましたが，今回は，400MIL 間隔で 24pin と 25pin を各々2 枚完成させています（下の写真参照）。



今回使用した部品は，秋月電子で入手できます。

・ブレッドボードは，「ブレッドボード　EIC-801」の品名のものを使用しています。

・ジャンパーワイヤは「ブレッドボード・ジャンパーワイヤ　14 種類×10 本」

・細ピンヘッダーは（細ピンヘッダー　1×40　アソートパック　（10 本入））です。

・LED は 100 個パックで購入したものです。今は新しいものに変わっているようです。ネットで見ると，「3mm 赤色 LED　70０°　OSR5JA3Z74A　（100 個入）」が近そうです。



・

抵抗は，1/4W 型のカーボン抵抗（5%精度）を 100 個パックで購入していたものを使っています。

## ４．　YRPBRL78G11 を使う準備作業（その２）

この回路を使って，ターゲット・ボードの動作確認するプログラムは，16 ビット・タイマ TAU0 のチャネル 0（TM00）で 100ms のインターバル割り込み（INTTM00）を発生させ，割り込み処理で 8bit の変数（counter）をカウントして，LED の点灯を制御しています。

変数 counter の bit0 が 1 なら LED1 を点灯，bit1 で LED2 を～bit6 で LED7 を点灯させます。右側の LED8～LED14 は，少しやり方を変えて，（ビットの対応を逆にして，）bit0 を LED14 に対応～bit6 を LED8 に対応させることにします。さらに，変数 counter の bit7 が 0 なら単純に右と左の LED が上下反転した状態になるだけですが，bit7 が 1 ならば，右側の LED は 1 で点灯し，0 で消灯するようにします（これで，カウント・ダウンするようになります）。

これを実現する INTTM00 割り込み処理プログラムを以下に示します。ここで，counter は割り込みの回数をカウントするグローバル変数，work が LED1～LED7 を制御するためのローカル変数，work2 が LED8～LED14 を制御するためのローカル変数です。

```
/***********************************************************************************************
* Function Name: r_tau0_channel0_interrupt
* Description  : This function INTTM00 interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************************************/
static void __near r_tau0_channel0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    uint8_t work;                       /* LED1～LED7の点灯制御用       */
    uint8_t work2;                      /* LED8～LED14の点灯制御用      */

    work = counter ^ 0xFF;              /* LEDが負論理（0で点灯）の為   */
    if ( ( counter & 0x80 ) )
    {
        work2 = counter;
    }
    else
    {
        work2 = counter ^ 0xFF;
    }

    P0_bit.no0 = ( work & 0x0001 ); work = ( work >> 1 );       /* LED1制御     */
    P0_bit.no1 = ( work & 0x0001 ); work = ( work >> 1 );       /* LED2制御     */
    P2_bit.no0 = ( work & 0x0001 ); work = ( work >> 1 );       /* LED3制御     */
    P2_bit.no1 = ( work & 0x0001 ); work = ( work >> 1 );       /* LED4制御     */
    P2_bit.no2 = ( work & 0x0001 ); work = ( work >> 1 );       /* LED5制御     */
    P5_bit.no6 = ( work & 0x0001 ); work = ( work >> 1 );       /* LED6制御     */
    P3_bit.no3 = ( work & 0x0001 ); work = ( work >> 1 );       /* LED7制御     */

    P2_bit.no3 = ( work2 & 0x0001 ); work2 = ( work2 >> 1 );    /* LED14制御    */
    P5_bit.no4 = ( work2 & 0x0001 ); work2 = ( work2 >> 1 );    /* LED13制御    */
    P5_bit.no3 = ( work2 & 0x0001 ); work2 = ( work2 >> 1 );    /* LED12制御    */
    P5_bit.no2 = ( work2 & 0x0001 ); work2 = ( work2 >> 1 );    /* LED11制御    */
    P5_bit.no1 = ( work2 & 0x0001 ); work2 = ( work2 >> 1 );    /* LED10制御    */
    P3_bit.no2 = ( work2 & 0x0001 ); work2 = ( work2 >> 1 );    /* LED9制御     */
    P3_bit.no1 = ( work2 & 0x0001 ); work2 = ( work2 >> 1 );    /* LED8制御     */


    P3_bit.no0 = P12_bit.no2;
    P5_bit.no5 = P13_bit.no7;

    counter++;

    /* End user code. Do not edit comment generated here */
```

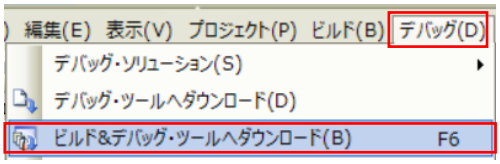
ここで，上側の２つの赤い四角で囲んだ部分は入力ポートを介してデータを入力するためにポートに出力しているところです。一番下が入力ポートを介して入力したデータで実際に LED をドライブしている部分です。

コード生成では，TM00 を 100ms のインターバル・タイマに設定する以外に，P40 以外の出力に設定できるポートを出力（データの初期値は 1）に設定しておきます。あとは，ウォッチドッグ・タイマ（WDT）を停止に，電圧検出回路を 2.75V でのリセットに設定してコードを生成しておきます（この 2 つは，プログラムではなく，オプション・バイトに反映されます）。

生成された「r_cg_tau_user.c」の「r_tau0_channel0_interrup()」関数に上のプログラムを書き込みます。後は，「r_cg_main.c」に以下のように部分を追加するだけです。

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        HALT();                         /* wait for INTTM00(100ms)      */
    }
    /* End user code. Do not edit comment generated here */
}
/***********************************************************************************************
* Function Name: R_MAIN_UserInit
* Description  : This function adds user code before implementing main function.
* Arguments    : None
* Return Value : None
***********************************************************************************************/
static void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    R_TAU0_Channel0_Start();            /* start 100ms interval timer   */
    EI();
    /* End user code. Do not edit comment generated here */
}
```

これで，「デバッグ(D)」メニューの「ビルド＆デバッグ・ツールへダウンロード(B)」を選択して，プロジェクトをビルドしてデバッガにダウンロードします。




ダウンロードが完了すると，以下のような画面になります。本来は，main 関数の先頭でブレークが掛かるはずなのですが，CC-RL の最適化により，main 関数の先頭の関数呼び出しが省略されたために，R_MAIN_UserInit 関数の先頭で停止しています。

```
                void main(void)
                {
                    R_MAIN_UserInit();
                    /* Start user code. Do not edit comment generated here */
                    while (1U)
                    {
0022a               HALT();                         /* wait for INTTM00(100ms)      */
                    }
                    /* End user code. Do not edit comment generated here */
                }
                /***********************************************************************************************
                * Function Name: R_MAIN_UserInit
                * Description  : This function adds user code before implementing main function.
                * Arguments    : None
                * Return Value : None
                ***********************************************************************************************/
                static void R_MAIN_UserInit(void)
                {
                    /* Start user code. Do not edit comment generated here */
00223               R_TAU0_Channel0_Start();            /* start 100ms interval timer   */
00227               EI();
                    /* End user code. Do not edit comment generated here */
                }
```

これで，  をクリックして実行を開始すると，LED が点灯していきます。配線に問題なければ，全ての LED が規則正しく点滅を繰り返します。これで，ポートと VDD の配線がチェックできますが，VSS はチェックできません。そこで，E2OB での実行を終わり，E2OB を外して，3V の電源（単 3 電池 2 本）で動作させます。これで，VSS の配線もチェックできます。

これで準備完了なので，次回から実際のプログラムになります。

## ５．　備考

RL78/G11 のハードウェア マニュアル（Renesas の HP からダウンロード可能）は下記 URL で確認できます。Rev2.00 が 2018.1 に公開されているようです。

「第 35 章　電気的特性（TA＝ －40～＋85°C）」の「35.3 DC 特性」に「35.3.1 端子特性」が掲載されています。この「ロウ・レベル出力電流」の「IOL2」の項目は，0.4mA（MAX）と他の出力ポートより小さなドライブ能力になっています。

https://www.renesas.com/ja-jp/doc/products/mpumcu/doc/rl78/r01uh0637jj0200-rl78g11.pdf?key=49d00f35371df08a21a9277c07fff37f

RL78 の公開されている出力特性（IOLvs VOL）は，以前と見せ方が変わってしまいました。RL78/G11 の場合は，RL78/G11 のページ（下記 URL）の「ドキュメント」を選択します。

https://www.renesas.com/ja-jp/products/microcontrollers-microprocessors/rl78/rl78g1x/rl78g11.html


「ドキュメント」を選択すると次ページに示すような画面となります。

この画面で「ドキュメントタイプ」に示されたドキュメントの種類の一覧に「特性データ」の項目があるので，「特性データ」をチェックして，その模擬したの「検索」ボタンをクリックすると，3 つのデータが表示されます。


| タイトル | 発行日<br>リビジョン | ドキュメントナンバー | サイズ<br>(KB) | 関連<br>製品 |
|---|---|---|---|---|
| RL78G11:IDD-VDD特性(電源電流) | Oct.18.16 | MR20161014-1 | 1107 | RL78/G11 |
| RL78/G11出力特性: IOH VS VOH | Oct.13.16 | MR20161006-1 | 246 | RL78/G11 |
| RL78/G11出力特性: IOL VS VOL | Oct.13.16 | MR20161006-2 | 262 | RL78/G11 |

今回は，一番下の「RL78/G11 出力特性 IOL vs VOL」が該当するドキュメントになります。温度と対象のポートで，全部で 24 ページ分のデータがありました。P20 端子の Ta = 25°C での条件の特性曲線を見ると 0.4mA で 0.05V までほぼ直線的に伸びています。まだまだ，流せそうな感じです。

以上