初心者のための RL78 入門コース(第2回:ポート出力例その1)

第2回の今回は実際に簡単なプログラムを作成して動作確認を行います。最初は、シミュレ ータで動作させます。ついでに、E1を使って BlueBoard-RL78/G13\_64pin で動作確認を行い ます。

今回の内容

1

6. コード生成を利用した実際のプログラム作成	,	•	•	•	•	•	•	•	•	•	•	•	P19
7. BlueBoard-RL78/G13_64pin での動作確認	•	•	•		•	•	•	•	•	•	•	•	P34

次回(第3回)は、以下の内容を予定しています

- 8. コード生成を利用した実際のプログラム作成(その2) ・・・・・ P40
- 9. コード生成を利用したプログラム作成 (ポート入力)・・・・・・ P47

ここまでで使用したプロジェクトは以下のフォルダに格納されています。

ポート出力フォルダの構成↓ 第1回分↓ +- RL78\_G13\_PORT共通部 --- プロジェクトを作成しただけ↓ 注:ビルドはしていませんが、クイック・ビルドの結果が残っています。↓ ↓ 第2回分↓ +- RL78\_G13\_PORT1 --- ポートでのLED制御のみ↓ +- RL78\_G13\_PORT1\_E1 --- ポートでのLED制御のみ(E1用)↓ +- RL78\_G13\_PORT1\_2 --- ソフトタイマでのLEDチカチカ↓ +- RL78\_G13\_PORT1\_2\_E1 --- ソフトタイマでのLEDチカチカ↓ +- RL78\_G13\_PORT1\_3 --- インターバル・タイマでのLEDチカチカ↓ # \$\$3回分↓ +- RL78\_G13\_PORT1\_3\_E1 --- インターバル・タイマでのLEDチカチカ↓ +- RL78\_G13\_PORT1\_3\_E1 --- インターバル・タイマでのLEDチカチカ↓ +- RL78\_G13\_PORT1\_4 --- タイマの方形波出力によるLEDチカチカ↓ +- RL78\_G13\_PORT1\_4\_E1 --- タイマの方形波出力によるLEDチカチカ↓(E1用)↓

P. 18

6. コード生成を利用した実際のプログラム作成

プログラムの例として,最初に環境に慣れることもかねて,LEDの点滅をやってみようと思います。まずは、P31に接続したLEDを点灯させることから始めます。最初に少しハードウェアについて触れます。

ここで制御対象の LED は右図に示すよう に接続されているものとします。このよう に接続するのは、ポートのドライブ能力は ロウ側がハイ側に比べて大きいためです。 ここで、R は LED に流れる電流を制限する ための電流制限抵抗です。この回路構成で は、P31 に 0 を設定すれば LED は点灯し、 1 を設定すれば消灯します。



ポートで LED 等の負荷をドライブするとき、負荷を動作させるための電流を流すことが できれば十分です。ロウでドライブしてもハイでドライブしても LED を目標の明るさで点 灯できれば構いません。 RL78/G13の通常のポートではハイ・レベルで流せる電流は最大 10mA で、ロウ・レベ ルでは 20mA です。 LED の定格が 20mA 程度と書かれていることがありますが、実際にそんな電流を流すと まぶしくてたまりません。実際はその半分程度でも十分な明るさになることがほとんどで す。効率が高い(高輝度タイプ)LED では 1mA でも分かります。使用する LED と目的に 合った電流にします。 通常, LED の順方向電圧(Vf)は 1.5V(赤色 LED)から 3V 程度(白色系)まであります。そこで、LED をドライブするための電源の電圧から LED の順方向電圧とポートでの 電圧降下分を引いた電圧と流したい電流から電流制限抵抗の値を決めます。 RL78/G13 のロウ・レベル出力電圧のスペックを見ると電源電圧 4V 以上で 20mA 流すと 1.3V (MAX) と記載されています。ところが HP で IOL と VOL の特性曲線を見ると 20mA 流しても電源電圧が 5V では 0.4V 程度で、電源電圧 4V 辺りの電圧降下を求めると 0.5V 強 となり、スペックの半分もありません。電源電圧が 5V のときに真面目にスペックで計算 した抵抗値を使うと、1.5倍も電流が流れることになります。

電流の流し過ぎに注意しましょう。

注: RL78/G13 の特性曲線は以下の URL にあります。今回はこの中の RL78/G13:VOL-IOL 特 性を参照しました。

https://www.renesas.com/ja-jp/search/keyword-

search.html#genre=document&documenttype=544&productlayer=115082

ハードウェアについては別途説明する予定です。

6.1 プロジェクト(RL78\_G13\_PORT1)を開きます。

「RL78\_G13\_PORT1」フォルダを開き、「RL78\_G13\_PORT1..mtpj」ファイルをダブル・ク リックしてプロジェクトを開きます。



6.2 コード生成でポート3を開きます。

初期状態では、全て「使用しない」になっているので、下に示すように P31 を「出力」に変更し、右側の「1」にチェックを入れます。



これは、P31を出カポートに設定し、出力するデータの初期値を1(初期状態で、LEDを消灯)にすることを示します。

今回のコード生成する内容はこれだけです。これで、「コード生成(G)」をクリックします。

I	四フロパティ 🏭 コー	ド生成* 🏹	main.c		
	🔀 端子配置へ反映	S-F4	主成(G)	🛓 💷 💉 🎜 💁 🔞 🖉 🗐 🌒 🐠	å i
	#−ト0 #−ト <mark>⊐−</mark> Ĕ	生成(G)	월 #-ト4	ポート5 ポート6 ポート7 ポート12 ポート13	ポート14
	- P30 ● 使用しない	<ul><li>○ 入力</li></ul>	〇出力	🗌 内蔵ブルアップ	1
	-P31 〇 使用しない	〇入力	• 出力	□ 内蔵ブルアップ	<b>v</b> 1

## すると、右下の出力ウィンドウに、結果が表示されます。

M0409001:ファイルを生成します:」 M0403001:ファイルを主成します。」 M04090000:r\_main.cを生成しました。」 M04090000:r\_systeminit.cを生成しました。」 M04090000:r\_cs\_macrodriver.hを生成しました。」 M0409000:r\_cg\_userdefine.hを生成しました。↓ MO409000:r\_cg\_cgc.cを生成しました。| M0409000:r\_cg\_cgc\_user.cを生成しました。J M0409000:r\_cg\_cgc.hを生成しました。J M0409000:r\_cg\_port.cを生成しました。」 M0409000:r\_cg\_port\_user.cを生成しました。」 M0409000:r\_cg\_port.hを生成しました。↓ M0409003:ファイルの生成を完了しました。」 M0409009:ウォッチドッグ・タイマの設定は、 M0409012:電圧検出回路の設定は、使用する、 使用しない'です。」 です。 M0409014:オンチッブ・デバッグ機能の設定は'使用する'です。↓

これで、コード生成が関係するファイルを生成しました。 右に示したのがコード生成で生成したファイルです。 r\_cg\_cgc.c 💐 r\_cg\_cgc.h ・「r\_cg\_cgc」で始まるのはクロックの設定です。 💐 r\_cg\_cgc\_user.c 「r cq port」で始まるのはポートの設定です。 r\_cg\_macrodriver.h ・「r\_main.c」が main 関数を含むファイルです。 r\_cg\_port.c 「r systeminit.c」はスタートアップ・ルーチンから 💐 r\_cg\_port.h 呼び出される hdwinit 関数を含んでおり、内蔵周辺 💐 r\_cg\_port\_user.c 機能の初期化を取りまとめています。 r\_cg\_userdefine.h 💐 r\_main.c r\_systeminit.c

6.3 生成された設定の内容

(1) ポートの初期設定

今回は P31 の設定だけなので,ポートの初期設定は以下のように簡単な R\_PORT\_Create 関数が生成されています。

```
void R_PORT_Create(void)
{
     P3 = _02_Pn1_OUTPUT_1;
     PM3 = _01_PMn0_NOT_USE | _00_PMn1_MODE_OUTPUT | _FC_PM3_DEFAULT;
}
```

P31を出カポートとして使うための設定は P3(出カラッチ)レジスタと PM3(ポート・モード)レジスタだけです。今回使用する 64 ピンの R5F100LE ではポート 3 は P30 と P31 の 2 本しかありません。P3 レジスタに 2 を設定しているのは P31 の初期値を 1 と指定したことに対応しています。PM3 レジスタのビット 1 が 0 になっているのが,P31を出力ポートに設定したことに対応しています。

(2) r\_main.c

このファイルには, main 関数と, R\_MAIN\_UserInit 関数の(実際には入れ物だけですが)2つだけです。

R\_MAIN\_UserInit 関数は初期設定が済んだ内蔵周辺機能を実際に起動(動作可能に)するために準備された関数で, main 関数は最初にこの R\_MAIN\_UserInit 関数を呼び出します。実際に生成された R\_MAIN\_UserInit 関数は以下のようになっています。

ここで、コード生成を使用する際の基本的なルールを示します。コード生成された関数には 下で朱書きで表示した2つのコメント行があり、プログラムを書く場合には必ずこの間に書い てください。もちろん、この範囲以外にも書くことは可能ですが、この範囲外に記述したプロ グラムは次にコード生成したときに消されてしまいます。これは、すべてのコード生成で生成 されたプログラムやヘッダファイルで共通です。

void R\_MAIN\_UserInit(void)

Ł

}

/\* Start user code. Do not edit comment generated here \*/ El(); /\* End user code. Do not edit comment generated here \*/

コード生成で生成された main 関数は,以下のようになっています。基本的には while 文の中 にプログラムを記述します。

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        ;
        }
        /* End user code. Do not edit comment generated here */
}
```

6.4 その他の項目

生成された R\_MAIN\_UserInit 関数の中に"El();"と言う記述がありますが, これは RL78 独自の機能拡張です。

これは、CPU を割り込み(ベクタ割り込み)制御するものです。これ含め、同じような機能 拡張として覚えておいて欲しいのが以下の5つです。これら5つは、組み込み用途向けに追加 されたものです。CC-RL で RL78 のプログラムを作成するのに必須だと考えてください。

- ・NOP(); RL78の NOP 命令を指定します。
- El(); 割り込み許可(RL78 の El 命令と同じ)を指定します。
- DI(); "EI();"と逆に割り込みの禁止(RL78のDI命令と同じ)を指定します。
- ・HALT(); CPU を停止させる RL78 の HALT 命令を指定します。
- ・STOP(); RL78 のクロック発振を停止させる STOP 命令を指定します。

6.5 ポートの制御プログラム

いよいよ、プログラムを書いてみます。

P31 に接続した LED を点灯させるプログラムです。LED を点灯させるには、P31 を 0 にします。このためには、P3\_bit.no1 に 0 を代入します(逆に、LED を消灯させるには、P3\_bit.no1 に 1 を代入します。)

これを、実際に main 関数に書き込みます。下に実際に書き込んだプログラムを示します。

void main(void)			
R_MAIN_UserInit(); /* Start user code. Do not edit comme while (1U)	nt g	enerated here */	
i P3_bit.no1 = 0; /*	* P3	.1をO(ロウ出力)にする */	
P3_bit.no1 = 1; /* NOP();	* P3	.1を1(ハイ出力)にする */	
} /* End user code. Do not edit comment	gen	erated here */	
.}		■『 ファイル(F) 編集(E) 表示(V) プロジェ	クト(P) ビルド(B)
これをビルドしてみます。	G1	ビルド・プロジェクト(B)	F7
ビルドは 右に示すように メニューバ		リビルド・プロジェクト(R)	Shift+F7
		クリーン・プロジェクト(C)	
ーの「ビルド (B)」をクリックしてメニ	TON	ラピッド・ビルド(A)	
ューを表示させ, 「ビルド・プロジェクト		依存関係の更新(P)	
(B)」を選択します。	Ð	RL78_G13_PORT1 をビルド(U)	
ビルド結果は出力ウィンドウで確認しま	L)	RL78_G13_PORT1 をリビルド(E)	
		RL78_G13_PORT1 をクリーン(L)	
<b>ग</b> ्		RL78_G13_PORT1 の依存関係の更新(D)	
	Ink	RL78_G13_PORT1 のリンク順を設定する(K)	
	-21	RL78_G13_PORT1 用に最適化性能比較ツー」	レを表示(Z)
	*	ビルドを中止(S)	Ctrl+F7
	Ty	ビルド・モードの設定(M)	
	T]	バッチ・ビルド(T)	
	Ty	ビルド・オプション一覧(0)	

P. 23

結果を確認するために, デバッグ・ツールに結果をダウンロードします。下に示すようにメ ニューバーの「デバッグ (D)」をクリックしてメニューを開き, 「デバッグ・ツールへダウン ロード (D)」を選択してください。



ダウンロードが完了すると、シミュレータの画面となり、下に示すようにダウンロードした プログラムの main 関数のソースが表示されます。

0		RL78_G13_PORT1 - RL78 シミュレータ - CS+ for CC - [r_main.c]	
		🐼 🐼 DefaultBuild 🔹 🔨 🖏 🗣	I III 🕑 🕑 🐂 🚳
<b>Q</b> , 7	(9-h(S) 🔒 🖶 🎒 🐰 🐚 🖄		
ファイル	(F) 編集(E) 表示(V) プロジェクト(P) ビルド	(B) デバッグ(D) ツール(T) ウインドウ(W) ヘルプ(H)	
	🗩 🔍 🔍 🗗 🚺 ソリューション一覧(S)		
אין אויד ב־ביא - דיד 🙀	ジェクト・ツリー 4 > ③ 2 ◎ ● シリアル ● A/Dコン/(-タ ● タイマ ● サイマ ● リアルタイム・クロック	<sup>●</sup> プロパティ * 逆アセンプル1 (「 r_mainc ・ ×     ・ ×     ・ ・     ・・      ・・      ・     ・     ・      ・      ・      ・      ・      ・      ・      ・      ・     ・	ローカル変数 2 表記(N)・ の わいント 名前
	<ul> <li>● 12とット・インターバル・タイマ</li> <li>● クロック出力/ブザー出力</li> <li>● DMAコントローラ</li> <li>● 配任検出回路</li> <li>− &lt; CC-RL (どルド・ツール)</li> <li>→ RL78 シミュレック (デパッグ・ツール)</li> <li>→ AL78 シミュレック (デパッグ・ツール)</li> <li>→ 3 ファイル</li> </ul>	B0         0016f         中         R_MAIN_UserInit();           61         /* Startuser code. Do not edit comment generated here */           62         **nite (U)           63         {           64         00172           65         00175           66         00176           67         00178           67         00179	

60 行目に黄色の右向きの矢印があり、黄色になったこの行が次に実行する行で、main 関数の 最初で、関数 R\_MAIN\_UserInit を呼び出しているところになります。

ここで、プログラムを実行する前に、シミュレータの設定を行います。このためにはタスク バーにある CS+のタスクを示す 2 をクリックすると、下のように2つのタスクが表示され るので、右側のシミュレータを選択します。



RL78/G13のシミュレータは命令だけでなく、内蔵された周辺機能の一部と外部素子もシミュレートできます。最初にその設定を行います。

シミュレータ GUI が表示されたら、下に示すようにメニューバーで「シミュレータ (S)」を クリックしてメニューを表示して「入出力パネル (P)」を選択します。

۲				シミュレータGUI
ファイ	<i>ι</i> ν(F)	オプション(0)	シミュレータ(S)	ウインドウ(W) ヘルプ(H)
1		信号データエデ	(夕(E)	2
*		入出カパネル(	P)	
t t		シリアル(S)		
		タイミングチャート	`(T)	<u>\$</u>

すると、下のように「入出カパネル1」が表示されます。

0	シミュレータGUI
ファイル(F) 編集(E) 表示(V) 部品(P) 図形(F) オプション(O) シミュレータ(S) ウインドウ	[W) ヘルプ(H)
🎦 🚅 🖬 🗠 🗠 👗 🛍 🛤 🙎	
<u> 총 표</u> H m	
**	
💻 🔜 🜐 💛 🎽 8. 🏽 💓 😫	
$\begin{smallmatrix} \mathbf{b} \\ \mathbf{c} $	
💆 入出力パネル1 💷 🖭	
^	

次に, メニューバーの「部品 (P)」をクリックして表示されたメニューから「LED (E)」を 選択し, 「入出カパネル 1」で LED を配置する場所をドラッグします。

•	シミュレータGUI
ファイル(F) 編集(E) 表示(V) 部品(P)	図形(F) オプション(O) シミュレータ(S) ウインドウ(W) ヘルプ(H)
ボタン (B)	2
アナログ ボタン(A)	
キーマトリクス (K)	
レベルゲージ(L)	<b>+</b>
LED (E)	2 1 A =
7セクメントLED (S)	
14セグメントLED (D)	
マトリクスLED(T)	
ブザー (Z)	
プルアップ/プルダウン設定(W)	

ドラッグした範囲で LED を示す部品が表示されます。



この部品を右クリックしてメニューを表示します。表示されたメニューの一番下にある「プロパティ(R)」を選択します。

	入出力パ	ネル1
)	切り取り(T) コピー(C) 貼り付け(P) 削除(D)	
	グループ化(G) 順序(O)	*
	プロパティ(R)	

下に示すようなプロパティ画面が表示されるので、「ラベル」に名前を入力します。

 Parts Led Properties
LED端子接続 スタイル
5/VL:
接続端子: 🗸
アクティブレベル: OLOW ⑧ HIGH

ここでは、「LED1」とでもしておきます。次に、「接続端子」の欄の右側のメニューをクリックしてリストを表示し、接続する端子として「P31/・・・・」を指定します。

	Parts Led Properties
LED端子接続	スタイル
5∧JL:	LED1
接続端子:	
アクティブレベ	P31/T103/T003/INTP2 P40/TOOL0 P41/T107/T007 P42/T104/T004

最後に、「アクティブレベル」を「LOW」にして、「OK」をクリックします。

Parts Led Properties	×
LED端子接続 スタイル	
∃rkjµ: LED1	
接続端子: P31/TI03/TO03/INTP· ▼	
アクティブレペル: ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・	
OK         キャンセル         週用(A)         ヘ	ルプ

これで、設定が完了し、下のように部品の周りの枠がなくなります。



シミュレータ GUI での設定が完了したので、プログラムに戻ります。そこで、下の図の 65 行目の2つの赤い枠で囲まれた部分の交わったところをクリックすると、赤っぽい色が NOP(); の行につきます。これはブレーク・ポイントが設定されたことを示し、そこでプログラムの実 行を停止させることができます。プログラムに「NOP();」の行を入れたのは、このようにブレ ーク・ポイントを設定するためです。



同様に、下側の「NOP();」にもブレーク・ポイントを設定します。

シミュレータ画面の右上に 🚮 💁 🐂 🔘 🕞 🕞 🐂 🞯 😒 🗊 📬 👬 があります が、これがシミュレータのコマンドボタンです。このボタンを使ってプログラムの実行を制御 します。

グリーン右向きの三角にマウスを合わせると,説明が表示されます。このボタンをクリック することで,ブレーク・ポイントを有効にして,プログラムが実行されます。



このボタンをクリックして、プログラムを実行させます。

この実行結果が、下に示す画面イメージとなります。ここでは、シミュレータ GUI が右側に 見えるようにウィンドウを配置しているので、実行結果とプログラムを同時に見ることができ ます。左側の GUI 画面で LED が点灯しているのが分かるかと思います(LED1の文字がはっき りと見えています)。プログラムウィンドウでは、上側の「NOP();」の行が黄色になり、そこ で実行が停止していることが分かります。つまり、その上の「P3\_bit.no1 = 0;」が実行されて LED が点灯したことになります。



再度,グリーン右向きの三角のボタンをクリックすると、下のように、下側の「NOP();」の 行が黄色になり、その上の「P3\_bit.no1 = 1;」が実行された結果 LED が消灯したことになりま す。



それでは、グリーンで白抜きの右向きの三角のボタンをクリックしてみましょう。これはブ レーク・ポイントを無効にして実行するボタンです。

すると、シミュレータ GUI ウィンドウが前面に表示され、LED がチカチカ点滅しているのが 分かります。これは、決して LED チカチカの状態ではなく、単にシミュレータの実行速度が遅 いためにこのように目に見える点滅が発生しているだけです。実際の RL78/G13 で実行する と、点滅は全く目に見えませんので、次は目に見えるように対策します。一応ここまでのプロ ジェクトを保存しておきます。

プログラムを停止させるには、シミュレータのウィンドウを表示させ、右上の赤い四角が表 示されたボタンをクリックします。



シミュレータを終了して、CS+のウィドウに戻ったら、メニューバーの「プロジェクト (P)」のメニューから「名前を付けてプロジェクトを保存(T)」を選択し、ファイル名を指定 して「保存(S)」をクリックして保存します。

	F ファイル(F) 編集(E) 表表	示(V) プロジェクト(P)
7	新しいプロジェクトを作成(N)	
3	プロジェクトを開く(P)	
	お気に入りのプロジェクト(F)	•
	追加(D)	•
72	RL78_G13_PORT1 をアクティブ・プロ	ジェクトに設定(A)
B-// B	依存プロジェクト設定(J)	
-	プロジェクトを閉じる(E)	
	プロジェクトを保存(R)	Ctrl+Shift+S
AB(C)	名前を付けてプロジェクトを保存(T)	
E.	プロジェクトから外す(R)	Shift+Delete
2	品質記録用データの一括出力(Q)	
6	プロジェクトと開発ツールをパックして保存	(K)

ファイル名(N):		~
ファイルの種類(T):	プロジェクト・ファイル (*.mtpj)	<
🍙 フォルダーの非表示	保存(S) キャンセル	]

CS+をクローズします。ここまでの結果を保存しておくために、「RL78\_G13\_PORT1」フォ ルダをコピーして、「RL78\_G13\_PORT1\_2」の名前を付けておきます。次は、この 「RL78\_G13\_PORT1\_2」フォルダの中のプロジェクト「RL78\_G13\_PORT1\_1.mtpj」を使っ ていきます。

6.6 LED の点滅プログラム

それでは、目に見える周期で LED を点滅させます。このために最初に考え付くのはソフトウ ェアでループを作ることです。単純に符号なしの 16 ビットの変数をカウントすることを考え てみます。この場合、1 ループで 10 クロック程度と考えると、32MHz の動作では、約 20ms 程度になります。残念ながらこれでは目に見えません。そこで、32 ビットの変数を使用するこ とになります。RL78 は 16 ビット MCU なので、32 ビットの演算は苦手です。1 ループで倍以 上のクロックが必要になります。ここでは、20 クロックと考えて 200ms の時間待ちを考えて みます。200ms のクロック数は 6,400,000 (=32,000,000 × 0.2) となり、1 ループを 20 クロ ックと考えると 320,000 回ループさせることになります。これでプログラムを追加してみま す。

r\_main.c の最後のユーザ領域に次ページに示すような wait\_200 関数を作成します。

ここで, 32 ビットの変数 time を定義し, for ループで 320,000 から減算していき, 0 になったら抜けるようにします。

```
/* Start user code for adding. Do not edit comment generated here */\downarrow
* Function Name: wait_200\downarrow
* Description : 200ms待つ↓
* Arguments
            : None↓
* Return Value : None
void wait_200(void)↓
{↓
   uint32_t time;↓
   for (\overline{t}ime = 320000 ; time > 0 ; time--)\downarrow
   {↓
      NOP (); \downarrow
   }↓
}↓
/* End user code. Do not edit comment generated here */\downarrow
[E0F]
```

これを呼び出す処理を, main 関数のポートを操作している処理の間に入れます。



実際に、シミュレータで実行する前に、シミュレータの設定を変更します。「RL78 シミュレ ータ (デバッグ・ツール)」を右クリックしてメニューから「プロパティ (P)」を選択し、「デ バッグ・ツール設定」タグを開きます。「タイマ」の項目をひらき、「タイマ機能を使用する」 がディフォルトでは「いいえ」になっているのを「はい」に変更します

	-	
<u>         RL78 G13 PORT1 1 (フロジェクト)*                   </u>	A ¥£U	
■ R5F100LE (マイクロコントローラ)	▷ メモリ・マッピング	[9]
□ 🥄 端子配置 (設計ツール)	▲ 実行中のメモリ・アクセス	
	実行中に表示更新を行う	(3\$)
	表示更新問題[ms]	500
	5 The b	
RI 78 ミンドコレータ (デバッグ・ツール)		
	▲ タイマ	
☆…割 ビルド・ツール生成ファイル	タイマ機能を使用する	いいえ
cstart.asm	▷ カバレッジ	はい
	▲ シミュレータGUI	いいえ
stkinit.asm	7312-2011年表示 45	
iodefine.h	主体明からに見前面主子オス	111
	タイマ機能を使用する	
C r main c	タイマ機能を使用するかどうかを指定します。	
-g r_systeminit.c		
r_cq_cqc.c	按結田設守   デバッガ・ツール設定 / ガーン/	
		L L DY MARKE V DODRETERRE V

これを行うことで、シミュレータで実行時間を測定することが可能になります。 それでは、ビルドして、デバッグ・ツールにダウンロードします。

	ビルド	ダウン	ロード
	二         ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B)           ビルド・プロジェクト(B)         F7	) 編集(E) 表示(V) プロジェク	ト(P) ビルド(B) デバッグ(D) ツ
<b>G</b>	リビルド・プロジェクト(R) Shift+F7	デバッグ・ソリューション(S)	•
	クリーン・プロジェクト(C)	デバッグ・ツールへダウンロー	F(D)
TON	ラピッド・ビルド(A)	レルドルデルデバッガ・ツールへない	1/Π-F(B) F6
-2	依存関係の更新(P)		
E	RL78_G13_PORT1 をビルド(U)	リビルド&デバック・ツールへろ	7ウンロード(W)
	RL78_G13_PORT1 をリビルド(E)	□□ デバッグ・ツールへ接続(C)	id
	RL78_G13_PORT1 をクリーン(L)	デバッグ・ツールからアップロー	-۴(U)
<b>.</b>	RL78_G13_PORT1 の依存関係の更新(D)	N TIP R AL A ANTHONY (AL	
	RL78_G13_PORT1 のリンク順を設定する(K)	画面 アバック・ツールかららのET(N	) Shift+F6
퀾	RL78_G13_PORT1 用に最適化性能比較ツールを表示(Z)	使用するデバッグ・ツール(L	) •
*	ビルドを中止(S) Ctrl+F7		Shift+F5
Tr	ビルド・モードの設定(M)	● 第行(G)	FS
E)	バッチ・ビルド(T)		15
Tę	ビルド・オプション一覧(0)	111201 リレークはす(1(上)	FR

ダウンロードが完了して、シミュレータが起動した画面を下に示します。ここで、右下の赤 く囲んだ部分(ステータス表示部)に注目してください。



この部分の拡大を下に示します。左側が 0x0016F でブレーク中であることを示しています。 右端の部分の「3.062µs」の部分ですが、ここまでの実行時間(実行開始してからブレーク状 態になるまでの時間)が 3.062µs であることを示しています。

BREAK Q 0x0016f C RL78 シミュレータ (3) 3.062 µs	BREAK	ᅌ 0x0016f	🚥 RL78 シミュレータ	👸 3.062 µs
--	-------	-----------	---------------	------------

そこで、次ページに示すように2つの時間待ちにブレーク・ポイントを設定します。

		while (1U) {	
00172		P3 bit.no1 = 0:	/* P3.1を0(ロウ出力)にする */
00175	<b>S</b>	wait_200();	
00178		P3_bit.nol = 1;	/* P3.1を1 (ハイ出力) にする */
0017b	-	wait_200();	
		}	

この状態で、 🕑 をクリックしてブレーク・ポイントを有効にして実行します。すると、以下 のように LED が点灯して、時間待ちの実行前でブレークします。



このときの画面右下のステータスを見ると、以下のように 468ns の実行時間となっていま す。これは、R\_MAIN\_UserInit 関数を実行し、P3\_bit.no1 = 0 の実行が完了するまでの時間で す。

BREAK 💫 0x00175 🚥 RL78 シミュレータ 👸 468.000 ns

再度, 💿 をクリックして, プログラムを実行して, 上側の時間待ちと次の P3\_bit.no1 = 1 を 実行します。その結果は以下のように LED が消灯した状態になります。



ここでステータスは以下のようになります。

🍛 0x0017b 🔍 🚥 RL78 シミュレータ 🝈 170.001 ms BREAK

実行時間が 170ms となっていることが確認されました。目標の 200ms よりは 15%程度短く なっていますが、ほぼ予想通りの時間です。

これを、より 200ms に近づけるためには、wait 200 のループ(20 クロックと予想)を3 ク ロック分長くすればいいことになります。このためには、以下のように NOP()を3個追加する のが簡単です。



このように変更したプログラムをビル ドしてシミュレータにダウンロードしま す。今回は、右に示すように、一気にビ ルドからダウンロードまでを実行しま す。

編	集(E)	表示(V)	プロジェクト(P)	ビルド(B)	デバッグ(D)
	デバッ	グ・ソリューシ	ション(S)		•
D,	デバッ	グ・ツールへ	ダウンロード(D)		
6	ビルド	&デバッグ・	ツールヘダウンロー	-ド(B)	F6
5	リビル	ド&デバッグ	・ツールヘダウンロ	–F(W)	
00	デバッ	グ・ツールへ	接続(C)		

これを、ブレーク付きで2回実行すると、以下のようになります。



ステータス部分を拡大すると以下のようにほぼ 200ms になりました。



このように, NOP()を使うと、1 クロック単位でチューニングすることができます。 これで、何とかほぼ 200ms ごとに LED を点灯/消灯を行わせることができたので、最初の 目標は達成できました。プロジェクトを保存しておきましょう

ただし、このプログラムは使用している環境(CS+CC-RLのV6.00.00)で200msが得られたものです。全く同じプログラム(ビットの記述方法が異なるのでそのままではだめですが)をCS+CA78K0Rでビルドすると、異なる時間になることが考えられます。

CA-78KOR と CC-RLでは、ビットの記述方法が異なります。
CC-RLでは iodefine.h ファイルでビット・フィールドを定義して対応していましたが、CA-78KOR では初期状態でビットに対応しています。
CC-RL ・・・・ P3\_bit.no1
CA78KOR ・・・ P3.1
そもそも、CA-78KORでは iodefine.h ファイルではなく、内部のデバイス・ファイルで SFR を
定義しています。そのため、デバイスでの定義に近い表現が可能になっています。

この結果を BlueBoard-RL78/G13\_64pin にダウンロードして,実行すると LED が点滅するは ずです。(この時点では,実機確認はまだやっていません。今後,E1 または E2Lite を用いたデ バッグの例を示します。) 7. BlueBoard-RL78/G13\_64pin での動作確認

E1 を使って, 実際に RL78/G13 (BlueBoard-RL78/G13\_64pin) で動作させます。 7.1 デバッグ・ツールの変更

使用するデバッグ・ツールを変更するには, プ ロジェクト・ツリーの RL78 シミュレータ (デバ ッグ・ツール)を右クリックしてメニューを表示 し,「使用するデバッグ・ツール (D)」を選択す ると,選択可能なツールが表示されるので, 「RL78 E1 (Serial) (L)」を選択します。これで デバッグ・ツールとして E1 が選択されます。



プロジェクト・ツリーの「RL78 E1 (Serial) (デバッグ・ツール)」を右クリックしてメニュ ーを表示して,「プロパティ (P)」を選択します。(E2 Lite を使用する場合は,「RL78 E2 Lite (E)」を選択します。)

「接続用設定」タグが表示されるので、「ターゲット・ボードとの接続」を開いて「エニュレ ータから電源供給をする(最大 200mA)」を見ると「いいえ」となっています。

▲ ターゲット・ホードとの接続 エミュレータから電源供給をする(最大200 mA)	いいえ	Y
<ul> <li>ノフランユ セキュリティID フラッシュ書き換えを許可する</li> </ul>	HEX 000000000000000000000000000000000000	
<b>エミュレータから電源供給をする(最大200 mA)</b> エミュレータからターゲット・ボードに電源を供給するかどうか指定します。		
・ 接待用語字 デバッグ・ハール語字 / ダウンロード・ファイル語字 / コ	>>>/	

今回はこれを「はい」に変更し、「供給電圧[V]」を 3.3V にします(E2Lite では、3.3V しか供給できません)。

』 カーゲット・ボードンの接待

 2 771 W LC0180A	
エミュレータから電源供給をする(最大200mA)	はい
供給電圧[V]	3.3V

## 次に、「デバッグ・ツール設定」タグを選択します。

🔊 RL78 E1(Serial) のプロパティ		م	-	+
4 XEU				
▶ メモリ・マッピング	[9]			
メモリ書き込み時にベリファイを行う	はい			
▲ 実行中のメモリ・アクセス				
実行を一瞬停止してアクセスする	しいえ			
実行中に表示更新を行う	はい			
表示更新間隔[ms]	500			
▷ ブレーク				
▲ 入力信号のマスク				
TARGET RESET信号をマスクする	いいえ			
INTERNAL RESET信号をマスクする	いいえ			
▲ ステップ実行				
指定セクションをスキップする	いいえ			
				_
メモリ・マッピング				
メモリ・マッピングの追加、削除、開始アドレスもしくは終了アドレスの変更は、[…]ボ	タンを選択し、「メモリ・マッピング]ダイアログで行ってください。			
接続用設定 デバック・ツール設定 タウンロード・ファイル設定 / フック	观理設定 /			•

ここで「ブレーク」の項目を開きます。

▲ ブレーク		
優先的に使用するブレークポイントの種類	ソフトウェア・ブレーク	
停止時にタイマ系周辺エミュレーションを停止する	いいえ	
停止時にシリアル系周辺エミュレーションを停止する	いいえ	

「優先的に使用するブレーク・ポイントの種類」が「ソフトウェア・ブレーク」となっていますが、これはブレーク・ポイントを設定した命令をブレーク命令に置き換えてしまうことで、プログラムの実行を中断(ブレーク)するものです。このためにデバッグ中には頻繁にフラッシュ・メモリしまいます。このために、デバッグで使用したデバイスは製品に使用しないように制限されています。

ソフトウェア・ブレークの他には、ハードウェア・ブレークがあります。しかし、設定でき るポイントが少ないことと、設定したポイントではなく、そこから少しずれてブレークするの で、使いにくいのであまりお勧めできません。

ここで、設定を変更したいのは、ブレーク時にタイマ系を停止させることです。

▲ フレーク		
優先的に使用するブレークポイントの種類	<u>いつトウエア・ブレーク</u>	
停止時にタイマ系周辺エミュレーションを停止する	はい	Y
停止時にシリアル系周辺エミュレーションを停止する	いいえ	

「停止時にタイマ系周辺エミュレータを停止する」を「はい」に変更します。こうすること で、ブレーク中タイマが停止し、変なタイミングでタイマからの割り込みが発生することがな くなります。(逆に言うと、この設定が「いいえ」になっていると、ブレーク中で CPU が停止 している状態でもタイマがカウント動作を継続し、次々と割り込みが発生してしまい、プログ ラムが正常に実行できなくなる可能性があるからです。

シリアル系は、停止しない方がいいでしょう。これは、通信は必ず通信相手が存在します。 できるだけ、相手と合わせることを考えると、動作させておいた方がいいからです。もちろ ん、これは場合場合で異なるので、どうしても止める必要があり、それで以降の動作に問題が 発生しないなら止めるのもありです。

次は、「ダウンロード・ファイル設定」タグを開きます。そこで、「デバッグ情報」の項目を 見ると、「CPU リセット後に指定シンボル位置まで実行する」が「はい」で、「指定シンボル」 が「\_main」となっています。これは、デバッガを起動したときに main 関数の先頭で停止する というものです。つまり、スタートアップ・ルーチンの実行が完了し、内蔵周辺機能の初期化 や RAM の初期化が完了しているところからデバッグを開始することを意味しています。

RL78 E1(Serial) のプロパティ	- 0
▲ ダウンロード	
▶ ダウンロードするファイル	[1]
ダウンロード後(CCPUをリセットする	はい
ダウンロード・モードの選択	スピード優先
ダウンロード前にフラッシュ ROMを消去する	いいえ
イベント設定位置の自動変更方法	イベントを保留にする
予約領域の上書きをチェックする	(4)
⊿ デバッグ情報	
CPUリセット後に指定シンボル位置まで実行する	(\$0)
指定シンボル	_main
メモリ使用量の上限サイズ[M/ドイト]	500
ダウンロードするファイル	
ダウンロードするファイルを指定します。[…]ボタンを押下するとダウンロード・	・ファイル ダイアログが聞きます。 ダウンロード・ファイル ダイアログで、 ダウンロードするファイルを指定してください。
◇ 接続用設定 / デバッガ・ツール設定 ↓ ダウンロード・ファイル設定	

ここで、「main」ではなく、「\_main」となっているのは、アセンブラでのシンボル名が使われているので、C表記のシンボルの前に「\_」が付いているからです。

7.2 ハードウェアの接続と実行

使用するハードウェアは、右に示すボードで す。左下の方に E1 を接続するための 14 ピンの コネクタがあり、そこに E1 を接続して、E1 か ら 3.3V の電源を供給しています。

右に示す写真は,下に示すように,LED を点灯 させた直後の NOP();でブレークをかけた状態で す。

このボードには、下側の辺の中央に AC アダプ タを接続するコネクタがあります。秋月の HP の 説明やドキュメントには、6.5V の AC アダプタ を接続するように書かれていますが、これは間 違いでしょう。(回路図では 7.5V と書かれてい ますし、そもそも 6.5V の AC アダプタは存在し ません。)





このときの LED の点灯状態が、下の左側の写真です。右側は、ブレークを掛けず、RUN(実行) させたときの写真です。短い周期で点滅を繰り返していますが、目でも写真でも点滅は見 えません。写真では、点灯状態と比べると暗くなっているのが分かる程度です。



200msのタイマを呼び出す処理を追加したプログラムでは、400ms 周期での点滅が確認できました。点滅の様子をビデオでもと思いましたが、ファイルが大きくなり過ぎるのでやめました。実際にハードウェアを動作させて、自分の目で確認してください。

ここでも、実際にプログラムを実行させて、実行時間を計測してみます。LED を消灯すると ころにブレーク・ポイントを設定してみました。一回目のブレーク時の測定結果は、

200.100ms となっています。なお、シミュレータの場合と異なり、この実行時間はあまり正確 ではありませんので、ご注意ください。

	× 🔤 プロパテ.	ィ も達アセンブル1	I r_main.c	<b>→</b> X	ローカル変数
RL78 G13 PORTL 1 (プロジェク RL78 G13 PORTL 1 (プロジェク FRF100LE (マイクロコントローラ) Dード生成 (設計ツール) GT-FL (設計ツール) C-FL (ジはドツール)		→ へ、へ、カラ、 アドレス □ (* 0016f	A* P_MAIN_UserInit(): /# Start user code. Do not edit comment generated here #/ while (1U) {	^	▲ aca(n) くう エンゴ (16) カレント ・ 名前 1i
AL78 E1(Serial) (デバッグ・ツール 日本 E1(Serial) (デバッグ・ツール ロージ ファイル Cstart.asm く	65 66 67 68 69	00172 00175 00178 0017b	P3_bit.nol = 0; /# P3.1老0(口口进力) wait_200(); /# P3.1老1 (小石进力) #ait_200(); /# P3.1老1 (小石进力) wait_200(); // F3.1本1 (小石进力)	こする */  こする */ 	
FIエディタ・パネ・・・・ FE 名前の変更 FE 次を検 CPUリセット後、プログラムを実行します。	F4次を置き	換 肟実行	FGビルド&デパッ・・・F7ビルド・プロジ・・・ FB ブレークせず・・・・ F日ブレ 67 行 1 桁 挿入日本語 (シフト JIS) ● BR	レークの設・・・ FH2ステップ・オ REAK 今 0x00178	t FTRステップ・イン FTE関数へジャンプ ■ RL78 E1(Serial) (3 200.100 ms

更に,ブレーク・ポイント付きで実行させると,下に示すように 400.100ms と表示されました。

プロジェクト・ツリー 🛛 🕂 🗙	ご プロパティ 電気 逆アセンブル1 ダ r_mainc	ローカル変数 4 ×
2 @ 2 2		② 表記(N)▼ 場 エンコード(C)▼
<ul> <li>RL78 G13 PORT1 1 (プロジェクト) へ</li> <li>R5F100LE (マイクロコントローラ)</li> <li>メーク 端子配置 (設計ツール)</li> <li>コード生成 (設計ツール)</li> </ul>	1     1 <td>カレント ・ ・ 名前 (1</td>	カレント ・ ・ 名前 (1
	64 65 00172 P3_bit.nol = 0; /# P3.1を0 (ロウ出力)にする */ 66 00175 v3.1 (2000); /# P3.1を0 (ロウ出力)にする */ 67 00178 v2 P3.1 (ロク出力)にする */	③ ローカル変数 (第 CPUレジスタ)
stkinit.asm	88 00176 * **********************************	出力 ♀ × すべての xwtz *デバッグ・ツール / ▼ ▲ 出力 → スマート・ブラウザー
エディタ・パネ・・・ F2名前の変更 F3次を検索	F3 次を置換 F5実行 F6 ビルド&デバッ・・・・ F7 ビルド・プロジ・・・・ F8 ブレークせず・・・・ F9 ブレークの酸・・・ F80 ステップ・フ	オー・・・・ FNステップ・イン 日2閲数へジャンプ
	67 行 1 桁 挿入 日本語 (シフト JIS) <b>BREAK 😂 0x00178</b>	👓 RL78 E1(Serial) 🛛 🖓 400.100 ms

これで、実際の RL78/G13 のデバイスでの動作も確認できました。

最後になりましたが、E1から電源を供給する場合には、他からの電源供給とぶつからないように注意してください。他から電源を供給するときは、E1からの電源供給はしないように設定してください。

また, E1 から供給できるのは 200mA までです。それ以上流れるような場合には, E1 から供給しないでください。

これらが守られないと、E1 が壊れる可能性があります。

なお、ボードに電源が供給されていない場合には、下のようなメッセージが表示されます。



【おまけ】

AC アダプタから 7.5V と 6V を入力して, CPU\_VDD を測定したところ, どちらの場合も 4.4V でした。少なくとも, RL78 は 2.7V 以上の電圧なら 32MHz で動作可能なので, AC アダ プタの電圧はあまり気にしないで進めていきます。ちなみに, 5V の AC アダプタを使ったと ころ, LCD がどうも表示できなくなるようです (スイッチや LED は使えそうですが)。

次回は、ポートの入力と出力を組み合わせた制御を行います(単に SW を押せば LED が点灯 するだけですが)。

以上