RX600 シリーズ
RPDL および PDG の使い方

R01AN0615JU0100
Rev.1.00
2011.09.27

要旨

ルネサス周辺ドライバライブラリ（RPDL）はルネサスマイクロコントローラ上で周辺モジュールを設定および制御するための統合 API です。サポートされている周辺機能としては、タイマ、ウォッチドッグタイマ、DMA、DTC、SPI、I2C、AD コンバータおよび DA コンバータなどがあります。下図に示すように、RPDL はユーザアプリケーションと対象 MCU の間にあるソフトウェア抽象化層です。RPDL は対象 MCU 上でサポートされる周辺機能レジスタのすべての読み出しと書き込みを処理します。RPDL はほとんどの周辺機能をサポートしていますが、USB、イーサネット、CAN などのより複雑な周辺機能はサポートしていないので、使用するには独自のドライバが必要です。これらのドライバによって RPDL を自由に使用することができます。

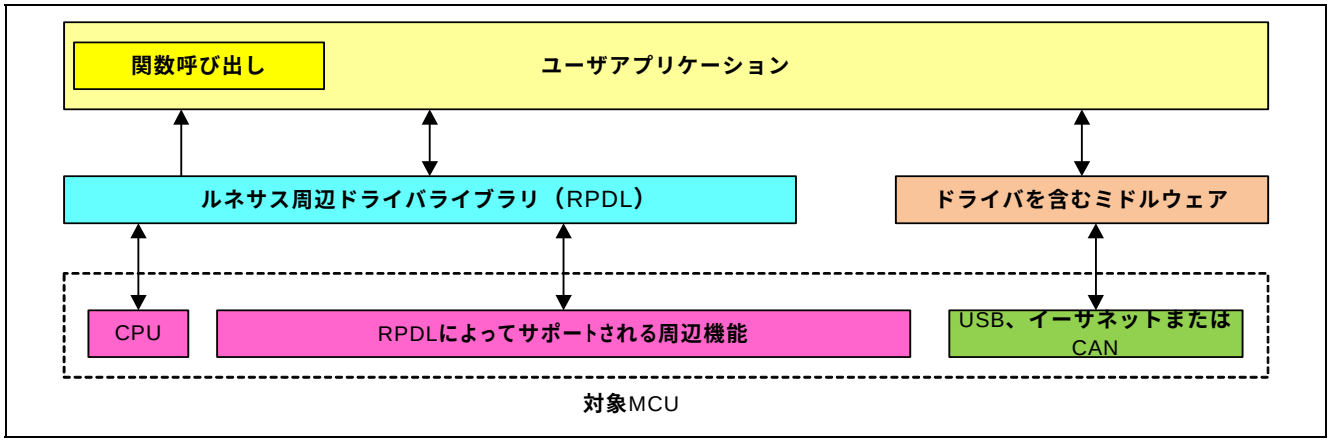


図　ソフトウェア層を持つ MCU

Peripheral Driver Generator（PDG）は PC 上で動作するグラフィカルツールで、RPDL 関数呼び出しを生成します。PDG は使いやすいグラフィカルインタフェースを通じて対象デバイスの周辺機能を構成する機能をユーザに提供します。ユーザが周辺機能を PDG に構成すると、周辺機能はコードを生成することができ、PDG はこれらのファイルを HEW プロジェクトに自動的に組み込みます。

このアプリケーションノートでは、RPDL の使い方と PDG によって付加される効果について説明します。最初に RPDL を使用して HEW で周辺機能のサポートを手動で実装する実例を示し、その後、PDG を介して同じサポートを生成する簡単な方法を示します。

動作確認デバイス

YRDKRX62N

目次

1. 必要なツール

このアプリケーションノートは YRDKRX62N プラットフォームを対象として作成されていますが、RPDL と PDG を使用する基本的な手順はどのようなプロジェクトにも適用することができます。

この手順を実行する前に、以下のものをインストールしてください。

- RX62N RDK DVD
  http://am.renesas.com/products/tools/introductory_evaluation_tools/renesas_demo_kits/yrdkrx62n/child_folder/downloads_child.jsp
- Peripheral Driver Generator　バージョン 2
  http://www.renesas.com/pdg_download

本書で参照される、RPDL を含むその他のコードは、このアプリケーションノートに同梱されている'Source' ディレクトリに収められています。

2. RPDL ワークスペースの作成

本章では、RPDL で使用するワークスペースのセットアップについて説明します。

手順：

ステップ 2.1 Start ⇒ All Programs ⇒ Renesas ⇒ High Performance Embedded Workshop ⇒ High Performance Embedded Workshop の順に選択して HEW を起動します。

ステップ 2.2 Welcome ウィンドウで、"Create a new project workspace"を選択して、"OK"をクリックします。

ステップ 2.3 "CPU family"ドロップダウンから"RX"を選択します。

ステップ 2.4 "Tool chain"ドロップダウンから"Renesas RX Standard"を選択します。

ステップ 2.5 "Project Types"の下のエントリから"Application"を選択します。

ステップ 2.6 "Workspace Name"フィールドに"RX_RPDL_Demo"を入力します。ウィンドウは図 2.1 のようになります。"OK"をクリックします。

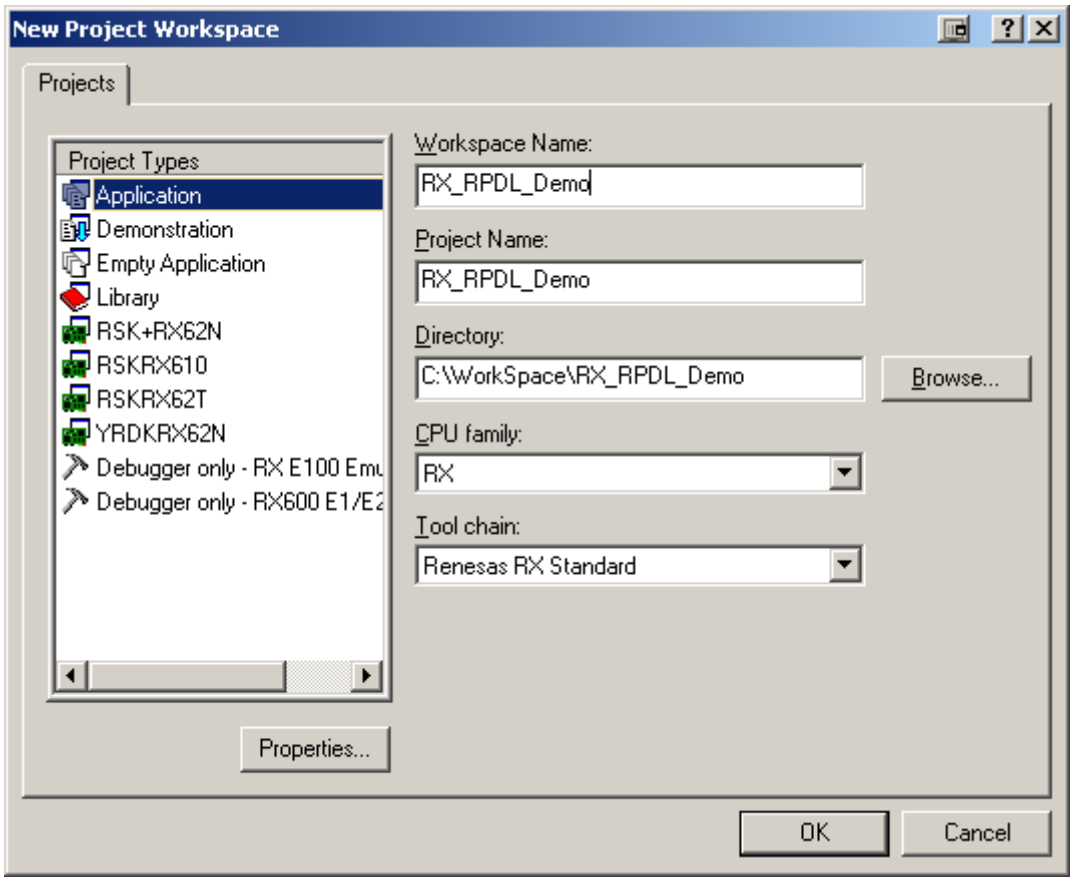


図 2.1 新しいワークスペースの作成

ステップ 2.7 ウィンドウ内で、"CPU Type"の下にあるリストから"RX62N"を選択します。"Next"をクリックします。

ステップ 2.8 タイトルバーに"New Project-3/10..."と表示されるウィンドウが出るまで"Next"をクリックします。このウィンドウで、"Changes code generation"ドロップダウンボックスから"None"を選択します。

ステップ 2.9 タイトルバーに"New Project-4/10..." と表示されるウィンドウ（図 2.2）が出るまで"Next"をクリックします。"I/O Register Definition Files"の横にあるボックスのチェックを外します。"Next"をクリックします。

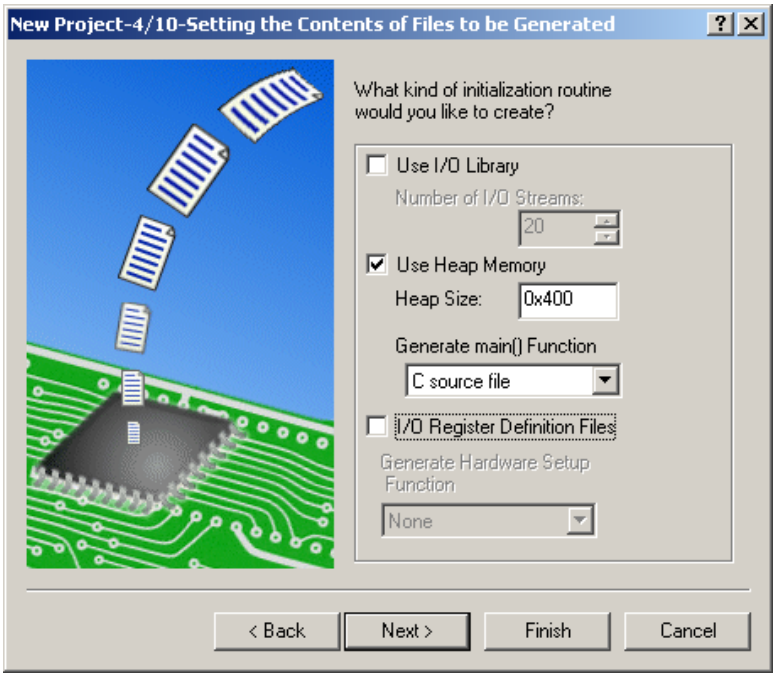


図 2.2　New Project ウィンドウ 4

ステップ 2.10　タイトルバーに"New Project-8/10..." と表示されるウィンドウ（図 2.3）が出るまで"Next" をクリックします。"RX600 Segger J-Link"の横にあるボックスをチェックして、"Finish" をクリックします。

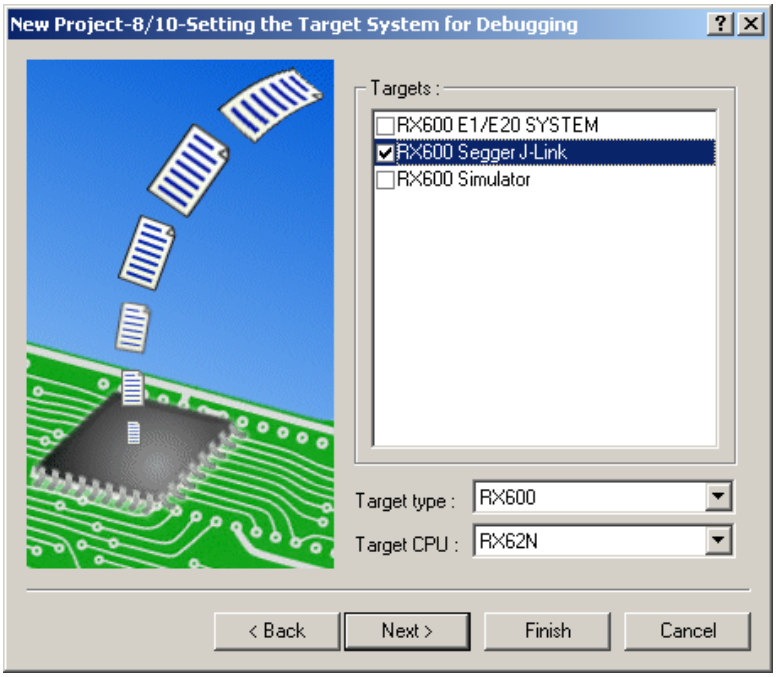


図 2.3　デバッガの選択

ステップ 2.11　Summary ウィンドウで"OK"をクリックします。

ステップ 2.12　次に、Explorer ウィンドウから、このアプリケーションノートの内容を抽出したフォルダの下にある'Source/RPDL_RX62N'フォルダに移動します。

ステップ 2.13　ファイル Copy_RPDL_RX62N.bat をダブルクリックします。コマンドウィンドウが開きます。

ステップ 2.14　デバイスを選択するように求められた場合は、LQFP100 ピンに対応する番号を使用して、"Enter"を押します。

ステップ 2.15　RPDL をインストールする場所を尋ねられた場合は、"C:¥Workspace¥RX_RPDL_Demo¥RX_RPDL_Demo"と入力し"Enter"を押します。RPDL ファイルがコピーされ、終了すると、ウィンドウは図 2.4 のようになります。いずれかのキーを押して、コマンドウィンドウを閉じます。

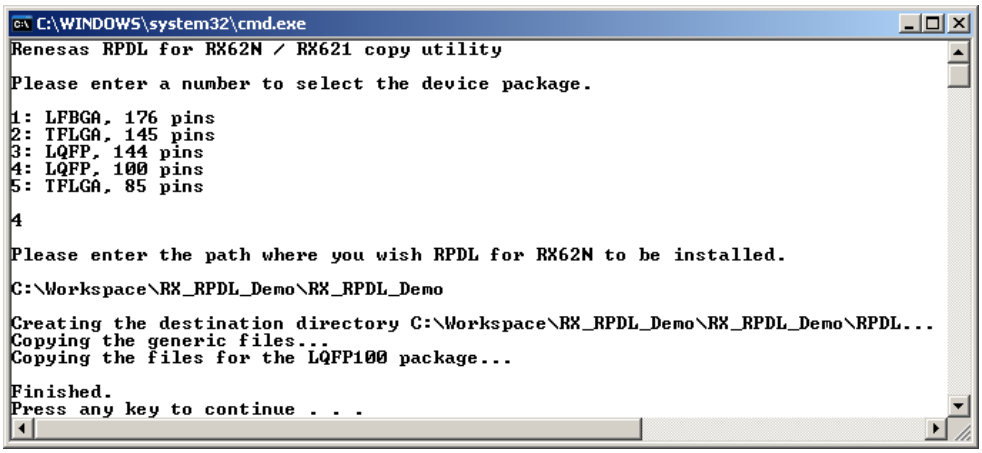


図 2.4　ワークスペースへの RPDL のコピー

このバッチファイルにより、適合する RPDL ライブラリと必要な C ソースおよびヘッダファイルがワークスペースにコピーされます。必要に応じて手動でファイルをコピーすることもできます。

ステップ 2.16　ディレクトリ"C:¥Workspace¥RX_RPDL_Demo¥RX_RPDL_Demo"を参照して、"RPDL"というフォルダがあることを確認します。このフォルダがない場合は、ステップ 2.15 を繰り返し、正しいインストールパスを入力します。

ステップ 2.17　HEW ワークスペースに戻り、Build ⇒ RX Standard Toolchain の順にクリックします。

ステップ 2.18　"C/C++"タブを選択して、"Show entries for"ドロップダウンを"Include file directories"に変更します。

ステップ 2.19　"Add"ボタンをクリックします。

ステップ 2.20　表示されるウィンドウで、"Relative to"ドロップダウンを"Project directory"に変更して、"Sub-Directory"欄に"RPDL"を入力します。ウィンドウは図 2.5 のようになります。"OK"をクリックします。

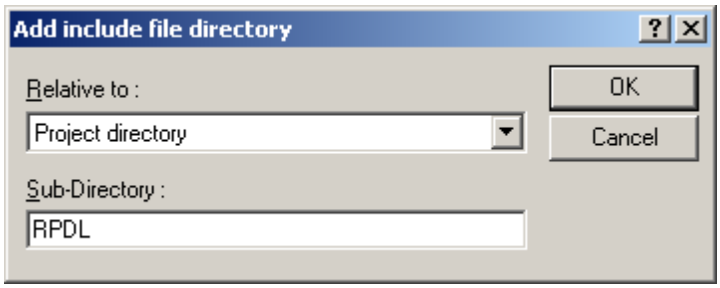


図 2.5　RPDL フォルダのインクルード

ステップ 2.21　同じ手順で別のディレクトリを追加します。今回は"Sub-Directory"欄に"."を入力します。（「ピリオド」1 個は「カレントディレクトリ」を意味し、プロジェクトディレクトリになります。）

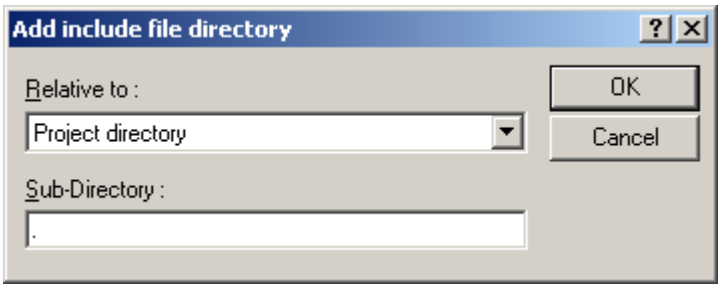


図 2.6　プロジェクトのルートフォルダのインクルード

ステップ 2.22　ウィンドウの上部の"Link/Library"タブをクリックします。

ステップ 2.23　"Category"ドロップダウンで"Input"が選択され、"Show entries for"で"Library files"が選択されていることを確認します。

ステップ 2.24　"Add"ボタンをクリックします。

ステップ 2.25　"Relative to"ドロップダウンを"Project directory"に変更します。

ステップ 2.26　"File path"欄に"RPDL¥RX62N_library.lib"を入力し、"OK"をクリックします。

ステップ 2.27　"OK"をクリックして、RX Standard Toolchain を終了します。

ステップ 2.28　Project ⇒ Add Files の順にクリックします。表示されるウィンドウで、"Files of type"ドロップダウンから"C source file"を選択します。

ステップ 2.29　プロジェクトフォルダの下の"RPDL"フォルダに移動し、Interrupt_EXDMAC.c を除くすべての C ソースファイル（*.c）を選択し強調表示させます。100 ピンおよび 85 ピンの RX62N 製品は EXDMA コントローラをサポートしていないので、Interrupt_EXDMAC.c は含まれていません。ファイルを選択し強調表示させた後、"Add"をクリックします。

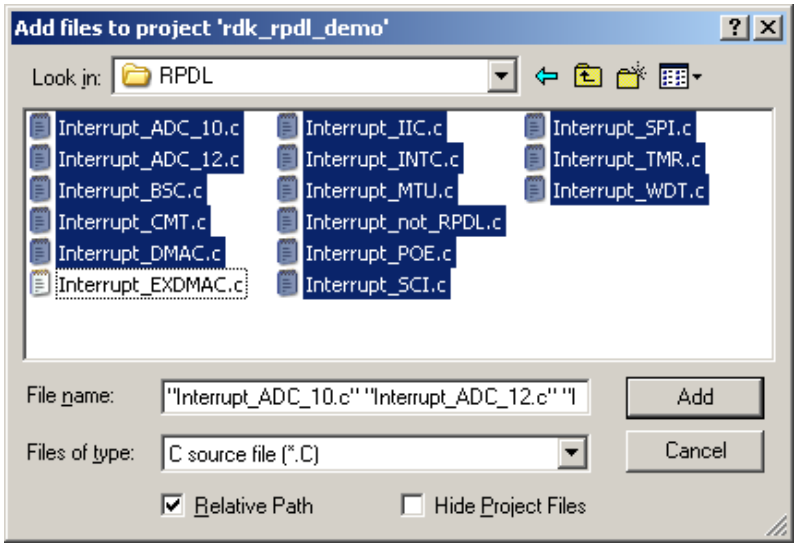


図 2.7　プロジェクトへの RPDL ファイルの追加

RPDL C ソースファイルを追加した後、プロジェクトペインが見にくくなることがあります。ファイルを整理するには、フォルダを作成して、そのフォルダに格納します。"C source file"フォルダを右クリックして、"Add Folder"を選択します。フォルダ名に"RPDL"を入力し、'OK'をクリックします。RPDL ファイルを新しいフォルダにドラッグアンドドロップすることができます。'RPDL'ファイルは"Interrupt_"で始まるので、簡単に見つけることができます。完了すると、プロジェクトペインは以下のようになります。

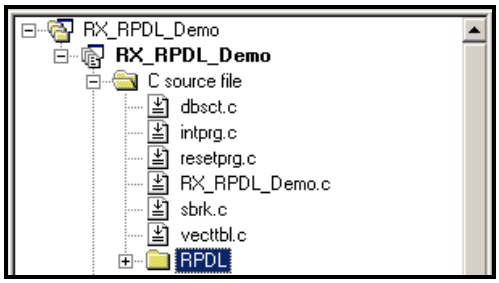


図 2.8 RPDL フォルダの作成

ステップ 2.30 RPDL はデバイス割り込みを処理するので、RX "Application"プロジェクトで自動的に生成される intprg.c および vecttbl.c ファイルと競合します。プロジェクトナビゲーションペインでこれらのファイルを選択し強調表示させたまま、右クリックして"Exclude Build ..."を選択して、両方のファイルを除外します。

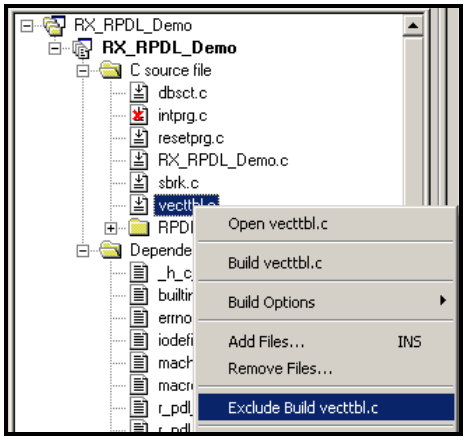


図 2.9 ベクタ処理ファイルの除外

ステップ 2.31 RPDL がプロジェクトに追加され、ビルドすることができます。Build All ボタン を押すか、Build ⇒ Build All の順に選択します。

プロジェクトをビルドするときに、'L1100 (W) Cannot find "PIntPRG" specified in option "start"'という警告が表示されます。これは'intprg.c'ファイルを以前に除外したためです。RPDL が割り込みを処理するので、この警告は無視して問題ありません。また、toolchain メニューに進み、'PIntPRG'セクション名を削除することにより警告を削除することもできます。Build ⇒ RX Standard Toolchain の順に選択し、"Link/Library"タブに切り換え、"Category"を"Section"に変更することにより、プロジェクトのセクションを表示することができます。

3. ルネサス周辺ドライバライブラリ（RPDL）の使い方

本章では RPDL の使い方を説明します。ユーザは AD コンバータを起動するためのタイマを作成します。AD 変換完了後、その値が LCD に表示されます。

手順：

ステップ 3.1 このアプリケーションノートの内容を解凍したフォルダの下にある"Source"ディレクトリに進みます。

ステップ 3.2 以下のファイルを"Source"ディレクトリから HEW ワークスペースプロジェクトディレクトリ（"C:¥Workspace¥RX_RPDL_Demo¥RX_RPDL_Demo"）にコピーします。

- glyph_api.h
- Glyphlib_v2.lib
- lcd.c
- lcd.h
- lcd_utilities.c
- lcd_utilities.h
- YRDKRX62N.h
- YRDKRX62N_RSPI_API.c
- YRDKRX62N_RSPI_API.h

ステップ 3.3 第 2 章で行ったとおり、Project ⇒ Add Files の順に選択して、ファイル lcd.c、lcd_utilities.c、YRDKRX62N_RSPI_API.c および Glyphlib_v2.lib をカレントプロジェクトに追加します。

ステップ 3.4 HEW でファイル RX_RPDL_Demo.c を開きます。

RPDL ライブラリをプロジェクトに含めましたが、RPDL 関数と定義を使用するために適切なヘッダファイルも含める必要があります。使用するすべての RPDL ソースファイルには、使用する 2 種類のヘッダファイルがあります。

1. r_pdl_definitions.h
   a. **このファイルにはデバイス固有の定義が含まれています。**

2. r_pdl_*PERIPHERAL*.h
   a. **これらのファイルにはドライバ関数プロトタイプが含まれます。**
   b. **例：**r_pdl_adc_10.h、r_pdl_cmt.h、r_pdl_tmr.h **など**

ドライバ関数プロトコルを宣言するヘッダは、常に r_pdl_definitions.h **ファイルの前のソースファイルに含まれます。**

ステップ 3.5 このデモでは 10 ビット ADC と 8 ビットタイマを一緒に使用します。また、システムクロックを設定するためにクロック発生回路（CGC）も使用します。適合するヘッダファイルを RX_RPDL_Demo.c の先頭に置きます。

```
#include "r_pdl_tmr.h"
#include "r_pdl_adc_10.h"
#include "r_pdl_cgc.h"
```

ステップ 3.6 これらのインクルードステートメントの後に、ファイル r_pdl_definitions.h のインクルードを追加します。

```
#include "r_pdl_definitions.h"
```

ステップ 3.7 AD 変換値を LCD に表示するのに必要なすべての基本的なハードウェアサポートを取得するために、lcd_utilities.h にある関数 UpdateLCD()を使用します。RSPI API ライブラリにより、LCD が常駐する RSPI バスへの共用アクセスが可能になります。これらのファイルを前のインクルードステートメントの後に置きます。

```
/* RSPI API library support */
#include "YRDKRX62N_RSPI_API.h"

#include "lcd_utilities.h"
#include "lcd.h"
```

ステップ 3.8 第 1 章で説明した PDG のインストールをまだ実行していない場合は、ここでインストールしてください。

ステップ 3.9 RPDL API の使用を開始する準備ができました。PDG インストールディレクトリに格納されている RPDL ユーザーズマニュアル 'C:¥Renesas¥PDG2¥manuals¥r20ut0084ee0103_RX62N.pdf'を開きます。

RPDL ユーザーズマニュアルに簡単にアクセスするための便利な方法は、それを HEW プロジェクトに追加することです。ソースファイルを追加する場合と同じように追加することができます。"Files of type"**ドロップダウンが**"All Files"**に選択されていることを確認してください。選択されていない場合は、ファイルは表示されません。ファイルを追加した後、ファイル名がプロジェクトナビゲーションペインに表示され、そのファイルをダブルクリックすると PDF ファイルが開きます。**

ステップ 3.10 RPDL を使用するために最初にすべきことは、システムクロックの設定です。その他の周辺機能をセットアップするときにクロック設定が必要なので、システムクロックの設定が必要となります。RPDL ユーザーズマニュアルの第 4 章「ライブラリリファレンス」を参照してください。また、4.2.1 項の「クロック発生回路」も参照してください。

ステップ 3.11 リストされている最初の関数は R_CGC_Set()です。少し時間を使って API の説明を読み、セットアップする方法を確認してください。

ステップ 3.12 RX_RPDL_Demo.c 内の main()関数を確認し、RPDL 関数呼び出しを追加して、以下のパラメータを使用してシステムクロックを設定します。

- 入力周波数 12MHz
- システムクロック = 96MHz
- 周辺モジュールクロック = 48MHz
- 外部バスクロック = 24MHz
- BCLK 出力を無効にする

ステップ 3.13　RPDL ユーザーズマニュアルのとおり、以下のコードで設定が実行されます。

```
/* Setup system clocks */
R_CGC_Set( 12E6, 96E6, 48E6, 24E6, PDL_CGC_BCLK_DISABLE );
```

ステップ 3.14　次に 10 ビット ADC のセットアップを行います。RPDL ユーザーズマニュアルの 4.2.25 項を参照してください。

ステップ 3.15　GPIO 初期化後、関数呼び出しを R_ADC_10_Create( )に追加して、以下の値を使用して ADC を作成します。

- ユニット 1
- チャネル AN4
- シングルモード
- TMR0 で起動
- 右揃え（LSB 詰め）
- 48MHz 変換クロック
- サンプリング時間 0.6us（ハードウェアマニュアルでは、最小値は 0.5us と記載）
- 変換が終了したら関数 ADC_Callback()を呼び出す。
- IPL 4 を使用する。

ステップ 3.16　RPDL ユーザーズマニュアルのとおり、以下のコードで設定が実行されます。

```
/* Setup 10bit ADC */
    R_ADC_10_Create(1,
                    PDL_ADC_10_CHANNELS_OPTION_1 |
                    PDL_ADC_10_MODE_SINGLE |
                    PDL_ADC_10_TRIGGER_TMR0_CM_A,
                    48E6,
                    6E-7,
                    ADC_Callback,
                    4 );
```

ステップ 3.17　AD 変換を起動するための 8 ビットタイマを設定します。使用可能な API 関数については、4.2.15 項を参照してください。ここから R_TMR_CreatePeriodic()関数の説明を見つけてください。

ステップ 3.18　R_ADC_10_Create()関数呼び出し後に、関数呼び出しを R_TMR_CreatePeriodic()に追加して、以下の値を使用して ADC をセットアップします。

- ユニット 0（TMR0 と TMR1 をカスケード接続として 16 ビットカウンタを作成します。）
- 周波数を 4Hz に設定します。
- AD コンバータを起動します。
- 50%デューティサイクル
- コールバック関数なし
- コールバックがないので IPL は 0 とする。

ステップ 3.19 RPDL ユーザーズマニュアルのとおり、以下のコードで設定が実行されます。

```
/* Setup TMR */
R_TMR_CreatePeriodic( PDL_TMR_UNIT0,
                      PDL_TMR_FREQUENCY | PDL_TMR_ADC_TRIGGER_ON,
                      4,
                      0.5,
                      PDL_NO_FUNC,
                      PDL_NO_FUNC,
                      0 );
```

ステップ 3.20 使用する最後の初期化コードは、LCD の設定です。これらの関数は lcd_utilities.c ファイルに収められています。R_TMR_CreatePeriodic()への関数呼び出しの後、関数呼び出しを YRDKRX62N_RSPI_INIT()および InitializeLCD()に追加します。また、実行により main()関数が終了しないように、この呼び出しの後に無限ループを追加します。

```
/* Initialise the LCD display on RSPI bus */
YRDKRX62N_RSPI_Init( 0 );

/* Setup LCD */
InitialiseLCD();

/* Infinite loop */
while(1);
```

ステップ 3.21 周辺機能を初期化したら、コールバックルーチンを書き込まなければなりません。RX_RPDL_Demo.c ファイルで、以下の関数を作成します。

```
/* Callback function when ADC conversion has finished */
void ADC_Callback(void)
{
    uint16_t ADC_value;

    /* Read ADC value */
    //Fill in this line

    /* Update ADC value on LCD */
    UpdateLCD(ADC_value);
}
```

ステップ 3.22 RPDL ユーザーズマニュアルの「4.2.25 10 ビット ADC」を参照して、関数 R_ADC_10_Read()を見つけます。この関数を使用して AD 変換値を読み出す必要があるコード内の位置を埋め込みます。R_ADC_10_Read()はデータを格納する位置にポインタを置くことに注意してください。C プログラミング言語では、変数名の前に'&'を置くことにより、その変数のアドレスを取得します。コードからわかるように、'ADC_value'変数は ADC 読み取り値を保持するようにすでにセットアップされています。どの ADC ユニットを使用しているかについては、ステップ 3.15 を参照してください。

ステップ 3.23 RPDL 呼び出しを追加した後、ADC_Callback()関数の関数プロトタイプを#include の下の RX_RPDL_Demo.c ファイルの上部に追加します。

```
void ADC_Callback(void);
```

ステップ 3.24 Build ボタン  を押すか、ショートカット F7 を使用します。

ステップ 3.25 付属の USB ケーブルを使用して YRDKRX62N ボードをワークステーションに接続します。"J-Link USB"というラベルの付いたボード上の USB ヘッダを使用してください。

ステップ 3.26　セッションを"DefaultSession"から"SessionRX600_Segger_J-Link"に変更します。現在のデバッグセッションを保存するように求めるウィンドウがポップアップした場合は、"Yes"をクリックします。

図 3.1　デバッグセッションの変更

ステップ 3.27　接続ウィンドウが自動的に表示されない場合は、Connection ボタン をクリックするか、Debug ⇒ Connect の順に選択します。

ステップ 3.28　ウィンドウを図 3.2 と同じ構成内容にして、接続が終了するまで OK を押します。JTAG クロックを指定するように求めるウィンドウが表示された場合は、16.5MHz を選択します。ファームウェアをアップグレードするように求めるウィンドウが表示された場合は、"OK"を選択します。ファームウェアのアップグレードが終了したら、USB ケーブルを YRDK から外してから、もう一度接続します。ここでステップ 3.27 に従ってボードを再び接続します。

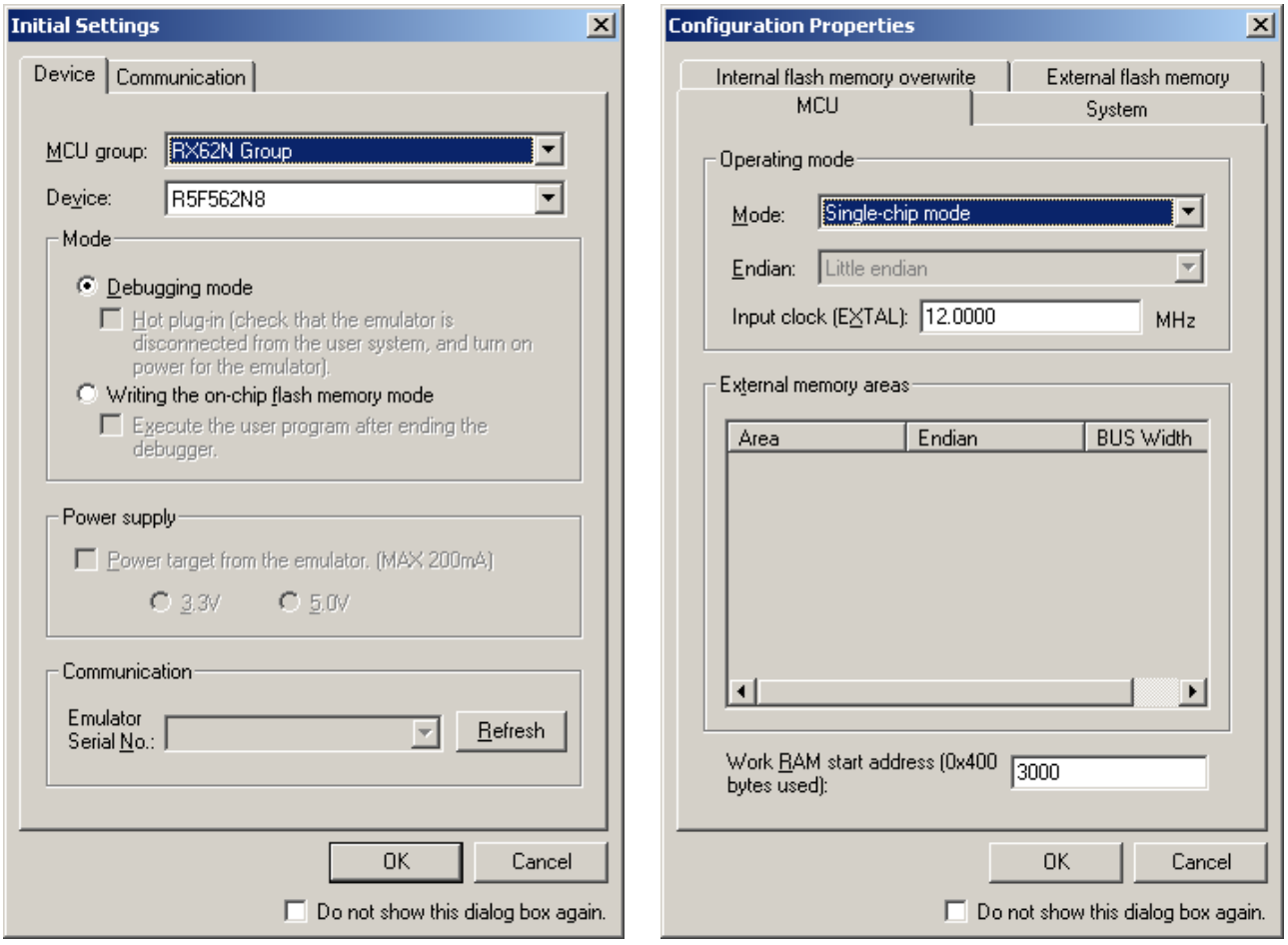


図 3.2　RSK への接続

ステップ 3.29　プロジェクトナビゲーション画面で RX_RPDL_Demo.abs ファイルの"Download"をダブルクリックするか、右クリックしてコードをボードにダウンロードします。

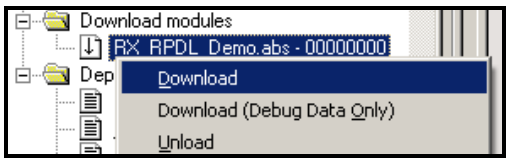


図 3.3　コードのダウンロード

ステップ 3.30　"Reset-Go"ボタン をクリックするか、Debug ⇒ Reset Go の順にクリックします。

ステップ 3.31　AD 変換値がボードの LCD に表示されることを確認します。ポテンショメータを回した場合、値が自動的に更新されることを確認します。

ステップ 3.32　MCU 実行を停止し、接続を解除し、HEW を閉じます。

4. Peripheral Driver Generator（PDG）の使い方

本章では、ここまで RPDL で手動でコーディングしたものと同じコードを PDG を用いて設定します。PDG は前のデモで手動でコーディングした RPDL 呼び出しを含むコードを生成します。

手順：

ステップ 4.1 PDG の使い方の実例を開始する前に、最初に HEW ターゲットサーバが登録されていることを確認してください。HEW を起動して、Tools ⇒ Administration の順に選択します。Extension Components フォルダを開いて、"HewTargetServer"が表示されていることを確認します。

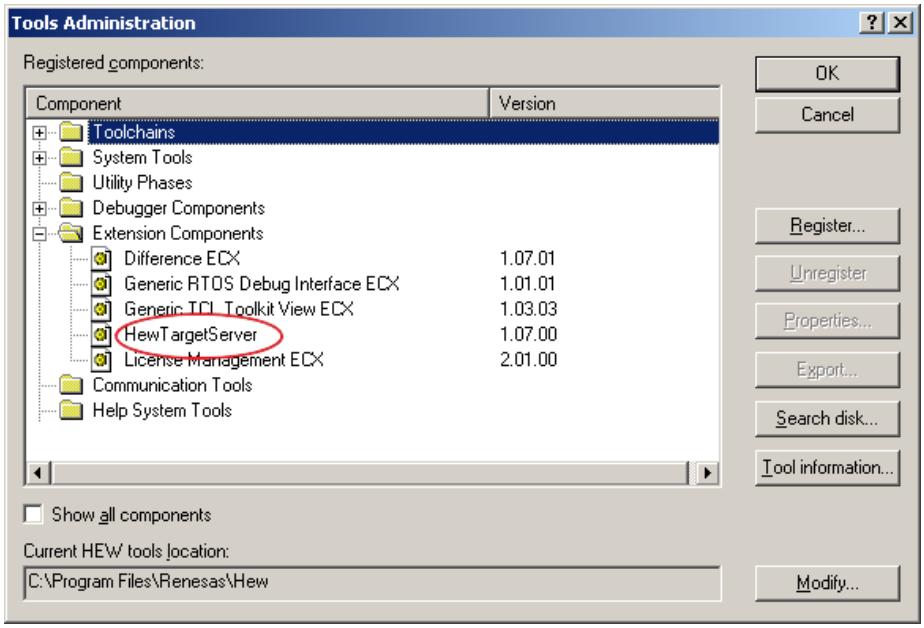


図 4.1 HTS レジストリの検証

ステップ 4.2 表示されていない場合は、"Search disk..."をクリックして、"Start"をクリックします。検索が終了したら、リストから"HewTargetServer"を選択して、"Register"をクリックします。完了したら、HEW を閉じます。

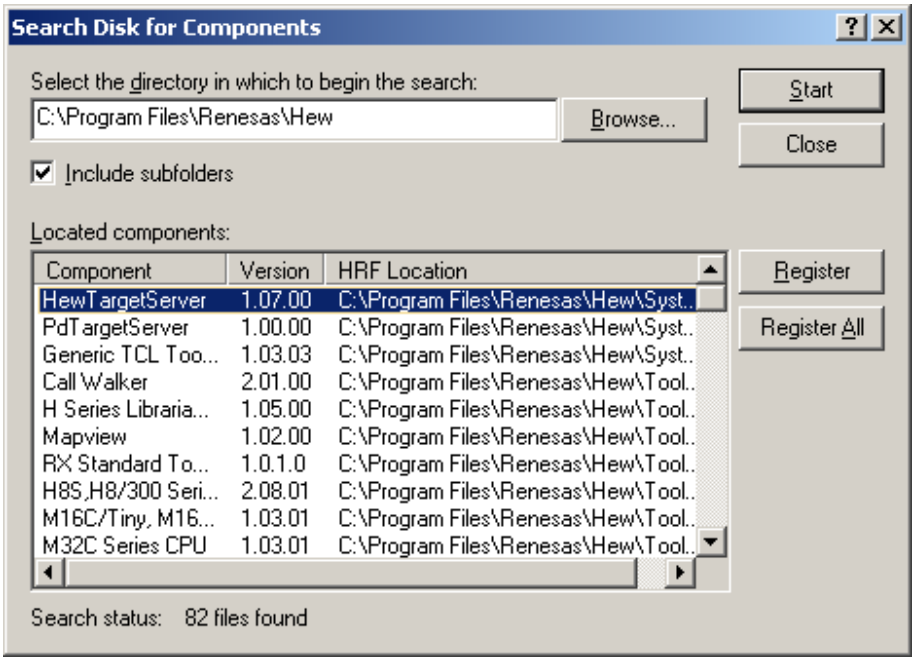


図 4.2 レジストリへの HTS の追加

ステップ 4.3 Start ⇒ All Programs ⇒ Renesas ⇒ Peripheral Driver Generator 2 の順に選択して PDG を起動します。

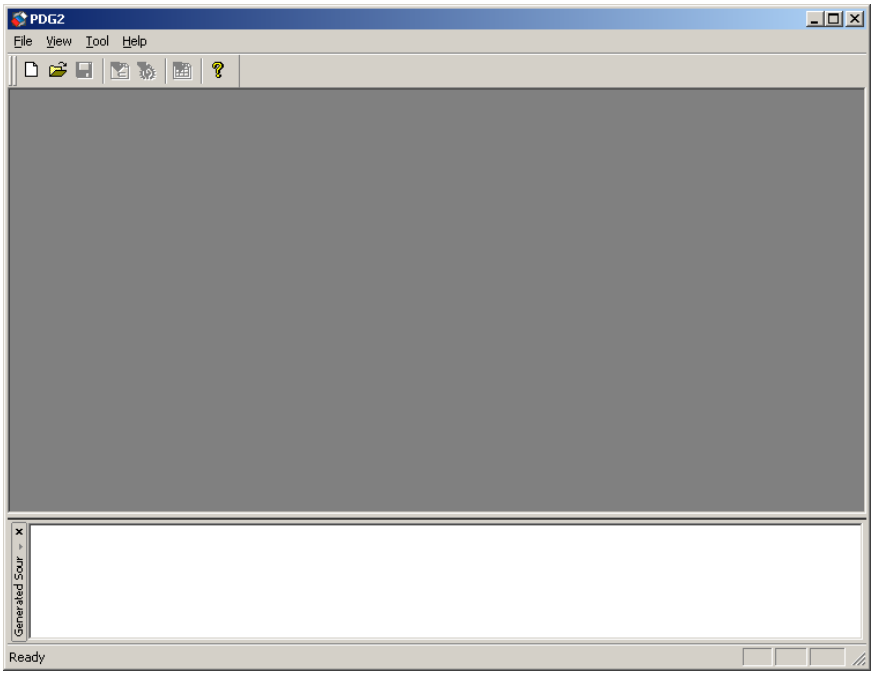


図 4.3 PDG アプリケーション

ステップ 4.4 File ⇒ New Project の順に選択して新しい PDG プロジェクトを起動します。以下のウィンドウが表示されます。

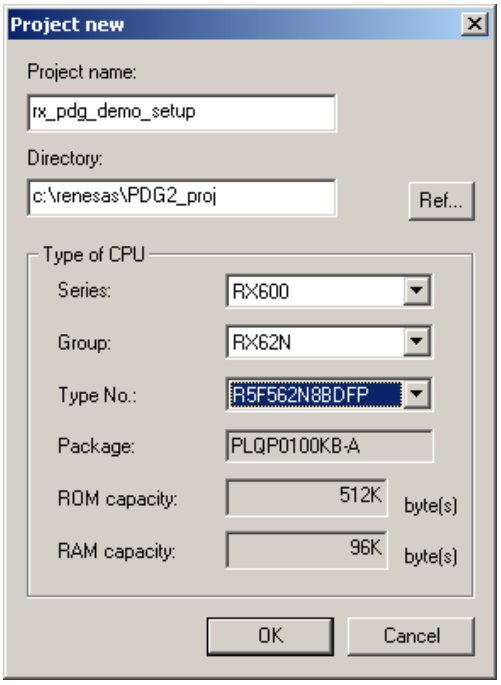


図 4.4　PDG 新規プロジェクトウィンドウ

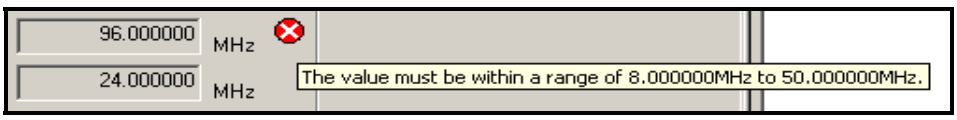
ステップ 4.5　　ウィンドウを図 4.4 と同じ構成内容にして、"OK"をクリックします。

PDG プロジェクトは HEW プロジェクトから分離されているので、同じディレクトリに格納する必要はありません。

ステップ 4.6　　RX 周辺機能を制御するウィンドウが表示されます。編集ペインの下部にあるタブを使用して、現在編集中の周辺機能を切り換えることができます。デフォルトでは、選択された最初のペインは、システムクロックを制御する"SYSTEM"です。上記と同じように、以下のパラメータでクロックをセットアップします。

- EXTAL 端子の入力周波数 12MHz
- システムクロック = 96MHz
- 周辺モジュールクロック = 48MHz
- 外部バスクロック = 24MHz
- 専用 USB クロック = 48Mhz
- BCLK 出力を無効にする

仮に PCLK を 96MHz に設定してみます。PDG はこの設定が不可であることを示す赤色のエラーマークを表示します。エラーマークにマウスを移動すると、エラーの原因と有効な設定値が表示されます。また、周辺機能にエラーがあるたびに、PDG は周辺機能タブに赤色のエラーマークを表示します。



ステップ 4.7　次に、'ADa'タブに切り換え、RPDL と同じように以下の設定で AD1 を設定します。終了すると、ウィンドウは図 4.5 のようになります。

- ユニット 1（AD1）
- チャネル AN4
- シングルモード
- LSB 詰め
- TMR0 コンペアマッチで起動
- 48MHz 変換クロック
- サンプリング時間 0.6us（ハードウェアマニュアルでは、最小値は 0.5us と記載）
- 変換が終了したら関数 ADC_Callback()を呼び出す。
- IPL 4 を使用する。

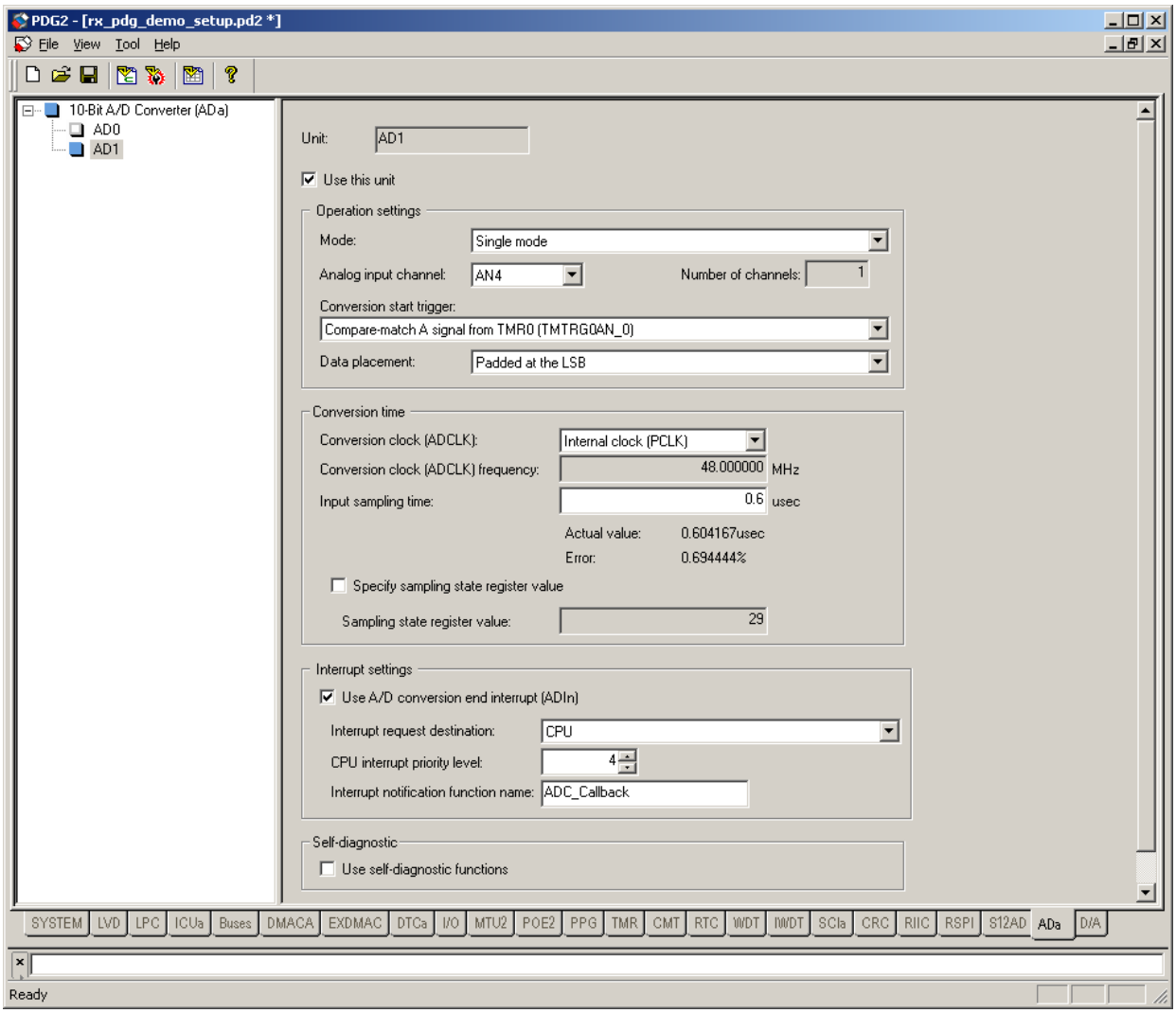


図 4.5　PDG による AD1 の構成

ステップ 4.8　　TMR タブに切り換え、以下の設定で構成します。

- ユニット 0（TMR0 と TMR1 をカスケード接続し 16 ビットカウンタとします。）
- 8192 分周した PCKL を使用する。
- 周波数を 4Hz に設定する（250ms 周期）
- コンペアマッチ A をクリアする。
- AD コンバータを起動
- 50%デューティサイクル
- コールバック関数なし

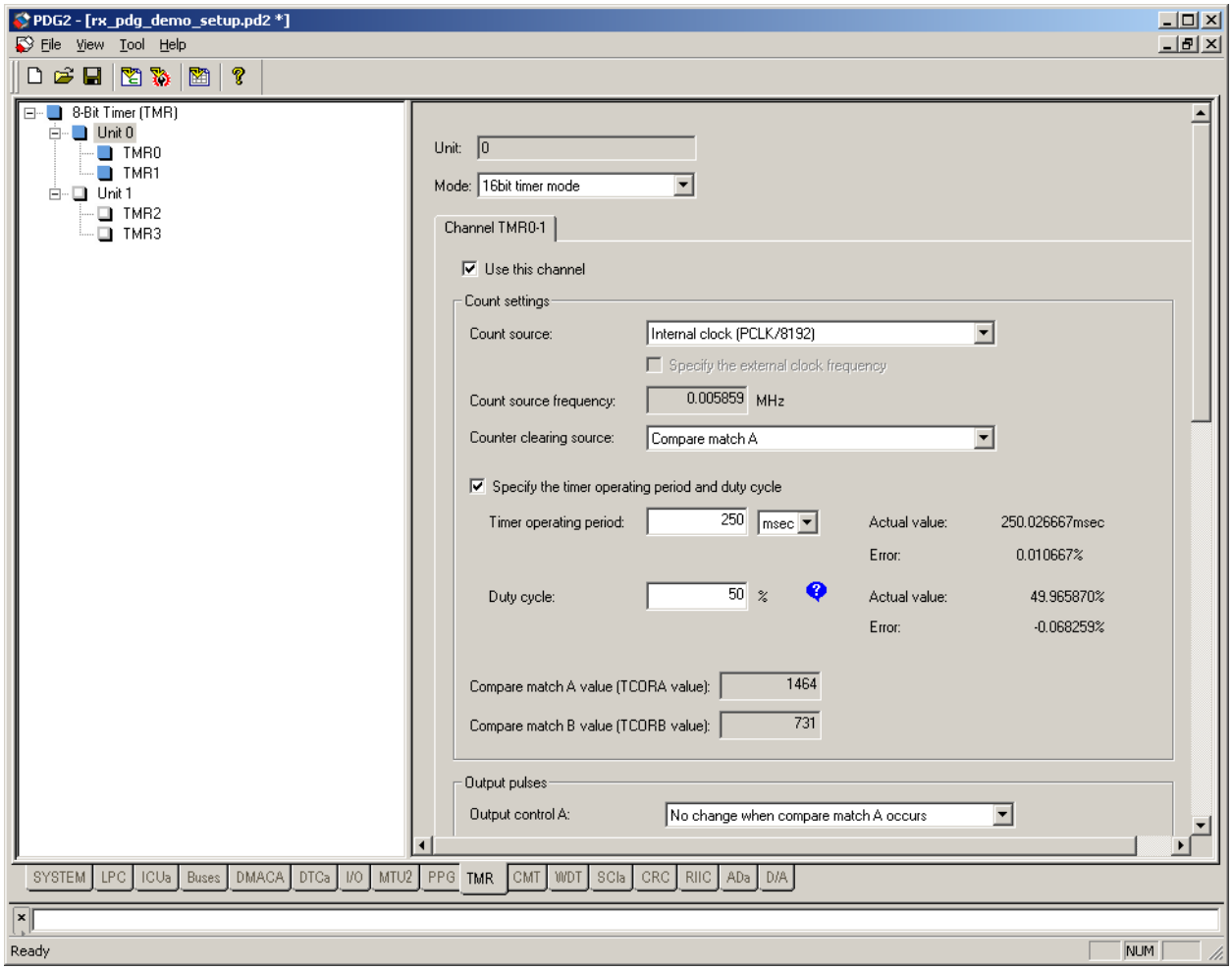


図 4.6　PDG による 8 ビットタイマの構成

TMR **ペインの下部にあるボックスをチェックして必ず** ADC **トリガを有効にしてください。**

ステップ 4.9　　File ⇒ Save の順に選択してプロジェクトを保存します。

ステップ 4.10　　HEW を開き、第 2 章と同様に新しいワークスペースを作成します。ただし、今回は "RX_PDG_Demo"という名前にして､HEW がワークスペースを作成した後に RPDL を追加しません（ステップ 2.1～2.11 を繰り返します）。

ステップ 4.11　第 3 章の始めに行ったように以下のファイルをプロジェクトにコピーします。

- glyph_api.h
- Glyphlib_v2.lib
- lcd.c
- lcd.h
- lcd_utilities.c
- lcd_utilities.h
- YRDKRX62N.h
- YRDKRX62N_RSPI_API.c
- YRDKRX62N_RSPI_API.h

ステップ 4.12　ファイル lcd.c、lcd_utilities.c、YRDKRX62N_RSPI_API.c および Glyphlib_v2.lib をカレントプロジェクトに追加します。

ステップ 4.13　コードを生成して、HEW に登録することができます。HEW で手動でファイルを登録することができますが、PDG は HEW ターゲットサーバを使用してこのサービスを自動的に実行します。最初にすることは、ソースファイルを生成することです。PDG で、Tool ⇒ Generate source files の順に選択します。正常に完了したことを示すウィンドウが表示されます。"OK"をクリックして、閉じます。

PDG を使用してソースファイルを生成する場合は、PDG プロジェクトフォルダの下に格納されます。HEW で登録した後でも、移動されません。これは、複数のプロジェクトでソースファイルを使用している場合、常に最新のファイルを保持することができるので便利です。

ステップ 4.14　次のステップでは、生成されたファイルと RPDL を HEW プロジェクトに登録します。登録するには、Tool ⇒ Register source files in HEW project の順にクリックします。

ステップ 4.15　任意のワークスペースが開いたことを確認するウィンドウが表示されます。"OK"をクリックします。

ステップ 4.16　ライブラリをリンクする順序を設定することを求めるウィンドウが表示されます。このデモでは 1 つのライブラリのみを使用しているので、順序を設定する必要はありません。"OK"をクリックします。

ステップ 4.17　ファイルが正しく登録されたことを示すウィンドウが表示されます。"OK"をクリックして、閉じます。

ステップ 4.18　HEW に戻り、以下のように変更されていることを確認します。

- PDG からの生成されたファイルを含む新しいフォルダ"AddFromPDG"があります。
- ファイル intprg.c および vecttbl.c は自動的に除外されています（以前のプロジェクトでは、これは手動で実行しなければなりませんでした）。
- ツールチェーンオプションウィンドウの Link/Library タブを見ると、RPDL ライブラリが登録されていることがわかります。（PDG では、この MCU の RPDL ライブラリの名前は RX62N_library_LQFP_100.lib となっています。）

ステップ 4.19　ファイル YRDKRX62N_RSPI_API.c に RPDL 関数呼び出しを使用するには、"include"ディレクトリを追加する必要があります。Build ⇒ RX Standard Toolchain の順に選択します。

ステップ 4.20　"C/C++"タブが選択されていることを確認し、"Show entries for"ドロップダウンを"Include file directories"に変更します。

ステップ 4.21　"Add"をクリックします。表示されるウィンドウで、"Relative to"ドロップダウンを"Custom directory"に変更します。

ステップ 4.22 "Browse"をクリックして、"C:¥Renesas¥PDG2¥source¥RX¥RX62N¥i_src"および click "Select" に移動します。"OK"をクリックします。

ステップ 4.23 "Show entries for"ドロップダウンを"Defines"に変更します。

ステップ 4.24 "Add"ボタンをクリックします。表示されるウィンドウで、"Macro"エントリに "DEVICE_PACKAGE_LQFP_100"を入力し、"Replacement"エントリには何も入力しません。 "OK"をクリックします。

"i_src"パスを追加し、DEVICE_PACKAGE_LQFP_100 define を追加した理由は、YRDKRX62N_RSPI_API.c ソースファイル内で RPDL 関数呼び出しを使用することができるようにするためです。以前と同じようにプロジェクトに RPDL に追加することもできますが、共通 RPDL ライブラリを保持するためにこの方法を使用しました。PDG ファイルは PDG ディレクトリの RPDL ライブラリを使用するので、そのディレクトリも使用しました。使用している MCU を選択するために#define を使用しました。以前はコマンドプロンプトでどの MCU を使用しているかを尋ねられたときに RPDL バッチでこれを処理していました。PDG の外部で RPDL を使用していない場合は、これらのステップは必要ありません。

ステップ 4.25 HEW プロジェクトに戻るまで"OK"をクリックします。

ステップ 4.26 PDG によって生成された関数を使用するには、プロジェクトと関連するヘッダファイルを含めなければなりません。RX_PDG_Demo.c を開き、ファイルの先頭にヘッダファイル r_pg_rx_pdg_demo_setup.h を含めます。これを実行しているときに、LCD にアクセスするために以前に使用したプロジェクトのヘッダファイルも含めます。

```
#include "r_pg_rx_pdg_demo_setup.h"

/* RSPI API library support */
#include "YRDKRX62N_RSPI_API.h"
#include "lcd_utilities.h"
#include "lcd.h"
```

PDG ファイルを識別しやすくするために、生成されるすべてのファイルは'R_PG'で始まります。ADC チャネル 1 に対して PDG が生成したコードを見つけるには、ファイル'R_PG_ADC_10_AD1.c'を探してください。

ステップ 4.27 ここで、PDG が生成した関数を呼び出すことができます。これらの関数名を理解するのに最も簡単な方法は、関連するヘッダファイルを開くことです。RPDL プロジェクトで実行したのと同じセットアップ手順に従って、最初にシステムクロックをセットアップします。ヘッダファイル R_PG_Clock.h を開きます。

ステップ 4.28 3 つの関数があり、その 1 つは R_PG_Clock_Set()という名前が付けられています。その他の関数（R_PG_Clock_Start_SUB()および R_PG_Clock_Stop_SUB()）もありますが、その他の関数がサブクロックと関連することを識別するのは簡単です。RX_PDG_Demo.c 内の main()関数呼び出しを探して、関数を call to R_PG_Clock_Set()に追加します。

ステップ 4.29 ファイル R_PG_Clock.c を開き、関数 R_PG_Clock_Set()を見つけます。以前に第 2 章で作成した同じ RPDL 呼び出しを使用していることを確認します。

ステップ 4.30 引き続き PDG 関数を使用してプロジェクトをビルドし、ADC と TMR をセットアップします。

ステップ 4.31 PDG 関数の後に関数呼び出しを追加して、YRDKRX62N_RSPI_Init()および InitialiseLCD()関数呼び出しを使用して RSPI バスと LCD を初期化します。また、main()関数の終わりに無限ループを追加します。

```
void main(void)
{
    R_PG_Clock_Set();

    R_PG_ADC_10_Set_AD1();
    R_PG_Timer_Start_TMR_U0();

    /* Initialise the LCD display on RSPI bus */
    YRDKRX62N_RSPI_Init( 0 );

    /* Setup LCD */
    InitialiseLCD();

    while(1);
}
```

ステップ 4.32 ステップ 3.21 で使用された ADC_Callback()関数をコピーし、RX_PDG_Demo.c に追加します。

ステップ 4.33 今回は R_ADC_10_Read() RPDL 関数を使用する代わりに、R_PG_ADC_10_AD1.h にある R_PG_ADC_10_GetResult_AD1() PDG 関数を使用します。

ステップ 4.34 Build ボタン  をクリックするか、Build ⇒ Build All の順に選択します。

ステップ 4.35 第 3 章の手順に従ってデバッグセッションを変更し、ボードに接続します。

ステップ 4.36 MCU にコードをダウンロードして、Reset-Go  をクリックするか、Debug ⇒ Reset Go の順にクリックします。

ステップ 4.37 コードが第 3 章と同じように実行されることを確認します。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/inquiry

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2011.09.27	—	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部 ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル） (03)5201-5307

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/inquiry

Colophon 1.0