RX600 シリーズ用リアルタイム OS RI600/4

リアルタイム OS　入門編

R20AN0024JJ0100
Rev.1.00
2010/12/10

本アプリケーションノートは、リアルタイム OS RI600/4 を初めてご使用になられるユーザのための入門ガイドです。

目次

1. 概要

本アプリケーションノートは、RX610 を搭載した CPU ボードをターゲットとして、リアルタイム OS RI600/4 を使用したシステムの開発手順を解説するものです。

簡単なシステムのサンプルを使って、コンフィギュレーションやロードモジュールの作成、プログラムの動作確認を説明します。

1.1 開発環境

本アプリケーションノートでは、下記の開発環境を使用しています。

1.1.1 ハードウェア

・ホスト PC

・RX610 スタータキット　CPU ボード

・E1 エミュレータ

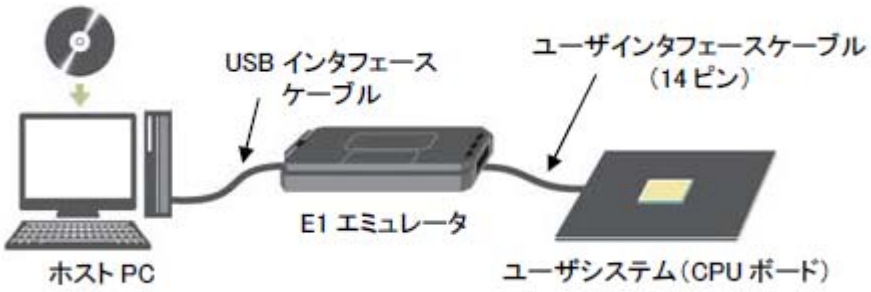


図 1－1　ハードウェア構成

1.1.2 ソフトウェア

・リアルタイム OS　RI600/4　（V.1.00 Release 02）

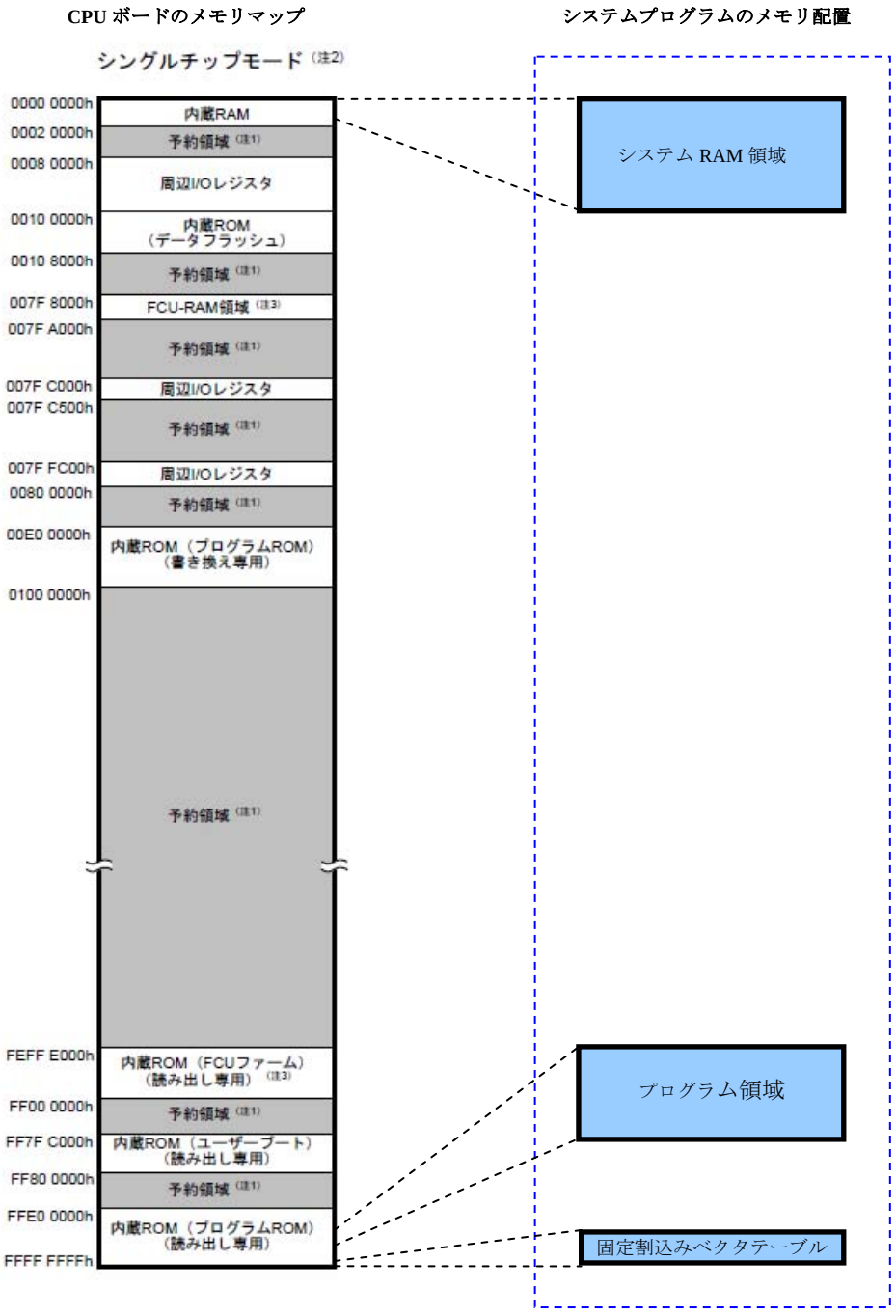
・GUI コンフィギュレータ　（Ver. 1.00.00.002）

・統合開発環境　High-performance Embedded Workshop（Version 4.07.00.007）　以下、HEW と略します。

・RX610 スタータキット付属のサンプルプログラム　（RSKRX610-100222 v0100）

1.2 メモリマップ

CPU ボードのマイコンボードのメモリマップを以下に示します。



注 1.予約領域は、アクセスしないでください。
注 2.ブートモード、ユーザーブートモードは、シングルチップモードと同じアドレス空間となります。
注 3.FCU についての詳細は、RX610 グループハードウェアマニュアル「26. ROM（コード格納用フラッシュメモリ）」、「27. データフラッシュ（データ格納用フラッシュメモリ）」をご参照ください。

図 1−2 メモリマップ

サンプルプログラムのメモリ配置を以下に示します。

表1－1　サンプルプログラムのメモリ配置

No.	名称 （配置アドレス）	セクション名	内容
1	システム RAM 領域 （0x00000000）	SI	システムスタック
2		SURI__STACK	タスクのスタック
3		B*	未初期化データ領域
4		R*	初期化データ領域
5	プログラム領域 （0xFFFF0000）	INTERRUPT_VECTOR	可変割込みベクタテーブル
6		P*	プログラム領域 （サンプルプログラムが配置されます。）
7		C*	定数領域
8		D*	初期化データ領域
9		W*	switch 文分岐テーブル領域 （サンプルでは未使用です。）
10	固定割込みベクタテーブル領域（0xFFFFFF80）	FIX_INTERRUPT_VECTOR	固定割込みベクタテーブル

2. 開発手順

2.1 概要

RI600/4 のシステム開発では、HEW と GUI コンフィギュレータを使用します。
システムの開発手順を以下に示します。

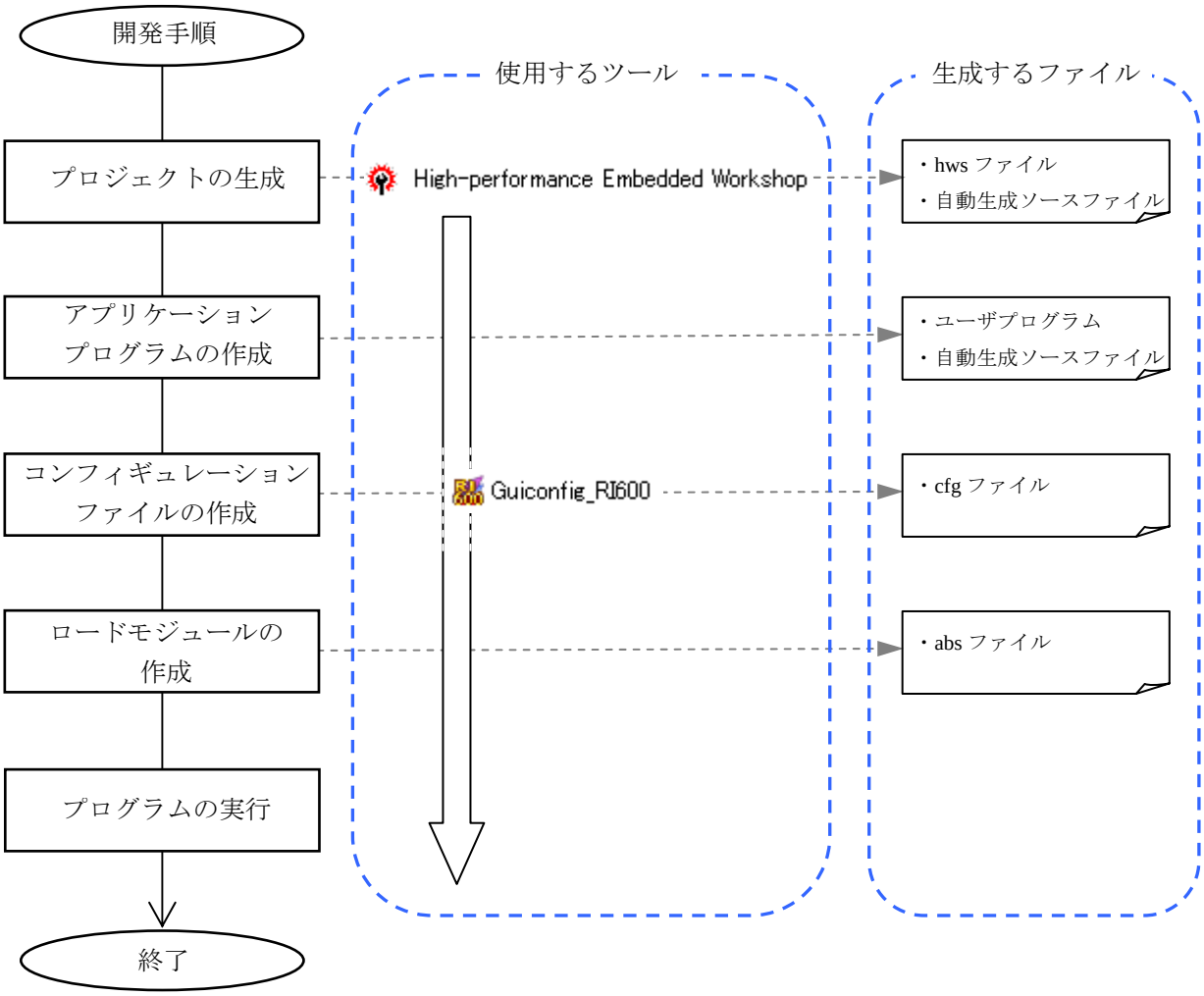


図2－1　システムの開発手順

(1) プロジェクトの生成

HEW を使用して、RI600/4 用のプロジェクトを生成します。

プロジェクトを生成すると、ワークスペースファイル（hws ファイル）が生成されます。

また、マイコンに対応したソースファイルがサンプルとして自動生成されます。自動生成されたファイルは必要に応じて変更する必要があります。

(2) アプリケーションプログラムの作成

タスクやハンドラとして動作するプログラムをコーディングします。

コーディングは HEW のエディタウィンドウやテキストエディタを使用します。

また、プロジェクトの生成で HEW が自動生成したサンプルのソースファイルをシステムに合わせて変更します。

(3) コンフィギュレーションファイル（cfg ファイル）の作成

GUI コンフィギュレータを使用して、コンフィギュレーションファイル（cfg ファイル）を作成します。

コンフィギュレーションファイルにはタスクやオブジェクトの各種情報を入力します。

この入力情報は、コンフィギュレータ設定ファイル（hcf ファイル）として保存することができます。

コンフィギュレーションファイルはテキストエディタにより直接作成することも可能ですが、本アプリケーションノートでは、GUI コンフィギュレータを使用します。

(4) ロードモジュールの作成

HEW のビルド機能を使用して、ロードモジュールを作成します。ビルドが正常に行われると、ロードモジュールとして abs ファイルを生成します。

(5) プログラムの実行

HEW のデバッグ機能を使用して、ロードモジュールを実行します。

ロードモジュールを実行する場合は、ターゲットシステムを接続して、ターゲット上にダウンロードする必要があります。

2.1.1 プロジェクトの生成

HEW を起動し、メニューに従いプロジェクトを作成します。

(1) HEW の起動

[スタート]メニューから HEW を起動します。

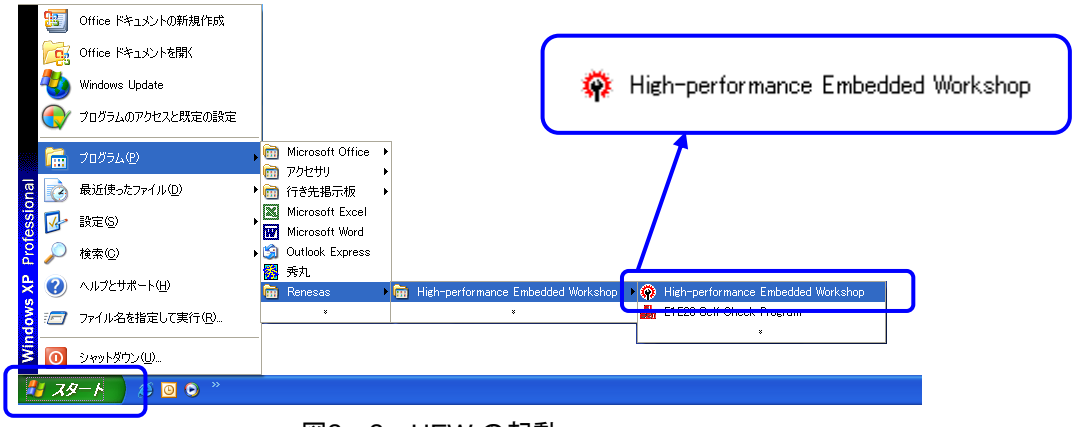


図2－2 HEW の起動

(2) 新規プロジェクトの作成

[ようこそ!]ダイアログボックスで、[新規プロジェクトワークスペースの作成]を選択し、[OK]をクリックします。

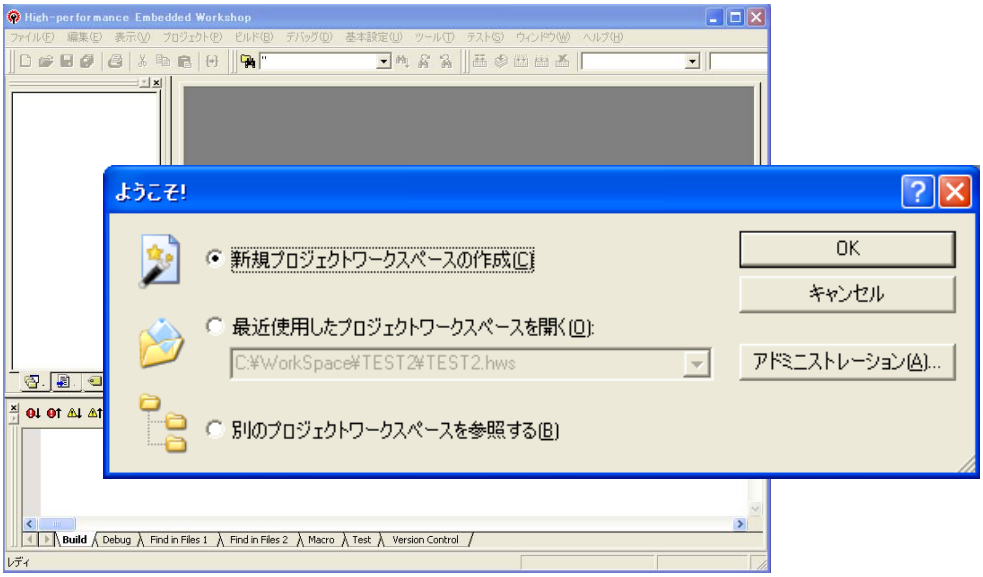


図2－3 新規プロジェクトワークスペースの作成

(3) ワークスペースの指定

ワークスペースの情報を指定します。

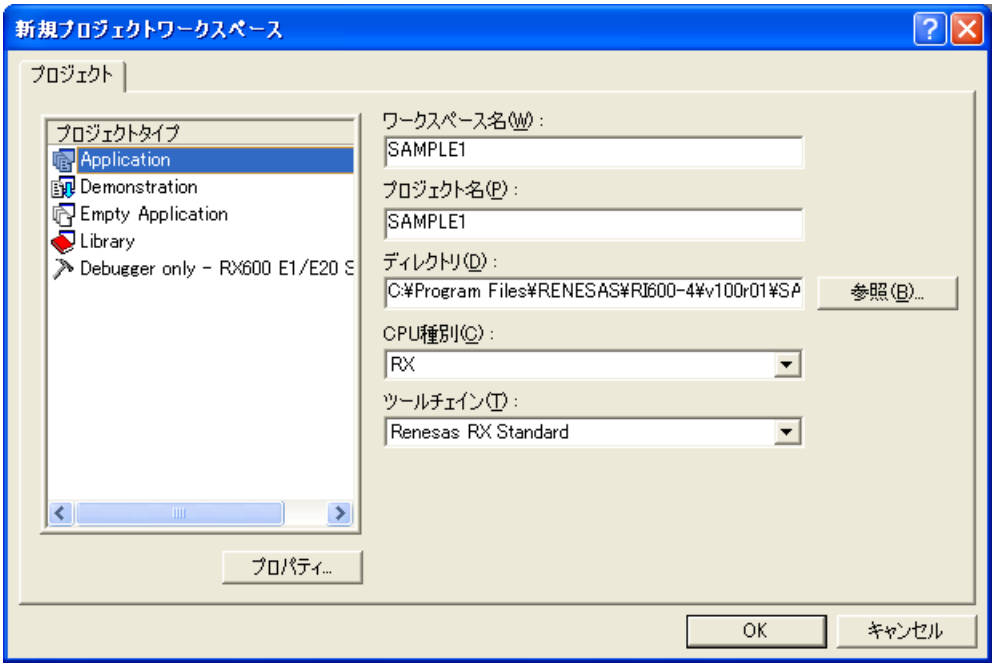


図2－4　プロジェクトの生成（1/11）

[ワークスペース名]	ワークスペース名を入力します。ここでは例として「SAMPLE1」とします。
[プロジェクト名]	プロジェクト名を入力します。ワークスペース名と同じで良い場合は、入力する必要はありません。
[ディレクトリ]	ワークスペースを作成するディレクトリを指定します。ここでは RI600/4 カーネルのインストールディレクトリを指定します。
[CPU 種別]	“RX”を選択します。
[ツールチェイン]	“Renesas RX Standard”を選択します。

(4) CPU の指定

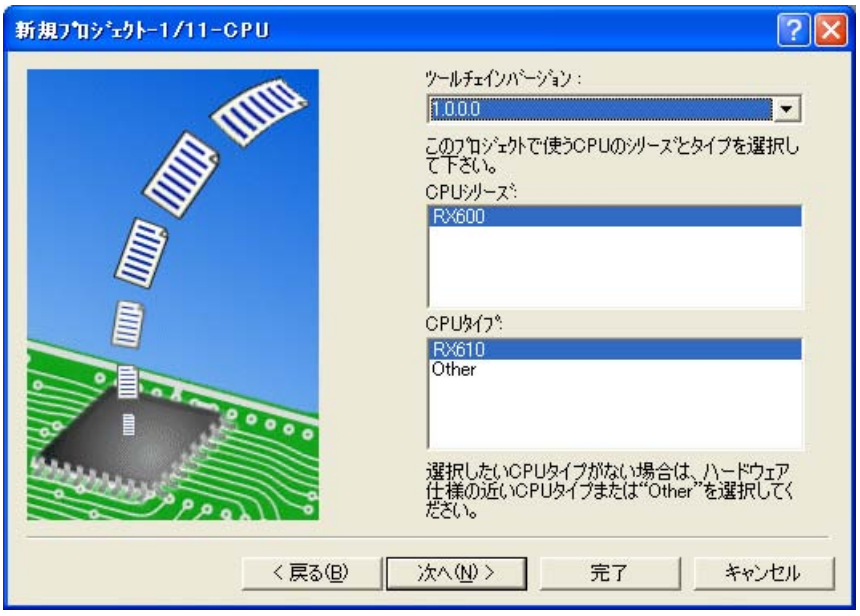


図2－4　プロジェクトの生成（2/11）

[ツールチェインバージョン]	“1.0.0.0”を選択します。
[CPU シリーズ]	“RX600”を選択します。
[CPU タイプ]	“RX610”を選択します。

(5) RTOS の指定

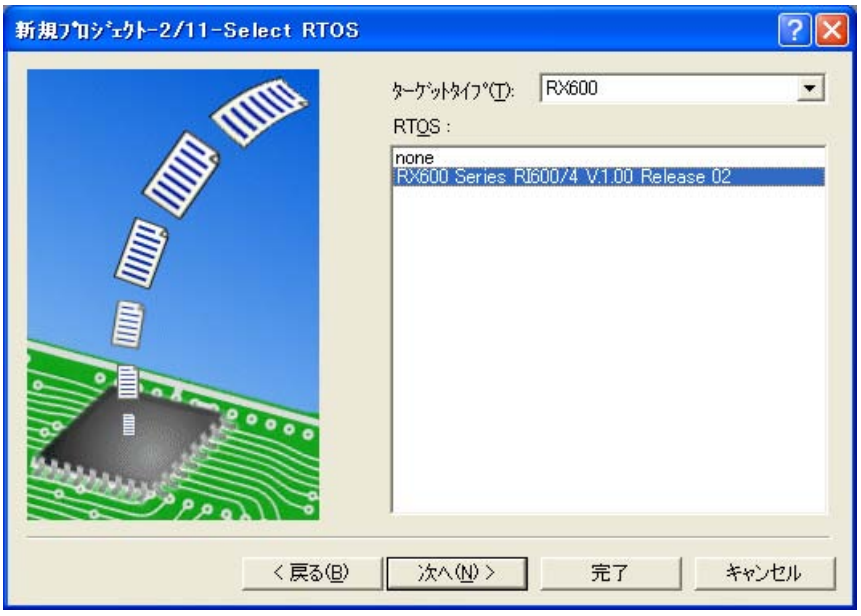


図 2－4　プロジェクトの生成（3/11）

[ターゲットタイプ]	“RX600”を選択します。
[RTOS]	“RX600 Series RI600/4 V.1.00 Release 02”を選択します。

(6) オプションの指定

図 2－4　プロジェクトの生成（4/11）

[エンディアン]	“Little”を選択します。 本アプリケーションノートでは、リトルエンディアンの指定で説明します。
その他の項目はデフォルトのままの指定です。	

(7) オプション(その２)の指定

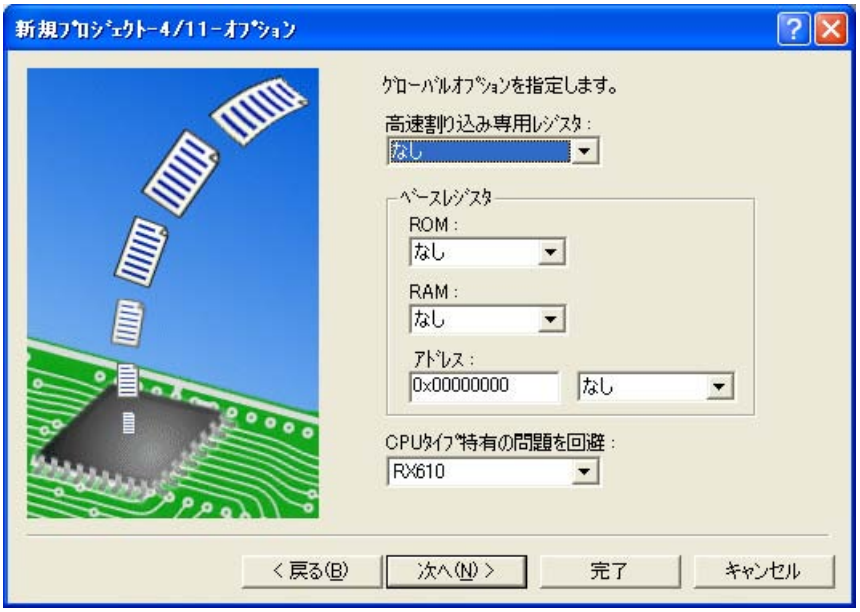


図 2－4　プロジェクトの生成（5/11）

この[オプション(その２)]はデフォルトのままの指定です。

(8)　生成ファイルの指定

図 2−4　プロジェクトの生成（6/11）

[ハードウェアセットアップ関数生成]	“C/C++ source file”を選択します。
その他の項目はデフォルトのままの指定です。	

(9)　標準ライブラリの指定

図 2−4　プロジェクトの生成（7/11）

この[標準ライブラリ]はデフォルトのままの指定です。

(10) ターゲットの指定

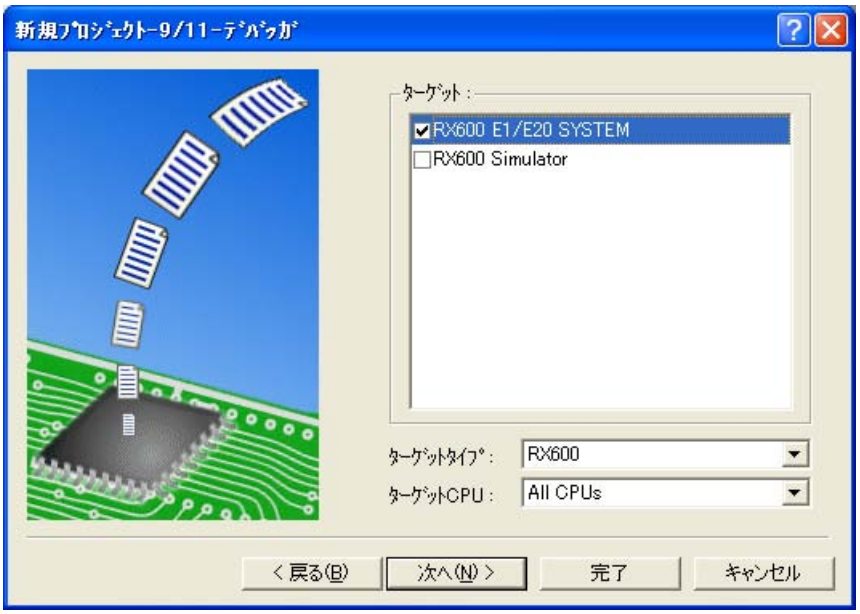


図 2−4　プロジェクトの生成（8/11）

[ターゲット]	“RX600 E1/E20 SYSTEM”を選択します。
その他の項目はデフォルトのままの指定です。	

(11) ターゲット名の指定

図 2−4　プロジェクトの生成（9/11）

この[デバッガオプション]はデフォルトのままの指定です。

(12) 生成ファイルの確認

HEW が自動生成するファイルの確認をして、[完了]をクリックします。

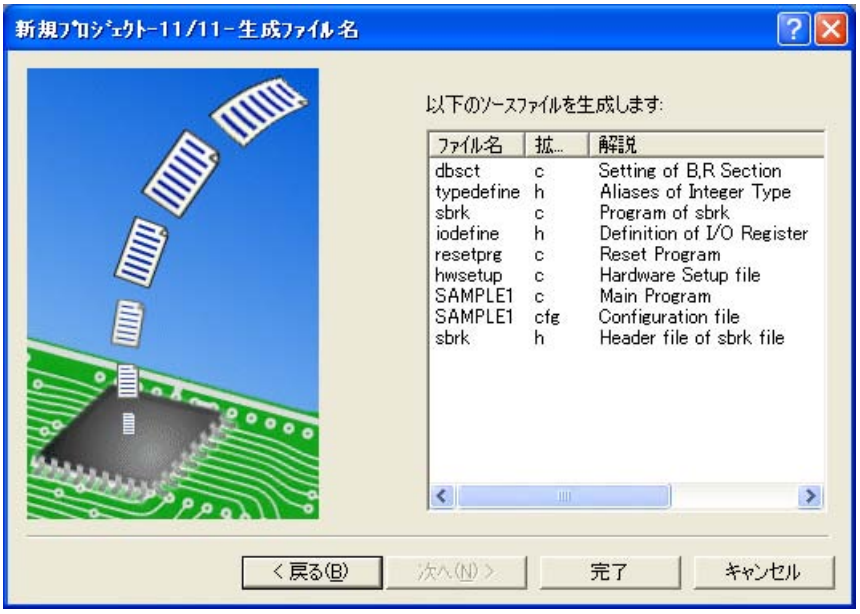


図 2－4　プロジェクトの生成（10/11）

(13) プロジェクト概要の確認

プロジェクトの概要を確認して、[OK]をクリックします。

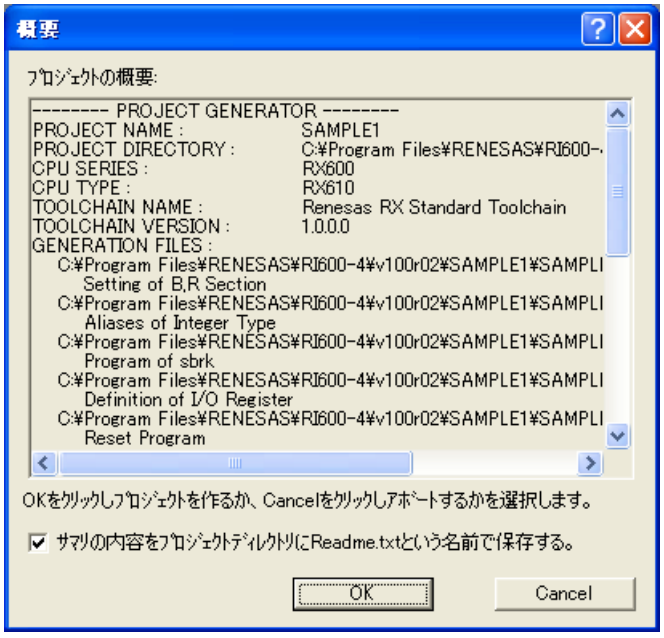


図 2－4　プロジェクトの生成（11/11）

以上でプロジェクトが生成され、表 2-1 に示すようなマイコンに対応したサンプルのソースファイルが自動生成されます。

本アプリケーションノートで使用するサンプルプログラムを動作させるためには、自動生成されたソースファイルを変更する必要があります。

表2－1　HEW の自動生成ソースファイル

No.	ファイル名	説明	サンプル変更内容
1	resetprg.c	リセットプログラム	－
2	hwsetup.c	ハードウェアセットアップ	“RI_hwsetup()”の呼出し追加
3	“ﾌﾟﾛｼﾞｪｸﾄ名”.c	メインプログラム	サンプルプログラムのファイルと入替え
4	“ﾌﾟﾛｼﾞｪｸﾄ名”.cfg	コンフィギュレーション ファイル	サンプルプログラム用 cfg ファイルと 入替え
5	dbsct.c	セクション設定ファイル	－
6	sbrk.c	ヒープ領域管理プログラム	－
7	sbrk.h	ヒープ領域管理ヘッダ	－
8	typedefine.h	typedefine ファイル	－
9	iodefine.h	I/O レジスタ定義ファイル	－

2.1.2 アプリケーションプログラムの作成

タスクやハンドラを、システムに合わせてコーディングします。

プロジェクトの生成時に、HEW はメインプログラムとして“プロジェクト名.c”を自動生成します。

自動生成されたメインプログラムは、必要に応じてコードを変更します。

2.1.3 コンフィギュレーションファイル（cfg ファイル）の作成

GUI コンフィギュレータを使用して、コンフィギュレーションファイル（cfg ファイル）を作成します。

GUI コンフィギュレータで定義した情報は、コンフィギュレータ設定ファイル（hcf ファイル）として保存することができます。

(1) GUI コンフィギュレータの登録

GUI コンフィギュレータを HEW の[基本設定]－[カスタマイズ]－[メニュー]から[ツールの追加]ダイアログで指定します。これにより、GUI コンフィギュレータを HEW の[ツール]から起動することが可能となります。

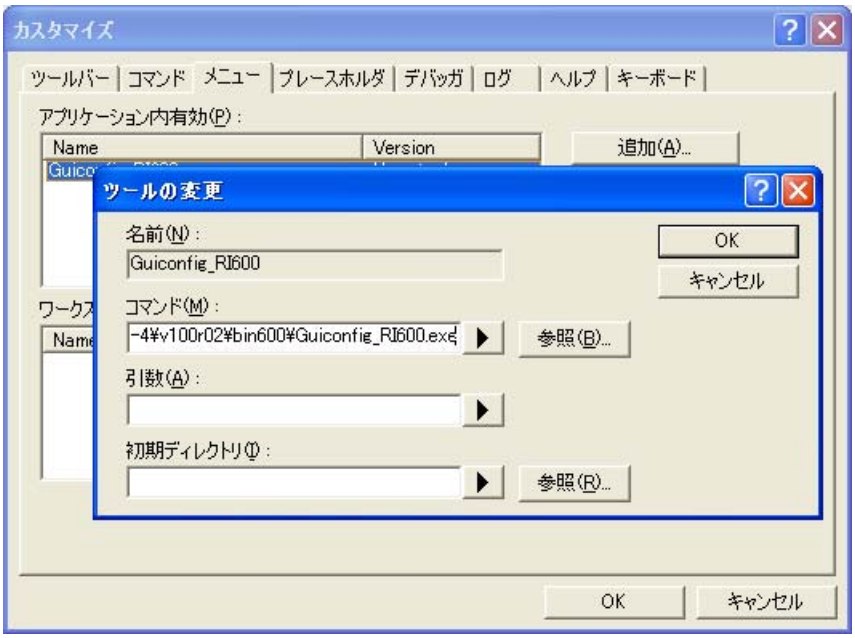


図2－5 GUI コンフィギュレータの登録

[名前]	“Guiconfig_RI600”を指定します。
[コマンド]	RI600/4 カーネルのインストールディレクトリ内の“¥bin600¥Guiconfig_RI600.exe”を指定します。

(2) コンフィギュレーション情報の入力と生成

GUI コンフィギュレータを、HEW の[ツール]から起動し、タスクやハンドラなどの各種情報を入力します。

HEW は、プロジェクトの生成時に“プロジェクト名.cfg”を自動生成しますが、ユーザシステムに合わせて生成したファイルと置き換える必要があります。

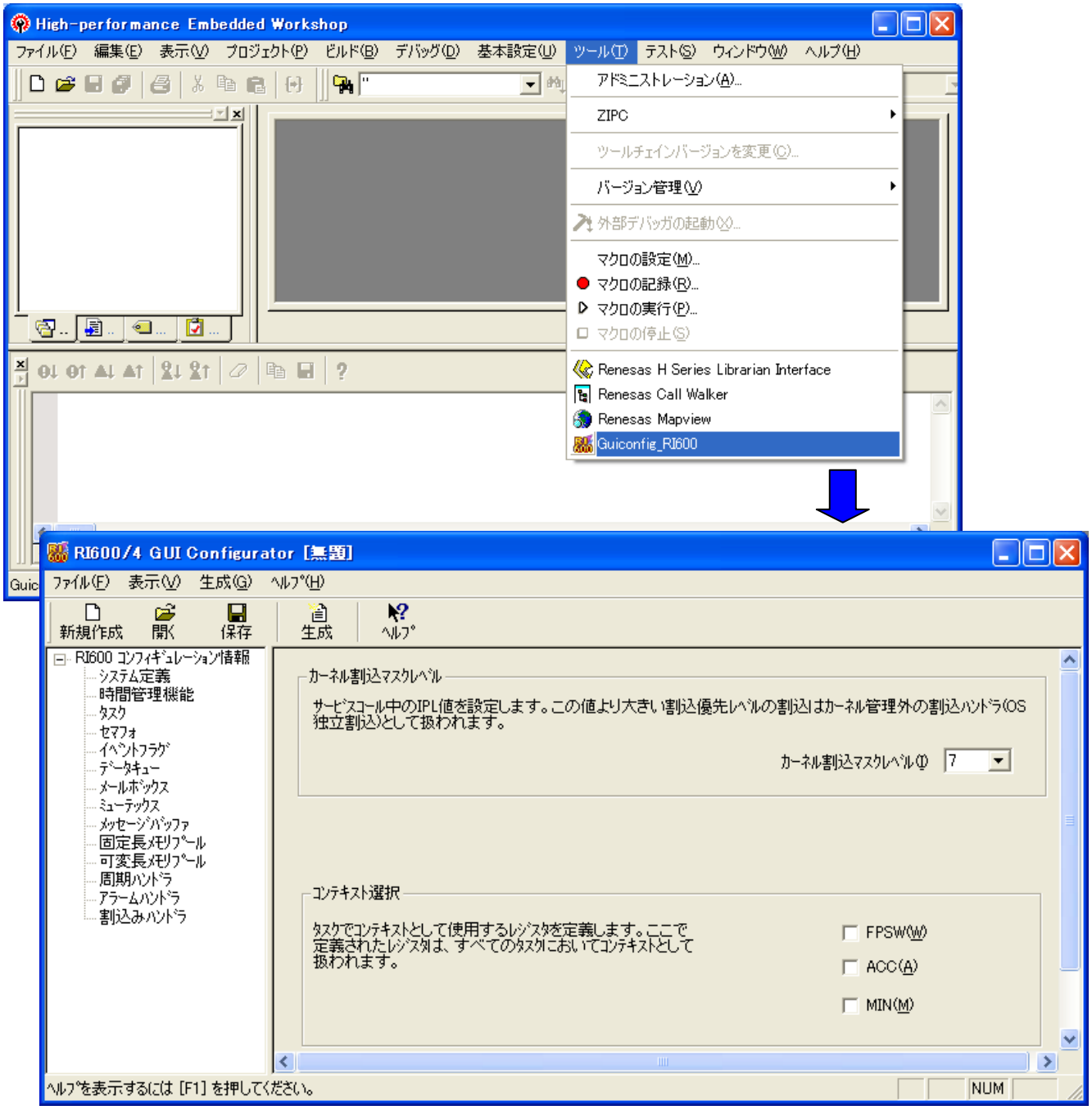


図2－6　GUI コンフィギュレータの起動

(a)　システム定義

システム定義では、[カーネル割込みマスクレベル]や[コンテキスト選択]を指定します。

サンプルプログラムではデフォルトのままの指定です。

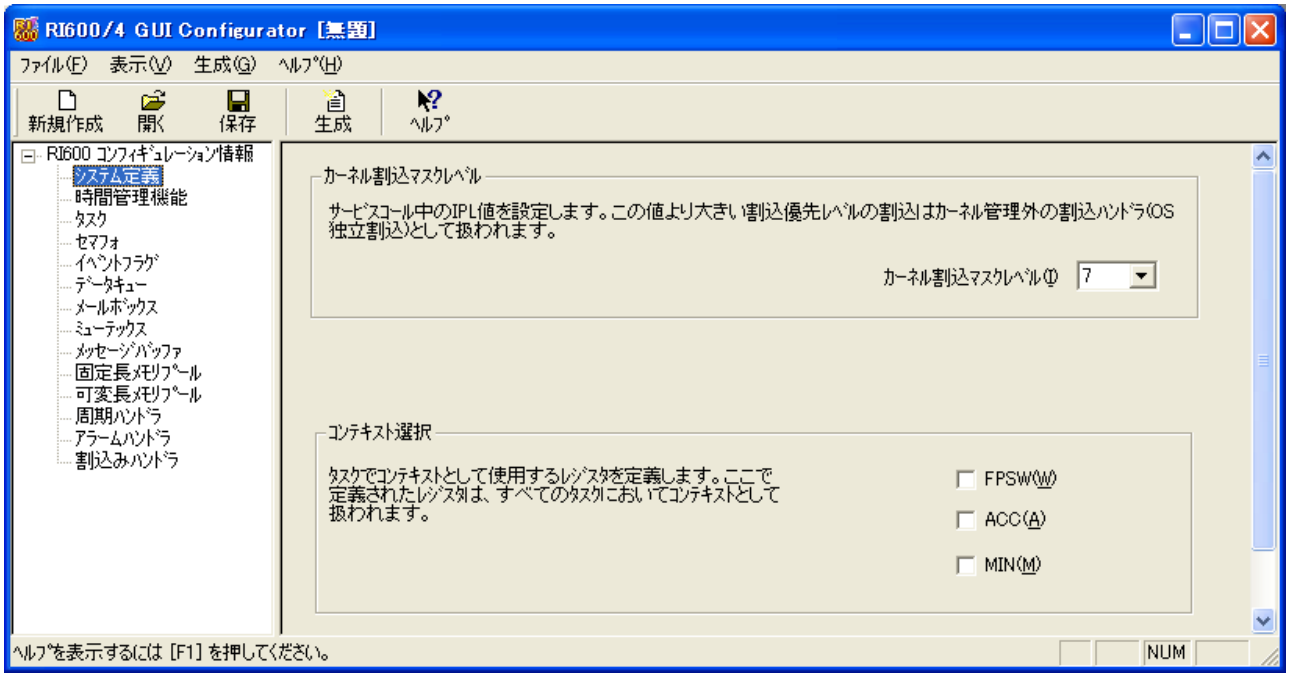


図2－7　GUI コンフィギュレータ　システム定義

(b)　時間管理機能

時間管理機能では、[システムクロック割込み優先レベル]などを指定します。

サンプルプログラムではデフォルトのままの指定です。

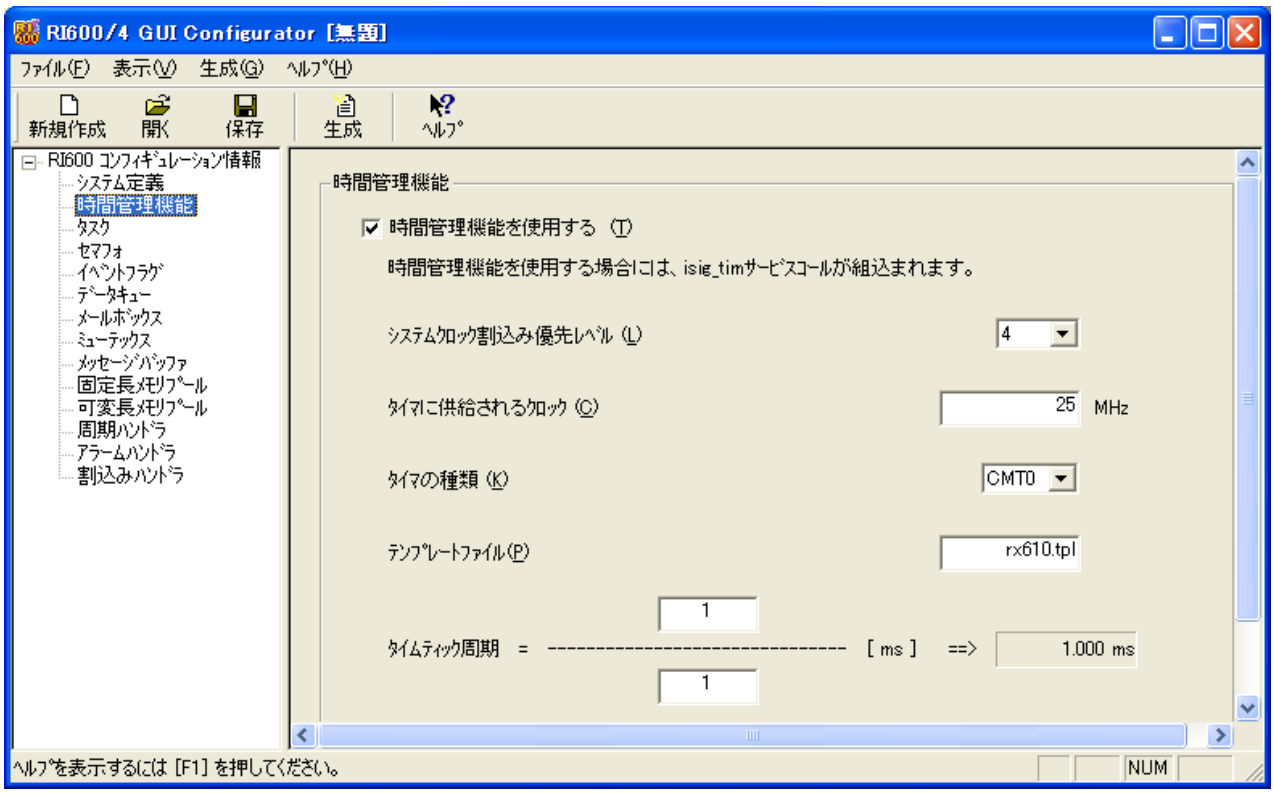


図2－8　GUI コンフィギュレータ　時間管理機能

(c)　**オブジェクトの定義**

システムで使用するタスクやセマフォなどの、オブジェクトの定義を行います。
サンプルプログラムでは、それぞれのシステムに必要な定義を行います。

[タスク]では、タスクの ID 番号やアドレスなど、タスクの生成に必要な情報を入力します。
セマフォやイベントフラグについても、システムに合わせて入力を行います。

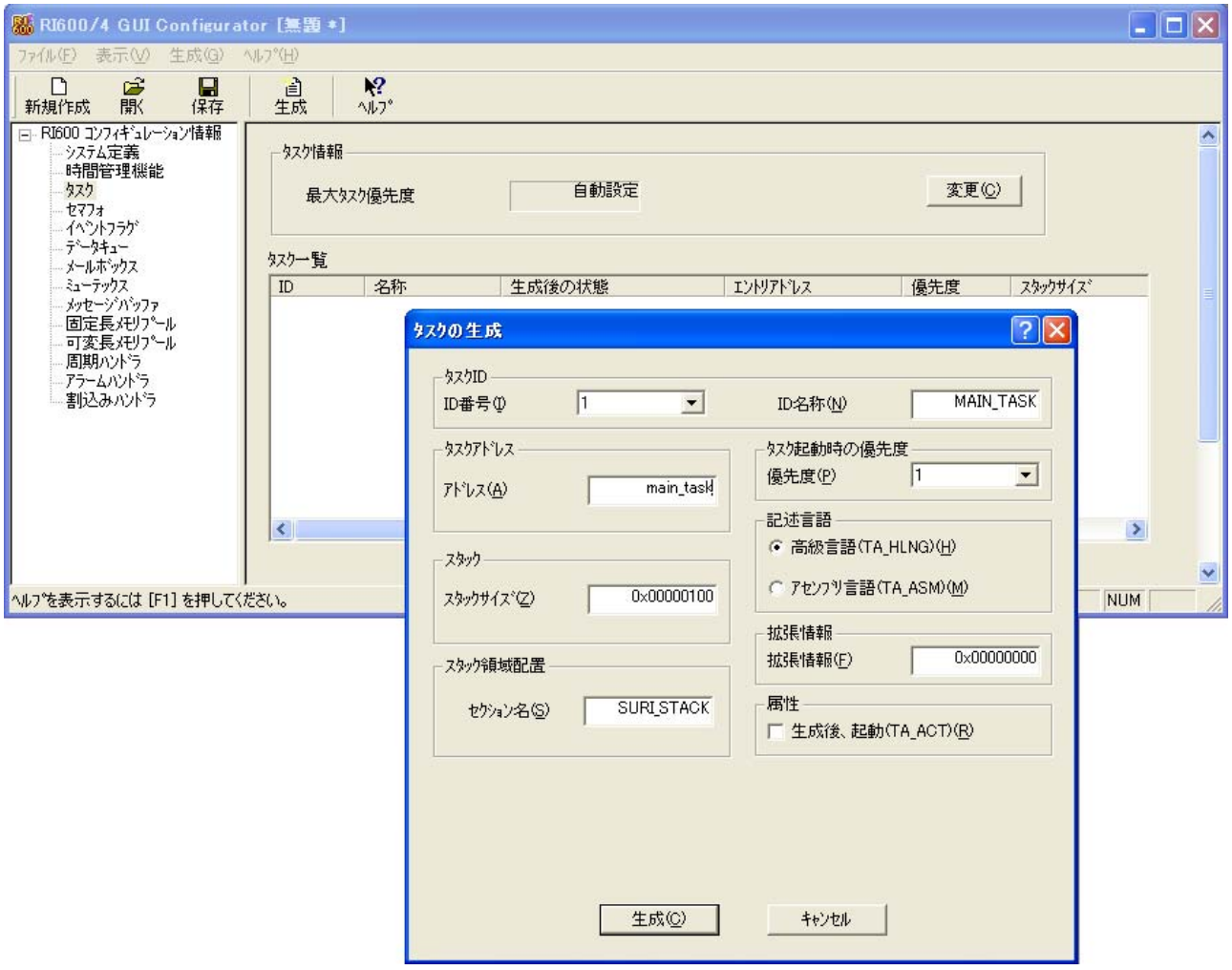


図2－9　GUI コンフィギュレータ　タスクの定義

(d)　割込みハンドラの定義

割込み情報の設定および割込みハンドラの定義を行います。

サンプルプログラムでは、LED の点滅処理や LCD への表示処理のタイミングを図るために、サンプルプログラムのタイマ用割込みハンドラを定義します。

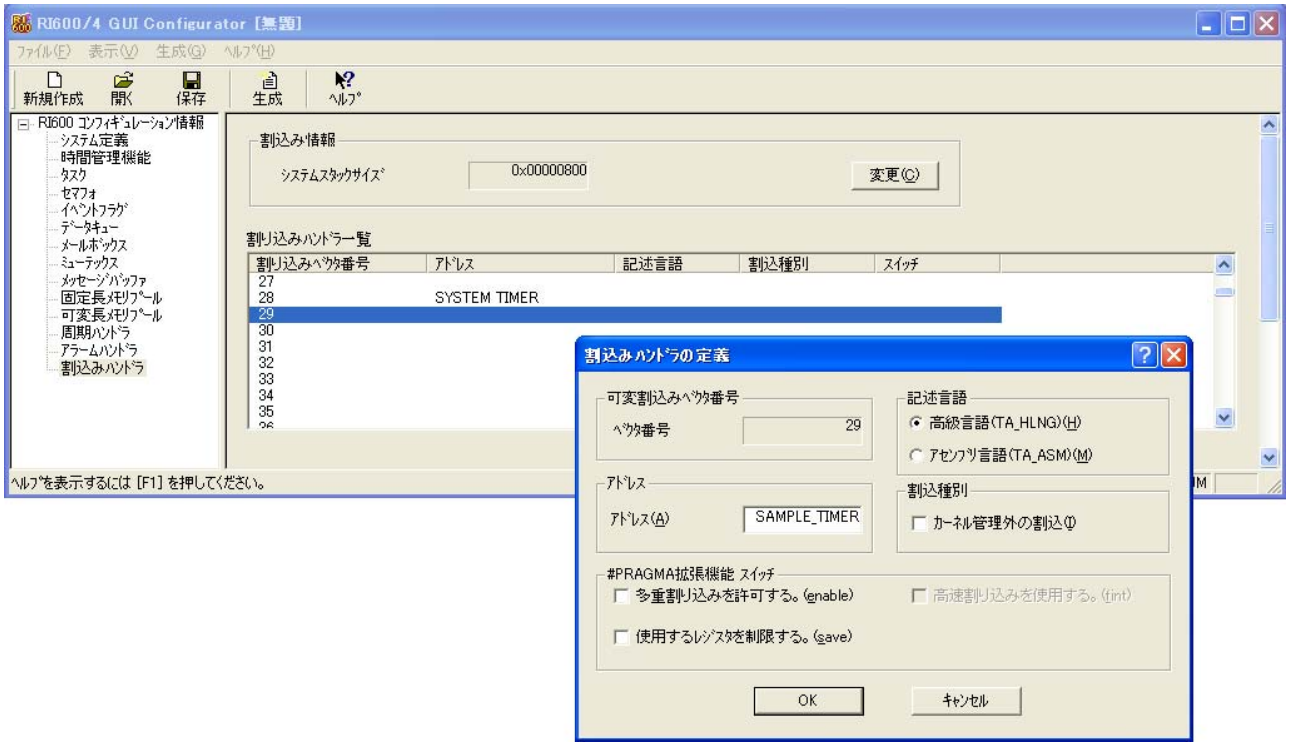


図2－10　GUI コンフィギュレータ　割込みハンドラの定義

[ベクタ番号]	29 を使用します。 サンプルプログラムでは CMT1（コンペアマッチタイマ）を使用します。
[アドレス]	“SAMPLE_TIMER”を指定します。
その他の項目はデフォルトのままの指定です。	

各情報の定義が完了したら、[生成]で cfg ファイルを生成します。

サンプルプログラムでの[コンフィギュレーションファイルの生成場所]は“プロジェクトディレクトリ”を指定します。

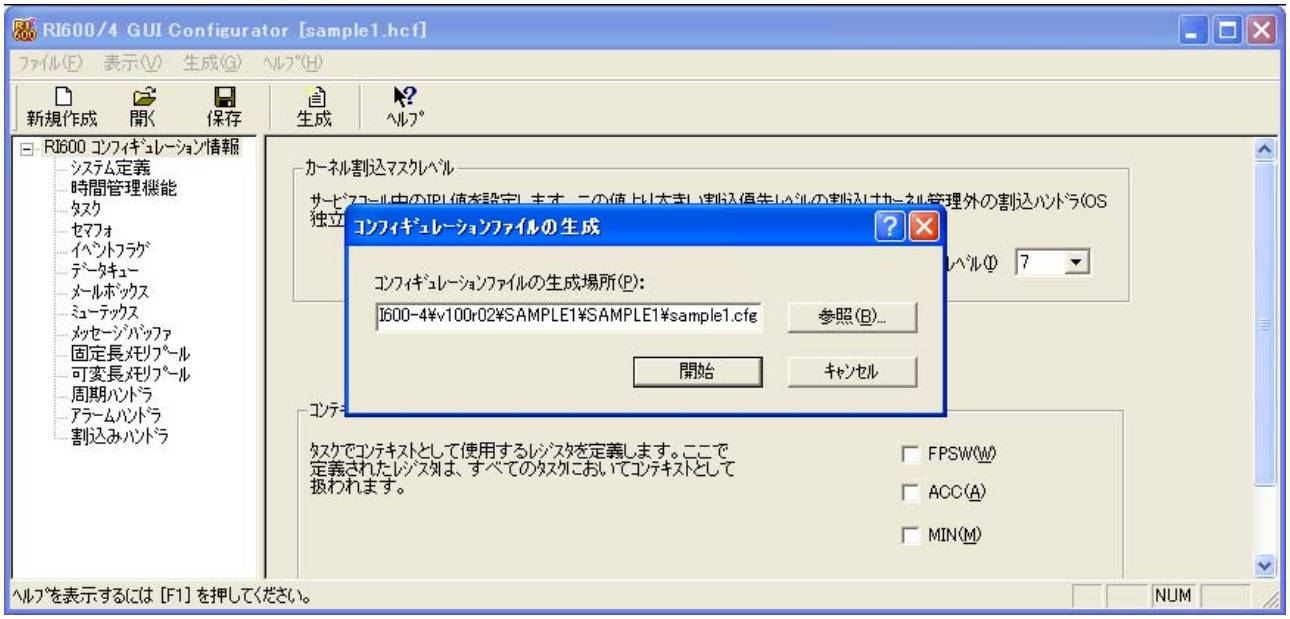


図2－11　コンフィギュレーションファイルの生成

(3)　コンフィギュレーション情報の保存

入力した情報をコンフィギュレータ設定ファイル（hcf ファイル）として保存します。

[保存]で生成場所を指定します。

サンプルプログラムでの[保存する場所]は“プロジェクトディレクトリ”を指定します。

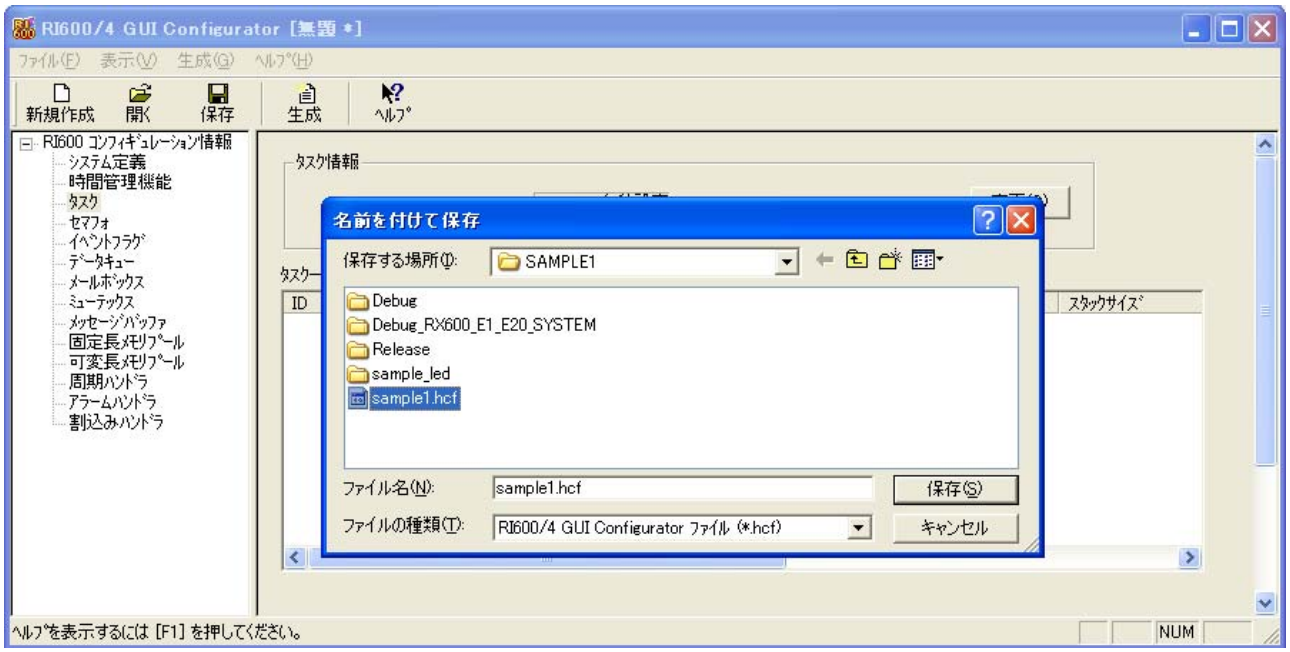


図2－12　コンフィギュレーション情報の保存

2.1.4 ロードモジュールの作成

ロードモジュールを作成するために必要な設定を以下に示します。

(1) アプリケーションプログラムのファイルの追加

アプリケーションプログラムのファイルをプロジェクトに追加します。
[プロジェクト]ー[ファイルの追加]でダイアログを開き、追加するファイルを指定します。

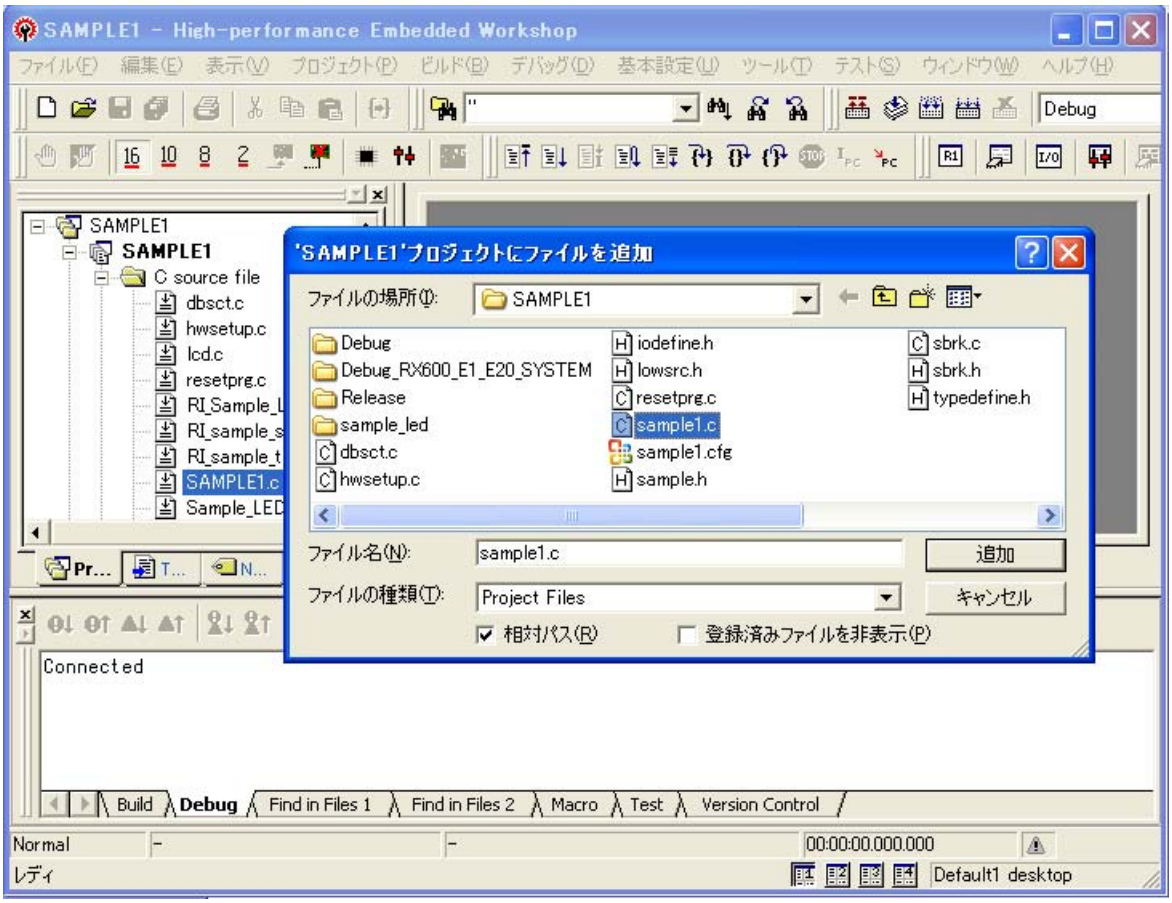


図2ー13 アプリケーションプログラムのファイルの追加

サンプルプログラムでは、表2－2に示すファイルをプロジェクトに追加します。

表2－2　サンプルプログラムで使用するファイル

No.	ファイル名	名称	備考
1	sample?.c	サンプルプログラム	“?”はサンプルの番号に対応しています。 プロジェクト生成時に登録されています。
2	RI_Sample_sub.c　*1	サンプル共通ファイル	各サンプルで使用します。 プロジェクトにファイルを追加します。
3	RI_Sample_LED.c　*1	サンプル LED ファイル	各サンプルで使用します。 プロジェクトにファイルを追加します。
4	Sample_LED.c　*1	スタータキット LED ファイル	スタータキットで提供しているサンプルプログラムです。 プロジェクトにファイルを追加します。
5	lcd.c　*1	スタータキット LCD ファイル	
6	RI_sample.h	サンプルプログラムヘッダファイル	sample?.c でインクルードしています。
7	sample_led.h　*1	スタータキット LED ヘッダファイル	スタータキットで提供しているサンプルヘッダファイルです。
8	lcd.h　*1	スタータキット LCD ヘッダファイル	
9	iodefineRX_LSB.h　*1	スタータキット I/O ヘッダファイル	

【注】*1　スタータキット付属のサンプルプログラムの sample_led フォルダに格納されています。

(2) コンフギュレーションファイルの登録

コンフィギュレーションファイルをプロジェクトに追加します。

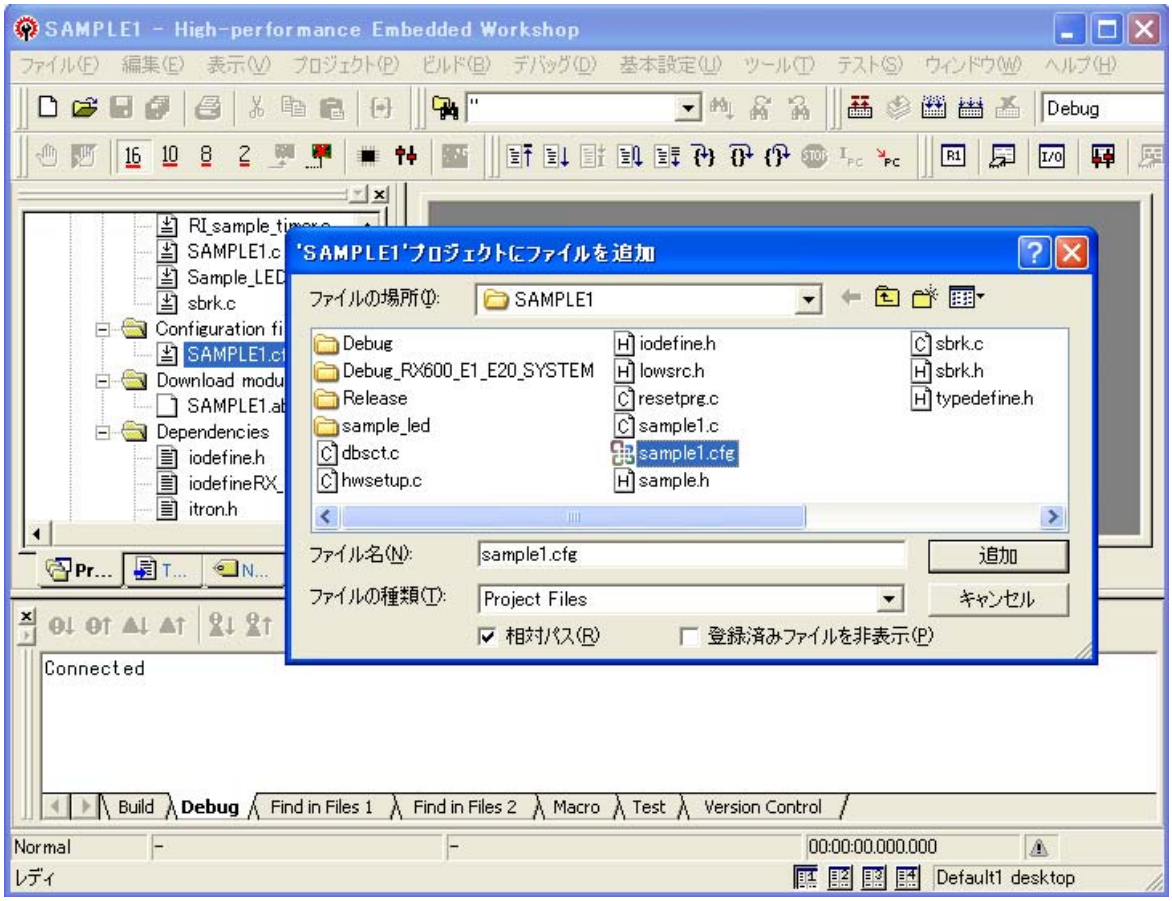


図2－14　コンフギュレーションファイルの追加

(3) ビルドの実行

ビルドを実行します。

ビルドが正常に行われると、ロードモジュール（abs ファイル）が生成されます。

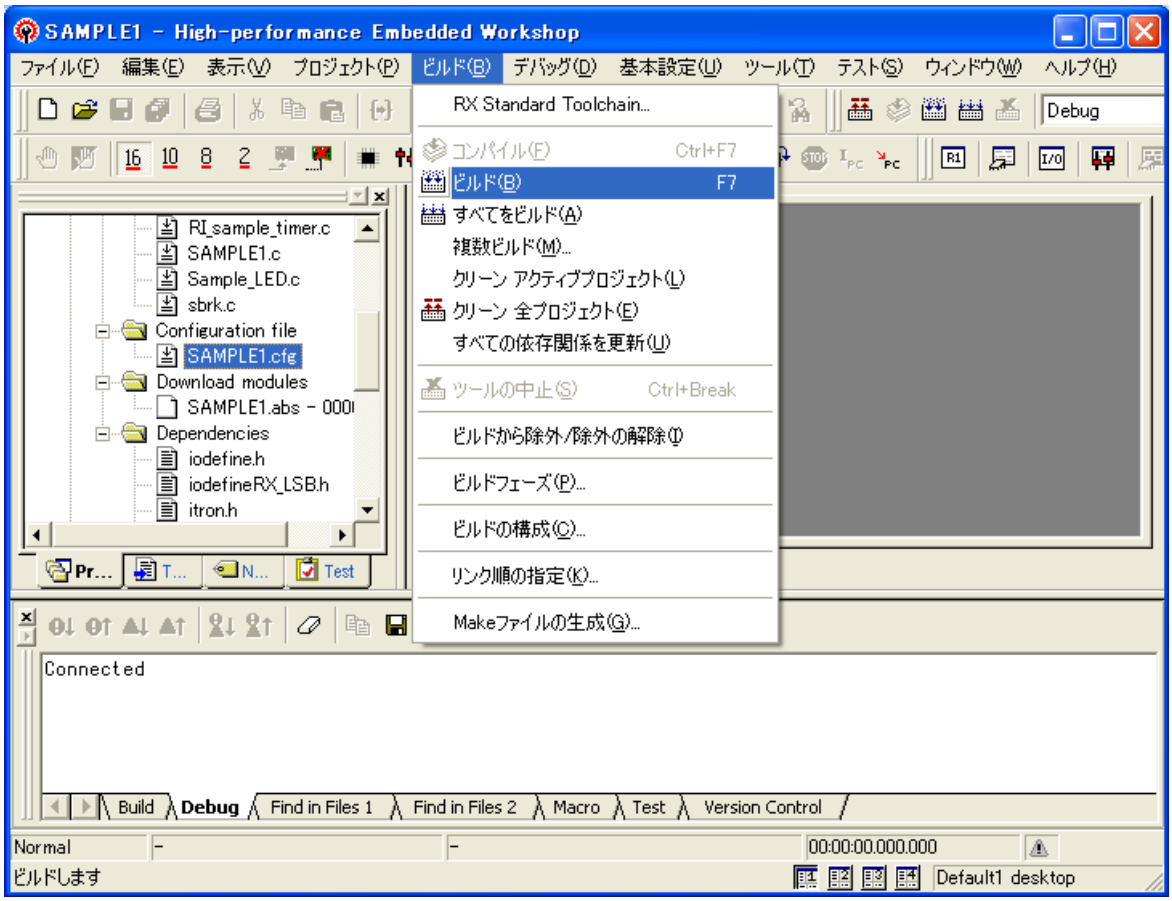


図2－15　ビルドの実行

2.1.5 プログラムの実行

(1) デバッグの設定

プログラムを実行するため、デバッグの設定を行います。

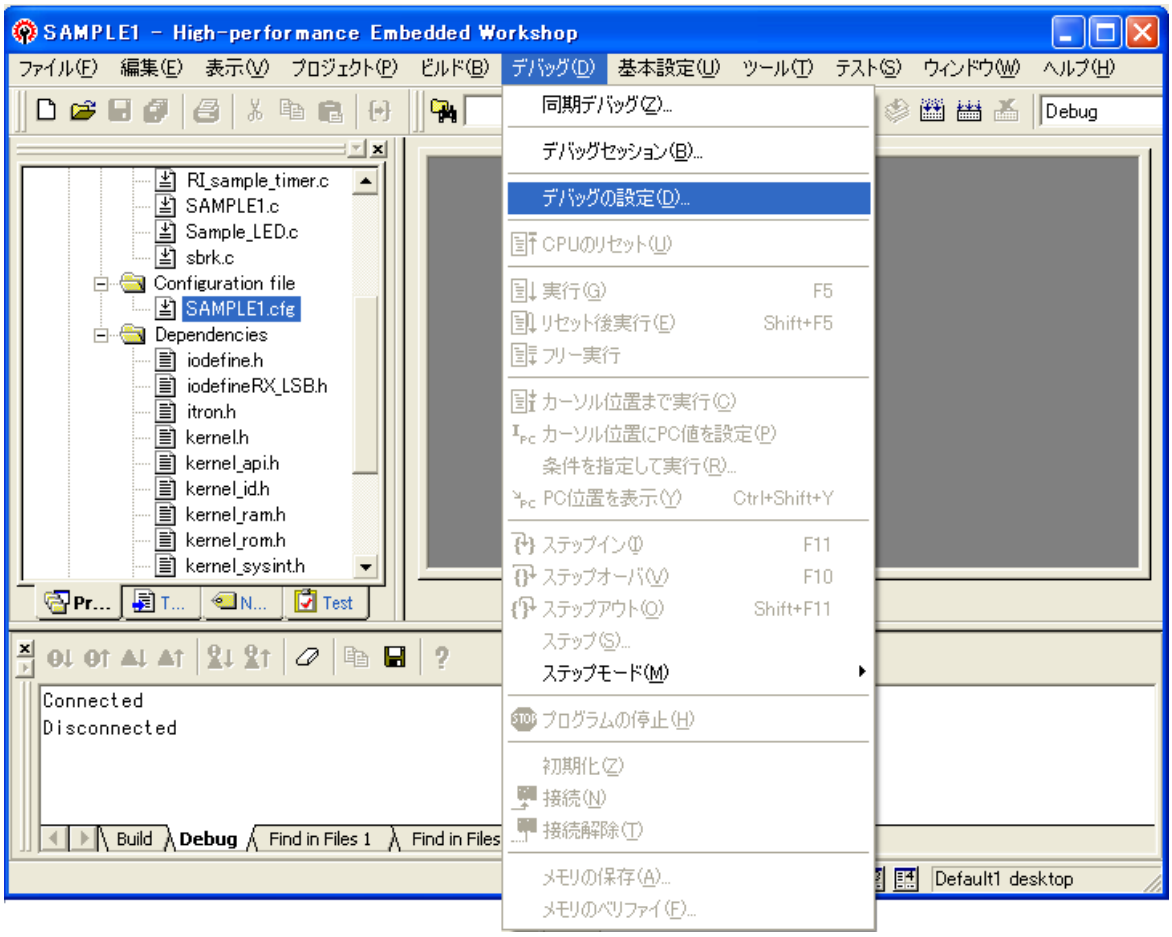


図2－16　デバッグの設定

[ダウンロードモジュール]には、ビルドで生成した abs ファイルを指定します。

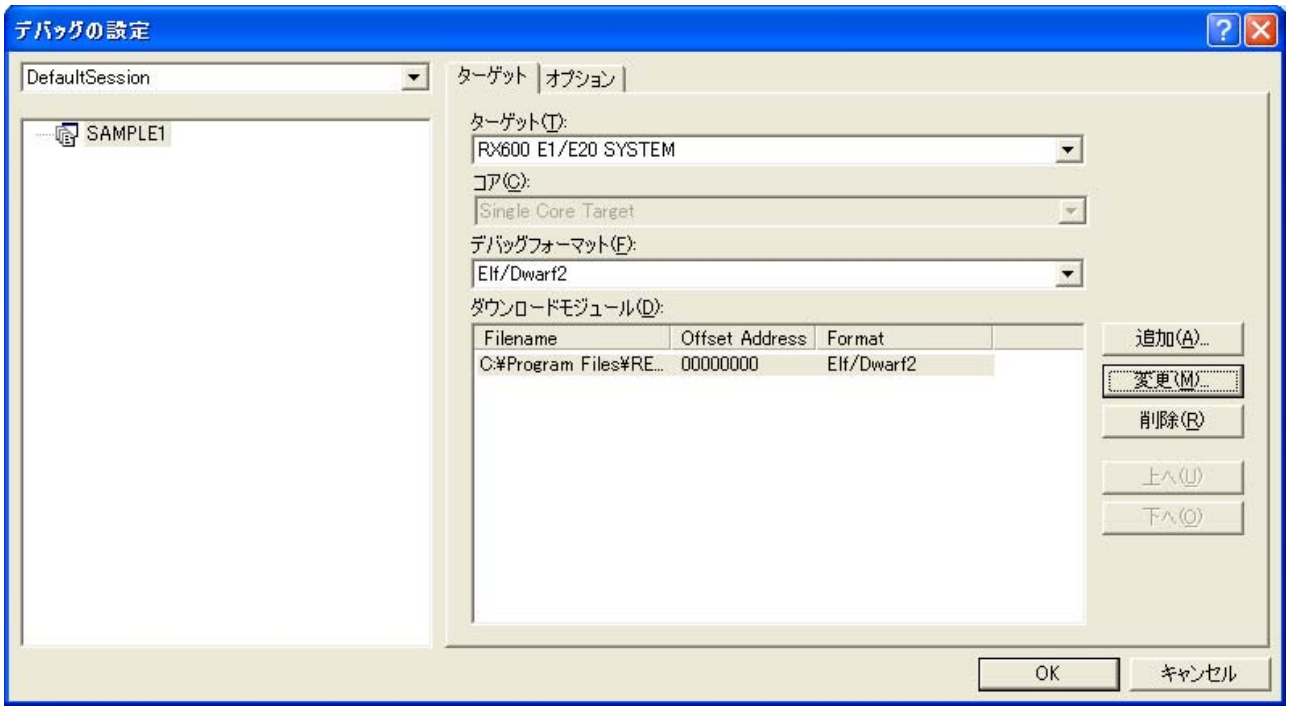


図2－17　デバッグの設定　ターゲット指定

[ターゲット]	“RX600 E1/ E20 SYSTEM”を選択します。
[デバッグフォーマット]	“Elf/Dwarf2”を選択します。
[ダウンロードモジュール]	ビルドで生成した abs ファイルを指定します。

デバッグの指定を行うと、ターゲットシステムへの接続を行います。

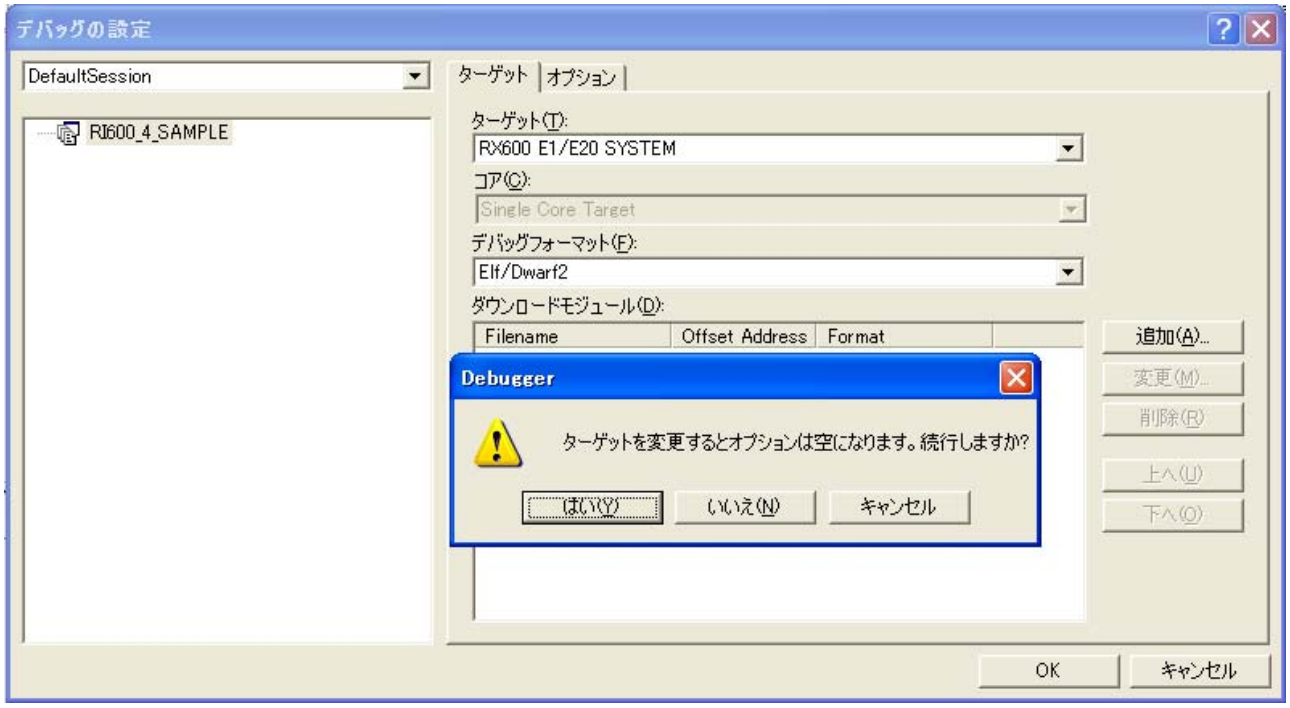


図2－18　デバッグの設定　ターゲット指定の確認

(2) ターゲットシステムの接続

ターゲットシステムを接続するため、デバイスと通信を設定します。本アプリケーションでは、RX610 スタータキットに合わせた設定を行います。

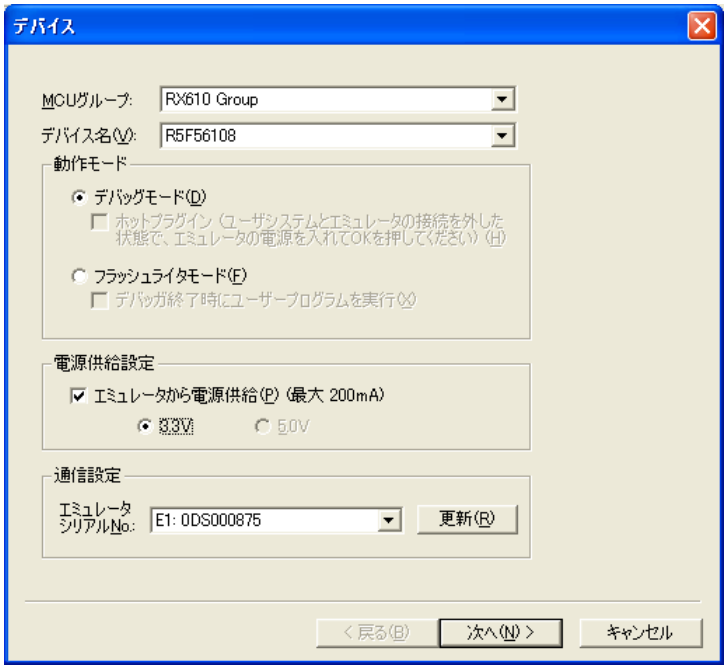


図2－19　ターゲットシステムの接続　デバイスの指定

[MCU グループ]	“RX610 Group”であることを確認します。
[デバイス名]	“R5F56108”を選択します。
[動作モード]	“デバッグモード”であることを確認します。
[電源供給設定]	“エミュレータから電源供給”の“3.3V”を指定します。
[通信設定]	[エミュレータシリアル No.]を確認します。

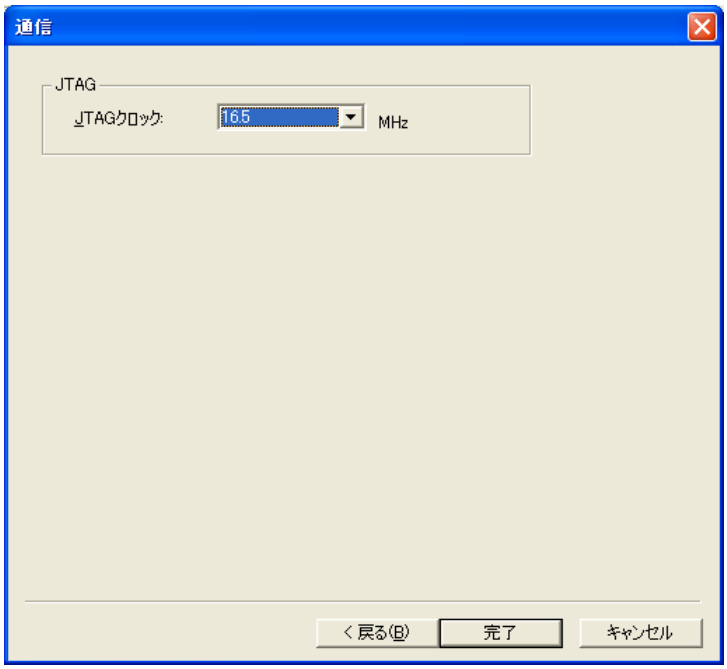


図2－20　ターゲットシステムの接続　通信の指定

[JTAG クロック]	“16.5”MHz であることを確認します。

[入力クロック]を設定します。

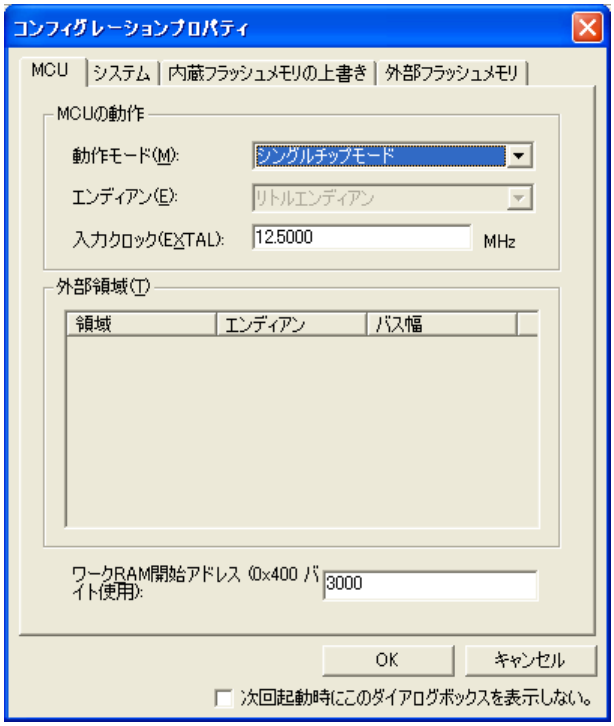


図2－21　ターゲットシステムの接続　コンフィギュレーションプロパティ

[MCU の動作]	“シングルチップモード”であることを確認します。
[入力クロック]	“12.5000”MHz を指定します。

これでターゲットシステムが接続され、デバッグの環境が整いました。

(3)　ダウンロード

[デバッグ]ー[ダウンロード]でロードモジュール（abs ファイル）をダウンロードします。

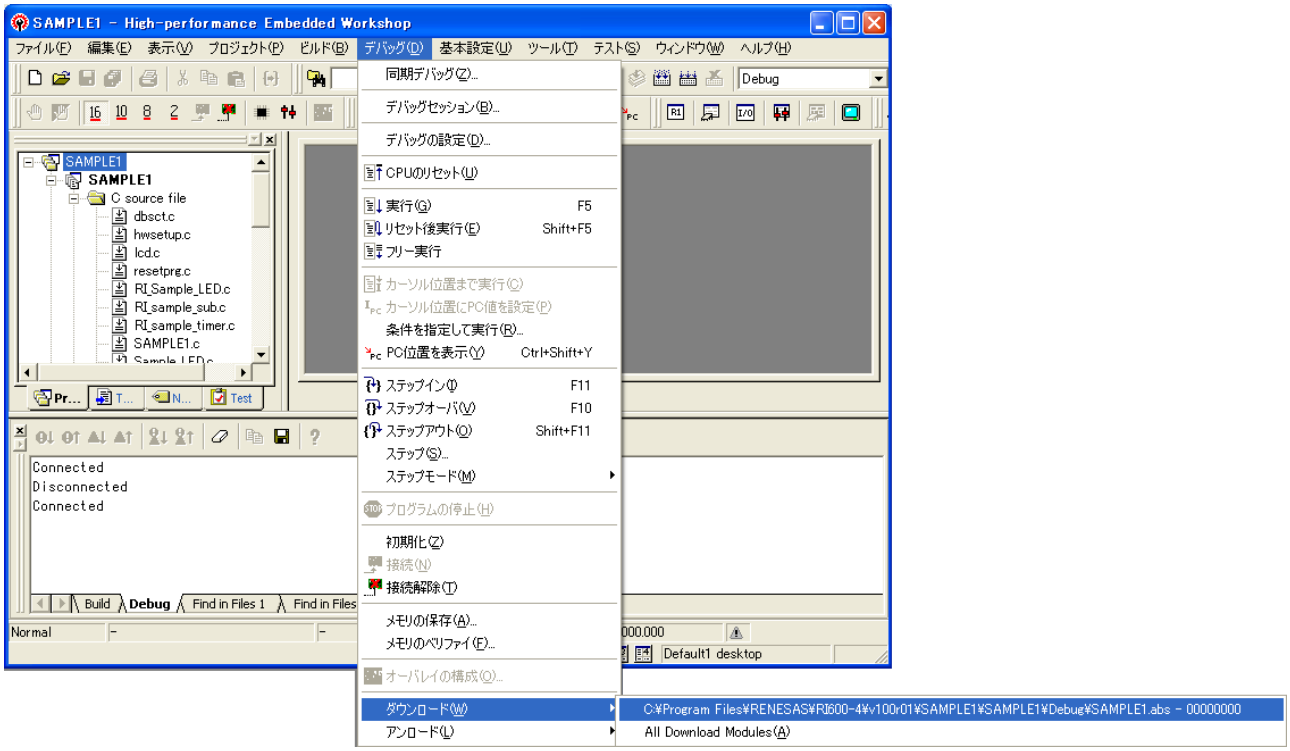


図2ー22　ロードモジュールのダウンロード

(4) プログラムの実行

[デバッグ]－[実行]でプログラムを実行します。

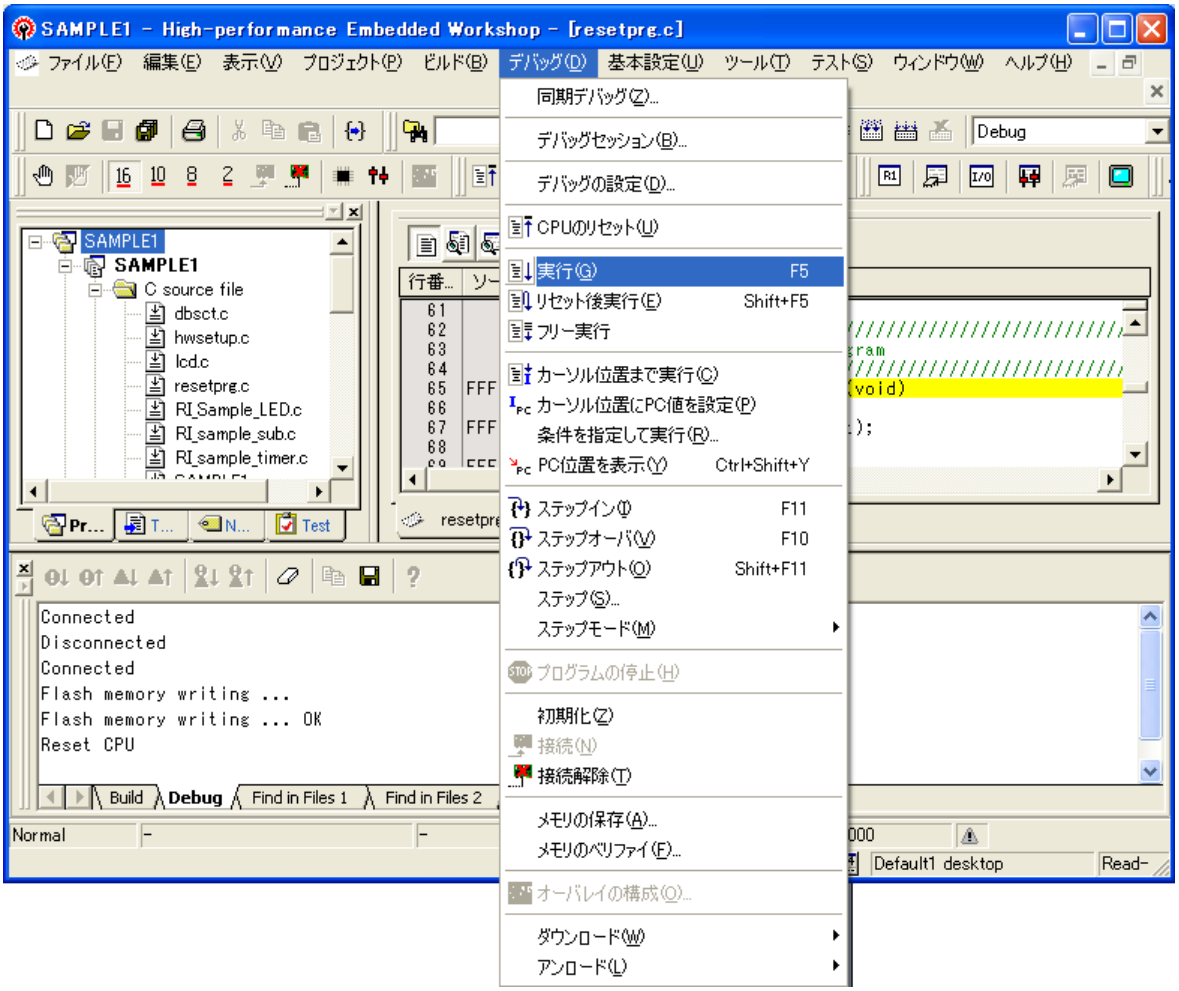


図2－23 プログラムの実行

3. サンプルプログラム

RI600/4 カーネルの機能を使用したサンプルプログラムを実行してみましょう。

SAMPLE1. タスクの起動と優先度

SAMPLE2. タスクの起動と同一優先度

SAMPLE3. タスクの待ち状態と待ち解除

SAMPLE4. イベントフラグと割込みハンドラ

SAMPLE5. タスクと周期ハンドラ

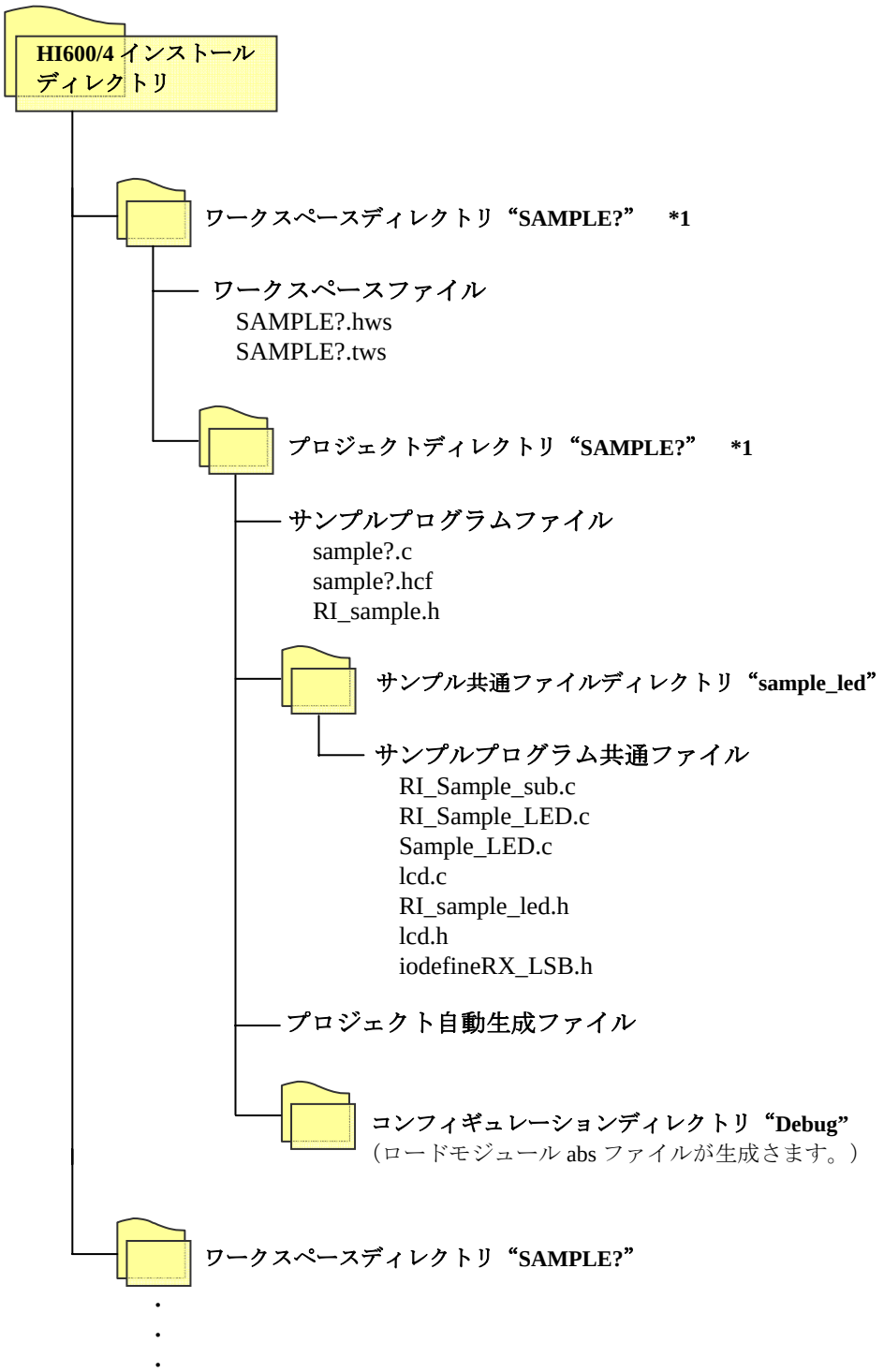
SAMPLE6. セマフォ

SAMPLE7. データキュー

SAMPLE8. メールボックスと固定長メモリプール

3.1　サンプルプログラムの構成

サンプルプログラムを、RI600/4 インストールディレクトリに格納します。
図3－1にサンプルプログラムのディレクトリ構成を示します。



【注】*1　SAMPLE?の“?“はサンプルプログラムの番号を意味します。

図3－1　サンプルプログラムディレクトリ構成

3.1.1 SAMPLE1：タスクの起動と優先度

SAMPLE1 では、1～5 の異なる優先度を与えられたタスクがどのような順番で実行状態になるかを確認します。

タスクには、処理の優先順位を意味する「タスク優先度」が付与され、実行可能状態（READY）のタスクの中で、最も高い優先度のタスクが実行状態（RUNNING）になります。

(1) アプリケーションプログラム

SAMPLE1 は、main_task、task2～task5 の 5 つのタスクからなります。

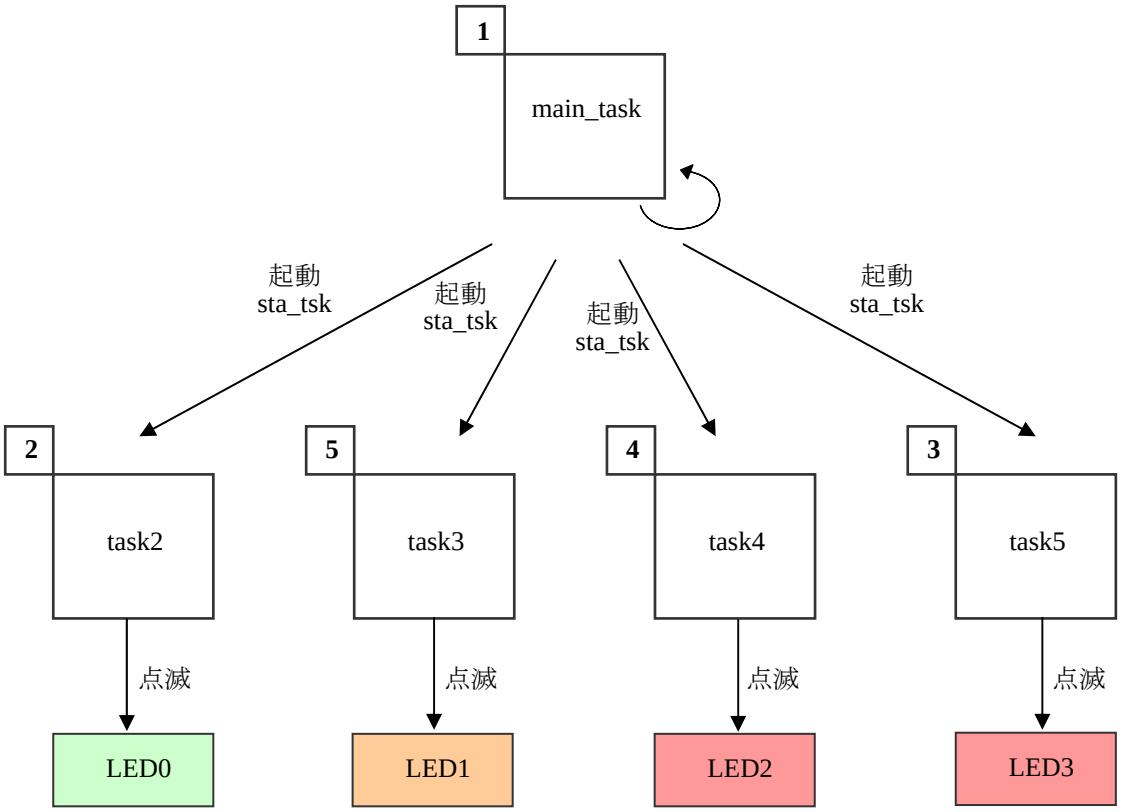
main_task から task2～task5 を起動し、優先度が高いタスク順番に実行されることを LED の点滅により確認することができます。

(a) 使用するオブジェクト

表3－1 使用するオブジェクト（SAMPLE1）

オブジェクト	アドレス	ID 番号	優先度	動作
タスク	main_task	1	1	初期起動 task2 から task5 の 4 つのタスクを起動
	task2	2	2	LED0 点滅
	task3	3	5	LED1 点滅
	task4	4	4	LED2 点滅
	task5	5	3	LED3 点滅

(b)　アプリケーションの動作



【注】sta_tsk はタスクを起動するサービスコールです。

図3－2　アプリケーションの動作（SAMPLE1）

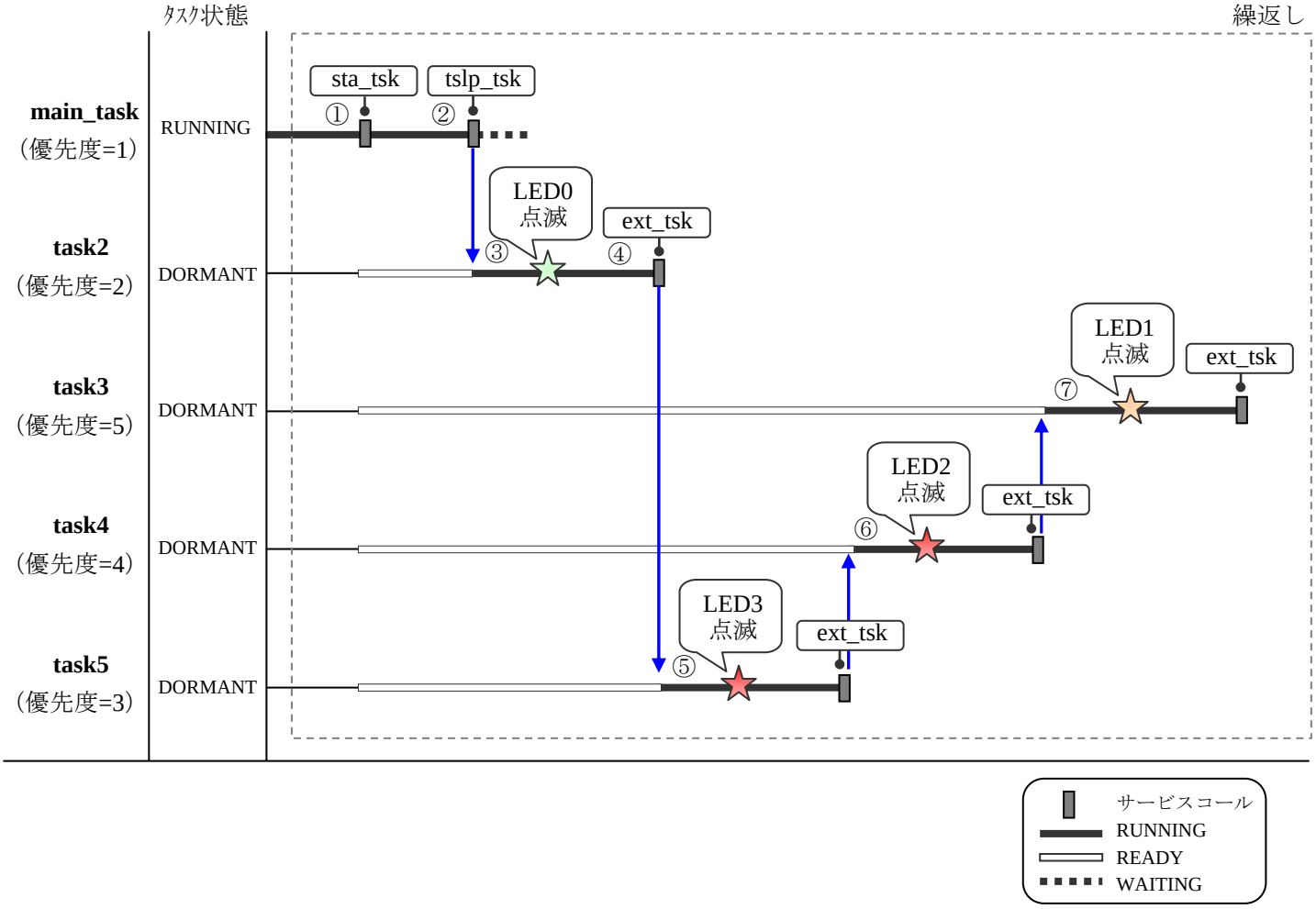


図3－3　アプリケーションのシーケンス（SAMPLE1）

① sta_tsk サービスコールで task2、task3、task4、task5 を起動します。

② main_task は tslp_tsk サービスコールで待ちに移行します。

③ READY 状態の中で、優先度の最も高い task2 が RUNNING 状態になり、LED0 を点滅させます。

④ ext_tsk サービスで task2 が終了します。

⑤ 次に優先度の高い task5 が RUNNING 状態になり、LED3 を点滅させます。

⑥ task5 が終了したら、次に優先度の高い task4 が RUNNING 状態になり、LED2 を点滅させます。

⑦ task4 が終了したら、次に task3 が RUNNING 状態になり、LED1 を点滅させます。

(c)　初期化処理の追加

プロジェクトの生成時に HEW が生成した初期化処理に、ハードウェアの初期化処理を追加します。

ハードウェアセットアップファイル hwsetup.c の関数にハードウェア初期化処理“RI_hwsetup()”を記述します。

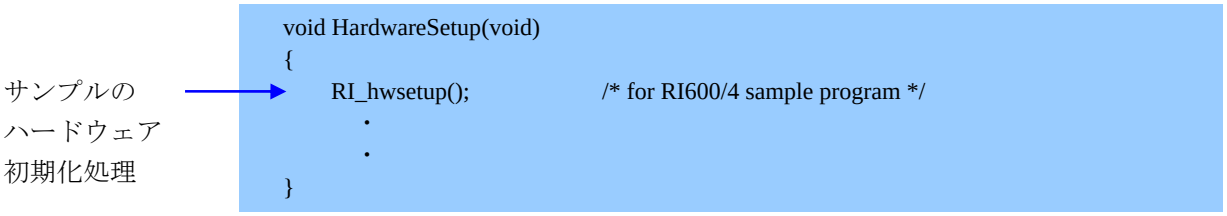


図3－4　初期化処理の追加（SAMPLE1）

(2)　コンフィギュレーションファイルの作成

HEW の[ツール]から GUI コンフィギュレータを起動して、コンフィギュレーションファイルを作成します。

ここでは、タスクの情報とサンプルのタイマ用割込みハンドラを定義します。

サンプルのタイマ用割込みハンドラの定義については『2.1.3　コンフィギュレーションファイル（cfgファイル）の作成』をご参照ください。

(a)　タスクの生成に必要な情報を入力します。

サンプルプログラムで定義するタスクの情報を 表3－2に示します。

表3－2　GUIコンフィギュレータ　タスクの定義情報（SAMPLE1）

No.	項目	タスク 1	タスク 2	タスク 3	タスク 4	タスク 5
1	[ID 番号]	1	2	3	4	5
2	[ID 名称]	ID_TASK1	ID_TASK2	ID_TASK3	ID_TASK4	ID_TASK5
3	[アドレス]	main_task	task2	task3	task4	task5
4	[タスク起動時の優先度]	1	2	5	4	3
5	[属性]	[生成後、起動]を指定します。	指定しません。（デフォルト）			
6	[スタックサイズ]	0x00000100（デフォルト）				
7	[スタック領域配置]	[セクション名]に“SURI_STACK”を指定します。（デフォルト）				
8	[記述言語]	[高級言語]（デフォルト）				
9	[拡張情報]	0x00000000（デフォルト）				

［タスク］ウィンドウからダイアログを開いて、タスクの情報を定義します。

図3－5　GUIコンフィギュレータ　タスクの定義（SAMPLE1）

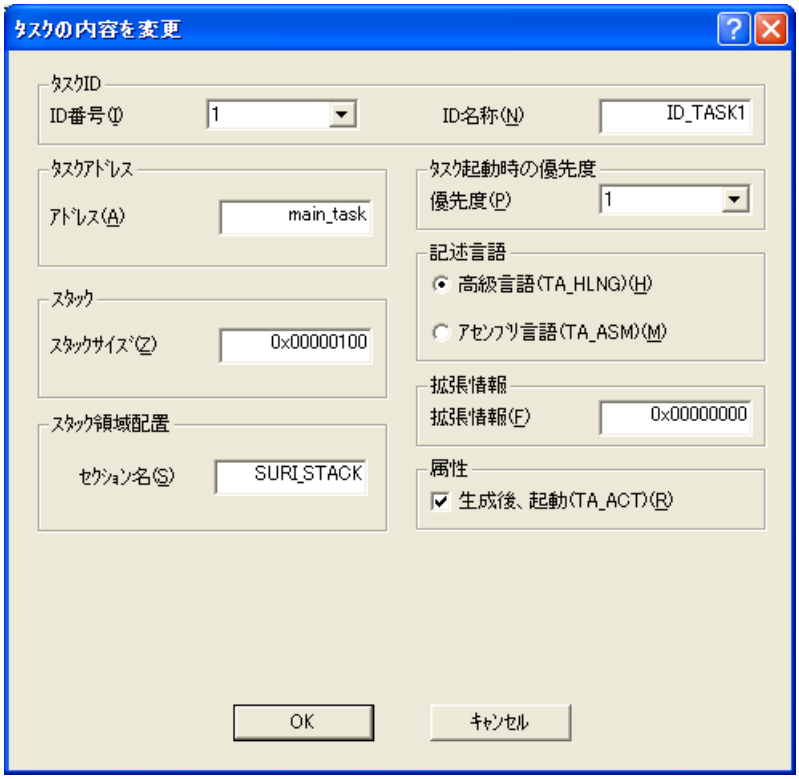


図3－6　GUIコンフィギュレータ　タスクの定義［ダイアログ］（SAMPLE1）

(b)　コンフィギュレーションファイルを生成します。

[生成]でコンフギュレーションファイル（cfgファイル）を生成します。SAMPLE1では、sample1.cfgを生成します。

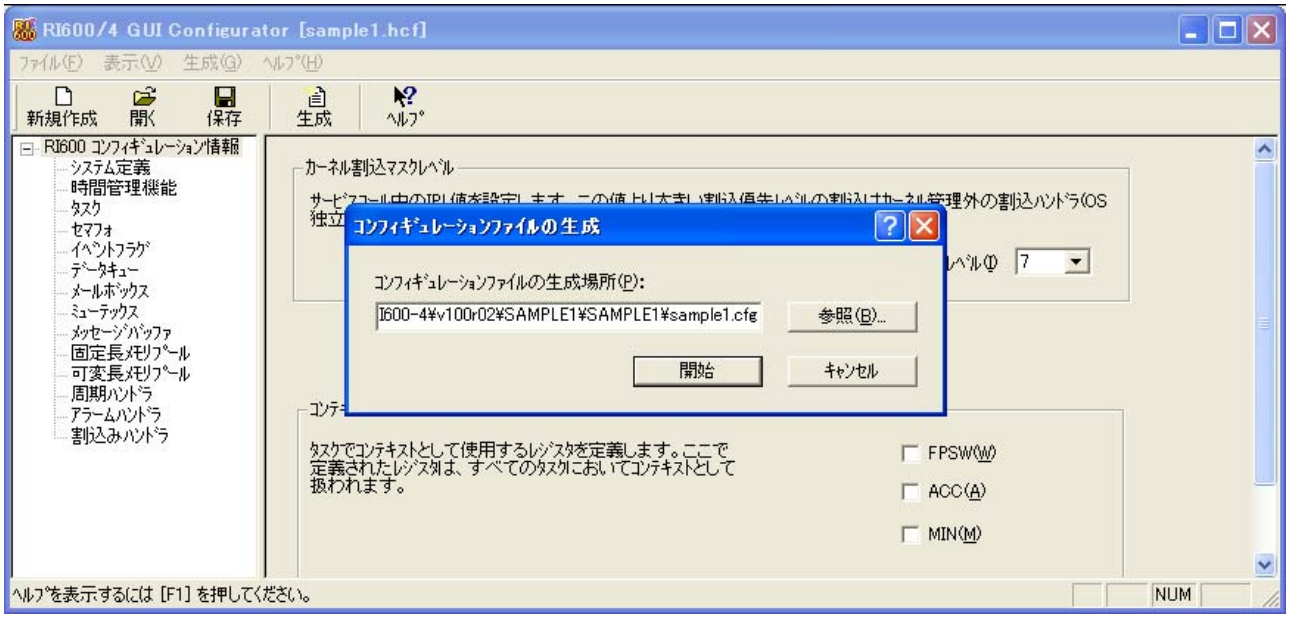


図3－7　コンフィギュレーションファイルの生成（SAMPLE1）

(3) ロードモジュールの作成

HEW のビルド機能を使って、ロードモジュールを作成します。

ロードモジュールの作成については『2.1.4　ロードモジュールの作成』をご参照ください。

(4) プログラムの実行

作成したプログラムを実際に実行します。

HEW のデバッグ機能を使って、デバッグの設定を行い、作成したロードモジュールをダウンロードします。

[実行]でプログラムが実行します。

プログラムの実行については『2.1.5　プログラムの実行』をご参照ください。

(5) プログラムの実行結果

task2 から task5 は、優先度の高い順に並べると、task2→task5→task4→task3 の順番になるので、LED が LED0→LED3→LED2→LED1 の順番で点滅を繰り返します。

以上で、SAMPLE1のシステムは完了です。

(6) サンプルプログラム

SAMPLE1のタスクのソースコードは 図3－8のように記述しています。

カーネル提供ファイル

cfg600 生成ファイル

1 main_task

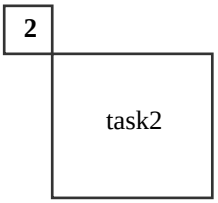
task2 起動（sta_tsk）

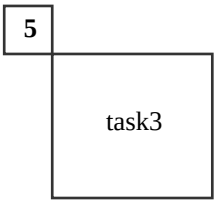
task3 起動（sta_tsk）

task4 起動（sta_tsk）

```
/************************************************************************
*
*     RI600/4 Sample program
*
************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

#include "RI_sample.h"                /* table for sample */

/************************************************************************
  main_task
************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd = 0;
      Number = 0L;

      while(1){

            sample_delay(1000);

            ercd = sta_tsk(ID_TASK2,stacd);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = sta_tsk(ID_TASK3,stacd);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = sta_tsk(ID_TASK4,stacd);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_TASK4, (VW)Number, 0L, 0L);
            }
            Number++;
```

図3－8　サンプルプログラムソースコード『SAMPLE1.c』（1/3）





task5 起動
（sta_tsk）

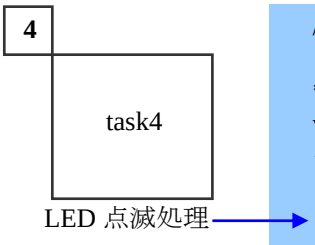
待ち状態へ
（tslp_tsk）

LED 点滅処理

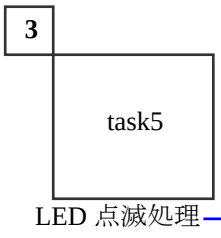
LED 点滅処理

```
            ercd = sta_tsk(ID_TASK5,stacd);
            if(ercd != E_OK)
            {
                vsys_dwn((ER)ID_TASK5, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = tslp_tsk(4000);
            if(ercd != E_TMOUT)
            {
                vsys_dwn((ER)ID_TASK1, (VW)Number, 0L, 0L);
            }
            Number++;
        }
        ext_tsk();
    }

    /*****************************************************************************
     task2
    *****************************************************************************/
    void task2(VP_INT exinf)
    {

        led_0();

        led_0();

        ext_tsk();
    }

    /*****************************************************************************
     task3
    *****************************************************************************/
    void task3(VP_INT exinf)
    {

        led_1();

        led_1();

        ext_tsk();
    }
```

図3－8　サンプルプログラムソースコード『SAMPLE1.c』（2/3）




```
/*****************************************************************************
 task4
*****************************************************************************/
void task4(VP_INT exinf)
{

        led_2();

        led_2();

        ext_tsk();
}
```




```
/*****************************************************************************
 task5
*****************************************************************************/
void task5(VP_INT exinf)
{

        led_3();

        led_3();

        ext_tsk();
}

/*****************************************************************************
*****************************************************************************/
```

図3－8　サンプルプログラムソースコード『Sample1.c』（3/3）

3.1.2 SAMPLE2：タスクの起動と同一優先度

SAMPLE2 では、同一優先度のタスクが複数存在する場合、同一優先度内でのタスクの優先順位がどのようになっているかを確認します。

同一の優先度のタスクが複数存在する場合には、その中で最初に実行可能状態になったタスクが実行状態になります。

(1) アプリケーションプログラム

SAMPLE2 は、main_task、task2～task5 の 5 つのタスクからなります。

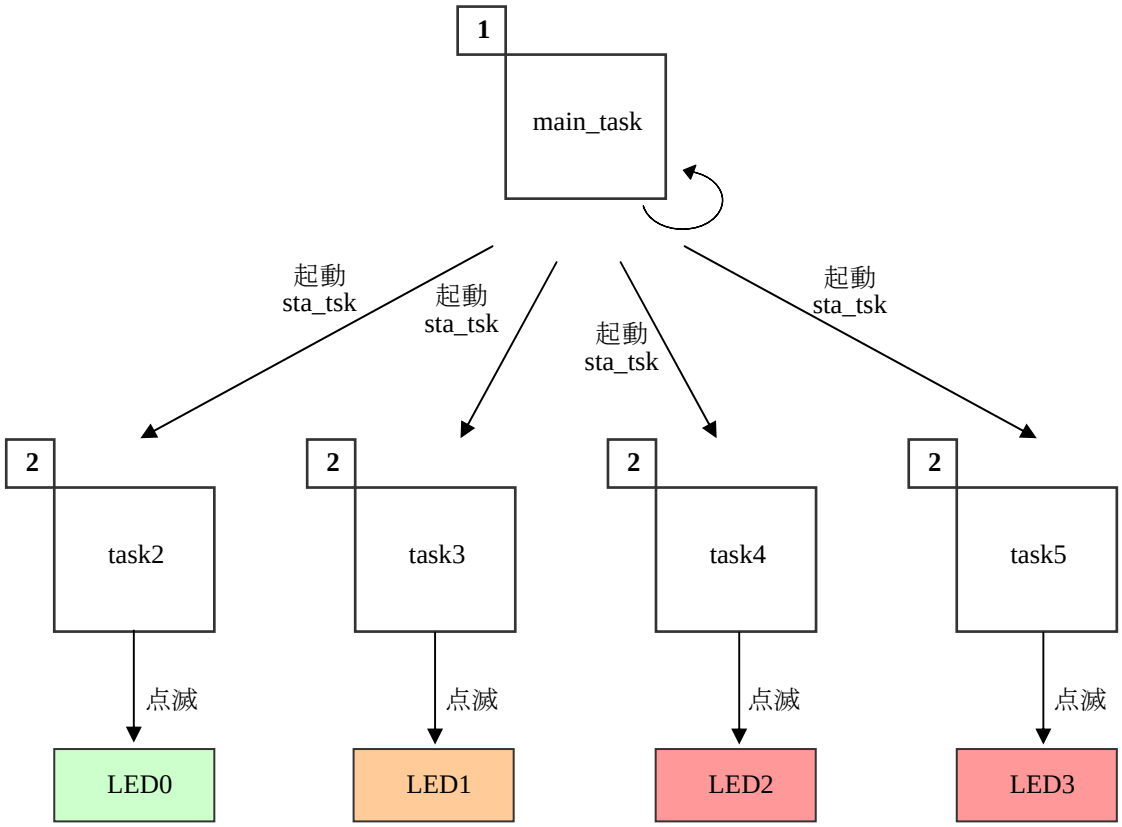
main_task から task2～task5 を起動し、実行可能状態になったタスクの順番に実行されることを LED の点滅により確認することができます。

(a) 使用するオブジェクト

表3－3 使用するオブジェクト（SAMPLE2）

オブジェクト	アドレス	ID 番号	優先度	動作
タスク	main_task	1	1	初期起動 task2 から task5 の 4 つのタスクを起動
	task2	2	2	LED0 点滅
	task3	3	2	LED1 点滅
	task4	4	2	LED2 点滅
	task5	5	2	LED3 点滅

(b)　アプリケーションの動作



【注】sta_tsk はタスクを起動するサービスコールです。

図3−9　アプリケーションの動作（SAMPLE2）

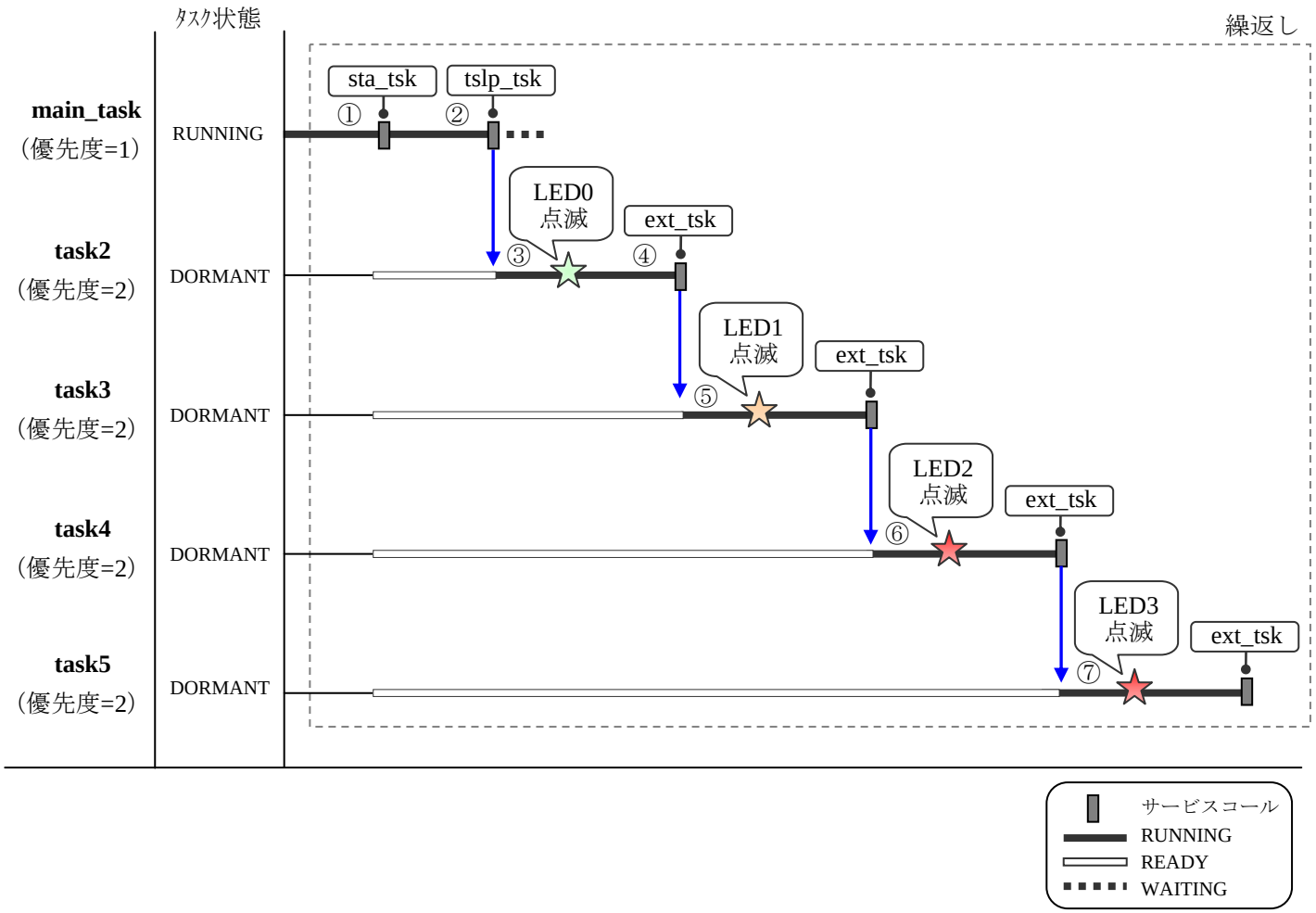


図3－10　アプリケーションのシーケンス（SAMPLE2）

① sta_tsk サービスコールで task2、task3、task4、task5 の順番でタスクを起動します。
② main_task は tslp_tsk サービスコールで待ちに移行します。
③ main_task から最初に起動された task2 が RUNNING 状態になり、LED0 を点滅させます。
④ ext_tsk サービスで task2 が終了します。
⑤ 次に起動された task3 が RUNNING 状態になり、LED1 を点滅させます。
⑥ task3 が終了したら、次に起動された task4 が RUNNING 状態になり、LED2 を点滅させます。
⑦ task4 が終了したら、次に起動された task5 が RUNNING 状態になり、LED3 を点滅させます。

(c)　初期化処理の追加

プロジェクトの生成時に HEW が生成した初期化処理に、ハードウェアの初期化処理を追加します。

ハードウェアセットアップファイル hwsetup.c の関数にハードウェア初期化処理“RI_hwsetup()”を記述します。

サンプルの
ハードウェア
初期化処理

```
void HardwareSetup(void)
{
    RI_hwsetup();              /* for RI600/4 sample program */
      ・
      ・
}
```

図3－11　初期化処理の追加（SAMPLE2）

(2)　コンフィギュレーションファイルの作成

HEW の[ツール]から GUI コンフィギュレータを起動して、コンフィギュレーションファイルを作成します。

ここでは、タスクの情報とサンプルのタイマ用割込みハンドラを定義します。

サンプルのタイマ用割込みハンドラの定義については『2.1.3　コンフィギュレーションファイル（cfgファイル）の作成』をご参照ください。

(a)　タスクの生成に必要な情報を入力します。

サンプルプログラムで定義するタスクの情報を 表3－4に示します。

表3－4　GUIコンフィギュレータ　タスクの定義情報（SAMPLE2）

No.	項目	タスク 1	タスク 2	タスク 3	タスク 4	タスク 5
1	[ID 番号]	1	2	3	4	5
2	[ID 名称]	ID_TASK1	ID_TASK2	ID_TASK3	ID_TASK4	ID_TASK5
3	[アドレス]	main_task	task2	task3	task4	task5
4	[タスク起動時の優先度]	1	2	2	2	2
5	[属性]	[生成後、起動]を指定します。	指定しません。（デフォルト）			
6	[スタックサイズ]	0x00000100（デフォルト）				
7	[スタック領域配置]	[セクション名]に“SURI_STACK”を指定します。（デフォルト）				
8	[記述言語]	[高級言語]（デフォルト）				
9	[拡張情報]	0x00000000（デフォルト）				

［タスク］ウィンドウからダイアログを開いて、タスクの情報を定義します。

図3－12　GUIコンフィギュレータ　タスクの定義（SAMPLE2）

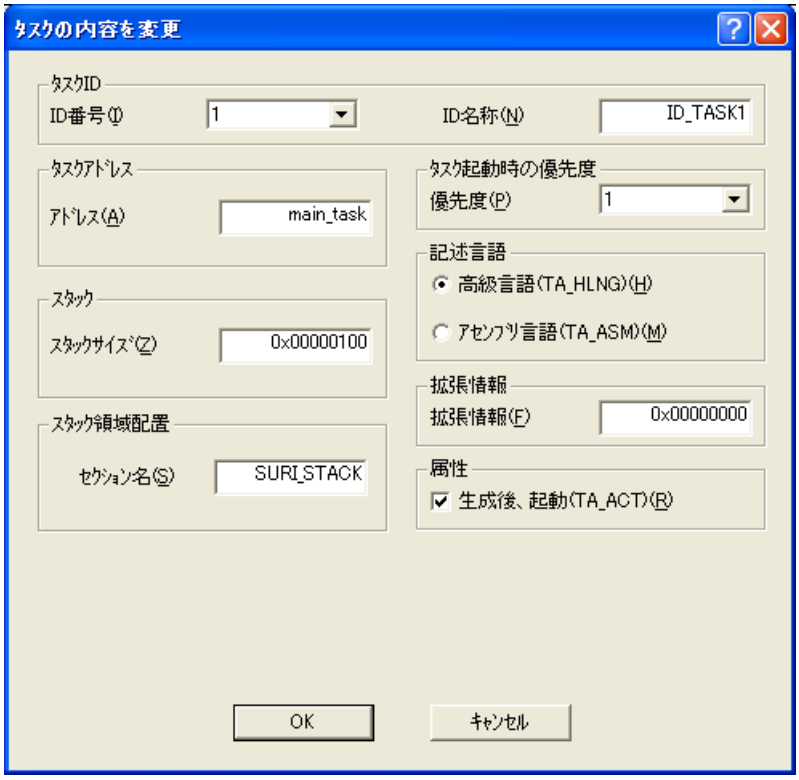


図3－13　GUIコンフィギュレータ　タスクの定義［ダイアログ］（SAMPLE2）

(b)　コンフィギュレーションファイルを生成します。

[生成]でコンフギュレーションファイル（cfgファイル）を生成します。SAMPLE2では、sample2.cfgを生成します。

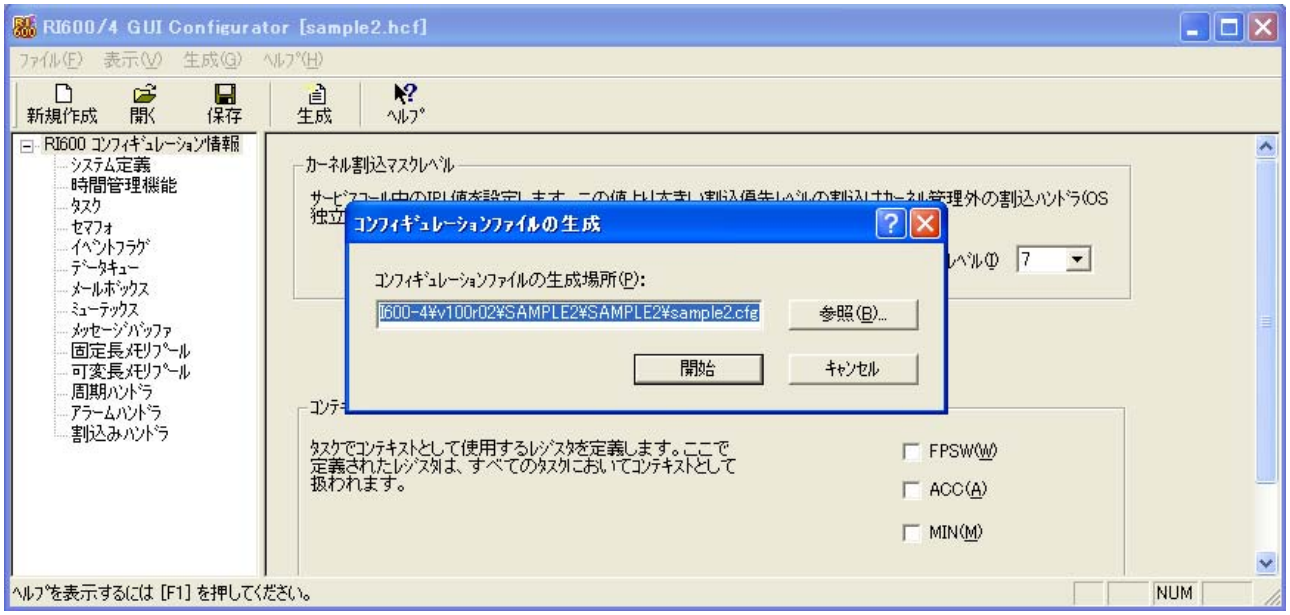


図3－14　コンフィギュレーションファイルの生成（SAMPLE2）

(3) ロードモジュールの作成

HEW のビルド機能を使って、ロードモジュールを作成します。

ロードモジュールの作成については『2.1.4　ロードモジュールの作成』をご参照ください。

(4) プログラムの実行

作成したプログラムを実際に実行します。

HEW のデバッグ機能を使って、デバッグの設定を行い、作成したロードモジュールをダウンロードします。

[実行]でプログラムが実行します。

プログラムの実行については『2.1.5　プログラムの実行』をご参照ください。

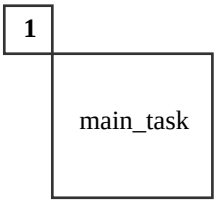
(5) プログラムの実行結果

同一優先度の task2 から task5 は、main_task から task2→task3→task4→task5 の順番で起動され、実行可能状態になったので、LED が LED0→LED1→LED2→LED3 の順番で点滅を繰り返します。

以上で、SAMPLE2のシステムは完了です。

(6) サンプルプログラム

SAMPLE2のタスクのソースコードは 図3－15のように記述しています。



カーネル提供ファイル

cfg600 生成ファイル

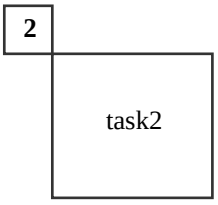
task2 起動（sta_tsk）

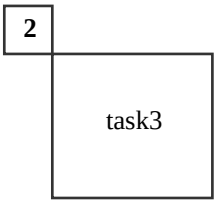
task3 起動（sta_tsk）

task4 起動（sta_tsk）

```
/*************************************************************************
*
*     RI600/4 Sample program
*
*************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

#include "RI_sample.h"                /* table for sample */

/*************************************************************************
  main_task
*************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd= 0;
      Number = 0L;

      while(1){

            sample_delay(1000);

            ercd = sta_tsk(ID_TASK2,stacd);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = sta_tsk(ID_TASK3,stacd);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = sta_tsk(ID_TASK4,stacd);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_TASK4, (VW)Number, 0L, 0L);
            }
            Number++;
```

図3－15　サンプルプログラムソースコード『SAMPLE2.c』（1/3）





task5 起動
（sta_tsk）

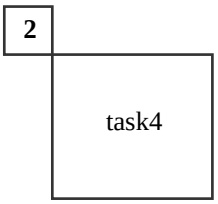
待ち状態へ
（tslp_tsk）

LED 点滅処理

LED 点滅処理

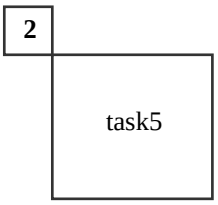
```
            ercd = sta_tsk(ID_TASK5,stacd);
            if(ercd != E_OK)
            {
                vsys_dwn((ER)ID_TASK5, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = tslp_tsk(4000);
            if(ercd != E_TMOUT)
            {
                vsys_dwn((ER)ID_TASK1, (VW)Number, 0L, 0L);
            }
            Number++;
        }
        ext_tsk();
    }

    /*****************************************************************************
     task2
    *****************************************************************************/
    void task2(VP_INT exinf)
    {

        led_0();

        led_0();

        ext_tsk();
    }

    /*****************************************************************************
     task3
    *****************************************************************************/
    void task3(VP_INT exinf)
    {

        led_1();

        led_1();

        ext_tsk();
    }
```

図3－15　サンプルプログラムソースコード『SAMPLE2.c』（2/3）



LED 点滅処理

```
/*****************************************************************************
 task4
*****************************************************************************/
void task4(VP_INT exinf)
{

    led_2();

    led_2();

    ext_tsk();
}
```



LED 点滅処理

```
/*****************************************************************************
 task5
*****************************************************************************/
void task5(VP_INT exinf)
{

    led_3();

    led_3();

    ext_tsk();
}

/*****************************************************************************
*****************************************************************************/
```

図3－15　サンプルプログラムソースコード『SAMPLE2.c』（3/3）

3.1.3 SAMPLE3：タスクの待ち状態と待ち解除

SAMPLE3では、カーネルのタスク付属同期機能を使って、タスクの起床待ちと起床待ち解除を行います。

タスクは、タスク間の同期をとるために待ち状態にしたり、待ち状態になったタスクを起床させたりすることができます。

(1) アプリケーションプログラム

SAMPLE3は、main_task、task2、task3 の 3 つのタスクからなります。

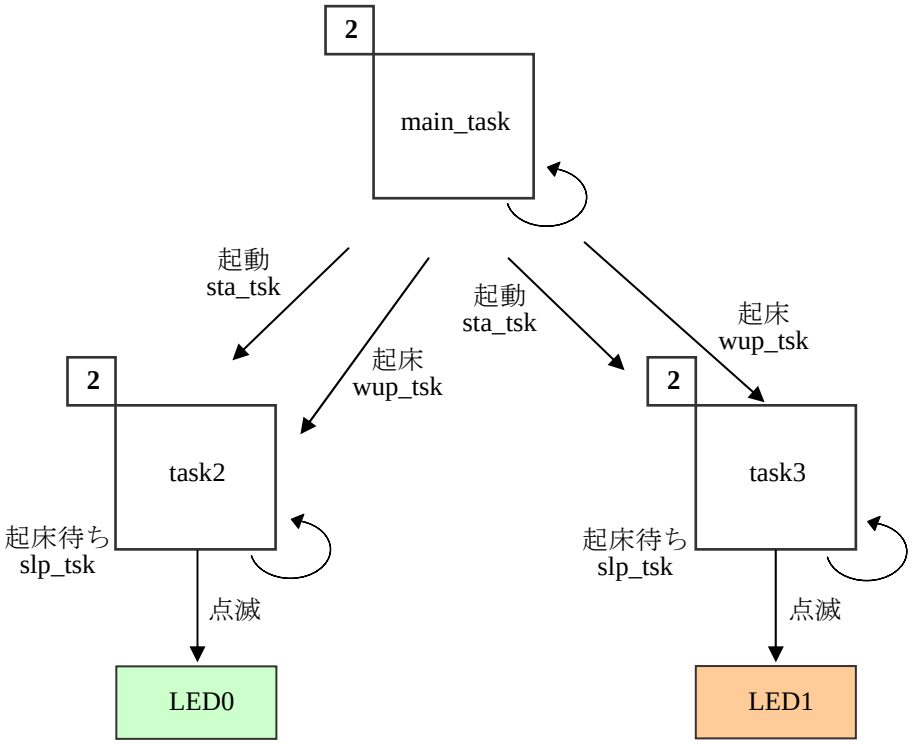
main_task は、task2 と task3 の起床待ちを解除します。起床されたタスクは LED の点滅処理を行います。

(a) 使用するオブジェクト

表3−5 使用するオブジェクト（SAMPLE3）

オブジェクト	アドレス	ID 番号	優先度	動作
タスク	main_task	1	2	初期起動 task2 と task3 のタスクを起動 task2 と task3 のタスクを起床
	task2	2	2	起床待ち、LED0 点滅
	task3	3	2	起床待ち、LED1 点滅

(b)　アプリケーションの動作



【注】sta_tsk はタスクを起動するサービスコールです。
slp_tsk はタスクを起床待ちにするサービスコールです。
wup_tsk はタスクを起床するサービスコールです。

図3－16　アプリケーションの動作（SAMPLE3）

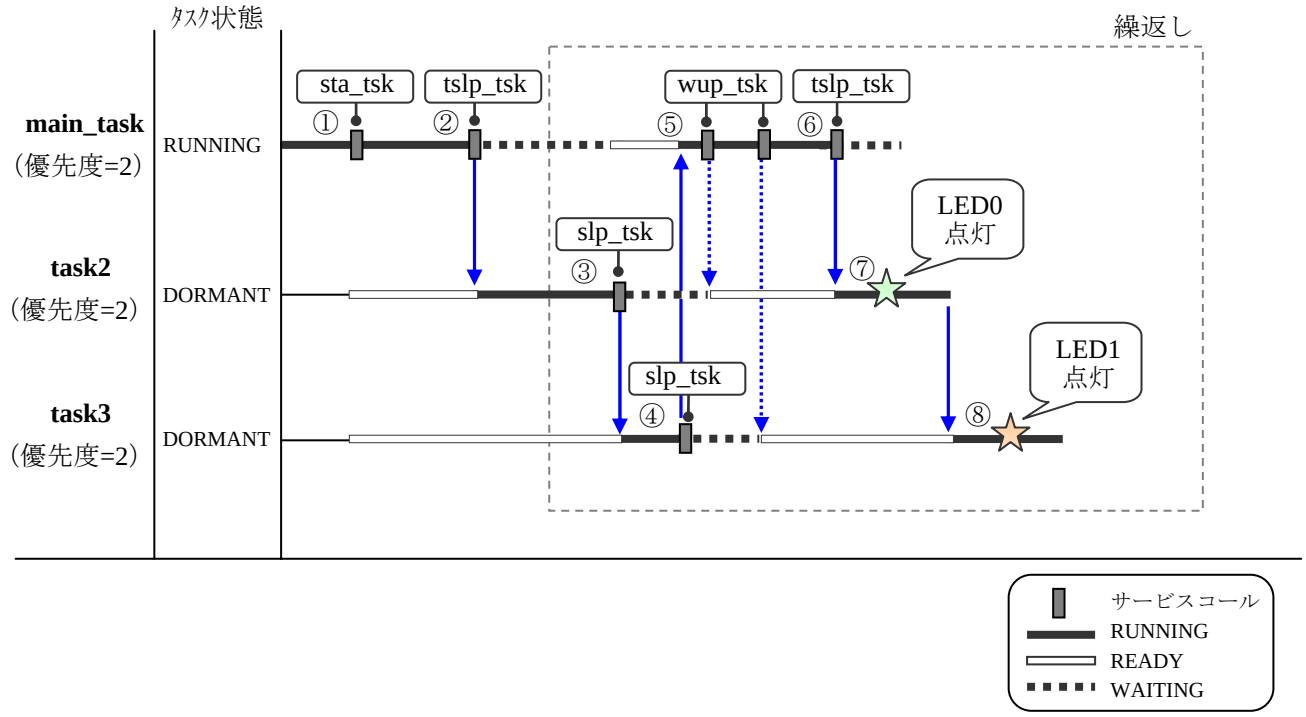


図3－17　アプリケーションのシーケンス（SAMPLE3）

① sta_tsk サービスコールで task2 と task3 を起動します。

② main_task は、tslp_tsk サービスコールで待ちに移行します。

③ READY 状態で先に起動された task2 が RUNNING 状態になり、slp_tsk サービスコールで待ちに移行します。

④ 次に起動された task3 が RUNNING 状態になり、slp_tsk サービスコールで待ちに移行します。

⑤ main_task が tslp_tsk サービスコールでタイムアウトになり RUNNING 状態に移行し、task2 と task3 を wup_tsk サービスコールで起床させます。

⑥ 他のタスクを動作させるため、tslp_tsk サービスコールで待ちに移行します。

⑦ task2 が RUNNING 状態になり、LED0 を点滅させます。

⑧ 次に task3 が RUNNING 状態になり、LED1 を点滅させます。

(c)　初期化処理の追加

プロジェクトの生成時に HEW が生成した初期化処理にハードウェアの初期化処理を追加します。

ハードウェアセットアップファイル hwsetup.c の関数にハードウェア初期化処理“RI_hwsetup()”を記述します。

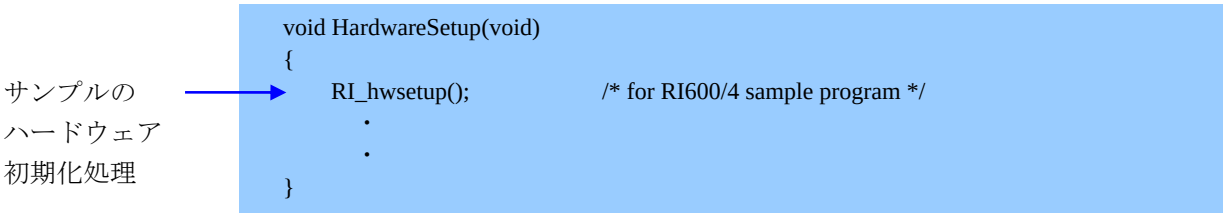


図3－18　初期化処理の追加（SAMPLE3）

(2)　コンフィギュレーションファイルの作成

HEW の[ツール]から GUI コンフィギュレータを起動して、コンフィギュレーションファイルを作成します。

ここでは、タスクの情報とサンプルのタイマ用割込みハンドラを定義します。

サンプルのタイマ用割込みハンドラの定義については『2.1.3　コンフィギュレーションファイル（cfgファイル）の作成』をご参照ください。

(a)　タスクの生成に必要な情報を入力します。

サンプルプログラムで定義するタスクの情報を 表3－6に示します。

表3－6　GUIコンフィギュレータ　タスクの定義情報（SAMPLE3）

No.	項目	タスク 1	タスク 2	タスク 3
1	[ID 番号]	1	2	3
2	[ID 名称]	ID_TASK1	ID_TASK2	ID_TASK3
3	[アドレス]	main_task	task2	task3
4	[タスク起動時の優先度]	2	2	2
5	[属性]	[生成後、起動]を指定します。	指定しません。（デフォルト）	
6	[スタックサイズ]	0x00000100（デフォルト）		
7	[スタック領域配置]	[セクション名]に“SURI_STACK”を指定します。（デフォルト）		
8	[記述言語]	[高級言語]（デフォルト）		
9	[拡張情報]	0x00000000（デフォルト）		

［タスク］ウィンドウからダイアログを開いて、タスクの情報を定義します。

図3－19　GUIコンフィギュレータ　タスクの定義（SAMPLE3）

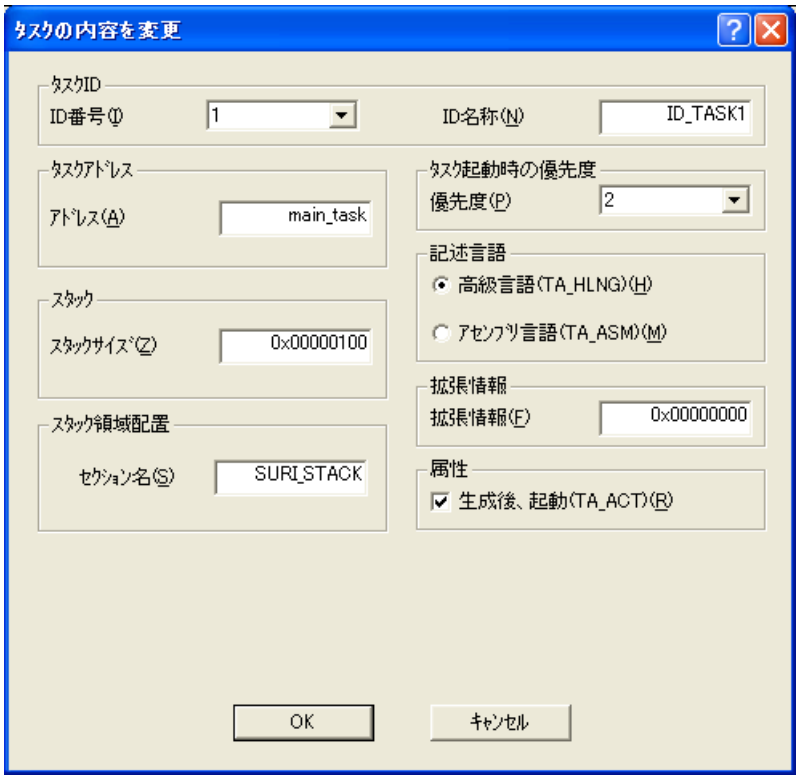


図3－20　GUIコンフィギュレータ　タスクの定義［ダイアログ］（SAMPLE3）

(b)　コンフィギュレーションファイルを生成します。

[生成]でコンフギュレーションファイル（cfgファイル）を生成します。SAMPLE3では、sample3.cfgを生成します。

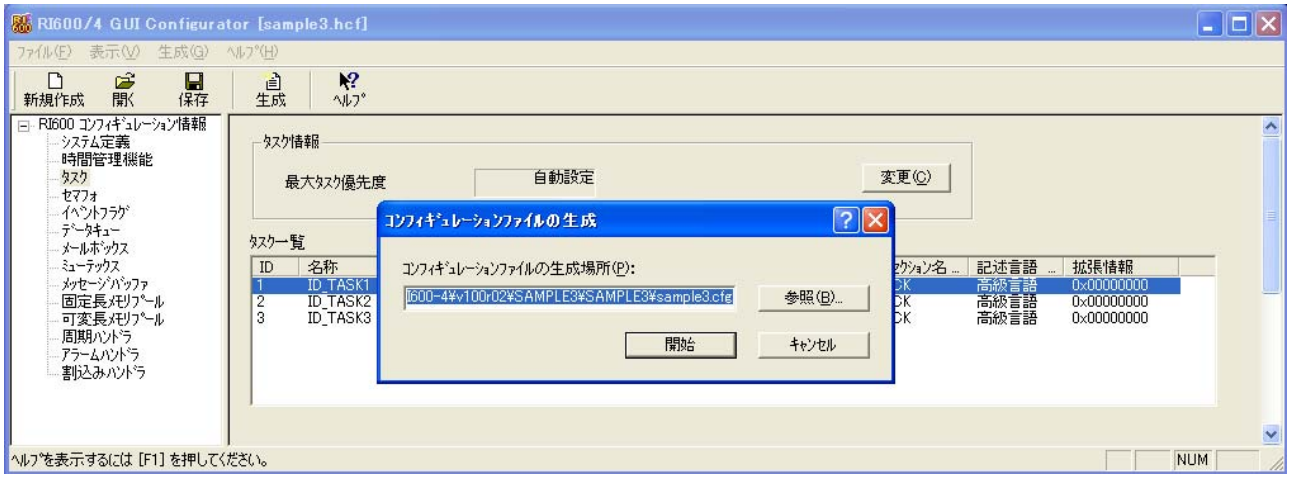


図3－21　コンフィギュレーションファイルの生成（SAMPLE3）

(3) ロードモジュールの作成

HEW のビルド機能を使って、ロードモジュールを作成します。
ロードモジュールの作成については『2.1.4　ロードモジュールの作成』をご参照ください。

(4) プログラムの実行

作成したプログラムを実際に実行します。
HEW のデバッグ機能を使って、デバッグの設定を行い、作成したロードモジュールをダウンロードします。
[実行]でプログラムが実行します。
プログラムの実行については『2.1.5　プログラムの実行』をご参照ください。

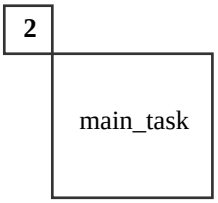
(5) プログラムの実行結果

起床待ち状態の task2 と task3 が、main_task から起床され、LED0 と LED1 の点滅を繰り返します。

以上で、SAMPLE3のシステムは完了です。

(6) サンプルプログラム

SAMPLE3のタスクのソースコードは 図3－22のように記述しています。



カーネル提供ファイル

cfg600 生成ファイル

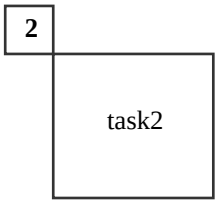
task2 起動（sta_tsk）

task3 起動（sta_tsk）

待ち状態へ（tslp_tsk）

```
/************************************************************************
*
*     RI600/4 Sample program
*
************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

#include "RI_sample.h"              /* table for sample */

/************************************************************************
  main_task
************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd= 0;
      Number = 0L;

      ercd = sta_tsk(ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk(ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = tslp_tsk(10);
      if(ercd != E_TMOUT)
      {
            vsys_dwn((ER)ID_TASK1, (VW)Number, 0L, 0L);
      }
      Number++;
```

図3－22　サンプルプログラムソースコード『SAMPLE3.c』（1/3）



task2 起床
（wup_tsk）

task3 起床
（wup_tsk）

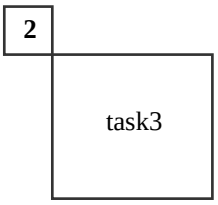
待ち状態へ
（tslp_tsk）

待ち状態へ
（slp_tsk）

LED 点灯処理

```
        while(1){

            led_off_0();
            led_off_1();

            ercd = wup_tsk(ID_TASK2);
            if(ercd != E_OK)
            {
                vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = wup_tsk(ID_TASK3);
            if(ercd != E_OK)
            {
                vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = tslp_tsk(10);
            if(ercd != E_TMOUT)
            {
                vsys_dwn((ER)ID_TASK1, (VW)Number, 0L, 0L);
            }
            Number++;

        }
        ext_tsk();
    }

    /*****************************************************************************
      task2
    *****************************************************************************/
    void task2(VP_INT exinf)
    {
        ER    ercd, Number;

        Number = 0L;

        while(1){

            ercd = slp_tsk();
            if(ercd != E_OK)
            {
                vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
            }

            led_0();
        }
        Number++;

        ext_tsk();
    }
```

図3－22　サンプルプログラムソースコード『SAMPLE3.c』（2/3）




```
/***********************************************************************************
  task3
***********************************************************************************/
void task3(VP_INT exinf)
{
        ER    ercd, Number;

        Number = 0L;

        while(1){

                ercd = slp_tsk();
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
                }

                led_1();
        }
        Number++;

        ext_tsk();
}
/***********************************************************************************
***********************************************************************************/
```

待ち状態へ（slp_tsk）

LED 点灯処理

図3－22　サンプルプログラムソースコード『SAMPLE3.c』（3/3）

3.1.4 SAMPLE4：イベントフラグと割込みハンドラ

SAMPLE4では、カーネルのイベントフラグを使って、タスクと割込み処理の同期・通信を行います。

CPU ボードのスイッチで起動する割込みハンドラから、イベントフラグをセットして、タスクのイベントフラグ待ちを解除します。

イベントフラグは、イベントの有無をビット毎のフラグで表現することにより同期をとるオブジェクトです。

(1) アプリケーションプログラム

SAMPLE4は、main_task、task2、task3 の 3 つのタスクと、inthdr1～inthdr3 の 3 つの割込みハンドラからなります。

main_task から起動された task2 と task3 は、イベントフラグ待ちに移行します。

inthdr2 と inthdr3 から、task2 と task3 のイベントフラグ待ちを解除します。待ちが解除されたタスクは、LED の点灯と消灯を交互に行います。

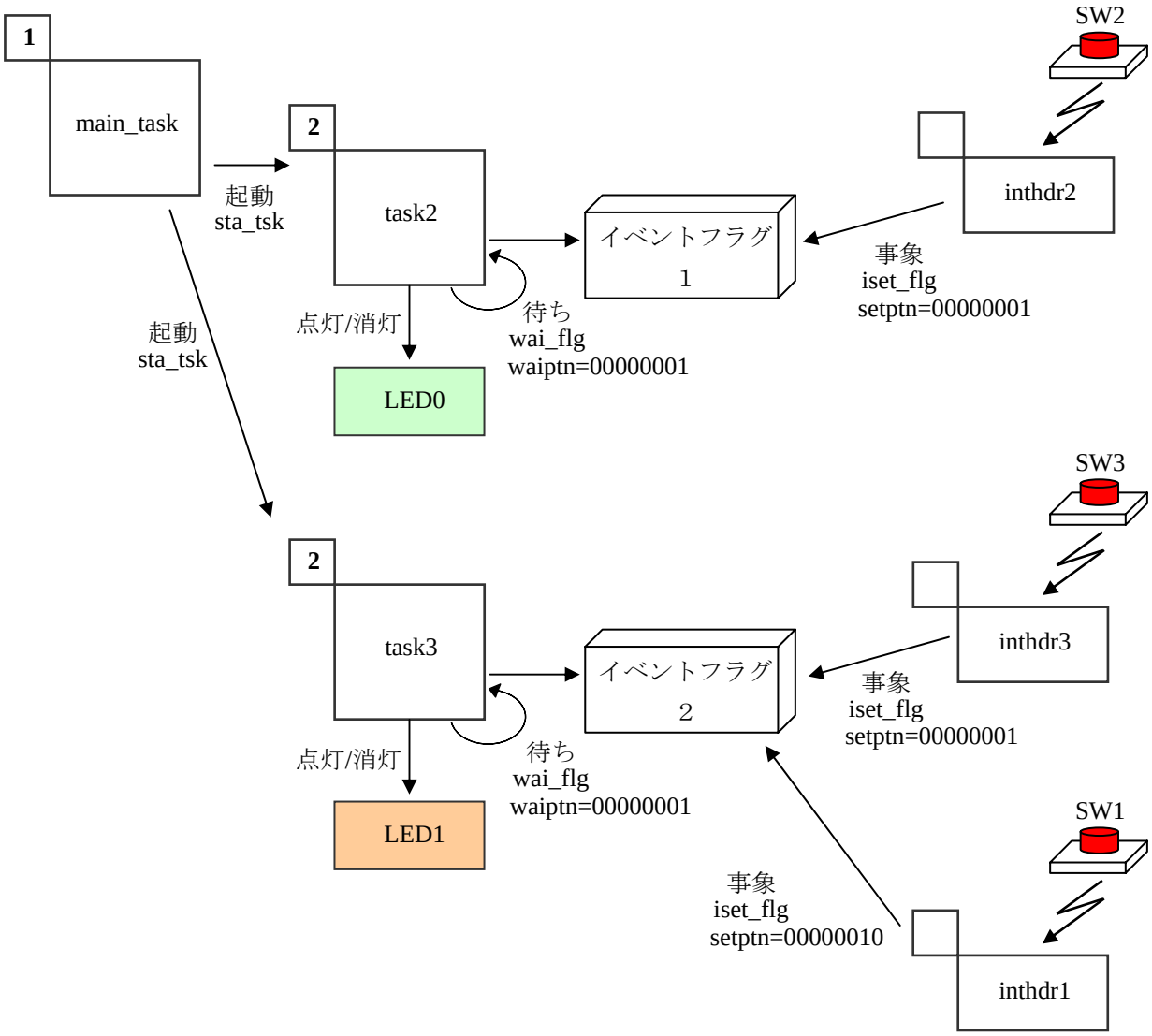
inthdr1 からはイベントフラグ待ちは解除されません。

(a) 使用するオブジェクト

表3－7 使用するオブジェクト（SAMPLE4）

オブジェクト	アドレス	ID 番号	優先度	動作
タスク	main_task	1	1	初期起動 task2 の起動、task3 の起動
	task2	2	2	イベントフラグ 1 の waiptn=00000001 で待つ 待ち解除後 LED0 点灯/消灯
	task3	3	2	イベントフラグ 2 の waiptn=00000001 で待つ 待ち解除後 LED1 点灯/消灯
イベントフラグ	－	1	－	待ち解除時ビットクリア指定、 待ちキュー：FIFO 順
	－	2	－	待ち解除時ビットクリア指定、 待ちキュー：FIFO 順
割込みハンドラ	inthdr1	－	－	SW1 で起動：ベクタ番号 72 イベントフラグ 2 に setptn=00000010 をセット
	inthdr2	－	－	SW2 で起動：ベクタ番号 73 イベントフラグ 1 に setptn=00000001 をセット
	inthdr3	－	－	SW3 で起動：ベクタ番号 67 イベントフラグ 2 に setptn=00000001 のセット

(b)　アプリケーションの動作



【注】sta_tsk はタスクを起動するサービスコールです。
wai_flg はイベントフラグを待つサービスコールです。
iset_flg はイベントフラグをセットするサービスコールです。

図3－23　アプリケーションの動作（SAMPLE4）

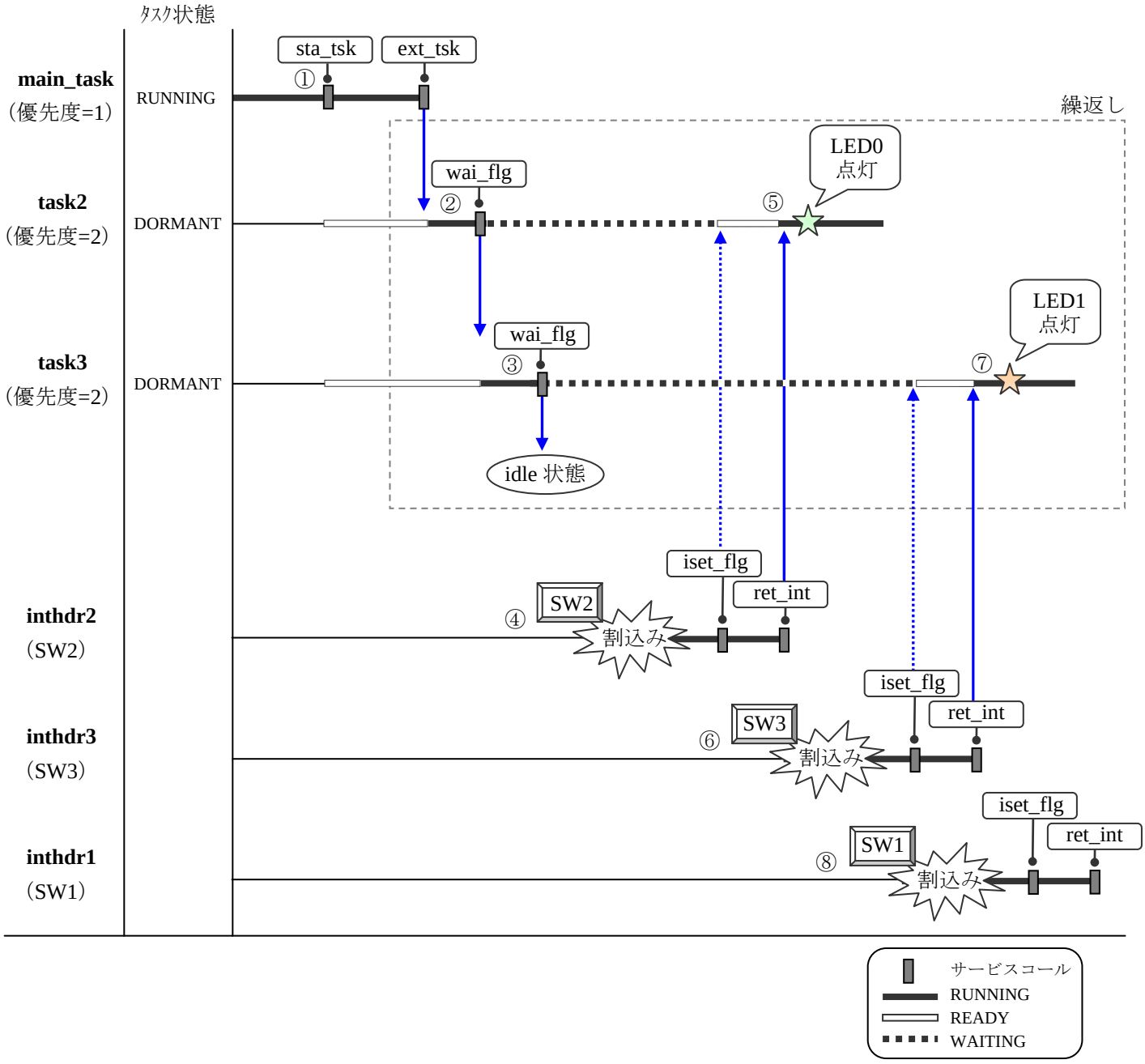


図3－24　アプリケーションのシーケンス（SAMPLE4）

① sta_tsk サービスコールで task2 と task3 を起動し、ext_tsk サービスコールで終了します。

② task2 が RUNNING 状態になり wai_flg サービスコール（ID=1）でイベントフラグ待ちに移行します。waiptn は 0x00000001 です。

③ 次に task3 が RUNNING 状態になり wai_flg サービスコール（ID=2）でイベントフラグ待ちに移行します。waiptn は 0x00000001 です。

④ SW2 を押して、inthdr2 が起動され、iset_flg サービスコール（ID=1）で task2 のイベント待ちが解除されます。

⑤ task2 はイベントフラグ待ちが解除され、RUNNING 状態になり、LED0 を点灯または消灯させます。

⑥ SW3 を押して、inthdr3 が起動され、iset_flg サービスコール（ID=2）で task3 のイベント待ちが解除されます。

⑦ task3 はイベントフラグ待ちが解除され、RUNNING 状態になり、LED1 を点灯または消灯させます。

⑧ SW1 を押して、inthdr1 が起動され、iset_flg サービスコール（ID=2）で setptn=0x00000010 をセットしますが、task3 は waiptn=0x00000001 で待っているため、イベントフラグ待ちは解除されません。

(c)　初期化処理の追加

プロジェクトの生成時に HEW が生成した初期化処理にハードウェアの初期化処理を追加します。

ハードウェアセットアップファイル hwsetup.c の関数にハードウェア初期化処理“RI_hwsetup()”を記述します。

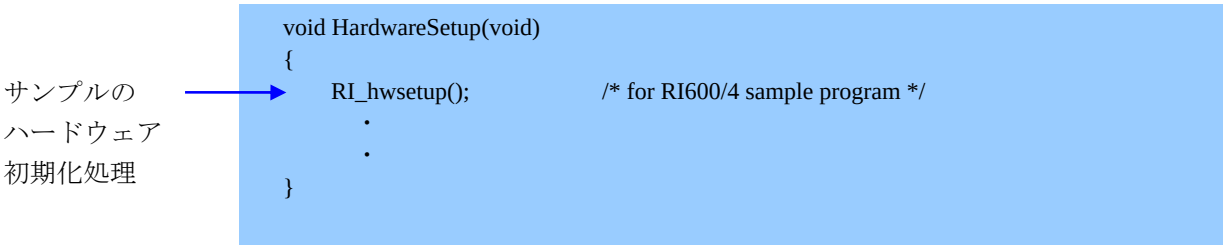


図3－25　初期化処理の追加（SAMPLE4）

(2)　コンフィギュレーションファイルの作成

HEW の[ツール]から GUI コンフィギュレータを起動して、コンフィギュレーションファイルを作成します。

ここでは、タスクの情報、イベントフラグの情報、割込みハンドラの情報、およびサンプルのタイマ用割込みハンドラを定義します。

サンプルのタイマ用割込みハンドラの定義については『2.1.3　コンフィギュレーションファイル（cfgファイル）の作成』をご参照ください。

(a)　タスクの生成に必要な情報を入力します。

サンプルプログラムで定義するタスクの情報を 表3－8に示します。

表3－8　GUIコンフィギュレータ　タスクの定義情報（SAMPLE4）

No.	項目	タスク 1	タスク 2	タスク 3
1	[ID 番号]	1	2	3
2	[ID 名称]	ID_TASK1	ID_TASK2	ID_TASK3
3	[アドレス]	main_task	task2	task3
4	[タスク起動時の優先度]	1	2	2
5	[属性]	[生成後、起動]を指定します。	指定しません。（デフォルト）	
6	[スタックサイズ]	0x00000100（デフォルト）		
7	[スタック領域配置]	[セクション名]に“SURI_STACK”を指定します。（デフォルト）		
8	[記述言語]	[高級言語]（デフォルト）		
9	[拡張情報]	0x00000000（デフォルト）		

［タスク］ウィンドウからダイアログを開いて、タスクの情報を定義します。

図3－26　GUIコンフィギュレータ　タスクの定義（SAMPLE4）

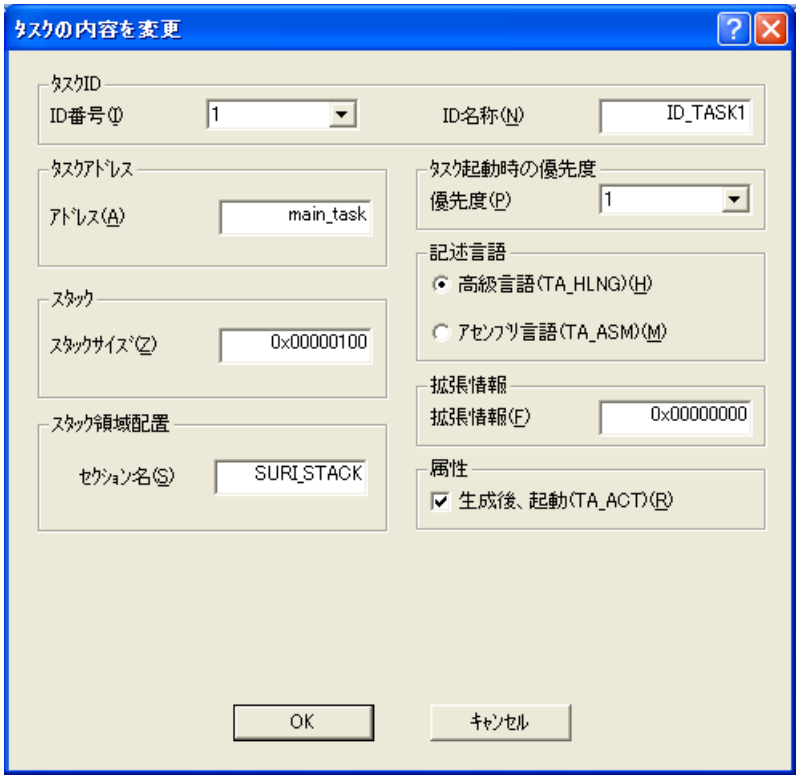


図3－27　GUIコンフィギュレータ　タスクの定義［ダイアログ］（SAMPLE4）

(b)　イベントフラグの生成に必要な情報を入力します。

サンプルプログラムで定義するイベントフラグの情報を 表3−9に示します。

表3−9　GUIコンフィギュレータ　イベントフラグ定義情報（SAMPLE4）

No.	項目	イベントフラグ 1	イベントフラグ 2
1	[ID 番号]	1	2
2	[ID 名称]	ID_FLG1	ID_FLG2
3	[初期ビットパターン]	0x00000000（デフォルト）	
4	[属性]	[複数タスクの待ちを許可(TA_WMUL)]を指定します。	
5		[待ち解除時にビットクリア(TA_CLR)]を指定します。	
6	[待ちキューの並び方]	[FIFO 順(TA_TFIFO)]を指定します。（デフォルト）	

［イベントフラグ］ウィンドウからダイアログを開いて、イベントフラグの情報を定義します。

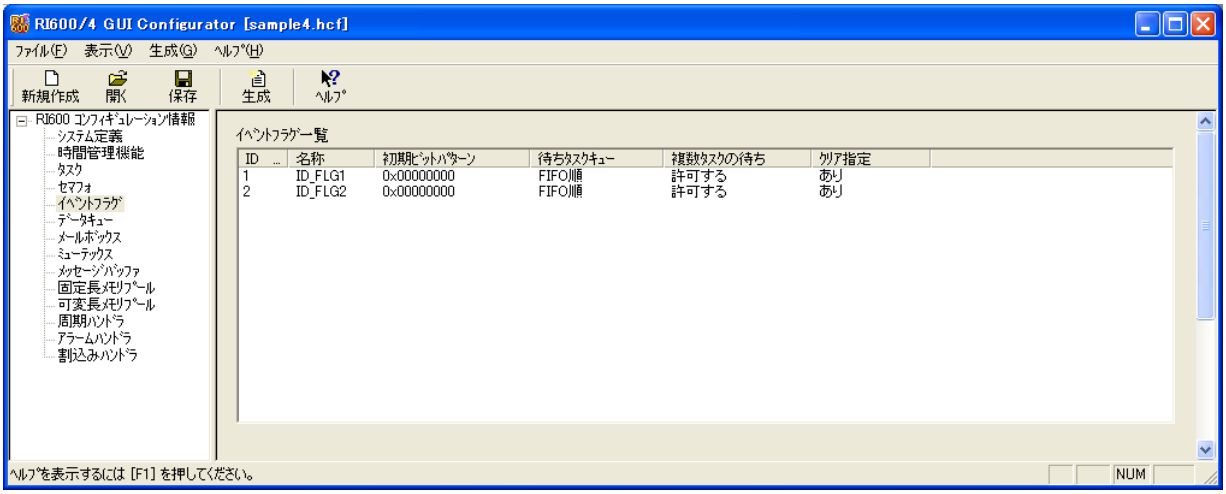


図3−28　GUIコンフィギュレータ　イベントフラグの定義（SAMPLE4）

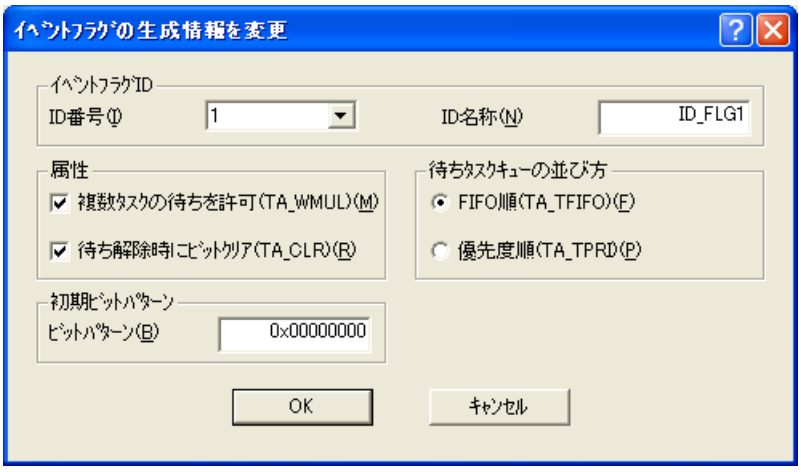


図3−29　GUIコンフィギュレータ　イベントフラグの定義［ダイアログ］（SAMPLE4）

(c)　割込みハンドラの定義に必要な情報を入力します。

サンプルプログラムで定義する割込みハンドラの情報を 表3－10に示します。

表3－10　GUIコンフィギュレータ　割込みハンドラの定義情報（SAMPLE4）

No.	項目	割込み 1	割込み 2	割込み 3
1	[ベクタ番号]	67	72	73
2	[アドレス]	inthdr3	inthdr1	inthdr2
3	[割込種別]	[カーネル管理外の割込み]を指定しません（デフォルト）		
4	[記述言語]	[高級言語]（デフォルト）		
5	[#PRAGMA 拡張機能スイッチ]	[多重割り込みを許可する。]を指定しません（デフォルト）		
6		[使用するレジスタを制限する。]を指定しません（デフォルト）		

［割込みハンドラ］ウィンドウからダイアログを開いて、割込みハンドラの情報を定義します。

図3－30　GUIコンフィギュレータ　割込みハンドラの定義（SAMPLE4）

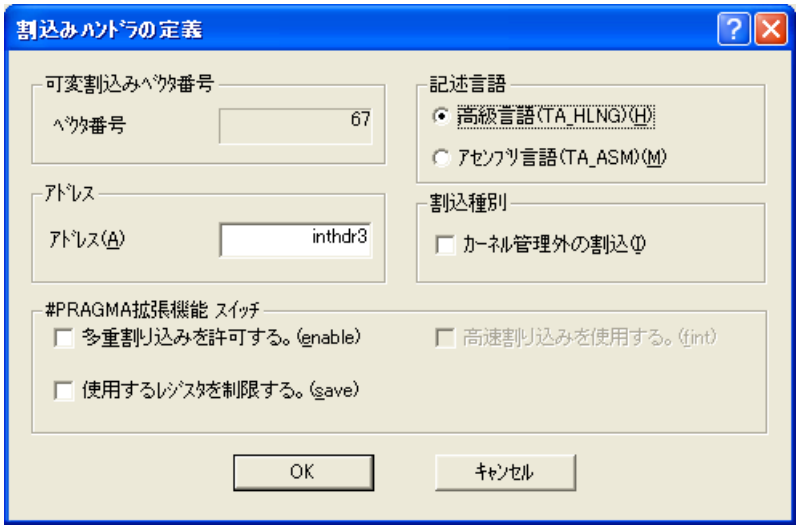


図3－31　GUIコンフィギュレータ　割込みハンドラの定義［ダイアログ］（SAMPLE4）

(d)　コンフィギュレーションファイルを生成します。

[生成]でコンフギュレーションファイル（cfgファイル）を生成します。SAMPLE4では、sample4.cfgを生成します。

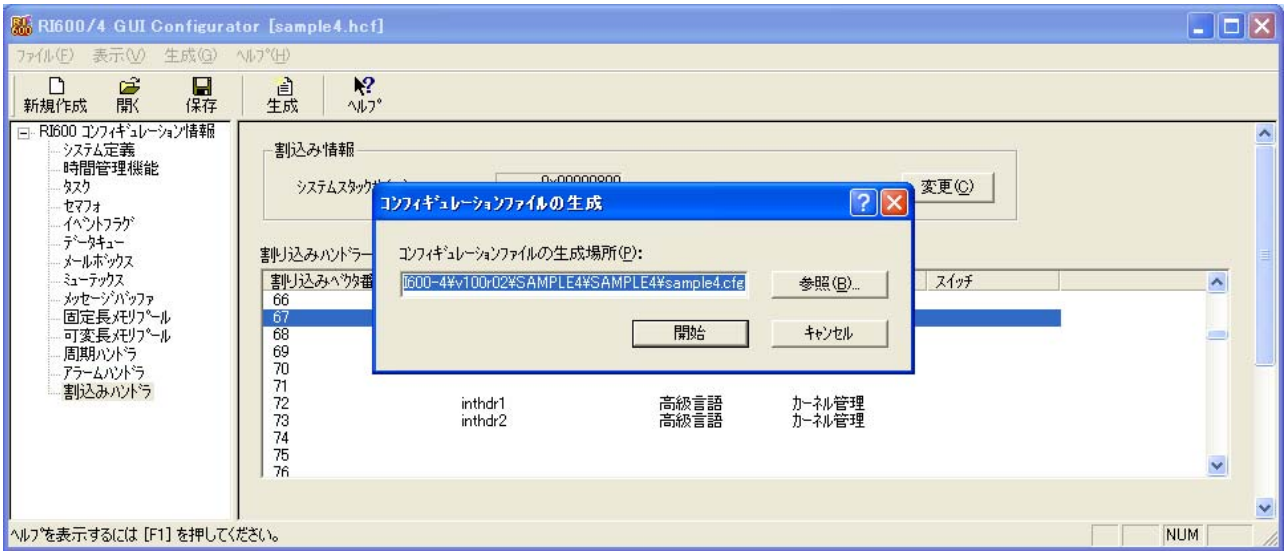


図3－32　コンフィギュレーションファイルの生成（SAMPLE4）

(3) ロードモジュールの作成

HEW のビルド機能を使って、ロードモジュールを作成します。

ロードモジュールの作成については『2.1.4　ロードモジュールの作成』をご参照ください。

(4) プログラムの実行

作成したプログラムを実際に実行します。

HEW のデバッグ機能を使って、デバッグの設定を行い、作成したロードモジュールをダウンロードします。

[実行]でプログラムが実行します。

プログラムの実行については『2.1.5　プログラムの実行』をご参照ください。

(5) プログラムの実行結果

スイッチにより起動される割込みハンドラから、タスクのイベントフラグ待ちが解除されます。

SW2 を押すと LED0 が点灯し、再び押すと消灯します。交互に点灯と消灯を繰り返します。

同様に、SW3 を押すと、LED1 が点灯と消灯を繰り返します。

また、SW1 を押しても何も変わらないことを確認します。

なお、サンプルプログラムではスイッチ（SW1、SW2、SW3）に対するチャタリング対策が施されていません。スイッチの 1 回の操作で点灯と消灯を行うことがありますので、ご注意ください。

HEW の RTOS 機能で、タスクとイベントフラグの状態を確認します。

[表示]－[RTOS]－[OS オブジェクト]でウィンドウを開きます。

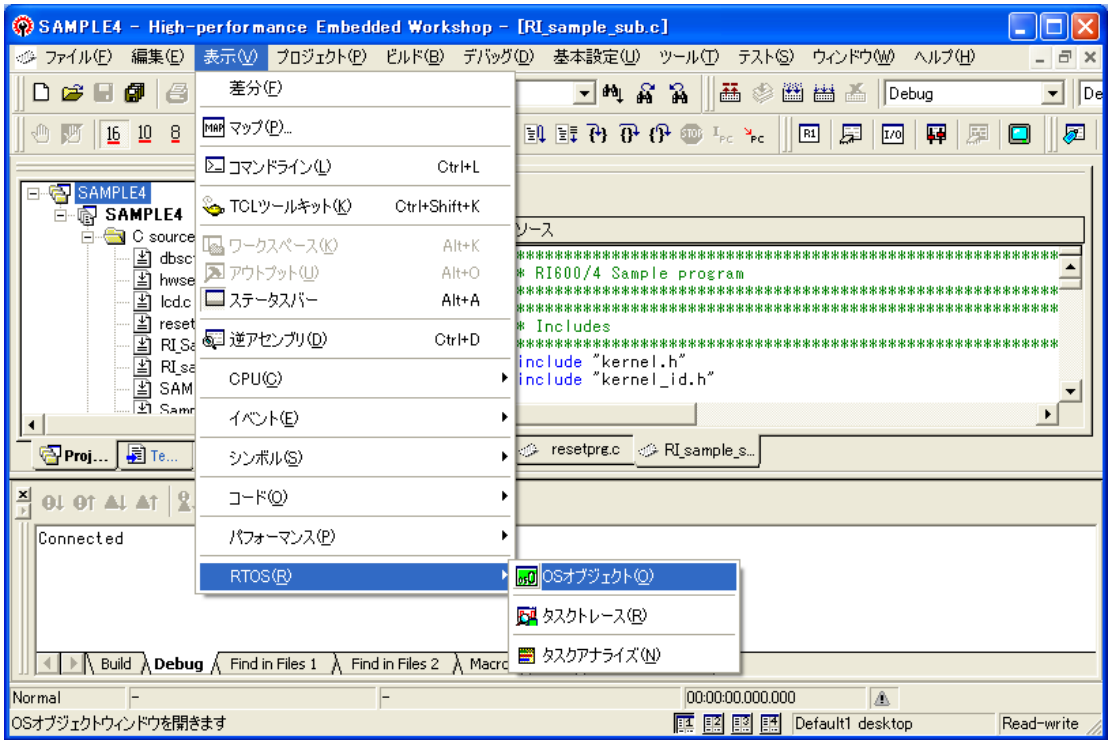


図3－33　プログラムの実行結果　RTOS機能（SAMPLE4）

main_task（ID=1）が休止状態（DORMANT）となり、task2（ID=2）と task3（ID=3）がイベントフラグ待ちになります。

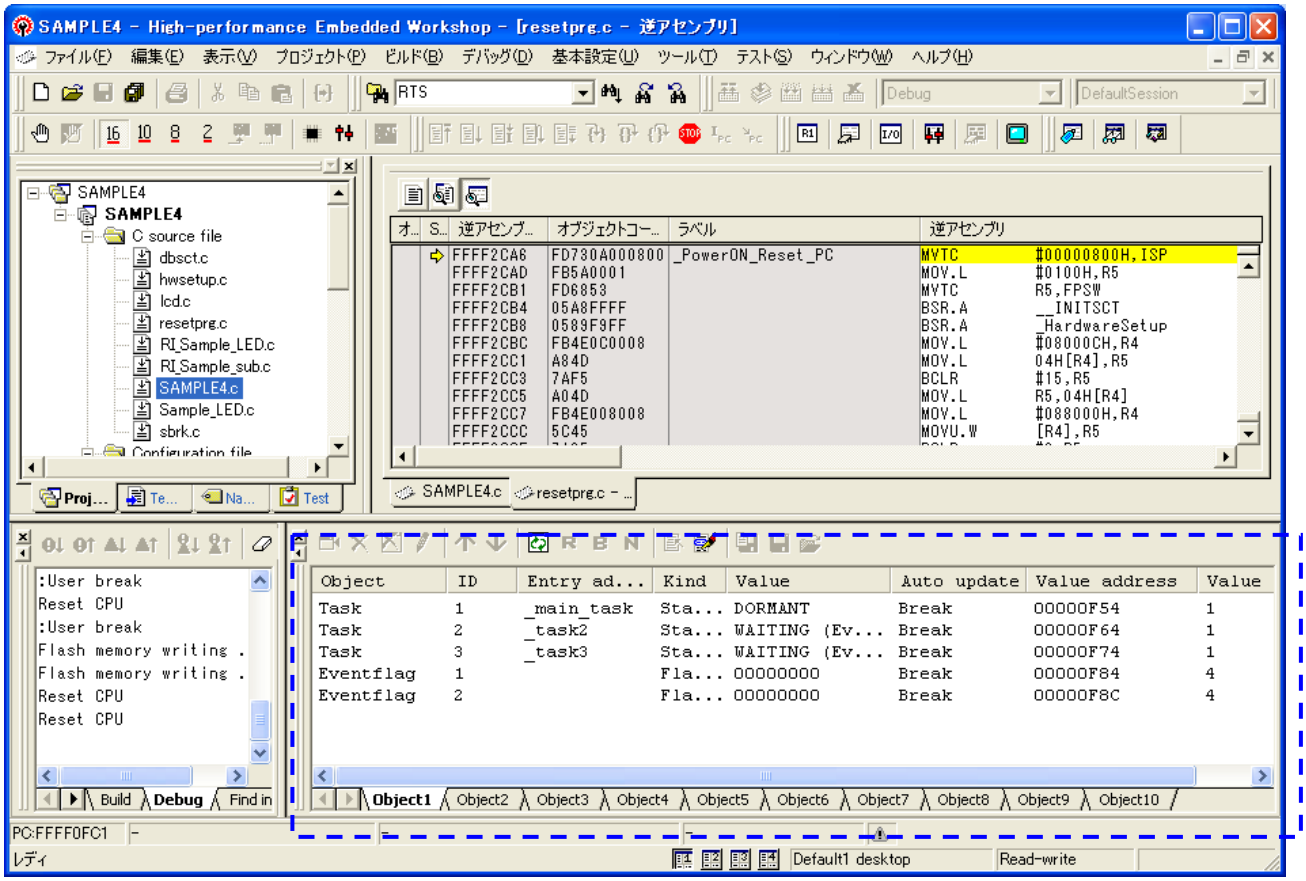
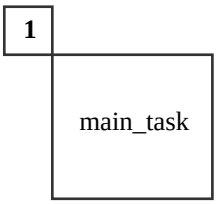


図3－34　プログラムの実行結果　タスクとイベントフラグの状態（SAMPLE4）

以上で、SAMPLE4のシステムは完了です。

(6)　サンプルプログラム

SAMPLE4のタスクと割込みハンドラのソースコードは 図3－35のように記述しています。



カーネル提供ファイル

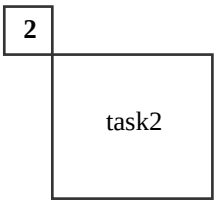
cfg600 生成ファイル

task2 起動（sta_tsk）

task3 起動（sta_tsk）

```
/*************************************************************************
*
*     RI600/4 Sample program
*
*************************************************************************/

#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

#include "RI_sample.h"                /* table for sample */

/*************************************************************************
  main_task
*************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd= 0;
      Number = 0L;
      test= 0;

      ercd = sta_tsk(ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;
      ercd = sta_tsk(ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }
      Number++;

      ext_tsk();
}
```

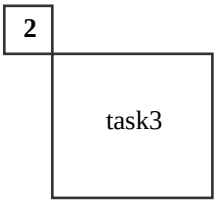
図3－35　サンプルプログラムソースコード『SAMPLE4.c』（1/3）




```
/***********************************************************************
  task2
***********************************************************************/
void task2(VP_INT exinf)
{
        ER    ercd, Number;
        FLGPTN waiptn;
        MODE wfmode;
        FLGPTN p_flgptn;

        Number = 0L;
        waiptn = 0x00000001;
        wfmode = TWF_ANDW;

        while(1){
                ercd = wai_flg(ID_FLG1, waiptn, wfmode, &p_flgptn);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_FLG1, (VW)Number, 0L, 0L);
                }
                led_0();
                sample_delay(1000);
                Number++;
        }
        ext_tsk();
}
```

イベントフラグ待ち（wai_flg）

LED 点灯処理

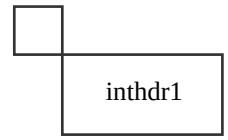



```
/***********************************************************************
  task3
***********************************************************************/
void task3(VP_INT exinf)
{
        ER    ercd, Number;
        FLGPTN waiptn;
        MODE wfmode;
        FLGPTN p_flgptn;

        Number = 0L;
        waiptn = 0x00000001;
        wfmode = TWF_ANDW;

        while(1){
                ercd = wai_flg(ID_FLG2, waiptn, wfmode, &p_flgptn);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_FLG2, (VW)Number, 0L, 0L);
                }
                led_1();
                sample_delay(1000);
                Number++;
        }
        ext_tsk();
}
```

イベントフラグ待ち（wai_flg）

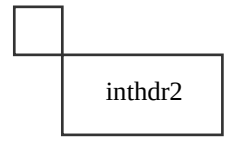
LED 点灯処理

図3－35　サンプルプログラムソースコード『SAMPLE4.c』（2/3）



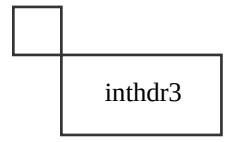
イベントフラグセット（iset_flg）

```
/***********************************************************************
 inthdr1 [SW1]
***********************************************************************/
void inthdr1(void)
{
    ER    ercd, Number;
    FLGPTN setptn;

    Number = 0L;
    setptn = 0x00000010;
    ercd = iset_flg(ID_FLG2, setptn);
    if(ercd != E_OK)
    {
        vsys_dwn((ER)ID_FLG2, (VW)Number, 0L, 0L);
    }
    Number++;
}
```



イベントフラグセット（iset_flg）

```
/***********************************************************************
 inthdr2 [SW2]
***********************************************************************/
void inthdr2(void)
{
    ER    ercd, Number;
    FLGPTN setptn;

    Number = 0L;
    setptn = 0x00000001;
    ercd = iset_flg(ID_FLG1, setptn);
    if(ercd != E_OK)
    {
        vsys_dwn((ER)ID_FLG1, (VW)Number, 0L, 0L);
    }
    Number++;
}
```



イベントフラグセット（iset_flg）

```
/***********************************************************************
 inthdr3 [SW3]
***********************************************************************/
void inthdr3(void)
{
    ER    ercd, Number;
    FLGPTN setptn;

    Number = 0L;
    setptn = 0x00000001;
    ercd = iset_flg(ID_FLG2, setptn);
    if(ercd != E_OK)
    {
        vsys_dwn((ER)ID_FLG2, (VW)Number, 0L, 0L);
    }
    Number++;
}

/***********************************************************************
***********************************************************************/
```

図3－35　サンプルプログラムソースコード『SAMPLE4.c』（3/3）

3.1.5 SAMPLE5：タスクと周期ハンドラ

SAMPLE5では、時間管理機能を使って、周期ハンドラを起動します。
周期ハンドラは、タスクから起動を開始します。

周期ハンドラは、一定周期で起動されるタイムイベントハンドラです。

(1) アプリケーションプログラム

SAMPLE5は、main_task、task2、task3 の 3 つタスクと、cychdr1 とcychdr2 の 2 つの周期ハンドラからなります。

main_task から起動された task2 と task3 は、cychdr1 と cychdr2 の起動を開始します。

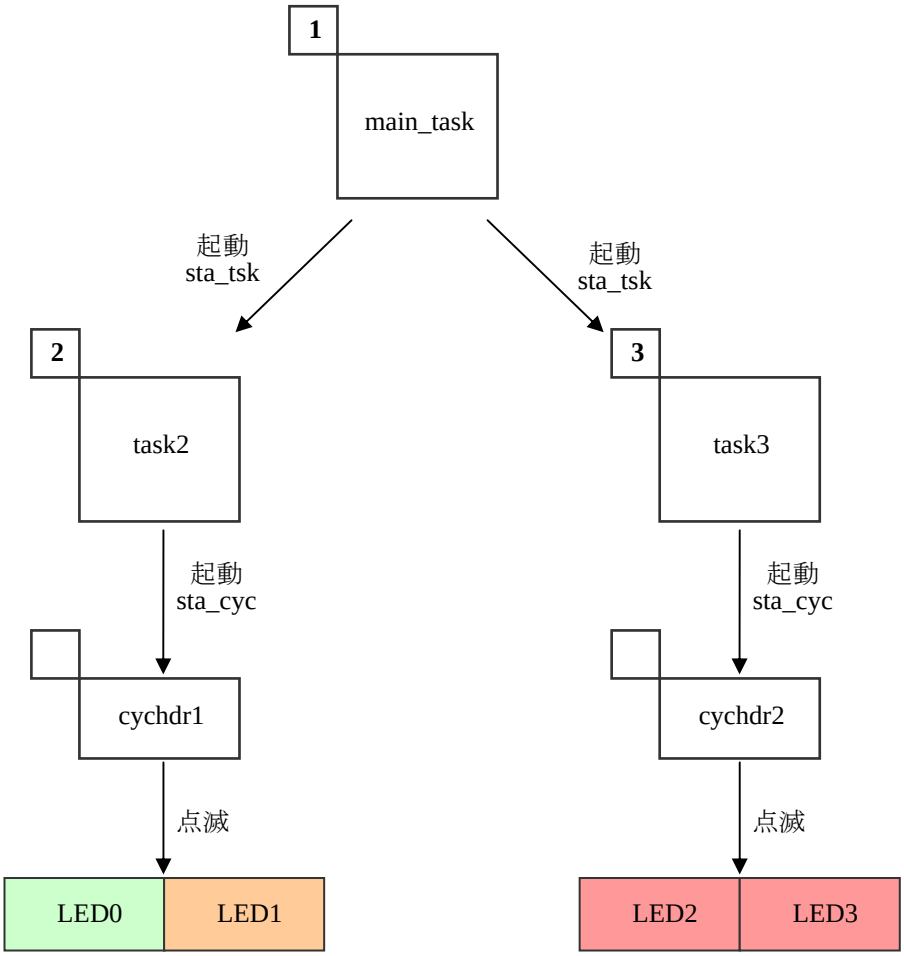
cychdr1 と cychdr2 は LED の点滅処理を行います。

(a) 使用するオブジェクト

表3－11 使用するオブジェクト（SAMPLE5）

オブジェクト	アドレス	ID 番号	優先度	動作
タスク	main_task	1	1	初期起動 task2 の起動、task3 の起動
	task2	2	2	周期ハンドラ 1 の起動開始
	task3	3	3	周期ハンドラ 2 の起動開始
周期ハンドラ	cychdr1	1	－	起動周期 200ms、LED0 と LED1 点滅
	cychdr2	2	－	起動周期 400ms、LED2 と LED3 点滅

(b)　アプリケーションの動作



【注】sta_tsk はタスクを起動するサービスコールです。
　　sta_cyc は周期ハンドラの起動を開始するサービスコールです。

図3－36　アプリケーションの動作（SAMPLE5）

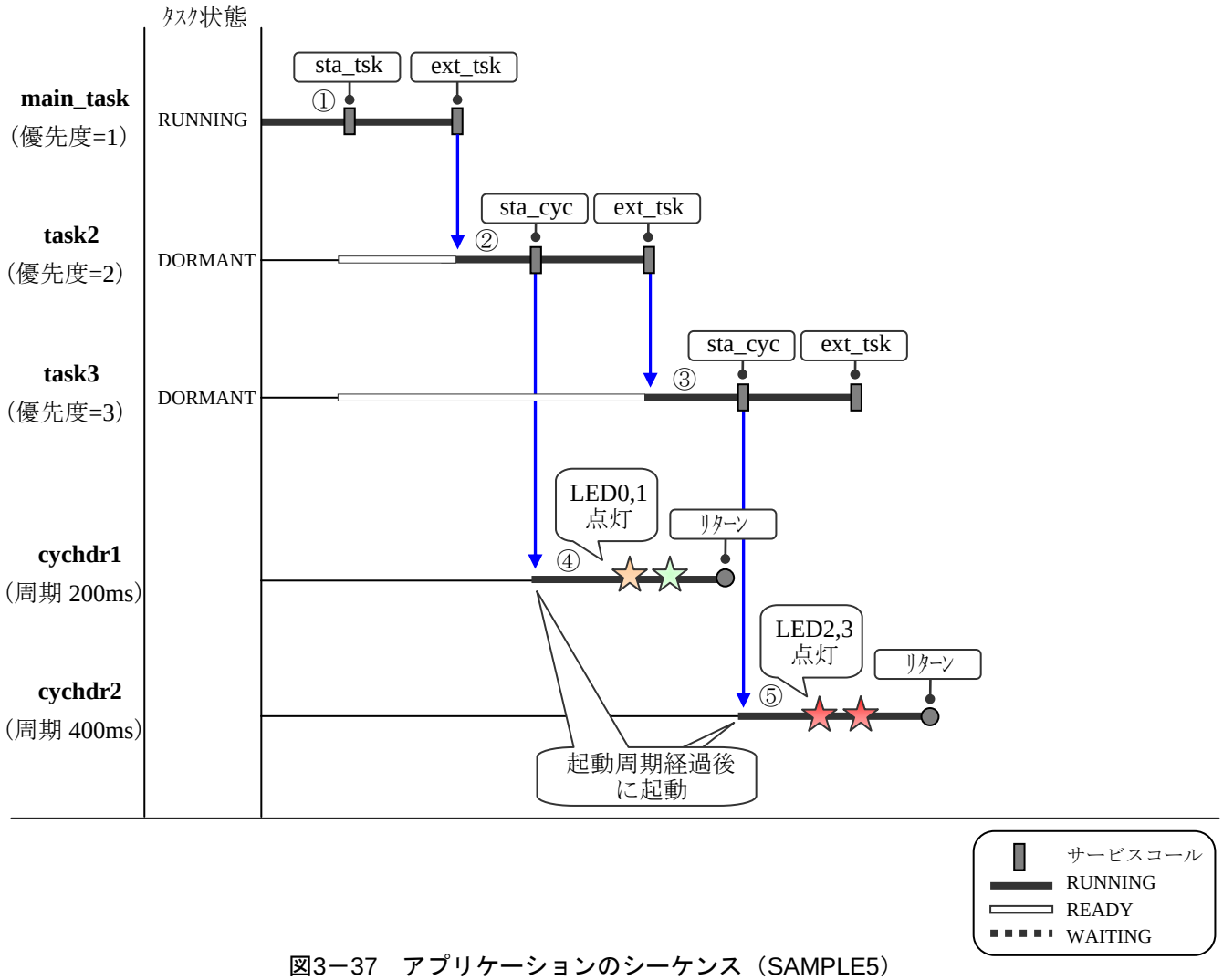


図3－37　アプリケーションのシーケンス（SAMPLE5）

① sta_tsk サービスコールで task2 と task3 を起動し、ext_tsk サービスコールで終了します。

② task2 が RUNNING 状態になり sta_cyc サービスコールで cychdr1 の起動を開始し、ext_tsk サービスコールで終了します。

③ 次に task3 が RUNNING 状態になり sta_cyc サービスコールで cychdr2 の起動を開始し、ext_tsk サービスコールで終了します。

④ cychdr1 が 200ms 周期で起動され、LED0 と LED1 を点滅させます。

⑤ cychdr2 が 400ms 周期で起動され、LED2 と LED3 を点滅させます。

(c)　初期化処理の追加

プロジェクトの生成時に HEW が生成した初期化処理にハードウェアの初期化処理を追加します。

ハードウェアセットアップファイル hwsetup.c の関数にハードウェア初期化処理“RI_hwsetup()”を記述します。

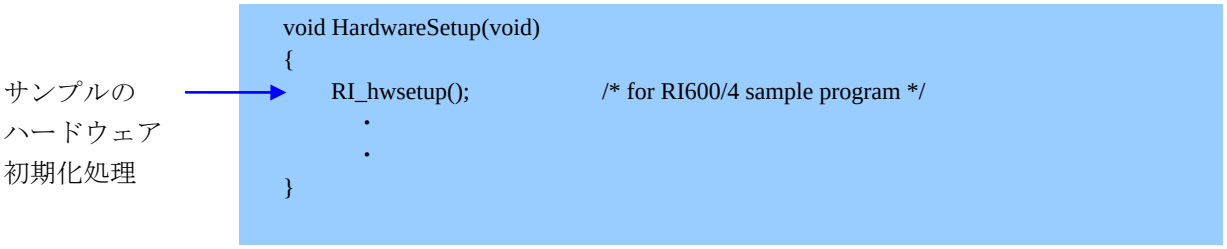


図3－38　初期化処理の追加（SAMPLE5）

(2)　コンフィギュレーションファイルの作成

HEW の[ツール]から GUI コンフィギュレータを起動して、コンフィギュレーションファイルを作成します。

ここでは、タスクの情報、周期ハンドラの情報、およびサンプルのタイマ用割込みハンドラを定義します。

サンプルのタイマ用割込みハンドラの定義については『2.1.3　コンフィギュレーションファイル（cfgファイル）の作成』をご参照ください。

(a)　タスクの生成に必要な情報を入力します。

サンプルプログラムで定義するタスクの情報を 表3－12に示します。

表3－12　GUIコンフィギュレータ　タスクの定義情報（SAMPLE5）

No.	項目	タスク 1	タスク 2	タスク 3
1	[ID 番号]	1	2	3
2	[ID 名称]	ID_TASK1	ID_TASK2	ID_TASK3
3	[アドレス]	main_task	task2	task3
4	[タスク起動時の優先度]	1	2	3
5	[属性]	[生成後、起動]を指定します。	指定しません。（デフォルト）	
6	[スタックサイズ]	0x00000100（デフォルト）		
7	[スタック領域配置]	[セクション名]に“SURI_STACK”を指定します。（デフォルト）		
8	[記述言語]	[高級言語]（デフォルト）		
9	[拡張情報]	0x00000000（デフォルト）		

［タスク］ウィンドウからダイアログを開いて、タスクの情報を定義します。

図3－39　GUIコンフィギュレータ　タスクの定義（SAMPLE5）

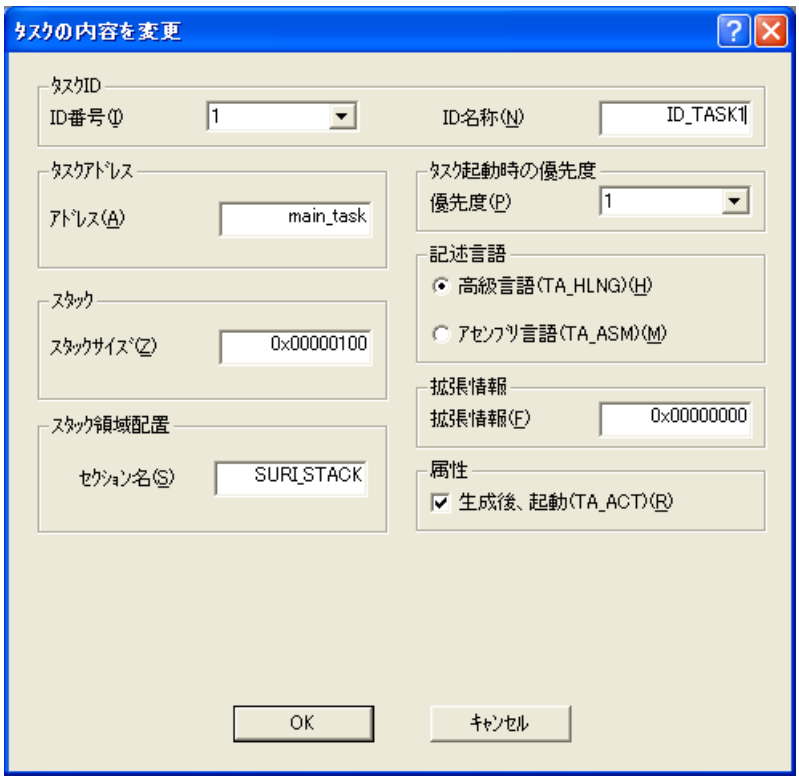


図3－40　GUIコンフィギュレータ　タスクの定義［ダイアログ］（SAMPLE5）

(b)　周期ハンドラの生成に必要な情報を入力します。

サンプルプログラムで定義する周期ハンドラの情報を 表3－13に示します。

表3－13　GUIコンフィギュレータ　周期ハンドラの定義情報（SAMPLE5）

No.	項目	周期ハンドラ 1	周期ハンドラ 2
1	[ID 番号]	1	2
2	[ID 名称]	ID_CYC1	ID_CYC2
3	[アドレス]	cychdr1	cychdr2
4	[起動周期]	200ms	400ms
5	[拡張情報]	0（デフォルト）	
6	[起動位相]	0（デフォルト）	
7	[属性]	[生成後、動作状態へ(TA_STA)]を指定しません。（デフォルト）	
8		[起動位相を保存する(TA_PHS)]を指定しません。（デフォルト）	
9	[記述言語]	[高級言語]（デフォルト）	

［周期ハンドラ］ウィンドウからダイアログを開いて、周期ハンドラの情報を定義します。

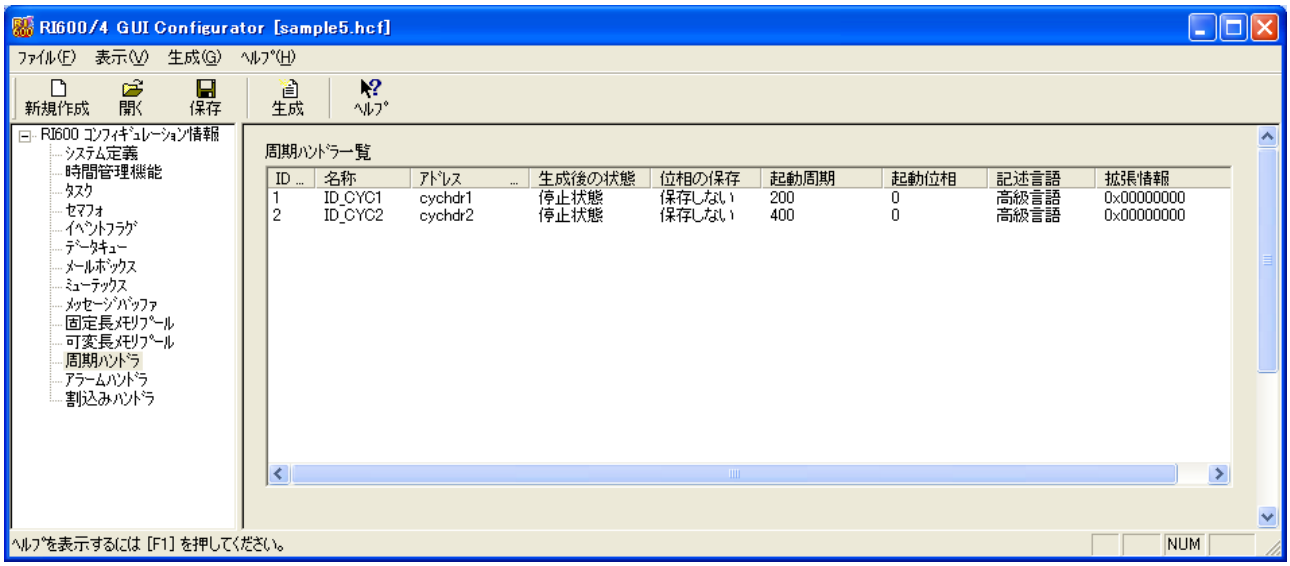


図3－41　GUIコンフィギュレータ　周期ハンドラの定義（SAMPLE5）

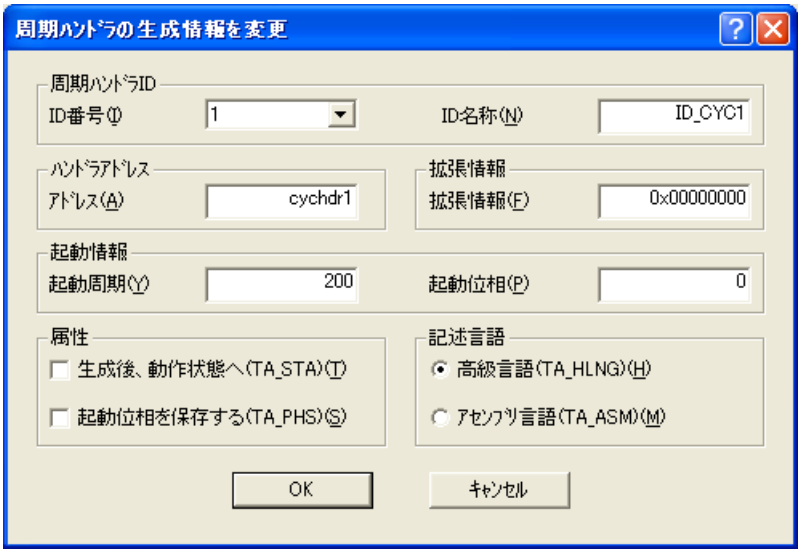


図3－42　GUIコンフィギュレータ　周期ハンドラの定義［ダイアログ］（SAMPLE5）

(c)　コンフィギュレーションファイルを生成します。

[生成]でコンフギュレーションファイル（cfgファイル）を生成します。SAMPLE5では、sample5.cfgを生成します。

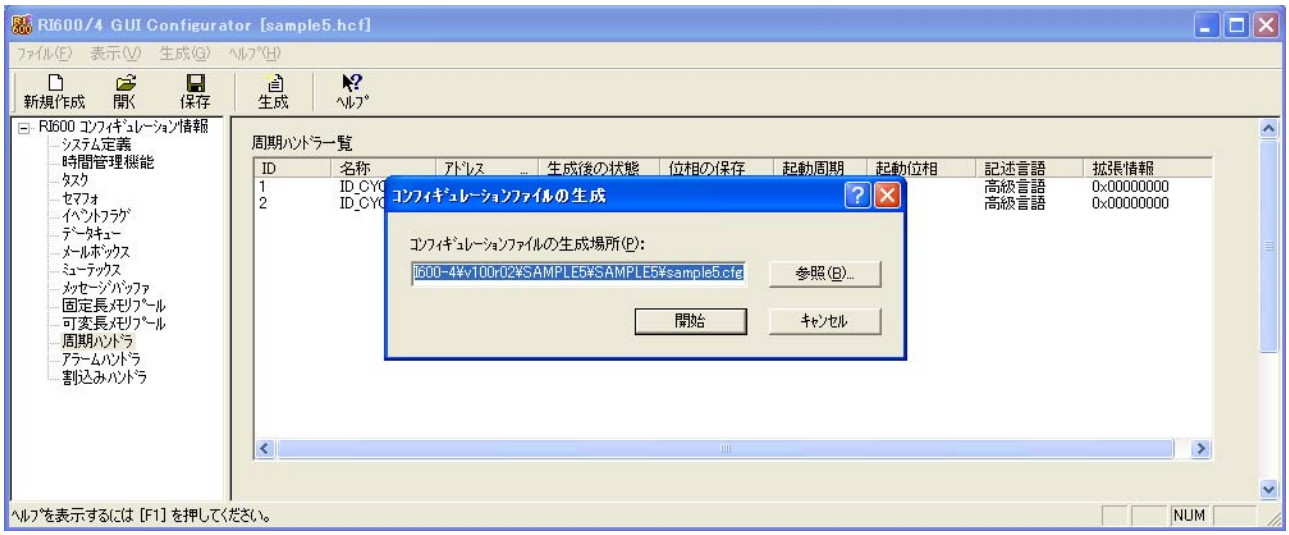


図3－43　コンフィギュレーションファイルの生成（SAMPLE5）

(3) ロードモジュールの作成

HEW のビルド機能を使って、ロードモジュールを作成します。

ロードモジュールの作成については『2.1.4　ロードモジュールの作成』をご参照ください。

(4) プログラムの実行

作成したプログラムを実際に実行します。

HEW のデバッグ機能を使って、デバッグの設定を行い、作成したロードモジュールをダウンロードします。

[実行]でプログラムが実行します。

プログラムの実行については『2.1.5　プログラムの実行』をご参照ください。

(5) プログラムの実行結果

cychdr1 の起動により、起動周期毎に LED0 と LED1 が点滅します。また、cychdr2 の起動により、起動周期毎に LED2 と LED3 が点滅します。

HEW の RTOS 機能で、タスクと周期ハンドラの状態を確認します。

[表示]－[RTOS]－[OS オブジェクト]でウィンドウを開きます。

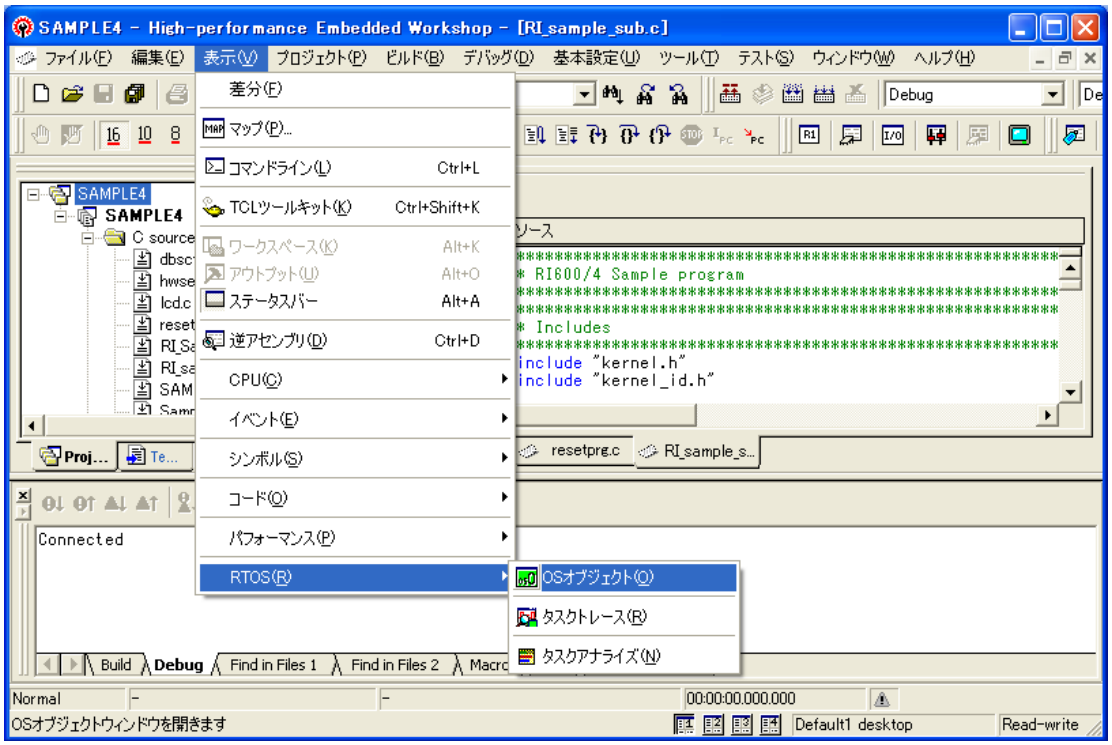


図3－44　プログラムの実行結果　RTOS機能（SAMPLE5）

main_task（ID=1）から task3（ID=3）までの 3 つのタスクが休止状態（DORMANT）になります。
cychdr1 と cychdr2 は周期時間毎に起動し、LED の点滅を続けます。

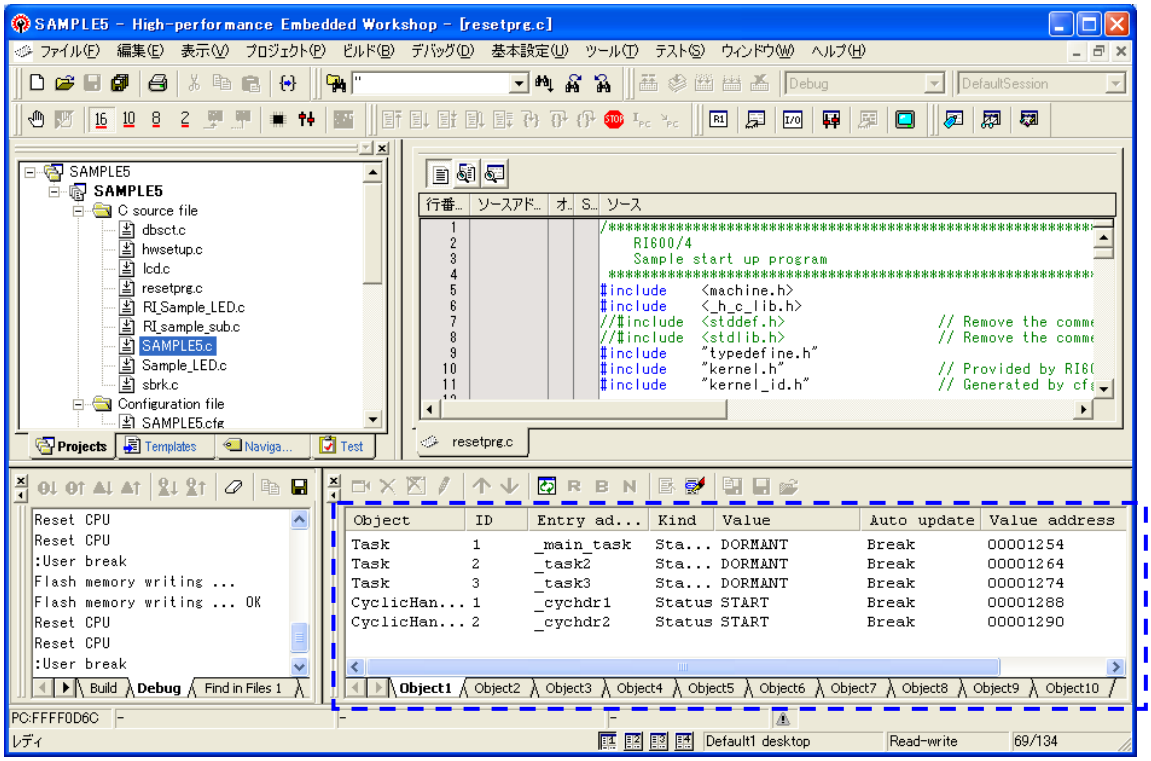
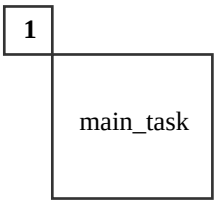


図3－45　プログラムの実行結果　タスクと周期ハンドラの状態（SAMPLE5）

以上で、SAMPLE5のシステムは完了です。

(6)　サンプルプログラム

SAMPLE5のタスクと周期ハンドラのソースコードは 図3－46のように記述しています。



カーネル提供ファイル

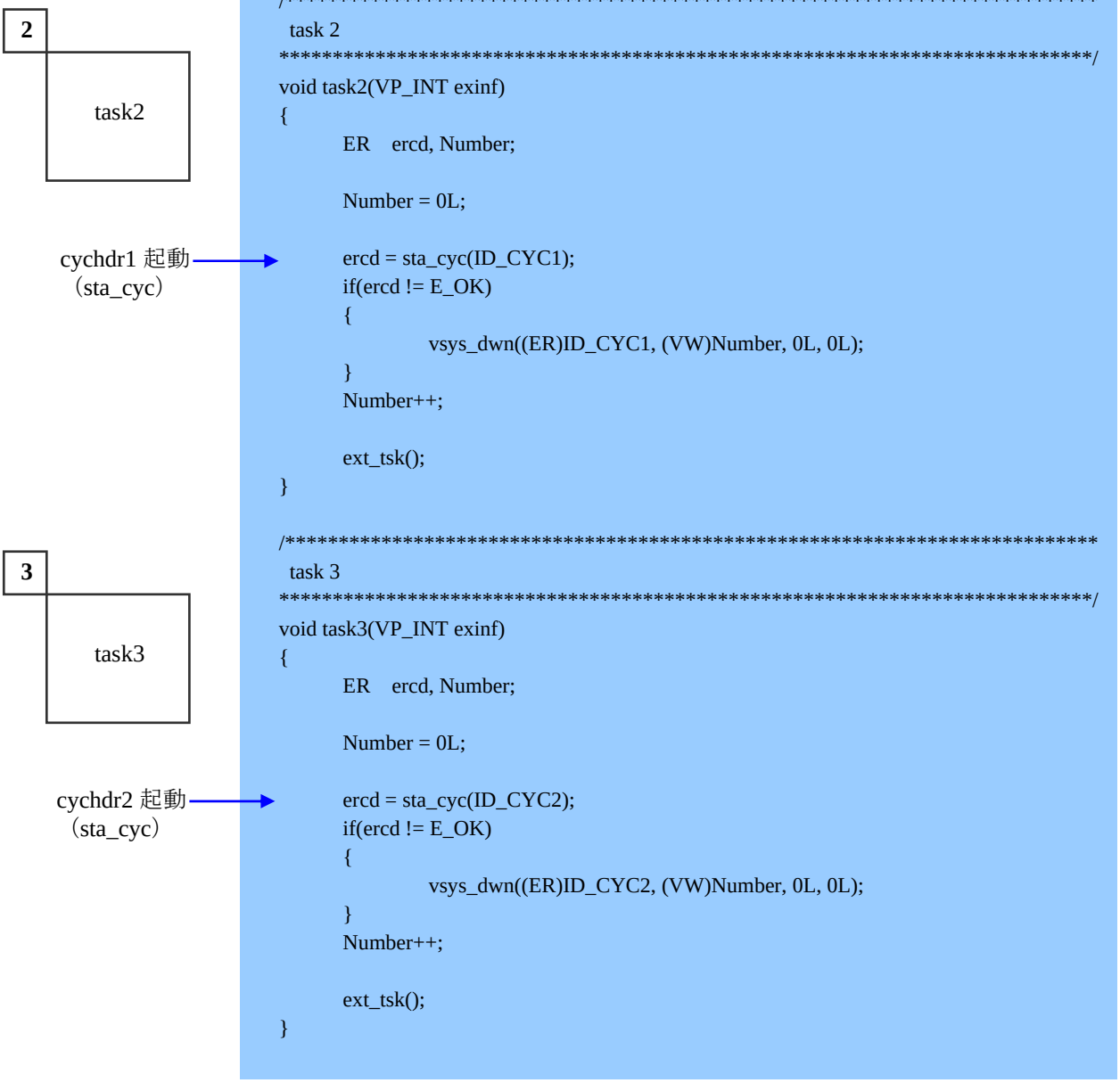
cfg600 生成ファイル

task2 起動（sta_tsk）

task3 起動（sta_tsk）

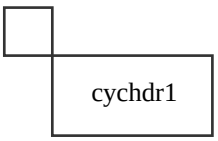
```
/***********************************************************************
  RI600/4
  Sample program
***********************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

#include "RI_sample.h"              // table for sample

/***********************************************************************
  Shared data
***********************************************************************/

/***********************************************************************
  main_task1
***********************************************************************/
void main_task(VP_INT exinf)
{
        ER    ercd, Number;
        VP_INT    stacd;

        Number = 0L;

        ercd = sta_tsk(ID_TASK2,stacd);
        if(ercd != E_OK)
        {
                vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }
        Number++;

        ercd = sta_tsk(ID_TASK3,stacd);
        if(ercd != E_OK)
        {
                vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
        }
        Number++;

        ext_tsk();
}
```

図3－46　サンプルプログラムソースコード『SAMPLE5.c』（1/3）



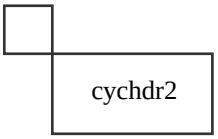

```
/**************************************************************************
 task 2
**************************************************************************/
void task2(VP_INT exinf)
{
        ER    ercd, Number;

        Number = 0L;

        ercd = sta_cyc(ID_CYC1);
        if(ercd != E_OK)
        {
                vsys_dwn((ER)ID_CYC1, (VW)Number, 0L, 0L);
        }
        Number++;

        ext_tsk();
}

/**************************************************************************
 task 3
**************************************************************************/
void task3(VP_INT exinf)
{
        ER    ercd, Number;

        Number = 0L;

        ercd = sta_cyc(ID_CYC2);
        if(ercd != E_OK)
        {
                vsys_dwn((ER)ID_CYC2, (VW)Number, 0L, 0L);
        }
        Number++;

        ext_tsk();
}
```

図3－46　サンプルプログラムソースコード『SAMPLE5.c』（2/3）



LED 点滅処理 →

```
/***********************************************************************
  cyclic handler 1
***********************************************************************/
void cychdr1(VP_INT exinf)
{

    led_0_1();

}

/***********************************************************************
  cyclic handler 2
***********************************************************************/
void cychdr2(VP_INT exinf)
{

    led_2_3();

}

/***********************************************************************
***********************************************************************/
```



LED 点滅処理 →

図3－46　サンプルプログラムソースコード『SAMPLE5.c』（3/3）

3.1.6 SAMPLE6：セマフォ

SAMPLE6では、セマフォを使ってLCDの表示処理の排他制御を行います。

セマフォは、複数のタスクで共有する装置や共有変数といった資源の競合を防ぐためのオブジェクトです。

例えば、タスクが LCD の更新中にタスクスイッチが発生し、別のタスクが LCD に対して更新してしまう可能性があります。セマフォを使用することで、このような競合を回避することができます。

(1) アプリケーションプログラム

SAMPLE6は、main_task、task2、task3 の 3 つのタスクからなります。

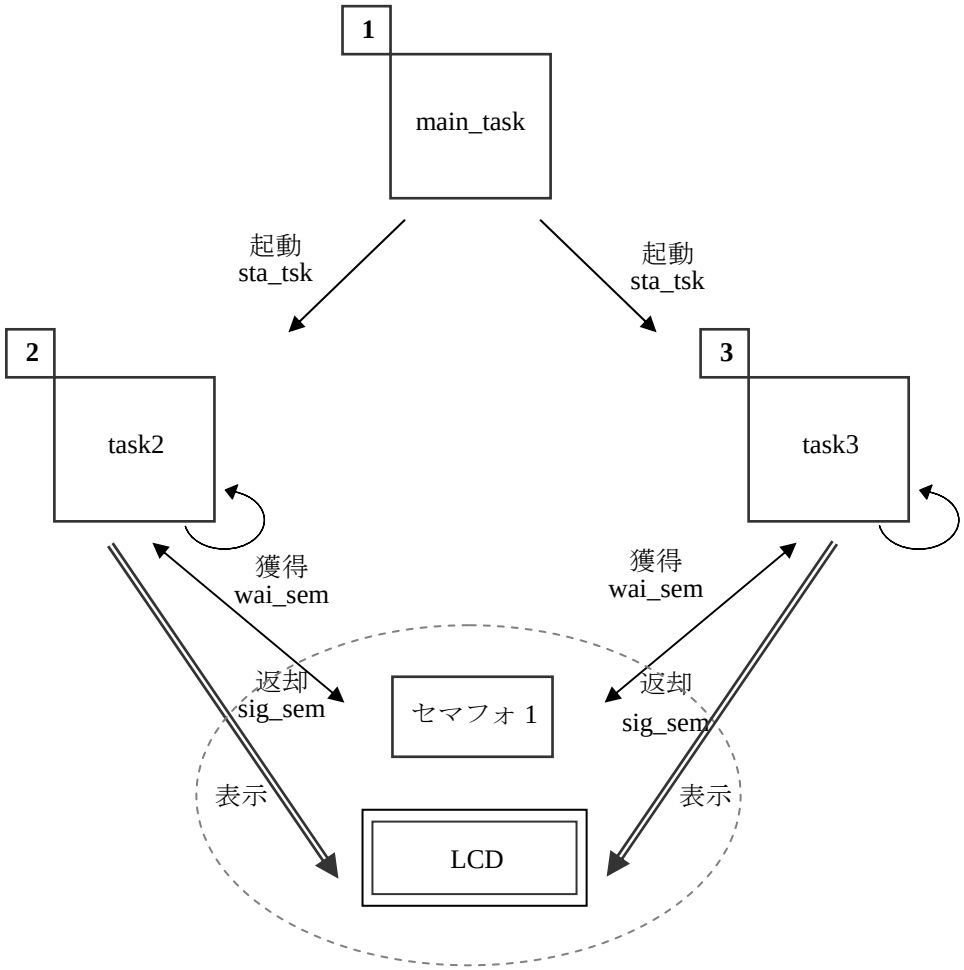
main_task から起動された task2 と task3 は、セマフォを獲得し LCD へ表示処理を行います。表示処理が完了したら、セマフォを返却します。

(a) 使用するオブジェクト

表3－14　使用するオブジェクト（SAMPLE6）

オブジェクト	アドレス	ID 番号	優先度	動作
タスク	main_task	1	1	初期起動 task2 の起動、task3 の起動
	task2	2	2	セマフォの獲得と返却 LCD への表示
	task3	3	3	セマフォの獲得と返却 LCD への表示
セマフォ	－	1	－	セマフォカウンタ初期値：1

(b)　アプリケーションの動作



【注】sta_tsk はタスクを起動するサービスコールです。
wai_sem はセマフォから資源を獲得するサービスコールです。
sig_sem はセマフォに資源を返却するサービスコールです。

図3－47　タスクの動作（SAMPLE6）

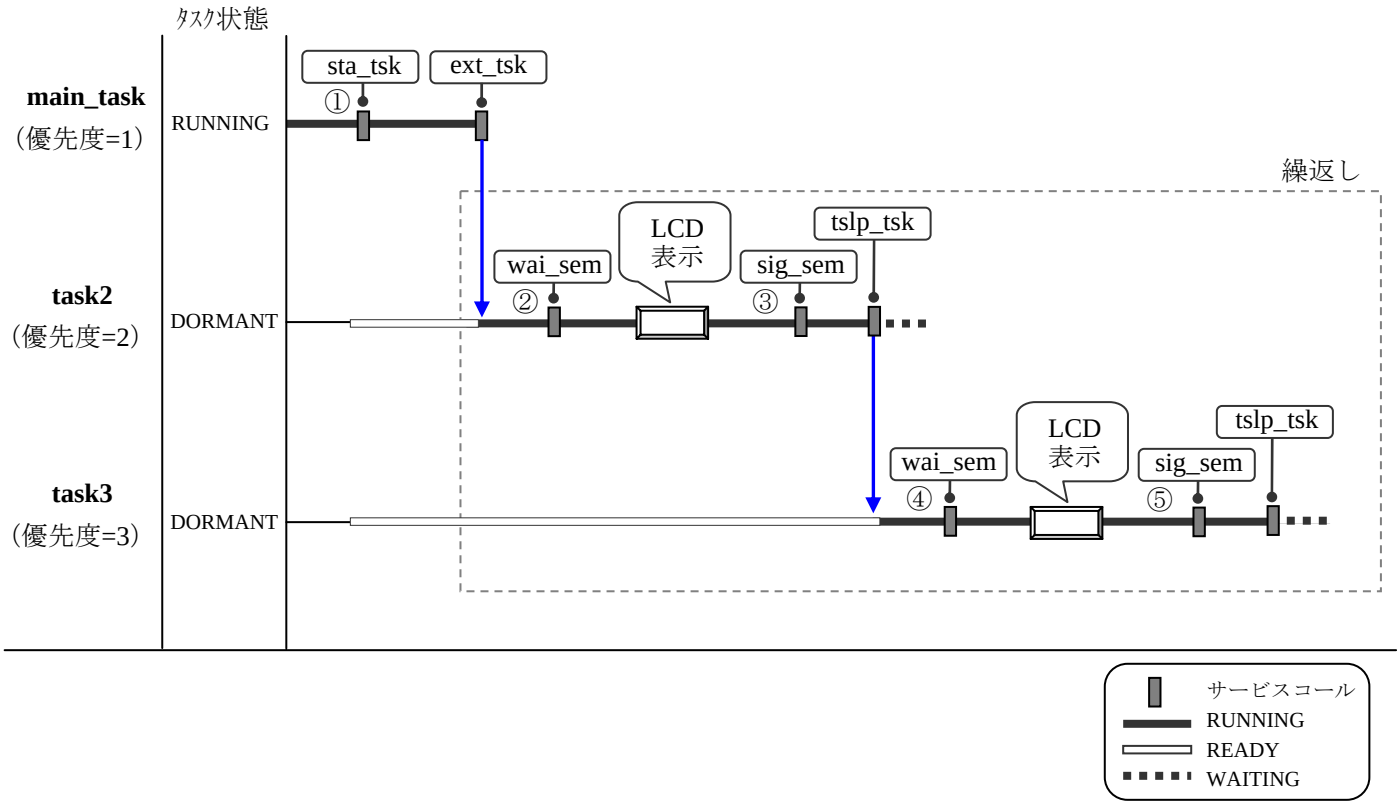


図3−48　アプリケーションのシーケンス（SAMPLE6）

① sta_tsk サービスコールで task2 と task3 を起動し、ext_tsk サービスコールで終了します。

② task2 が RUNNING 状態になり wai_sem サービスコールで資源を獲得して、LCD への表示を行います。

③ sig_sem サービスコールで資源を返却します。次のタスクを動作させるため tslp_tsk サービスコールで待ちに移行します。

④ 次に task3 が RUNNING 状態になり wai_sem サービスコールで資源を獲得して、LCD への表示を行います。

⑤ sig_sem サービスコールで資源を返却します。次のタスクを動作させるため tslp_tsk サービスコールで待ちに移行します。

(c)　初期化処理の追加

プロジェクトの生成時に HEW が生成した初期化処理に、ハードウェアの初期化処理を追加します。

ハードウェアセットアップファイル hwsetup.c の関数にハードウェア初期化処理“RI_hwsetup()”を記述します。

サンプルの
ハードウェア
初期化処理

```
void HardwareSetup(void)
{
    RI_hwsetup();              /* for RI600/4 sample program */
      ・
      ・
}
```

図3－49　初期化処理の追加（SAMPLE6）

(2)　コンフィギュレーションファイルの作成

HEW の[ツール]から GUI コンフィギュレータを起動して、コンフィギュレーションファイルを作成します。

ここでは、タスクの情報、セマフォの情報、およびサンプルのタイマ用割込みハンドラを定義します。

サンプルのタイマ用割込みハンドラの定義については『2.1.3　コンフィギュレーションファイル（cfgファイル）の作成』をご参照ください。

(a)　タスクの生成に必要な情報を入力します。

サンプルプログラムで定義するタスクの情報を 表3－15に示します。

表3－15　GUIコンフィギュレータ　タスクの定義情報（SAMPLE6）

No.	項目	タスク 1	タスク 2	タスク 3
1	[ID 番号]	1	2	3
2	[ID 名称]	ID_TASK1	ID_TASK2	ID_TASK3
3	[アドレス]	main_task	task2	task3
4	[タスク起動時の優先度]	1	2	3
5	[属性]	[生成後、起動]を指定します。	指定しません。（デフォルト）	
6	[スタックサイズ]	0x00000100（デフォルト）		
7	[スタック領域配置]	[セクション名]に“SURI_STACK”を指定します。（デフォルト）		
8	[記述言語]	[高級言語]（デフォルト）		
9	[拡張情報]	0x00000000（デフォルト）		

［タスク］ウィンドウからダイアログを開いて、タスクの情報を定義します。

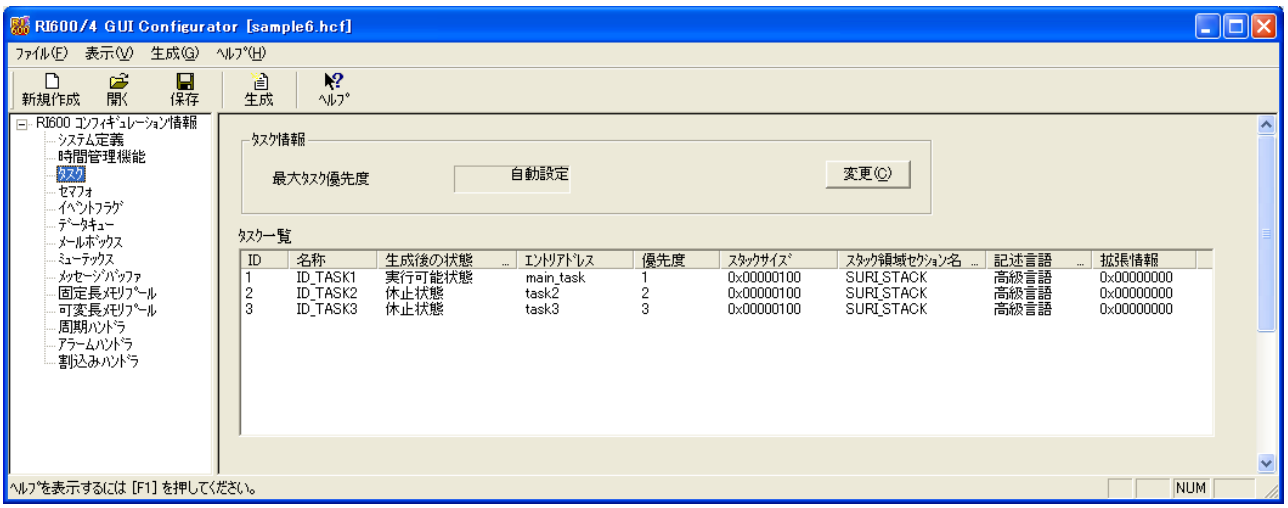


図3－50　GUIコンフィギュレータ　タスクの定義（SAMPLE6）

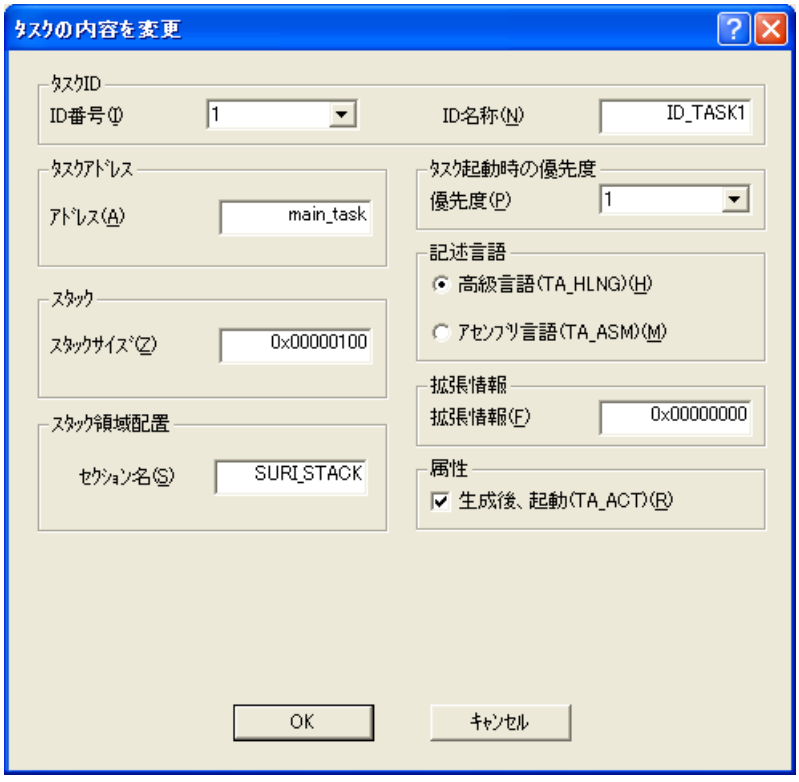


図3－51　GUIコンフィギュレータ　タスクの定義［ダイアログ］（SAMPLE6）

(b)　セマフォの生成に必要な情報を入力します。

サンプルプログラムで定義するセマフォの情報を 表3－16に示します。

表3－16　GUIコンフィギュレータ　セマフォの定義情報（SAMPLE6）

No.	項目	セマフォ 1
1	[ID 番号]	1
2	[ID 名称]	ID_SEM1
3	[セマフォ資源数]	[最大値]に 1 を指定します。（デフォルト）
4		[初期値]に 1 を指定します。（デフォルト）
5	[待ちタスクキューの並び方]	[FIFO 順(TA_TFIFO)]を指定します。（デフォルト）

［セマフォ］ウィンドウからダイアログを開いて、セマフォの情報を定義します。

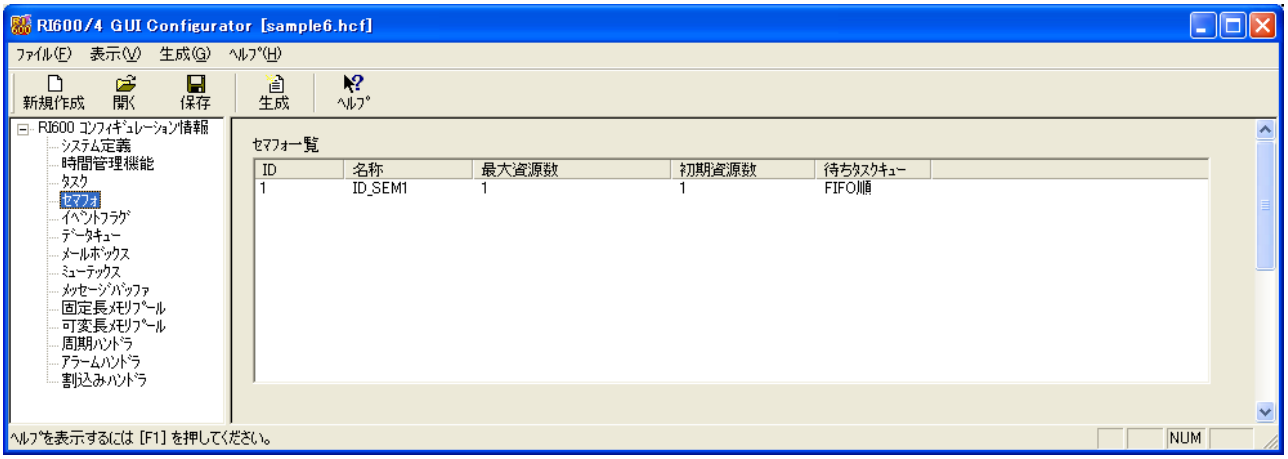


図3－52　GUIコンフィギュレータ　セマフォの定義（SAMPLE6）

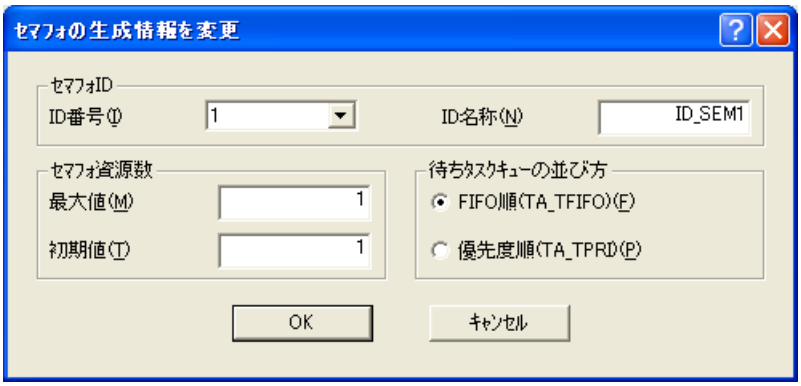


図3－53　GUIコンフィギュレータ　セマフォの定義［ダイアログ］（SAMPLE6）

(c)　コンフィギュレーションファイルを生成します。

[生成]でコンフギュレーションファイル（cfgファイル）を生成します。SAMPLE6では、sample6.cfgを生成します。

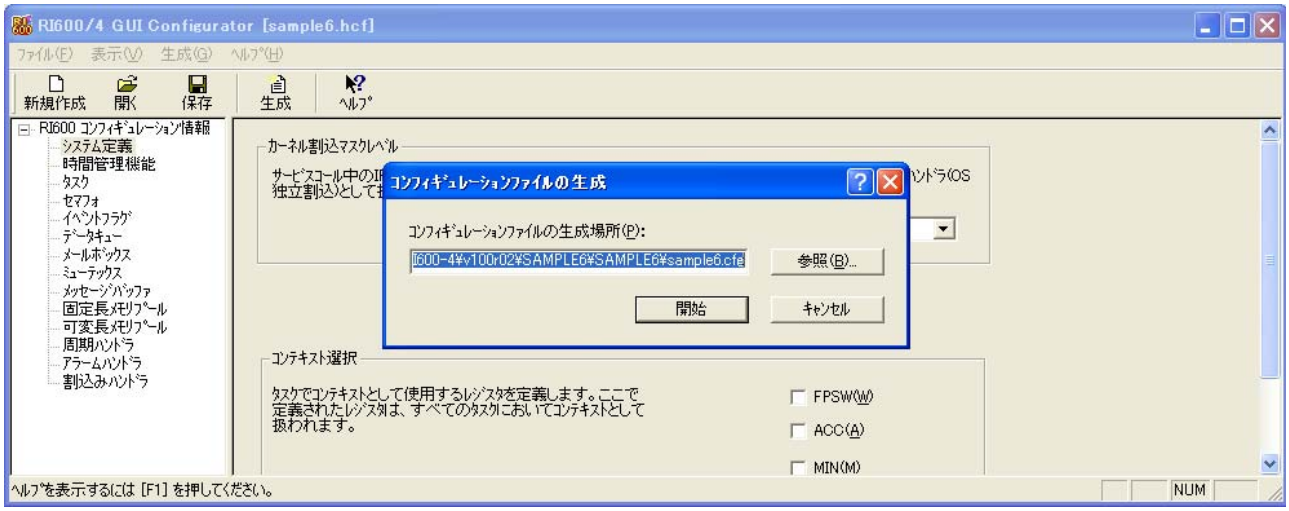


図3－54　コンフィギュレーションファイルの生成（SAMPLE6）

(3) ロードモジュールの作成

HEW のビルド機能を使って、ロードモジュールを作成します。
ロードモジュールの作成については『2.1.4　ロードモジュールの作成』をご参照ください。

(4) プログラムの実行

作成したプログラムを実際に実行します。
HEW のデバッグ機能を使って、デバッグの設定を行い、作成したロードモジュールをダウンロードします。
[実行]でプログラムが実行します。
プログラムの実行については『2.1.5　プログラムの実行』をご参照ください。

(5) プログラムの実行結果

task2 と task3 が、LCD に表示を行います。
CPU ボードの LCD に文字が表示されることを確認します。

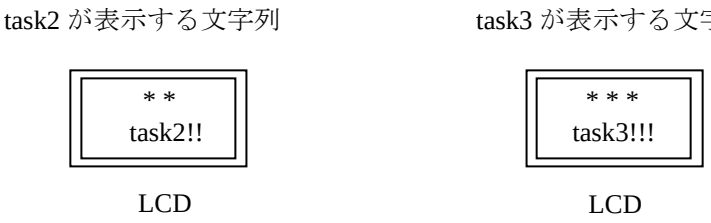
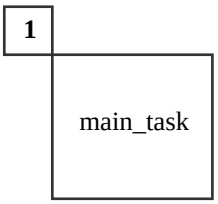


図3－55　プログラムの実行結果　LCDへの表示（SAMPLE6）

以上で、SAMPLE6のシステムは完了です。

(6) サンプルプログラム

SAMPLE6のタスクのソースコードは 図3－56のように記述しています。



カーネル提供ファイル

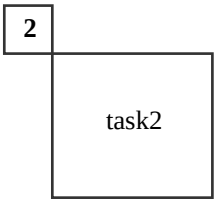
cfg600 生成ファイル

task2 起動（sta_tsk）

task3 起動（sta_tsk）

```
/*************************************************************************
*
*     RI600/4 Sample program
*
*************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

#include "RI_sample.h"                /* table for sample */

/*************************************************************************
  main_task
*************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      Number = 0L;

      ercd = sta_tsk(ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk(ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }
      Number++;

      ext_tsk();
}
```

図3－56　サンプルプログラムソースコード『SAMPLE6.c』（1/3）



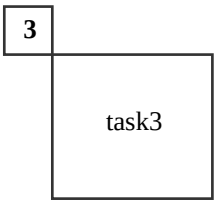
セマフォ獲得
（wai_sem）

LCD 表示処理

セマフォ返却
（sig_sem）

```
/*****************************************************************************
 task2
*****************************************************************************/
void task2(VP_INT exinf)
{
	ER	ercd, Number;

	Number = 0L;

	while(1){

		ercd = wai_sem(ID_SEM1);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_SEM1, (VW)Number, 0L, 0L);
		}
		Number++;

		DisplayString( 0,"    * *      ");
		DisplayString(16," task2!!");
		sample_delay(500);

		ercd = sig_sem(ID_SEM1);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_SEM1, (VW)Number, 0L, 0L);
		}
		Number++;

		ercd = tslp_tsk(10);
		if(ercd != E_TMOUT)
		{
			vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
		}
		Number++;

	}
	ext_tsk();
}
```

図3－56　サンプルプログラムソースコード『SAMPLE6.c』（2/3）




```
/*****************************************************************************
 task3
*****************************************************************************/
void task3(VP_INT exinf)
{
	ER	ercd, Number;

	Number = 0L;

	while(1){

		ercd = wai_sem(ID_SEM1);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_SEM1, (VW)Number, 0L, 0L);
		}
		Number++;

		DisplayString( 0," * * *    ");
		DisplayString(16,"task3!!!");
		sample_delay(500);

		ercd = sig_sem(ID_SEM1);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_SEM1, (VW)Number, 0L, 0L);
		}
		Number++;

		ercd = tslp_tsk(10);
		if(ercd != E_TMOUT)
		{
			vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
		}
		Number++;

	}
	ext_tsk();
}
```

セマフォ獲得（wai_sem）

LCD 表示処理

セマフォ返却（sig_sem）

図3－56　サンプルプログラムソースコード『SAMPLE6.c』（3/3）

3.1.7 SAMPLE7：データキュー

SAMPLE7では、データキューを使って、タスク間で送受信したデータをLCDに表示します。

データキューは、1 ワード（32 ビット）のデータ通信を行うオブジェクトです。

(1) アプリケーションプログラム

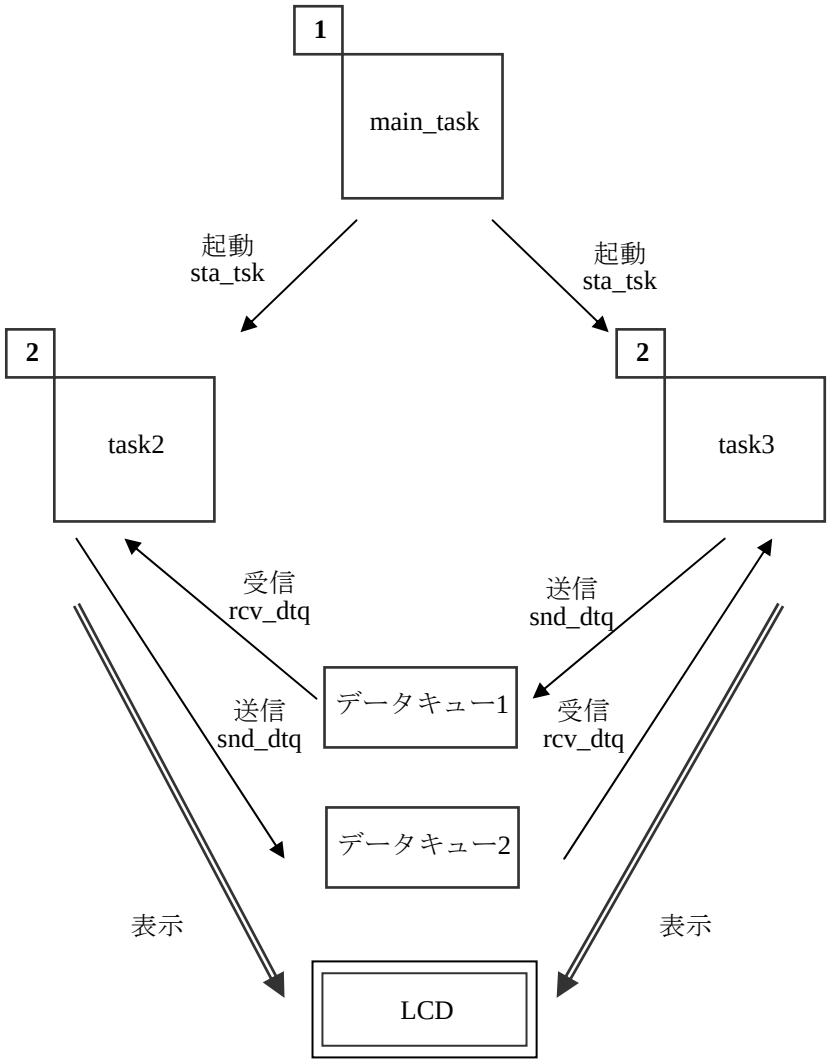
SAMPLE7は、main_task、task2、task3 の 3 つのタスクからなります。
main_task から起動された task2 と task3 は、それぞれに受信したデータを LCD に対して表示します。

(a) 使用するオブジェクト

表3－17　使用するオブジェクト（SAMPLE7）

オブジェクト	アドレス	ID 番号	優先度	動作
タスク	main_task	1	1	初期起動 task2 の起動、task3 の起動
	task2	2	2	データキューの送信と受信 LCD への表示
	task3	3	2	データキューの送信と受信 LCD への表示
データキュー	－	1	－	データ最大数：1
	－	2	－	データ最大数：1

(b)　アプリケーションの動作



【注】sta_tsk はタスクを起動するサービスコールです。
　　snd_dtq はデータキューへデータを送信するサービスコールです。
　　rcv_dtq はデータキューからデータを受信するサービスコールです。

図3－57　アプリケーションの動作（SAMPLE7）

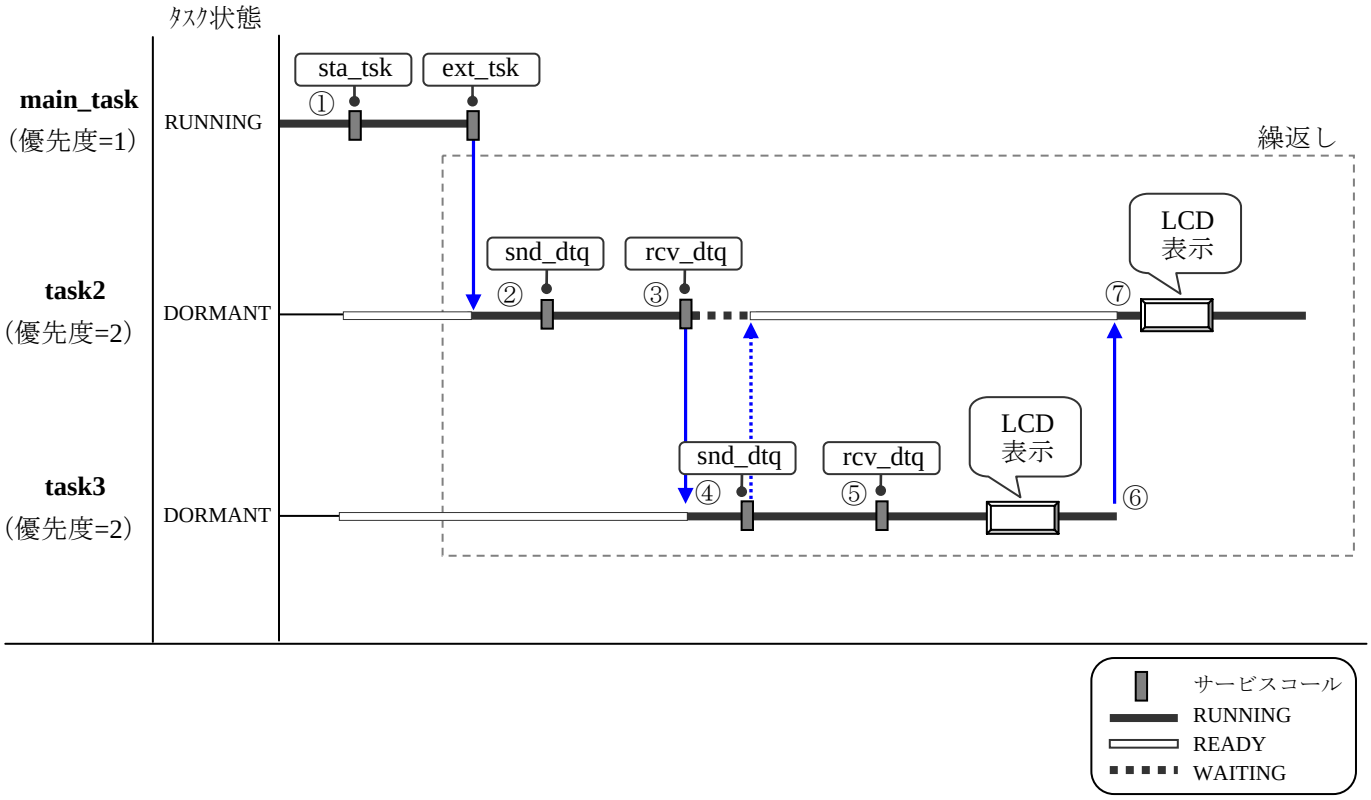


図3－58　アプリケーションのシーケンス（SAMPLE7）

① sta_tsk サービスコールで task2 と task3 を起動し、ext_tsk サービスコールで終了します。

② task2 が RUNNING 状態になり、snd_dtq サービスコール（ID=2）でデータを送信します。

③ rcv_dtq サービスコール（ID=1）でデータを受信して、待ちに移行します。

④ 次に task3 が RUNNING 状態になり、snd_dtq サービスコール（ID=1）でデータを送信します。

⑤ rcv_dtq サービスコール（ID=2）でデータを受信します。このとき、データがあるので待ちに移行しません。受信したデータを LCD の 2 列目に表示します。

⑥ task3 は、再び snd_dtq サービスコール（ID=1）でデータを送信して、rcv_dtq サービスコール（ID=2）でデータの受信待ちに移行します。

⑦ task2 は受信待ちが解除され、受信したデータを LCD の 1 列目に表示します。

(c)　初期化処理の追加

プロジェクトの生成時に HEW が生成した初期化処理に、ハードウェアの初期化処理を追加します。

ハードウェアセットアップファイル hwsetup.c の関数にハードウェア初期化処理“RI_hwsetup()”を記述します。

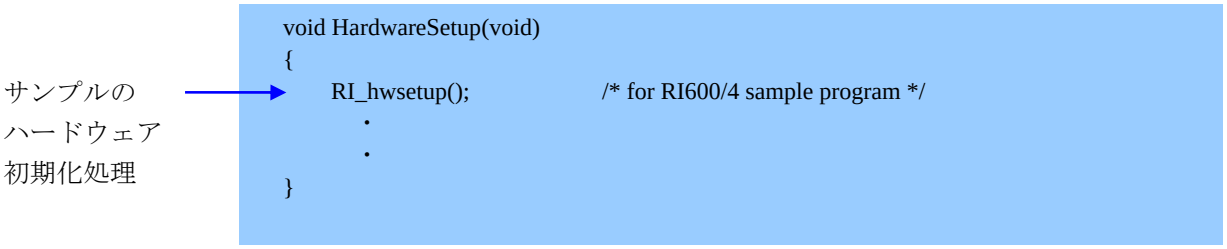


図3－59　初期化処理の追加（SAMPLE7）

(2)　コンフィギュレーションファイルの作成

HEW の[ツール]から GUI コンフィギュレータを起動して、コンフィギュレーションファイルを作成します。

ここでは、タスクの情報、データキューの情報、およびサンプルのタイマ用割込みハンドラを定義します。

サンプルのタイマ用割込みハンドラの定義については『2.1.3　コンフィギュレーションファイル（cfgファイル）の作成』をご参照ください。

(a)　タスクの生成に必要な情報を入力します。

サンプルプログラムで定義するタスクの情報を 表3－18に示します。

表3－18　GUIコンフィギュレータ　タスクの定義情報（SAMPLE7）

No.	項目	タスク 1	タスク 2	タスク 3
1	[ID 番号]	1	2	3
2	[ID 名称]	ID_TASK1	ID_TASK2	ID_TASK3
3	[アドレス]	main_task	task2	task3
4	[タスク起動時の優先度]	1	2	2
5	[属性]	[生成後、起動]を指定します。	指定しません。（デフォルト）	
6	[スタックサイズ]	0x00000100（デフォルト）		
7	[スタック領域配置]	[セクション名]に“SURI_STACK”を指定します。（デフォルト）		
8	[記述言語]	[高級言語]（デフォルト）		
9	[拡張情報]	0x00000000（デフォルト）		

［タスク］ウィンドウからダイアログを開いて、タスクの情報を定義します。

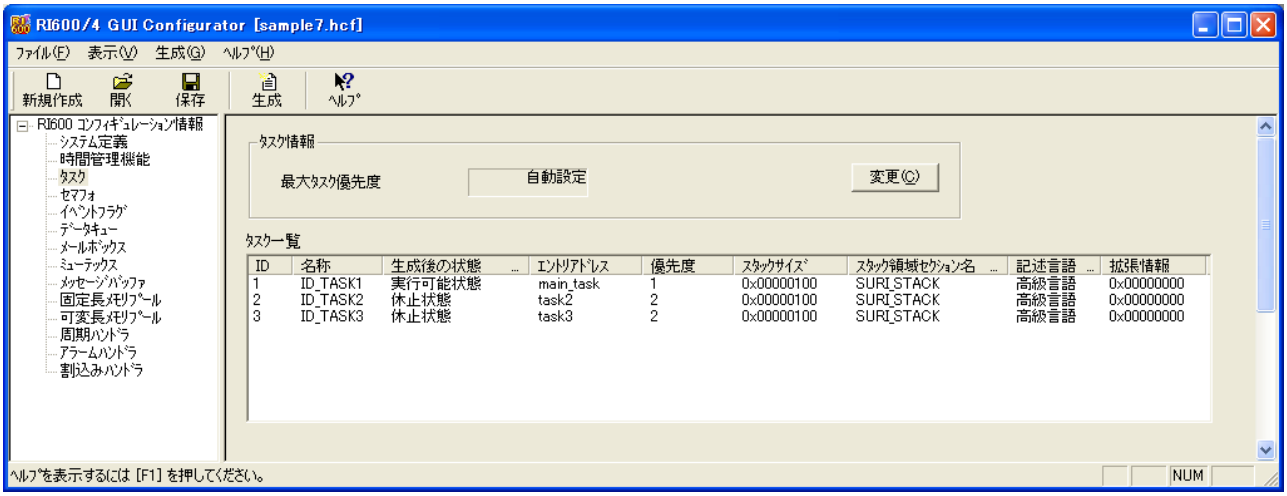


図3－60　GUIコンフィギュレータ　タスクの定義（SAMPLE7）

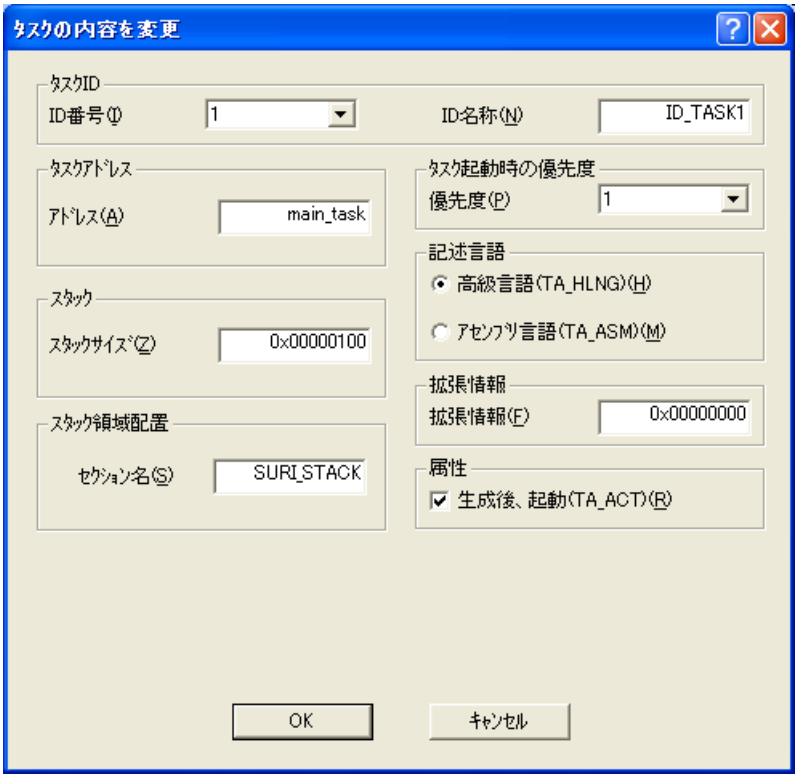


図3－61　GUIコンフィギュレータ　タスクの定義［ダイアログ］（SAMPLE7）

(b)　データキューの生成に必要な情報を入力します。

サンプルプログラムで定義するデータキューの情報を 表3－19に示します。

表3－19　GUIコンフィギュレータ　データキューの定義情報（SAMPLE7）

No.	項目	データキュー1	データキュー2
1	[ID 番号]	1	2
2	[ID 名称]	ID_DTQ1	ID_DTQ2
3	[データキュー]	[データ数]に 1 を指定します。	
4	[待ちタスクキューの並び方]	[FIFO 順(TA_TFIFO)]を指定します。（デフォルト）	

［データキュー］ウィンドウからダイアログを開いて、データキューの情報を定義します。

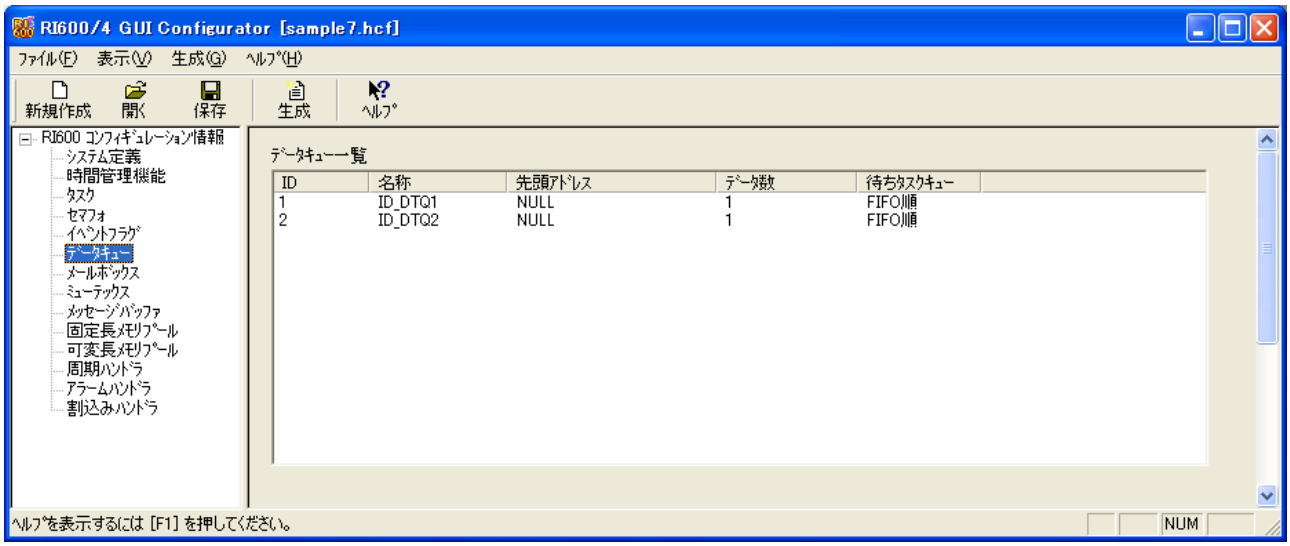


図3－62　GUIコンフィギュレータ　データキューの定義（SAMPLE7）

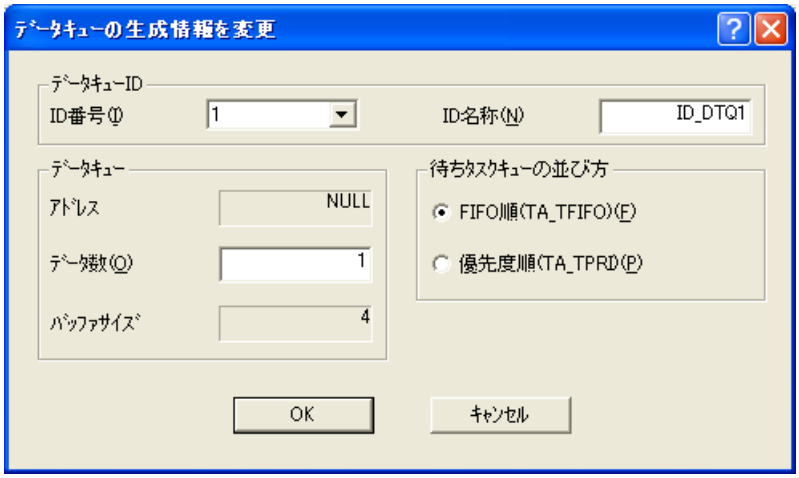


図3－63　GUIコンフィギュレータ　データキューの定義［ダイアログ］（SAMPLE7）

(c)　コンフィギュレーションファイルを生成します。

[生成]でコンフギュレーションファイル（cfgファイル）を生成します。SAMPLE7では、sample7.cfgを生成します。

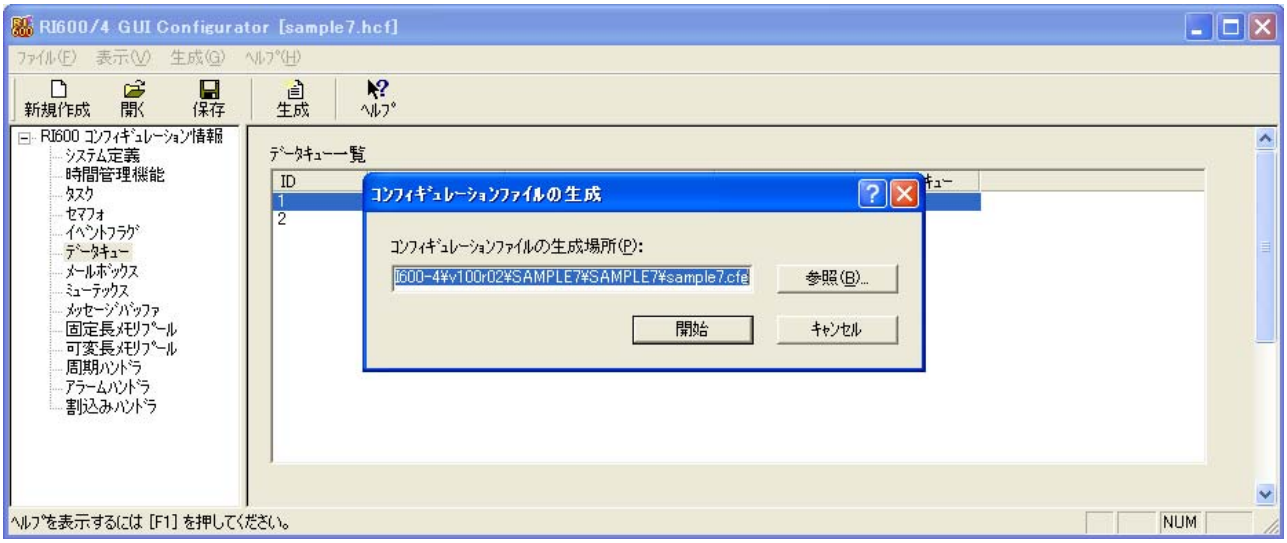


図3－64　コンフィギュレーションファイルの生成（SAMPLE7）

(3) ロードモジュールの作成

HEW のビルド機能を使って、ロードモジュールを作成します。
ロードモジュールの作成については『2.1.4　ロードモジュールの作成』をご参照ください。

(4) プログラムの実行

作成したプログラムを実際に実行します。
HEW のデバッグ機能を使って、デバッグの設定を行い、作成したロードモジュールをダウンロードします。
[実行]でプログラムが実行します。
プログラムの実行については『2.1.5　プログラムの実行』をご参照ください。

(5) プログラムの実行結果

task2 と task3 が、それぞれ受信したデータを LCD に表示します。
CPU ボードの LCD に文字が表示されることを確認します。

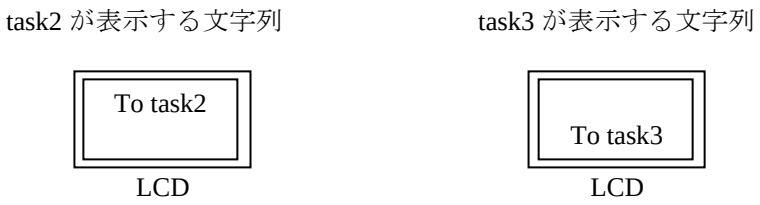
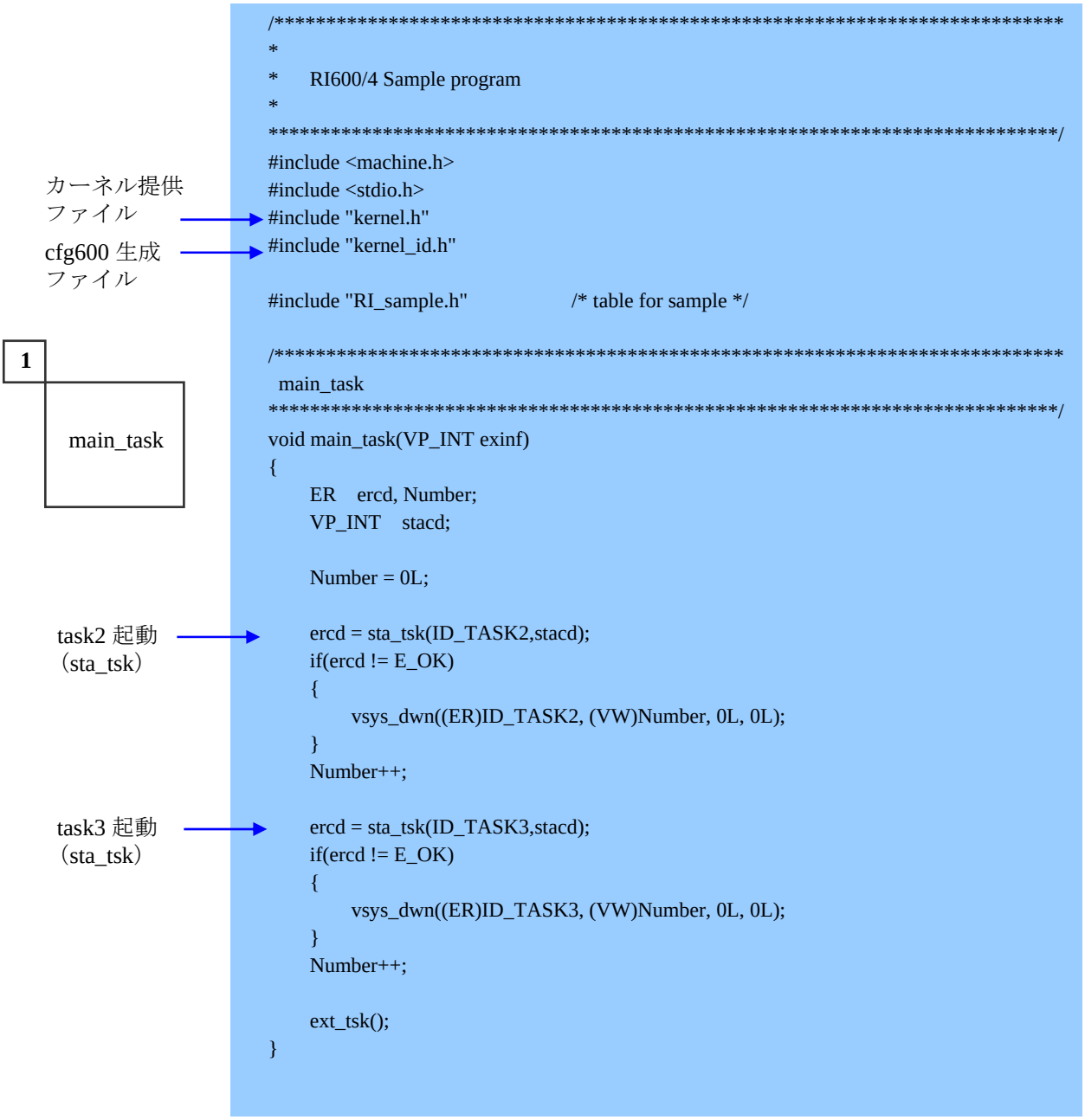


図3－65　プログラムの実行結果　LCDへの表示（SAMPLE7）

以上で、SAMPLE7のシステムは完了です。

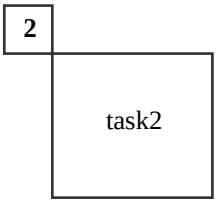
(6) サンプルプログラム

SAMPLE7のタスクのソースコードは 図3－66のように記述しています。




```
/************************************************************************
*
*     RI600/4 Sample program
*
************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

#include "RI_sample.h"                /* table for sample */

/************************************************************************
  main_task
************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      Number = 0L;

      ercd = sta_tsk(ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk(ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }
      Number++;

      ext_tsk();
}
```

図3－66　サンプルプログラムソースコード『SAMPLE7.c』（1/3）



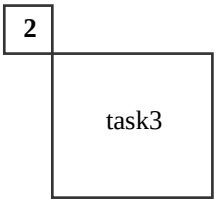
データキュー送信
（snd_dtq）

データキュー受信
（rcv_dtq）

LCD 表示処理

```
/*****************************************************************************
 task2
*****************************************************************************/
void task2(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT data1;

      Number = 0L;

      while(1){

            data1= (VP_INT)"To task3";
            ercd = snd_dtq(ID_DTQ2,data1);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_DTQ2, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = rcv_dtq(ID_DTQ1,&data1);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_DTQ1, (VW)Number, 0L, 0L);
            }
            Number++;

            DisplayString( 0,data1);
            sample_delay(1000);
            DisplayString( 0,LCD_CLR);

      }
      ext_tsk();

}
```

図3－66　サンプルプログラムソースコード『SAMPLE7.c』（2/3）



データキュー
送信（snd_dtq）

データキュー
受信（rcv_dtq）

LCD 表示処理

```
/*****************************************************************************
 task3
*****************************************************************************/
void task3(VP_INT exinf)
{
	ER    ercd, Number;
	VP_INT data1;

	Number = 0L;

	while(1){

		data1= (VP_INT)"To task2";
		ercd = snd_dtq(ID_DTQ1,data1);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_DTQ1, (VW)Number, 0L, 0L);
		}
		Number++;

		ercd = rcv_dtq(ID_DTQ2,&data1);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_DTQ2, (VW)Number, 0L, 0L);
		}
		Number++;

		DisplayString(16,data1);
		sample_delay(1000);
		DisplayString(16,LCD_CLR);

	}
	ext_tsk();
}

/*****************************************************************************
*****************************************************************************/
```

図3－66　サンプルプログラムソースコード『SAMPLE7.c』（3/3）

3.1.8 SAMPLE8：メールボックスと固定長メモリプール

SAMPLE8では、メールボックスを使って、タスク間で送受信したメッセージをLCDに表示します。

メールボックスは、メッセージと呼ぶ任意のサイズのデータ通信を行うオブジェクトです。

メールボックスはメッセージアドレスの受け渡しによってデータ通信を行うため、メッセージサイズに依存しない高速な通信が行われます。

固定長メモリプールは、決められたサイズのメモリブロックを動的に獲得・返却するオブジェクトです。

メッセージの領域をメモリプールに割り当てることにより、データの不正な上書きなどを防ぐことができます。

(1) アプリケーションプログラム

SAMPLE8は、main_task、task2、task3 の 3 つのタスクからなります。

main_task から起動された task2 と task3 は、それぞれに受信したメッセージを LCD に対して表示します。

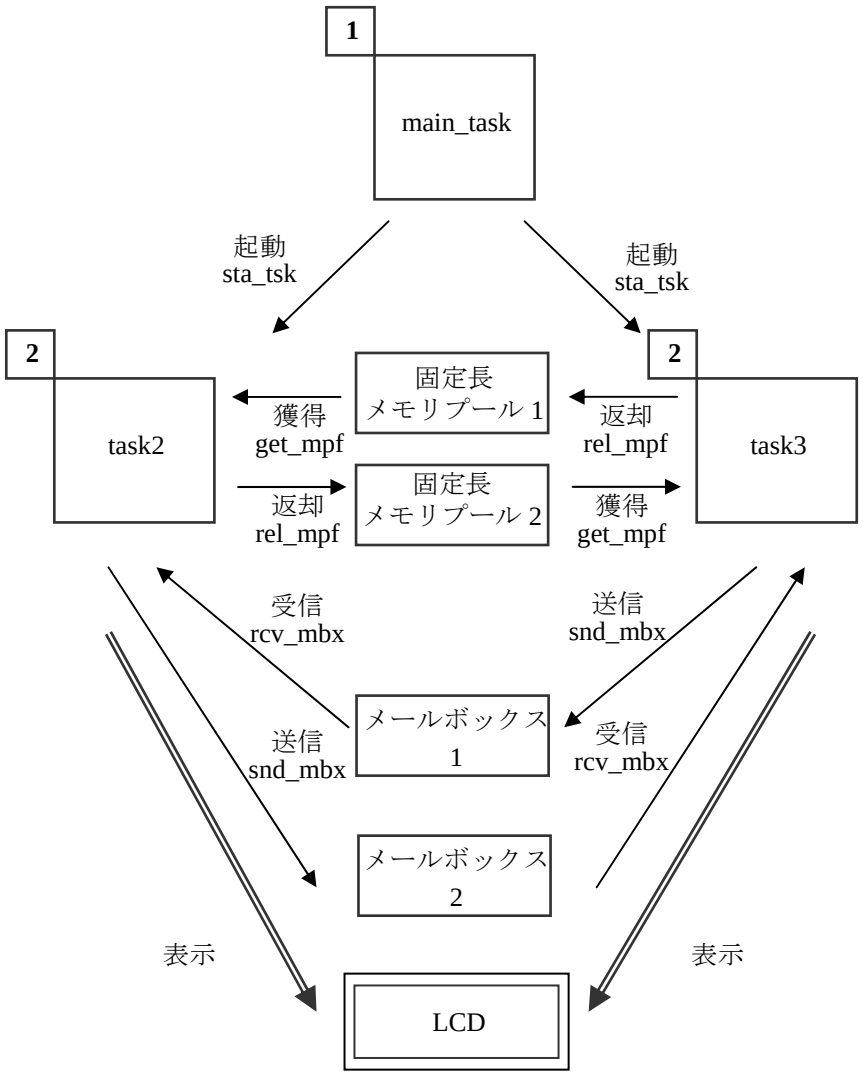
メッセージの領域は固定長メモリプールを使用します。

(a) 使用するオブジェクト

表3－20　使用するオブジェクト（SAMPLE8）

オブジェクト	アドレス	ID 番号	優先度	動作
タスク	main_task	1	1	初期起動 task2 の起動、task3 の起動
	task2	2	2	メモリブロックの獲得と返却 メッセージの送信と受信 LCD への表示
	task3	3	2	メモリブロックの獲得と返却 メッセージの送信と受信 LCD への表示
メールボックス	－	1	－	メッセージキューの並び方：FIFO 順
	－	2	－	
固定長メモリプール	－	1	－	メモリブロックサイズ：16 バイト メモリブロック数：1 つ
	－	2	－	メモリブロックサイズ：16 バイト メモリブロック数：1 つ

(b)　アプリケーションの動作



【注】sta_tsk はタスクを起動するサービスコールです。
snd_mbx はメールボックスへメッセージを送信するサービスコールです。
rcv_mbx はメールボックスからメッセージを受信するサービスコールです。
get_mpf は固定長メモリプールからメモリブロックを獲得するサービスコールです。
rel_mpf は固定長メモリプールへメモリブロックを返却するサービスコールです。

図3－67　アプリケーションの動作（SAMPLE8）

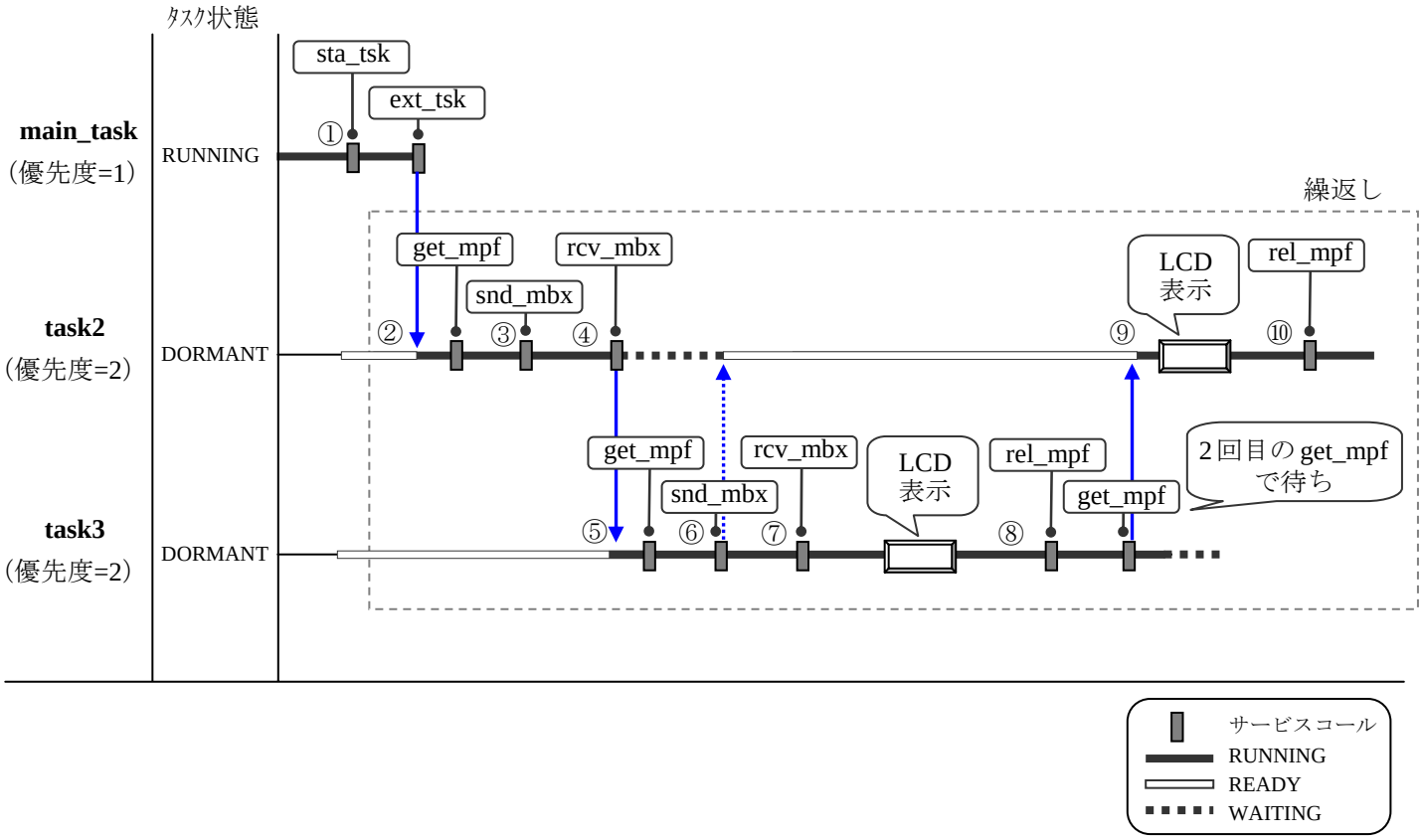


図3－68　アプリケーションのシーケンス（SAMPLE8）

① sta_tsk サービスコールで task2 と task3 を起動し、ext_tsk サービスコールで終了します。

② task2 が RUNNING 状態になり get_mpf サービスコール(ID=1)でメモリブロックを獲得します。

③ メールボックス 2 に snd_mbx サービスコールでメッセージを送信します。

④ メールボックス 1 に rcv_mbx サービスコールでメッセージの受信待ちに移行します。

⑤ 次に task3 が RUNNING 状態になり get_mpf サービスコール（ID=2）でメモリブロックを獲得します。

⑥ メールボックス 1 に snd_mbx サービスコールでメッセージを送信します。

⑦ メールボックス 2 に rcv_mbx サービスコールでメッセージを受信します。このときメッセージがあるので、待ちに移行しません。受信したメッセージを LCD に表示します。

⑧ task3 は受け取ったメッセージの領域に使用していたメモリブロックを rel_mpf サービスコール（ID=1）で返却します。get_mpf（ID=2）サービスコールを呼び出し、メモリブロックの獲得待ちに移行します。

⑨ task2 が RUNNING 状態になり、LCD に表示します。

⑩ task2 は受け取ったメッセージの領域に使用していたメモリブロックを rel_mpf サービスコール（ID=2）で返却します。

(c)　初期化処理の追加

プロジェクトの生成時に HEW が生成した初期化処理に、ハードウェアの初期化処理を追加します。

ハードウェアセットアップファイル hwsetup.c の関数にハードウェア初期化処理“RI_hwsetup()”を記述します。

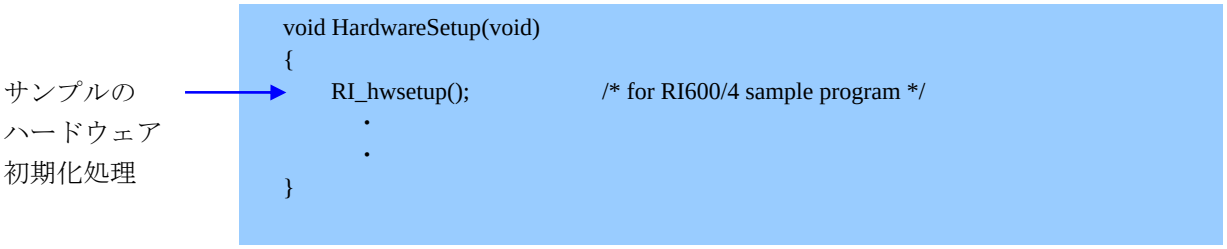


図3－69　初期化処理の追加（SAMPLE8）

(2)　コンフィギュレーションファイルの作成

HEW の[ツール]から GUI コンフィギュレータを起動して、コンフィギュレーションファイルを作成します。

ここでは、タスクの情報、メールボックスの情報、固定長メモリプールの情報、およびサンプルのタイマ用割込みハンドラを定義します。

サンプルのタイマ用割込みハンドラの定義については『2.1.3　コンフィギュレーションファイル（cfgファイル）の作成』をご参照ください。

(a)　タスクの生成に必要な情報を入力します。

サンプルプログラムで定義するタスクの情報を 表3－21に示します。

表3－21　GUIコンフィギュレータ　タスクの定義情報（SAMPLE8）

No.	項目	タスク 1	タスク 2	タスク 3
1	[ID 番号]	1	2	3
2	[ID 名称]	ID_TASK1	ID_TASK2	ID_TASK3
3	[アドレス]	main_task	task2	task3
4	[タスク起動時の優先度]	1	2	2
5	[属性]	[生成後、起動]を指定します。	指定しません。（デフォルト）	
6	[スタックサイズ]	0x00000100（デフォルト）		
7	[スタック領域配置]	[セクション名]に“SURI_STACK”を指定します。（デフォルト）		
8	[記述言語]	[高級言語]（デフォルト）		
9	[拡張情報]	0x00000000（デフォルト）		

［タスク］ウィンドウからダイアログを開いて、タスクの情報を定義します。

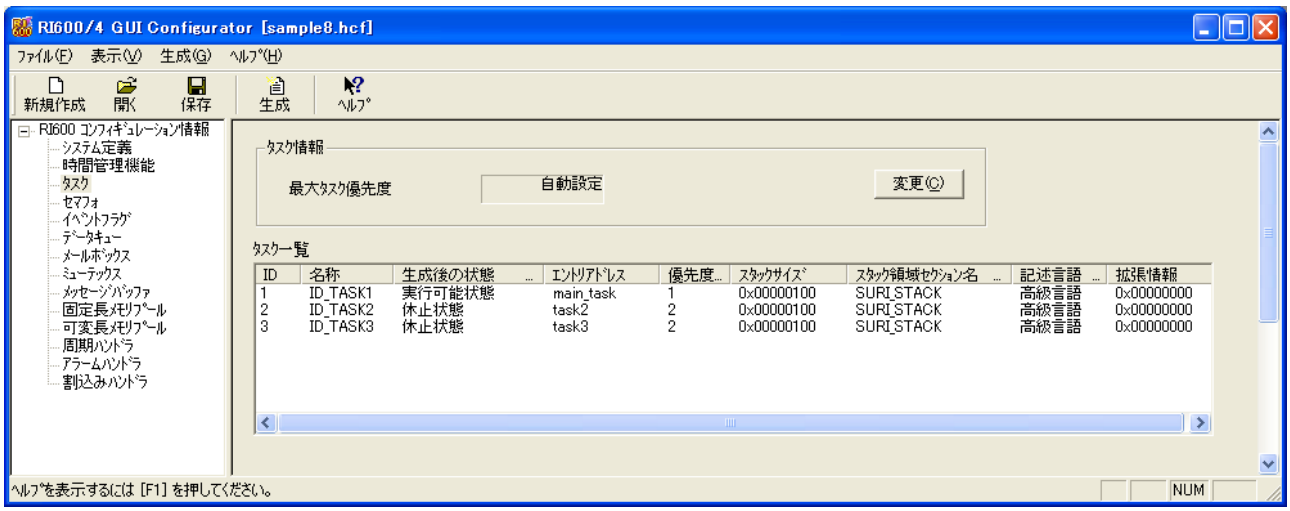


図3－70　GUIコンフィギュレータ　タスクの定義（SAMPLE8）

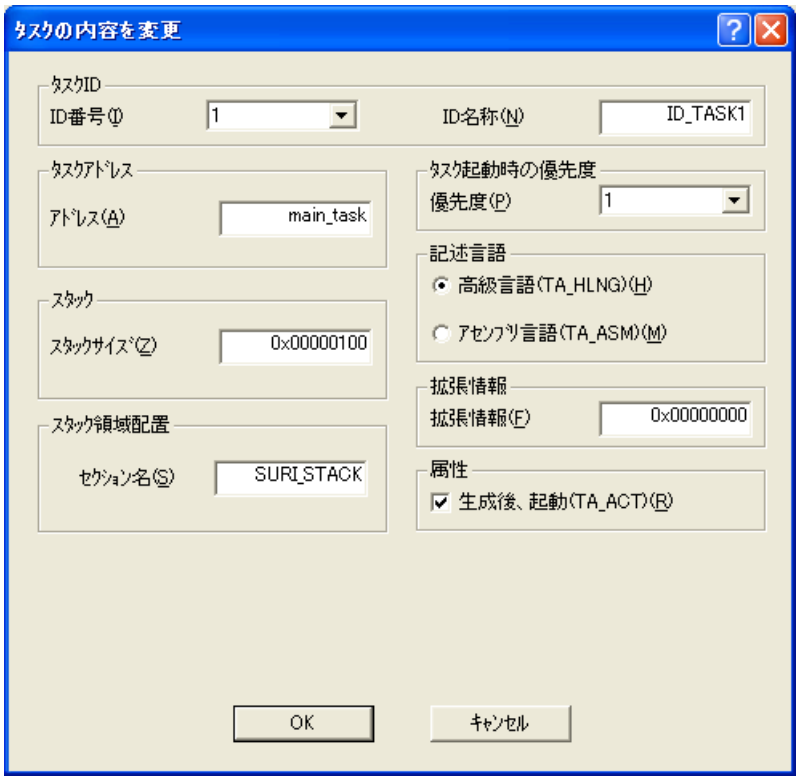


図3－71　GUIコンフィギュレータ　タスクの定義［ダイアログ］（SAMPLE8）

(b)　メールボックスの生成に必要な情報を入力します。

サンプルプログラムで定義するメールボックスの情報を 表3－22に示します。

表3－22　GUIコンフィギュレータ　メールボックスの定義情報（SAMPLE8）

No.	項目	メールボックス 1	メールボックス 2
1	[ID 番号]	1	2
2	[ID 名称]	ID_MBX1	ID_MBX2
3	[メッセージ]	[最大優先度]を 1 に指定します。（デフォルト）	
4	[待ちタスクキューの並び方]	[FIFO 順(TA_TFIFO)]を指定します。（デフォルト）	
5	[メッセージキューの並び方]	[FIFO 順(TA_MTFIFO)]を指定します。（デフォルト）	

［メールボックス］ウィンドウからダイアログを開いて、メールボックスの情報を定義します。

図3－72　GUIコンフィギュレータ　メールボックスの定義（SAMPLE8）

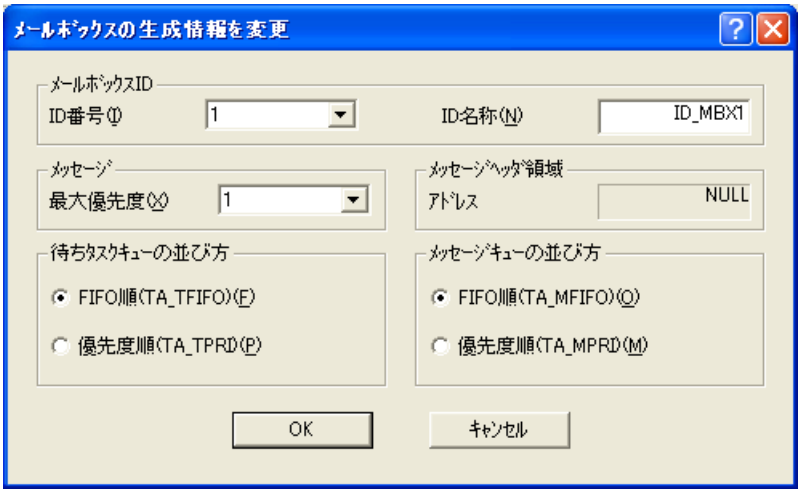


図3－73　GUIコンフィギュレータ　メールボックスの定義［ダイアログ］（SAMPLE8）

(c)　固定長メモリプールの生成に必要な情報を入力します。

サンプルプログラムで定義する固定長メモリプールの情報を 表3－23に示します。

表3－23　GUIコンフィギュレータ　固定長メモリプールの定義情報（SAMPLE8）

No.	項目	固定長メモリプール 1	固定長メモリプール 2
1	[ID 番号]	1	2
2	[ID 名称]	ID_MPF1	ID_MPF2
3	[メモリブロック]	[獲得可能数]を 1 に指定します。（デフォルト）	
4		[サイズ]を 16 に指定します。	
5	[メモリプール領域配置]	[セクション名]を “BRI_HEAP” に指定します。（デフォルト）	
6	[待ちタスクキューの並び方]	[FIFO 順(TA_TFIFO)]を指定します。（デフォルト）	

［固定長メモリプール］ウィンドウからダイアログを開いて、固定長メモリプールの情報を定義します。

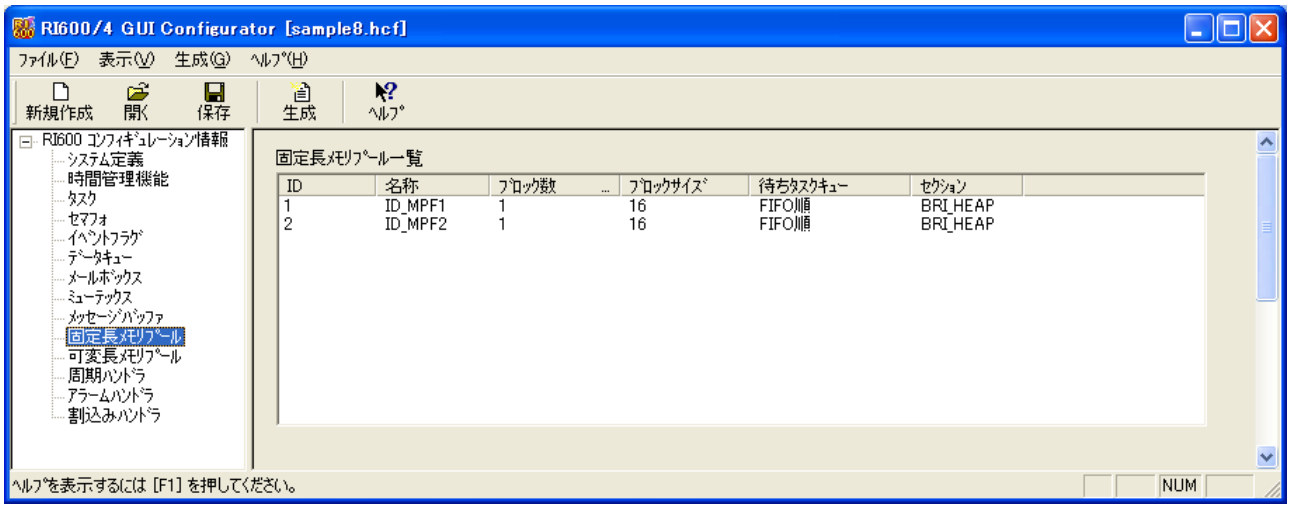


図3－74　GUIコンフィギュレータ　固定長メモリプールの定義（SAMPLE8）

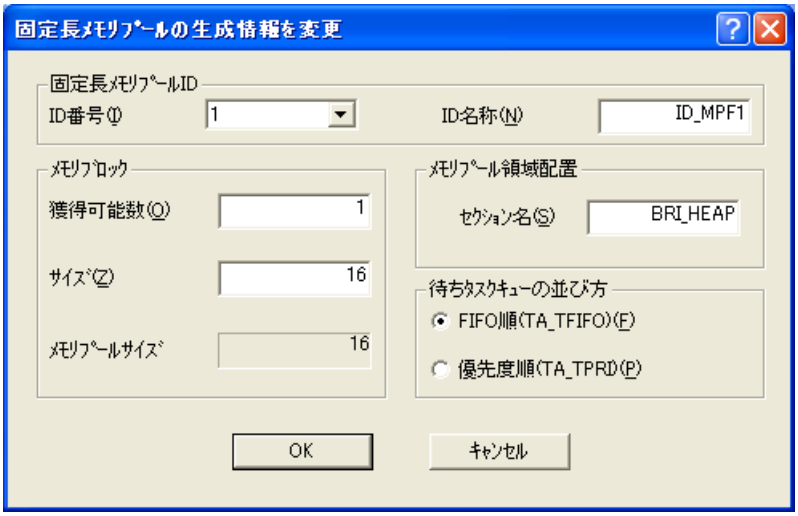


図3－75　GUIコンフィギュレータ　固定長メモリプールの定義［ダイアログ］（SAMPLE8）

(d)　コンフィギュレーションファイルを生成します。

[生成]でコンフギュレーションファイル（cfgファイル）を生成します。SAMPLE8では、sample8.cfgを生成します。

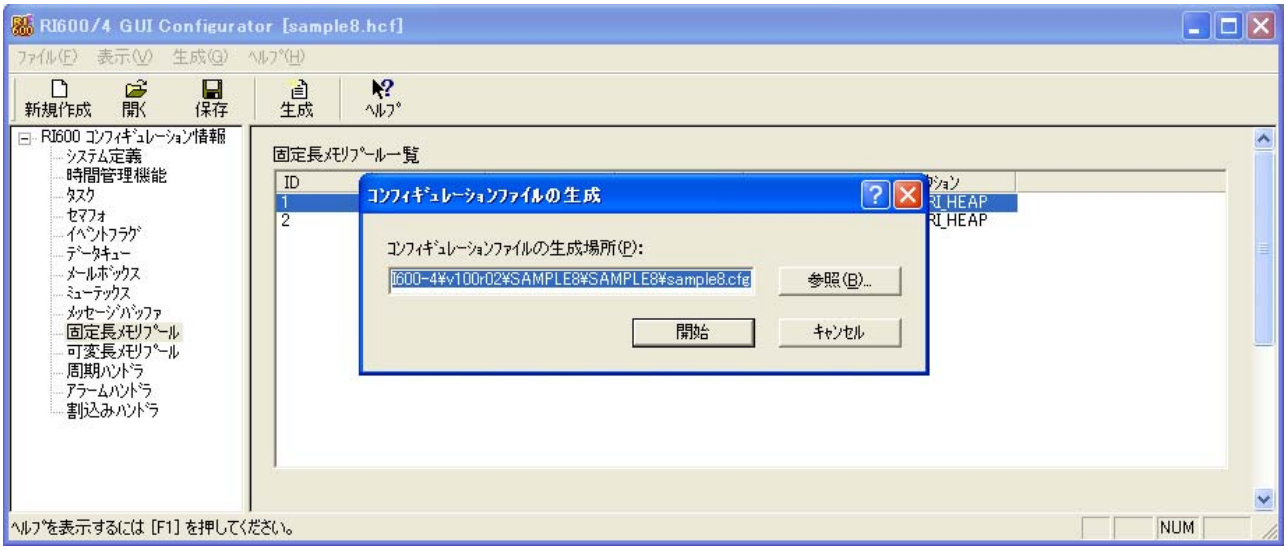


図3－76　コンフィギュレーションファイルの生成（SAMPLE8）

(3) ロードモジュールの作成

HEW のビルド機能を使って、ロードモジュールを作成します。
ロードモジュールの作成については『2.1.4　ロードモジュールの作成』をご参照ください。

(4) プログラムの実行

作成したプログラムを実際に実行します。
HEW のデバッグ機能を使って、デバッグの設定を行い、作成したロードモジュールをダウンロードします。
[実行]でプログラムが実行します。
プログラムの実行については『2.1.5　プログラムの実行』をご参照ください。

(5) プログラムの実行結果

task2 と task3 が、それぞれ受信したメッセージを LCD に表示します。
CPU ボードの LCD に文字が表示されることを確認します。

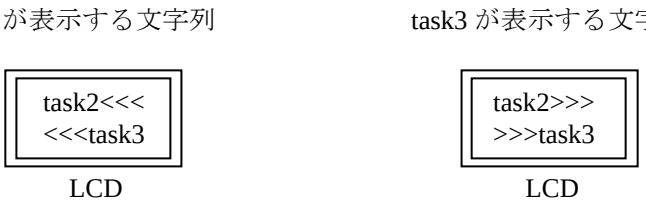


図3－77　プログラムの実行結果　LCDへの表示（SAMPLE8）

以上で、SAMPLE8のシステムは完了です。

(6) サンプルプログラム

SAMPLE8のタスクのソースコードは 図3－78のように記述しています。

カーネル提供ファイル →
cfg600 生成ファイル →

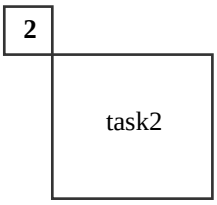
```
/*****************************************************************************
*
*     RI600/4 Sample program
*
*****************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

#include "RI_sample.h"              /* table for sample */

/*****************************************************************************
 Shared data
*****************************************************************************/
typedef struct {
      T_MSG     msg;
      VP    msg_data;
} UserMSG;

typedef struct {
      VP save_blk1;
      VP save_blk2;
} t_blk;
t_blk inf_blk;
/*****************************************************************************
 main_task
*****************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      Number = 0L;

      ercd = sta_tsk(ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk(ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }
      Number++;

      ext_tsk();
}
```

1 main_task

task2 起動（sta_tsk） →

task3 起動（sta_tsk） →

図3－78　サンプルプログラムソースコード『SAMPLE8.c』（1/3）




```
#define msg_task2    "task2>>>>>>task3 "
#define msg_task3    "task2<<<<<<task3"
/*****************************************************************************
 task2
*****************************************************************************/
void task2(VP_INT exinf)
{
      ER    ercd, Number;
      VP    blk;

      UserMSG *msgp;
      UserMSG g_usermsgp;
      Number = 0L;

      while(1){

            ercd = get_mpf(ID_MPF1,&blk);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_MPF1, (VW)Number, 0L, 0L);
            }
            Number++;

            inf_blk.save_blk1= blk;
            cpy_str(blk,msg_task2,(UINT)16);
            g_usermsgp.msg_data = blk;
            g_usermsgp.msg.msghead = 0;
            ercd = snd_mbx(ID_MBX2,(T_MSG *)&g_usermsgp);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_MBX2, (VW)Number, 0L, 0L);
            }
            Number++;

            ercd = rcv_mbx(ID_MBX1,(T_MSG **)&msgp);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_MBX1, (VW)Number, 0L, 0L);
            }
            Number++;

            DisplayString( 0,(*msgp).msg_data);
            DisplayString(16,((UW)((*msgp).msg_data)+8));
            sample_delay(1000);

            blk = inf_blk.save_blk2;
            ercd = rel_mpf(ID_MPF2,blk);
            if(ercd != E_OK)
            {
                  vsys_dwn((ER)ID_MPF2, (VW)Number, 0L, 0L);
            }
            Number++;
      }
      ext_tsk();
}
```

メモリブロック獲得（get_mpf）

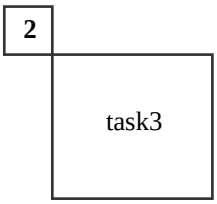
メッセージ送信（snd_mbx）

メッセージ受信（rcv_mbx）

LCD 表示処理

メモリブロック返却（rel_mpf）

図3－78　サンプルプログラムソースコード『SAMPLE8.c』（2/3）




```
/**********************************************************************************
 task3
**********************************************************************************/
void task3(VP_INT exinf)
{
    ER    ercd, Number;
    VP blk;

    UserMSG *msgp;
    UserMSG g_usermsgp;
    Number = 0L;

    while(1){

        ercd = get_mpf(ID_MPF2,&blk);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_MPF2, (VW)Number, 0L, 0L);
        }
        Number++;

        inf_blk.save_blk2 = blk;
        cpy_str(blk,msg_task3,(UINT)16);
        g_usermsgp.msg_data = blk;
        g_usermsgp.msg.msghead = 0;
        ercd = snd_mbx(ID_MBX1,(T_MSG *)&g_usermsgp);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_MBX1, (VW)Number, 0L, 0L);
        }
        Number++;

        ercd = rcv_mbx(ID_MBX2,(T_MSG **)&msgp);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_MBX2, (VW)Number, 0L, 0L);
        }
        Number++;

        DisplayString( 0,(*msgp).msg_data);
        DisplayString(16,((UW)((*msgp).msg_data)+8));
        sample_delay(1000);

        blk = inf_blk.save_blk1;
        ercd = rel_mpf(ID_MPF1,blk);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_MPF1, (VW)Number, 0L, 0L);
        }
        Number++;
    }
    ext_tsk();
}

/**********************************************************************************
**********************************************************************************/
```

メモリブロック獲得（get_mpf）

メッセージ送信（snd_mbx）

メッセージ受信（rcv_mbx）

LCD 表示処理

メモリブロック返却（rel_mpf）

図3－78　サンプルプログラムソースコード『SAMPLE8.c』（3/3）

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

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