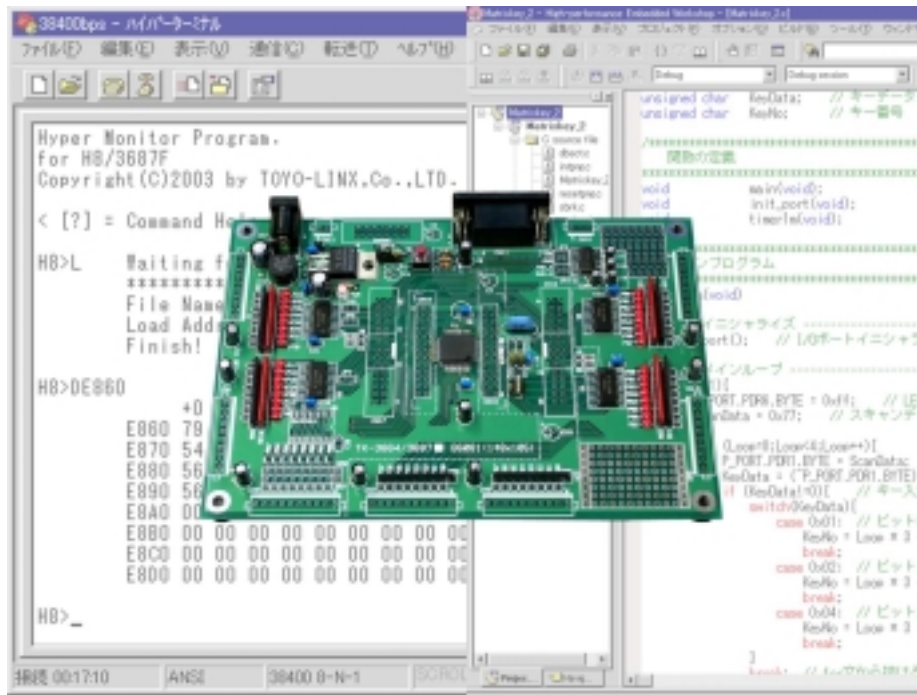


TK-3687

ユーザズマニュアル

C 言語版

(Ver.3.00)



目次

1. はじめに, , ,	1
2. 組み立て(キットでお買い求めのユーザへ)	2
3. 簡易モニタ“ハイパーH8”と動作チェック	5
4. メモリマップ	9
5. 無償評価版コンパイラ“HEW”のインストール	11
6. プログラムの作成	13
7. “ハイパーH8”を使ったダウンロードと実行方法	28
8. エミュレータ“E8”を使ったダウンロードと実行方法	31
9. 実習プログラム	38
10. 応用プログラム	44
11. 付録 ユーザプログラムとハイパーH8 を書き込む	74
回路図 部品表	83
C のプログラムからアセンブラのサブルーチンを呼び出す方法	85
旧マニュアル(Ver.2.xx)のコピー(etc)	119

1 はじめに , , ,

TK-3687 はこれからマイコンを修得しようとする方々が手軽に使えて、マイコンのハードソフトに親しみながら学習できるように作られたトレーニングボードです。H8/300H Tiny シリーズのうち標準的かつ多機能な H8/3687 (H8/3664 の上位コンパチブル)を採用し、扱い易さと機能の面で 16 ビットワンチップマイコンの主流となった H8 シリーズのベース基板としてご期待に沿えるハードおよびソフト構成になっております。

H8/3687 に用意されている I/O ポートはポート単位で基板周囲から 10 ピンコネクタを介して接続できます。また、インターフェースを配慮し LED による状態表示、プルアップなどの処理がなされています。また、シリアルポートは RS232C レベルで 1 ポート用意されており、9 ピン D-Sub コネクタが取り付けられ市販のモデムケーブル(ストレート)でパソコンと接続できます。

TK-3687 のプログラミングはルネサステクノロジが提供している無償版開発環境、「無償評価版コンパイラ」、HEW (High-performance Embedded Workshop)^{*1}を使用します。“HEW”のコンパイラは C 言語とアセンブラの双方に対応しており、メーカによるサポートはないものの実際の製品開発でも使用可能な性能を持っています。

プログラムのダウンロードには TK-3687 のフラッシュメモリにあらかじめ書き込まれている“ハイパーH8”を使用します。“ハイパーH8”は Windows に標準で搭載されているターミナルソフト「ハイパーターミナル」を使用した簡易モニタです。お手持ちのパソコンと TK-3687 のシリアルポートを RS-232C ケーブルで接続することで、C コンパイラが作成する HEX ファイルをダウンロード・実行することができます。“ハイパーH8”は C 言語のソースには対応していないものの、アセンブラレベルであれば基本的なデバッグ機能を備えています。トレース(=ステップ)実行時には、コンディションレジスタ、マシン語、及び逆アセンブルによるニーモニック表示が可能です。

C 言語の本格的なデバッグのために、エミュレータ “E8”^{*2}(ルネサステクノロジ)が用意されており、内蔵フラッシュメモリにプログラムをダウンロード後、C 言語でのソースデバッグができます。“E8”を使用することで、リアルタイムでのトレースや変数のウォッチなど、本格的な実機デバッグが可能になります。

効果的なマイコンの学習のためには CPU ボードだけではなく、CPU ボードに接続するインターフェースも必要です。本マニュアルには、実習テーマの一例を紹介するとともに、その回路を多少バージョンアップしたオプションが幾つか用意されています(弊社ホームページに記載)。

*1:HEW はルネサステクノロジーのサイトから最新版をダウンロードして下さい。(5 章参照)

*2:“E8”の詳細については弊社ユーザサポートにお尋ねください。

マニュアルについて

ルネサステクノロジーのサイト(<http://japan.renesas.com/>)から、マニュアルのダウンロードサイトに移動し、次のマニュアルをダウンロードして下さい。技術文書のため読みこなすのはかなりたいへんですが、欠かすことができない資料です。

「H8/3687 グループ ハードウェアマニュアル」(以降、ハードウェアマニュアルと呼びます)

「H8/300H シリーズ プログラミングマニュアル」(以降、プログラミングマニュアルと呼びます)

あとは HEW と一緒にパソコンにコピーされるマニュアルが、アセンブラや C の言語仕様を説明しています。これも読むのはたいへんですが、やはり欠かすことができません。

2 組み立て（キットでお買い求めのお客様へ）

完成品をお買い求め頂いたお客様はこの作業は不要です。次の章へお進み下さい。

■必要な工具

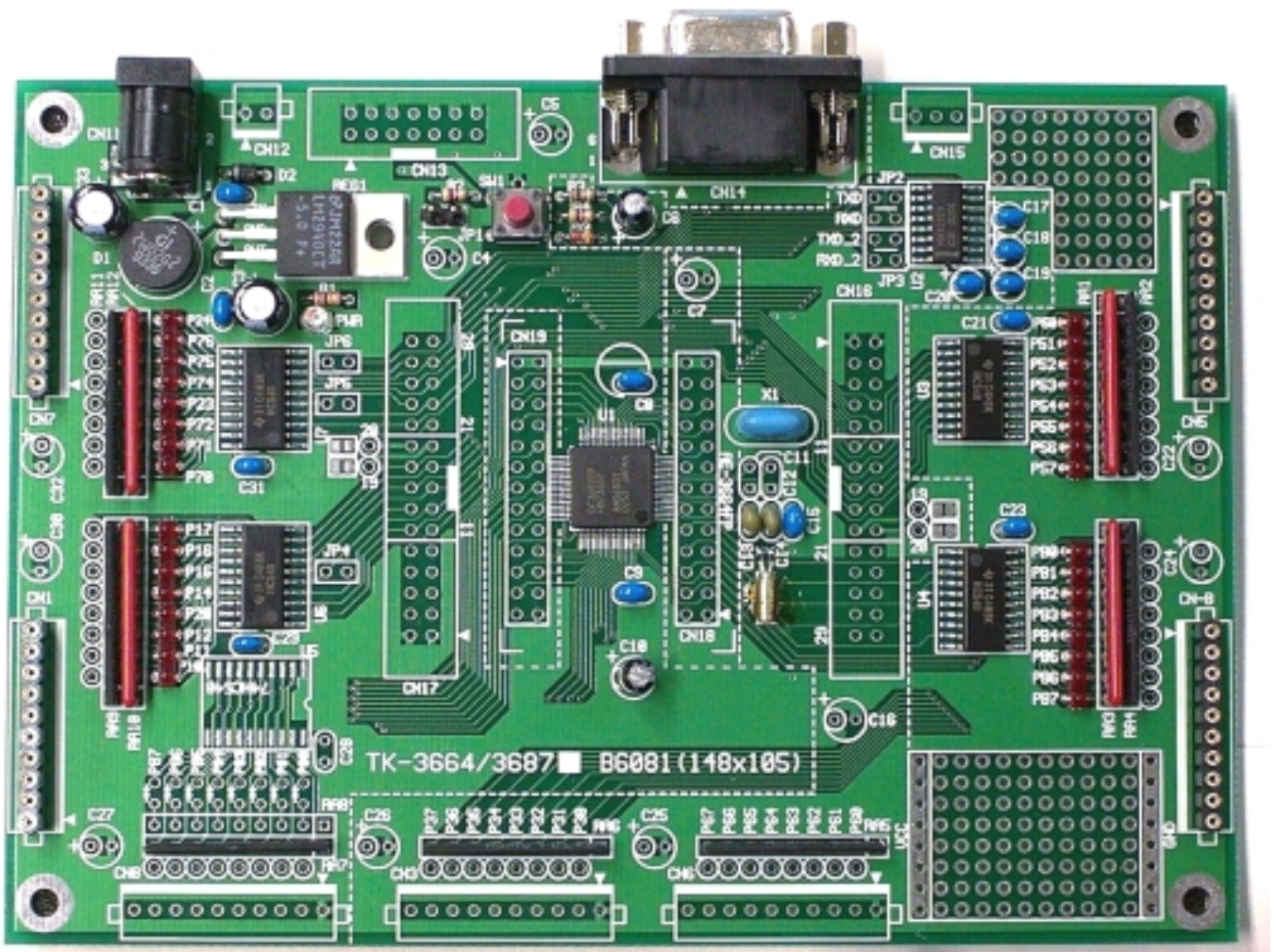
TK-3687 を組み立てるに当たって、次の工具が必要です。

半田ごて、ハンダ、ニッパー

その他、ピンセット、小手先を拭う為の濡らした布切れ等があると便利です。半田ごては大変熱くなりますので火傷には注意してください。うっかり触れてしまった際には急いで氷か水で冷やしてください。

■実装部品と配置

TK-3687 の完成写真を示します。実装を始める前に、大体の部品の種類と半田付けする位置を実際の基板を見て把握しておきましょう。



※写真は CN1,5,7,B に丸ピンを実装した状態です。丸ピンの実装は接続するインターフェースによって使い分けしてください。

■部品の確認

まず部品が全てそろっているか、巻末の部品表と照らし合わせて確認して下さい。

■基板の組み立て

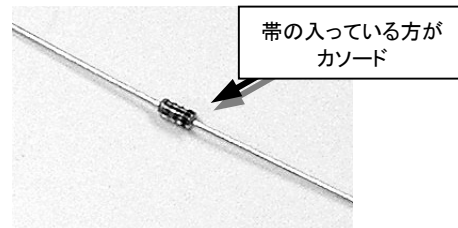
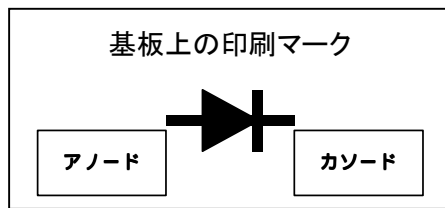
部品の確認ができれば基板に部品を半田付けしていきます。部品は全て部品面(白い印刷のある面)から挿し込み半田面(白い印刷の無い面)で半田付けしていきます。半田付けは次の順で行うと作業がし易いです。

抵抗・ダイオード→LED→セラミックコンデンサ→モジュール抵抗→その他

取り付けに注意が必要な部品のみ以下に示します。挿し間違えない様、よく説明文に目を通してから半田付けして下さい。もし、間違っ取り付けてしまった場合には、ハンダをしっかりと取り去ってから部品を外します。

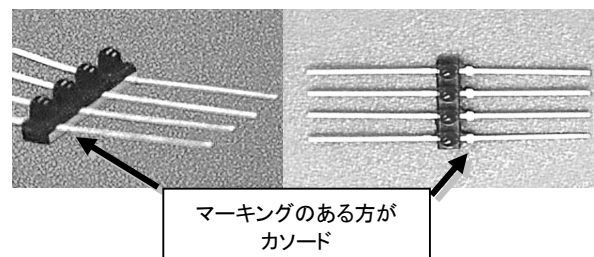
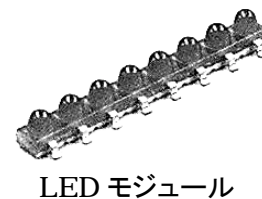
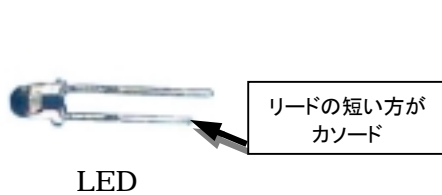
※ダイオード

部品に帯の入っている方がカソードです。逆に取り付けない様、基板に印刷されている記号に合わせて部品を挿し込み半田付けして下さい。



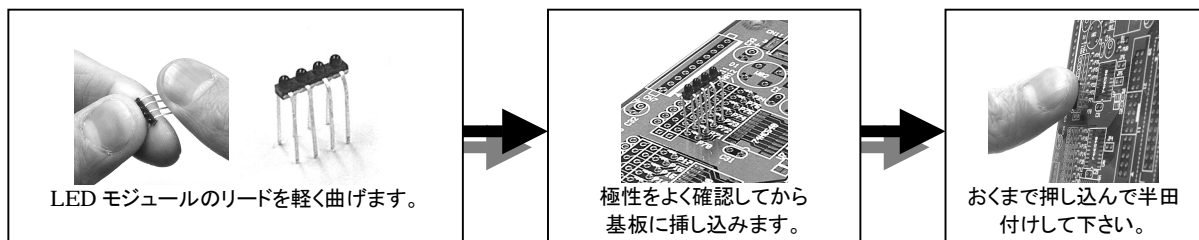
※LED

基板上で“PWR”には透明な LED を、また“Pxx”と記された8ヶ並んでいる場所には4連又は8連 LED モジュールを実装します。極性は、4連、又は8連の LED モジュールはリード(足)に出っ張りのある方、もしくは LED 本体のリードが出ている側面が白色に塗られている方がカソード(K)です。透明な LED は足の短い方がカソードです。基板に印刷されている記号に合わせて部品を挿し込み半田付けして下さい。

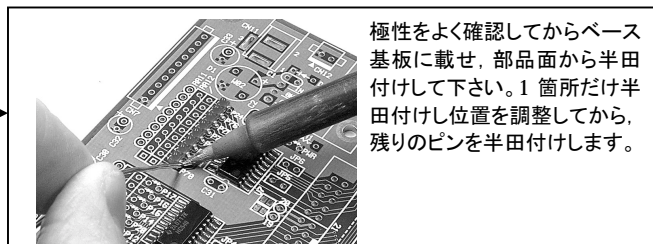
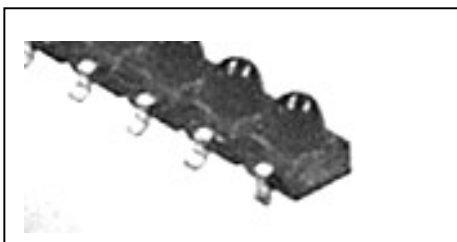


LED モジュールの実装方法を以下に示しますので説明文に従って実装して下さい。

※4連 LED モジュールの実装方法



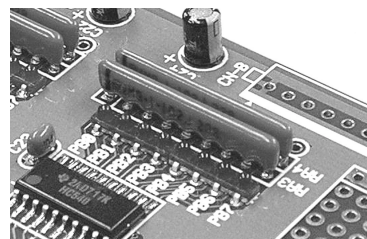
※8連 LED モジュールの実装方法



極性をよく確認してからベース基板に載せ、部品面から半田付けして下さい。1箇所だけ半田付けし位置を調整してから、残りのピンを半田付けします。

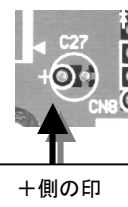
※抵抗モジュール

部品端の黒いライン又は白い点側と基板上の四角で囲んである穴を合わせて挿し込み半田付けします。尚、同じ形状でも定数が違うので、部品番号と定数を確認してから半田付けして下さい。



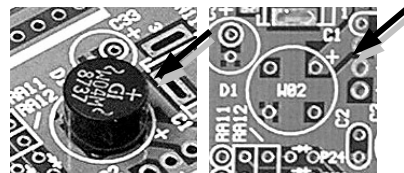
※電解コンデンサ

電解コンデンサはリードの長い方が“+”，短い方が“-”です。また、マイナス側には部品本体に白のラインが入っています。基板上ではプラス側に“+”印と挿し込む穴が円で囲ってありますので、それに合わせて実装して下さい。



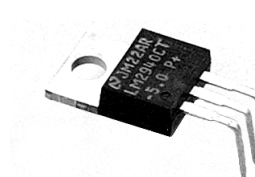
※ダイオードブリッジ

部品上面に“+”の印刷があるので、基板上の“+”の穴に合わせて挿し込み半田付けします。



※レギュレータ

リードを根元から約7mmのところまで曲げてから、基板に挿し込んで半田付けします。



全ての部品を実装し終わったら、最後にハンダ面の基板四隅にゴム足を貼り付けて完成です。



ハードが完成したら、次に動作チェックを行いません。次の章では、あらかじめ“TK-3687”のフラッシュメモリに書き込まれている簡易モニタ“ハイパーH8”を使用して I/O チェックプログラム“_piochk.mot”を実行する手順を説明します。

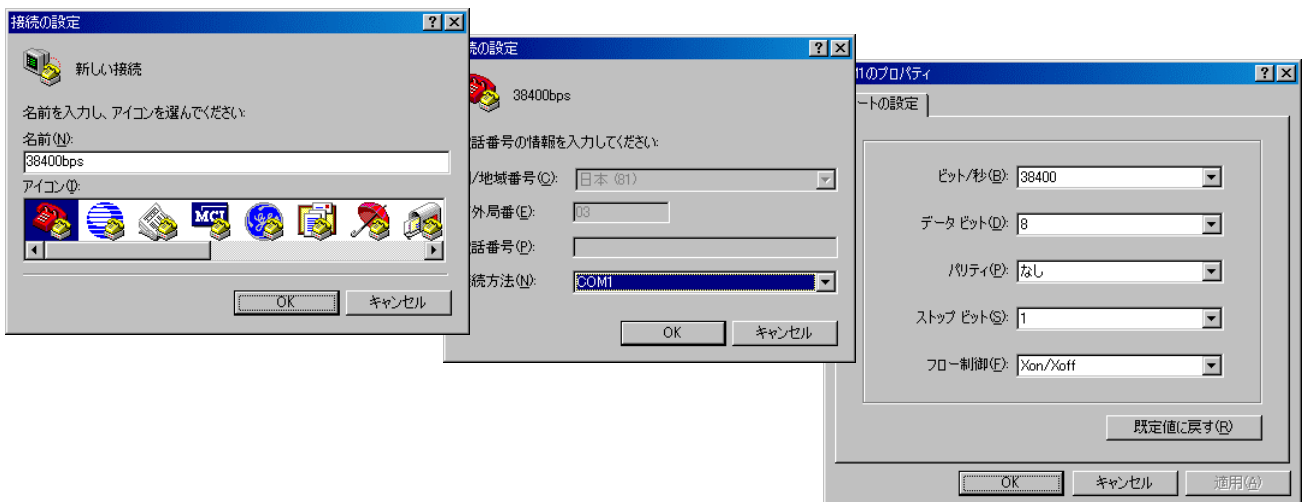
3 簡易モニタ“ハイパーH8”と動作チェック

■ハイパーH8とは

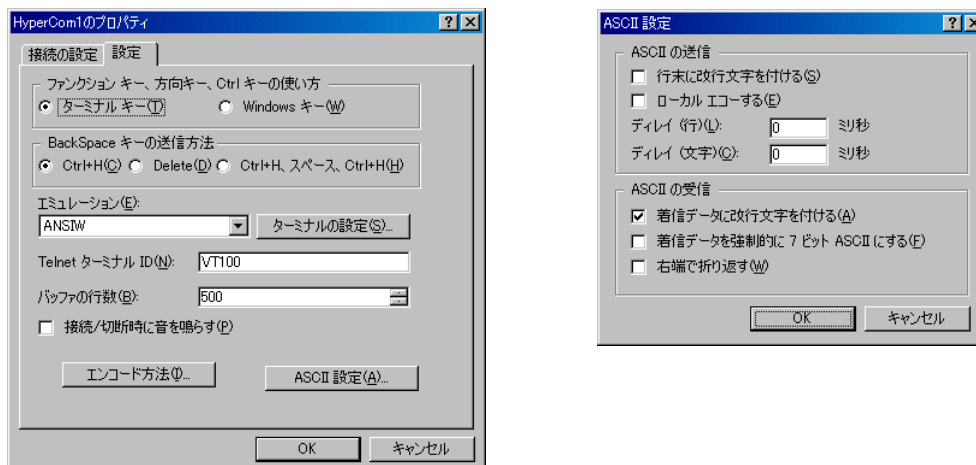
“ハイパーH8”は Windows95/98 等に標準で搭載されているターミナルソフト「ハイパーターミナル」を使用した簡易モニタです。お手持ちのパソコンとTK-3687のシリアルポートをRS-232Cケーブルで接続することで、簡単なモニタ環境を作ることができます。高度なデバッグ機能を必要としないビギナー向けのモニタで、HEWが出力するSタイプファイルのロードやプログラムの実行、ダンプ表示、メモリアド/ライト、レジスタ操作など、簡単なアセンブラアプリケーションのデバッグであれば十分な機能を備えています。但し、Cのソースデバッグに対応していないためここではプログラムのダウンローダとして使用します。本格的なデバッグにはエミュレータ“E7”を使用します。

■ハイパーターミナルの設定

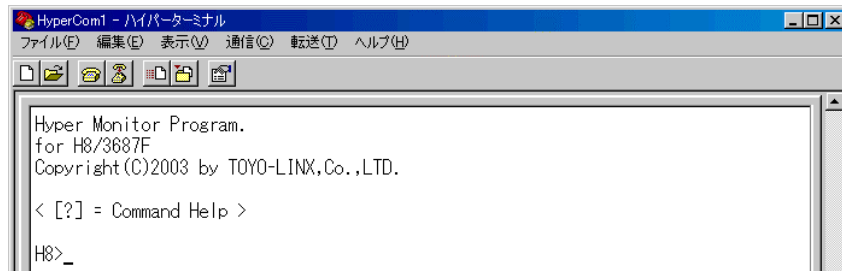
ハイパーH8を使用する為にハイパーターミナルの設定を行います。ハイパーターミナルを起動すると設定ダイアログが表示されるのでそれに従い設定して下さい。名前は接続速度がわかるように“38400bps”とします。接続方法はComNへダイレクト(Nは接続するComポートの番号)、ポートの設定は38400bps, 8bit, Non-P, Stop-1, Xon/Xoffです。設定し終わったらOKをクリックします。



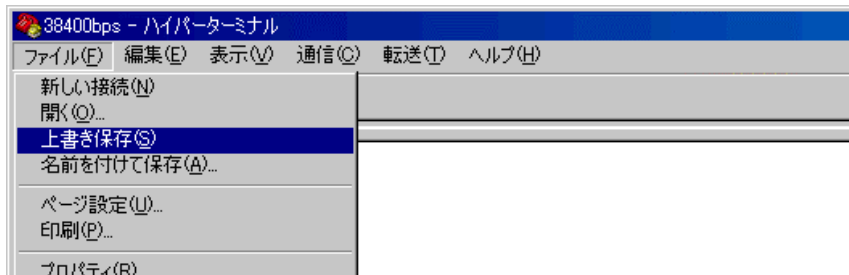
ポートの設定を終えたら次にファイル > プロパティからプロパティを開きます。設定タブをクリックしエミュレーションをANSIWに指定して下さい。次にASCII設定ボタンをクリックし、ASCIIの受信欄内の“着信データに改行文字を付ける”のみチェックしてOKをクリックします。



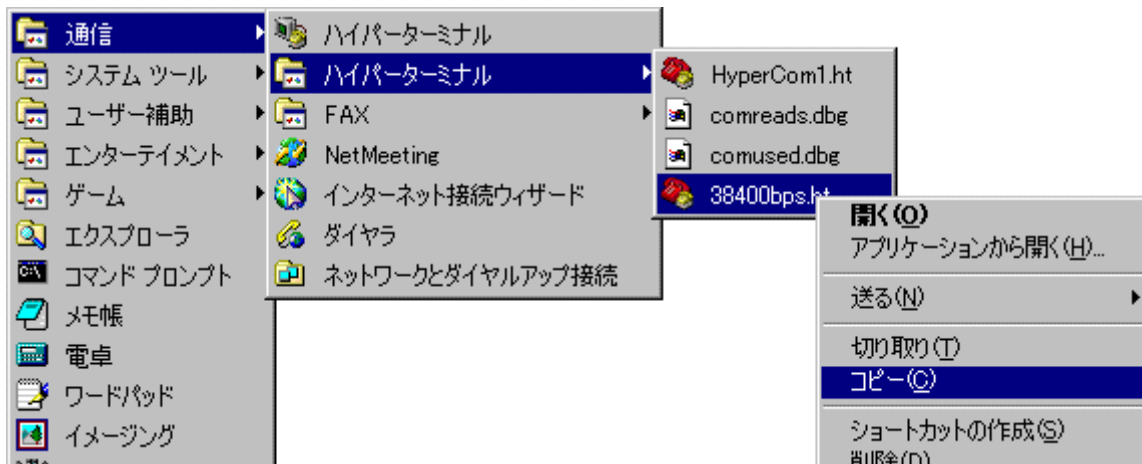
最後に CPU ボードとパソコンとを **RS-232C ストレートケーブル**で接続し電源を入れます。すると下記のような画面が表示されます。



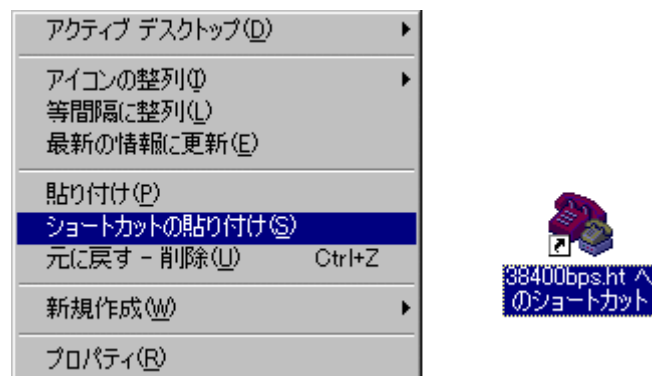
接続できたらハイパーターミナルの設定を保存しておきましょう。**ファイル > 上書き保存** を選択し保存して下さい。



すぐに呼び出せるようデスクトップにショートカットを作成しましょう。スタートメニューから **スタート > プログラム > アクセサリ > 通信 > ハイパーターミナル > 38400bps.ht** までカーソルを進め右クリックします。プルダウンメニューの中の **コピー** を選択して下さい。



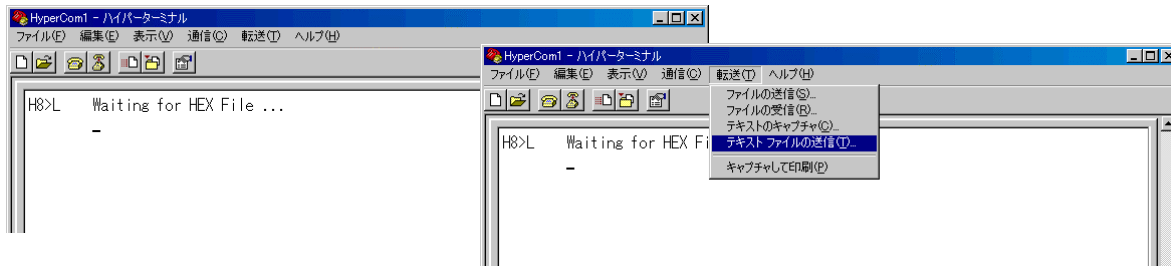
デスクトップで再度右クリックし **ショートカットの貼り付け** を選択してショートカットを作成します。



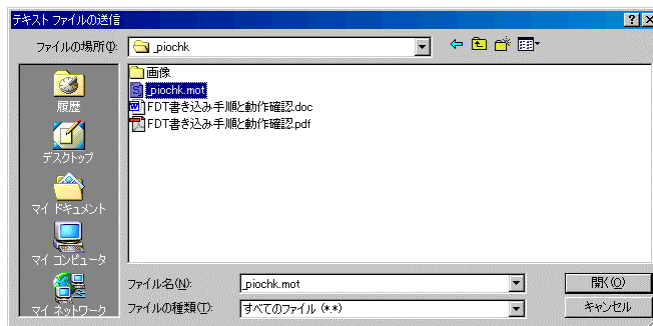
※ここで示した方法は Windows2000 の場合です。上記の方法でショートカットが作成できない場合はエクスプローラやファイルの検索を使用してデスクトップにショートカットを作成してください。

■HEX ファイルのロード

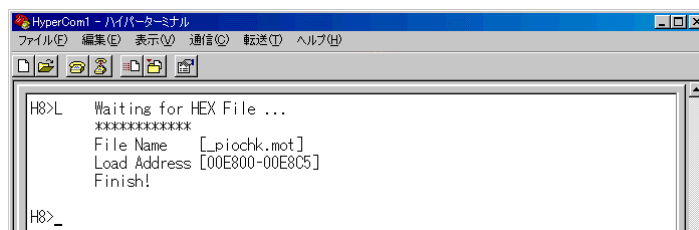
まずは I/O チェックプログラム“_piochk.mot”をロードします。ファイルは付属の CD-ROM に添付されています。コマンド“L”を入力し **Enter** キーを押すと“Waiting for HEX File...”の表示とともに HEX ファイルのロード待ちになるので、メニューから**転送 > テキストファイルの送信**を選択します。



テキストファイルの送信ウィンドウから転送する HEX ファイル“_piochk.mot”を選択します。なお、HEW で生成される HEX ファイルはモトローラ形式(.mot)ですので予めファイルの種類を全てのファイル(*.*)にして下さい。ファイルの場所は、E:¥TK-3687¥マニュアル¥_piochk.mot (CD-ROM ドライブが E の場合) です。

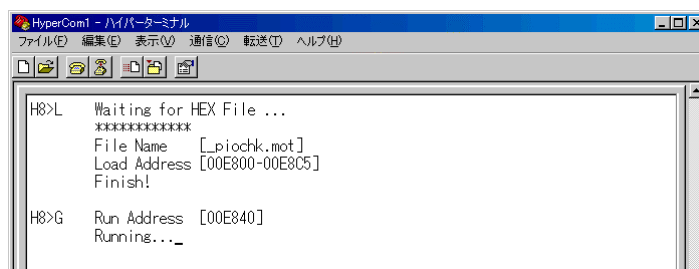


転送の経過は * で表示されます (プログラムが極端に短い場合は表示されない場合があります)。ロードアドレスと Finish! の文字が表示されればロード完了です。

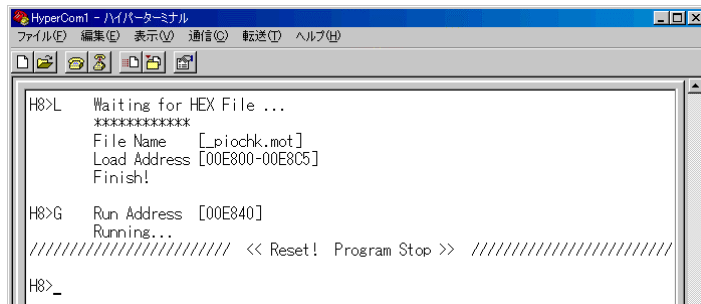


■実行

ロードが終了したらプログラムを実行してみましょう。コマンド“G”を入力し **Enter** キーを押します。ロードしたプログラムの先頭アドレスから実行が開始され、基板上的 LED がスウィングします。これで、動作チェック OK です。

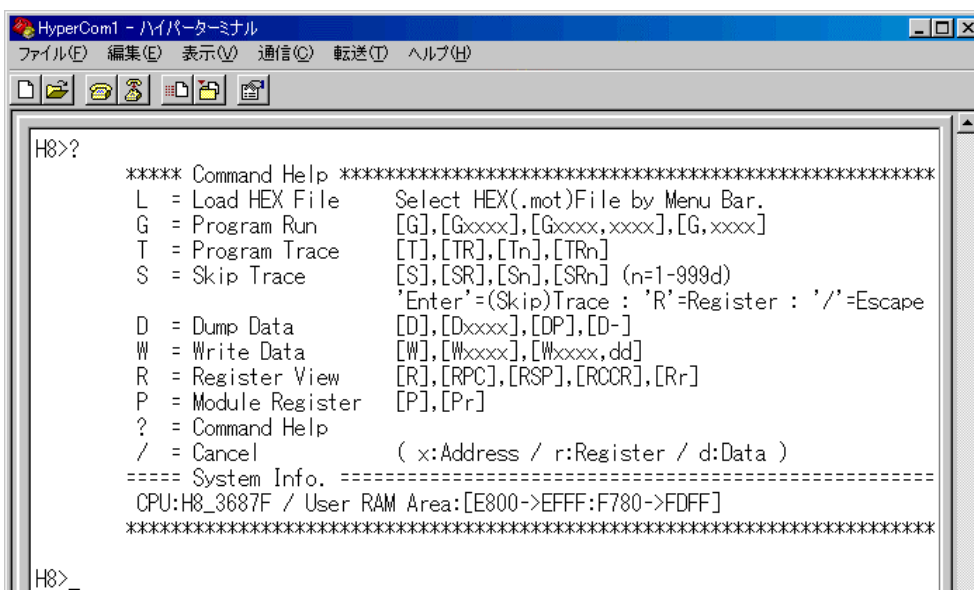


実行を終了する場合はボードの **Reset** ボタンを押して下さい。リセットしてもロードしたプログラムは保持されています。

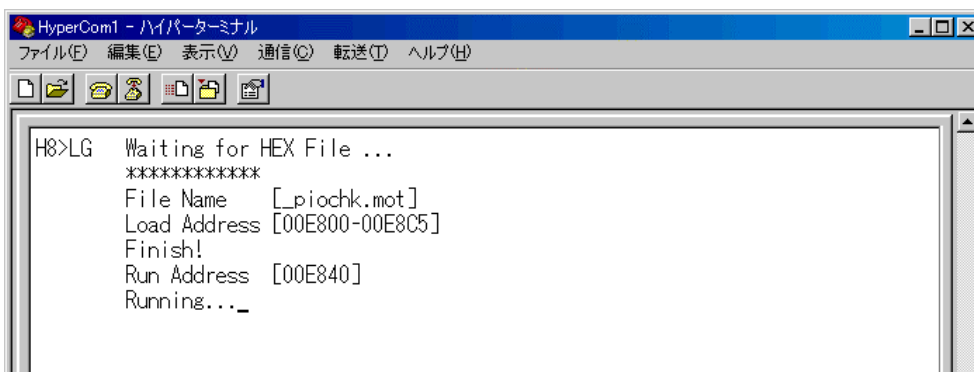


■その他の機能

ハイパーH8 には今まで使用したコマンド以外にもトレース実行、レジスタの表示・変更やメモリダンプ等、アセンブラのデバッグをサポートする機能が盛り込まれています。“?”を入力するとコマンドの一覧と入力形式が表示されますので、初心者でもマニュアル無しに簡単に使いこなすことができます。



更にコマンドの連結が可能で、例えば“LG”と入力するとプログラムをロード後、直ちに実行させることもできます。



また、直前のコマンドをリピートする際には、**Enter** キーのみでOKです。

4 メモリーマップ

HEW を使うときのコツの一つは、メモリーマップを意識する、ということです。プログラムがどのアドレスに作られて、データはどのアドレスに配置されるか、ちょっと意識するだけで、HEW を理解しやすくなります。ハイパーH8 を使うときのメモリーマップは次のとおりです。

0000 番地	モニタプログラム 'ハイパーH8'		ROM/フラッシュメモリ (56K バイト)
DFFF 番地	未使用		未使用
E000 番地	未使用		未使用
E7FF 番地	未使用		未使用
E800 番地	ハイパーH8 ユーザ割り込みベクタ		RAM (2K バイト)
E860 番地	PResetPRG	リセットプログラム	
	PIntPRG	割り込みプログラム	
EA00 番地	P	プログラム領域	
	C	定数領域	
	C\$DSEC	初期化データセクションのアドレス領域	ユーザ RAM エリア
	C\$BSEC	未初期化データセクションのアドレス領域	
	D	初期化データ領域	
EFFF 番地	未使用		
F000 番地	未使用		未使用
F6FF 番地	未使用		未使用
F700 番地	I/O レジスタ		I/O レジスタ
F77F 番地	I/O レジスタ		I/O レジスタ
F780 番地	B	未初期化データ領域 初期化データ領域 (変数領域)	RAM (1K バイト) フラッシュメモリ書換え用 ワークエリアのため、 FDT と E7, E8 使用時は、 ユーザ使用不可
	R		
FB7F 番地			
FB80 番地			
FD80 番地	S	スタック領域	RAM (1K バイト)
FDFE 番地			
FE00 番地	ハイパーH8		
FF7F 番地	ワークエリア		
FF80 番地	I/O レジスタ		I/O レジスタ
FFFF 番地	I/O レジスタ		I/O レジスタ

メモリーマップのうちユーザ RAM エリアの部分だけが自由に使用できるエリアです。

“E8”使用時(デフォルトの HEW の設定値)の TK-3687 のメモリーマップを示します。マップ中の“ユーザプログラムエリア”, “ユーザ RAM エリア”の範囲がユーザが自由に使用できるエリアで, H8/3687 の全てのメモリエリアを自由に使うことができます。基本的にセクションは HEW が自動的に割り振ります。プログラムの都合でセクションを変更する場合は下記のアドレスを参考にして割り振ってください。

0000 番地	割り込みベクタ			
0400 番地	PResetPRG	リセットプログラム	ユーザプログラム エリア	ROM/フラッシュメモリ (56K バイト)
	PIntPRG	割り込みプログラム		
0800 番地	P	プログラム領域		
	C	定数領域		
	C\$DSEC	初期化データセクションのアドレス領域		
	C\$BSEC	未初期化データセクションのアドレス領域		
	D	初期化データ領域		
DFFF 番地				
E000 番地	未使用			未使用
E7FF 番地				
E800 番地	B	未初期化データ領域 初期化データ領域 (変数領域)	ユーザ RAM エリア	RAM (2K バイト)
	R			
E8FF 番地				
EF00 番地	S	スタック領域		
EFFF 番地				
F000 番地	未使用			未使用
F6FF 番地				
F700 番地	I/O レジスタ			I/O レジスタ
F77F 番地				
F780 番地	フラッシュメモリ書換用ワークエリア (使用禁止)			RAM (1K バイト)
FB7F 番地				
FB80 番地			ユーザ RAM エリア	RAM (1K バイト)
FF7F 番地				
FF80 番地	I/O レジスタ			I/O レジスタ
FFFF 番地				

5 無償評価版コンパイラ“HEW”のインストール

ルネサステクノロジは現在、High-performance Embedded Workshop V. 4 (HEW4)に対応した無償評価版コンパイラを公開しています (Tiny/SLP 無償版コンパイラはなくなりました)。無償評価版コンパイラは、はじめてコンパイルした日から 60 日間は製品版と同等の機能と性能のままです。61 日目以降はリンクサイズが 64K バイトまでに制限されますが、H8/3687 はもともとアクセスできるメモリサイズが 64K までバイトなので、この制限は関係ありません。また、無償評価版コンパイラは製品開発では使用できないのですが、H8/300H Tiny シリーズ (H8/3687 も含まれる) では許可されています。この項では無償評価版コンパイラのダウンロードからインストールまでを説明します。

1. HEW の入手

TK-3687 のプログラミングで使用するアセンブラは、High-performance Embedded Workshop V.4 (HEW4)に対応した無償評価版コンパイラに含まれており、株式会社ルネサステクノロジのホームページよりダウンロードします。ダウンロードサイトの URL は以下の通りです。(ただし E8 には同封されます)



株式会社ルネサステクノロジ
<http://www.renesas.com/jpn/>

無償評価版コンパイラ
ダウンロードサイト
[http://www.renesas.com/jpn/products/mpumcu/
tools/download/h8c/index.html](http://www.renesas.com/jpn/products/mpumcu/tools/download/h8c/index.html)

ダウンロードサイトの下の方にある「ダウンロードのページへ」をクリックして下さい。次のページで必須事項を入力してダウンロードを開始します。ダウンロード先はデスクトップにすると便利です。全部で 69.4 MByte になりますので、ADSL か光回線でない、かなり大変なのが実情です。‘h8cv601r00.exe’ というファイルがダウンロードされます。

最新版の HEW を手に入れましょう

HEW はときどきバージョンアップされます。HEW はルネサステクノロジのマイコン全てに対応しているため、H8 シリーズはもとより、R8 シリーズや SH シリーズなど、対応するマイコンが増えるとそのたびにマイナーチェンジされるようです。また、その際に報告されていた不具合を一緒に修正することもあります。

それで、ルネサステクノロジのホームページは定期的にのぞいてみることをおすすめします。特にデバイスアップデータの情報は要注意です。

ところで、ここでダウンロードした無償評価版コンパイラには不具合があることが報告されています。それで、ルネサステクノロジが公開しているデバイスアップデータを使用して不具合を修正します。デバイスアップデータは下記の URL のサイトからダウンロードできます。

この画面は 2005 年 4 月現在です

バージョン	公開日	更新内容			ダウンロード
		SHC/C++ コンパイラ	H8C/C++ コンパイラ	Tiny/SLP コンパイラ	
更新一覧(SH)		更新一覧(H8)	更新一覧(Tiny/SLP)	Download	
更新一覧(SH)		更新一覧(H8)	更新一覧(Tiny/SLP)	-	
更新一覧(SH)		更新一覧(H8)	更新一覧(Tiny/SLP)	-	
DeviceUpdater V.1.02	2004.11.05	更新一覧(SH)	更新一覧(H8)	更新一覧(Tiny/SLP)	-
DeviceUpdater V.1.01	2004.08.05	更新一覧(SH)	更新一覧(H8)	更新一覧(Tiny/SLP)	-
DeviceUpdater V.1.00	2004.06.21	更新一覧(SH)	更新一覧(H8)	更新一覧(Tiny/SLP)	-

【特定デバイスを使用する際の注意事項】
DeviceUpdaterを以下のコンパイラパッケージに適用した場合、ある特定のデバイスに対しては、以下の特別な設定が必要になりますのでご注意ください。

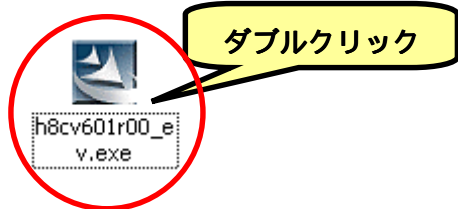
デバイス名	対象コンパイラパッケージ	必要な設定
H8/38086 H8/38076 H8/38002	Tiny/SLPコンパイラ	プロジェクト構築後、ビルド前にCPUオプションを以下の手順により「Tiny」に変更してください。 1. HEWメニュー「オプション」→「H8_Tiny/SLP」を選択してください。 2. 「CPU」タブの「CPU」リストを選択してください。 3. 「CPU」ドロップダウンリストの選択権を、「Tiny」に変更してください。 4. ビルドを行ってください。

デバイスアップデータ
ダウンロードサイト
http://www.renesas.com/jpn/products/mpumcu/tool/download2/coding_tool/hew/utilities/device_updata/index.html

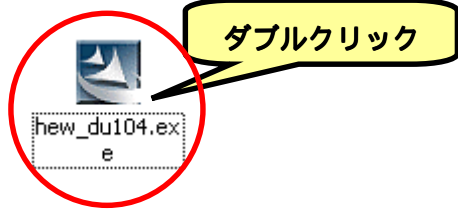
ページの下の方にある「Download」をクリックしてください。ダウンロード先はデスクトップにすると便利です。全部で 3.55MByte になります。'hew_du104.exe' というファイルがダウンロードされます。

2. HEW のインストール

ダウンロード先をデスクトップにした場合で説明します。ダウンロードした 'h8cv601r00.exe' をダブルクリックしてください。すると、インストールが始まります。画面の指示に従ってインストールしてください。

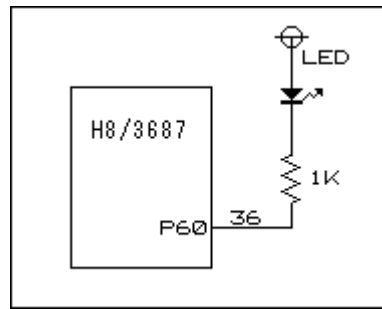


次に、無償評価版コンパイラをアップデートします。ダウンロードした 'hew_du104.exe' をダブルクリックしてください。すると、インストールが始まります。画面の指示に従ってインストールしてください。



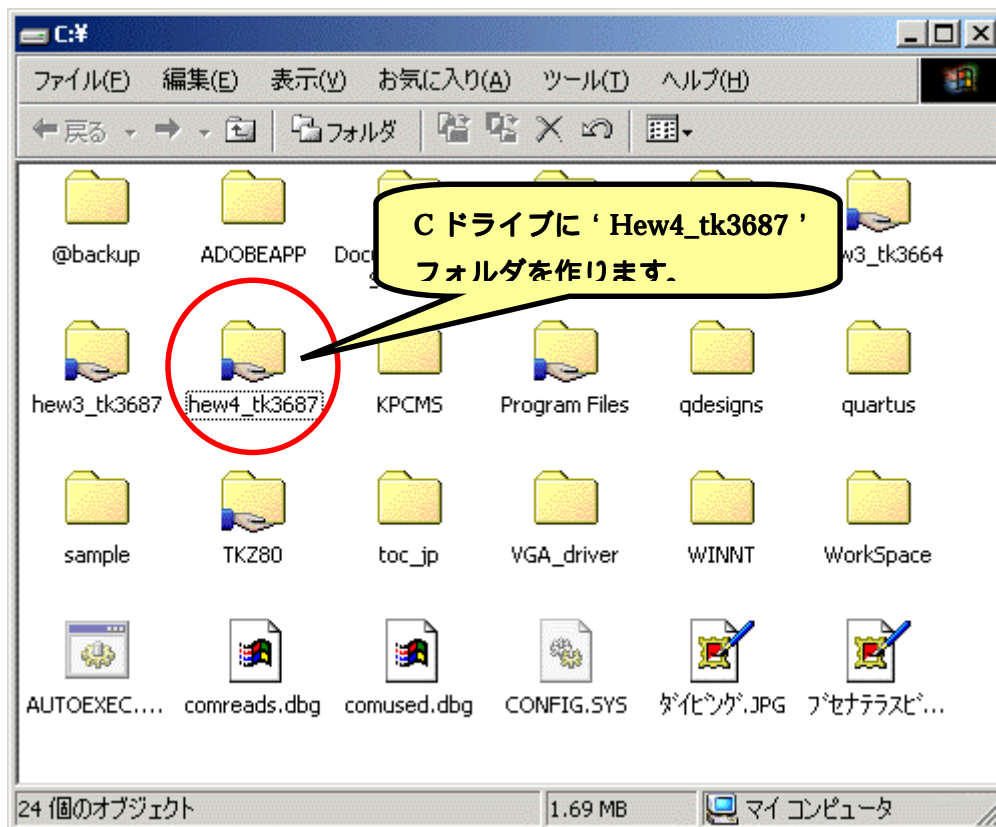
6 プログラムの作成

ここで作るプログラムは P60 につないだ LED を点滅させるというものです。回路図は次のとおりです。

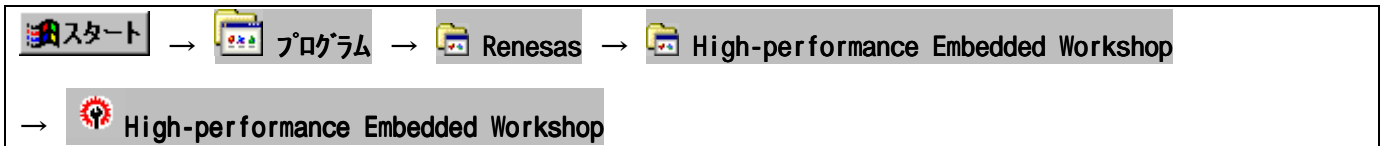


HEW ではプログラム作成作業をプロジェクトと呼び、そのプロジェクトに関連するファイルは1つのワークスペース内にまとめて管理されます。通常はワークスペース、プロジェクト、メインプログラムには共通の名前がつけられます。この章で作るプロジェクトは 'IoPort_led_c' と名付けます。以下に、新規プロジェクト 'IoPort_led_c' を作成する手順と動作確認の手順を説明します。

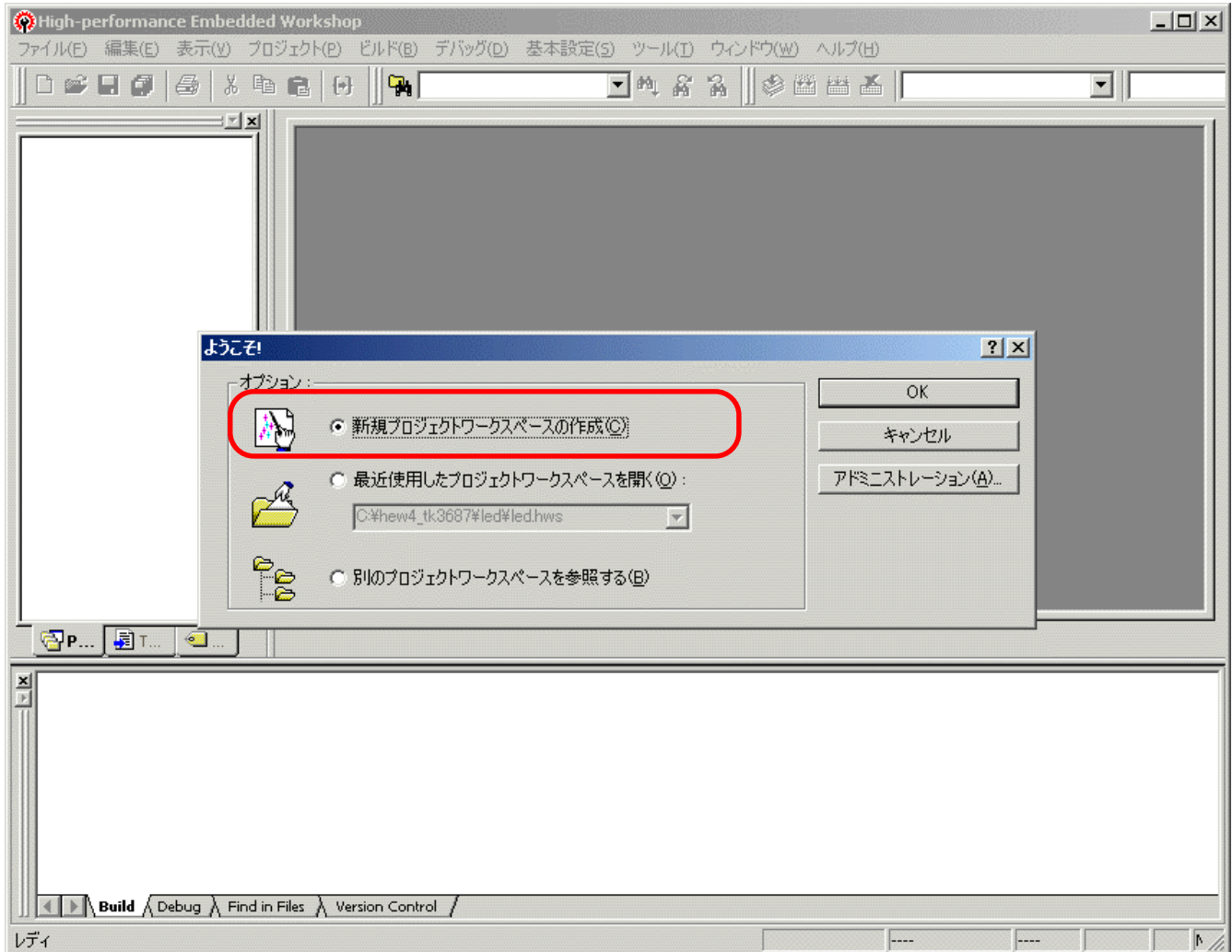
しかしその前に、HEW 専用作業フォルダを作っておきましょう。C ドライブに 'Hew4_tk3687' を作ってください。このマニュアルのプロジェクトは全てこのフォルダに作成します。



では、HEW を起動しましょう。スタートメニューから起動します。



HEW を起動すると下記の画面が現れるので、「新規プロジェクトワークスペースの作成」を選択して「OK」をクリックします。



前に作ったプロジェクトを使うとき

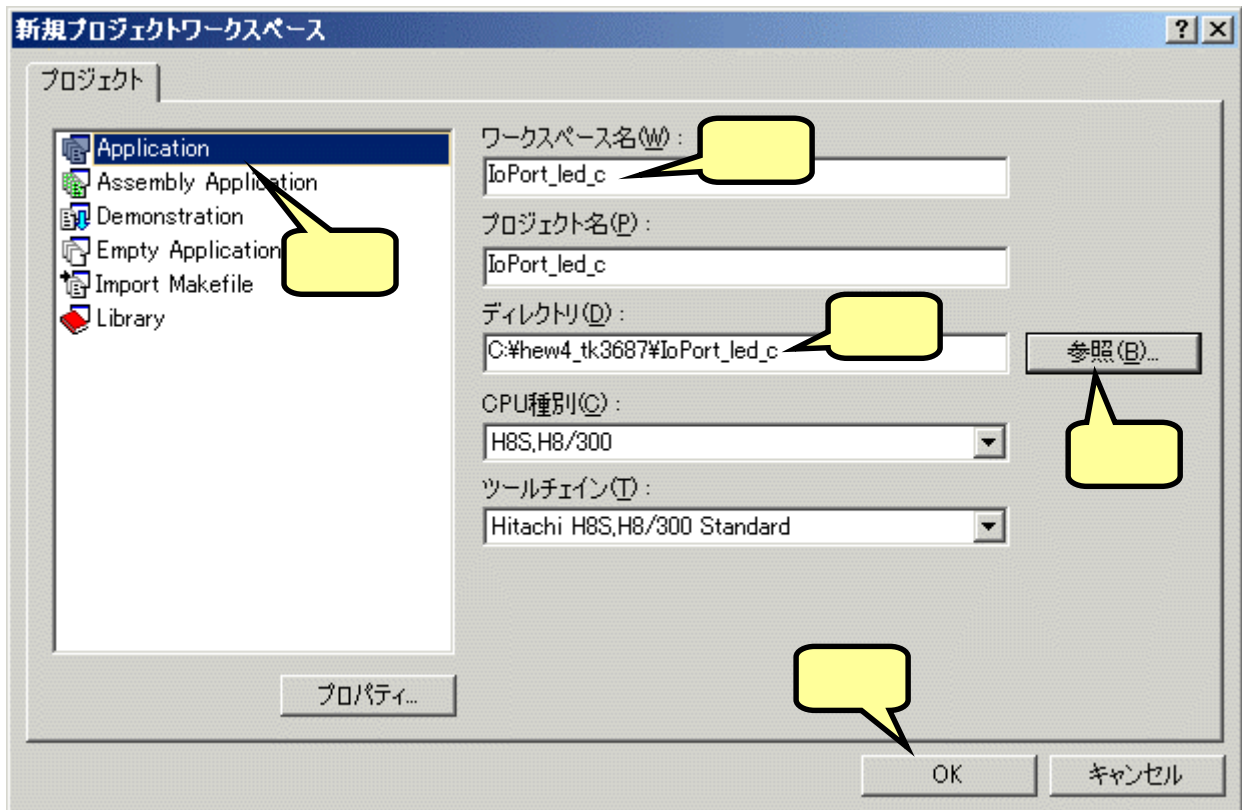
その場合は、「ようこそ!」ダイアログで「最近使用したプロジェクトワークスペースを開く」を選択して「OK」をクリックします。そのプロジェクトの最後に保存した状態で HEW が起動します。

まず、①「ワークスペース名(W)」(ここでは 'IoPort_led_c')を入力します。「プロジェクト名(P)」は自動的に同じ名前になります。

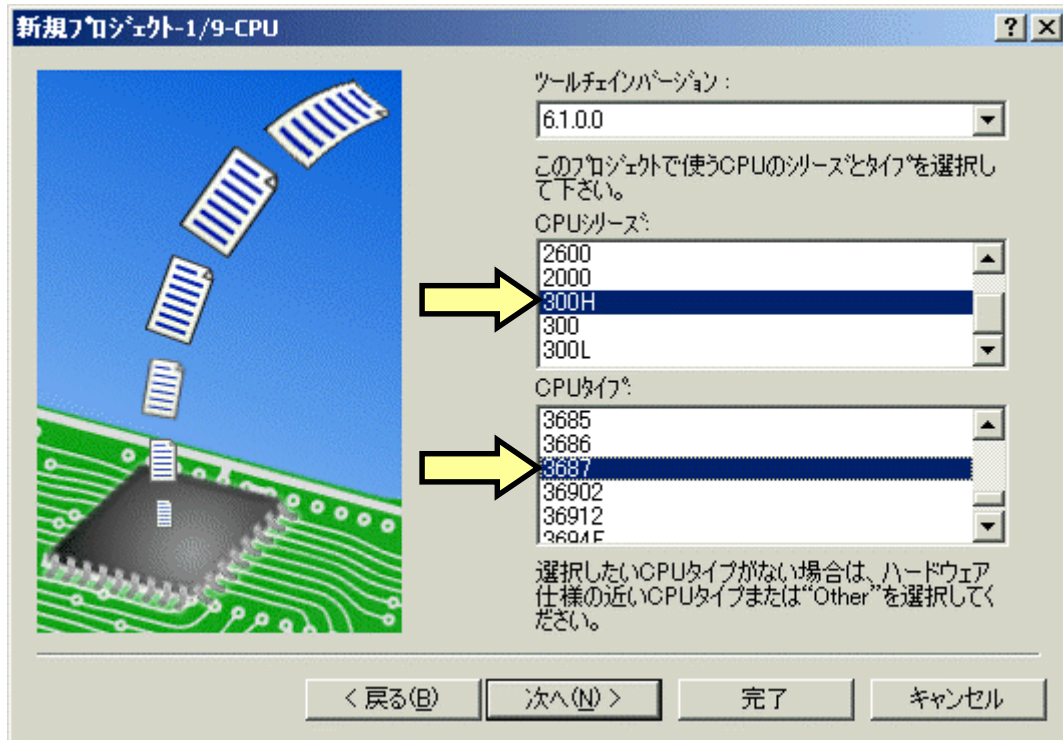
ワークスペースの場所を指定します。②右の「参照(B)...」ボタンをクリックします。そして、あらかじめ用意した HEW 専用作業フォルダ(ここでは Hew4_tk3687)を指定します。設定後、「ディレクトリ(D)」が正しいか確認して下さい。(③)

次にプロジェクトを指定します。今回は C 言語なので④「Application」を選択します。

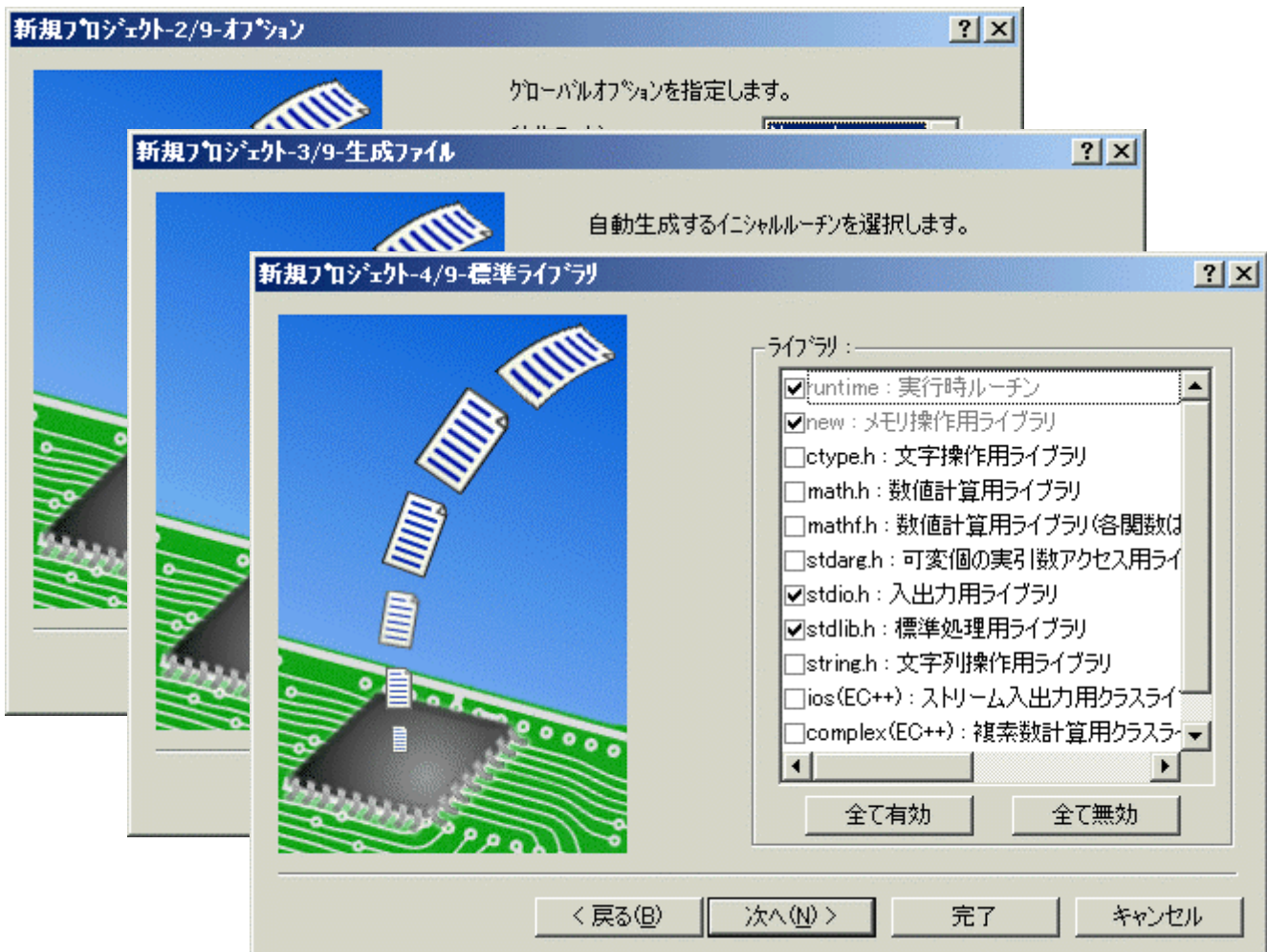
入力が終わったら⑤「OK」をクリックして下さい。



「新規プロジェクト-1/9-CPU」で、使用する CPU シリーズ(300H)と、CPU タイプ(3687)を設定し、「次へ(N)>」をクリックします。



「新規プロジェクト-2/9-オプション」、「新規プロジェクト-3/9-生成ファイル」、「新規プロジェクト-4/9-標準ライブラリ」は変更しません。「次へ(N)>」をクリックして次の画面に進みます。



***** E7, E8 を使用するとき *****

「新規プロジェクト-5/9-スタック領域」は変更しません。「次へ(N)>」をクリックして次の画面に進みます。



***** ハイパーH8 を使用するとき *****

「新規プロジェクト-5/9-スタック領域」でスタックのアドレスとサイズを変更します。ハイパーH8 を使用するので、①スタックポインタを H'FE00 に、②スタックサイズを H'80 にします。設定が終わったら「次へ(N)>」をクリックします。

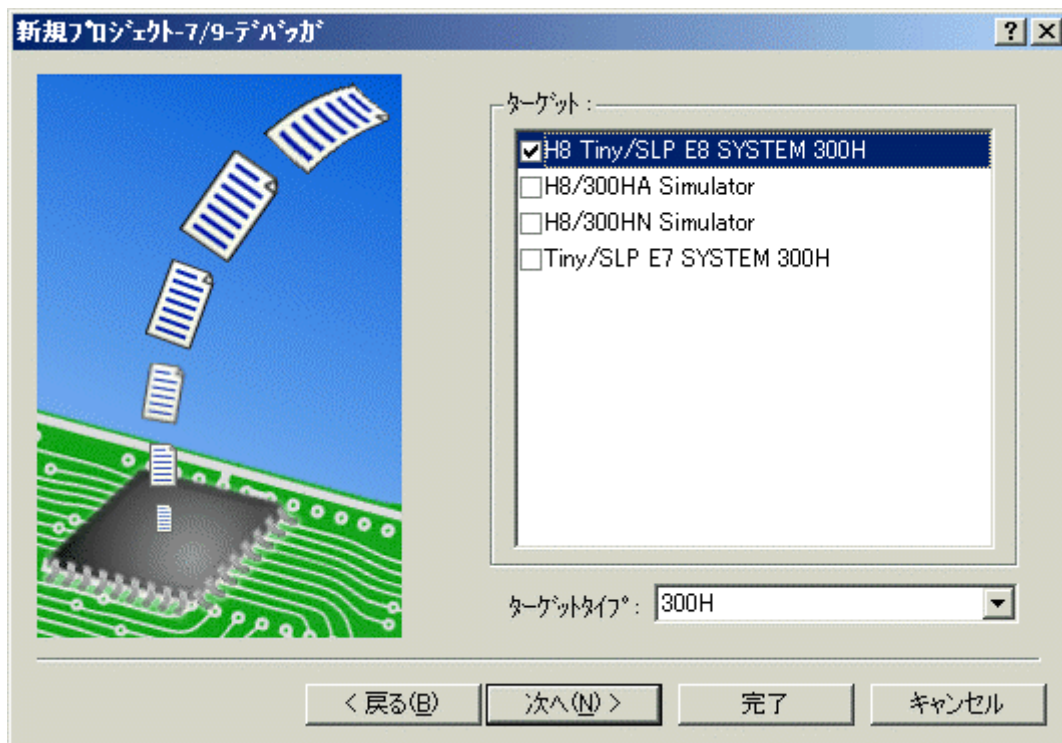


「新規プロジェクト-6/9-ベクタ」は変更しません。「次へ(N)>」をクリックして次の画面に進みます。



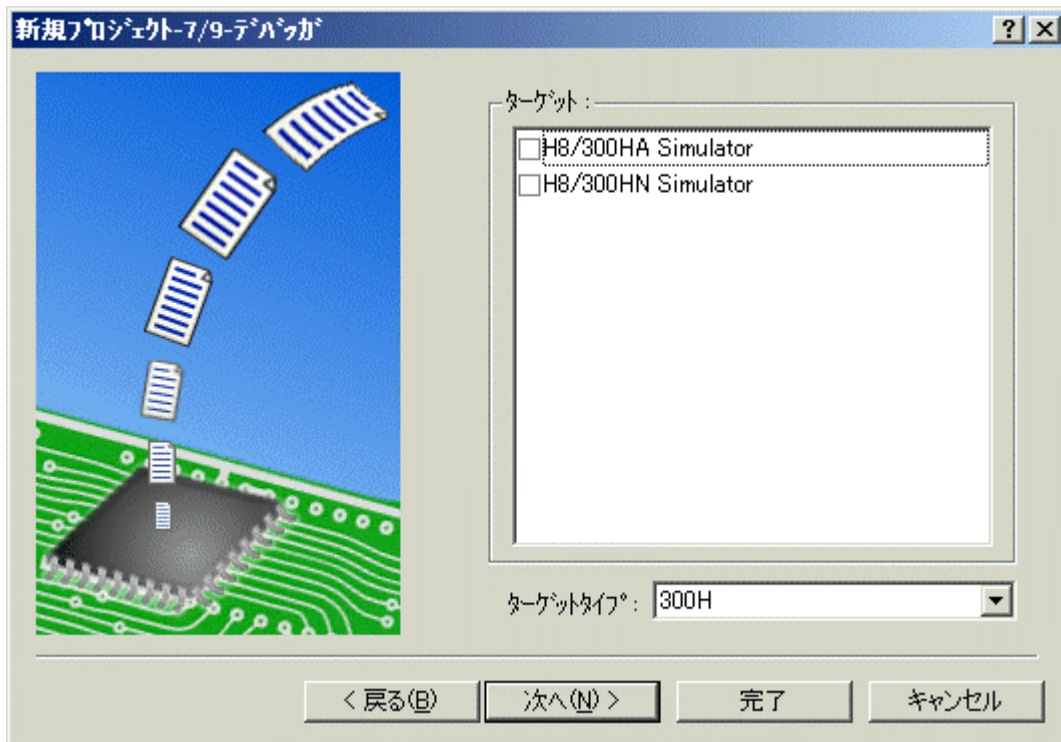
******* E7, E8 を使用するとき *******

「新規プロジェクト-7/9-デバッグ」で、使用するデバッグにチェックを入れて設定し(E8 のときは「H8 Tiny/SLP E8 SYSTEM 300H」、E7 のときは「Tiny/SLP E7 SYSTEM 300H」)、「次へ(N)>」をクリックします。



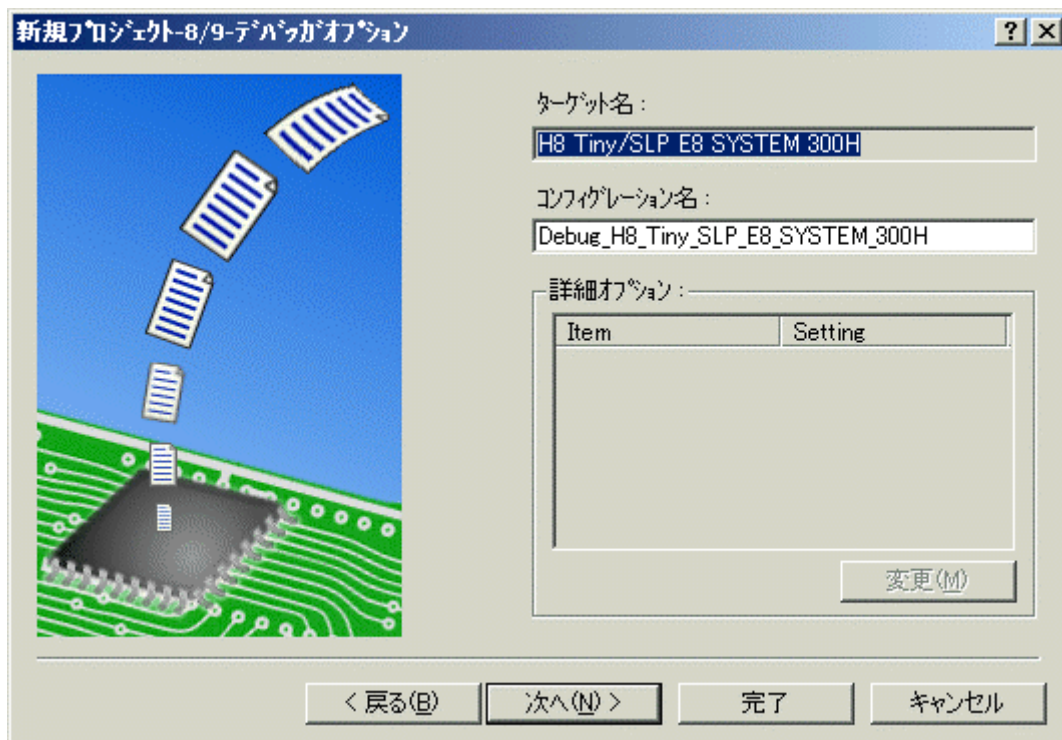
***** ハイパーH8 を使用するとき *****

「新規プロジェクト-7/9-デバッガ」は変更しません。「次へ(N)>」をクリックして次の画面に進みます。



***** E7, E8 を使用するとき *****

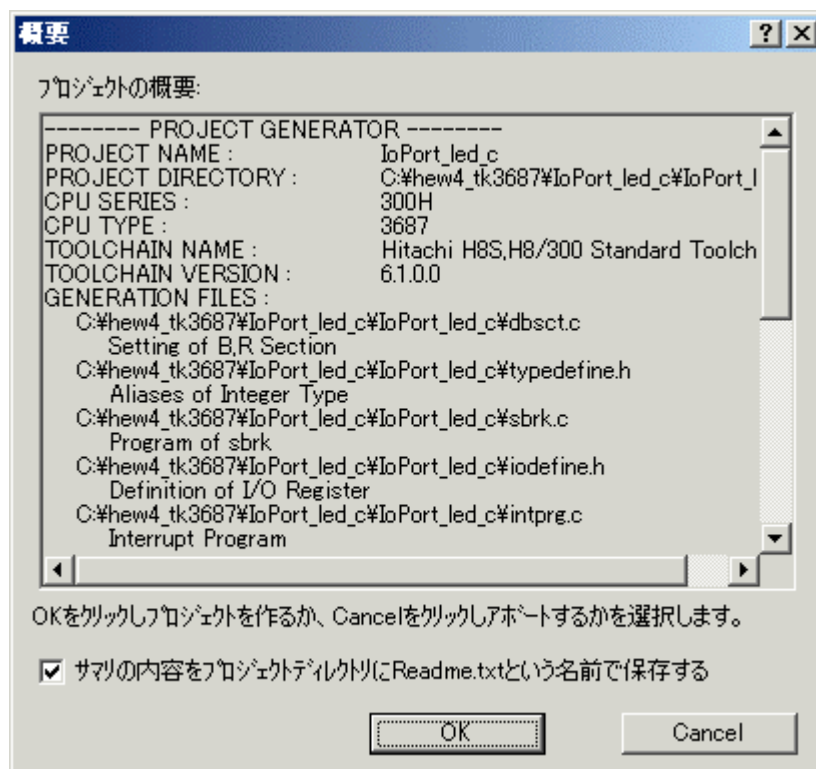
E7, E8 を使用しているときだけ「新規プロジェクト-8/9-デバッガオプション」のダイアログが開きます。ここは変更しません。「次へ(N)>」をクリックして次の画面に進みます。



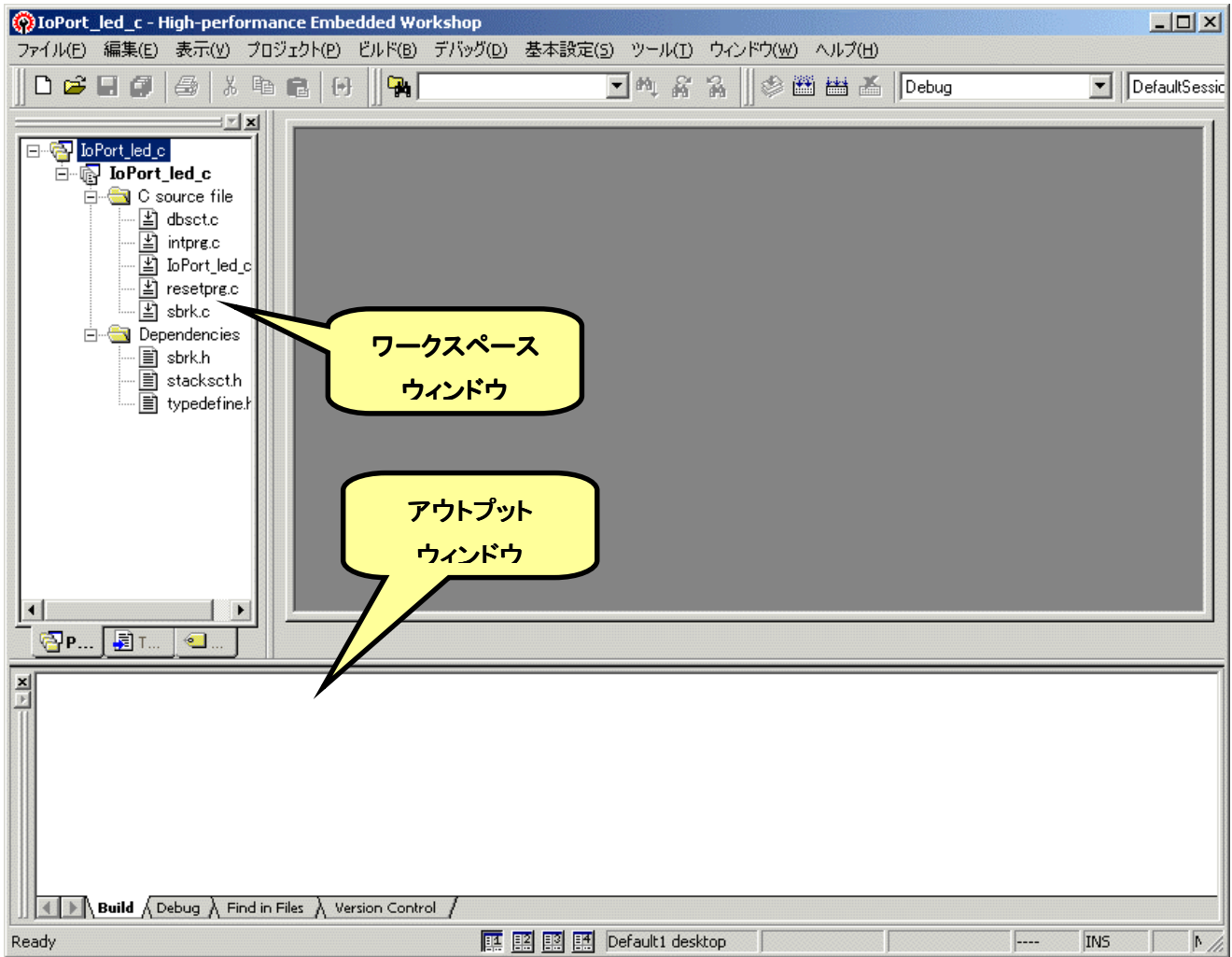
次は「新規プロジェクト-9/9-生成ファイル名」です。ここも変更しません。「完了」をクリックします。



すると、「概要」が表示されるので「OK」をクリックします。

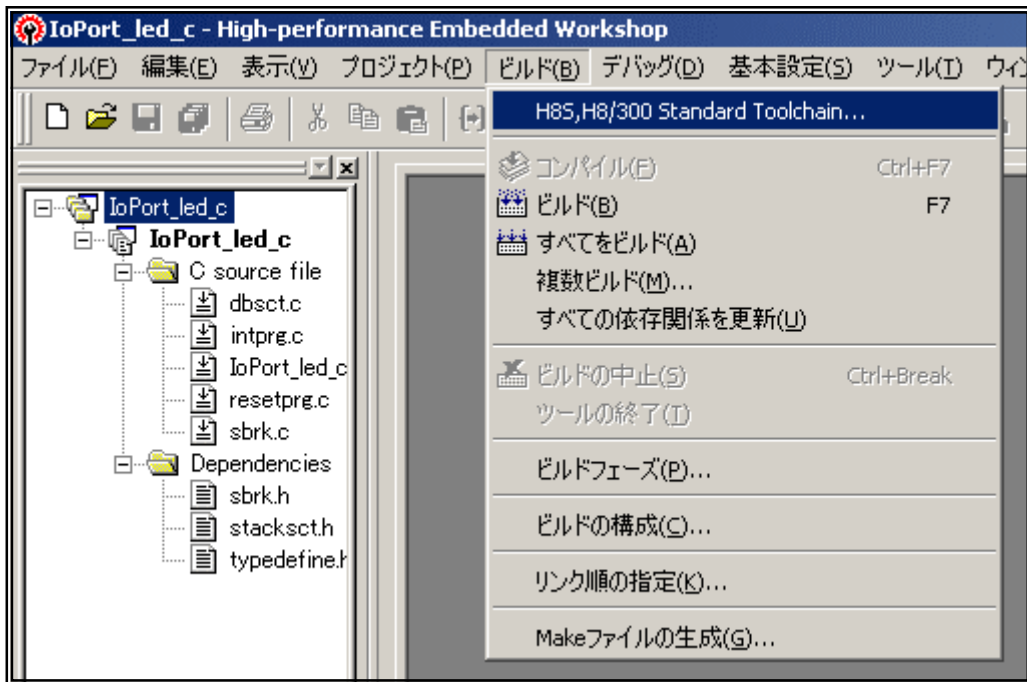


これで、プロジェクトワークスペースが完成します。HEW はプロジェクトに必要なファイルを自動生成し、それらのファイルは左端のワークスペースウィンドウに一覧表示されます。

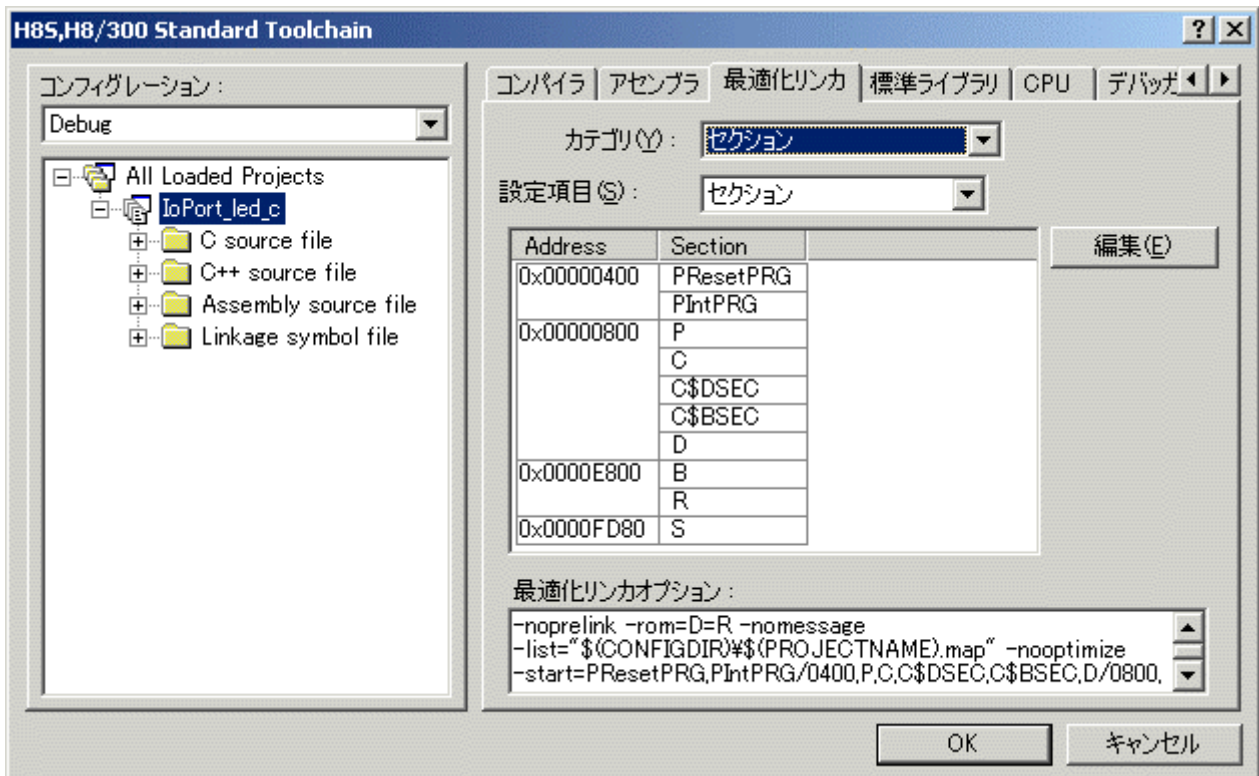


さて、これでプロジェクトは完成したのですが、ハイパー-H8 を使うときはセクションを変更してプログラムがRAM上に行けるようにします。(当然ながら、ハイパー-H8を使わないときは変更する必要はなく、そのままOK。)

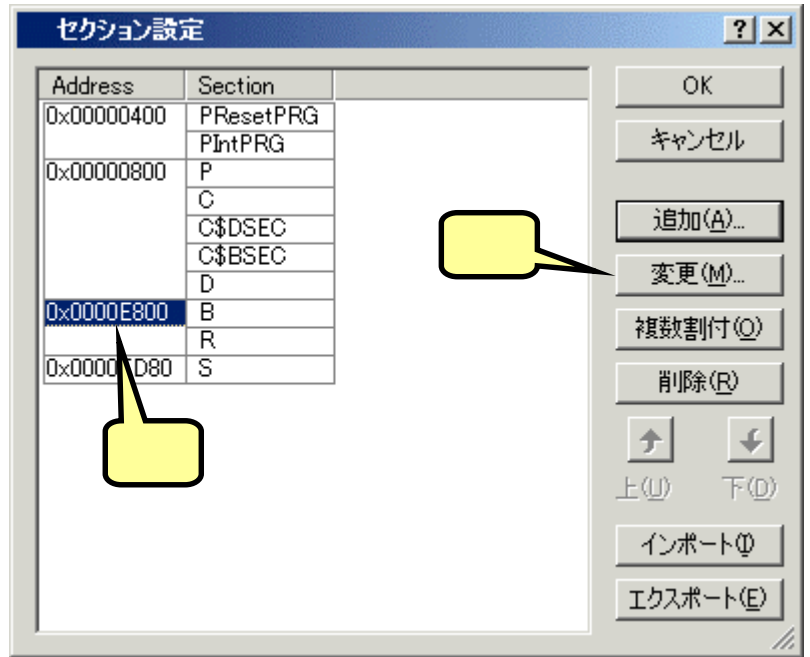
下図のように、メニューバーから「H8S,H8/300 Standard Toolchain...」を選びます。



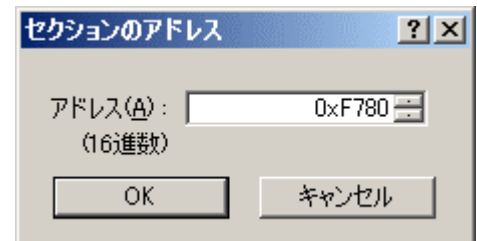
すると、「H8S, H8/300 Standard Toolchain」ウィンドウが開きます。「最適化リンク」のタブを選び、「カテゴリ (Y)」のドロップダウンメニューの中から「セクション」を選択します。すると、下図のような各セクションの先頭アドレスを設定する画面になります。「編集 (E)」ボタンをクリックしてください。



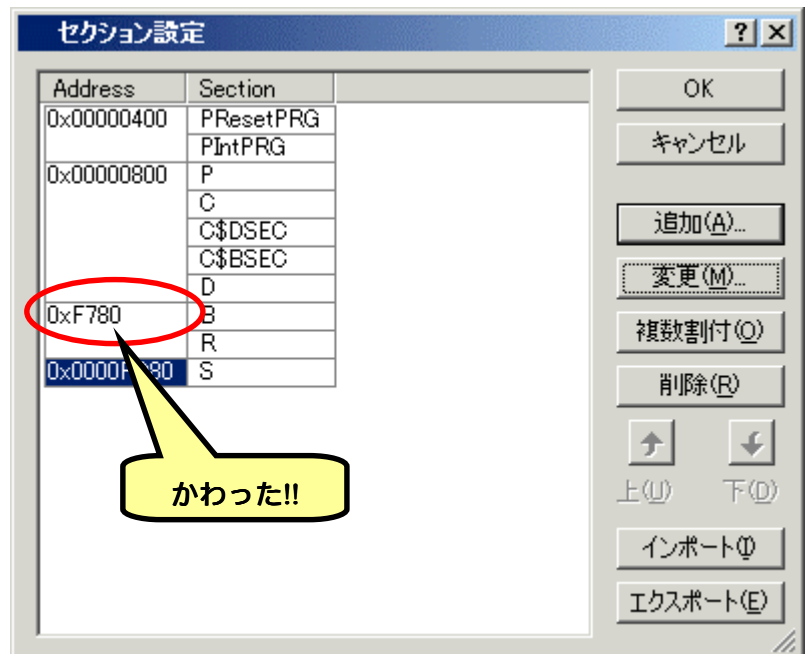
「セクション設定」ダイアログが開きます。それでは、「1. メモリマップの確認」で調べたメモリマップにあわせて設定していきましょう。最初に 'B' Section のアドレスを変更します。デフォルトでは E800 番地になっていますね。① '0x0000E800' というところをクリックして下さい。それから、②「変更(M)...」をクリックします。



そうすると、「セクションのアドレス」ダイアログが開きます。'B' Section は F780 番地から始まりますので、右のように入力して 'OK' をクリックします。



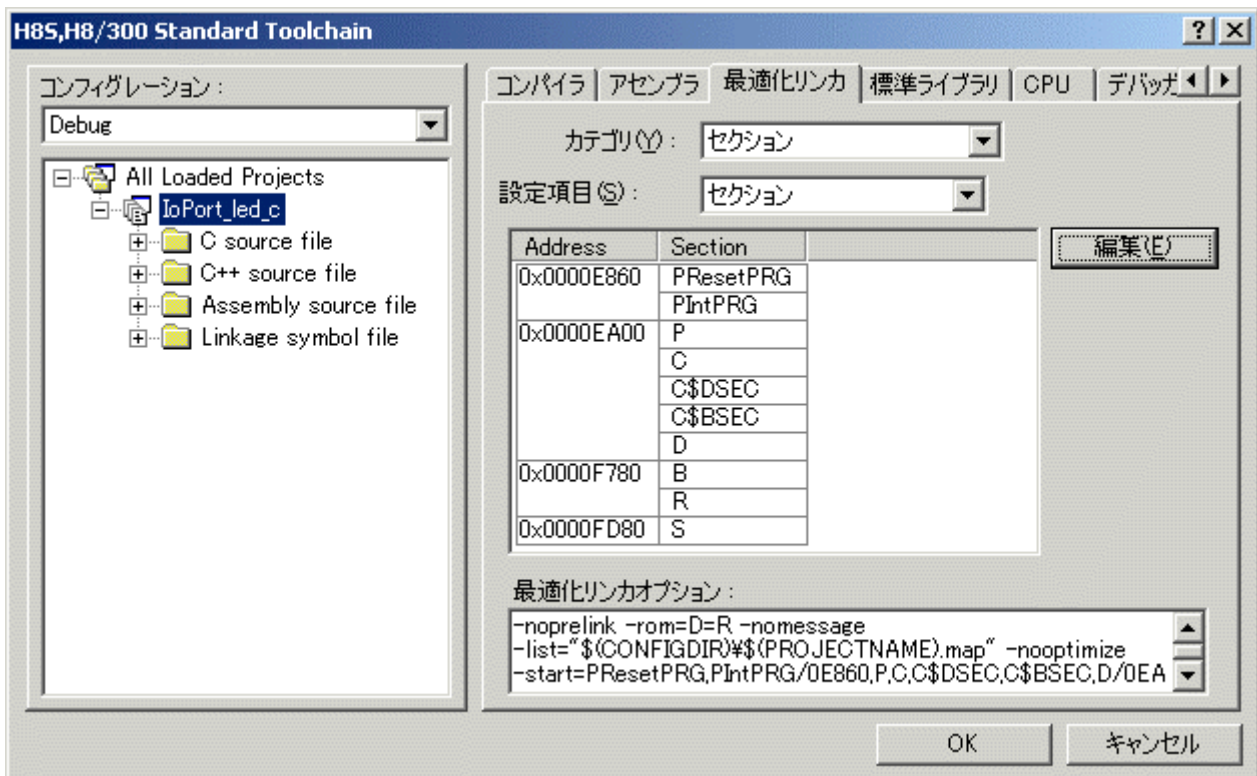
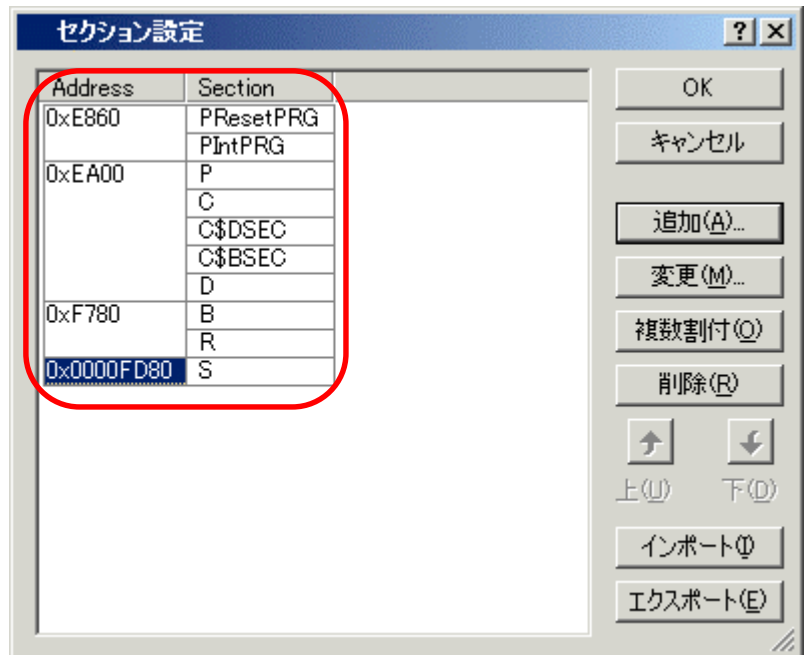
すると...



同じように、他のセクションも変更しましょう。メモリマップと同じように Section が指定されていることを確認します。ちゃんと設定されていたら「OK」をクリックします。

セクション設定の保存

今回のために今修正したセクション情報を保存することができます。下段の「エクスポート (E)」ボタンをクリックしてください。保存用のダイアログが開きますので好きな名前を付けて保存します。次回は「インポート (I)」ボタンをクリックすると保存したセクション設定を呼び出すダイアログが開きます。(おすすぬ!!)



もう一度確認してから「OK」をクリックして 'H8S, H8/300 Standard Toolchain' ウィンドウを閉じます。

HEW のワークスペースウインドウの 'IoPort_led_c.c' をダブルクリックしてください。すると、自動生成された 'IoPort_led_c.c' ファイルが開きます。

```

/*****
/*
/* FILE      :IoPort_led_c.c
/* DATE      :Wed, Apr 20, 2005
/* DESCRIPTION :Main Program
/* CPU TYPE  :H8/3687
/*
/* This file is generated by Renesas Project Generator (Ver.4.0).
/*
*****/

#ifdef __cplusplus
extern "C" {
void abort(void);
#endif
void main(void);
#ifdef __cplusplus
}
#endif

void main(void)
{
}

#ifdef __cplusplus
void abort(void)
{
}
#endif

```

```

/*****
/*
/* FILE      :IoPort_led_c.c
/* DATE      :Wed, Apr 20, 2005
/* DESCRIPTION :Main Program
/* CPU TYPE  :H8/3687
/*
/* This file is programmed by TOYO-LINX Co.,Ltd. / yKikuchi
/*
*****/

/*****
      インクルードファイル
*****/
#include <machine.h> // H8特有の命令を使う
#include "iodefine.h" // 内蔵I/Oのラベル定義

/*****
      関数の定義
*****/
void main(void);
void wait(void);

/*****
      メインプログラム
*****/
void main(void)
{
    IO.PCR6 = 0x01; // ポート6のbit0(P60)を出力に設定

    while(1){
        IO.PDR6.BIT.B0 = 0; // LEDオン
        wait();
        IO.PDR6.BIT.B0 = 1; // LEDオフ
        wait();
    }
}

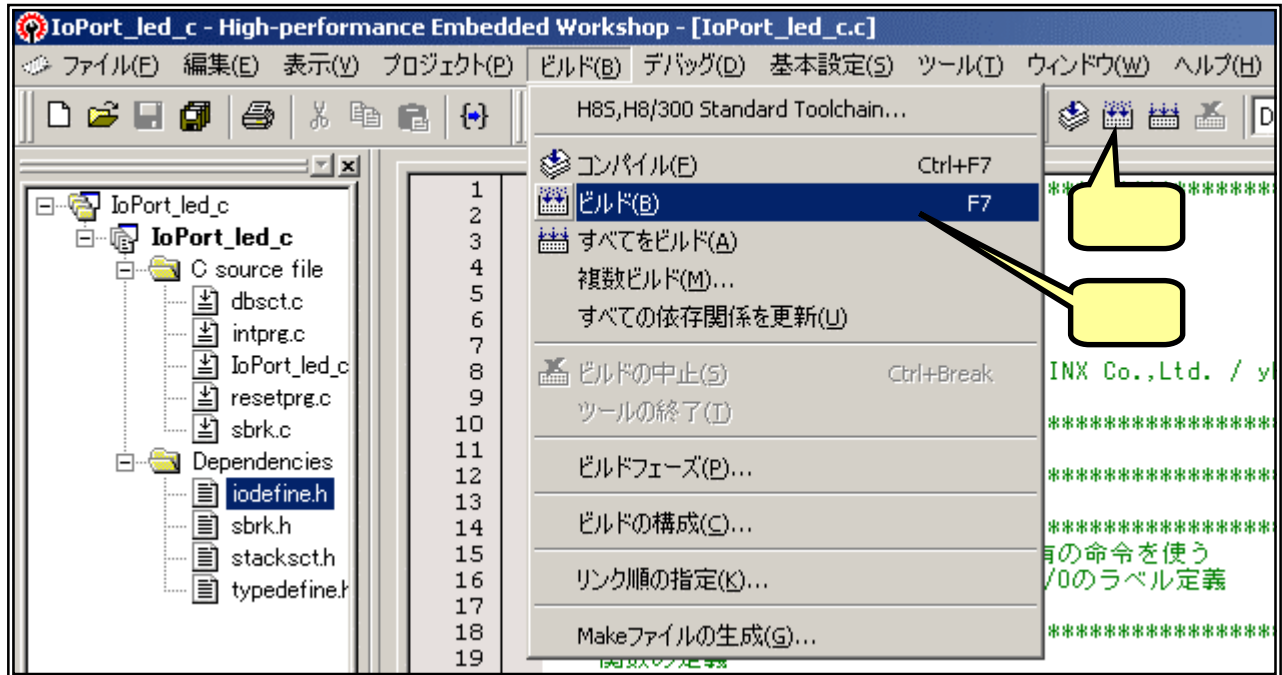
/*****
      ウェイト
*****/
void wait(void)
{
    unsigned long i;

    for (i=0;i<1666666;i++){
}

```

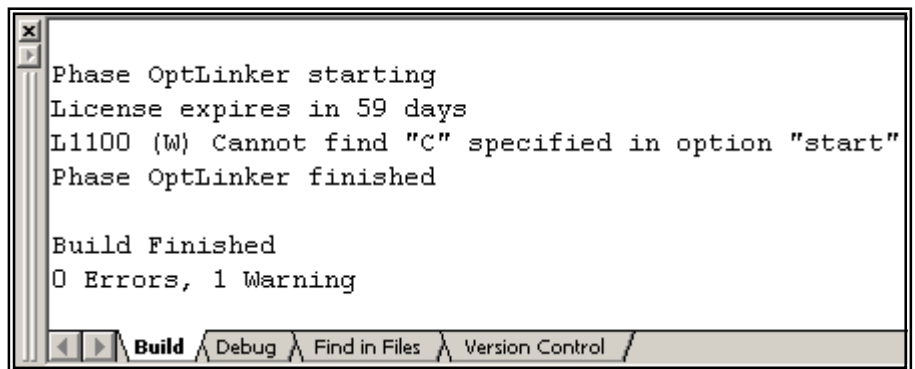
このファイルに追加・修正していきます。左のリストのとおり入力してみてください。なお、C言語の文法については、HEWをインストールしたときに一緒にコピーされる「H8S, H8/300 シリーズ C/C++コンパイラ, アセンブラ, 最適化リンケージエディタ ユーザーズマニュアル」の中で説明されています。

では、ビルドしてみましょう。ファンクションキーの[F7]を押すか、図のように①メニューバーから‘ビルド’を選ぶか、②ツールバーのビルドのアイコンをクリックして下さい。



ビルドが終了するとアウトプットウィンドウに結果が表示されます。文法上のまちがいがいかチェックされ、なければ「0 Errors」と表示されます。

エラーがある場合はソースファイルを修正します。アウトプットウィンドウのエラー項目にマウスカーソルをあててダブルクリックすると、エラー行に飛んでいきます(このあたりの機能が統合化環境の良いところですね。)ソースファイルと前のページのリストを比べてまちがいをなく入力しているかももう一度確認して下さい。



さて、図では「1 Warning」と表示されています。これは「まちがいではないかもしれないけど、念のため確認してね」という警告表示です。例えばこの図の「L1100(W) Cannot find "C" specified in option "start"」は、Cセクションを設定したのにCセクションのデータがないとき表示されます。今回のプログラムではCセクションは使っていないので、この警告が出てても何も問題ありません。

もっとも、Warningの中には動作に影響を与えるものもあります。「H8S, H8/300シリーズ C/C++コンパイラ, アセンブラ, 最適化リンケージエディタ ユーザーズマニュアル」の 539 ページからコンパイラのエラーメッセージが、621 ページから最適化リンケージエディタのエラーメッセージが載せられていますので、問題ないか必ず確認して下さい。

エラーがなければ、‘ハイパーH8’、または、エミュレータ‘E7’を使いTK-3687にファイルをダウンロード、トレース実行し、実際の動作をマイコン上で確認する作業に入ります。ここではHEWは最小化しておきましょう。動作確認して、考えたとおり動かない時は再度HEWに戻り、プログラム修正する必要があるからです。

もし、ここで作業を中断する場合はファイル保存し、終了ボタンをクリックします。次回の作業再開は作業フォルダ‘IoPort_led_c’のワークスペースファイル‘IoPort_led_c.hws’をダブルクリックすれば、HEWが立ち上がり終了時の画面が復帰します。



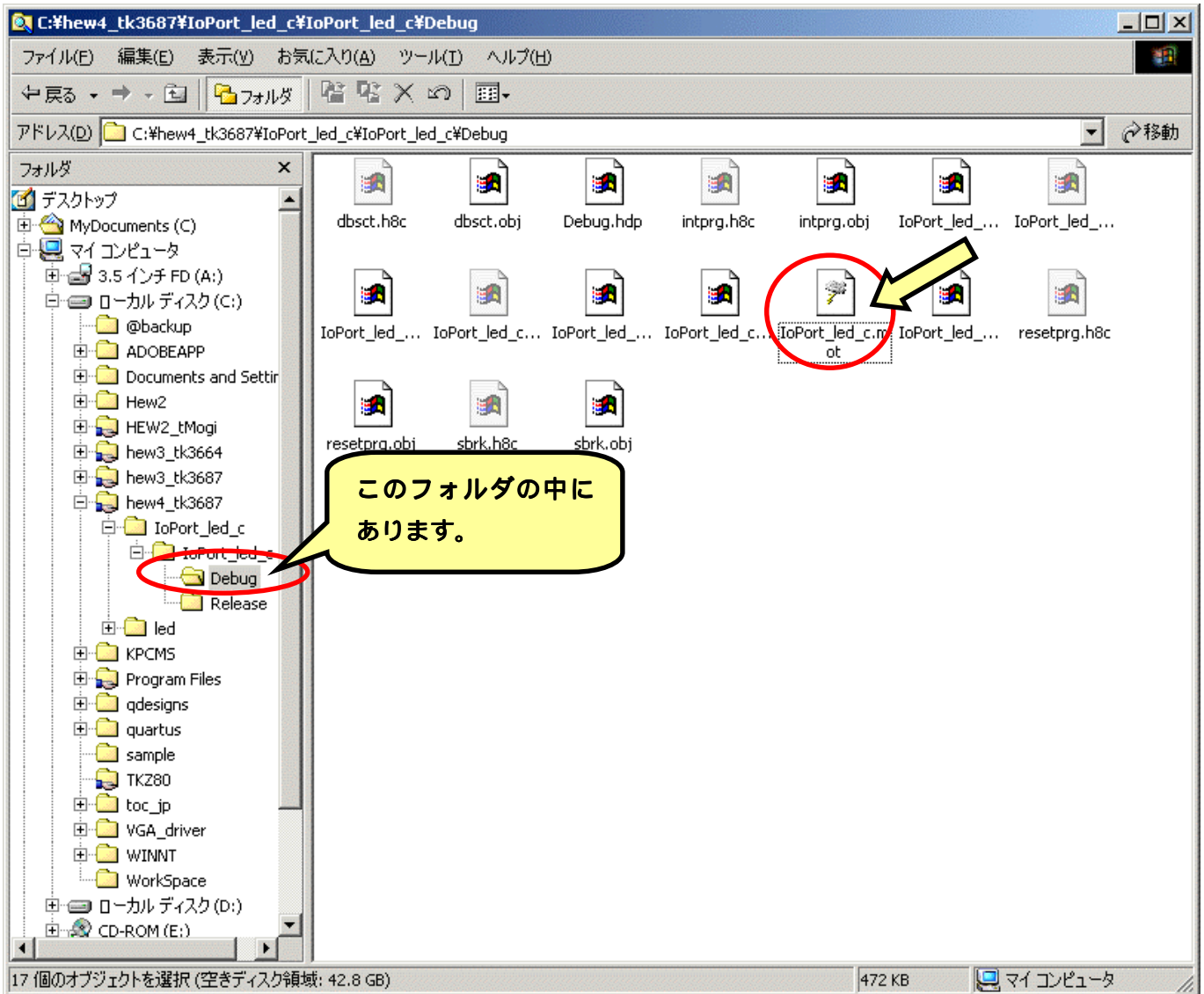
次にプログラムの確認作業に入ります。

‘ハイパーH8’をお使いの方 → 7章へお進みください。

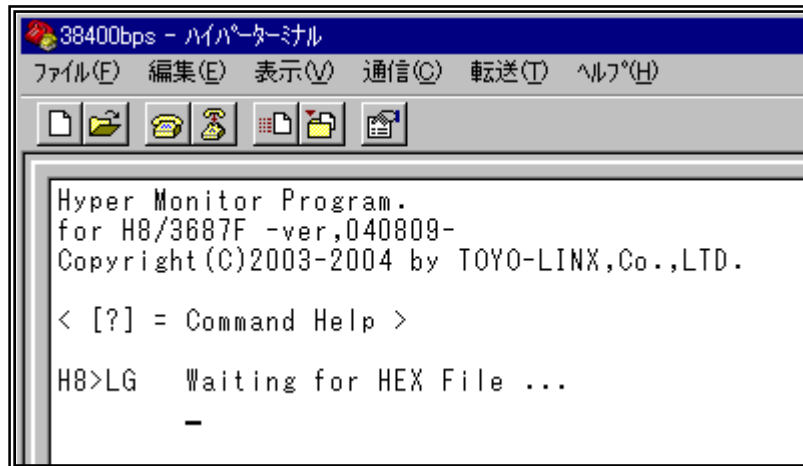
‘E8’をお使いの方 → 8章へお進みください。

7 “ハイパーH8”を使ったダウンロードと実行方法

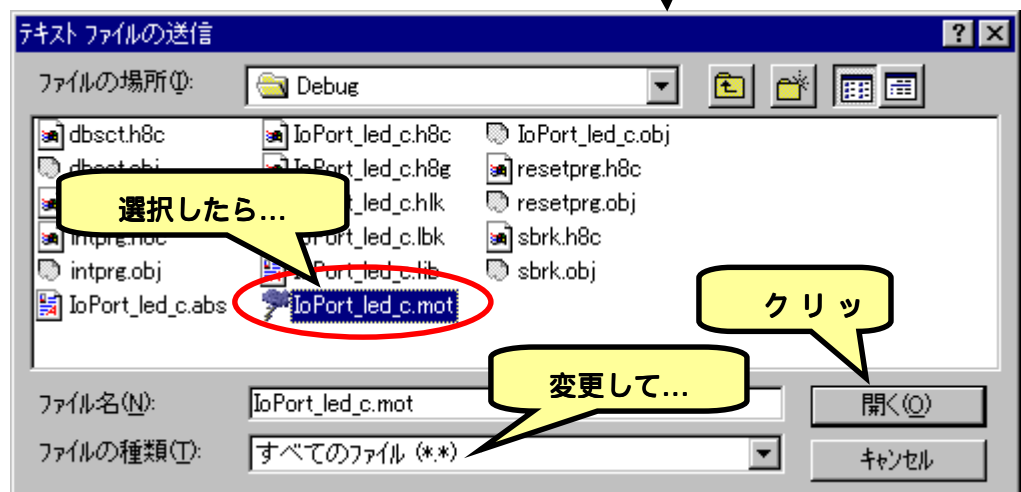
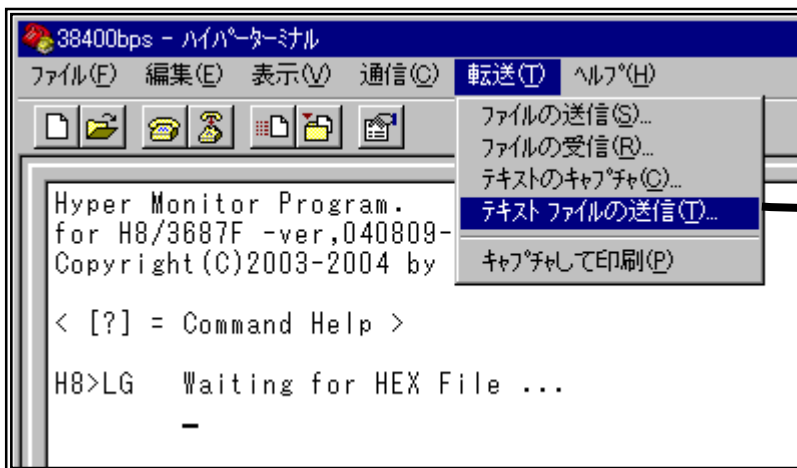
アセンブルすると‘IoPort_led_c. mot’ というファイルが作られます。拡張子が‘. mot’ のファイルは「S タイプファイル」と呼ばれていて、マシン語の情報が含まれているファイルです。このファイルは次のフォルダ内に作られません。



それでは実行してみましょう。ハイパーH8 を起動して下さい。実行の方法はアセンブラと同じです。パソコンのキーボードから 'LG' と入力して 'Enter' キーを押します。



次に、メニューの '転送(T)' から 'テキストファイルの送信(T)' を選び、「テキストファイルの送信」ウィンドウを開きます。ファイルの種類を 'すべてのファイル' にして、'IoPort_led_c. mot' を選びます。



ダウンロードが終了すると(プログラムが短いのであつという間です), 続いてロードしたプログラムを実行します。



いかがでしょうか。ちゃんとLEDは点滅しましたか。うまく動作しないときはプログラムの入力ミスの可能性が大きいかもしれません。もう一度ちゃんと入力しているか確認してみてください。

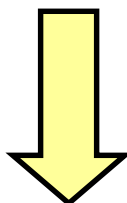


本章の最後に

以上が HEW におけるプログラム作成から動作確認までの手順です。ここでは、'IoPort_led_c' を例にし説明をしてきましたが、これで一通りプログラムの作成手順が飲み込めたのではないのでしょうか。この後は、各自で簡単なプログラムにトライして見て下さい。

C の制御文(if, for, while など)が理解できれば、第 1 ステップ合格です。また、実習・応用プログラムを 9 章以降に載せてありますので、それらを参考にすれば、入力/出力ポートの制御やより高度なプログラミング・テクニックを身に付けることが可能です。

文中では、第 2 の関門は出てきませんでしたが、第 1 の関門をクリアしダウンロード・実行して、旨く動作しないこともあります。それが第 2 の関門です。プログラムの内容にも依りますが、第 2 の関門をクリアすることが極めて大変な辛い仕事になります。そこで机上デバッグやモニタを駆使してバグを探すわけですが、一番重要なことは '何か変だ' とか '何かちょっと違う' といった些細な疑問や変化を決して見逃さない事です。それを心がければ、バグを見つけることはそう大変なことではありません。

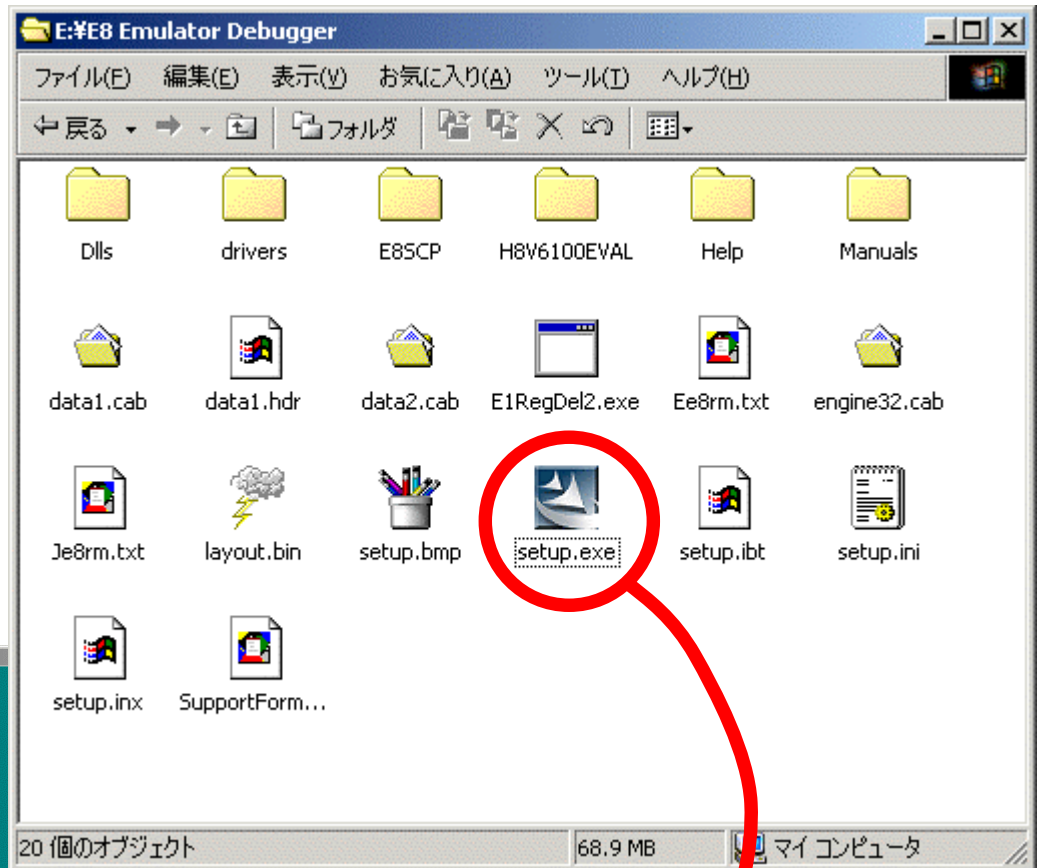


9 章にお進みください。

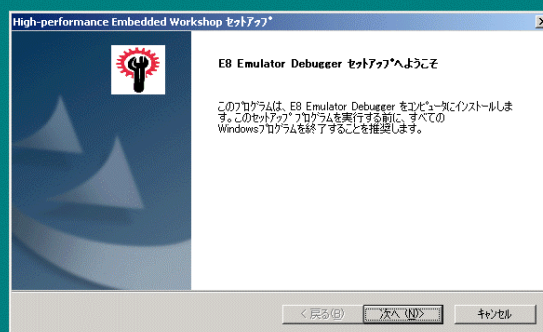
8 エミュレータ“E8”を使ったダウンロードと実行方法

それでは、作成した‘IoPort_led_c’を“E8”でダウンロード・実行してみましょう。その前に、“E8”のエミュレータソフトのインストールはお済みでしょうか。まだの方は以下の手順でインストールしてください。

“E8”の CD 中の‘setup.exe’を実行して下さい。あとは、インストールウィザードに従いインストールします。さらにここで“E8”とパソコンを接続する際の各種設定(USB ドライバ, E8 のファームウェア等)を済ませます。設定方法の詳細については“E8”のマニュアルをご覧ください。

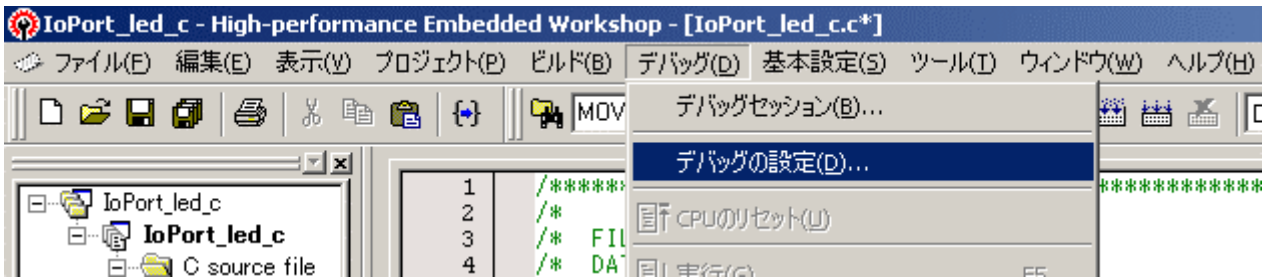


High-performance Embedded Workshop セットアップ
E8 Emulator Debugger

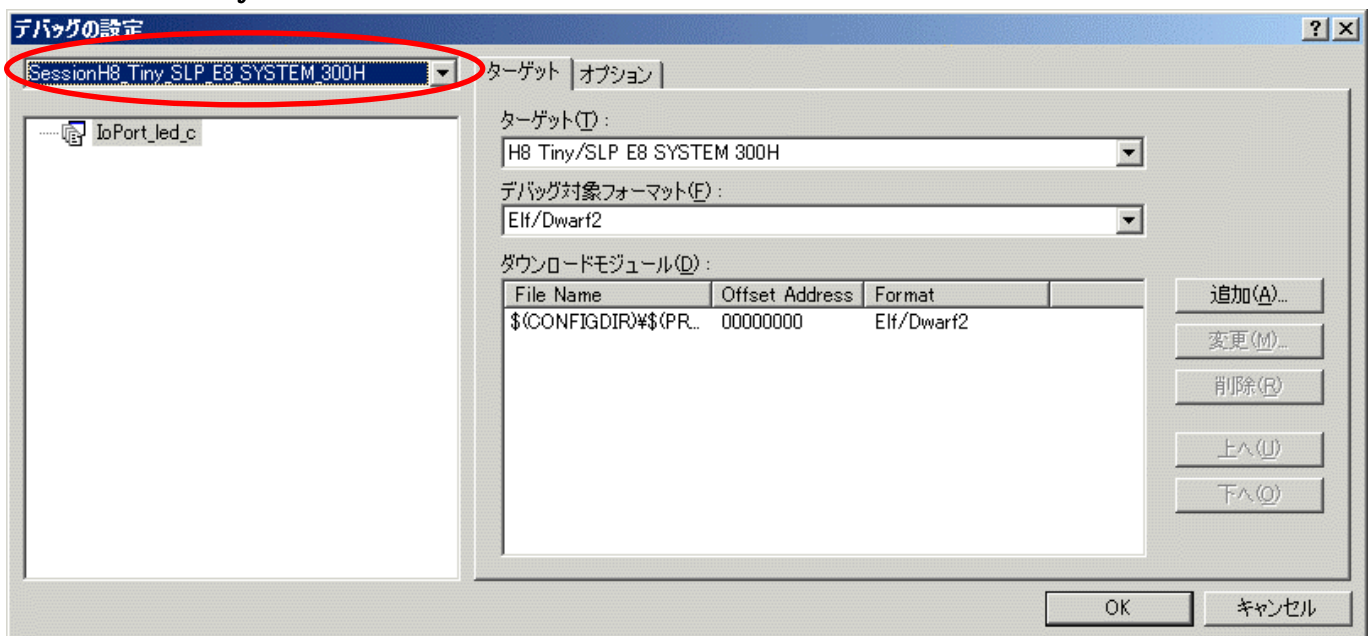


それでは、'IoPort_led_c.hws' をダブルクリックして“HEW”を起動してください。先ほど入力・コンパイルした時の画面が再現されるはずですが。

次に、“E8 エミュレータ起動時の設定”を行いません。メニューからデバッグの設定を選択して下さい。

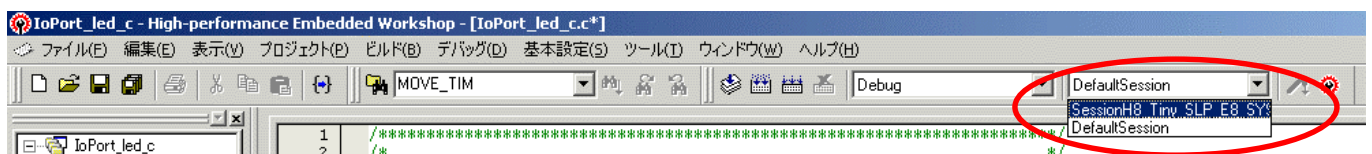


すると、'デバッグの設定' ダイアログボックスが開きます。'デバッグセッション' ドロップダウンリストボックスから 'SessionH8_Tiny_SLP_E8_SYSTEM_300H' を選びます。あとの項目は自動的に変更されます。

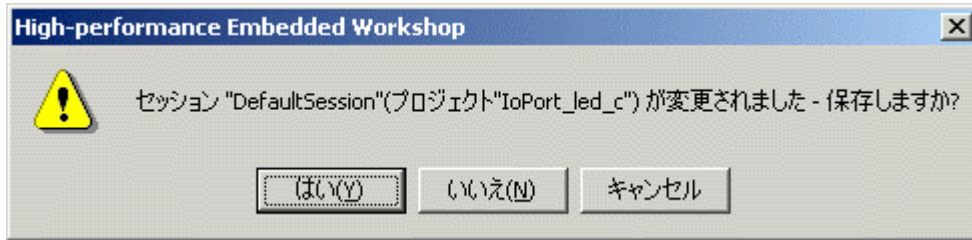
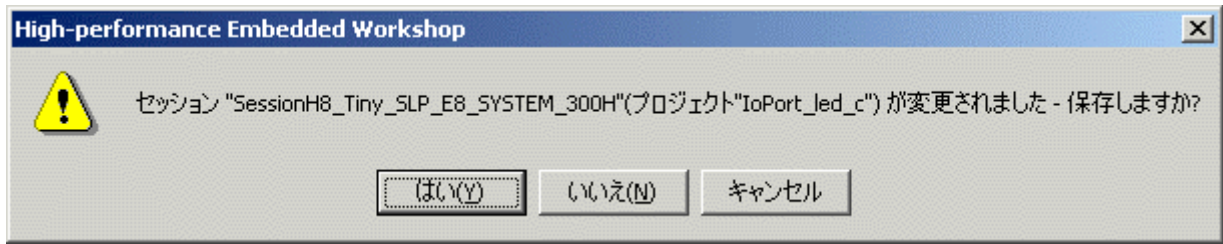


これで、設定は終わりました。次は、いよいよ“HEW”をデバッグモードにして“E8”に接続します。まず、TK-3687 の電源はオフにしてください。“E8”とパソコンを USB ケーブルで接続します。そして、“E8”と TK-3687 の CN13 (14 ピンコネクタ) を付属のケーブルで接続します。なお、この時点ではまだ TK-3687 の電源はオフのままにしておいてください。

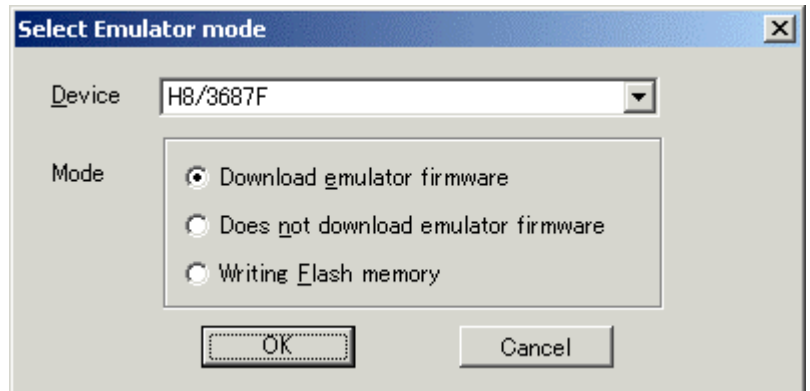
ツールバーの中、下図丸印のリストボックスから 'SessionH8_Tiny_SLP_E8_SYSTEM_300H' を選択します。そうすると、“E8”との接続が開始されます。



次のような警告が出ますが、‘はい’をクリックします。



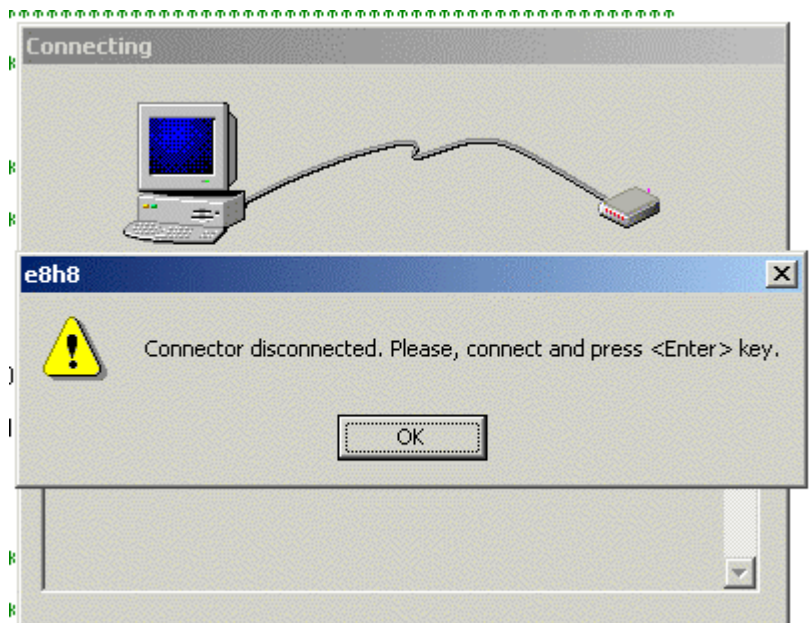
‘Select Emulator mode’ ダイアログボックスが開きます。内容が右図の通りか確認して‘OK’をクリックします。



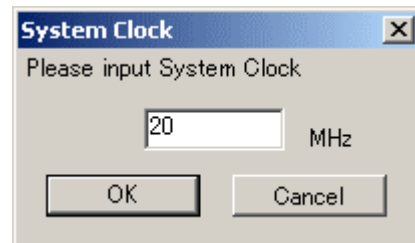
‘Power Supply’ ダイアログボックスが開いたときは右のように何もチェックをつけずに‘OK’をクリックします。



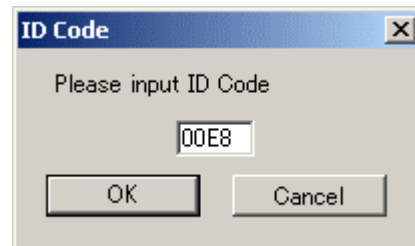
右のダイアログボックスが出たら TK-3687 の電源をオンします。それから、‘Enter’ キーを押すか、‘OK’ をクリックします。



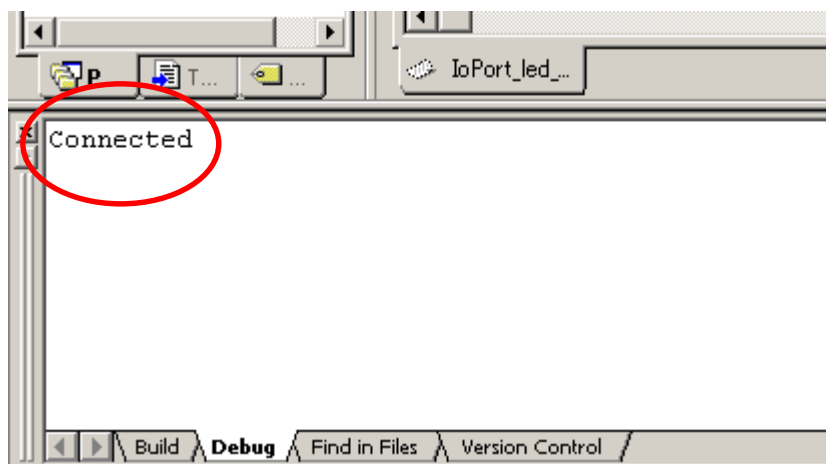
システムクロックの周波数を入力します。TK-3687 の場合は 20MHz です。
入力したら 'OK' をクリックします。



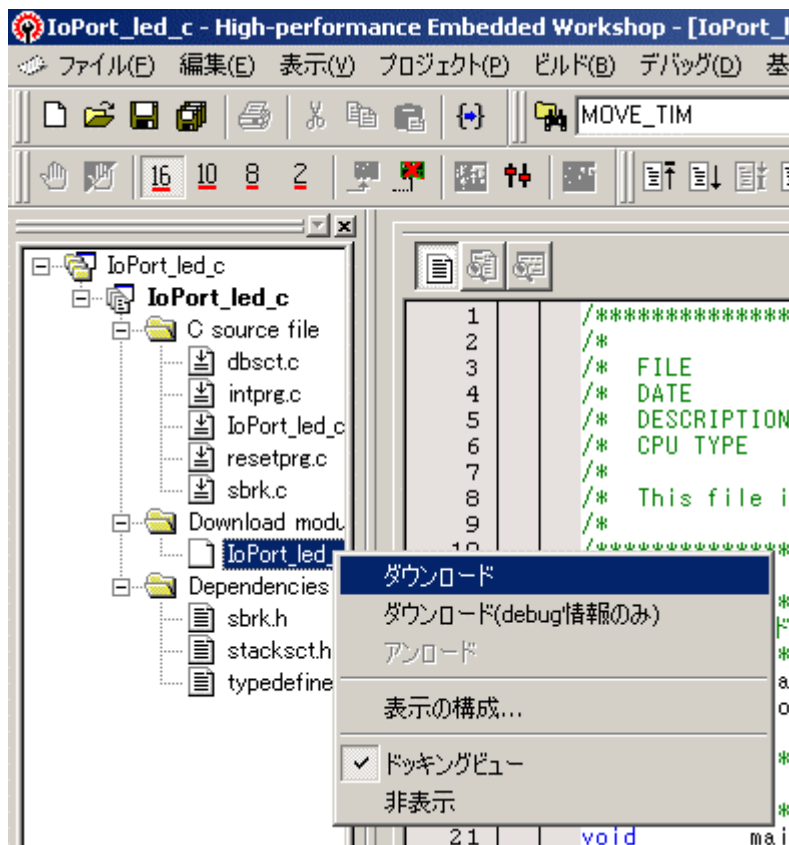
任意の ID コードを入力します。入力したら 'OK' をクリックします。



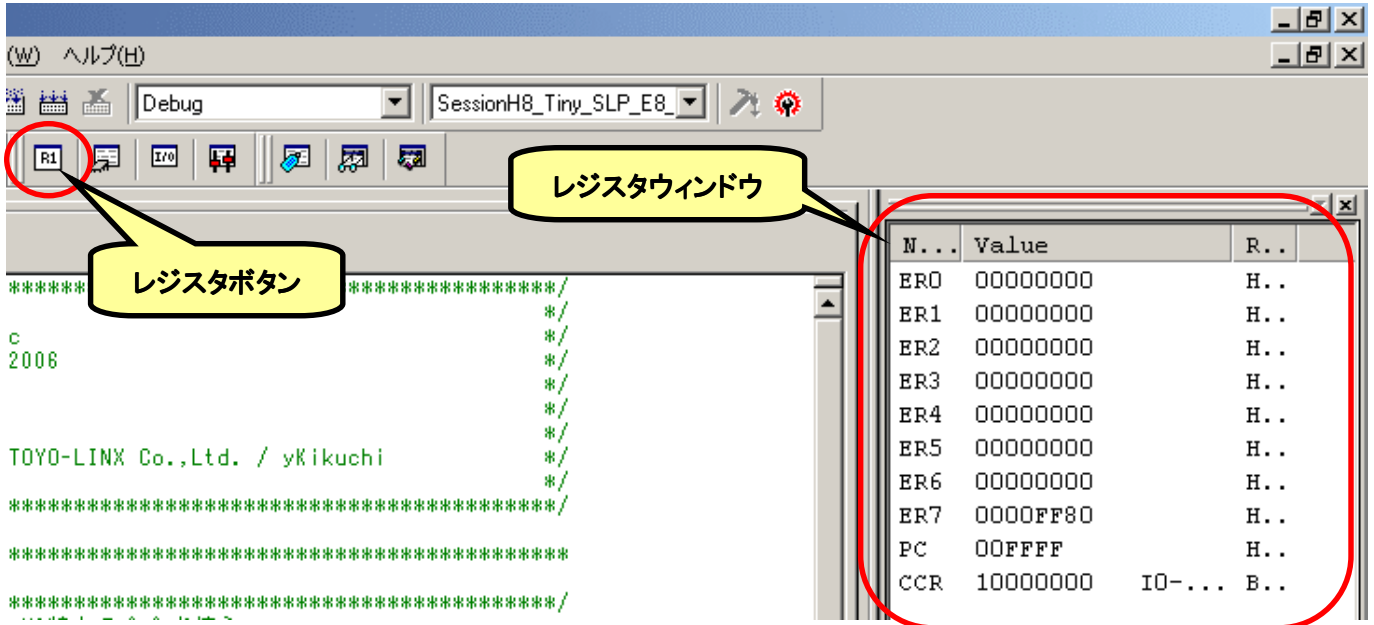
右図のように、アウトプットウィンドウに 'Connected' と表示されたら "E8" との接続完了です。



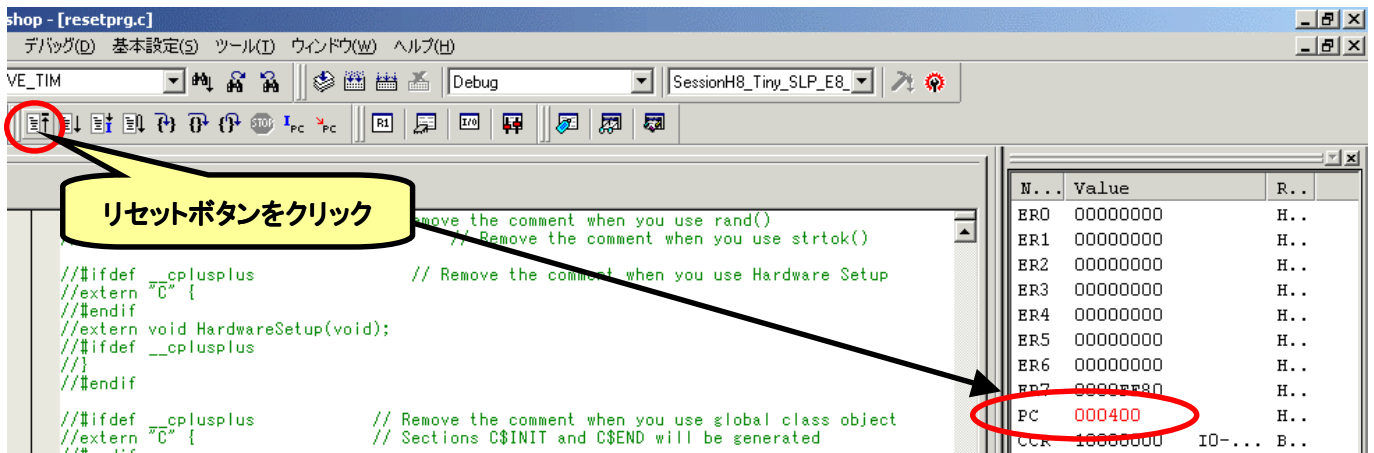
しかし、このときにはまだプログラムはダウンロードされていません。ワークスペースウィンドウの 'IoPort_led_c. abs' を右クリックしてメニューを出し、'ダウンロード' をクリックします。これで、TK-3687 に実装されている 'H8/3687' のフラッシュメモリにプログラムが書き込まれました。



続いて、プログラムを実行してみましょう。この時点ではプログラムカウンタは不定のため、あらためて指定する必要があります。ツールバーのレジスタボタンをクリックして、レジスタウィンドウを開きます。

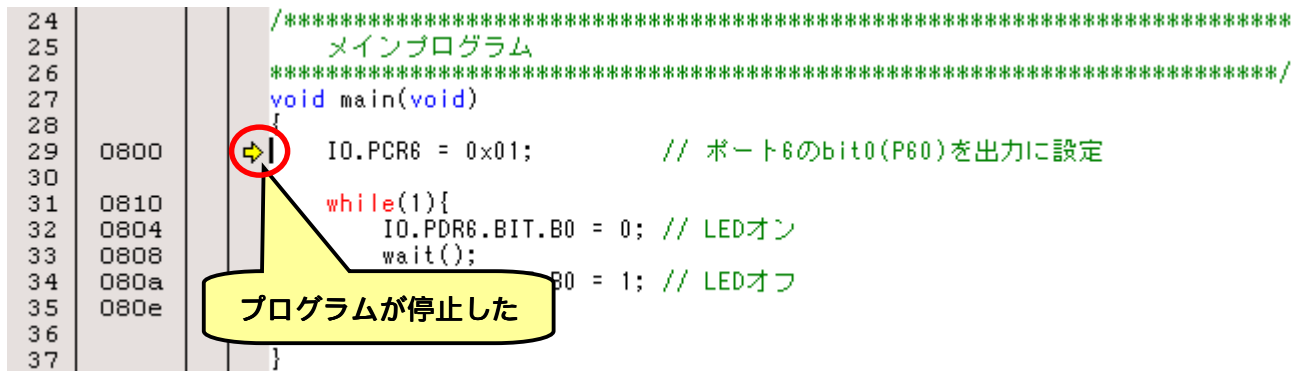
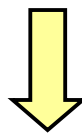
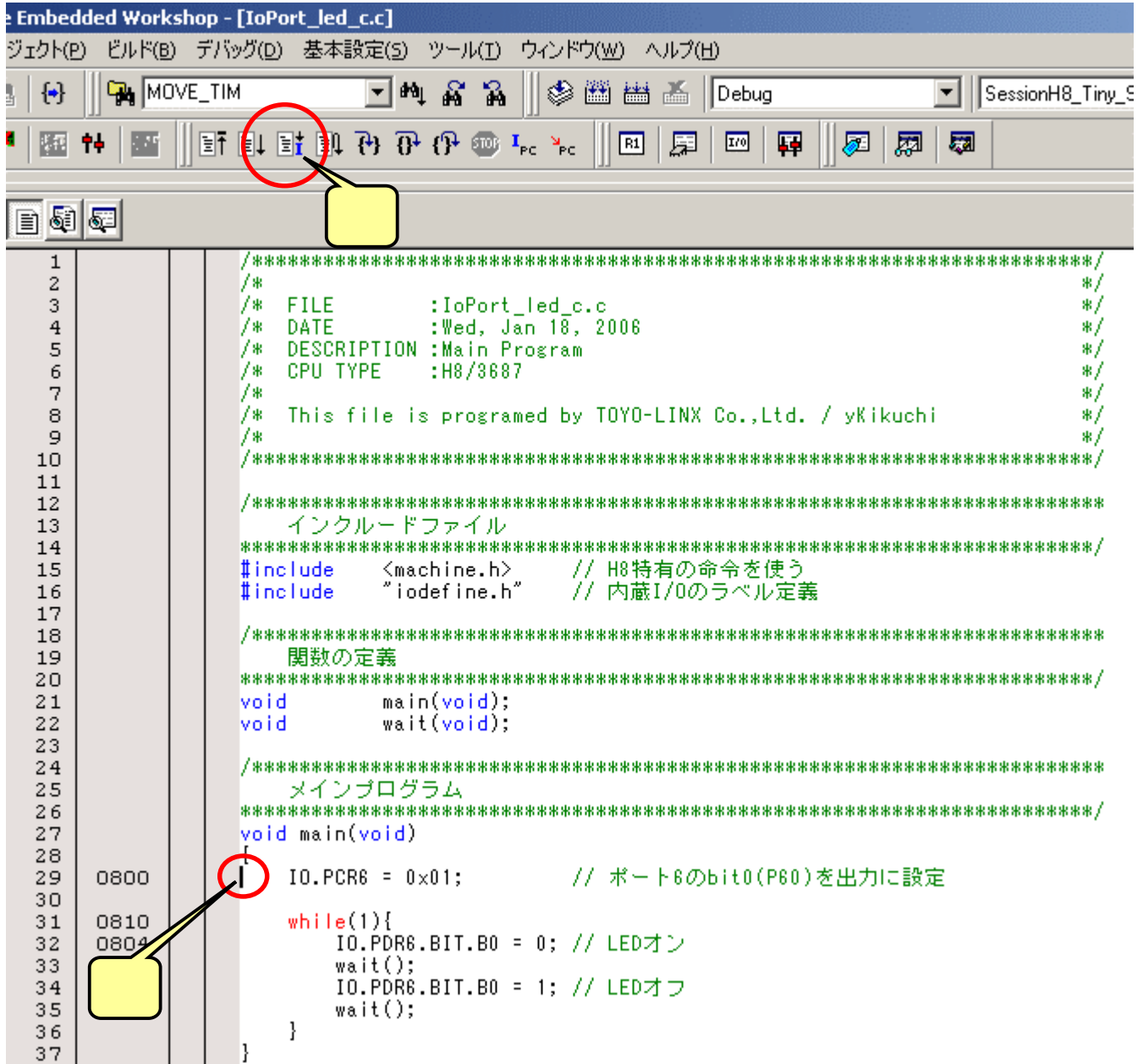


PC(プログラムカウンタ)を先頭アドレスに設定します。ツールバーのリセットボタンをクリックすると、リセットベクタアドレスの値(この場合 H'000400)がセットされます。



ここまでセットしたら、本格的なデバッグが可能になります。一例として‘カーソルまで実行’の手順を説明します。

- ①プログラムを停止したい行にテキストカーソルをおく。
- ②ツールバーの‘カーソルまで実行’ボタンをクリックする。



あとは、トレース実行で1行ずつ実行していきます。‘IO. PDR6. BIT. B0 = 0;’を実行すると、ポート6のビット0のLEDが点灯します。

“E8”はその他にも、ステップ実行、ブレークポイント設定、メモリリード・ライト、I/O リード・ライト、C のラベルによる変数のウォッチ機能など、本格的なデバッグに必要な機能を十分に備えています。詳細については、“E8”のユーザーズマニュアルをご覧ください。

“E8”でダウンロードした時点で‘H8/3687’のフラッシュメモリにプログラムが書き込まれていますので、“E8”を接続しないでTK-3687の電源をオンにすると書き込まれたプログラムを実行します。

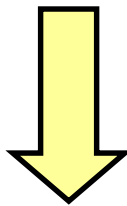


本章の最後に

以上が HEW におけるプログラム作成から動作確認までの手順です。ここでは、‘IoPort_led_c’を例にし説明をして来ましたが、これで一通りプログラムの作成手順が飲み込めたのではないのでしょうか。この後は、各自で簡単なプログラムにトライして見て下さい。

C の制御文(if, for, while など)が理解できれば、第1ステップ合格です。また、実習・応用プログラムを9章以降に載せてありますので、それらを参考にすれば、入力/出力ポートの制御やより高度なプログラミング・テクニックを身に付けることが可能です。

文中では、第2の関門は出てきませんでした。第1の関門をクリアしダウンロード・実行して、旨く動作しないこともあります。それが第2の関門です。プログラムの内容にも依りますが、第2の関門をクリアすることが極めて大変な辛い仕事になります。そこで机上デバッグやモニタを駆使してバグを探すわけですが、一番重要なことは‘何か変だ’とか‘何かちょっと違う’といった些細な疑問や変化を決して見逃さない事です。それを心がければ、バグを見つけることはそう大変なことではありません。



次章にお進みください。

9 実習プログラム

この章では C によるプログラミングに慣れることを目標に、実習プログラムを作ります。各プログラムにはそれぞれ幾つかの課題がありますので、リストのプログラムが完成したら、課題に挑戦してみてください。これで、プログラミングの基礎力が付きます。ここでしっかり基礎を作れば、独自にプログラムを作ることもできるはずですし、次章の応用プログラムにも入って行けます。なお、次章の応用プログラムではハードの追加が要りますが、本章では一切必要ありません。

実習に入る前に一般的なプログラムの構造を表に示します。サブルーチン領域が無かったり、どの実習プログラムでも当てはまるわけではありませんが、概ね以下のようなブロックに分かれていることを把握して、リストと照らし合わせてみてください。

領域	内容
タイトル欄	HEW が自動作成するタイトル; プログラム名, 作成日, CPU 名等 プログラムの所属, 名前 プログラムの説明やデータの割り振り 修正履歴など
インクルード宣言	インクルードファイル(ライブラリ等)の宣言
外部結合宣言	外部結合する関数を宣言する
アドレス, 変数定義	I/O アドレスや変数の定義をする
関数宣言	関数の戻り値の型や引数の型を宣言する
メ イ ン 関 数	初期化プログラム I/O の入出力などの設定を行なう 変数の初期値の設定を行なう
	メインプログラム ループさせ、ループの中で色々な処理を行なう (このループを通常メインループと呼ぶ)
関数	メインプログラムで呼ばれる関数(サブルーチン)はこの場所にまとめる

実習プログラムは3つあり、プログラムリストを以降に示します。まずはリストの通りに入力し実行して見ましょう。但し、タイトルやコメント(//以降, または/*...*/には含まれた部分)は適当に省略して構いません。

```

/*****/
/*                                          */
/* FILE      :samp_02.c                    */
/* DATE      :Fri, Oct 03, 2003           */
/* DESCRIPTION :Main Program              */
/* CPU TYPE   :H8/3687                    */
/*                                          */
/* This file is programmed by TOYO-LINUX Co.,Ltd. / yKikuchi. */
/*                                          */
/* ----- */
/*   samp_01の復習                          */
/*   変数にデータをセット                    */
/*   56h Data0, 78h Data1, 34h Data2        */
/*   加算, 減算した結果を変数にセットする  */
/*   56h+78h      Data4                      */
/*   56h+78h-34h  Data5                      */
/*   演算結果をポート5へLED表示する        */
/* ----- */
/*   課題に挑戦!                            */
/*   課題1. 加算をAND, 減算をORに置き換える */
/*   課題2. 加算をXOR, 減算をANDに置き換える */
/*   課題3.                                  */
/* ----- */
/*****/

/*****
      インクルードファイル
*****/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"  // 内蔵I/Oのラベル定義

/*****
      定数の定義
*****/
#define WAIT_TIM 0x140000 // wait()のループ回数

/*****
      グローバル変数の定義
*****/
unsigned char Data0;
unsigned char Data1;
unsigned char Data2;
unsigned char Data3;
unsigned char Data4;
unsigned char Data5;

/*****
      関数の定義
*****/
void          main(void);

```



```

void          wait(void);

/*****
   メインプログラム
 *****/
void main(void)
{
    IO.PCR5 = 0xff; // PORT5を出力に設定

    Data0 = 0x56;   // 56hをData0にセット
    Data1 = 0x78;   // 78hをData1にセット
    Data2 = 0x34;   // 34hをData2にセット

    while(1){
        Data4 = Data0 + Data1;
        IO.PDR5.BYTE = Data4;      // PORT5に出力
        wait();

        Data5 = Data0 + Data1 - Data2;
        IO.PDR5.BYTE = Data5;      // PORT5に出力
        wait();
    }
}

/*****
   ウェイト
 *****/
void wait(void)
{
    unsigned long   i;

    for (i=0;i<WAIT_TIM;i++){ // i=WAIT_TIMまでループする
}

```

```

/*****/
/*                                     */
/* FILE      :samp_03.c                */
/* DATE      :Fri, Oct 03, 2003        */
/* DESCRIPTION :Main Program           */
/* CPU TYPE   :H8/3687                 */
/*                                     */
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi. */
/*                                     */
/* ----- */
/*   ポート5の一つのLEDを1秒間隔で点滅させる */
/*   1秒のウェイトはサブルーチンとして作成する */
/* ----- */
/*   課題に挑戦! */
/*   課題1. LEDのON/OFF時間を変更する */
/*   課題2. 出力するLEDをP51~P57のいずれかに変更する */
/*   課題3. ポート5以外のポート1またはポート7に変更する */
/*   課題4. 表示のパターンを変更する(例えば55hなど) */
/*   課題5. */
/* ----- */
/*****/

/*****
    インクルードファイル
*****/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"  // 内蔵I/Oのラベル定義

/*****
    定数の定義
*****/
#define WAIT_TIM    0x140000    // wait()のループ回数

/*****
    関数の定義
*****/
void    main(void);
void    wait(void);

/*****
    メインプログラム
*****/
void main(void)
{
// ----- イニシャライズ -----

    IO.PCR5      = 0xff;    // PORT5を出力にする
    IO.PDR5.BYTE = 0x00;    // PORT5に00hを出力する(初期出力)

// ----- メインループ -----

```

```

while(1){
    IO.PDR5.BYTE = IO.PDR5.BYTE | 0x01; // PORT5,bit0 = 1
    wait();
    IO.PDR5.BYTE = IO.PDR5.BYTE & 0xfe; // PORT5,bit0 = 0
    wait();
}
}

/*****
ウェイト
*****/
void wait(void)
{
    unsigned long    i;

    for (i=0;i<WAIT_TIM;i++){ // i=WAIT_TIMまでループする
}

```

9-3. “samp_04”; ローテーション命令でLEDを循環させる(電飾イメージの作成)

```

/*****
/*
/* FILE      :samp_04.c
/* DATE      :Fri, Oct 03, 2003
/* DESCRIPTION :Main Program
/* CPU TYPE  :H8/3687
/*
/* -----
/*   回転命令(ローテート)の演習
/*   PDR1,5を使って,電飾をイメージさせる
/*   Cにはローテート命令がないので,H8特有の命令を関数として組み込む
/*   'machine.h'の組み込み
/*
/* -----
/*   課題に挑戦!
/*   課題1.パターンの変更
/*   課題2.逆回転
/*   課題3.全消灯,全点灯の間欠動作
/*   課題4.PDR7の追加
/*   課題5.
/*
*****/

/*****
インクルードファイル
*****/
#include <machine.h> // H8特有の命令を使う
#include "iodefine.h" // 内蔵I/Oのラベル定義

```

```

/*****
  定数の定義
*****/
#define    WAIT_TIM    0x30000 // wait()のループ回数

/*****
  グローバル変数の定義
*****/
unsigned int    DispData = 0x0003; // LED出力データ

/*****
  関数の定義
*****/
void            main(void);
void            wait(void);

/*****
  メインプログラム
*****/
void main(void)
{
// ----- イニシャライズ -----

    IO.PCR1 = 0xff; // PORT1を出力に設定
    IO.PCR5 = 0xff; // PORT5を出力に設定

// ----- メインループ -----

    while(1){
        DispData = rotll(1,DispData); // 1ビット左ローテート
        IO.PDR1.BYTE = (unsigned char)(DispData & 0x00ff);
        IO.PDR5.BYTE = (unsigned char)(DispData / 0x0100);
        wait();
    }
}

/*****
  ウェイト
*****/
void wait(void)
{
    unsigned long    i;

    for (i=0;i<WAIT_TIM;i++){ // i=WAIT_TIMまでループする
}

```

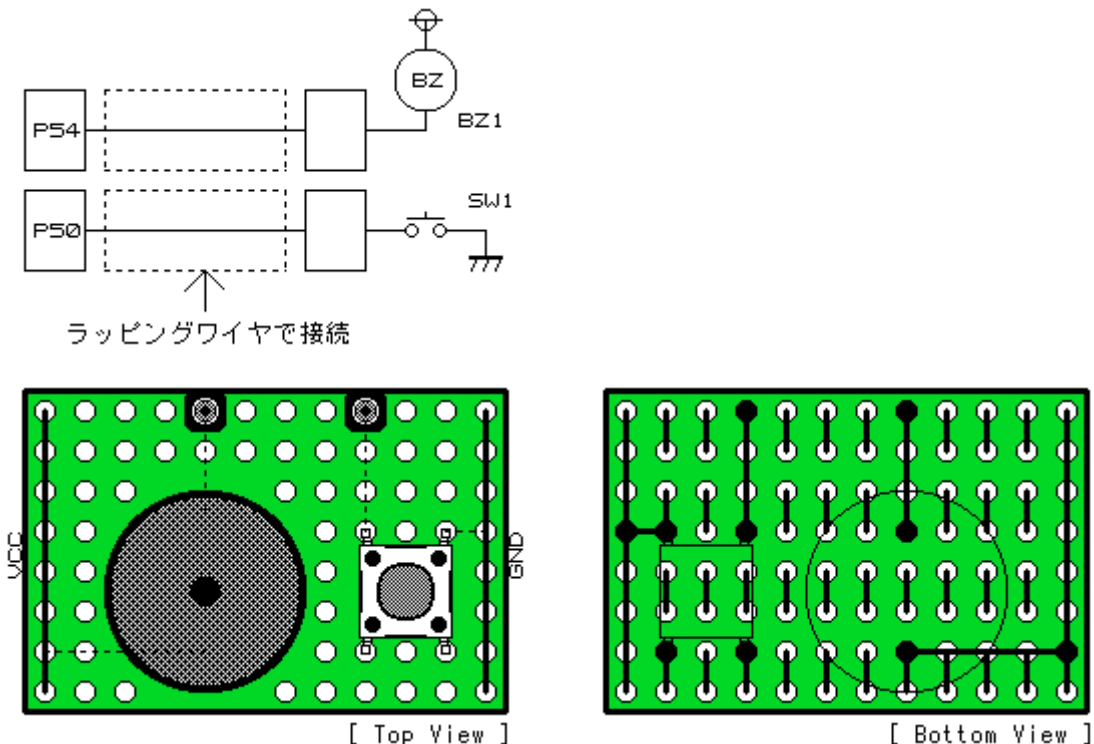
10 応用プログラム

10-1 スイッチとブザーのワンショット動作

スイッチを押した時のみブザーを鳴らすワンショット動作をプログラムしてみましょう。ワンショット動作ですので押し続けてもブザーが鳴るのは押した瞬間のみです。

■ハードの組み立て

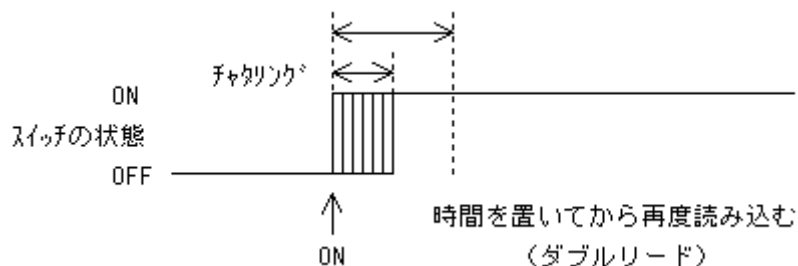
まずはブザーとスイッチを TK-3687 上のユニバーサルエリアに実装します。必要な工具はハンダごて、ハンダ、ニッパ、ワイヤストリッパです。下図 10-1-1 の回路図と実装図に従って実装して下さい。また、P5 のテストスルー (RA2 隣の 8 個のスルー) にも丸ピンソケットをハンダ付けします。ハンダ付けが終了したらラッピングワイヤでポートと接続して下さい。接続先はスイッチが P50、ブザーが P54 です。



<図 10-1-1 回路図と実装図>

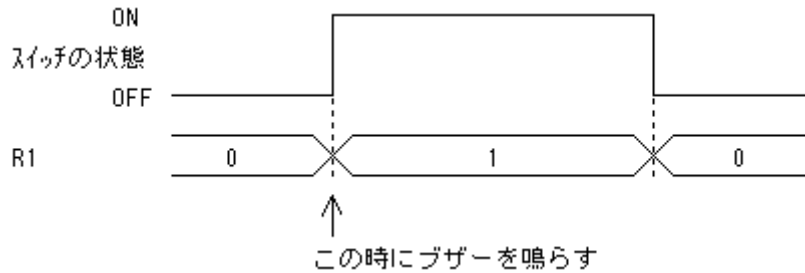
■プログラム

ハードが組みあがったら次にソフトを考えます。スイッチをポートで読む時にまず考えなくてはならないのはチャタリングの除去です。チャタリングとは機械式スイッチを押した時に接点がバウンドして ON/OFF が繰り返される現象です。単純にポートを読むだけですとバウンドの ON/OFF を読み取ってしまいます。チャタリングを回避する為に一定時間置いてから再度読む 2 度読み (ダブルリード) を行います。



<図 10-1-2 チャタリングとダブルリード>

次にワンショット動作を考えます。スイッチが ON になった瞬間のみを検出する為にスイッチの状態を変数に保持しておきます。ここでは 'SwStatus' をスイッチの状態として使用する事にします。'SwStatus=0' ならスイッチは押されていない, 'SwStatus=1' ならスイッチが押されている, と条件付けます。スイッチを押した瞬間を検出したいので 'SwStatus' が 0 から 1 へ変化した時のみブザーを鳴らします。



＜図 10-1-3 スwitchの状態とレジスタ R1＞

以上 2 点を考慮して作成したプログラムのリストとタイミングチャートを掲載します。ダブルリードの時間は 10msec としました。また、ブザーの鳴動時間は 100msec です。

```

/*****
    インクルードファイル
    *****/
#include <machine.h> // H8特有の命令を使う
#include "iodefine.h" // 内蔵I/Oのラベル定義

/*****
    定数の定義
    *****/
#define WAIT10_TIM 0x8235 // wait()のループ回数
#define WAIT100_TIM 0x51612 // wait()のループ回数

/*****
    グローバル変数の定義
    *****/
unsigned char SwStatus = 0;

/*****
    関数の定義
    *****/
void main(void);
void wait10m(void);
void wait100m(void);

/*****
    メインプログラム
    *****/
void main(void)
{
// ----- イニシャライズ -----

    IO.PCR5 = 0xf0; // P50~51入力, P54~57出力
    IO.PDR5.BYTE = 0x10; // ブザーオフ

```

```
// ----- メインループ -----

while(1){
  if (IO.PDR5.BIT.B0==1){ // スイッチオフ
    SwStatus = 0;
  }
  else{ // スイッチオン
    wait10m(); // チャタリング除去
    if (IO.PDR5.BIT.B0==1){ // ダブルリード, スイッチオフ
      SwStatus=0;
    }
    else{ // ダブルリード, スイッチオン
      if (SwStatus==0){ // 今まで押されていなかった
        SwStatus = 1;
        IO.PDR5.BIT.B4 = 0; // ブザーオン
        wait100m(); // 鳴動時間
        IO.PDR5.BIT.B4 = 1; // ブザーオフ
      }
    }
  }
}

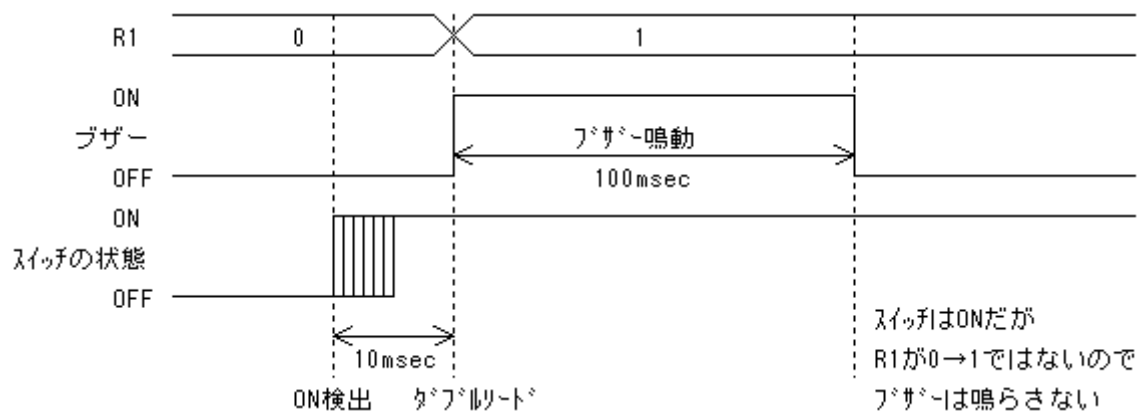
/*****
ウェイト
*****/
void wait10m(void)
{
  unsigned long i;

  for (i=0;i<WAIT10_TIM;i++){ // i=WAIT10_TIMまでループする
  }

void wait100m(void)
{
  unsigned long i;

  for (i=0;i<WAIT100_TIM;i++){ // i=WAIT100_TIMまでループする
  }
}

```

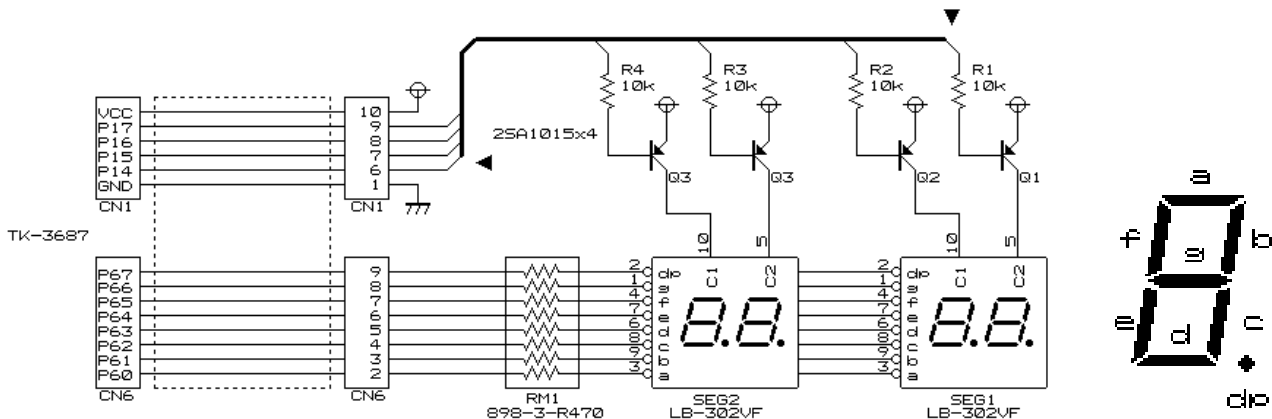


<図 10-1-4 タイミングチャート>

10-2 ダイナミック表示

ここではダイナミック表示の応用プログラムを紹介します。部品は付属していませんので回路図を見て揃えてください。尚、オプションとして部品一式・ドキュメントをセットにした“7セグメントLED表示&キー入力キット:B6086”を用意しています。

図 10-2-1 はダイナミック表示の回路図とセグメントの割付けです。7セグメントLEDの各セグメントとアルファベットとの割付けは図のようになっています。この回路ではアノードコモンLEDを使っているため共通端子が“H”，a～g及びdpの端子が“L”になった時対応するセグメントが点灯するようになっています。例えば“1”という文字を点灯させる時はb, cを“L”，共通端子を“H”にします。



<図 10-2-1 表示部回路図>

P14～17の信号を“L”にするとトランジスタがONの状態になり共通端子は“H”になります。その時P60～67の方は点灯させたいビットを“L”にするとLEDに電流が流れて点灯します。まずは、セグメントの下1桁に“4”を表示させるプログラムを考えてみましょう。<Dscan_1>

```

/*****
    インクルードファイル
    *****/
#include <machine.h> // H8特有の命令を使う
#include "iodefine.h" // 内蔵I/Oのラベル定義

/*****
    関数の定義
    *****/

void main(void);
void init_port(void);

/*****
    メインプログラム
    *****/

void main(void)
{
// ----- インシャライズ -----
    init_port(); // I/Oポートイニシャライズ

// ----- メインループ -----

```



```

while(1){
    IO.PDR1.BYTE = 0xee;    // スキャン出力
    IO.PDR6.BYTE = ~(0x66); // セグメントデータ4を出力
}
}

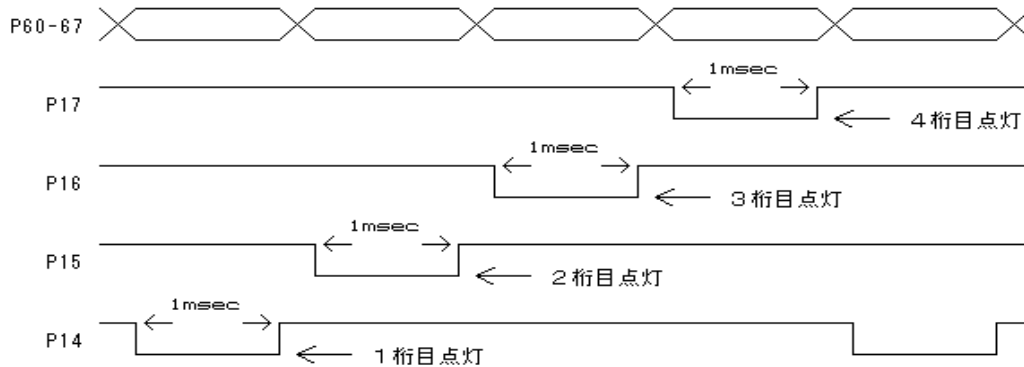
/*****
    I/Oポート イニシャライズ
*****/
void init_port(void)
{
    IO.PCR1      = 0xf0;    // P10~13入力, P14~17出力
    IO.PDR1.BYTE = 0xf0;    // P14~17初期出力
    IO.PCR6      = 0xff;    // P60~67出力
    IO.PDR6.BYTE = 0xf0;    // P60~67初期出力
}

```

<Dscan_1>

1桁の表示ができたら次に4桁を時分割により表示させるプログラムを考えましょう。

ある一つの桁を表示している間は他の桁を消灯し、順に4桁繰り返し行います。タイミングチャートで表すと図11-2-2のようになります。



<図 10-2-2 ダイナミック表示タイミングチャート>

図 10-2-2 のタイミングチャートをもとに“1234”を表示するようなプログラムを作ってみます。<Dscan_2>

```

/*****
    インクルードファイル
*****/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"  // 内蔵I/Oのラベル定義

/*****
    定数の定義
*****/
#define TIMER1_TIM 0x0d02 // timer1m()のループ回数

/*****
    関数の定義
*****/
void main(void);

```

```

void          init_port(void);
void          timer1m(void);

/*****
    メインプログラム
*****/
void main(void)
{
// ----- イニシャライズ -----
    init_port();    // I/Oポートイニシャライズ

// ----- メインループ -----
    while(1){
        IO.PDR1.BYTE = 0xee;    // スキャン出力(P14=Low)
        IO.PDR6.BYTE = ~(0x66); // セグメントデータ4を出力
        timer1m();
        IO.PDR1.BYTE = 0xdd;    // スキャン出力(P15=Low)
        IO.PDR6.BYTE = ~(0x4f); // セグメントデータ3を出力
        timer1m();
        IO.PDR1.BYTE = 0xbb;    // スキャン出力(P16=Low)
        IO.PDR6.BYTE = ~(0x5b); // セグメントデータ2を出力
        timer1m();
        IO.PDR1.BYTE = 0x77;    // スキャン出力(P17=Low)
        IO.PDR6.BYTE = ~(0x06); // セグメントデータ1を出力
        timer1m();
    }
}

/*****
    I/Oポート イニシャライズ
*****/
void init_port(void)
{
    IO.PCR1      = 0xf0;    // P10~13入力, P14~17出力
    IO.PDR1.BYTE = 0xf0;    // P14~17初期出力
    IO.PCR6      = 0xff;    // P60~67出力
    IO.PDR6.BYTE = 0xf0;    // P60~67初期出力
}

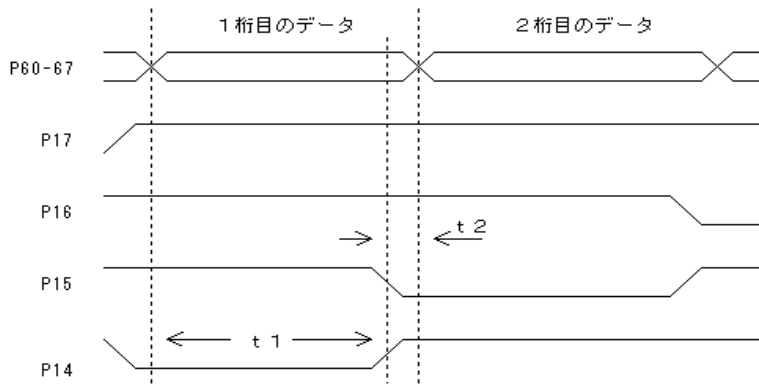
/*****
    タイマ
*****/
void timer1m(void)
{
    unsigned long  i;

    for (i=0; i<TIMER1_TIM; i++){    // i=TIMER1_TIMまでループする
}

```

<Dscan_2>

タイミングチャートのようにプログラムを作りました。一見目的通り点灯しているように見えますが、スキップ実行でプログラムをトレースしてみてください。一つ前の桁の数が表示されてからその桁の数が表示されているのが分かると思います。何故このように一つ前の桁の数を表示してしまうかというと、P14～17 のスキャンを切り替える際、桁は切り替わっているのに P6 の表示データは切り替わっていない為です。

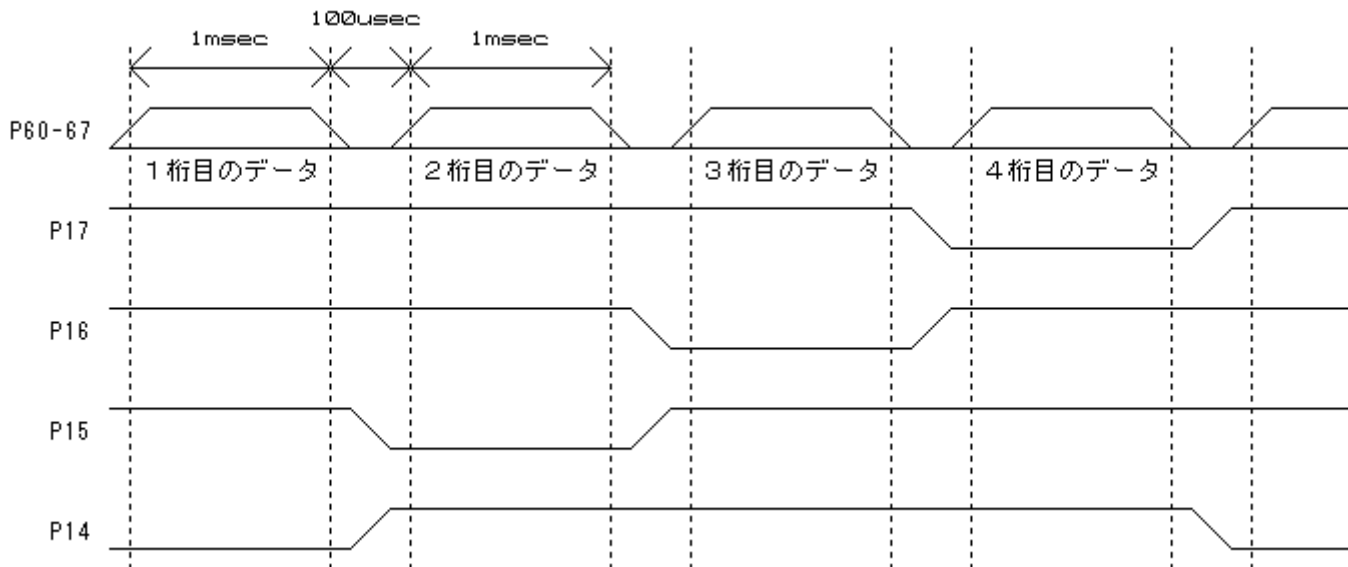


＜図 10-2-3 Dscan_2 のタイミングチャート＞

その様子を示したのが図 10-2-3 です。細かく見ていくとまず P14 が“L”の時 P6 のデータを表示します(t1)。次にスキャンを移動し P15 を“L”にすると、P15 のためのデータに変更するまでの時間、P14 のデータを表示してしまうわけです(t2)。このことを解決する為に次の 2 つのことを考えてプログラムを作り直しました。＜Dscan_3＞

1. 桁スキャン P14～17 を変更する前に P6 のデータを“L”にして LED を一旦消灯する。
2. 1 の状態を安定させる為にタイマーを入れる。タイマーの時間は点灯している時間に対して十分に短くし、見かけ上 LED が消灯していることが分からない位の 100 μ sec とする。

図 10-2-4 はそのタイミングチャートです。



＜図 10-2-4 Dscan_3 のタイミングチャート＞

```

/*****
    インクルードファイル
    *****/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"  // 内蔵I/Oのラベル定義

/*****
    定数の定義
    *****/
#define TIMER1_TIM 0x0d02 // timer1m()のループ回数
#define TIMER01_TIM 0x014a // timer100u()のループ回数

/*****
    関数の定義
    *****/
void main(void);
void init_port(void);
void timer1m(void);
void timer100u(void);

/*****
    メインプログラム
    *****/
void main(void)
{
// ----- イニシャライズ -----
    init_port(); // I/Oポートイニシャライズ

// ----- メインループ -----
    while(1){
        IO.PDR1.BYTE = 0xee; // スキャン出力(P14=Low)
        IO.PDR6.BYTE = ~(0x66); // セグメントデータ4を出力
        timer1m();
        IO.PDR6.BYTE = ~(0x00); // 消灯
        timer100u();

        IO.PDR1.BYTE = 0xdd; // スキャン出力(P15=Low)
        IO.PDR6.BYTE = ~(0x4f); // セグメントデータ3を出力
        timer1m();
        IO.PDR6.BYTE = ~(0x00); // 消灯
        timer100u();

        IO.PDR1.BYTE = 0xbb; // スキャン出力(P16=Low)
        IO.PDR6.BYTE = ~(0x5b); // セグメントデータ2を出力
        timer1m();
        IO.PDR6.BYTE = ~(0x00); // 消灯
        timer100u();

        IO.PDR1.BYTE = 0x77; // スキャン出力(P17=Low)
        IO.PDR6.BYTE = ~(0x06); // セグメントデータ1を出力
        timer1m();
        IO.PDR6.BYTE = ~(0x00); // 消灯
        timer100u();
    }
}

```

```

    }
}

/*****
I/Oポート イニシャライズ
*****/
void init_port(void)
{
    IO.PCR1      = 0xf0;    // P10 ~ 13入力, P14 ~ 17出力
    IO.PDR1.BYTE = 0xf0;    // P14 ~ 17初期出力
    IO.PCR6      = 0xff;    // P60 ~ 67出力
    IO.PDR6.BYTE = 0xf0;    // P60 ~ 67初期出力
}

/*****
タイマ
*****/
void timer1m(void)
{
    unsigned long i;

    for (i=0; i<TIMER1_TIM; i++){ // i=TIMER1_TIMまでループする
}

void timer100u(void)
{
    unsigned long i;

    for (i=0; i<TIMER01_TIM; i++){ // i=TIMER01_TIMまでループする
}

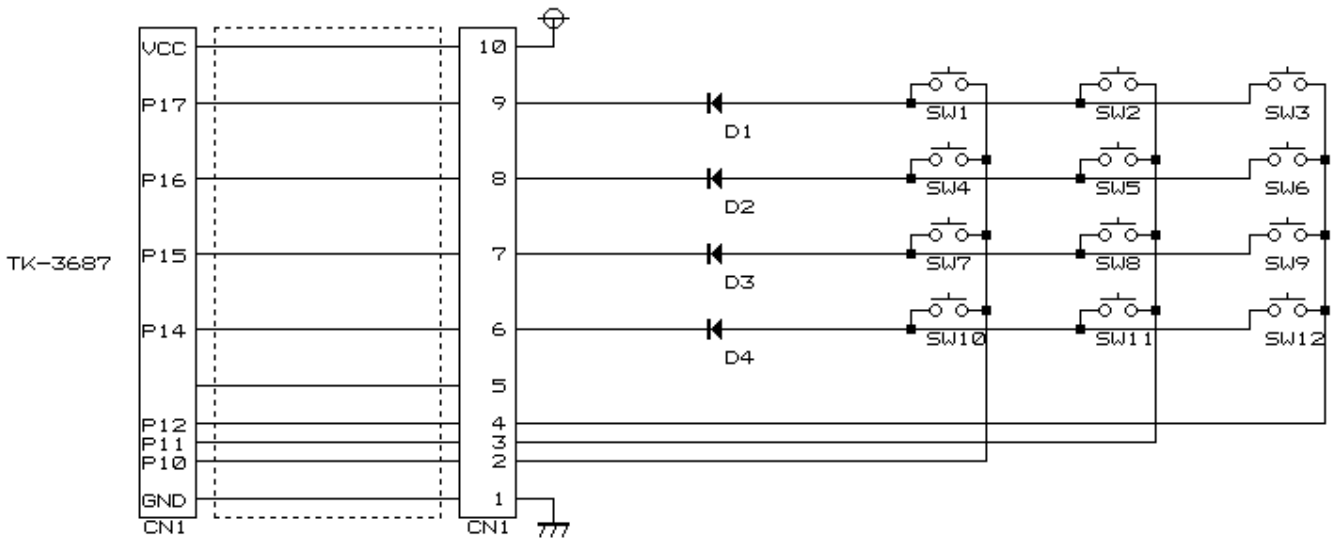
```

<Dscan_3>

10-3 マトリックスキー

ここではマトリックスキーの読み込みプログラムを紹介します。入力の結果を“10-2ダイナミック表示”で製作した7セグメントに表示しますのでまだ作成されていない方は先に表示部を作ってください。10-2同様部品は付属していませんので回路図を見て揃えてください。尚、オプションとして部品一式・ドキュメントをセットにした“7セグメントLED表示&キー入力キット:B6086”を用意しています。

図 10-3-1 はマトリックスキーのみ抜き出した回路図です。P14～17 がスキャンライン、P10～12 が読み込む為の入力ラインです。キーを読み込むには読み込みたいキーのスキャンラインを Low にしてポートを読みます。TK-3687 でポートはプルアップされているのでキーが押されたビットは Low, 押されていなければ Hi となります。



<図 10-3-1 マトリックスキー回路図>

まずはスキャンを行わずにキーを読んでみましょう。スキャンライン P17 のみを Low にして SW1～3 のいずれかが押されたらセグメントに表示します。スキャンラインは表示と共通なのでセグメントの表示は最上桁になります。

```

/*****
    インクルードファイル
    *****/
#include <machine.h> // H8特有の命令を使う
#include "iodefine.h" // 内蔵I/Oのラベル定義

/*****
    関数の定義
    *****/
void main(void);
void init_port(void);

/*****
    メインプログラム
    *****/
void main(void)
{
// ----- インシャライズ -----
    init_port(); // I/Oポートイニシャライズ

// ----- メインループ -----

```

```

while(1){
    IO.PDR1.BIT.B7 = 0; // P17=Low
    if (((~IO.PDR1.BYTE)&0x07)==0x00){ // 何も押されていない
        IO.PDR6.BYTE = ~0x00;
    }
    else{ // 何か押されている
        IO.PDR6.BYTE = ~0x5c;
    }
}
}

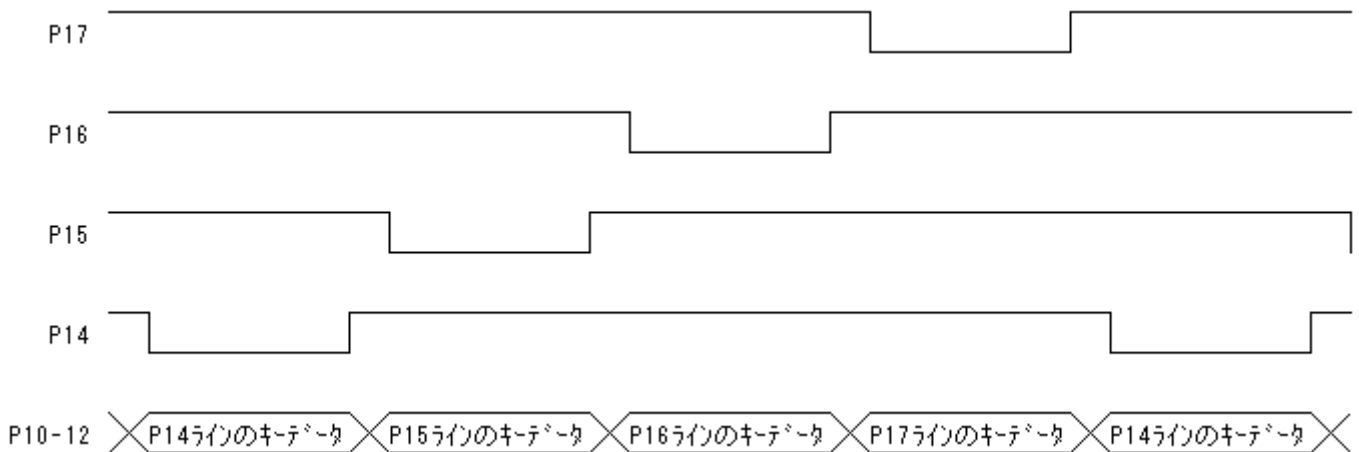
/*****
I/Oポート イニシャライズ
*****/
void init_port(void)
{
    IO.PCR1      = 0xf0;    // P10~13入力, P14~17出力
    IO.PDR1.BYTE = 0xf0;    // P14~17初期出力
    IO.PCR6      = 0xff;    // P60~67出力
    IO.PDR6.BYTE = 0xf0;    // P60~67初期出力
}

```

<Matrixkey_1>

1 ラインの読み込みができたなら次は各ラインをスキャンしてキーを読み込みましょう。

1 つのスキャンラインを Low にしてキーを読み込み、1 ライン読み込んだら次のスキャンラインを Low に、これをスキャンライン分繰り返します。スキャンでのキー読み込みタイミングチャートを図 10-3-2 に示します。



<図 10-3-2 スキャンでのキー読み込みタイミングチャート>

上記のタイミングを元に各キーを読み込むプログラムを作ってみましょう。それぞれのキー番号をセグメントの最下桁に表示させます。マトリックスキーの読み込みを分かり易くするためにダイナミック表示は行わずに表示させます。‘ScanData’ がスキャンデータ、‘Loop’ はループの回数、ここではスキャンラインが 4 本あるので 4 回ループさせます。何かキーが押されていたら、その時のキーの状態を‘KeyData’ にセットし、‘Loop’ と‘KeyData’ の値からキー番号‘KeyNo’を取得します。キーは番号の若い方が優先順位が高く、幾つかのキーが押された場合は優先順位の高い方を表示します。

```

/*****
    インクルードファイル
    *****/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"   // 内蔵I/Oのラベル定義

/*****
    定数の定義
    *****/
#define    TIMER1_TIM  0xd02    // timer1m()のループ回数

/*****
    テーブルの定義
    *****/
const unsigned char SegTable[16] = {0x3f,0x06,0x5b,0x4f,
                                     0x66,0x6d,0x7d,0x07,
                                     0x7f,0x67,0x77,0x7c,
                                     0x39,0x5e,0x79,0x71};

/*****
    グローバル変数の定義
    *****/
unsigned char  ScanData;    // LED出力データ
unsigned char  Loop;       // ループカウンタ
unsigned char  KeyData;    // キーデータ
unsigned char  KeyNo;      // キー番号

/*****
    関数の定義
    *****/
void          main(void);
void          init_port(void);
void          timer1m(void);

/*****
    メインプログラム
    *****/
void main(void)
{
// ----- イニシャライズ -----
    init_port();    // I/Oポートイニシャライズ

// ----- メインループ -----
    while(1){
        IO.PDR6.BYTE = 0xff;    // LED消灯
        ScanData = 0x77;    // スキャンデータの初期化

        for (Loop=0;Loop<4;Loop++){
            IO.PDR1.BYTE = ScanData;    // スキャンデータ出力
            KeyData = (~IO.PDR1.BYTE) & 0x07;    // キーの読込
            if (KeyData!=0){    // キー入力あり
                switch(KeyData){
                    case 0x01:    // ビット0=1

```



```

        KeyNo = Loop * 3 + 1;
        break;
    case 0x02: // ビット1=1
        KeyNo = Loop * 3 + 2;
        break;
    case 0x04: // ビット2=1
        KeyNo = Loop * 3 + 3;
        break;
    }
    break; // for文から抜ける
}
ScanData = rotrc(1,ScanData); // 次のスキャンデータを得る
}

IO.PDR1.BYTE = 0xee; // 表示用スキャン出力
if (Loop<4) {IO.PDR6.BYTE = ~SegTable[KeyNo];} // 入力あり
else {IO.PDR6.BYTE = 0xff;} // 入力なし
timer1m(); // 表示時間
IO.PDR1.BYTE = 0xff; // スキャンオフ
}
}

/*****
I/Oポート イニシャライズ
*****/
void init_port(void)
{
    IO.PCR1 = 0xf0; // P10~13入力, P14~17出力
    IO.PDR1.BYTE = 0xf0; // P14~17初期出力
    IO.PCR6 = 0xff; // P60~67出力
    IO.PDR6.BYTE = 0xf0; // P60~67初期出力
}

/*****
タイマ
*****/
void timer1m(void)
{
    unsigned long i;

    for (i=0; i<TIMER1_TIM; i++){ // i=TIMER1_TIMまでループする
}

```

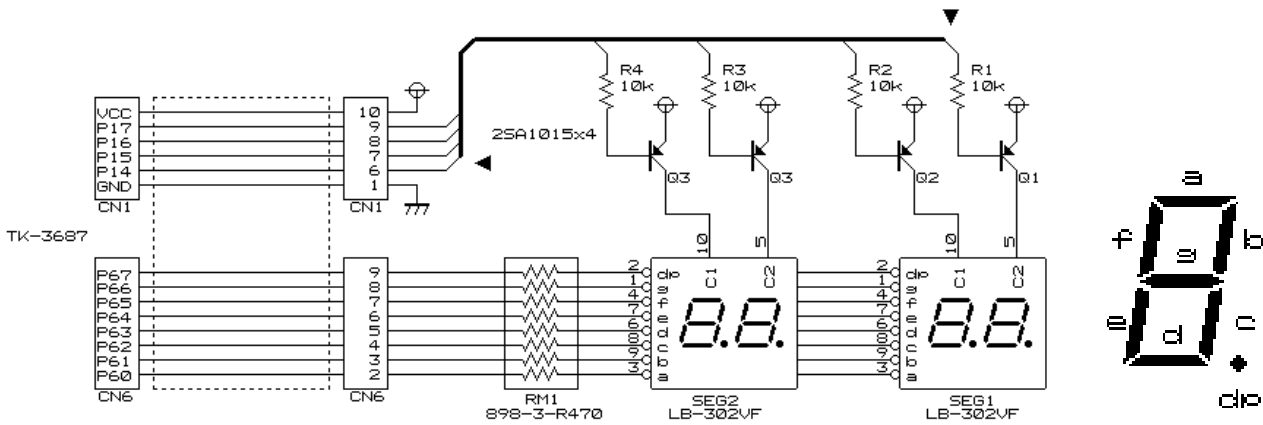
<Matrixkey_2>

10-4 タイマ V 割り込みを利用したダイナミック表示

10-2 でダイナミック表示の考え方を調べましたが、スキャン切替の時間をソフトウェアで行なっているため、その大半は待ち時間になっています。しかし、たいていは表示以外にもいろいろな処理を行ないます(どちらかというと表示よりも重要)。それで、通常の処理を行ないながら必要な時に表示の処理(スキャン切替)を行なう方が効率がよくなります。

問題は表示処理のタイミングをどうやって CPU に知らせるかです。このような時に使うのがタイマ割り込みです。H8/3687 にはいくつかのタイマが用意されていますが今回はタイマ V を使います。一定の間隔(今回は 1ms)で CPU に割り込みをかけて表示処理(スキャン切替)を実行します。割り込みとはハードウェア的にサブルーチンを実行する方法、とも言えるでしょう。

まず回路図を再掲します。



<図 10-4-1 表示部回路図>

タイマ V の概要

タイマ V は 8 ビットカウンタをベースにした 8 ビットタイマです。タイマ V に内蔵されているタイマカウンタ V (TCNTV) は CPU クロック (TK-3687 の場合は 20MHz) を分周したクロック (1/128, 1/64, 1/32, 1/16, 1/8, 1/4 のいずれか選択可能) によって常に +1 されます。TCNTV はタイムコンスタントレジスタ A (TCORA) とタイムコンスタントレジスタ B (TCORB) と比較されており、一致すると CPU に割り込みをかけることができます。その際、TCNTV を 0 にクリアするよう設定することもできます。

今回は CPU クロック (20MHz) の 1/128 (=156.25kHz) で TCNTV を +1 します。そして、TCORA に 156 をセットし、一致したら割り込みをかけると同時に TCNTV を 0 にクリアします。TCNTV は常に +1 されるので、割り込みは約 1ms 毎 (1kHz) にかかることとなります。このタイミングで LED のスキャンを切り替えるようにします。

なお、タイマ V の詳細については「H8/3687 シリーズ ハードウェアマニュアル」の「11. タイマ V」をご覧ください。

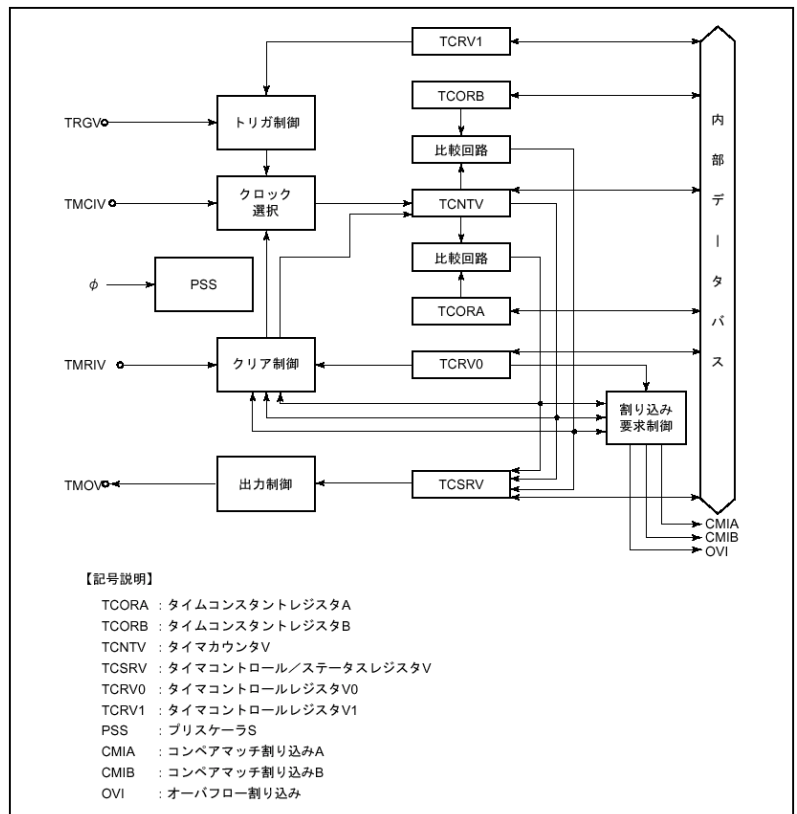


図 12-1 タイマ V のブロック図


```

/*****
    インクルードファイル
*****/
#include <machine.h>      //H8特有の命令を使う
#include "iodefine.h"    //内蔵I/Oのラベル定義

/*****
    定数エリアの定義(ROM)
*****/
const unsigned char  ScanData[4] = {0xe0,0xd0,0xb0,0x70}; //スキャンデータ

const unsigned char  SegTable[16] =  {0x3f,0x06,0x5b,0x4f,
                                       0x66,0x6d,0x7d,0x27,
                                       0x7f,0x6f,0x77,0x7c,
                                       0x39,0x5e,0x79,0x71};

/*****
    グローバル変数の定義とイニシャライズ(RAM)
*****/
unsigned int    CntData = 0; //カウント値,この値を表示する

unsigned char   ScanCnt = 0; //スキャンカウンタ
unsigned char   DispBuf[4] = {0x40,0x00,0x00,0x40}; //表示バッファ

/*****
    関数の定義
*****/
void    main(void);
void    wait(void);
void    intprog_tmv(void);

/*****
    メインプログラム
*****/
void main(void)
{
    // ポートイニシャライズ -----
    IO.PCR1      = 0xf0; //ポート1,P10-13入力,P14-17出力
    IO.PDR1.BYTE = 0xf0;
    IO.PCR6      = 0xff; //ポート6,P60-67出力
    IO.PDR6.BYTE = 0x00;

    //タイマVイニシャライズ -----
    TV.TCSR.V.BYTE = 0x00; //TOMV端子は使わない
    TV.TCOR.A      = 156;  //周期=1ms(1kHz)
    TV.TCRV1.BYTE = 0x01; //TRGVトリガ入力禁止,
    TV.TCRV0.BYTE = 0x4b; //コンペアマッチA 割込みイネーブル
                        //コンペアマッチA でTCNTVクリア
                        //内部クロック /128(=156.25kHz)

    // メインループ -----
    while(1){
        CntData++;
        DispBuf[2] = SegTable[(CntData & 0x00f0) / 0x0010];
        DispBuf[1] = SegTable[ CntData & 0x000f      ];
        wait();
    }
}

```

```

/*****
 ウェイト(250ms)
 *****/
void wait(void)
{
#pragma asm
    push.l    er0
    mov.l    #833333,er0      ;6クロック × 833333 ÷ 20MHz=250ms
loop:
    dec.l    #1,er0          ;2クロック
    bne     loop            ;4クロック
    pop.l    er0
#pragma endasm
}

/*****
 タイマV 割り込み
 *****/
#pragma regsave (intprog_tmv)
void intprog_tmv(void)
{
    TV.TCSR.V.BIT.CMFA = 0;      //コンペアマッチフラグA クリア

    IO.PDR1.BYTE = 0xf0;        //表示を消す
    IO.PDR6.BYTE = 0x00;

    IO.PDR6.BYTE = DispBuf[ScanCnt]; //データ出力
    IO.PDR1.BYTE = ScanData[ScanCnt]; //スキャン信号出力

    ScanCnt++; if (ScanCnt>=4) {ScanCnt = 0;}
}

```

割り込みを使うためにはソースファイルだけではなく、'intprg.c'にも手を加える必要があります。タイマVの割り込みが発生すると'intprg.c'の'INT_TimerV'にジャンプします。割り込み処理自体はメインプログラムに記述しているので(関数名:intprog_tmv)、そこにジャンプするように指定します。下記のリストをご覧ください。

```

/*****
/*
/* FILE      :intprg.c
/* DATE      :Thu, Mar 17, 2005
/* DESCRIPTION :Interrupt Program
/* CPU TYPE   :H8/3687
/*
/* This file is generated by Hitachi Project Generator (Ver.2.1).
/*
*****/

```

```
#include <machine.h>
```

'intprog_tmv' を外部参照にする

```
extern void intprog_tmv(void);
```

```
#pragma section IntPRG
```

```

// vector 1 Reserved

// vector 2 Reserved

// vector 3 Reserved

// vector 4 Reserved

// vector 5 Reserved

// vector 6 Reserved

// vector 7 NMI
__interrupt(vect=7) void INT_NMI(void) { /* sleep(); */}
// vector 8 TRAP #0
__interrupt(vect=8) void INT_TRAP0(void) { /* sleep(); */}
// vector 9 TRAP #1
__interrupt(vect=9) void INT_TRAP1(void) { /* sleep(); */}
// vector 10 TRAP #2
__interrupt(vect=10) void INT_TRAP2(void) { /* sleep(); */}
// vector 11 TRAP #3
__interrupt(vect=11) void INT_TRAP3(void) { /* sleep(); */}
// vector 12 Address break
__interrupt(vect=12) void INT_ABRK(void) { /* sleep(); */}
// vector 13 SLEEP
__interrupt(vect=13) void INT_SLEEP(void) { /* sleep(); */}
// vector 14 IRQ0
__interrupt(vect=14) void INT_IRQ0(void) { /* sleep(); */}
// vector 15 IRQ1
__interrupt(vect=15) void INT_IRQ1(void) { /* sleep(); */}
// vector 16 IRQ2
__interrupt(vect=16) void INT_IRQ2(void) { /* sleep(); */}
// vector 17 IRQ3
__interrupt(vect=17) void INT_IRQ3(void) { /* sleep(); */}
// vector 18 WKP
__interrupt(vect=18) void INT_WKP(void) { /* sleep(); */}
// vector 19 RTC
__interrupt(vect=19) void INT_RTC(void) { /* sleep(); */}
// vector 20 Reserved

// vector 21 Reserved

// vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) {intprog_tmV();}
// vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void) { /* sleep(); */}
// vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) { /* sleep(); */}
// vector 25 ADI
__interrupt(vect=25) void INT_ADI(void) { /* sleep(); */}
// vector 26 Timer Z0
__interrupt(vect=26) void INT_TimerZ0(void) { /* sleep(); */}
// vector 27 Timer Z1
__interrupt(vect=27) void INT_TimerZ1(void) { /* sleep(); */}
// vector 28 Reserved

// vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void) { /* sleep(); */}

```

タイマVの割り込みで
'intprog_tmV'をコールする

```
// vector 30 Reserved

// vector 31 Reserved

// vector 32 SCI3_2
__interrupt(vect=32) void INT_SCI3_2(void) { /* sleep(); */ }
```

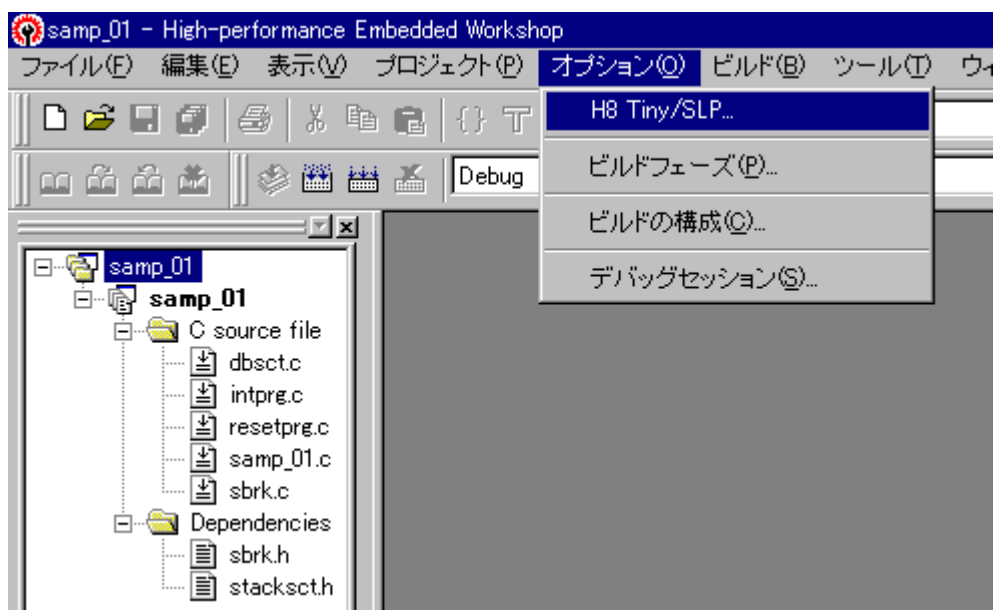
ところで、今までは 'resetprg.c' の割り込みを許可している行をコメントにしていたのですが、今回は割り込みを使うので**変更せずに HEW が自動的に作成した状態のまま**使います。

さて、割り込み処理とは全く関係ないのですが、今回のプログラムでは C のプログラムとアセンブラのプログラムを結合する勉強のために 250ms のウェイトをでアセンブラで記述しています。もう一度その部分を見てみましょう。

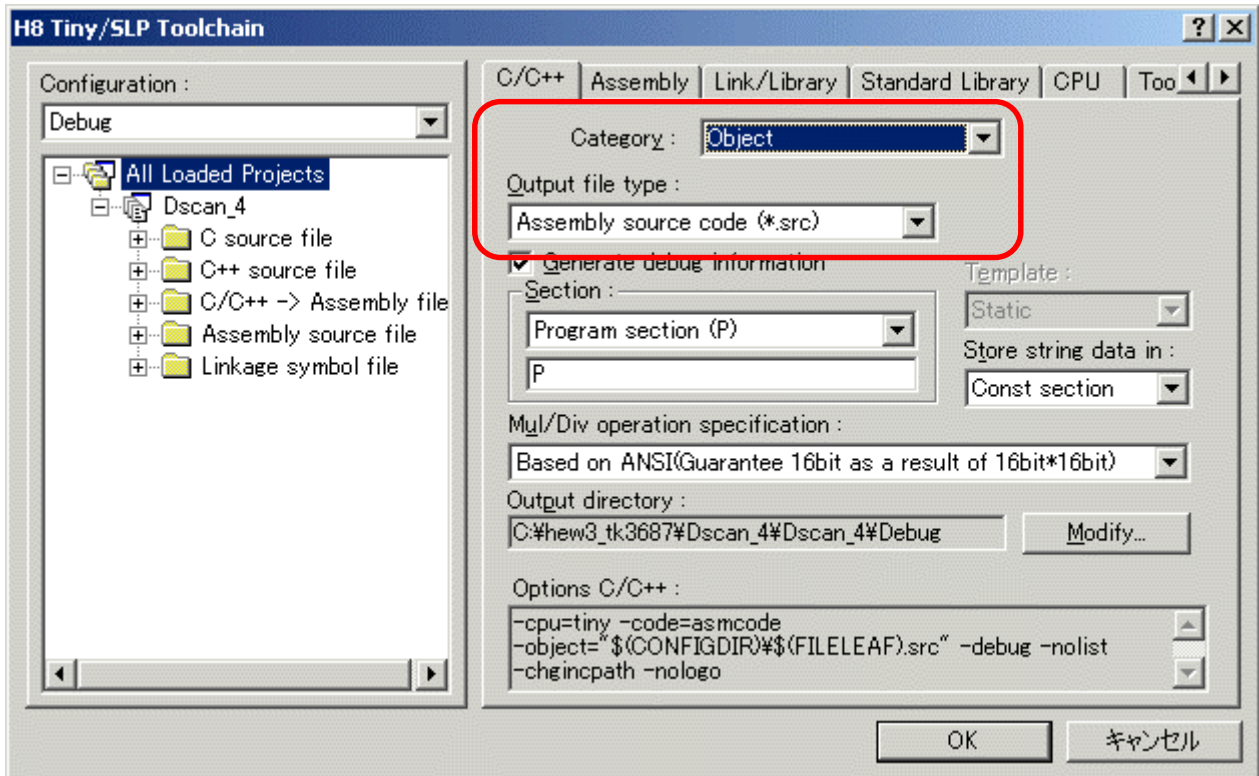
```
/*
 * ウェイト(250ms)
 */
void wait(void)
{
#pragma asm
    push.l    er0
    mov.l    #833333,er0      ;6クロック × 833333 ÷ 20MHz=250ms
loop:
    dec.l    #1,er0          ;2クロック
    bne     loop             ;4クロック
    pop.l    er0
#pragma endasm
}
```

このように、'#pragma asm' ~ '#pragma endasm' で囲まれた範囲にアセンブラの命令を記述することができます。このような記述方法をインラインアセンブルと言います。ただし、HEW のコンパイラは、記述されたアセンブラの命令が C のプログラムに与える影響や、記述されたアセンブラの命令の文法のチェックを行なわないので、プログラマが配慮しなければなりません。例えば、上のプログラムは ER0 レジスタを使っていますが、それがプログラムの動作をおかしくする可能性もあります。それで、ER0 を使う前にスタックにプッシュし、使い終わったらポップします。

もう一つ、インラインアセンブルを使う時は HEW の設定を変更する必要があります。セクションのアドレス設定を行なったときと同じように、右図のようにメニューバーから 'H8Tiny/SLP...' を選びます。



すると、'H8 Tiny/SLP Toolchain' ウィンドウが開きます。'C/C++' のタブを選び、'Category' のドロップダウンメニューを開いて 'Object' をクリックすると、下図のような画面になります。そこで、'Output file type' のドロップダウンメニューを開いて 'Assembly source code (*.src)' を選びます。最後に 'OK' ボタンをクリックしてウィンドウを閉じます。



あとは通常どおりビルドすれば MOT ファイルが作成されます。



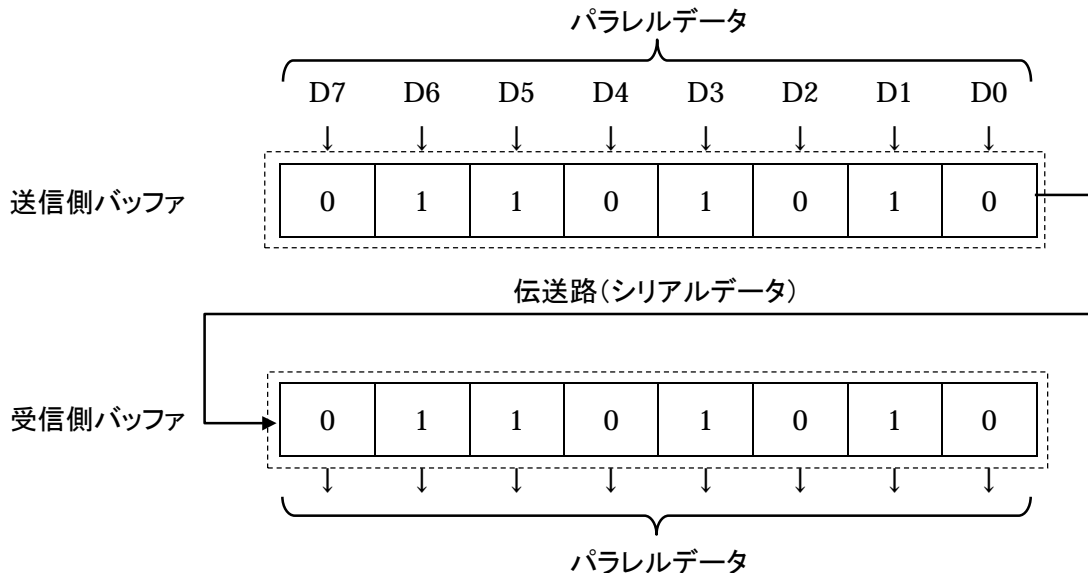
ここで使ったインラインアセンブルは比較的手軽に C のプログラムにアセンブラのプログラムを組み込むことができます。しかし、アセンブラの処理が複雑になってきた場合は、C とアセンブラのソースファイルを別にしてリンカでつなげたほうが便利です。「付録 4 C のプログラムからアセンブラのサブルーチンを呼び出す方法」でこの点は説明しています。是非ご覧下さい。

10-5 リングバッファを用いた通信プログラム

パソコンと TK-3687 を繋いでシリアル通信を行うプログラムを作成してみましょう。パソコンのキーボードから入力されたキーを受信し、それをそのまま送信してターミナル画面上にアンサーバックするだけのプログラムです。パソコンのソフトにはターミナルソフト“ハイパーターミナル”を使用します。TK-3687 のプログラムは、パソコンからいつデータが送られてくるかわからない為、割り込み処理でデータを受信しリングバッファという構造のバッファに蓄え、メインループで読み取った後データを送り返します。プログラムを作成する前にシリアル通信と調歩同期(非同期)式通信について、またリングバッファについて検討してみます。

■シリアル通信

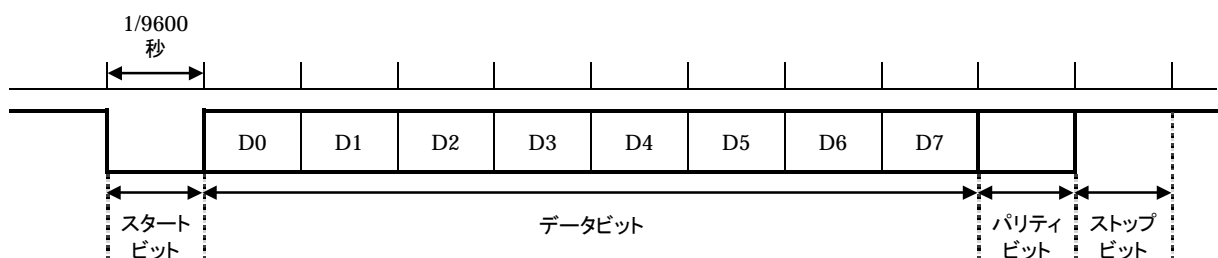
パソコンの COM ポートはシリアルで通信を行います。シリアル通信とは情報を時分割し、それを信号に乗せて 1 対の伝送路で伝える方式です。つまり送信側はパラレル(並列)データをシリアル(直列)データに、受信側はシリアルデータをパラレルデータに変換してデータを送受信します。こうすることで通信に要する信号線を少なく出来るメリットがあります。遠距離間をつなぐ LAN や ADSL 回線などもシリアル通信です。



【図 10-4-1 シリアル通信】

■調歩同期(非同期)式

COM ポートは調歩同期(非同期)式のシリアル通信です。調歩同期(非同期)式とは、各キャラクタ(送受信するデータ)の先頭にスタート・ビット、最後にストップ・ビットを付加して、1 キャラクタ毎に同期を取る方式です。よく『ボーレート 9600bps・データ長 8bit・ストップビット 1bit・パリティ無し』というような表現を耳にする事があるかと思いますが、これは調歩同期式の通信設定を表しています。図 10-4-2 はボーレート 9600ps、データ長 8bit、パリティ有り、ストップビット 1bit でのフォーマットを示しています。次にそれぞれの設定項目について検討してみましょう。



【図 10-4-2 調歩同期式のフォーマット】

- **ボーレート:**

シリアル通信は 1 つの信号線に 1 か 0 のビット情報が時々刻々と送られてきます。この時ビットを送り出す間隔と取り込む間隔が同じでなければ正確にデータを送受信することは出来ません。その間隔を示しているのがボーレートです。9600bps (bps=bit per second) なら約 0.1m 秒間が 1bit であることを表し、ボーレートを双方が合わせておくことでデータをやり取り出来るようになります。

- **スタート・ビット:**

ボーレートでビットの間隔を合わせることは出来ました。しかしデータが何も載っていない時の信号は“H”レベル(実際の信号線には RS232C ドライバでレベル変換・反転され-12V、“L”レベルは+12V)のため、“H”から始まるデータを送信しても受信側はレベルが変化せず、データの始まりだとは分かりません。そこで通信を始めるにはまず 1 ビット分信号を“L”にしてデータの始まりを示します。これがスタートビットです。

- **データ長(データ・ビット):**

スタートビットの後に続くデータのビット長です。TK-3687 では 8bit を指定します。

- **パリティ・ビット:**

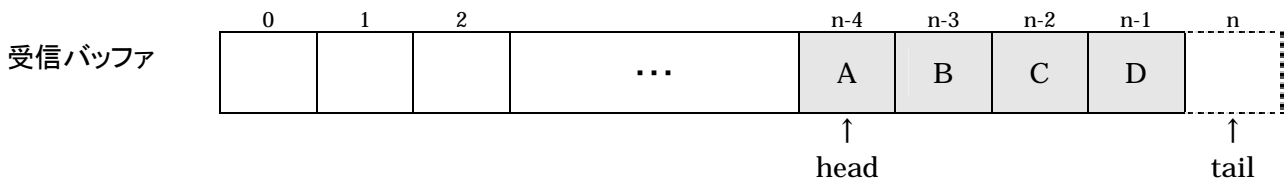
受信したデータの誤りを検出する為のビットです。データの“1”のビットが奇数になるように付加されるのが奇数パリティ、逆に偶数になるように付加されるのが偶数パリティです。パリティ無し(Non-Parity)の時はパリティ・ビットは付加されません。

- **ストップビット:**

スタートビットと同様に、仮にデータの最後のビットやパリティ・ビットが“L”だった場合、データを送信し終わってからデータが何も無い状態の“H”に戻さなくてはなりません。また連続してデータを送信する場合でも一定期間“H”を保ってからスタートビットを出力します。この“H”の期間を表しているのがストップ・ビットです。

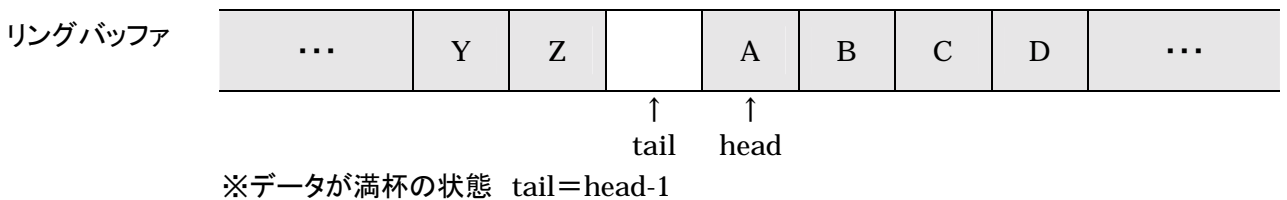
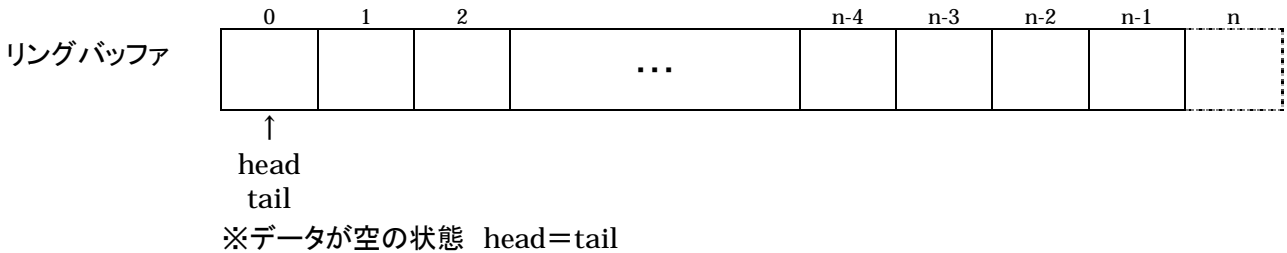
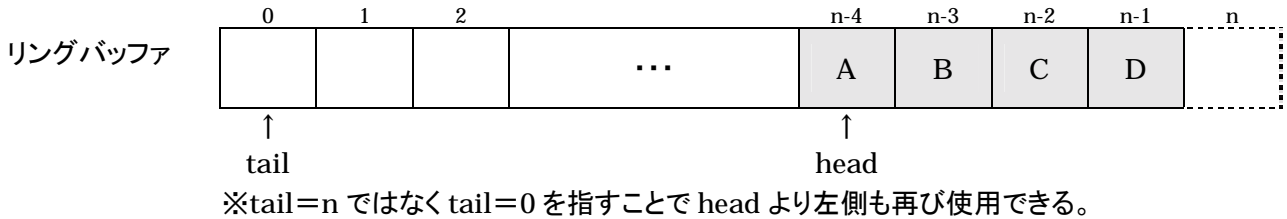
■リングバッファ

通常、受信バッファの構造はパソコンから送られてきたデータを古い順から読み出す、つまり先入れ先出し(FIFO: First In First Out)の構造です。受信バッファを管理するにはデータが格納されている先頭を示すポインタ head と、最後を示すポインタ tail の二つのポインタが必要です。データが無い時は head=tail となり、データをセットするには tail の示すポイントにデータをセットして tail を +1 します。逆に取り出すときは head の示すポイントからデータを取り出し head を +1 します。しかし、これではバッファの有効範囲がどんどん右にシフトしてしまい、head の左側に空き領域があるにも関わらずデータを格納できない状態になってしまいます。



※head より左側が空いているにも関わらず、データをセットする事が出来ない。

そこで tail=n になったら tail を 0 番目に戻すようにすれば空いている左側の部分を再び使用することが出来ます。これがリングバッファの考え方です。バッファの先頭と終わりをくっつけてリング状にしたイメージを持つと理解しやすいのではないかと思います。尚、リングバッファではデータが満杯になると head=tail となり空の時と区別がつかなくなってしまうので、一つ空いた状態 tail=head-1 で満杯とします。



それではプログラムを作成しましょう。冒頭で受信処理は割り込みで行うと記しました。割り込み処理の記述の仕方を説明します。修正は幾つかのソースファイルにまたがりますが、そう難しくはありません。

H8/3687 の受信割り込みは、[CCR の bit7(I:割り込みマスクビット)を 0 にセット]、[シリアルコントロールレジスタ(SCR3)の bit6(RIE:レシーブインタラプトイネーブル)を 1 にセット]すれば受信割り込みが受け付けられます。CCR の bit7 は今まで resetprg.c ファイルでコメント化していた部分をコメント化せずに行っておけば、0 がセットされます。

```

/*****
/*
/* FILE      : resetprg.c
/* DATE      : Wed, Nov 26, 2003
/* DESCRIPTION : Reset Program
/* CPU TYPE  : H8/3687
/*
// _s1ptr=NULL; // Remove the comment when you use strtok()
// HardwareSetup(); // Remove the comment when you use Hardware Setup
set_imask_ccr(0);
main();
// _CLOSEALL(); // Remove the comment when you use SIM I/O
// _CALL_END(); // Remove the comment when you use global class object

```

この行を
コメント化しない

次に intprg.c ファイルに割り込みプログラムを入力します。入力する場所は vector23 SCI3 の{ }内です。ここでは割り込みで行う受信処理の関数のみを入力し、関数の内容はメインプログラム内で定義・記述します。こうすることでメインプログラムのファイルを見ればプログラム全体を見渡すことが出来ます。

```

/*****
/*
/* FILE      :intprg.c
/* DATE      :Wed, Nov 26, 2003
/* DESCRIPTION :Interrupt Program
/* CPU TYPE   :H8/3687
/*
/* This file is generated by Hitachi Project Generator (Ver.2.1).
/*
/*****

#include <machine.h>
#pragma section IntPRG

extern void  set_rxd(void);    // 外部参照

// vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) { /* sleep(); */}

/*****
SCI3割り込み処理
*****
// vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void)
{
    set_rxd();    // 受信割り込み処理
}

// vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) { /* sleep(); */}

```

使用する関数は外部参照

Vector23 SCI3に受信処理の関数を入力

以上で割り込み処理の記述は終わりです。

メインプログラムのリスト“queue_buf.c”を以下に示します。冒頭で述べたように受信処理は割り込みで行い関数 set_rxd();でデータの受信・リングバッファへのセットを行います。メインループでは関数 get_q0();でデータがあればリングバッファから取り出し、関数 send_csi3();で送信します。

```

/*****
/*
/* FILE      :queue_buf.c
/* DATE      :Wed, Nov 26, 2003
/* DESCRIPTION :Main Program
/* CPU TYPE   :H8/3687
/*
/* This file is generated by Toyo-linx,Co.,Ltd. / yFurukawa
/*
/*****

```

```

/*****
    インクルードファイル
    *****/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"  // 内蔵I/Oのラベル定義

/*****
    定数の定義
    *****/
#define MAX_SIZE      8
#define OK             0 // 戻りコード
#define NG             -1 // 戻りコード

/*****
    変数の定義
    *****/
char RxBUF[MAX_SIZE]; // 受信リングバッファ
char *head;           // 先頭ポインタ
char *tail;           // 最後ポインタ
char *queue_min;      // 配列の最初
char *queue_max;      // 配列の最後
char RxData;          // 受信データ
char BufData;         // 受信リングバッファ読み出しデータ

/*****
    関数の定義
    *****/
void main(void);
void set_rxd(void); // 受信割り込み処理
void sci3_ini(void); // SCI3の初期化
void rxbuf_ini(void); // 受信リングバッファの初期化
int put_q(char); // 受信データの格納
int get_q(char *); // 受信リングバッファから取り出し
int send_sci3(char); // シリアルポートSCI3送信
int recv_sci3(char *); // シリアルポートSCI3受信

/*****
    メインプログラム
    *****/
void main(void)
{
// ----- イニシャライズ -----
    sci3_ini(); // SCI3の初期化
    rxbuf_ini(); // 受信リングバッファの初期化

// ----- メインループ -----
    while(1){
        int sci3_sta; // SCI3ステータス
        int rxbuf_sta; // 受信リングバッファステータス
        // バッファから読み出してシリアルポートへ送信
        rxbuf_sta = get_q(&BufData);
        if (rxbuf_sta == OK){
            sci3_sta = send_sci3(BufData);
        }
    }
}

```

```

    }
}

/*****
    SCI3:受信割り込み処理
*****/
#pragma regsave (set_rxd)
void set_rxd(void)
{
    int    sci3_sta;           // SCI3ステータス
    int    rxbuf_sta;         // 受信リングバッファステータス
                                // シリアルポート受信とバッファへのセット
    sci3_sta = recv_sci3(&RxData); // シリアルポートSCI3受信
    if (sci3_sta == OK){      // 受信データがあればバッファへセット
        rxbuf_sta = put_q(RxData);
    }
}

/*****
    SCI3の初期化
*****/
void sci3_ini(void)
{
    #define    MHz    20           // Clock=20MHz
    #define    BAUD    38400       // baudrate
    #define    BITR    (MHz*1000000)/(BAUD*32)-1
    #define    WAIT_1B (MHz*1000000)/6/BAUD

    unsigned long    i;

    IO.PMR1.BIT.TXD = 1;
    SCI3.SCR3.BYTE = 0x00;        // 動作停止
    SCI3.SMR.BYTE = 0x00;        // 非同期・8bit・P-non・CLK/1
    SCI3.BRR = BITR;            // ビットレート
    for (i=0; i<WAIT_1B; i++){    // 1bit期間 wait
        SCI3.SCR3.BYTE = 0x70;    // 送受信・受信割り込み イネーブル
    }

/*****
    受信リングバッファの初期化
*****/
void rxbuf_ini(void)
{
    head = RxBUF;                //headの初期化
    tail = head;                 //tailの初期化
    queue_min = RxBUF;
    queue_max = RxBUF+MAX_SIZE-1;
}

/*****
    データの格納
    戻り値 OK or NG
*****/
int put_q(char Data)
{

```

```

int ret_code;

if ((head == queue_min && tail == queue_max) || tail == head-1)
    ret_code = NG;
else {
    *tail = Data;          //データを格納
    tail++;                //tailの更新
    if (tail > queue_max) tail = queue_min;
    ret_code = OK;
}
return ret_code;
}

/*****
データの取り出し
返却値 OK or NG
*****/
int get_q(char *pd)
{
    int ret_code;

    if (tail == head) ret_code = NG;
    else {
        *pd = *head;      //データの取り出し
        head++;           //headの更新
        if (head > queue_max) head = queue_min;
        ret_code = OK;
    }
    return ret_code;
}

/*****
シリアルポート(SCI3)送信
返却値 OK or NG
*****/
int send_sci3(char txd)
{
    char ret_code;

    if (SCI3.SSR.BIT.TDRE == 1){
        SCI3.TDR = txd;
        ret_code = OK;
    }
    else ret_code = NG;

    return ret_code;
}

/*****
シリアルポート(SCI3)受信
返却値 OK or NG
*****/
int recv_sci3(char *pd)
{
    char ret_code;

```

```
if      (SCI3.SSR.BIT.RDRF == 1){
    *pd = SCI3.RDR;
    ret_code = OK;
}
else    ret_code = NG;

return ret_code;
}
```


10-6 PB0~7(A/D コンバータアナログ入力端子兼用)の 8 ビットパラレル入力方法

A/D コンバータのアナログ入力端子と兼用しているポート B を I/O ポートとして使用した場合のデータリードについて説明します。

プログラムの仕様

ポート B から入力したデータを、そのままポート 5 に出力します。

プログラムの考え方

普通に考えると次のようなプログラムで OK です。

```
void main(void)
{
    IO.PCR5 = 0xff;          // ポート5を出力に設定

    while(1){
        IO.PDR5.BYTE = IO.PDRB.BYTE;
    }
}
```

ところが、これだとビット 0 が Low のままになってしまいます。なぜでしょうか。

ここで、H8/3687 のハードウェアマニュアルをもう一度見てみましょう。「9. 8. 1 ポートデータレジスタ B (PDRB)」をご覧ください。PDRB はポート B の汎用入力データレジスタですが、説明の部分にこのように記されています。

「このレジスタをリードすると各端子の入力値が読み出されます。ただし、A/D 変換器の ADCSR によりアナログ入力チャンネルに指定されている端子はリードすると 0 が読み出されます。」

つまり、A/D 変換器を使う/使わないに関係なく ADCSR でアナログ入力に選択されている端子は Low 固定になります。ADCSR はリセットでチャンネル 0 (端子は PB0 と兼用) になります。そのためビット 0 が Low になってしまうのです。

ではどのようにすればポート B のデータを正しく読み出すことができるでしょうか。

まず、ADCSR でチャンネル 0 を選択します。この状態でポート B を読み出すとビット 1~7 は正常に読み出され、ビット 0 は Low になります。この値をメモリにストアします。

次に、ADCSR でチャンネル 1 に切り替えます。この状態でポート B を読み出すとビット 0 とビット 2~7 は正常に読み出され、ビット 1 は Low になります。

この値と、先ほどメモリにストアした値のオア(論理和)をとります。そうすると、ポート B のデータを正しく読み出すことができます。

プログラム例を次のページに掲載します。

```

/*****/
/* */
/* FILE      :pb_p5.c */
/* DATE      :Wed, May 25, 2005 */
/* DESCRIPTION :Main Program */
/* CPU TYPE   :H8/3687 */
/* */
/* This file is generated by Renesas Project Generator (Ver.4.0). */
/* */
/*****/

#include <machine.h> // H8特有の命令を使う
#include "iodefine.h" // 内蔵I/Oのラベル定義

void main(void);

void main(void)
{
    unsigned char BUF;

    IO.PCR5 = 0xff; // ポート5を出力に設定

    while(1){
        AD.ADCSR.BYTE = 0x00;
        BUF = IO.PDRB.BYTE;
        AD.ADCSR.BYTE = 0x01;
        IO.PDR5.BYTE = BUF | IO.PDRB.BYTE;
    }
}

```

付録-1 ユーザプログラムとハイパーH8 を書き込む

ユーザプログラムを FlashROM に書き込み、それをハイパーH8 で実行する場合、ただ空いているエリアにユーザプログラムを配置し書き込めばよいかと言うとそうではありません。それはアドレス H'0000 の割り込みベクタテーブルをハイパーH8 とユーザプログラムの両方が使用するからです。ユーザのソースをコンパイルするとベクタテーブルのアドレスは必ず H'0000 になってしまうので、重複を避けなくてはなりません。以下にモニタとユーザプログラムを混在させて TK-3687 に書き込む方法を示します。

■ユーザプログラムの配置

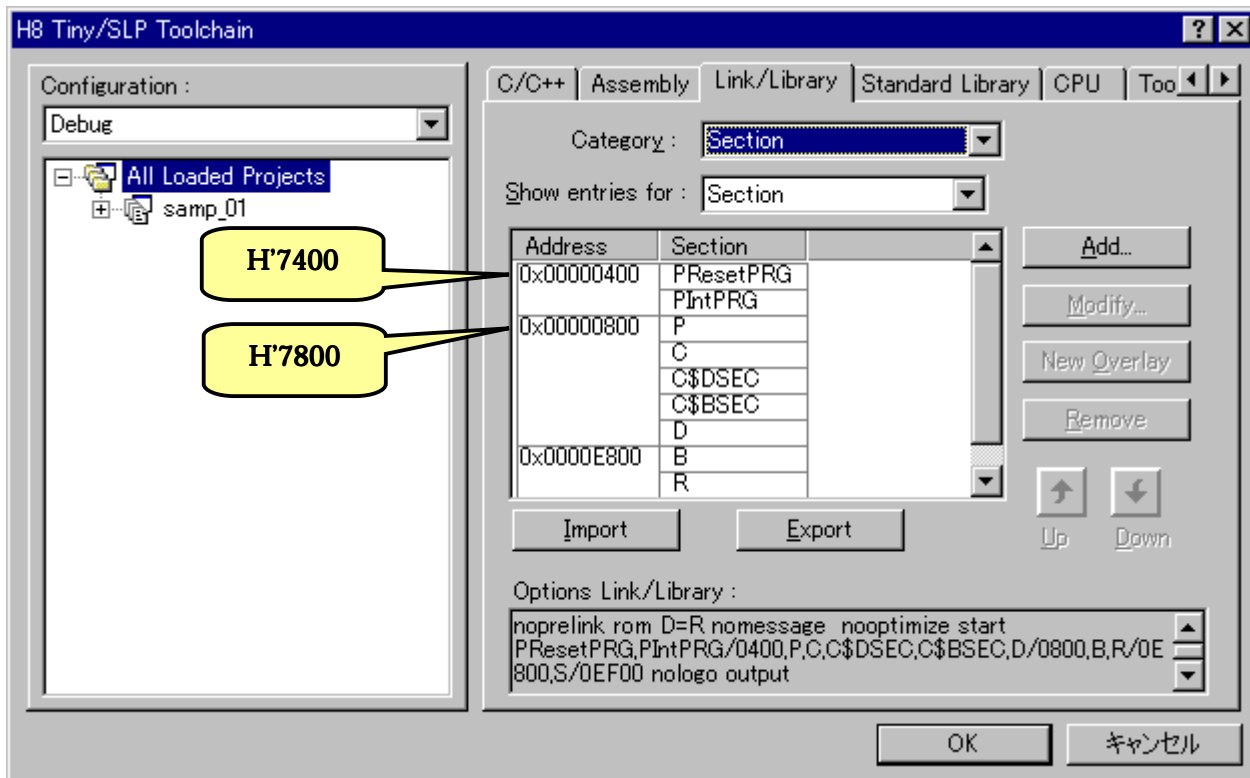
ハイパーH8 とユーザプログラムがバッティングしないようにユーザプログラムのセクションを変更します。ハイパーH8 とデモプログラムが使用しているエリアは以下のようになっています。

アドレス	メモリマップ	
H'0000	モニタプログラムエリア	内蔵 ROM (56kByte)
H'5FFF		
H'6000 H'6FFF		
H'7000		
H'DFFF	ユーザプログラムエリア (28kByte)	
	未使用	未使用
H'E800	ユーザ RAM エリア	内蔵 RAM (2kByte)
H'EFFF		
	未使用	未使用
H'F700 H'F77F	内部 I/O レジスタ	内部 I/O レジスタ
H'F780	ユーザ RAM エリア	内蔵 RAM (2kByte)
H'FDFF		
H'FE00 H'FF7F	モニタワークエリア	
H'FF80 H'FFFF	内部 I/O レジスタ	内部 I/O レジスタ

ハイパーH8 が使用しているエリアと空きエリア

メモリマップ上の網掛けは使用できないエリアを表しています。ユーザに開放されてるエリアは ROM が H'7000 ~ H'DFFF、RAM が H'E800 ~ H'EFFF と H'F780 ~ H'FDFE です、それに合わせて各セクションのアドレスを変更します。本文では例としてアドレス H'7000 にユーザプログラムを配置します。

まず、セクションを変更します。メニューバーから 'オプション>H8 Tiny/SLP...' で H8 Tiny/SLP Toolchain ウィンドウを開き、'Link/Library' タブを選択、Category のドロップダウンメニューから 'Section' を選択します。各セクションとアドレスが表示されますので次のように変更し 'OK' をクリックして下さい。



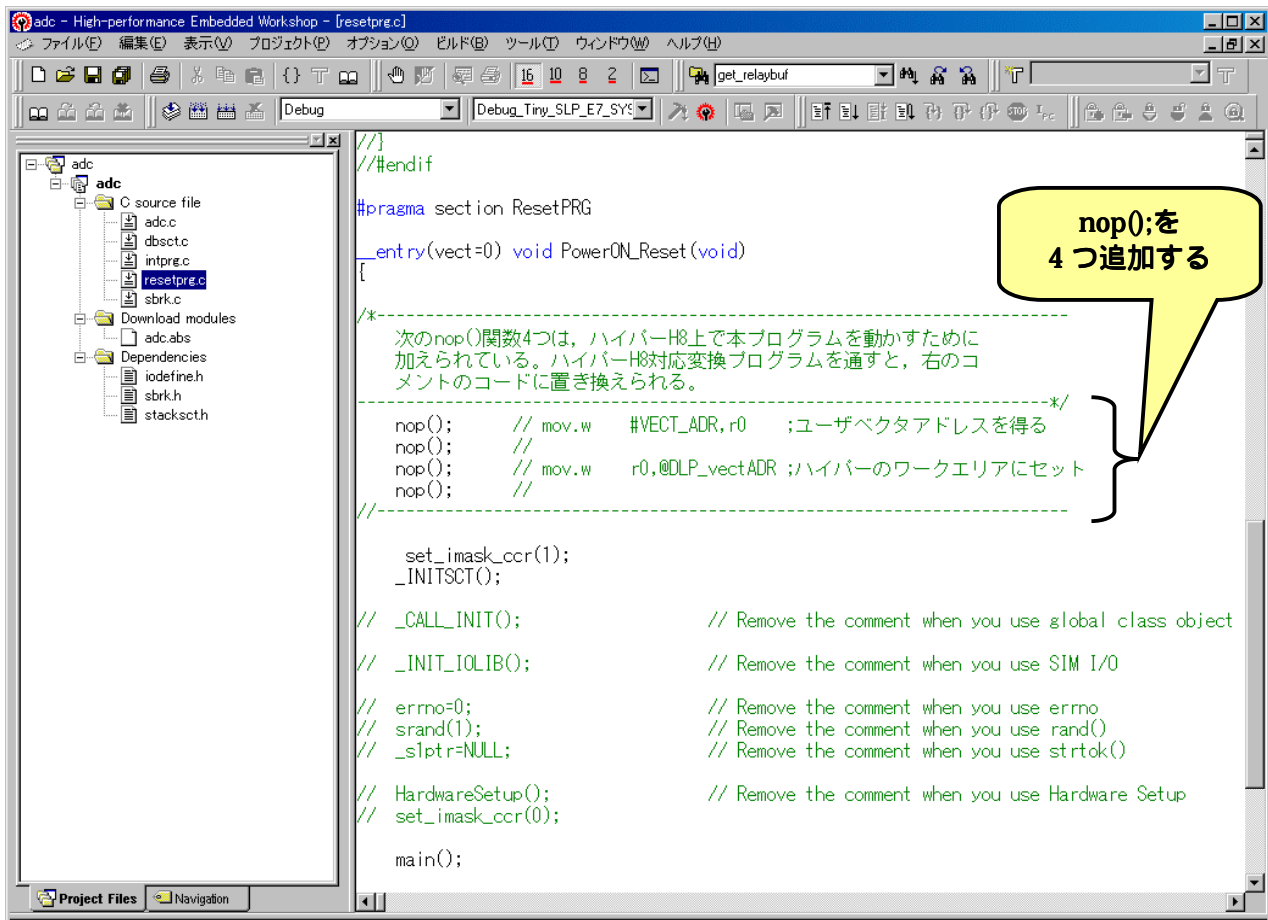
この後の作業でユーザ割り込みベクタを 'プログラム先頭アドレス-H'400' に再配置しますが、プログラムの先頭(上図の場合は PResetPRG)アドレスは、

$$\text{PResetPRG (プログラムの先頭アドレス)} - \text{H'400} \geq \text{H'7000 (ユーザプログラムエリアの先頭アドレス)}$$

として下さい。もし、プログラムの先頭アドレスを H'7000 とするとベクタテーブルのアドレスは H'6C00 となり、デモプログラムエリアを侵食してしまいます。従って、ユーザプログラムの先頭アドレスの下限は H'7400 となります。

■resetprg.c に nop 命令を追加する

リセットプログラムソースファイル 'resetprg.c' 内にモニタプログラムと共存させる為の細工を施す空きエリアを作ります。'resetprg.c' ファイルを開き、リセットプログラム void PowerON_Reset(void) の先頭に nop(); を 4 つ追加します。以下に追加したソースを示しますのでこれに倣って入力して下さい。



後の作業でこの `nop()` で作成された空きエリアに、ユーザ割り込みベクタのアドレスをハイパーH8 のワークエリアにセットするプログラムコードが展開されます。尚、`nop()` 命令はプログラムカウンタを進めるだけの命令ですので、ハイパーH8 と共存させない場合に残しておいても動作に影響を与えません。

以上でユーザプログラムの変更は終了です。コンパイルしてSレコードファイル(以降、motファイル)を作成してください。次に作成された mot ファイルを変換プログラムを使ってモニタと共存できるように変換します。

■ mot ファイルの変換

作成されたユーザプログラムの mot ファイルは、割り込みベクタテーブルのアドレスが H'0000 になっています。そこで変換プログラム 'movevec.exe' を使ってベクタテーブルのアドレスを移動します。また resetprg.c ファイルで作成した空きエリアにベクタテーブルのアドレスをハイパーH8 にセットする命令のコードを展開します。

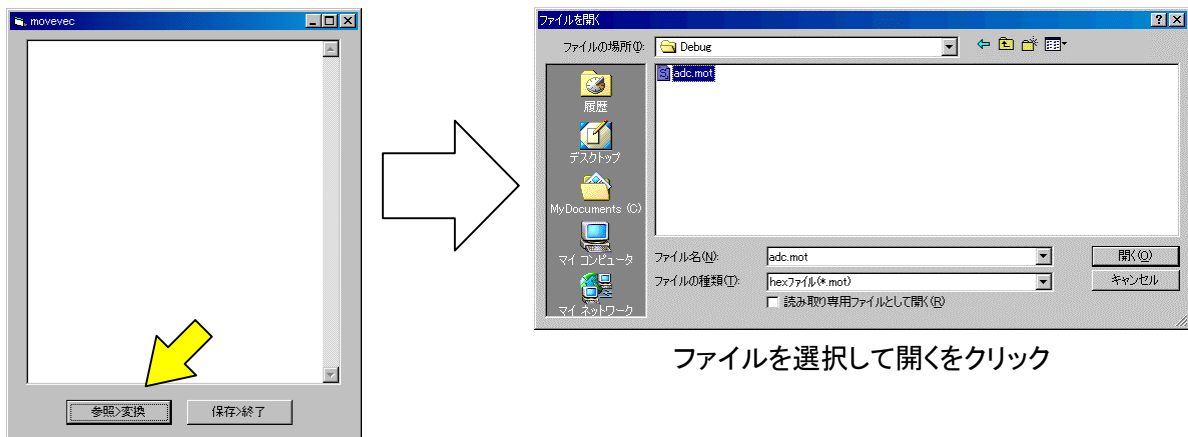
1. movevec.exe をコピーする

まず movevec.exe をパソコンへコピーしましょう。movevec.exe は付属 CD-ROM の次のディレクトリにあります。
CD-ROM¥TK-3687¥変換プログラム¥ movevec.exe

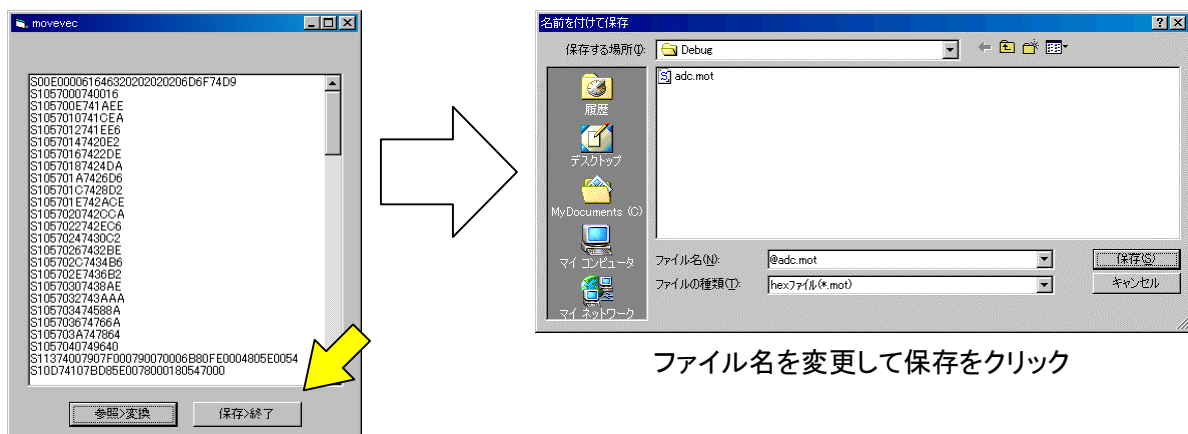
コピー先はどこでも構いませんがアプリケーションなので、通常は C:¥Program Files 内にフォルダを作成しそこにコピーすると管理し易いです。また、すぐ起動できるようにデスクトップかワークフォルダにショートカットを作成しておきましょう。

2. movevec.exe を起動して mot ファイルを変換する

作成した movevec.exe のショートカットをダブルクリックしてプログラムを起動します。'参照>変換' ボタンをクリックし、コンパイルして作成した mot ファイルを選択して '開く' をクリックします。



3. 変換を行いウィンドウに変換結果を表示します。このままではまだ保存されていないので '保存>終了' ボタンで変換した内容を保存します。'保存>終了' ボタンをクリックすると保存ウィンドウが開きますので、元のファイルと同じでも構いませんが区別する為にファイル名を変更して '保存' を選択して下さい(ここでは頭に@を付けています)。



入力したファイル名で変換後の mot ファイルを保存し、movevec.exe は自動的に終了します。以上で mot ファイルの変換は終了です。次は FDT で TK-3687 にプログラムを書き込みます。

4. ■FDTでTK-3687にプログラムを書き込む

ユーザプログラムをTK-3687にダウンロードするためのプログラムFDT(Flash Development Toolkit)を用いてTK-3687にプログラムを書き込みます。FDTをお持ちの方は次へお進み下さい。お持ちでない方は株式会社ルネサステクノロジのホームページよりダウンロードします。

※FDTのダウンロード

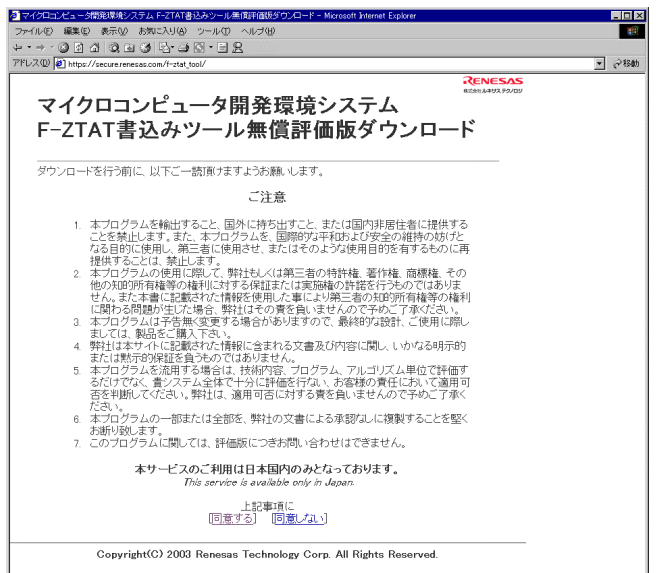
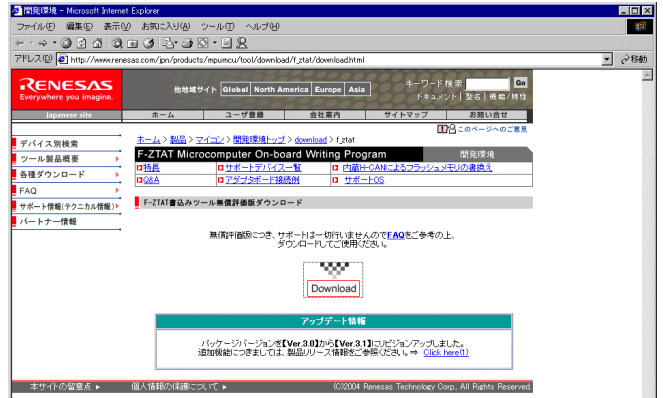
以下のURLからダウンロードします。

http://www.renesas.com/jpn/products/mpumcu/tool/download/f_ztat/download.html
(2004年4月現在)

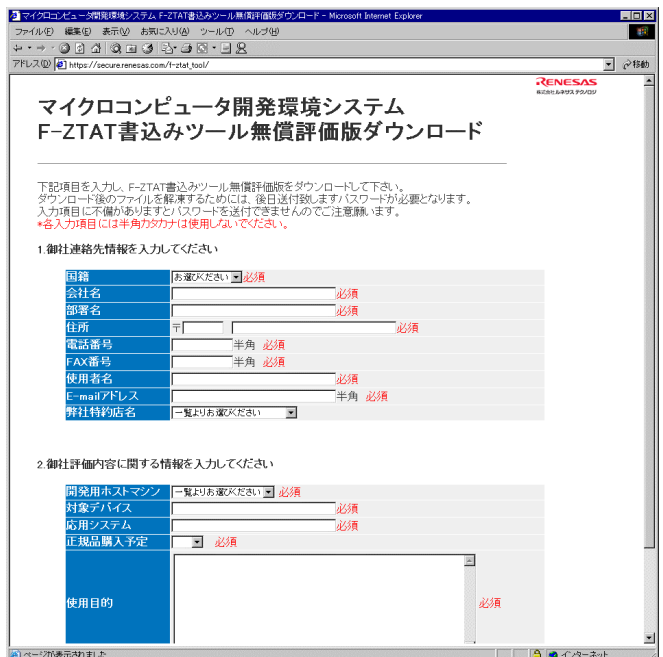


ダウンロードするには **Download** をクリックします。

注意事項が記されたページへ移動するので内容を
読み、同意した上で[同意する]をクリックします。



ユーザ情報を入力し、プログラムをダウンロードします。入力したメールアドレスに、プログラムを解凍する際必要なパスワードが送られてくるので入力事項に間違いが無い様、注意して下さい。パスワードが送られてきたらプログラムを解凍しインストールします。

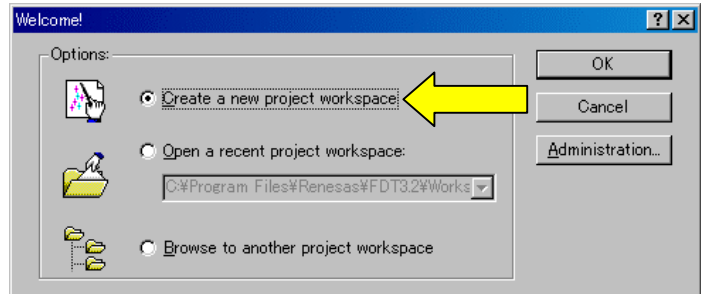


■FDT のセッティング(ワークスペースとプロジェクトの立ち上げ)

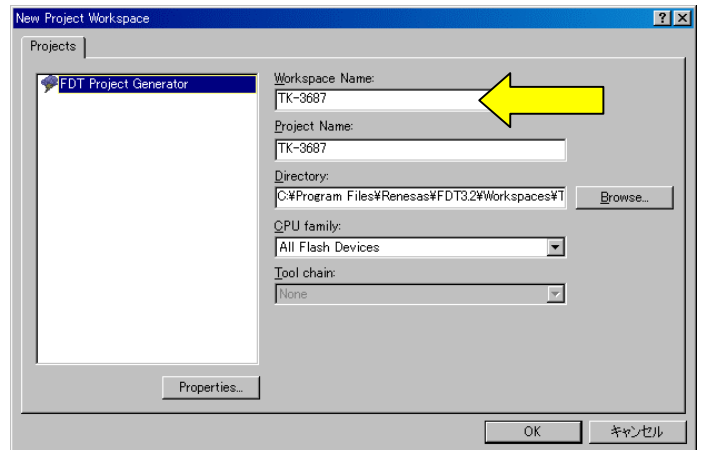
1. スタートメニューから“Flash Development Toolkit 3.2”を起動します。



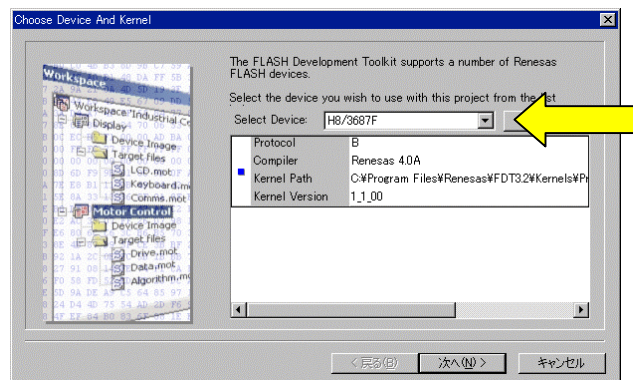
2. 右図のようなダイアログが開くので、“Create a new Workspace”を選択して **OK** をクリックします。



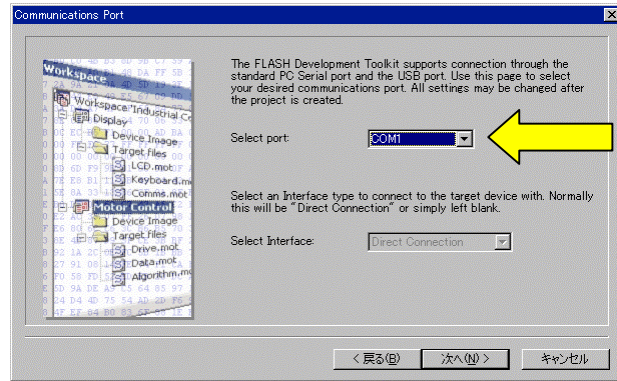
3. “Workspace Name”を決定します。名前は自由に決めて結構です(ここでは TK-3687 としています)。またワークスペースを作成するディレクトリを指定したい場合は“Directory:”の **Browse...** をクリックしディレクトリを指定して下さい。よければ **OK** をクリックし次へ進みます。



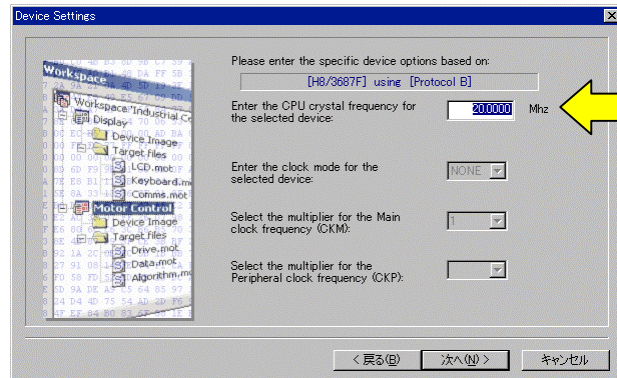
4. デバイスを選択します。“Select Device”の欄で“H8/3687F”を選択し、 **次へ(N) >** をクリックします。



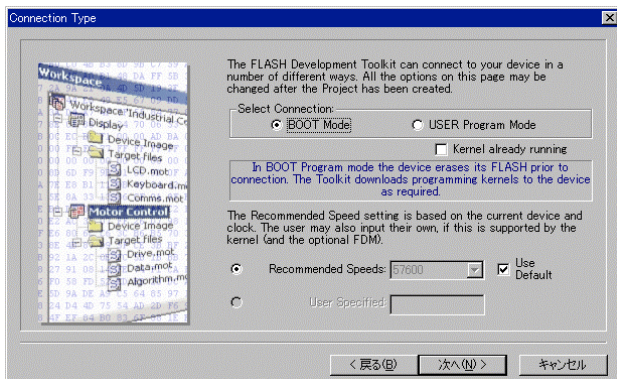
5. 使用する Com ポートを選択します。“**Select port**”で接続する Com ポートを選択し、**次へ(N) >** をクリックします。



6. CPU のクロックを入力します。“**Enter the CPU crystal frequency ...**”の欄に実装されているクロックの周波数“**20.00**”MHz を入力し、**次へ(N) >** をクリックします。



7. この後出てくる項目は入力・変更の必要はないので **次へ(N) >** をクリックします。

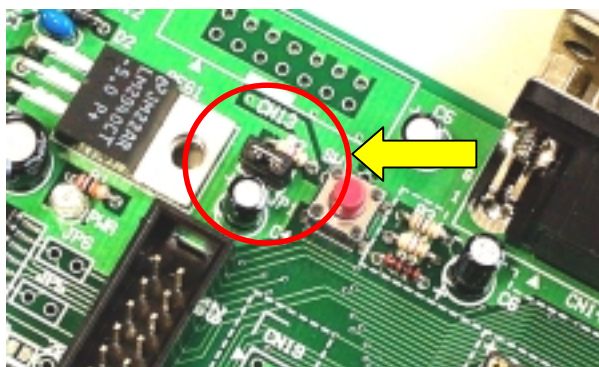


8. ここでも変更は無いので **完了** をクリックします。以上でワークスペースとプロジェクトの立ち上げは完了です。

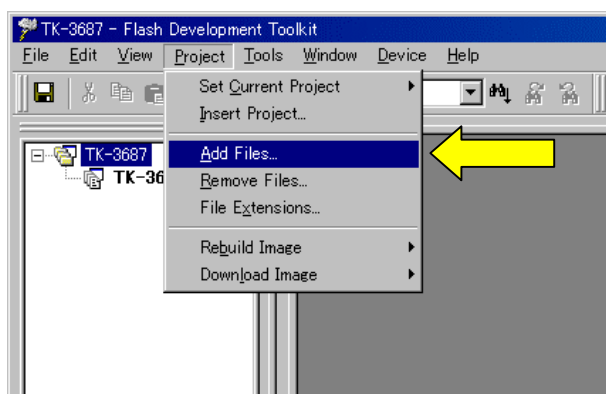



■ファイルのダウンロード

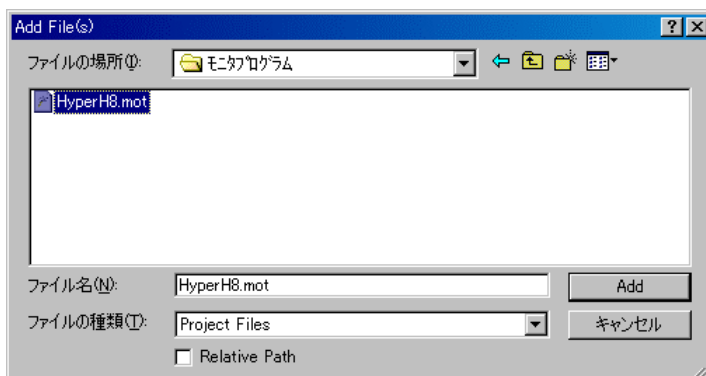
1. まず TK-3687 とパソコンとを接続します。基板上のジャンパ JP1 をワイヤ又はジャンパソケットでショートし、RS-232C ケーブルでパソコンと接続し電源を投入、又はリセットをして下さい(ブートモードで起動)。



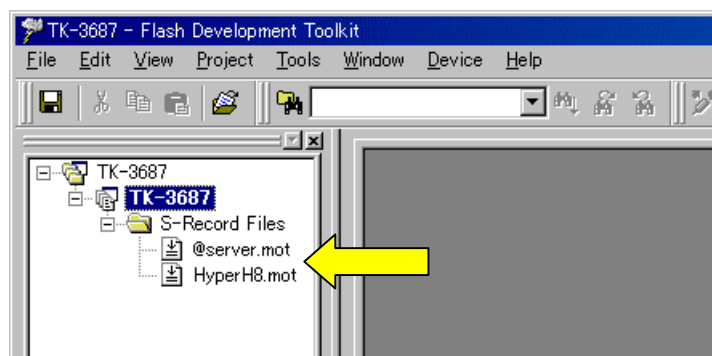
2. 次にダウンロードするファイルプロジェクトに追加します。メニューバーから“Project > Add Files...”を選択します。



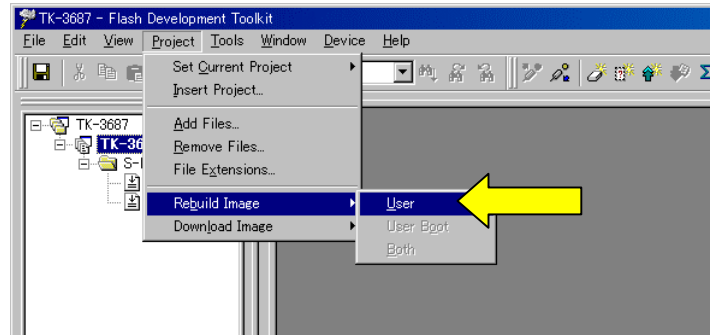
3. まずハイパーH8 のファイルを選択します(既に追加されている場合は次へ)。付属 CD-ROM にある ‘HyperH8.mot’ ファイルを選択し  をクリックして下さい。 ‘HyperH8.mot’ は次のディレクトリにあります。CD-ROM¥TK-3687¥ モニタプログラム ¥HyperH8.mot



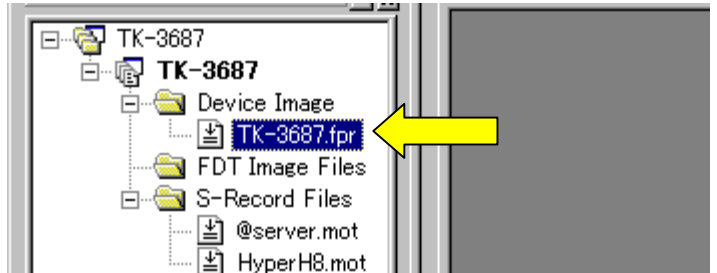
4. 画面左のルートディレクトリ内“S-Record files”に HyperH8.mot が追加されたのを確認して下さい。次にユーザプログラムを追加します。2, 3 の手順で先程変換したユーザの mot ファイルを選択し、追加して下さい。



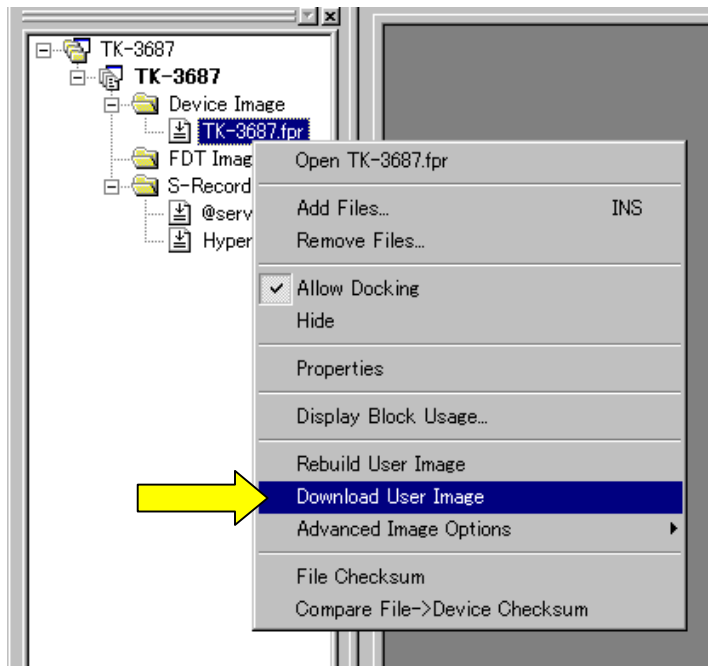
5. 次にハイパー H8 とユーザプログラムを TK-3687 にダウンロードする為、2 つのファイルを 1 つにまとめた“**デバイスイメージ**”を作成します。メニューバーから“**Project > Rebuild Image > User**”を選択して下さい。



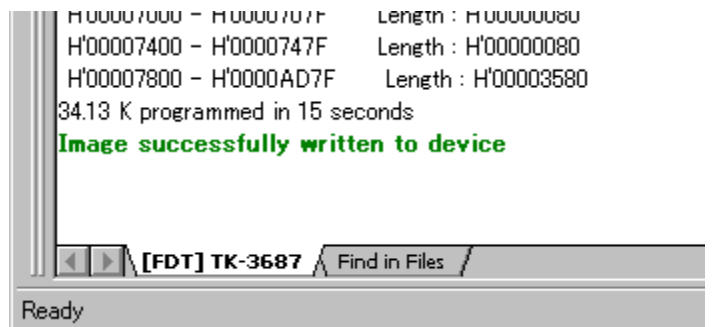
“**Image Build Succeeded:...User Image added to workspace**”のメッセージが表示され、画面左ルートディレクトリ、“**Device Image**”フォルダ下にデバイスイメージ“**(プロジェクト名).fpr**”が作成されます。(右図では TK3687.fpr です。)



6. 最後に作成したデバイスイメージを TK-3687 にダウンロードします。作成されたデバイスイメージファイルを右クリックし“**Download User Image**”を選択して下さい。



7. 右図の“**Image successfully written to device**”のメッセージが表示されれば終了です。基板のジャンパ JP1 を外し、リセットスイッチを押して下さい(通常モード)。ハイパー H8 が起動します。ハイパー H8 の実行コマンド‘G8000’で書き込んだユーザプログラムを実行してみましょう。書き込んだユーザプログラムが動き出せば作業完了です。



付録-3 部品表

TK-3687部品表

台数: 1

	部品番号	型名, 規格	メーカー	使用数	数量	備考
1	U1	H8/3687F (Flat)	ルネサス	1	1	実装済み
2	U2	MAX232ACSE (Flat)	MAXIM	1	1	実装済み
3	U3,4,5,6,7	74HC540 (Flat)		4	5	実装済み (U5を除く)
4						
5	REG1	LM2940CT-5.0	NS	1	1	
6						
7	X1	20MHz		1	1	水晶又はセラロック
8	X2	32.768kHz		1	1	
9	D1	W02		1	1	
10	D2			1	1	REG1保護
11	D3	1SS133-T72	ROHM	1	1	
12	P50-57,PB0-7 P80-87,10-17,P20 P70-76,P23,P24	HLMP-6001#A04	HP	8 ※1	10 ※1	連結LEDモジュール P80-P87は実装しません
13	PWR(LED)			1	1	パワーオン表示
14						
15	R1	4.7k~10kΩ (茶黒橙金)		1	1	PWR LED の輝度で調整
16	R2,4	4.7kΩ (黄紫赤金)		2	2	
17	R3	100Ω (茶黒茶金)		1	1	
18	RA1,3,10,12	M9-1-561~102J	BI	4	4	560~1kΩ LEDの輝度で調整
19	RA2,4,5,6,7,9,11	M9-1-103J~473J	BI	7	7	10k~47kΩ
20						
21	C11,14	22pF		1	2	※2
22	C12,13	33pF		1	2	※2
23	C1,2,8,9,15,21 23,28,29,31	0.1μF (積セラ)		10	10	
24	C17,18,19,20	0.1μF		4	4	
25	C4,5,6,7,10,16,22 24,25,26,27,30,32	10μF/16V (電解)		13	13	
26	C3,33	47~100μF/16V (電解)		2	2	
27						
28	SW1	SKHHAK/AM/DC	ALPS	1	1	
29	JP1,4-6	2pin		4	8	※3
30	丸ピンソケット	20pin		1	1	
31	CN11	DCジャック(2.1φ)		1	1	
32	CN14	D-Sub9pin		1	1	メス,ライトアングル
33	CN16,17	Box,30pin		0	2	※4
34	CN1,3,5,6,7,8 CN-B	B10P-SHF-1AA	JST	0	7	※4
35	CN12	B2P-SHF-1AA	JST	0	1	※4(5V供給用)
36	CN13	Box,14pin,ライトアングル		1	1	E7接続用
37	ゴム足	B-21	タカチ	4	4	
38						
39	ベース基板	B6081(TK-3664/3687)	東洋リンクス	1	1	
40						

※相当品を使用する場合があります

※1 8連の時は1/2個

※2 X1セラロック使用時はC11,C12実装せず

※3 丸ピンソケットを2pin×4ヶに切断して使用

※4 付属していません

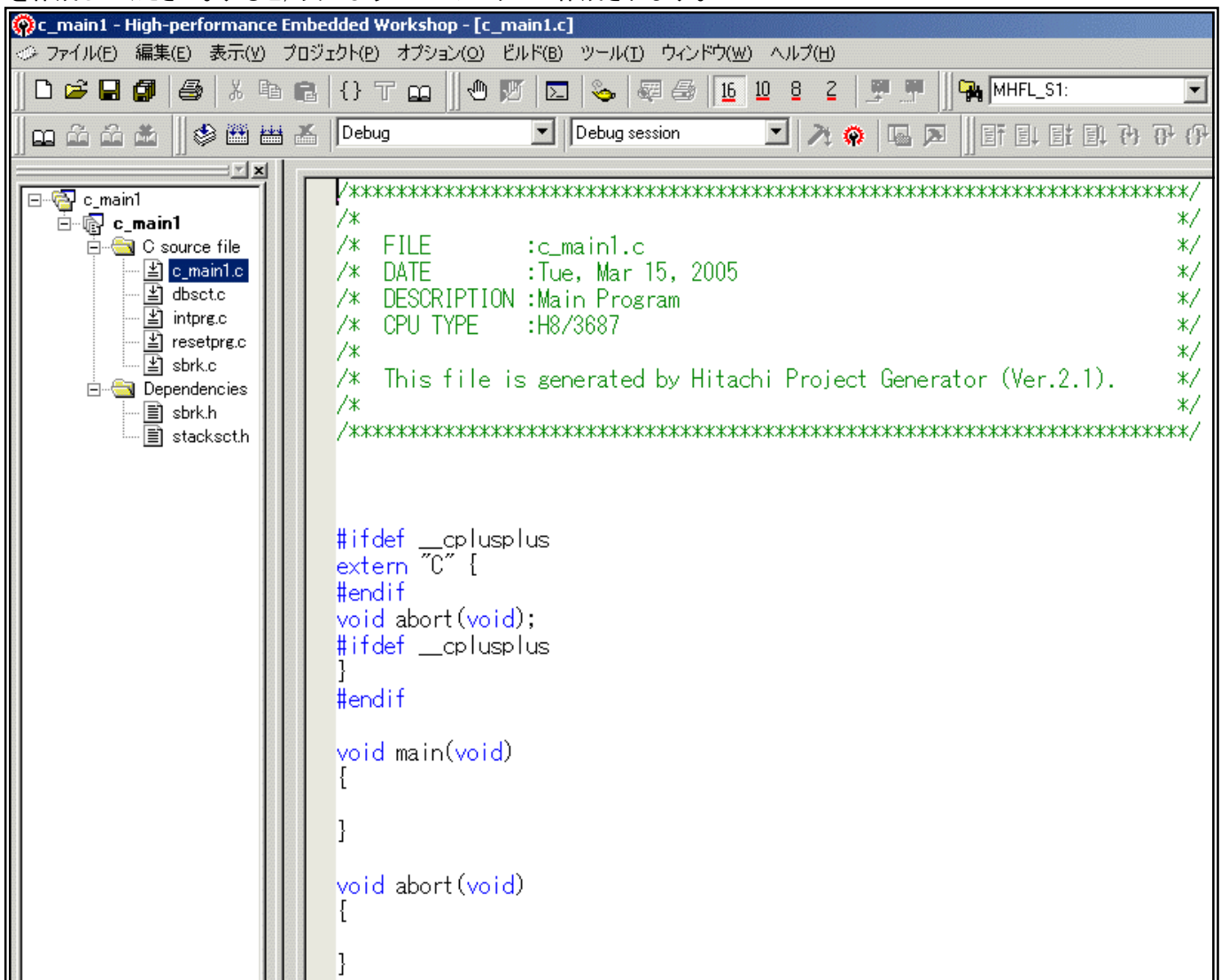
付録4 C のプログラムからアセンブラのサブルーチン呼び出す方法

HEWのC言語は#pragma, キーワードなどの拡張機能や組み込み関数をサポートすることにより, 機器組み込み用プログラムに必要な全ての機能をC言語で記述できます。しかし, ハードウェアのタイミング要求やメモリサイズの制限などのように性能要求が厳しい場合, サブルーチン(関数)をアセンブラで記述しCのプログラムから呼び出す必要が生じることもあります。ここではCのプログラムからアセンブラのサブルーチン呼び出す基本的な方法を説明します。

この項の内容については, ルネサステクノロジが配布している, 「H8S, H8/300 シリーズ C/C++コンパイラ, アセンブラ, 最適化リンケージエディタ ユーザーズマニュアル」の「9. プログラミング」, 「9. 3 C/C++プログラムとアセンブリプログラムとの結合」にさらに詳しく説明されています。この資料は HEW をインストールするとハードディスクにコピーされるか, HEW の CD-ROM に収められています。また, ルネサステクノロジのサイトからダウンロードできます。

1. C のプログラムからアセンブラのサブルーチン呼び出す方法

まずはCのプロジェクトにアセンブラのソースファイルを追加します。ここでは, プロジェクト名を 'c_main1', Cのソースファイル名を 'c_main1.c', アセンブラのソースファイル名を 'asm_sub1.src' とします。通常どおりCのプロジェクトを作成してください。すると, 次のようにプロジェクトが作成されます。



```
/*
 * FILE      :c_main1.c
 * DATE     :Tue, Mar 15, 2005
 * DESCRIPTION :Main Program
 * CPU TYPE  :H8/3687
 *
 * This file is generated by Hitachi Project Generator (Ver.2.1).
 */

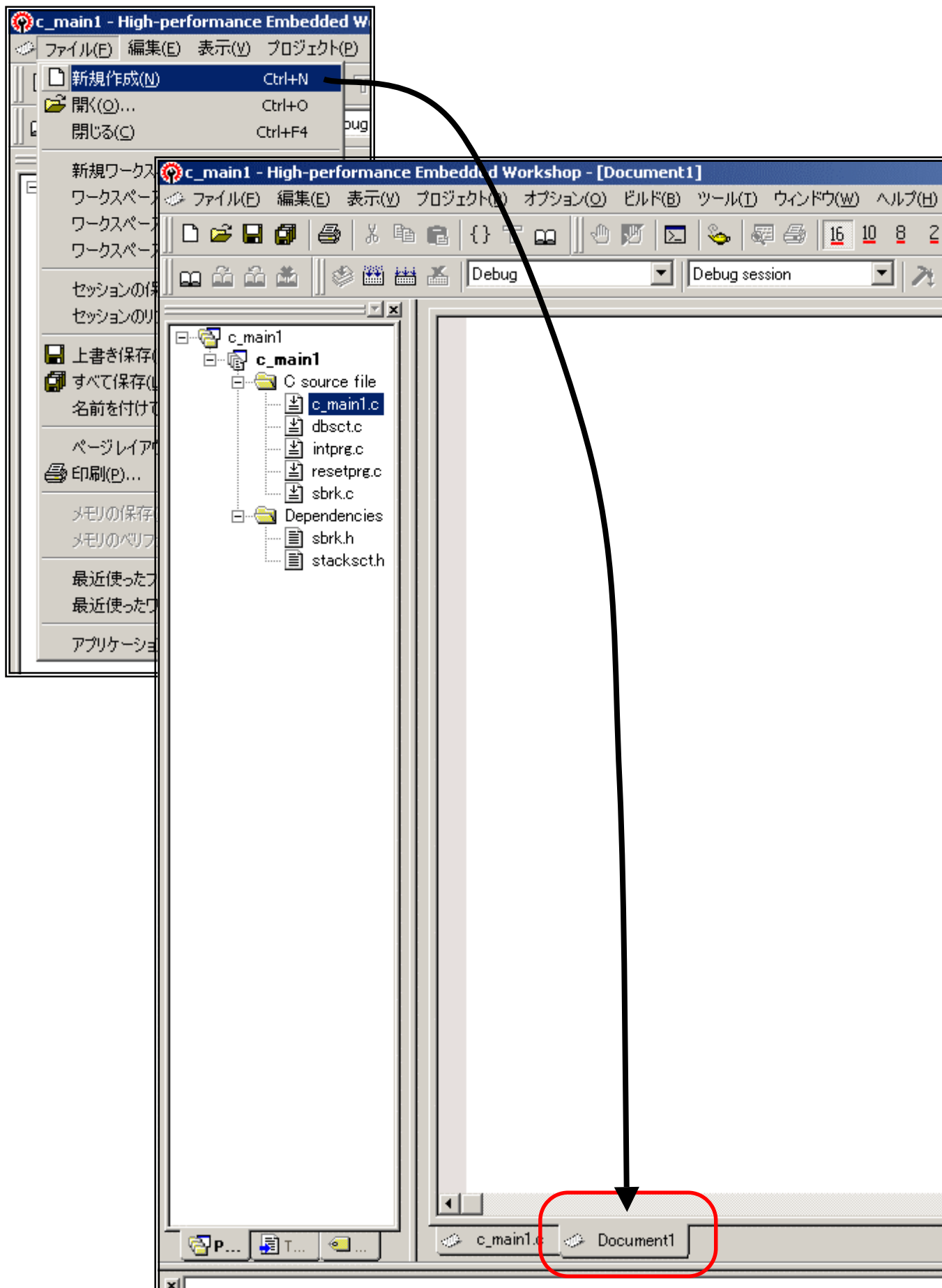
#ifdef __cplusplus
extern "C" {
#endif
void abort(void);
#ifdef __cplusplus
}
#endif

void main(void)
{

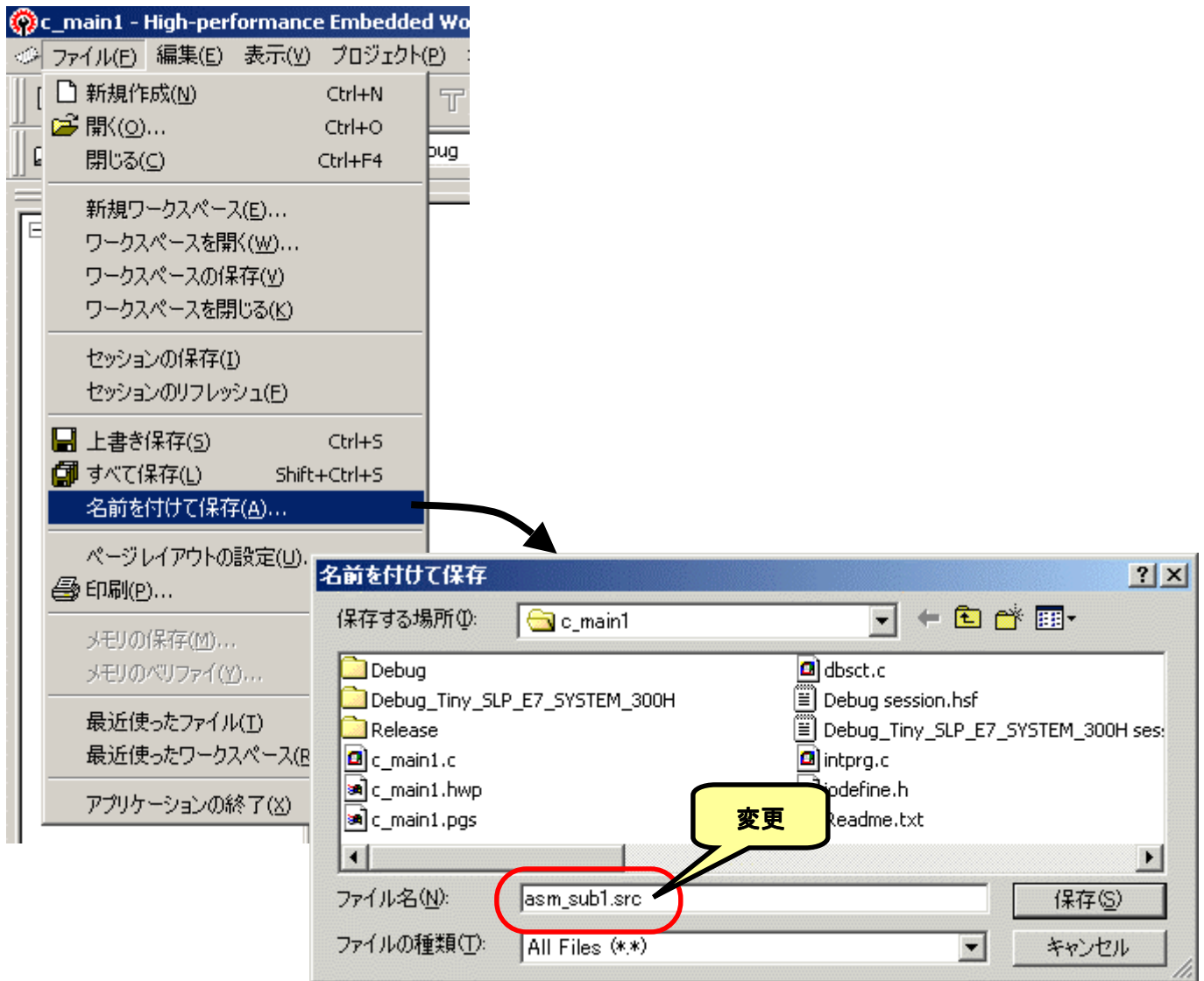
}

void abort(void)
{
}
```

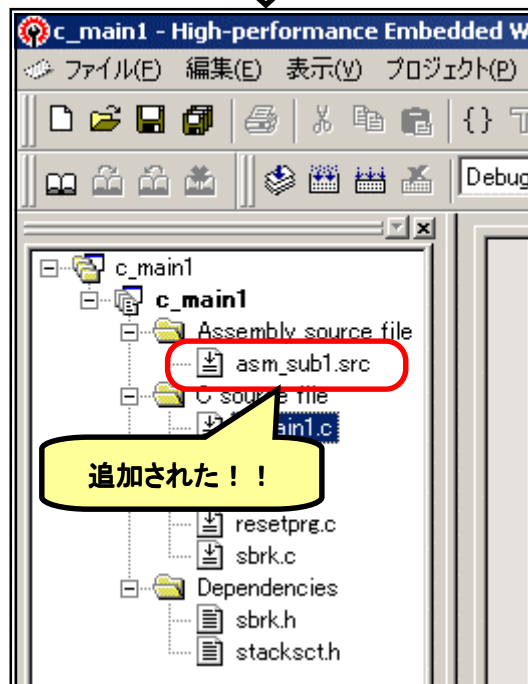
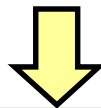
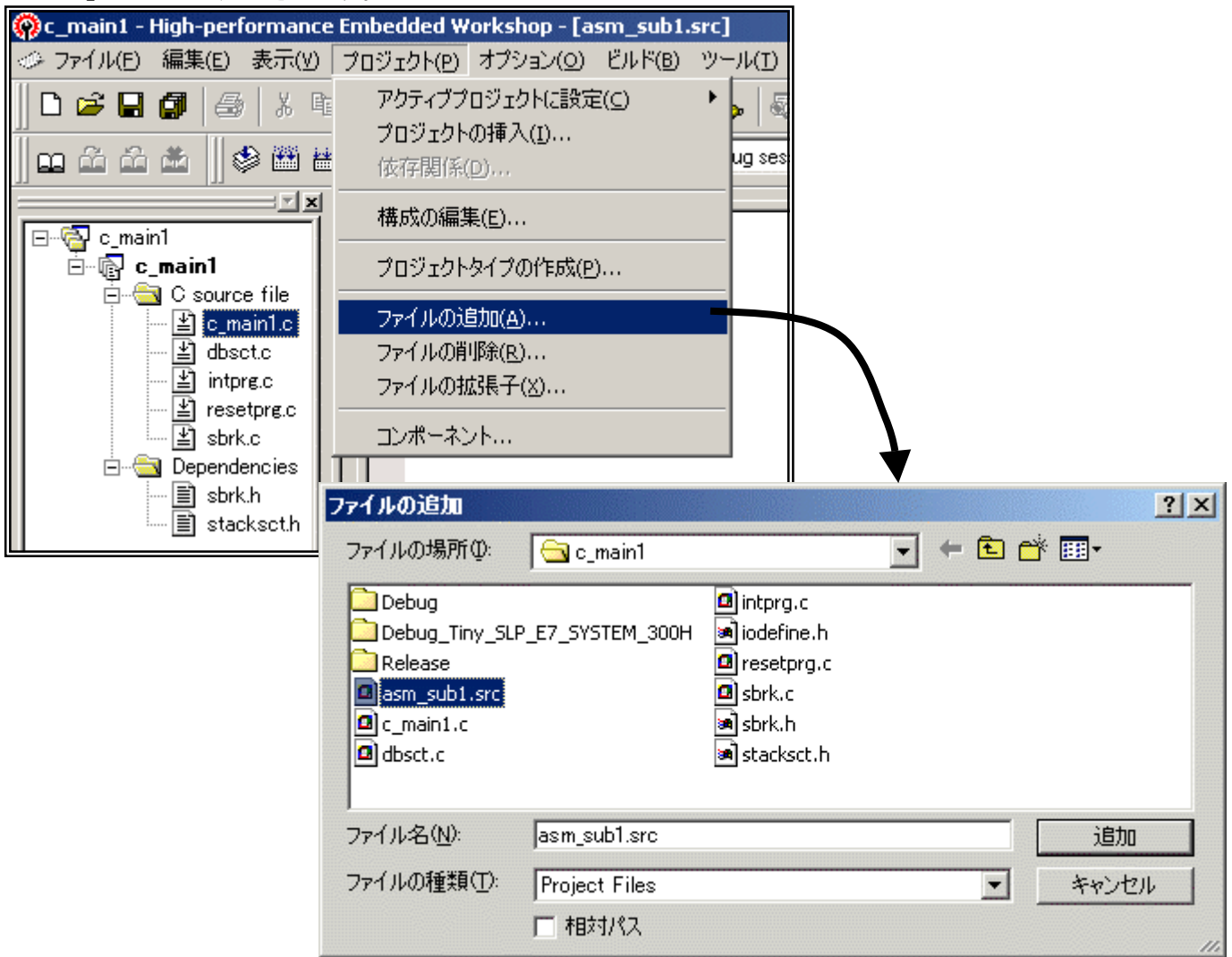
ここにアセンブラのソースファイルを追加します。‘ファイル’メニューから‘新規作成’を選ぶと‘Document1’が追加されます。



今作った 'Document1' を 'asm_sub1.src' として保存しましょう。'Document1' をアクティブにした状態で 'ファイル' メニューから '名前を付けて保存' を選ぶとダイアログが開きます。ファイル名を 'asm_sub1.src' に変更して '保存' をクリックします。



最後に今保存した 'asm_sub1.src' をプロジェクトに追加します。'プロジェクト' メニューから 'ファイルの追加' を選びます。'ファイルの追加' ダイアログが開くので 'asm_sub1.src' を選んで '追加' をクリックします。するとプロジェクトに 'asm_sub1.src' が追加されます。



あとは C とアセンブラのソースファイルにプログラムを入力します。ここでは例題として次のようなプログラムを作成します。

- C のプログラムはポート 5(P50~57)に 00h から FFh を+1 しながら順番に出力する。
- アセンブラのプログラムはポート 7(P70~77)に 00h と FFh を交互に出力する。このサブルーチンは C から呼び出す。

まずは C のプログラムです。アセンブラのサブルーチンを呼び出すからといって特に変わったところはありません。C の関数を呼び出すときと同じように、アセンブラのサブルーチンを定義、コールします。

```
/*
 *
 * FILE      :c_main1.c
 * DATE      :Tue, Mar 15, 2005
 * DESCRIPTION :Main Program
 * CPU TYPE  :H8/3687
 *
 * This file is programmed by TOYO-LINX Co.,Ltd. / yKikuchi
 */
*****

*****
    インクルードファイル
*****
#include <machine.h>    //H8特有の命令を使う
#include "iodefine.h"  //内蔵I/Oのラベル定義

*****
    関数の定義
*****
void    main(void);
void    wait(void);

void    port7_on_off(void);    //アセンブラのサブルーチンの定義

*****
    メインプログラム
*****
void main(void)
{
    IO.PCR5      = 0xff;    //ポート5を出力に設定
    IO.PDR5.BYTE = 0x00;
    IO.PCR7      = 0xff;    //ポート7を出力に設定
    IO.PDR7.BYTE = 0x00;

    while(1){
        IO.PDR5.BYTE = IO.PDR5.BYTE + 1;
        port7_on_off();    //アセンブラのサブルーチンをコール
        wait();
    }
}

*****
    ウェイト
*****
void wait(void)
{
```

```

unsigned long i;

for (i=0;i<833333;i++){
}

```

次はアセンブラのプログラムです。ポイントは①アセンブラの関数名を‘. export’で外部定義シンボル宣言すること、②Cから呼び出されるアセンブラのサブルーチン名称はCで定義した名称の先頭に‘_’（アンダーバー、アンダースコア）を付加すること、③レジスタ、ER0とER1は退避せずに(PUSHせずに)使用できるがER3～ER6は退避してから使用すること、です。

```

;-----
;
; FILE      :asm_sub1.src
; DATE      :Thu, Mar 15, 2005
; DESCRIPTION :Sub Program
; CPU TYPE  :H8/3687
;
; This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi
;
;-----

```

```

.export    _port7_on_off
.section   P, CODE, ALIGN=2
;-----
; 定数定義
;-----
PDR7      .equ    h'FFDA    ;ポ-ト-レジスタ 7

;-----
; ポート7の出力をオン オフする
;-----

```

```

_port7_on_off:
    mov.b   @PDR7, r0l
    not.b   r0l
    mov.b   r0l, @PDR7
    rts

;-----
.end

```

あとは通常どおりビルドすればCのプログラムとアセンブラのサブルーチンを結合したMOTファイルが作られます。FDTでフラッシュメモリに書いて実行することはもちろん、ハイパーH8でダウンロードしたり、E7でデバッグすることも可能です。

2. アセンブラのサブルーチンでCのプログラムのグローバル変数を参照する方法

Cのプログラムからアセンブラのサブルーチン呼び出す場合、アセンブラのサブルーチンからCのプログラムの変数を参照したり更新したりしたい場合があります。例題として次のようなプログラムを考えてみましょう

- Cのプログラムは'Port5_Data'の内容をポート5(P50~57)に出力する。
- アセンブラのプログラムは'Port5_Data'の内容を左ローテートする。このサブルーチンはCから呼び出す。

Cのプログラムは特に変わったところはありません。普通どおりグローバル変数を定義します。

```
/*
 *
 * FILE      :c_main2.c
 * DATE      :Thu, Mar 10, 2005
 * DESCRIPTION :Main Program
 * CPU TYPE  :H8/3687
 *
 * This file is programmed by TOYO-LINX Co.,Ltd. / yKikuchi
 *
 */

*****
          インクルードファイル
*****
#include <machine.h>      //H8特有の命令を使う
#include "iodefine.h"    //内蔵I/Oのラベル定義

*****
          グローバル変数の定義とイニシャライズ(RAM)
*****
unsigned char   Port5_Data =0x01;    //ポート5に出力するデータ

*****
          関数の定義
*****
void   main(void);
void   wait(void);

void   rotl_Port5_Data(void);        //アセンブラのサブルーチン

*****
          メインプログラム
*****
void main(void)
{
    IO.PCR5      = 0xff;        //ポート5を出力に設定
    IO.PDR5.BYTE = Port5_Data;
    while(1){
        rotl_Port5_Data();        //Port5_Dataを左ローテート
        IO.PDR5.BYTE = Port5_Data;
        wait();
    }
}

*****
```

```

ウェイト
*****/
void wait(void)
{
    unsigned long i;

    for (i=0;i<833333;i++){
    }
}

```

次はアセンブラのプログラムです。ポイントは①C の変数名を‘. import’ で外部参照シンボル宣言すること, ②変数の名称は C で定義した名称の先頭に‘_’ (アンダーバー, アンダースコア)を付加すること, です。

```

;-----
;
; FILE      :asm_sub2.src
; DATE      :Thu, Mar 10, 2005
; DESCRIPTION :Sub Program
; CPU TYPE  :H8/3687
;
; This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi
;-----

        .export    _rotl_Port5_Data
.import    _Port5_Data
        .section   P, CODE, ALIGN=2
;-----
;
; Port5_Dataを左ローテートする
;-----
_rotl_Port5_Data:
mov.b    @_Port5_Data, r0l
rotl.b   r0l
mov.b    r0l, @_Port5_Data
rts

;-----
        .end

```

あとは通常どおりビルドすれば OK です。

3. アセンブラのサブルーチンにCのプログラムから引数を渡す方法

Cのプログラムでは関数に値を引数で渡します。同じようにCのプログラムからアセンブラのサブルーチン呼び出す場合でも引数を渡すことができます。例題として次のようなプログラムを考えてみましょう

- Cのプログラムは'Port5_Data'の内容とポート5のアドレスをアセンブラのサブルーチンに引数として渡す。
- アセンブラのプログラムは指定されたアドレスに指定されたデータをセットする。

Cのプログラムは特に変わったところはありません。通常どおり関数の定義で引数の型を指定し、関数を呼び出すときはその定義どおり引数をセットします。

```
/*
 *
 * FILE      :c_main3.c
 * DATE      :Fri, Mar 11, 2005
 * DESCRIPTION :Main Program
 * CPU TYPE  :H8/3687
 *
 * This file is programmed by TOYO-LINX Co.,Ltd. / yKikuchi
 *
 */

*****
          インクルードファイル
*****
#include <machine.h>      //H8特有の命令を使う
#include "iodefine.h"    //内蔵I/Oのラベル定義

*****
          グローバル変数の定義とイニシャライズ(RAM)
*****
unsigned char   Port5_Data =0x00;    //ポート5に出力するデータ

*****
          関数の定義
*****
void   main(void);
void   wait(void);

void   out_port(unsigned char *,unsigned char); //アセンブラのサブルーチン

*****
          メインプログラム
*****
void main(void)
{
    IO.PCR5      = 0xff;      //ポート5を出力に設定
    IO.PDR5.BYTE = Port5_Data;
    while(1){
        Port5_Data++;
        out_port(&IO.PDR5.BYTE,Port5_Data); //ポート5に出力
        wait();
    }
}
```

```

/*****
   ウェイト
 *****/
void wait(void)
{
    unsigned long l;

    for (l=0;l<833333;l++){
    }
}

```

引数は基本的にレジスタ, ER0 と ER1 に割り付けられます。割り付けの順番は, ソースプログラムの宣言順に, 番号の小さいレジスタの LSB 側から割り付けられます。例題の場合, 最初の引数はポインタ型なので, ノーマルモードの場合サイズは 2 バイトです。よって R0 レジスタに割り付けられます。次の引数は unsigned char 型なのでサイズは 1 バイトです。レジスタの順番からすると E0 レジスタなのですが, E0 レジスタのサイズは 2 バイトのため割り付けることができません。そこで, その次の R1L レジスタに割り付けられます。それで, アセンブラのプログラムは次のようになります。

```

;-----
;
;
; FILE      :asm_sub3.src
; DATE      :Thu, Mar 10, 2005
; DESCRIPTION :Sub Program
; CPU TYPE  :H8/3687
;
; This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi
;-----
;
; .export   _out_port
; .section  P,CODE,ALIGN=2
;-----
; (R0)にR1Lのデータをセットする
;-----
_out_port:
    mov.b   r1l,@er0 ;ノーマルモードなのでE0は何でもよい
    rts
;-----
;
; .end

```

なお, 引数の数が多い場合など, ER0, ER1 だけでは足りない場合はスタックに割り付けられます。詳細は「H8S, H8/300 シリーズ C/C++コンパイラ, アセンブラ, 最適化リンケージエディタ ユーザーズマニュアル」をご覧ください。特に, 「9. 3. 3 引数割り付けの具体例」に様々なケースが例として載せられていて参考になります。

4. アセンブラのサブルーチンからCのプログラムにリターン値を渡す方法

Cのプログラムの関数はリターン値を設定することができます。同じようにアセンブラのサブルーチンもリターン値を設定することができます。例題として次のようなプログラムを考えてみましょう

- Cのプログラムはポート5のデータをアセンブラのサブルーチンに引数として渡し、リターン値をポート5に出力する。
- アセンブラのプログラムはデータを左ローテートしてリターン値として渡す。

Cのプログラムは特に変わったところはありません。通常どおり関数の定義でリターン値の型を指定します。

```
/*
 *
 * FILE      :c_main4.c
 * DATE      :Fri, Mar 11, 2005
 * DESCRIPTION :Main Program
 * CPU TYPE  :H8/3687
 *
 * This file is programmed by TOYO-LINX Co.,Ltd. / yKikuchi
 */

*****
          インクルードファイル
*****
#include <machine.h>      //H8特有の命令を使う
#include "iodefine.h"    //内蔵I/Oのラベル定義

*****
          関数の定義
*****
void main(void);
void wait(void);

unsigned char  rotl_data(unsigned char); //アセンブラのサブルーチン

*****
          メインプログラム
*****
void main(void)
{
    IO.PCR5      = 0xff;      //ポート5を出力に設定
    IO.PDR5.BYTE = 0x01;
    while(1){
        IO.PDR5.BYTE = rotl_data(IO.PDR5.BYTE); //ポート5に出力
        wait();
    }
}

*****
          ウェイト
*****
void wait(void)
{
    unsigned long i;
```



```
    for (i=0; i<833333; i++){  
    }
```

リターン値は型に応じて R0L, R0, ER0 にセットします。例題の場合、関数のリターン値は unsigned char 型なので 1 バイトです。それで、R0L レジスタにセットします。下記のプログラムでは引数は R0L レジスタにセットされてアセンブラのサブルーチンに渡されます。それで、R0L レジスタを左ローテートしてリターンすれば、C のプログラムにリターン値として渡されます。

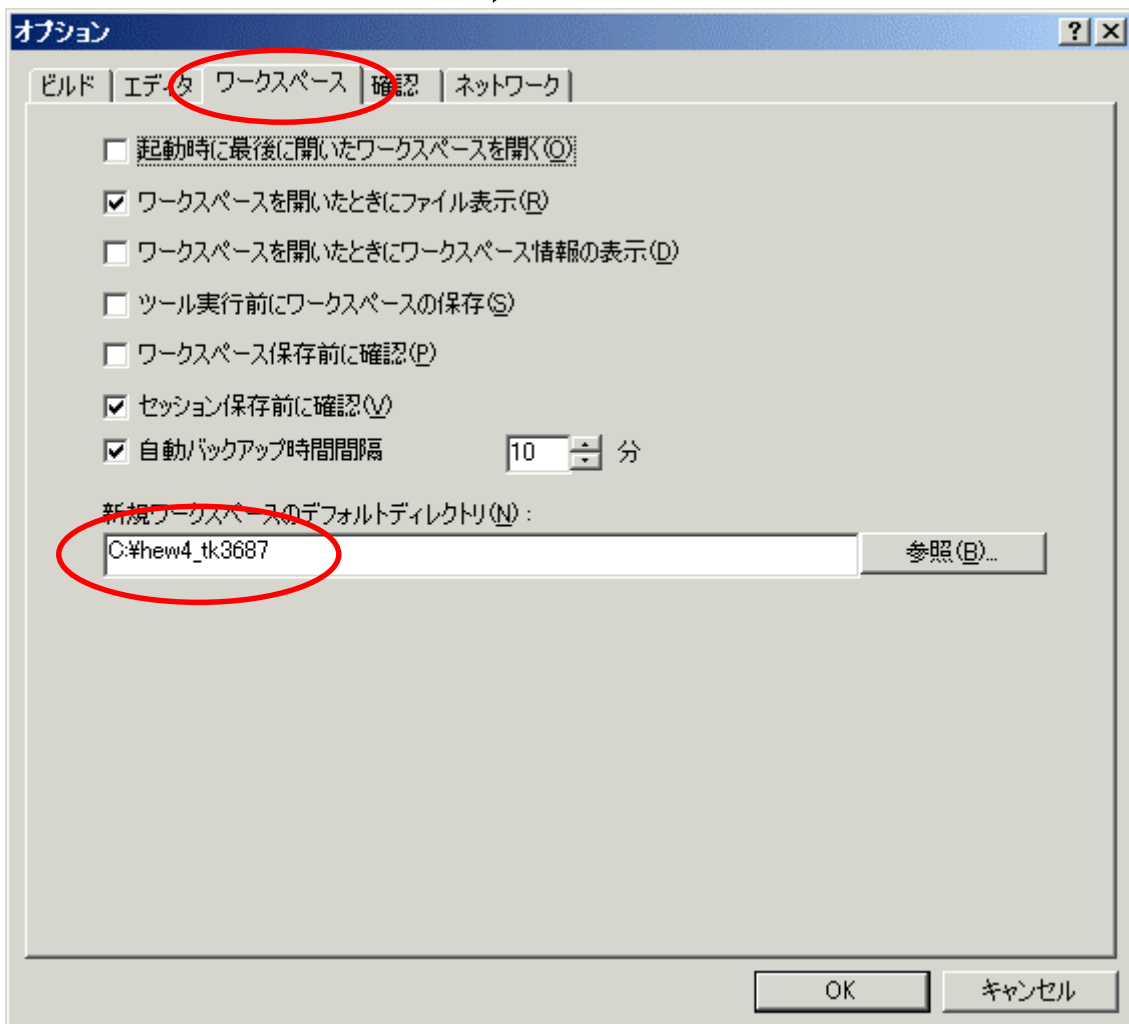
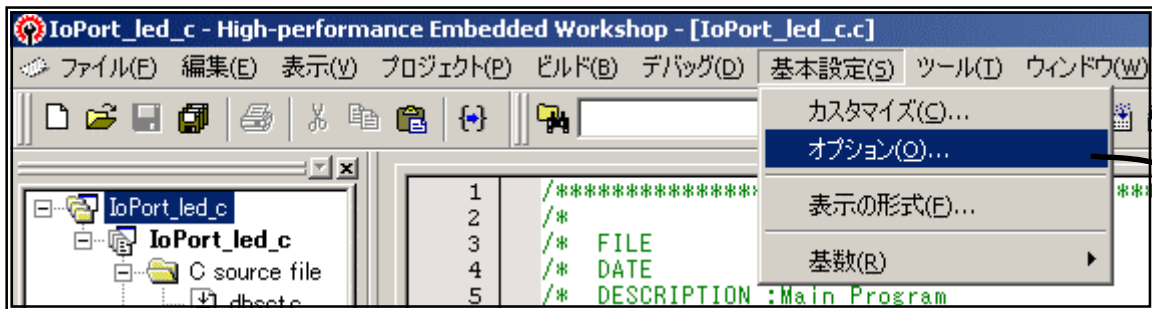
```
-----  
;  
;  
; FILE      :asm_sub4.src  
; DATE      :Thu, Mar 10, 2005  
; DESCRIPTION :Sub Program  
; CPU TYPE   :H8/3687  
;  
; This file is programmed by TOYO-LINX Co.,Ltd. / yKikuchi  
;  
-----  
  
    .export    _rotl_data  
    .section   P, CODE, ALIGN=2  
; *****  
; R0Lを左ローテートする  
; *****  
_rotl_data:  
    rotl.b   r0l  
    rts  
  
;-----  
    .end
```

なお、ER0 レジスタにセットしきれない場合はメモリ経由でリターン値を渡すこととなります。詳細は「H8S, H8/300 シリーズ C/C++コンパイラ, アセンブラ, 最適化リンケージエディタ ユーザーズマニュアル」をご覧ください。

付録-5 HEW の便利な設定

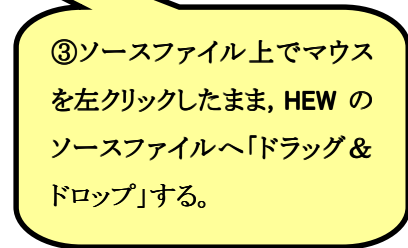
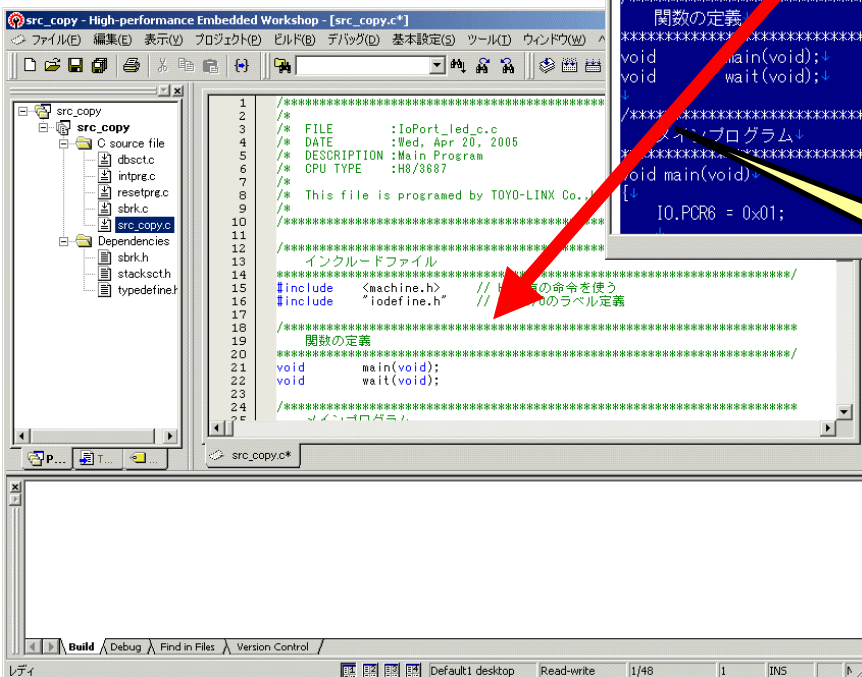
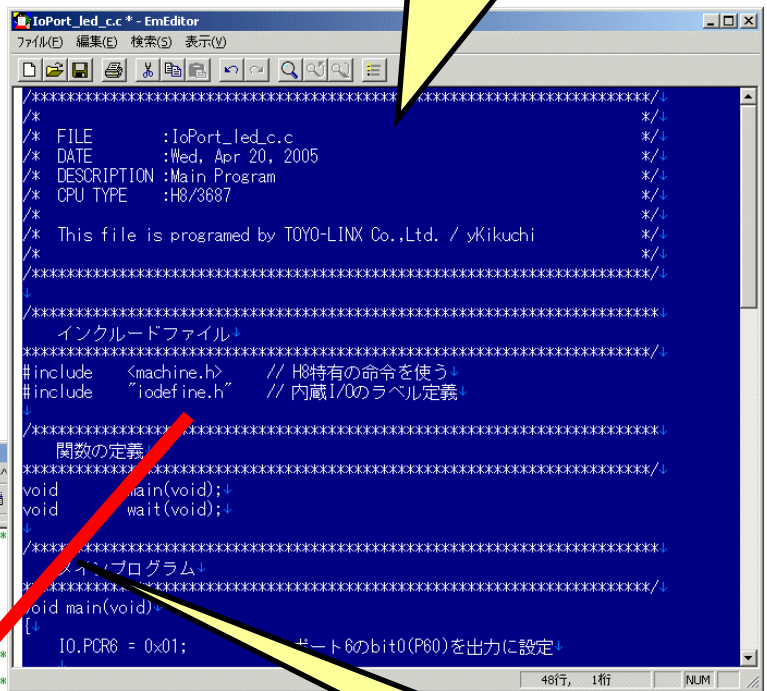
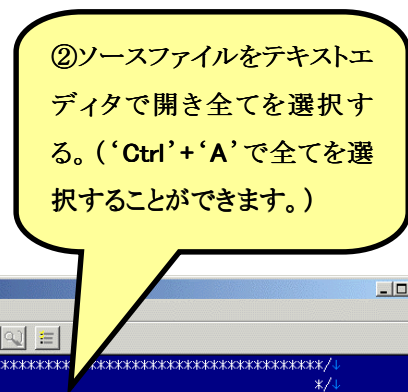
■ ワークスペースの指定

ワークスペースのデフォルトディレクトリを指定することができます。一回指定すると二回目以降はこのディレクトリがデフォルトになります。



付録-6 ソースファイルの入力方法

ソースファイルの内容を入力する際、リストしかないときはキーボードで入力するしかありません。しかし、ソースファイルがあればコピーしたあと貼り付けることができます。この方法は別ファイルの一部コピーや HEW のバージョンが異なる場合にも有効です。



付録-7 “iodefine. h”一覧

HEW でプロジェクトを作成すると、自動的に“iodefine. h”が生成されます。通常 H8/3687 の I/O にアクセスする際、“iodefine. h”で定義されている名称を使います。しかし、“iodefine. h”は構造体や共用体を駆使して定義されているため、最初はとっつきにくく感じるかもしれません。それでも慣れてくると非常に便利です。この項では、“iodefine. h”のリスト上の記述を表にまとめてみました。タイマ V を参考に、表の見方とソースリスト上でどのように記述すれば目的のレジスタをアクセスできるか説明します。

“iodefine. h”は HEW のバージョンによって変更されることがあります。ソースリストを記述する前に変更されていないか確認して下さい。なお、この資料は、ルネサステクノロジが 2005 年 6 月 7 日に公開した「C/C++コンパイラパッケージ無償評価版 V. 6. 01 Release 00」を対象としています。

“iodefine. h”でタイマ V は次のように定義されています。(黄色で塗りつぶしている行はあとの説明で使う部分)

```

struct st_tv {
    union {
        unsigned char BYTE;
        struct {
            unsigned char CMIEB:1;
            unsigned char CMIEA:1;
            unsigned char OVIE :1;
            unsigned char CCLR :2;
            unsigned char CKS  :3;
        } BIT;
    } TCRVO;

    union {
        unsigned char BYTE;
        struct {
            unsigned char CMFB:1;
            unsigned char CMFA:1;
            unsigned char OVF :1;
            unsigned char   :1;
            unsigned char OS  :4;
        } BIT;
    } TCSR;

    unsigned char TCORA;
    unsigned char TCORB;
    unsigned char TCNTV;
    union {
        unsigned char BYTE;
        struct {
            unsigned char   :3;
            unsigned char TVEG:2;
            unsigned char TRGE:1;
            unsigned char   :1;
            unsigned char ICKS:1;
        } BIT;
    } TCRV1;
};

#define TV (*(volatile struct st_tv *)0xFFA0) /* TV Address*/

```

それではタイマコントロールレジスタ V0 に値をセットしましょう。セットする値は 4Bh です。このマニュアルでは“iodefine. h”を参考にタイマコントロールレジスタ V0(TCRV0)は次のように表現しています。

タイマ V: タイマコントロールレジスタ V0									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TV	TCRV0	CMIEB	CMIEA	OVIE	CCLR			CKS	

ソースリストから分かるように TCRV0 は共用体を使ってバイトアクセスとビットアクセスができるように定義しています。ここでは TCRV0 に 4Bh をセットするのでバイトアクセスです。次のように記述します。

```
TV.TCRV0.BYTE = 0x4b;
```

さて、コンペアマッチインタラプトイネーブル A(CMIEA)だけを一時的に 0 にする場合はどうでしょうか。今度は特定のビットだけを書き替えるのでビットアクセスです。次のように記述します。

```
TV.TCRV0.BIT.CMIEA = 0;
```

また、クロックセレクト(CKS)だけを現在の 3 から 2 に変更する場合もビットアクセスなので次のように記述します。

```
TV.TCRV0.BIT.CKS = 2;
```

では次に、タイムコンスタントレジスタ A(TCOR A)に値をセットしてみましょう。セットする値は 100 です。このマニュアルでは“iodefine. h”を参考にタイムコンスタントレジスタ A は次のように表現しています。

タイマ V: タイムコンスタントレジスタ A									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TV	TCORA	(ビットアクセスは未定義)							

ソースリストから分かるように TCORA は共用体を使ってビットアクセスするようには定義されていません。それで常に 1 バイト単位でアクセスします。ここでは TCORA に 100 をセットするので次のように記述します。

```
TV.TCOR A = 100;
```

なお、16 ビットで定義されているレジスタもあります(例:タイマ Z のタイマカウンタ_0)。このマニュアルでは次のように表現します。

タイマ Z: タイマカウンタ_0															
モジュール名称	レジスタ名称	ビット名称													
		15	14	13	12	11	10	9	8	7	6	5	4	3	2
TZ0	TCNT	(ビットアクセスは未定義)													



次のページから“iodefine. h”の定義を表にしたものを掲載します。

I/O ポート

■ ポート 1

I/O ポート: ポートモードレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PMR1	IRQ3	IRQ2	IRQ1	IRQ0	TXD2	PWM	TXD	TMOW

I/O ポート: ポートコントロールレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PCR1	(ビットアクセスは未定義)							

I/O ポート: ポートデータレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PDR1	B7	B6	B5	B4	-	B2	B1	B0

I/O ポート: ポートプルアップコントロールレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PUCR1	B7	B6	B5	B4	-	B2	B1	B0

■ ポート 2

I/O ポート: ポートデータレジスタ 2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PDR2	-	-	-	B4	B3	B2	B1	B0

I/O ポート: ポートコントロールレジスタ 2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PCR2	(ビットアクセスは未定義)							

I/O ポート: ポートモードレジスタ 3

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PMR3	-	-	-	POF24	POF23	-	-	-

!!! PMR3 となっていますがポート 2 のモードを設定するレジスタです。(ルネサスの定義による) !!!

■ ポート 3

I/O ポート: ポートコントロールレジスタ 3									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PCR3	(ビットアクセスは未定義)							

I/O ポート: ポートデータレジスタ 3									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PDR3	B7	B6	B5	B4	B3	B2	B1	B0

■ ポート 5

I/O ポート: ポートモードレジスタ 5									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PMR5	POF57	POF56	WKP5	WKP4	WKP3	WKP2	WKP1	WKP0

I/O ポート: ポートコントロールレジスタ 5									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PCR5	(ビットアクセスは未定義)							

I/O ポート: ポートデータレジスタ 5									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PDR5	B7	B6	B5	B4	B3	B2	B1	B0

I/O ポート: ポートプルアップコントロールレジスタ 5									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PUCR5	-	-	B5	B4	B3	B2	B1	B0

■ ポート 6

I/O ポート: ポートコントロールレジスタ 6									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PCR6	(ビットアクセスは未定義)							

I/O ポート: ポートデータレジスタ 6									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PDR6	B7	B6	B5	B4	B3	B2	B1	B0

■ ポート 7

I/O ポート: ポートコントロールレジスタ 7									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PCR7	(ビットアクセスは未定義)							

I/O ポート: ポートデータレジスタ 7									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PDR7	-	B6	B5	B4	-	B2	B1	B0

■ ポート 8

I/O ポート: ポートコントロールレジスタ 8									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PCR8	(ビットアクセスは未定義)							

I/O ポート: ポートデータレジスタ 8									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PDR8	B7	B6	B5	-	-	-	-	-

■ ポート B

I/O ポート: ポートデータレジスタ B									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PDRB	B7	B6	B5	B4	B3	B2	B1	B0

タイマ B1

タイマ B1: タイマモードレジスタ B1									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TB1	TMB1	RLD	-	-	-	-	CKS		

タイマ B1: タイマカウンタ B1									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TB1	TCB1	(ビットアクセスは未定義)							

タイマ V

タイマ V: タイマコントロールレジスタ V0

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TV	TCRV0	CMIEB	CMIEA	OVIE	CCLR		CKS		

タイマ V: タイマコントロール/ステータスレジスタ V

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TV	TCSR V	CMFB	CMFA	OVF	-	OS			

タイマ V: タイムコンスタントレジスタ A

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TV	TCORA	(ビットアクセスは未定義)							

タイマ V: タイムコンスタントレジスタ B

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TV	TCORB	(ビットアクセスは未定義)							

タイマ V: タイマカウンタ V

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TV	TCNTV	(ビットアクセスは未定義)							

タイマ V: タイマコントロールレジスタ V1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TV	TCRV1	-	-	-	TVEG		TRGE	-	ICKS

タイマ Z

■ 共通

タイマ Z: タイマスタートレジスタ

モジュール名称	レジスタ名称	ビット名称								
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
TZ	TSTR	-	-	-	-	-	-	-	STR1	STR0

タイマ Z: タイマモードレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ	TMDR	BFD1	BFC1	BFD0	BFC0	-	-	-	SYNC

タイマ Z: タイマ PWM モードレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ	TPMR	-	PWMD1	PWMC1	PWMB1	-	PWMD0	PWMC0	PWMB0

タイマ Z: タイマファンクションコントロールレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ	TFCR	-	STCLK	ADEG	ADTRG	OLS1	OLS0	CMD	

タイマ Z: タイマアウトプットマスタイネーブルレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ	TOER	ED1	EC1	EB1	EA1	ED0	EC0	EB0	EA0

タイマ Z: タイマアウトプットコントロールレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ	TOCR	TOD1	TOC1	TOB1	TOA1	TOD0	TOC0	TOB0	TOA0

■ チャンネル 0

タイマ Z: タイマコントロールレジスタ_0									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ0	TCR	CCLR			CKEG		TPSC		

タイマ Z: タイマ I/O コントロールレジスタ A_0									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ0	TIORA	-	IOB			-	IOA		

タイマ Z: タイマ I/O コントロールレジスタ C_0									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ0	TIORC	-	IOD			-	IOC		

タイマ Z: タイマステータスレジスタ_0									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ0	TSR	-	-	-	OVF	IMFD	IMFC	IMFB	IMFA

タイマ Z: タイマインターラプトイネーブルレジスタ_0									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ0	TIER	-	-	-	OVIE	IMIED	IMIEC	IMIEB	IMIEA

タイマ Z: PWM モードアウトプットレベルコントロールレジスタ_0									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ0	POCR	-	-	-	-	-	POLD	POLC	POLB

タイマ Z: タイマカウンタ_0															
モジュール名称	レジスタ名称	ビット名称													
		15	14	13	12	11	10	9	8	7	6	5	4	3	2
TZ0	TCNT	(ビットアクセスは未定義)													

タイマ Z: ジェネラルレジスタ A_0															
モジュール名称	レジスタ名称	ビット名称													
		15	14	13	12	11	10	9	8	7	6	5	4	3	2
TZ0	GRA	(ビットアクセスは未定義)													

タイマ Z: ジェネラルレジスタ B_0																
モジュール名称	レジスタ名称	ビット名称														
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
TZ0	GRB	(ビットアクセスは未定義)														

タイマ Z: ジェネラルレジスタ C_0																
モジュール名称	レジスタ名称	ビット名称														
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
TZ0	GRC	(ビットアクセスは未定義)														

タイマ Z: ジェネラルレジスタ D_0																
モジュール名称	レジスタ名称	ビット名称														
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
TZ0	GRD	(ビットアクセスは未定義)														

■ チャネル 1

タイマ Z: タイマコントロールレジスタ_1									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ1	TCR	CCLR				CKEG		TPSC	

タイマ Z: タイマ I/O コントロールレジスタ A_1									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ1	TIORA	-	IOB			-	IOA		

タイマ Z: タイマ I/O コントロールレジスタ C_1									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ1	TIORC	-	IOD			-	IOC		

タイマ Z: タイマステータスレジスタ_1									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ1	TSR	-	-	UDF	OVF	IMFD	IMFC	IMFB	IMFA

タイマ Z: タイマインターラプトイネーブルレジスタ_1									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ1	TIER	-	-	-	OVIE	IMIED	IMIEC	IMIEB	IMIEA

タイマ Z: PWM モードアウトプットレベルコントロールレジスタ_1									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TZ1	POCR	-	-	-	-	-	POLD	POLC	POLB

タイマ Z: タイマカウンタ_1															
モジュール名称	レジスタ名称	ビット名称													
		15	14	13	12	11	10	9	8	7	6	5	4	3	2
TZ1	TCNT	(ビットアクセスは未定義)													

タイマ Z: ジェネラルレジスタ A_1															
モジュール名称	レジスタ名称	ビット名称													
		15	14	13	12	11	10	9	8	7	6	5	4	3	2
TZ1	GRA	(ビットアクセスは未定義)													

タイマ Z: ジェネラルレジスタ B_1															
モジュール名称	レジスタ名称	ビット名称													
		15	14	13	12	11	10	9	8	7	6	5	4	3	2
TZ1	GRB	(ビットアクセスは未定義)													

タイマ Z: ジェネラルレジスタ C_1															
モジュール名称	レジスタ名称	ビット名称													
		15	14	13	12	11	10	9	8	7	6	5	4	3	2
TZ1	GRC	(ビットアクセスは未定義)													

タイマ Z: ジェネラルレジスタ D_1															
モジュール名称	レジスタ名称	ビット名称													
		15	14	13	12	11	10	9	8	7	6	5	4	3	2
TZ1	GRD	(ビットアクセスは未定義)													

リアルタイムクロック(RTC)

リアルタイムクロック(RTC)： 秒データレジスタ/フリーランカウンタデータレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
RTC	RSECDR	BSY	SC1			SC0			

リアルタイムクロック(RTC)： 分データレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
RTC	RMINDR	BSY	MN1			MN0			

リアルタイムクロック(RTC)： 時データレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
RTC	RHRDR	BSY	HR1			HR0			

リアルタイムクロック(RTC)： 曜日データレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
RTC	RWKDR	BSY	-	-	-	WK			

リアルタイムクロック(RTC)： RTCコントロールレジスタ1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
RTC	RTCCR1	RUN	MD	PM	RST	-	-	-	-

リアルタイムクロック(RTC)： RTCコントロールレジスタ2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
RTC	RTCCR2	-	-	FOIE	WKIE	DYIE	HRIE	MNIE	SEIE

リアルタイムクロック(RTC)： クロックソースセレクトレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
RTC	RTCCSR	-	CKSO			-	CKSI		

ウォッチドッグタイマ

ウォッチドッグタイマ: タイマコントロール/ステータスレジスタ WD

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
WDT	TCSRWD	B6WI	TCWE	B4WI	TCSRWE	B2WI	WDON	BOWI	WRST

ウォッチドッグタイマ: タイマカウンタ WD

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
WDT	TCWD	(ビットアクセスは未定義)							

ウォッチドッグタイマ: タイマモードレジスタ WD

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
WDT	TMWD	-	-	-	-	CKS			

14ビット PWM

14ビット PWM: PWM コントロールレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
PWM	PWCR	-	-	-	-	-	-	-	CKS

14ビット PWM: PWM データレジスタ U

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
PWM	PWDRU	(ビットアクセスは未定義)							

14ビット PWM: PWM データレジスタ L

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
PWM	PWDRL	(ビットアクセスは未定義)							

シリアルコミュニケーションインターフェース 3 (SCI3)

■ チャネル 1

SCI3: レシーブデータレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3	RDR	(ビットアクセスは未定義)							

SCI3: トランスミットデータレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3	TDR	(ビットアクセスは未定義)							

SCI3: シリアルモードレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3	SMR	COM	CHR	PE	PM	STOP	MP	CKS	

SCI3: シリアルコントロールレジスタ 3

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3	SCR3	TIE	RIE	TE	RE	MPIE	TEIE	CKE	

SCI3: シリアルステータスレジスタ 3

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3	SSR	TDRE	RDRF	OER	FER	PER	TEND	MPBR	MPBT

SCI3: ビットレートレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3	BRR	(ビットアクセスは未定義)							

■ チャンネル 2

SCI3_2: レシーブデータレジスタ_2									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3_2	RDR	(ビットアクセスは未定義)							

SCI3_2: トランスミットデータレジスタ_2									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3_2	TDR	(ビットアクセスは未定義)							

SCI3_2: シリアルモードレジスタ_2									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3_2	SMR	COM	CHR	PE	PM	STOP	MP	CKS	

SCI3_2: シリアルコントロールレジスタ 3_2									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3_2	SCR3	TIE	RIE	TE	RE	MPIE	TEIE	CKE	

SCI3_2: シリアルステータスレジスタ 3_2									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3_2	SSR	TDRE	RDRF	OER	FER	PER	TEND	MPBR	MPBT

SCI3_2: ビットレートレジスタ_2									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SCI3_2	BRR	(ビットアクセスは未定義)							

I²C バスインターフェース 2 (IIC2)

IIC2: I²C バスコントロールレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IIC2	ICCR1	ICE	RCVD	MST	TRS	CKS			

IIC2: I²C バスコントロールレジスタ 2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IIC2	ICCR2	BBSY	SCP	SDAO	SDAOP	SCLO	-	IICRST	-

IIC2: I²C バスモードレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IIC2	ICMR	MLS	WAIT	-	-	BCWP	BC		

IIC2: I²C バスインタラプトイネーブルレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IIC2	ICIER	TIE	TEIE	RIE	NAKIE	STIE	ACKE	ACKBR	ACKBT

IIC2: I²C バスステータスレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IIC2	ICSR	TDRE	TEND	RDRF	NACKF	STOP	ALOVE	AAS	ADZ

IIC2: スレーブアドレスレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IIC2	SAR	SVA							FS

IIC2: I²C バス送信データレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IIC2	ICDRT	(ビットアクセスは未定義)							

IIC2: I²C バス受信データレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IIC2	ICDRR	(ビットアクセスは未定義)							

A/D 変換器

A/D 変換器: A/D データレジスタ A

モジュール名称	レジスタ名称	ビット名称															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	ADDRA	(ビットアクセスは未定義)															

A/D 変換器: A/D データレジスタ B

モジュール名称	レジスタ名称	ビット名称															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	ADDRB	(ビットアクセスは未定義)															

A/D 変換器: A/D データレジスタ C

モジュール名称	レジスタ名称	ビット名称															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	ADDRC	(ビットアクセスは未定義)															

A/D 変換器: A/D データレジスタ D

モジュール名称	レジスタ名称	ビット名称															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	ADDRD	(ビットアクセスは未定義)															

A/D 変換器: A/D コントロール/ステータスレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
AD	ADCSR	ADF	ADIE	ADST	SCAN	CKS	CH		

A/D 変換器: A/D コントロールレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
AD	ADCR	TRGE	-	-	-	-	-	-	-

割り込み

割り込み： 割り込みエッジセレクト入力レジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	IEGR1	NMIEG	-	-	-	IEG3	IEG2	IEG1	IEG0

割り込み： 割り込みエッジセレクト入力レジスタ 2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	IEGR2	-	-	WPEG5	WPEG4	WPEG3	WPEG2	WPEG1	WPEG0

割り込み： 割り込みイネーブルレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	IENR1	IENDT	IENTA	IENWP	-	IEN3	IEN2	IEN1	IEN0

割り込み： 割り込みイネーブルレジスタ 2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	IENR2	-	-	IENTB1	-	-	-	-	-

割り込み： 割り込みフラグレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	IRR1	IRRDT	IRRTA	-	-	IRRI3	IRRI2	IRRI1	IRRI0

割り込み： 割り込みフラグレジスタ 2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	IRR2	-	-	IRRTB1	-	-	-	-	-

割り込み： ウェイクアップ割り込みフラグレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	IWPR	-	-	IWPF5	IWPF4	IWPF3	IWPF2	IWPF1	IWPF0

低消費電力

低消費電力： システムコントロールレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	SYSCR1	SSBY	STS			NESEL	-	-	-

低消費電力： システムコントロールレジスタ 2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	SYSCR2	SMSEL	LSON	DTON	MA			SA	

低消費電力： モジュールスタンバイコントロールレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	MSTCR1	-	MSTIIC	MSTS3	MSTAD	MSTWD	-	MSTTV	MSTTA

低消費電力： モジュールスタンバイコントロールレジスタ 2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	MSTCR2	MSTS3_2	-	-	MSTTB1	-	-	MSTTZ	MSTPWM

フラッシュメモリ

フラッシュメモリ: フラッシュメモリコントロールレジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
FLASH	FLMCR1	-	SWE	ESU	PSU	EV	PV	E	P

フラッシュメモリ: フラッシュメモリコントロールレジスタ 2

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
FLASH	FLMCR2	FLER	-	-	-	-	-	-	-

フラッシュメモリ: ブロック指定レジスタ 1

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
FLASH	EBR1	-	EB6	EB5	EB4	EB3	EB2	EB1	EB0

フラッシュメモリ: フラッシュメモリパワーコントロールレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
FLASH	FLPWCR	PDWND	-	-	-	-	-	-	-

フラッシュメモリ: フラッシュメモリエnableレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
FLASH	FENR	FLSHE	-	-	-	-	-	-	-

アドレスブレーク

アドレスブレーク: アドレスブレークコントロールレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ABRK	CR	RTINTE	CSEL		ACMP			DCMP	

アドレスブレーク: アドレスブレークステータスレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ABRK	SR	ABIF	ABIE	-	-	-	-	-	-

アドレスブレーク: ブレークアドレスレジスタ

モジュール名称	レジスタ名称	ビット名称															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRK	*BAR	(ビットアクセスは未定義)															

!!!このレジスタはポインタで定義されています!!!

アドレスブレーク: ブレークデータレジスタ

モジュール名称	レジスタ名称	ビット名称															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRK	BDR	(ビットアクセスは未定義)															

低電圧検出回路(オプション)

低電圧検出回路: 低電圧検出コントロールレジスタ

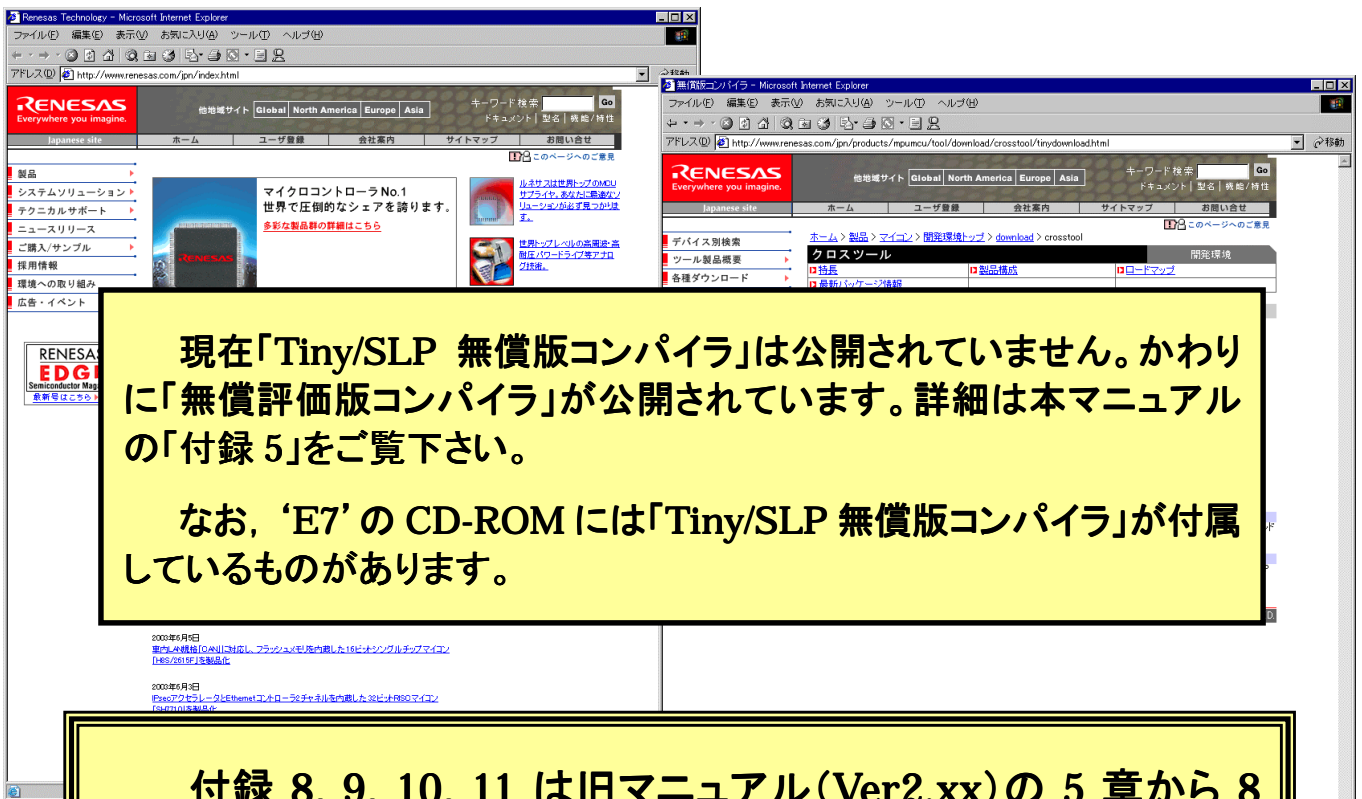
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LVD	CR	LVDE	-	-	-	LVDSEL	LCDRE	LVDDE	LVDUE

低電圧検出回路: 低電圧検出ステータスレジスタ

モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LVD	SR	-	-	-	-	-	-	LVDDF	LVDUF

付録-8 無償版コンパイラ“HEW”のインストール(旧バージョン)

TK-3687 のプログラミングで使用する無償版コンパイラ, HEW(High-performance Embedded Workshop) は株式会社ルネサステクノロジのホームページよりダウンロードします。ダウンロードサイトの URL は以下の通りです。(下記の“Attention”をご覧ください)



現在「Tiny/SLP 無償版コンパイラ」は公開されていません。かわりに「無償評価版コンパイラ」が公開されています。詳細は本マニュアルの「付録 5」をご覧ください。

なお、‘E7’の CD-ROM には「Tiny/SLP 無償版コンパイラ」が付属しているものがあります。

付録 8, 9, 10, 11 は旧マニュアル(Ver2.xx)の 5 章から 8 章までをそのままコピーしたものです。「無償版コンパイラ」や「E7」をご利用のお客様にとっては必要な資料と考え、付録ではありますがマニュアルに残すことにしました。現在ご利用の開発環境に応じてご利用いただければ幸いです。

ダウンロードサイトで“Download”をクリックし必須事項を入力してダウンロードを開始します。入力したメールアドレス宛にファイルを解凍する為のパスワードが送られてきますので、送られてきたパスワードでダウンロードしたファイルを解凍しインストールして下さい。

!!!Attention!!!

“E7”を購入された方は CD-ROM に印刷されているソフトウェアのバージョンをご確認ください。Version2.0.00 以降の場合、無償版コンパイラも“E7”に添付されています。“E7”のエミュレータソフトをインストールする際、同時に無償版コンパイラもインストールされます。

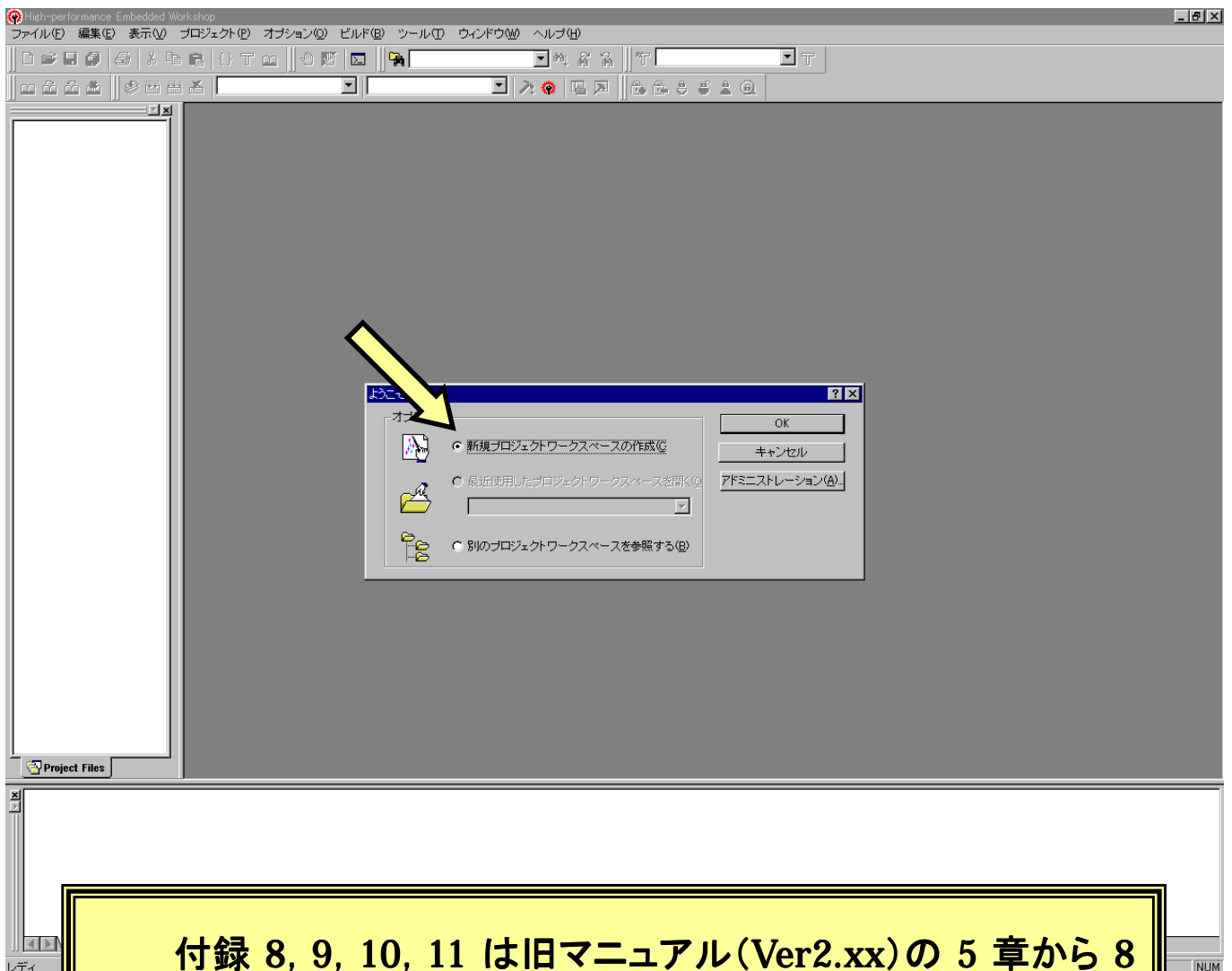
なお、ここで紹介している「Tiny/SLP 無償版コンパイラ」はインターネット上では公開されていません。ルネサステクノロジは現在、High-performance Embedded Workshop V. 4(HEW4)に対応した「無償評価版コンパイラ」を公開しています。「無償評価版コンパイラ」については本マニュアルの「付録 5」をご覧ください。“E7”付属のコンパイラもいずれ「無償評価版コンパイラ」に移行する予定だそうです。

付録-9 プログラムの作成(旧バージョン)

統合化環境; HEW ではプログラム作成作業をプロジェクトと呼び、そのプロジェクトに関連するファイルは 1 つのワークスペース内にまとめられ管理されます。通常はワークスペース、プロジェクト、メインプログラムに共通の名前が付けられます。以下に、新規プロジェクトとして 'samp_01' を作成する手順と動作確認の手順を示しますが、作業に先立ち、HEW 専用作業フォルダとして、HEW2_xxx(xxxには個人名を入れます)を作っておきます。ここでは、C:\¥Hew2_yKikuchi としていますので、個人名の部分を置き換えてお読み下さい。

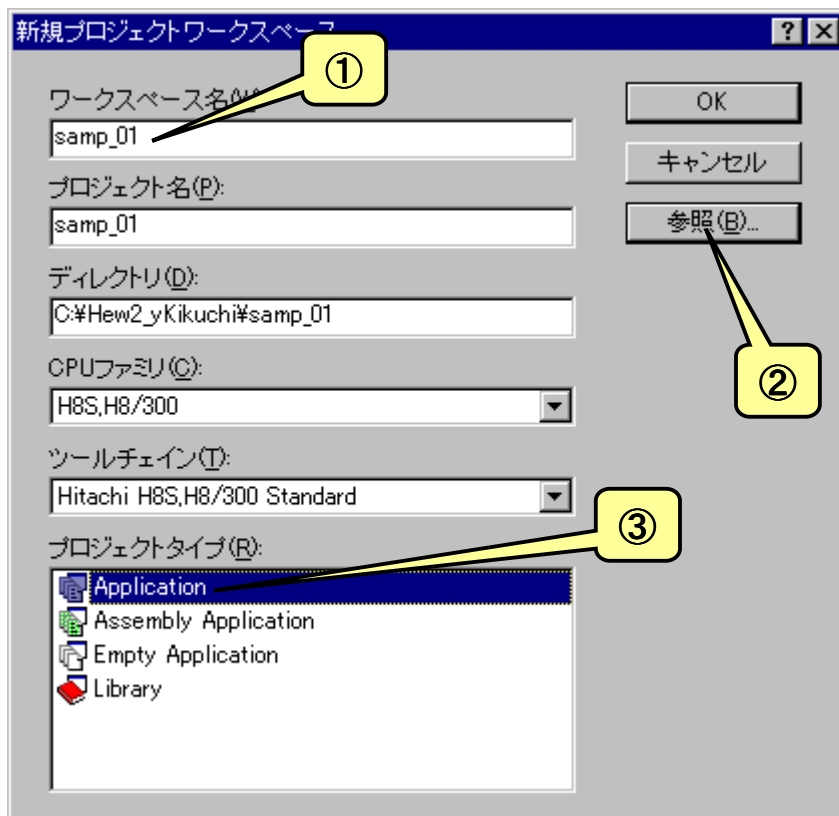


HEW を開くと、下記の画面が現れるので、新規作成の場合、“新規プロジェクトワークスペースの作成”を選択します。尚、過去に作成したワークスペースであれば、“最近使用したプロジェクトワークスペースを開く”をチェックします。ここでは、新規作成にチェックを入れて OK を押します。するとワークスペースの設定画面が開きます。

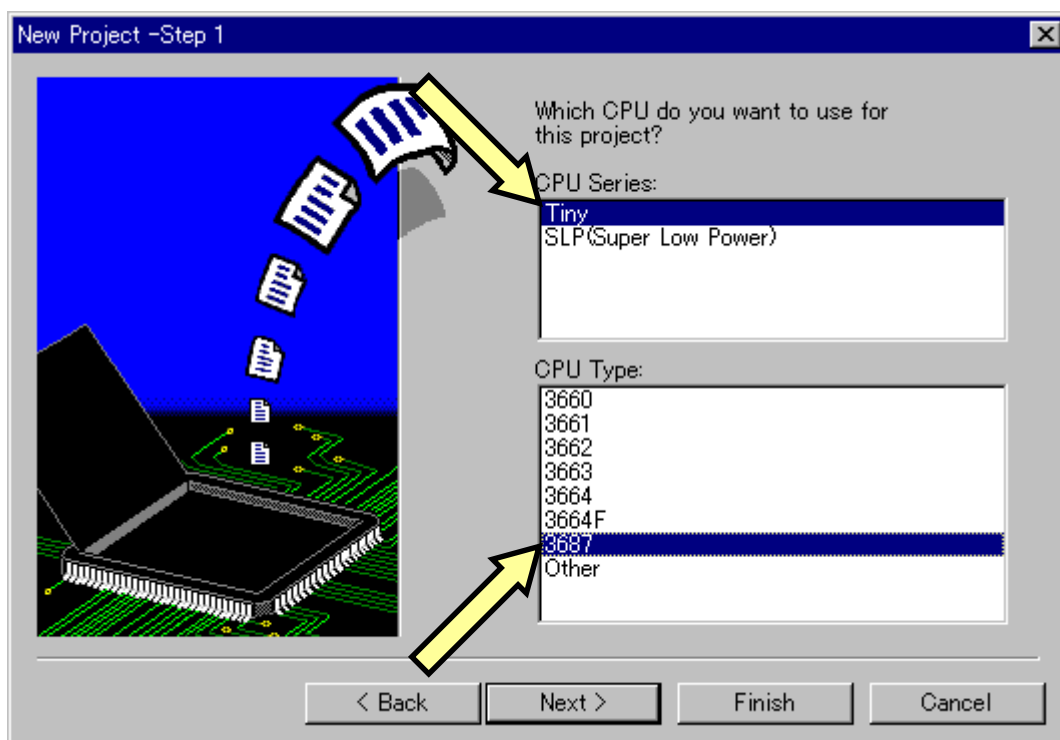


付録 8, 9, 10, 11 は旧マニュアル(Ver2.xx)の 5 章から 8 章までをそのままコピーしたものです。「無償版コンパイラ」や「E7」をご利用のお客様にとっては必要な資料と考え、付録ではありますがマニュアルに残すことにしました。現在ご利用の開発環境に応じてご利用いただければ幸いです。

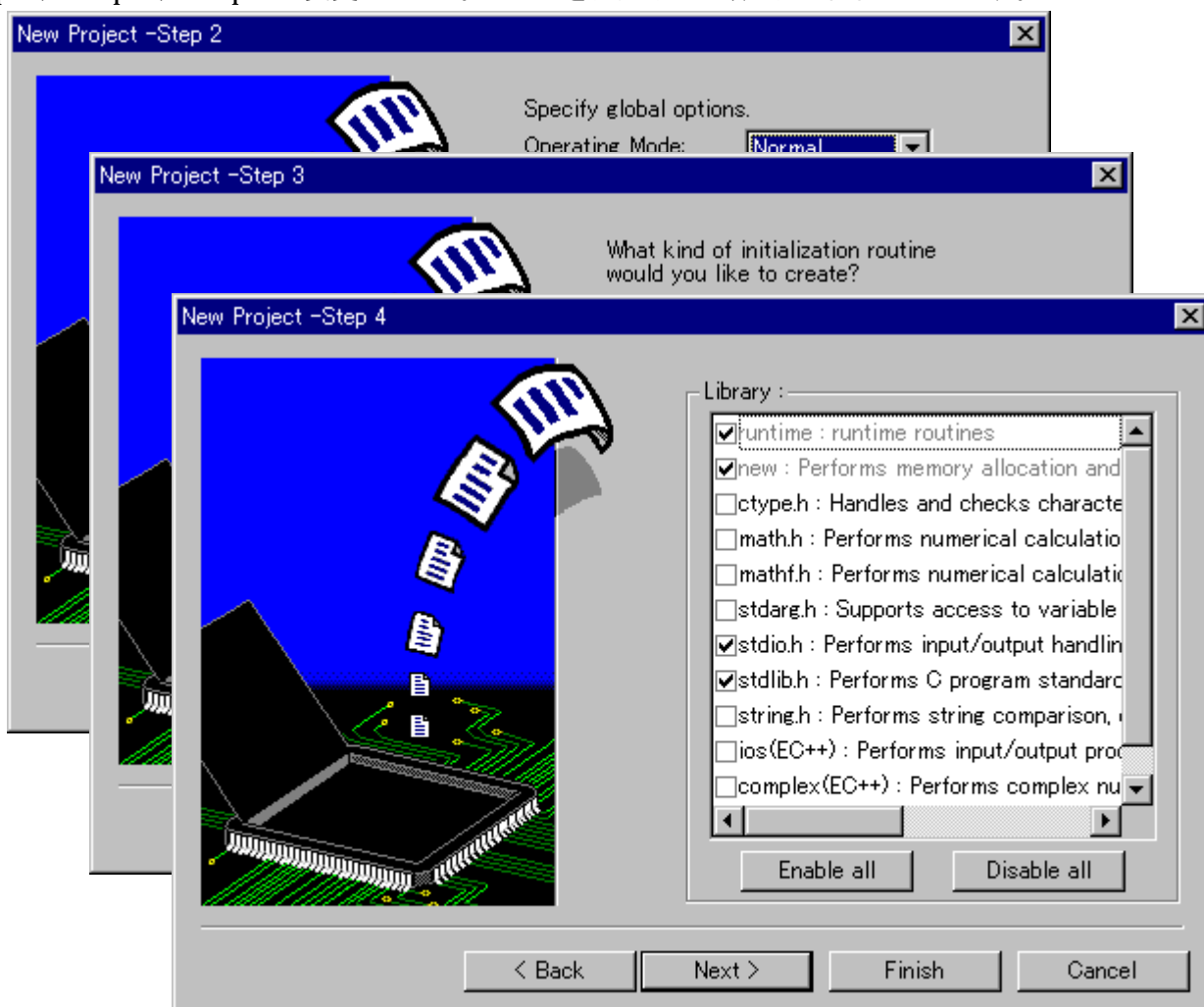
先ずワークスペース名を入力します。プロジェクト名と同じとしますから、①最初の項目に samp_01 と入力して下さい。プロジェクト名は自動的に同名で入力されます。②ワークスペースの場所は右 3 段目の '参照' ボタンをクリックし、予め用意した HEW 専用作業フォルダ(ここでは、Hew2_yKikuchi)を指定します。設定後、3 項目の 'ディレクトリ' 欄が正しい事を確認します。③次に、作成するプログラム言語を選択します。ここではアセンブラではなく C ですから、**Application** を選択し、OK をクリックして下さい。



'Step1' で使用する CPU のシリーズ (Tiny) と CPU タイプ (3687) を設定し、'Next' をクリックします。



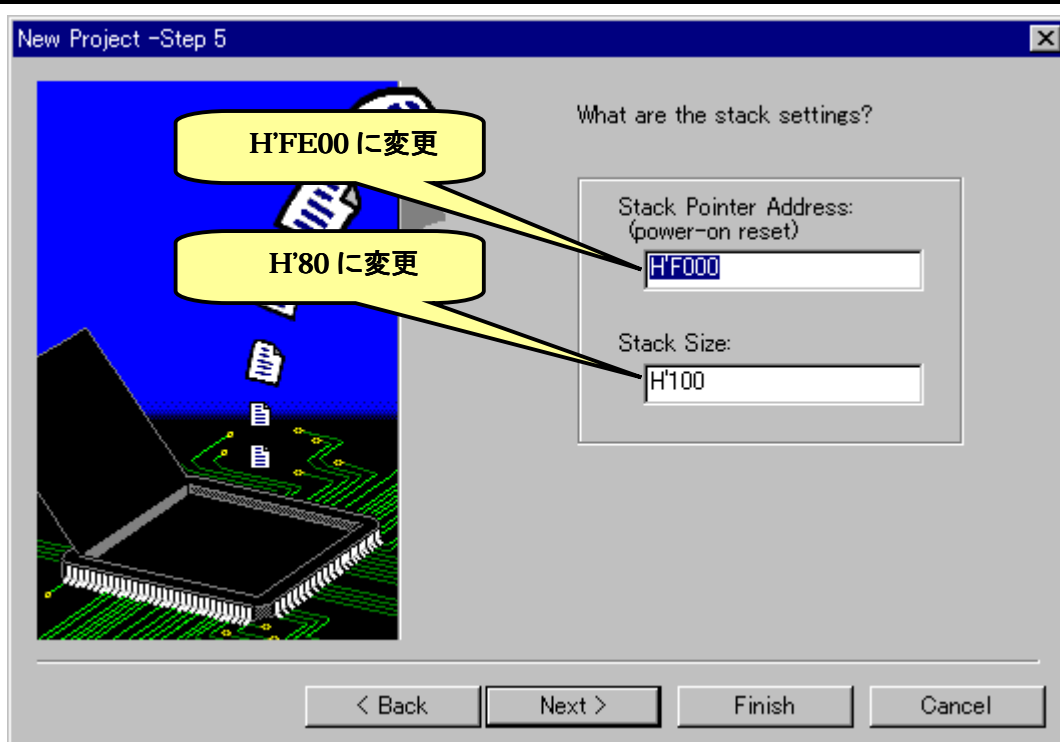
‘Step2’, ‘Step3’, ‘Step4’ は変更しません。‘Next’ をクリックして順に次の画面に進みます。



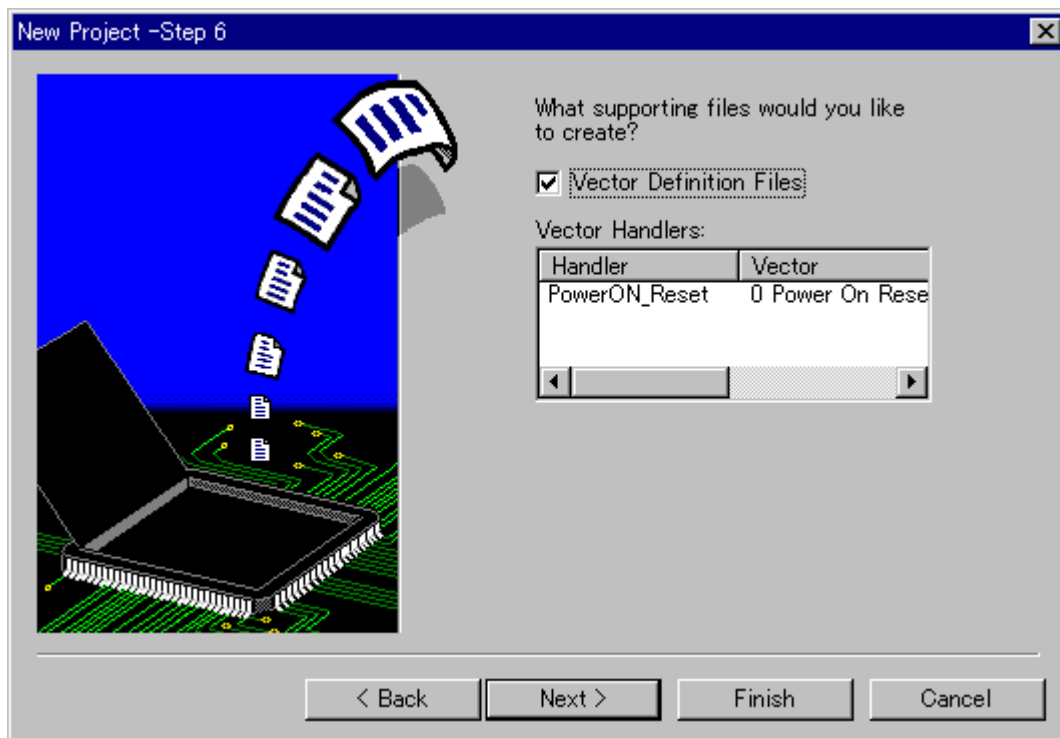
‘ハイパーH8’を使用するときは‘Step5’でスタックのアドレスとサイズを変更します。‘E7’を使用するときなどその他の場合は変更する必要はありません。デフォルトのままでお使いください。

→ ‘Next’ をクリックして ‘Step6’ へ進みます。

‘ハイパーH8’を使用するときは‘Step5’でスタックのアドレスとサイズを変更します。メモリマップにあわせて、①スタックポインタをH'FE00に、②スタックサイズをH'80にします。‘Next’をクリックして次の画面に進みます。

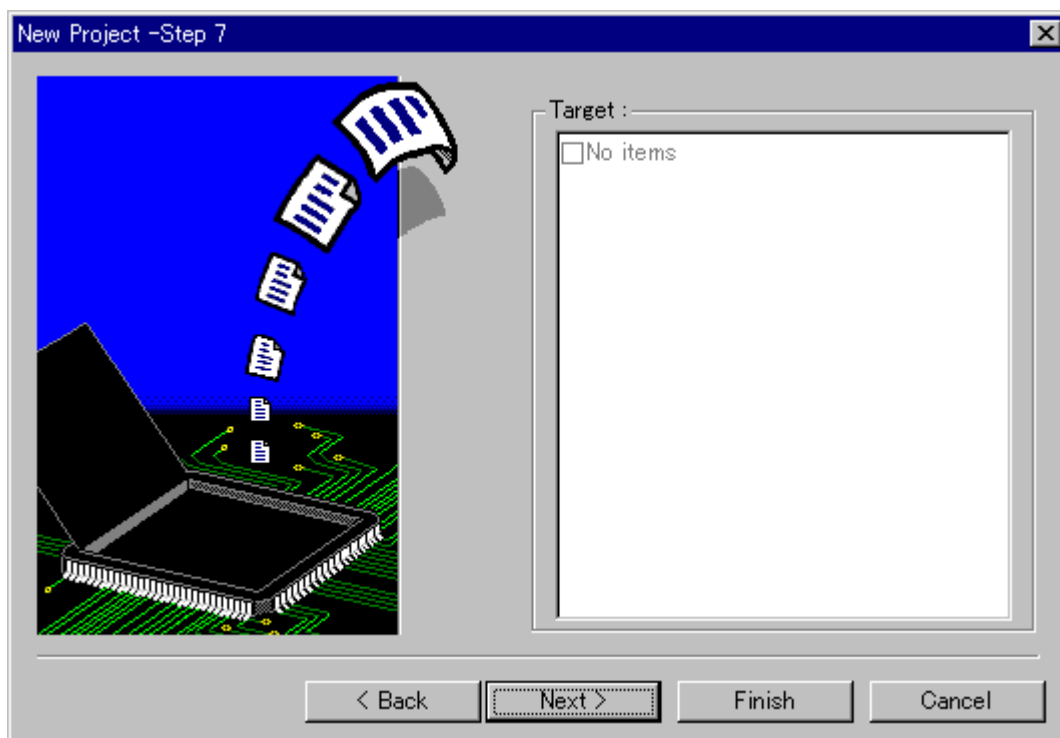


‘Step6’は変更しません。‘Next’をクリックして次の画面に進んで下さい。

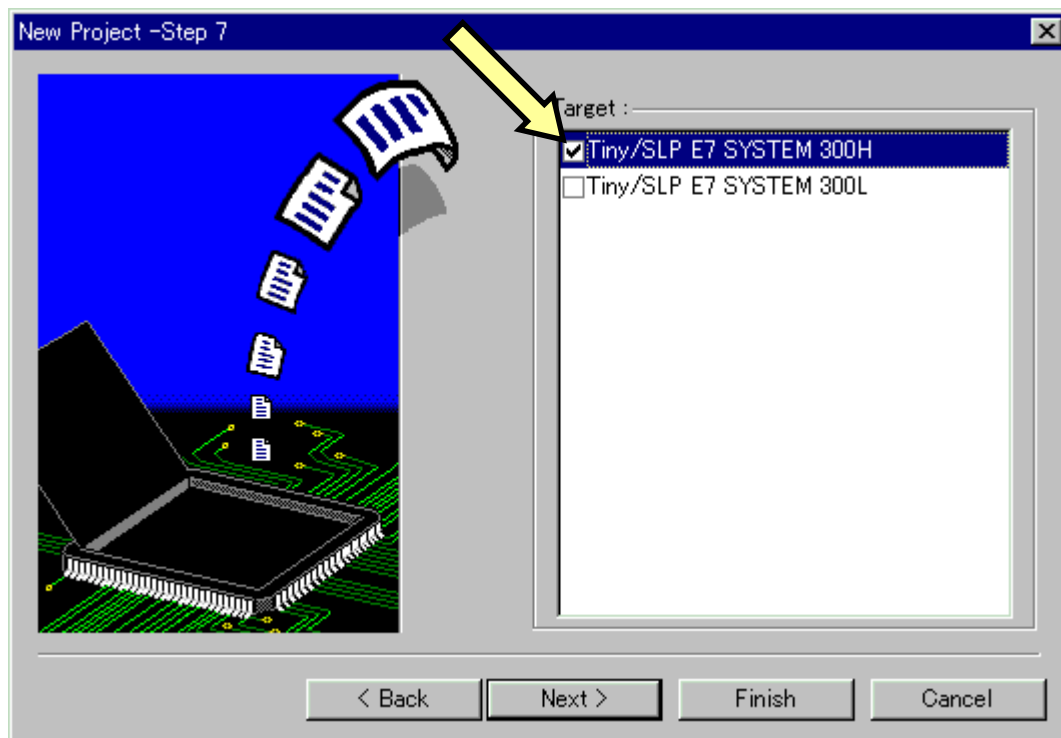


‘Step7’は‘E7’をインストールしていないときと、インストールしているときで、設定する項目が異なります。ご注意ください。

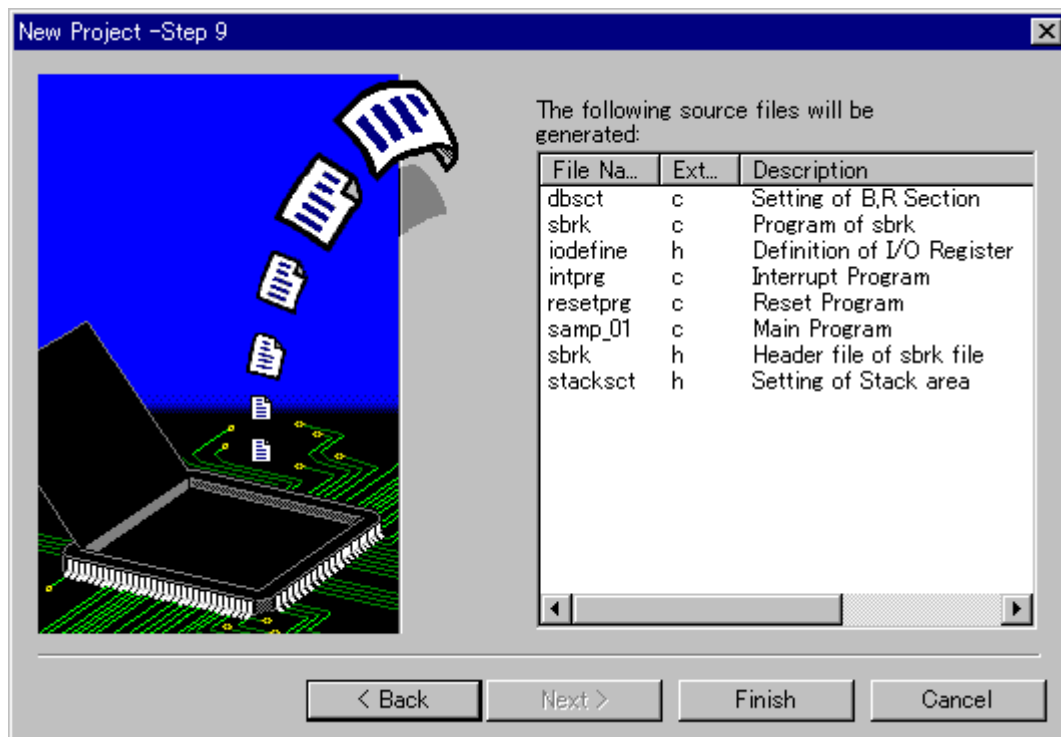
‘E7’をインストールしていないとき
‘Step7’は右のような画面になります。
変更しませんので、
‘Next’をクリックして
次の画面に進んで下さい。



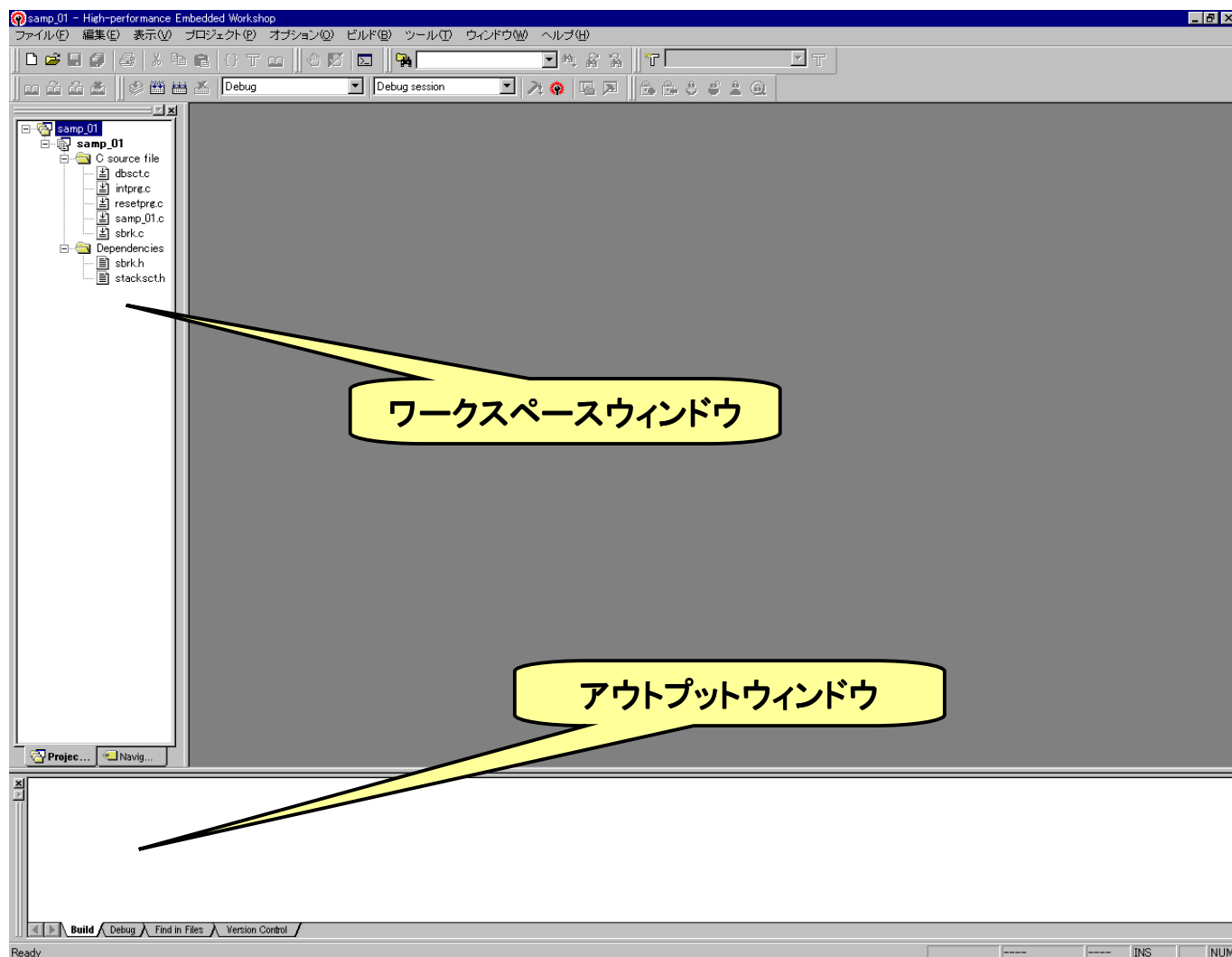
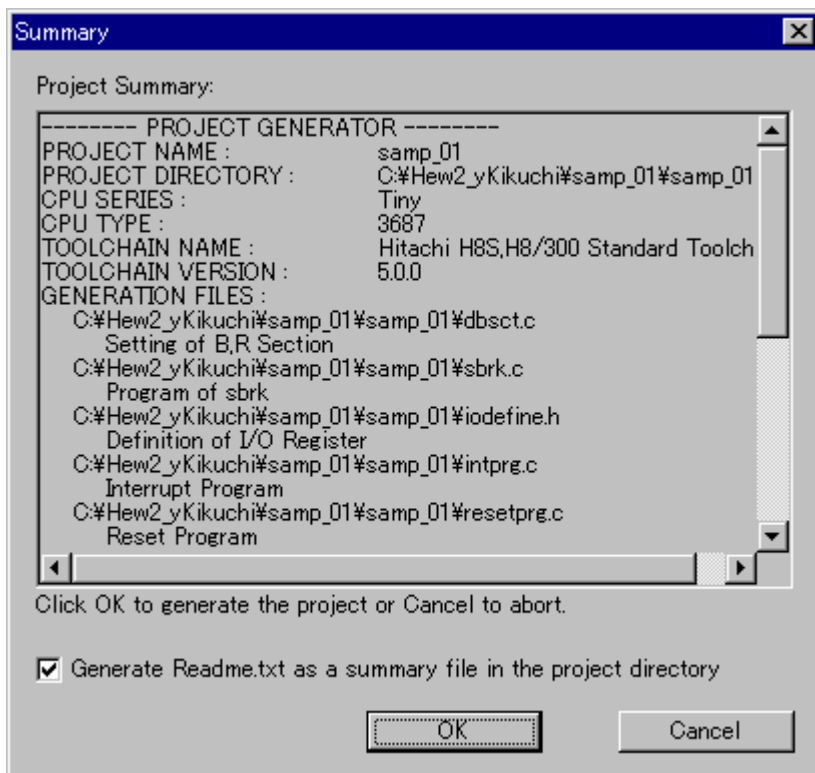
‘E7’をインストールしているとき‘Step7’は右のような画面になります。‘Tiny/SLP E7 SYSTEM 300H’にチェックをつけてください。そして、‘Next’をクリックして次の画面に進みます。



次は‘Step9’です(‘Step8’はない)。「Step9」は変更しません。「Finish」をクリックします。

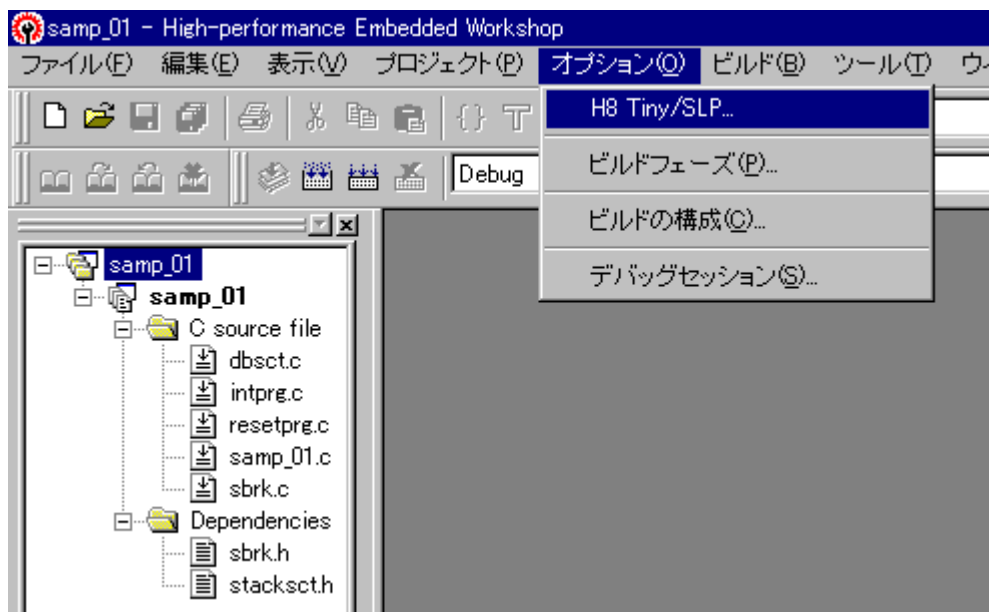


すると、'Project Summary'が表示されるので、'OK'をクリックします。これで、ワークスペースが完成します。統合化環境;HEW はプロジェクトに必要なファイルを自動生成し、それらのファイルは左端のワークスペースウィンドウに一覧表示されます。

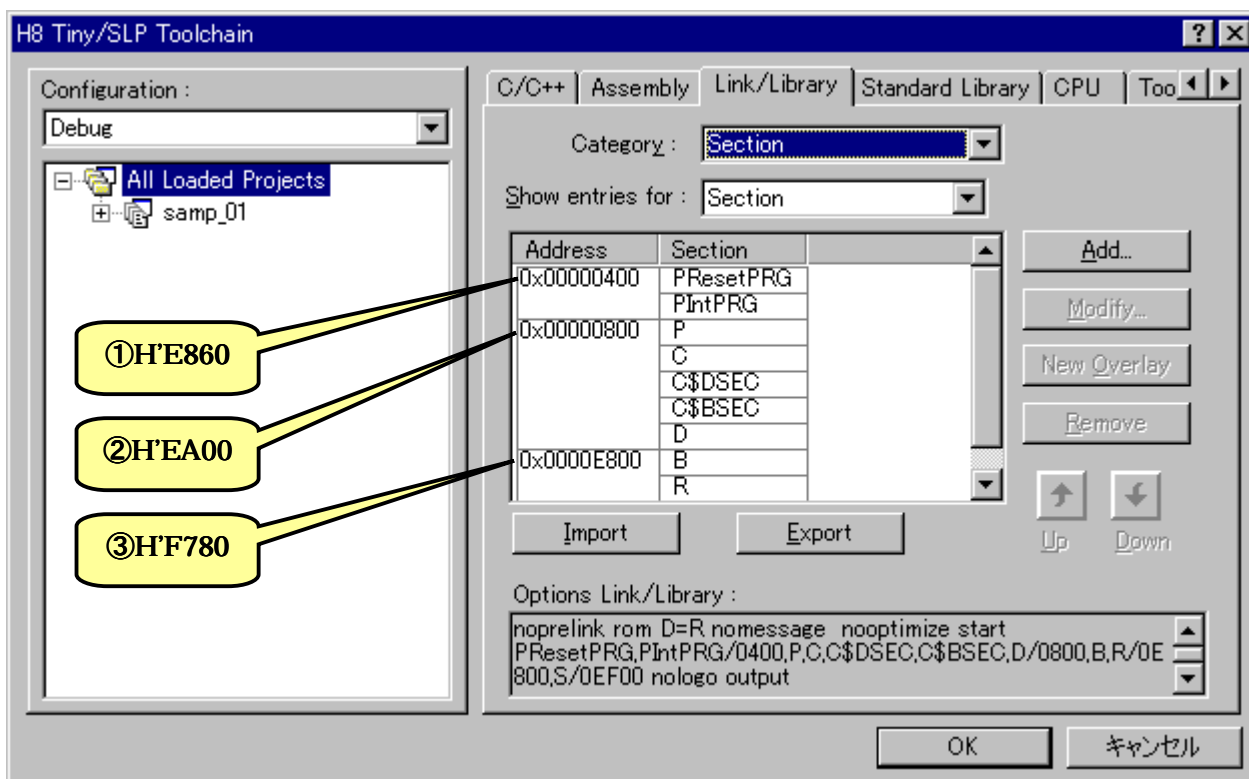


‘ハイパーH8’を使用するときは Section の変更を行いません。‘E7’を使用するときなどその他の場合は変更する必要はありません。デフォルトのままでお使いください。→次のページへお進みください。

‘ハイパーH8’を使用する時は、リンカに渡す各セクションのアドレス設定を行います。右図のようにメニューバーから‘H8Tiny/SLP...’を選びます。



すると、‘H8 Tiny/SLP Toolchain’ ウィンドウが開くので、‘Link/Library’のタブを選び、‘Category’のドロップダウンメニューを開いて‘Section’をクリックすると、下図のような各 Section の先頭アドレスを設定する画面になります。そこで、‘PResetPRG’、‘P’、‘B’の各アドレス欄をマウスで選択してから‘Modify...’をクリックし、それぞれの値をメモリマップ(4章参照)にあわせて、①H'E860、②H'EA00、③H'F780に変更します。



確認後‘OK’で閉じますが、その前に今後作成するプロジェクトのことを考慮し、このセクションの設定を‘Export’ボタンを使って HEW2 専用作業フォルダに保存しておきます。ファイル名は‘section.his’とします。次の新規プロジェクトではこのファイルを‘Import’すれば簡単に済ませることができます。

HEW が作成したり
 セットプログラム,
 'resetprg.c' を修正しま
 す。HEW はリセットプロ
 グラム終了後すぐに割込
 みを受け付けるように自
 動作成しますが、ここ
 では、メインプログラム
 内で割り込み受け付けを
 制御するように変更し
 ます。変更を行なうに
 は、ワークスペースウ
 インドウ内の 'resetpr
 g.c' をダブルクリック
 すれば自動的に HEW
 のエディタが起動し、
 'resetprg.c' の編集
 モードに入ります。図
 のように割り込みを許
 可している行をコメント
 に変更します。

```

//extern void _CALL_INIT(void);
//extern void _CALL_END(void);
//#ifdef __cplusplus
//}
//endif

#pragma section ResetPRG

__entry(vect=0) void PowerON_Reset(void)
{
    set_imask_ccr(1);
    _INIT_SCT();

    // _CALL_INIT();           // Remove the comment when you use global class object
    // _INIT_IOLIB();         // Remove the comment when you use SIM I/O
    // ermo=0;                 // Remove the comment when you use ermo
    // srand(1);               // Remove the comment when you use srand(1)
    // _slptr=NULL;           // Remove
    // hardwareReset();       // Remove
    // set_imask_ccr(0);      // set_imask_ccr(0);
    main();

    // _CLOSEALL();          // Remove the comment when you use SIM I/O
    // _CALL_END();          // Remove the comment when you use global class object

    sleep();
}

__interrupt(vect=1) void Manual_Reset(void) // Remove the comment when you use Manual Reset
{
}

```

次に、メインプログラムを修正します。ワークスペースウィンドウ内の 'samp_01.c' をダブルクリックして編集モードにします。すると、図のように自動作成された事がわかります。これに追加・修正していきます。次の 6 箇所です。

- ① このプログラムを作った人の所属・名前を記載します。
- ② 必要ないところを削除します(2 箇所)。
- ③ メイン関数にプログラムを追加します。
- ④ インクルードするファイルの定義を追加します。
- ⑤ 変数の定義を追加します。
- ⑥ プログラムを判りやすくするためにサブタイトルやコメントを追加します。

```

/*****
/*
/* FILE      : samp_01.c
/* DATE      : Mon, Oct 06, 2003
/* DESCRIPTION : Main Program
/* CPU TYPE   : H8/3687
/*
/* This file is generated by Hitachi Project Generator (Ver.2.1).
/*
*****/

#ifdef __cplusplus
extern "C" {
#endif
void abort(void);
#ifdef __cplusplus
}
#endif

void main(void)
{
}

void abort(void)
{
}

```

以上の追加・修正を加えた後のリストは次のとおりです。


```

/*****
/*
/* FILE      :samp_01.c
/* DATE      :Mon, Oct 06, 2003
/* DESCRIPTION:Main Program
/* CPU TYPE  :H8/3687
/*
/* This file is programmed by TOYO-LINX Co.,Ltd / yKikuchi
/*
*****/

*****
      インクルードファイル
*****
#include <machine.h>
#include "iodefine.h"

*****
      スタティック変数の定義
*****
int a = 32;
int b = 53;
int c;

*****
      メインプログラム
*****
void main(void)
{
    P_PORT.PCR5.BYTE = 0xff;
    while(1){
        c = a + b;
        P_PORT.PDR5.BYTE = c;
    }
}

```

メインプログラムの中で、ポートの構造体名、'P_PORT' が使われていますが、HEW のバージョンによっては 'IO' で宣言されている場合があります。未定義エラーが生じる場合は 'P_PORT' を 'IO' に置き換えてみてください。

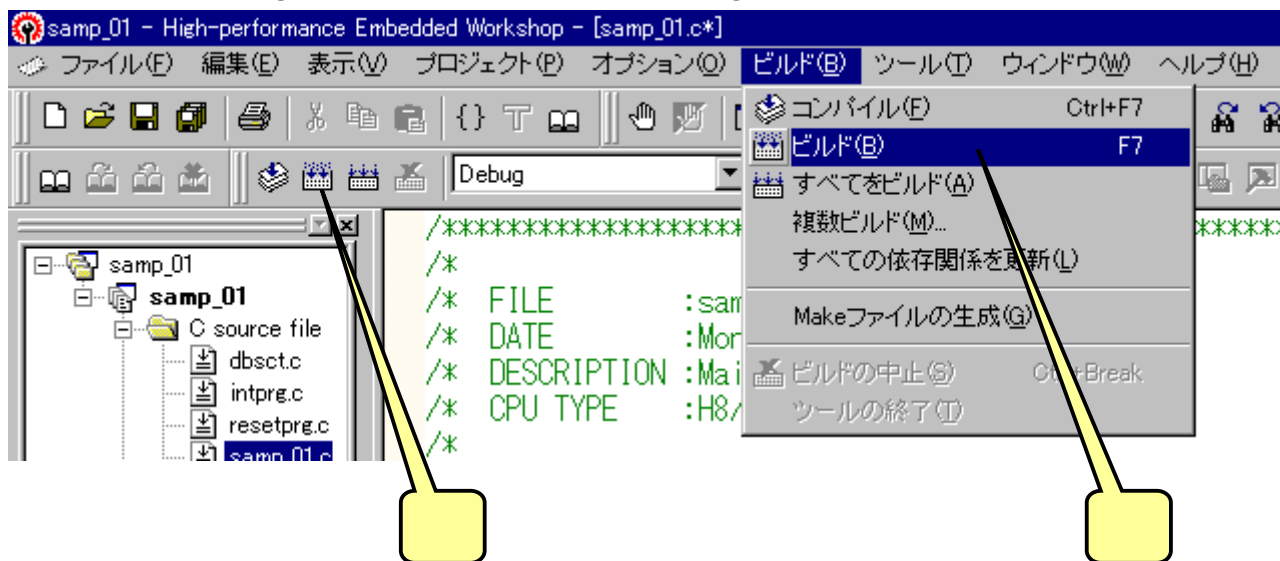
```

IO.PCR5 = 0xff;

while(1){
    c = a + b;
    IO.PDR5.BYTE = c;
}

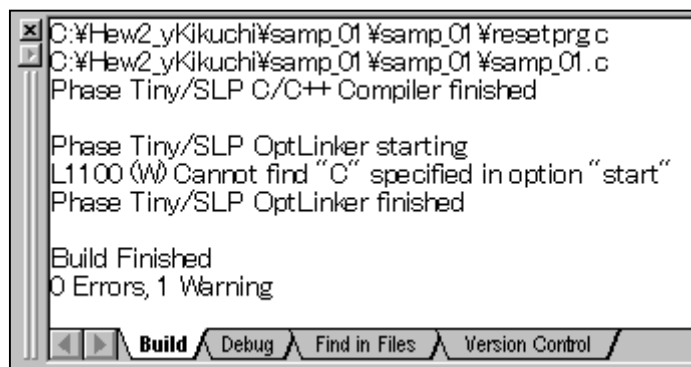
```

いよいよコンパイルします(ここが第一の関門でコンパイルが通るかどうかが問題です)。ファンクションキーの[F7]を押すか、図のように①メニューバーから‘ビルド’を選ぶか、②ツールバーのアイコンをクリックして下さい。



コンパイルが終了するとアウトプットウィンドウに結果が表示されます。表記上の間違いの有無がチェックされ、なければ‘0 Errors’と表示されます。

エラーがある場合はソースファイルを修正しなければなりません。その際、エラー項目にマウスカーソルを当ててクリックするとエラー行に飛んでいきます。(このあたりの機能が統合化環境のありがたいところです)



図では‘1 Warning’と表示されていますが、内容によっては問題ありません。例えば、この例の「L1100 (W) Cannot find “C” specified in option “start”」は‘const’で修飾された変数がプログラム上にないときに表示されます。しかし、‘Warnings’のなかには動作に影響を与えるものもありますのでコンパイラの説明書と比較して問題ないか必ず確認して下さい。

どうしても‘Warnig’が気になる人へ…

‘Warnig’を出さない方法も幾つか考えられます。

- ① ‘const’で修飾された変数をダミーで定義する。
- ② P.18 下の図にて、‘H8 Tiny/SLP Toolchain’ダイアログでCセクションを削除する。

いずれかをお選びください。

エラーがなければ、‘ハイパーH8’、または、エミュレータ‘E7’を使いTK-3687にファイルをダウンロード、トレース実行し、実際の動作をマイコン上で確認する作業に入ります。ここではHEWは最小化しておきましょう。動作確認して、考えたとおり動かない時は再度HEWに戻り、プログラム修正する必要があるからです。

もし、ここで作業を中断する場合はファイル保存し、終了ボタンをクリックします。次回の作業再開は作業フォルダ‘samp_01’のワークスペースファイル‘samp_01.hws’をダブルクリックすれば、HEWが立ち上がり終了時の画面が復帰します。



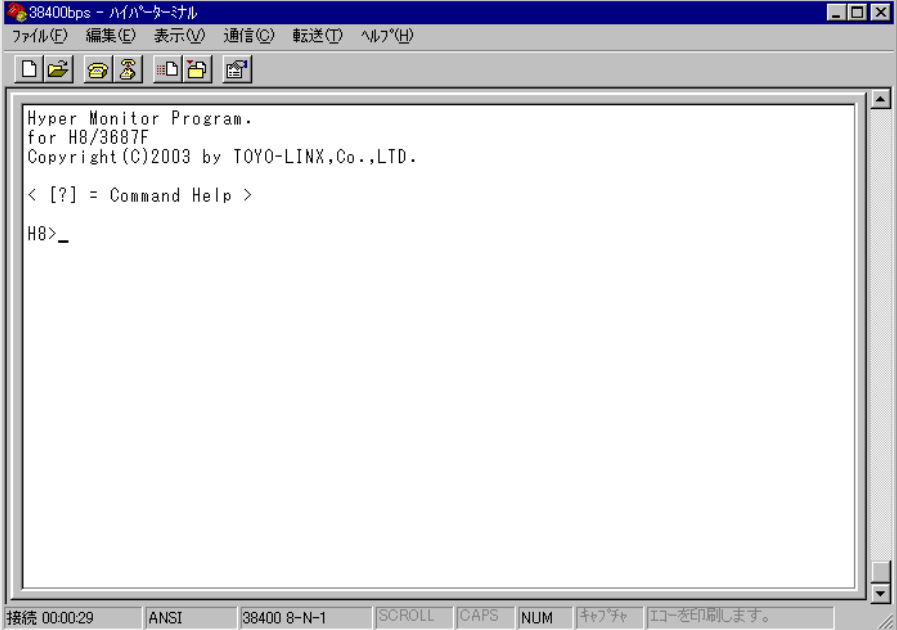
次にプログラムの確認作業に入ります。

- ‘ハイパーH8’をお使いの方 → 7章へお進みください。
- ‘E7’をお使いの方 → 8章へお進みください。

付録-10 “ハイパーH8”を使ったダウンロードと実行(旧バージョン)

それでは、6章で作成したプログラム‘samp_01’を実行しましょう。3章でその手順を説明しましたが、復習もかねてもう一度説明します。

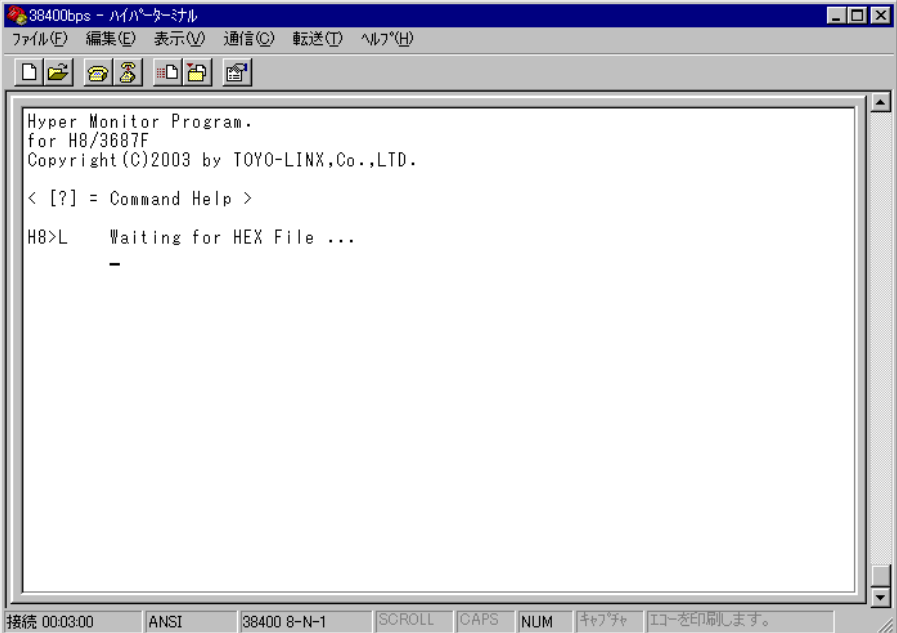
パソコンと TK-3687 を RS-232C ケーブルで接続したあと、3章で作成したハイパーターミナルのショートカット、‘38400bps’をダブルクリックします。次に、TK-3687の電源を入れます。すると、右のような画面が表示されます。



```
Hyper Monitor Program.
for H8/3887F
Copyright (C)2003 by TOYO-LINX,Co.,LTD.
< [?] = Command Help >
H8>_
```

‘L’コマンドで‘samp_01.mot’をマイコンにダウンロードします。まず、‘L’を入力し Enter キーを押します。

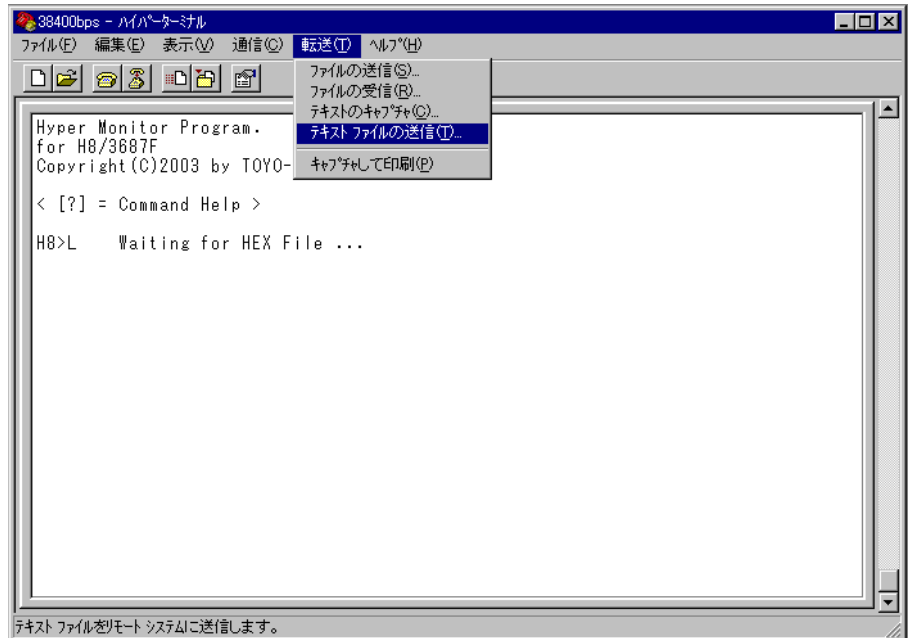
H8>L



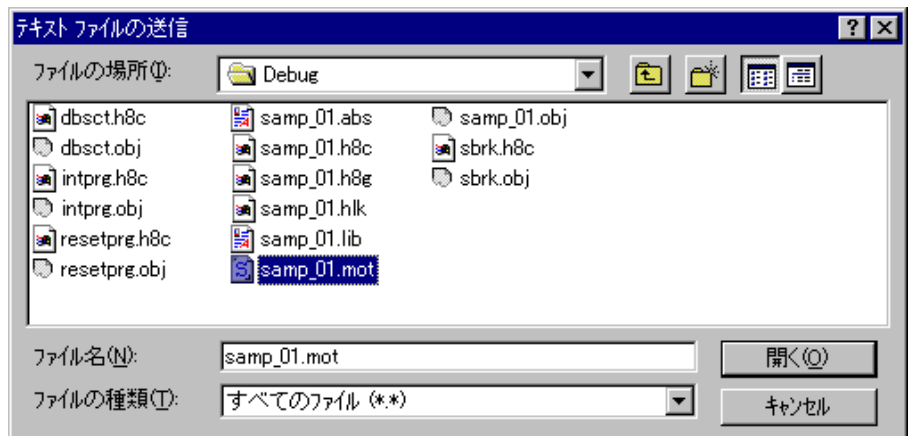
```
Hyper Monitor Program.
for H8/3887F
Copyright (C)2003 by TOYO-LINX,Co.,LTD.
< [?] = Command Help >
H8>L  Waiting for HEX File ...
-
```

付録 8, 9, 10, 11 は旧マニュアル(Ver2.xx)の 5 章から 8 章までをそのままコピーしたものです。「無償版コンパイラ」や「E7」をご利用のお客様にとっては必要な資料と考え、付録ではありませんがマニュアルに残すことにしました。現在ご利用の開発環境に応じてご利用いただければ幸いです。

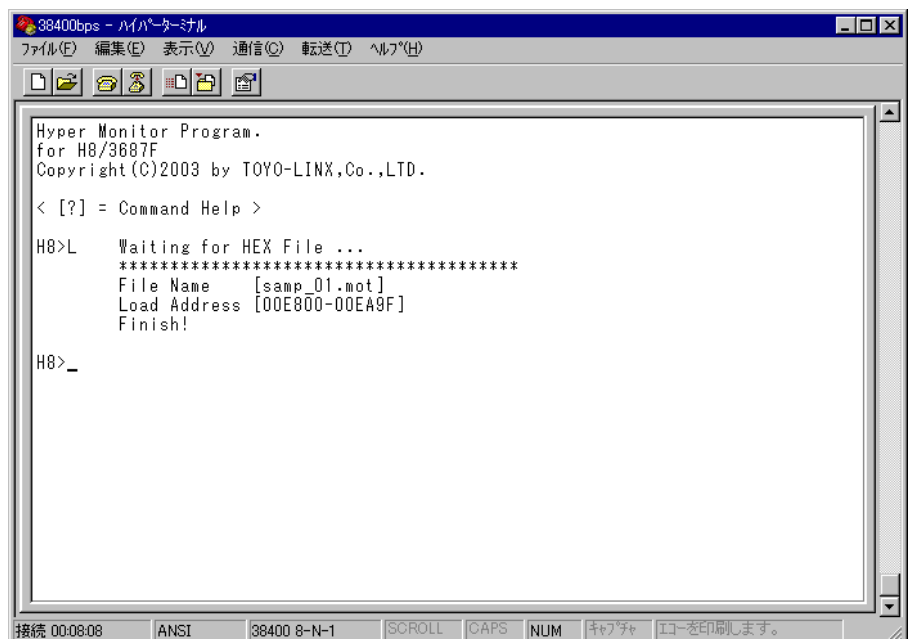
メニューから‘テキストファイルの送信’を選択します。



‘テキストファイルの送信’ウィンドウが開きますので、‘samp_01.mot’を開きます。

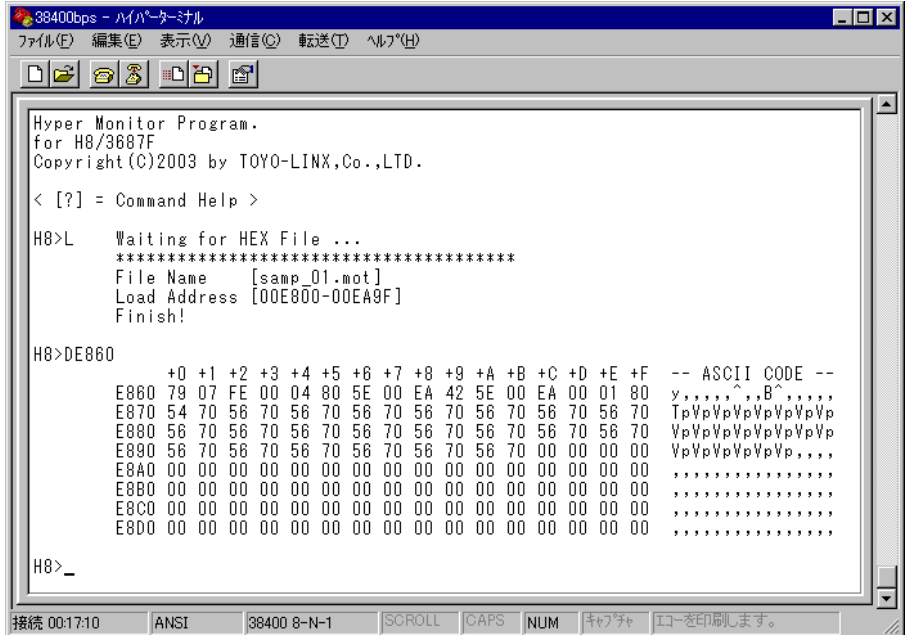


ファイルのダウンロードがスタートします。終了すると、右画面になります。



ちゃんとダウンロードできたか確認しておきましょう。プログラムはH'E860 番地から始まります。'D'コマンドでダンプします。

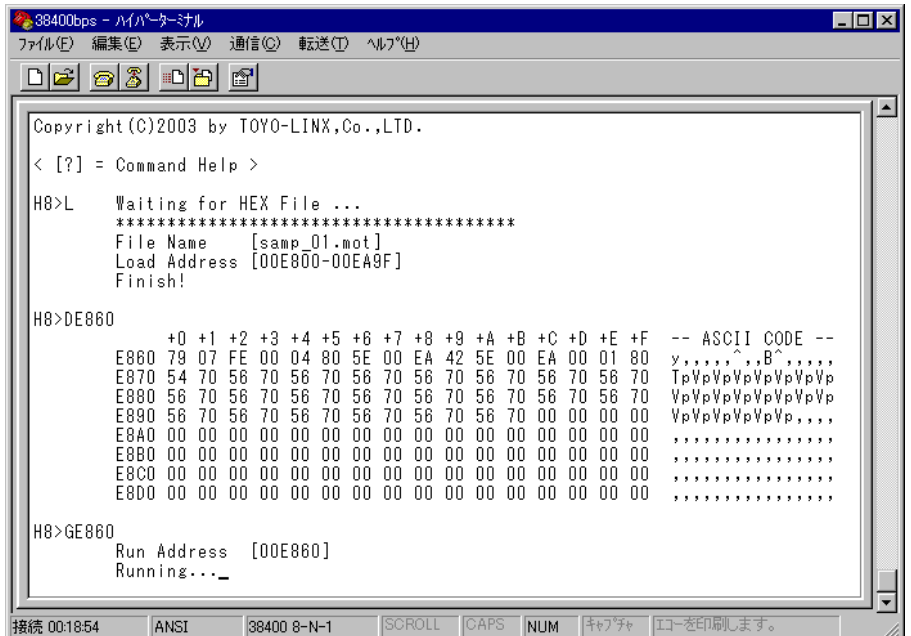
```
H8>DE860
```



いよいよ実行します。プログラムが始まるH'E860 番地から'G'コマンドで実行します。

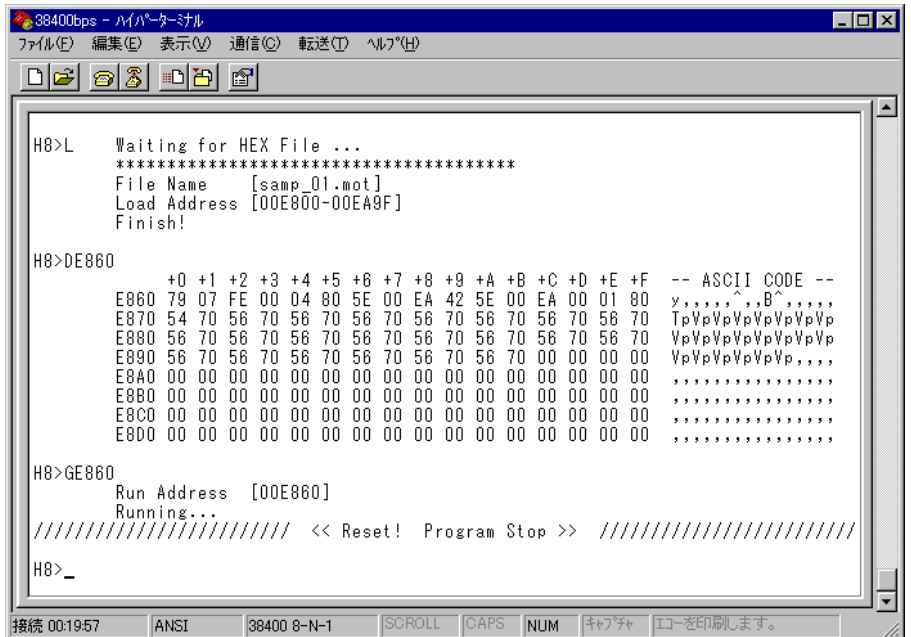
```
H8>GE860
```

計算結果がポート5のLEDに表示されればOKです。



実行を終了する時はボードのリセットスイッチを押して下さい。

ちなみに、ダウンロード後、即、実行させる場合はLとGを組み合わせ'LG'と入力することもできます。ファイルを指定すれば実行までされるので、とても便利な機能です。



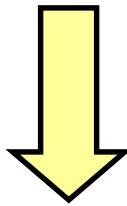


本章の最後に

以上が HEW におけるプログラム作成から動作確認までの手順です。ここでは、'samp_01' を例にし説明をして来ましたが、これで一通りプログラムの作成手順が飲み込めたのではないのでしょうか。この後は、各自で簡単なプログラムにトライして見て下さい。

C の制御文 (if, for, while など) が理解できれば、第 1 ステップ合格です。また、実習・応用プログラムを 9 章以降に載せてありますので、それらを参考にすれば、入力/出力ポートの制御やより高度なプログラミング・テクニックを身に付けることが可能です。

文中では、第 2 の関門は出てきませんでしたが、第 1 の関門をクリアしダウンロード・実行して、旨く動作しないこともあります。それが第 2 の関門です。プログラムの内容にも依りますが、第 2 の関門をクリアすることが極めて大変な辛い仕事になります。そこで机上デバッグやモニタを駆使してバグを探すわけですが、一番重要なことは '何か変だ' とか '何かちょっと違う' といった些細な疑問や変化を決して見逃さない事です。それを心がければ、バグを見つけることはそう大変なことではありません。



9 章にお進みください。

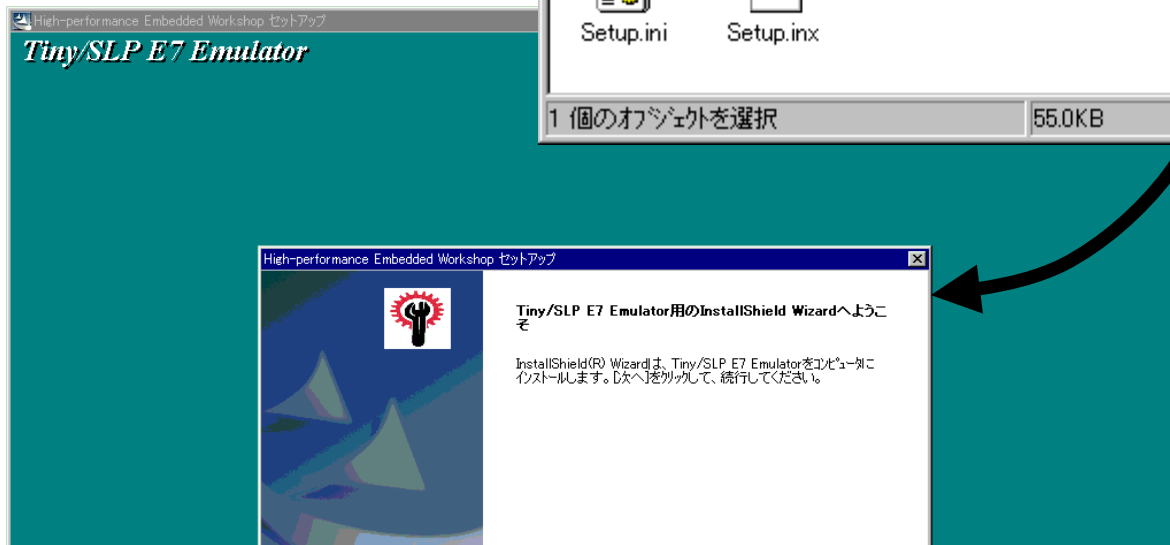
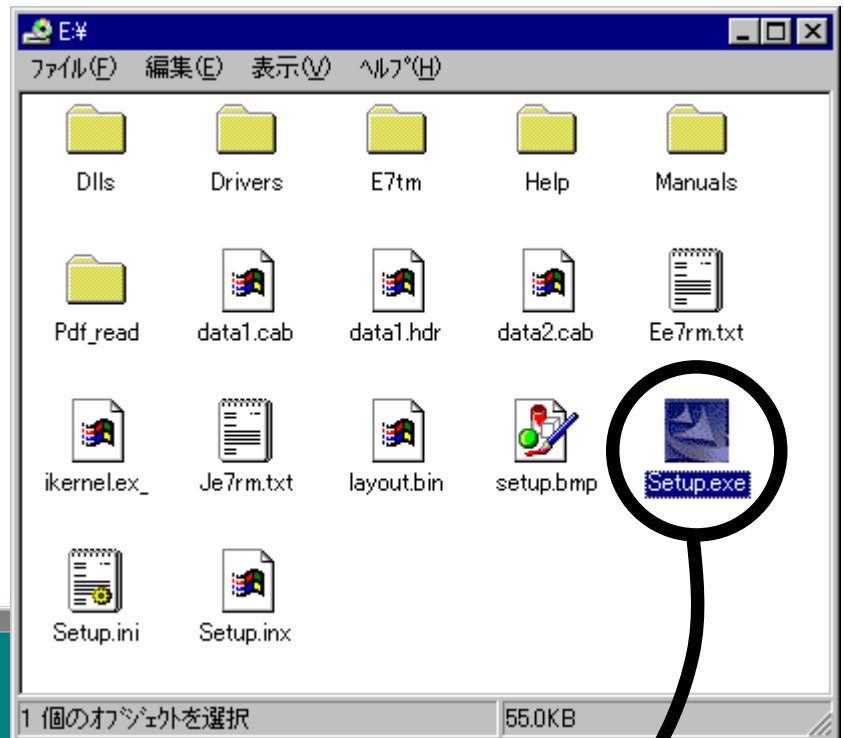
付録-11 エミュレータ“E7”を使ったダウンロードと実行(旧バージョン)

それでは、6章で作成した’samp_01’を“E7”でダウンロード・実行してみましょう。その前に、“E7”のエミュレータソフトのインストールはお済みでしょうか。まだの方は以下の手順でインストールしてください。

まず最初に、インストールされている無償版コンパイラ“HEW”のバージョンが Version2.2(release15)以降かご確認ください。それより前のバージョンで“E7”は動作しません。5章を参照し、最新版の“HEW”をインストールしてください。(“E7”の CD-ROM のバージョンが Version2.0.00 以降の時はエミュレータソフトのインストールと同時に最新の“HEW”もインストールされます。)

次に、“E7”の CD 中の’setup.exe’を実行して下さい。あとは、インストールウィザードに従いインストールします。このとき、無償版コンパイラの“HEW”がインストールされているフォルダと同一フォルダに、“E7”の“HEW”をインストールしてください。ビルドとデバッグを一つの“HEW”から操作する事ができるようになります。

さらにここで“E7”とパソコンを接続する際の各種設定(USBドライバ、E7のファームウェア等)を済ませます。設定方法の詳細については“E7”のマニュアルをご覧ください。



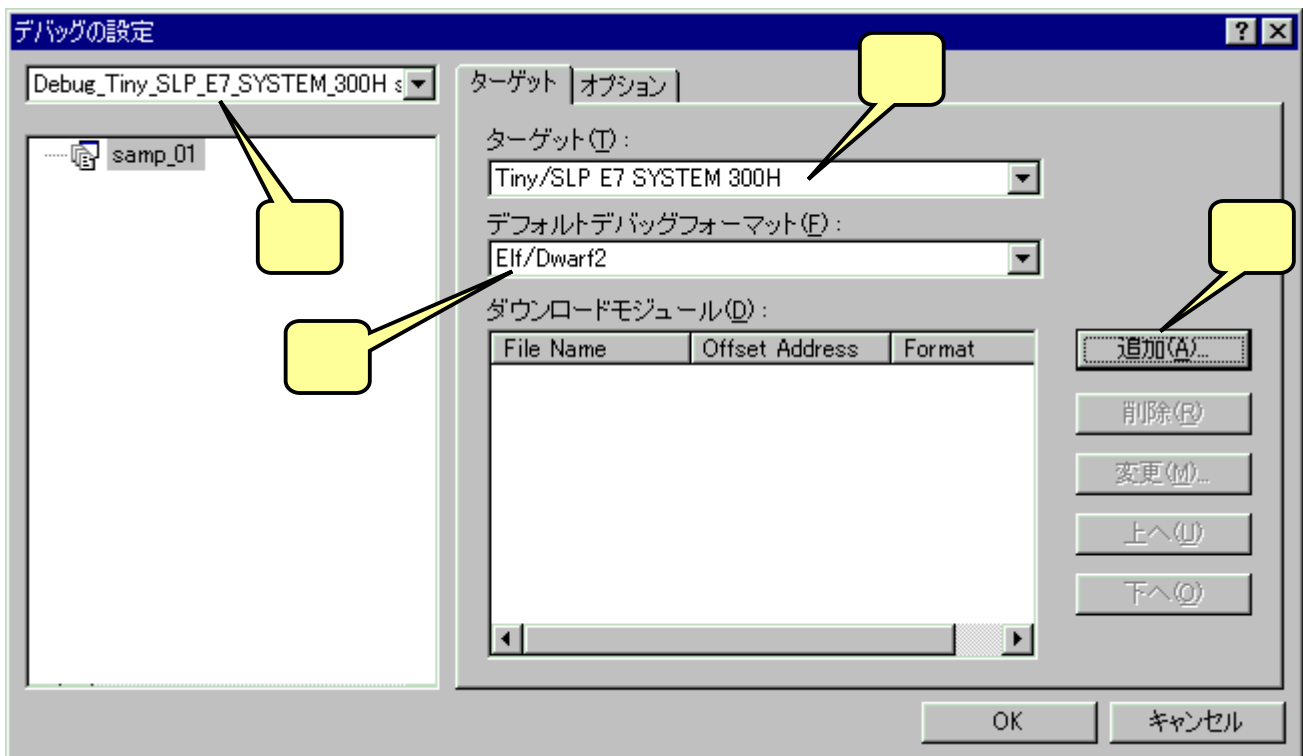
付録 8, 9, 10, 11 は旧マニュアル(Ver2.xx)の 5 章から 8 章までをそのままコピーしたものです。「無償版コンパイラ」や「E7」をご利用のお客様にとっては必要な資料と考え、付録ではありますがマニュアルに残すことにしました。現在ご利用の開発環境に応じてご利用いただければ幸いです。

それでは、'samp_01.hws' をダブルクリックして“HEW”を起動してください。先ほど入力・コンパイルした時の画面が再現されるはずです。

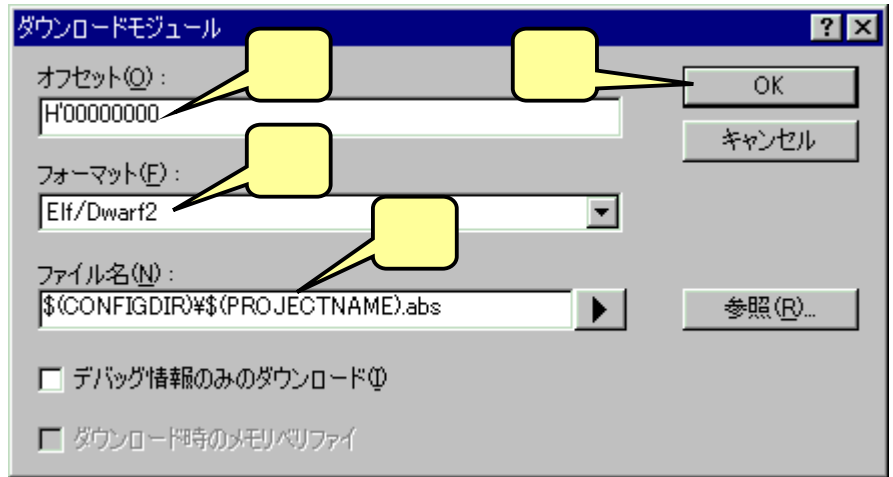
次に、“E7 エミュレータ起動時の設定”を行ないます。メニューからデバッグの設定を選択して下さい。



すると、'デバッグの設定' ダイアログボックスが開きます。① 'デバッグセッション' ドロップダウンリストボックスから 'Debug_Tiny_SLP_E7_SYSTEM_300H_session' を選びます。② 'ターゲット' ドロップダウンリストボックスから 'Tiny/SLP E7 SYSTEM 300H' を選びます。③ 'デフォルトデバッグフォーマット' ドロップダウンリストボックスから 'Elf/Dwarf2' を選びます。④ 'ダウンロードモジュール' を追加するため '追加' ボタンをクリックします。

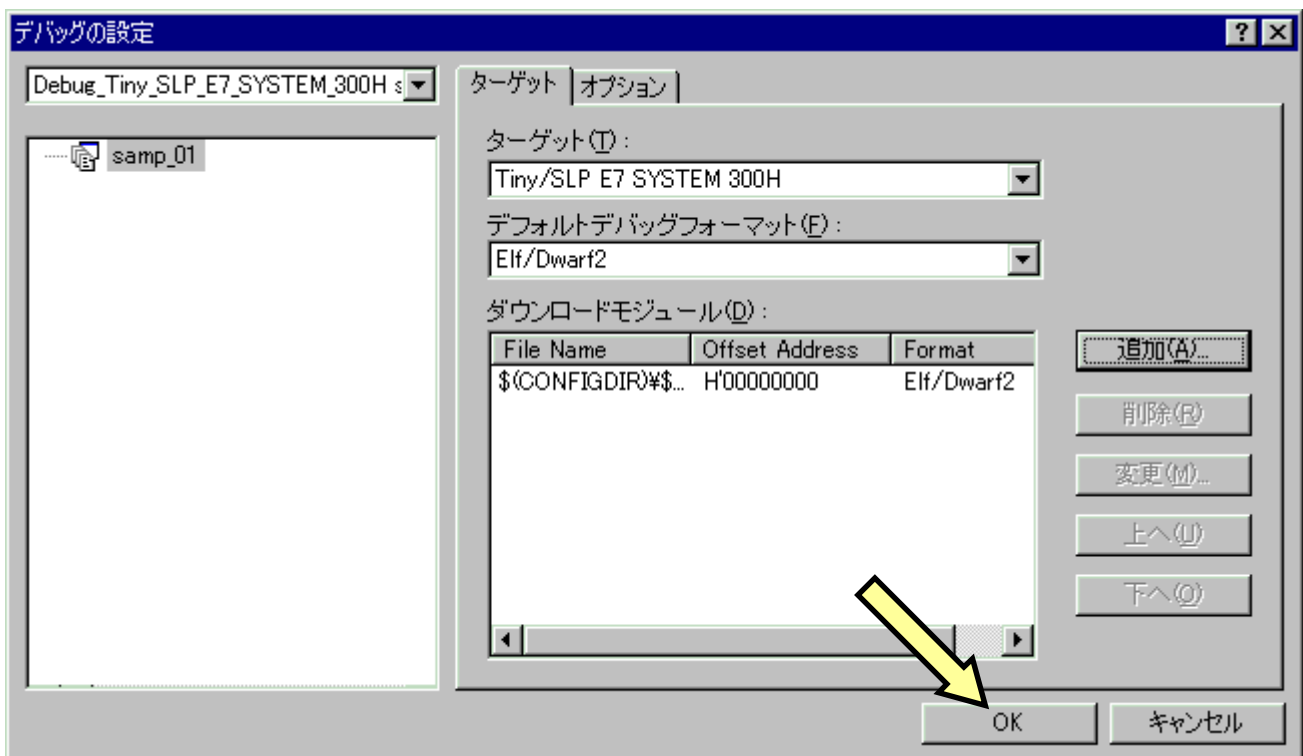


‘ダウンロードモジュール’ダイアログボックスが開きます。①‘オフセット’を‘H'00000000’に設定、②‘フォーマット’ドロップダウンリストボックスから‘Elf/Dwarf2’を選択、③‘ファイル名’に‘\$(CONFIGDIR)¥\$(PROJECTNAME).abs’を入力して、④‘OK’をクリックします。



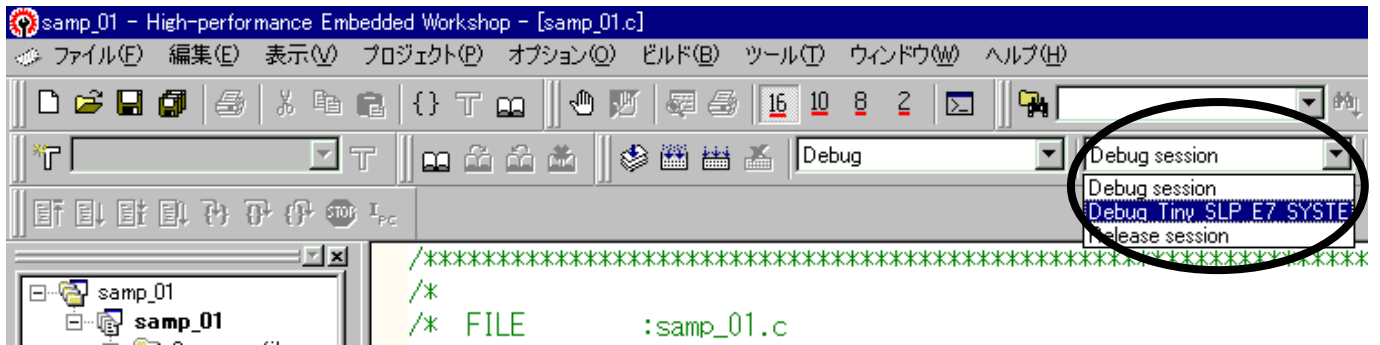
注意:③のときに、右のプルダウンメニューを使って入力した場合、‘¥’と‘.abs’を忘れやすいので確認して下さい。

そうすると、‘デバッグの設定’ダイアログボックスは次のようになります。‘OK’をクリックしてダイアログボックスを閉じます。



これで、全ての設定は終わりました。次は、いよいよ“HEW”をデバッグモードにして“E7”に接続します。まず、TK-3687の電源はオフにしてください。“E7”とパソコンをUSBケーブルで接続します。そして、“E7”とTK-3687のCN13(14ピンコネクタ)を付属のケーブルで接続します。なお、この時点ではまだTK-3687の電源はオフのままにしておいてください。

ツールバーの中、下図丸印のリストボックスから 'Debug Tiny SLP E7 SYSTEM 300H session' を選択します。そうすると、“E7”との接続が開始されます。



右のような警告が出ますが、'はい' をクリックします。



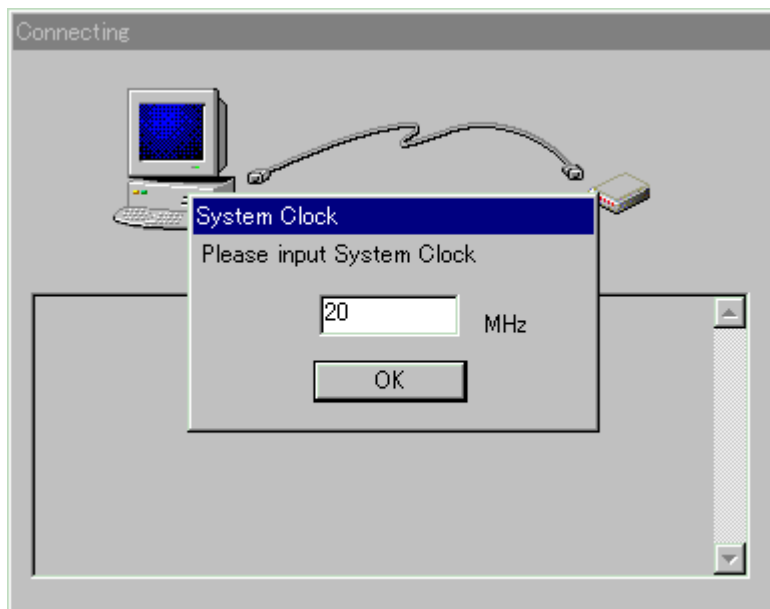
'Select Emulator mode' ダイアログボックスが開きます。内容が右図の通りか確認して 'OK' をクリックします。



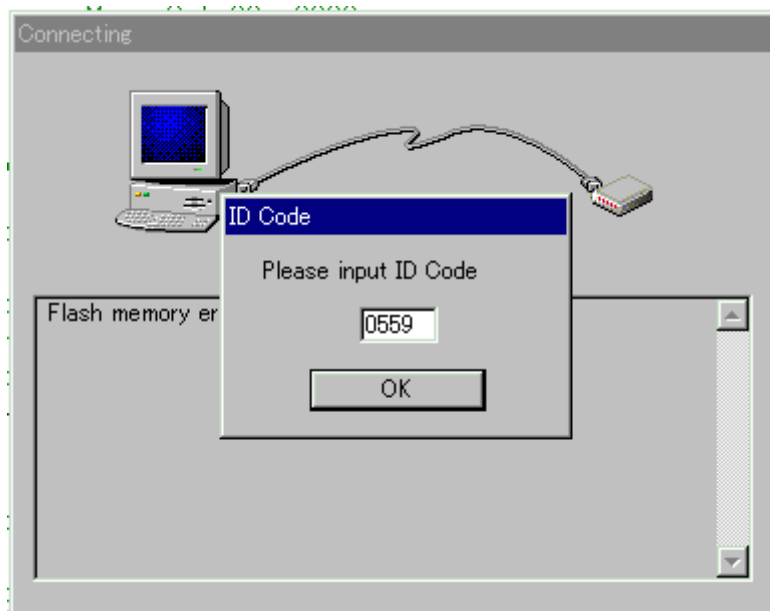
右のダイアログボックスが出たら TK-3687 の電源をオンします。それから、'Enter' キーを押すか、'OK' をクリックします。



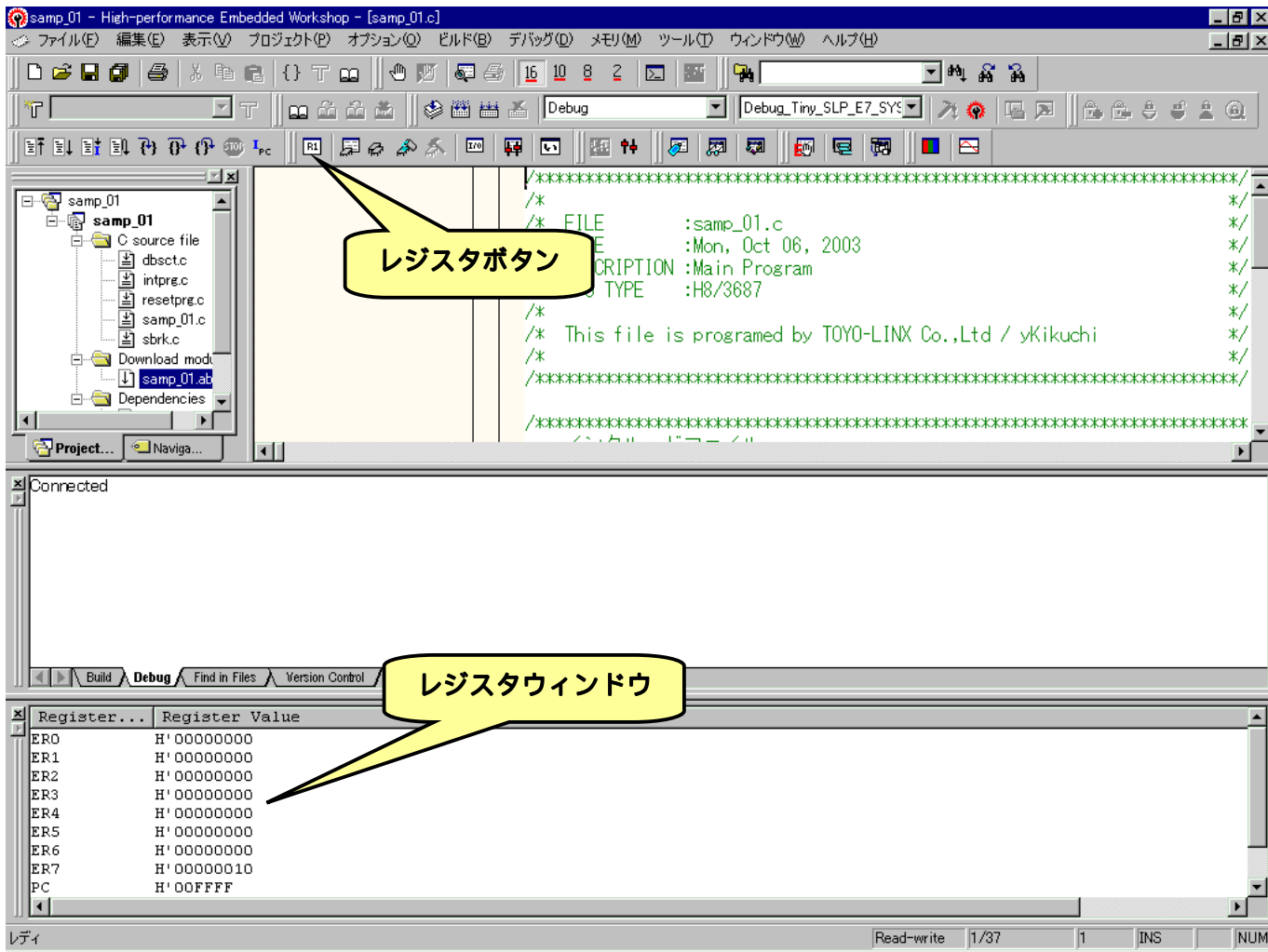
システムクロックの周波数を入力します。
TK-3687 の場合は 20MHz です。入力したら
'OK' をクリックします。



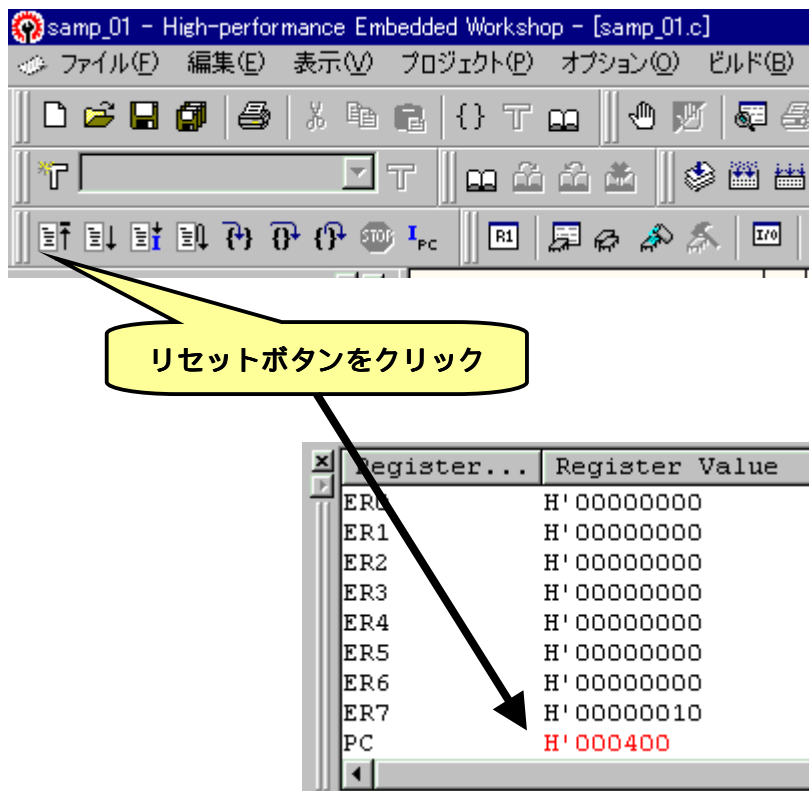
任意の ID コードを入力します。入力した
ら 'OK' をクリックします。



続いて、プログラムを実行してみましょう。この時点ではプログラムカウンタもスタックポインタも不定のため、あらかじめ指定する必要があります。ツールバーのレジスタボタンをクリックして、レジスタウィンドウを開きます。



まず、PC(プログラムカウンタ)を先頭アドレスに設定します。ツールバーのリセットボタンをクリックします。すると、リセットベクタアドレスの値(この場合 H'000400)がセットされます。



次に ER7(スタックポインタとして使用される)を設定します。レジスタウィンドウの ER7 をダブルクリックするとレジスタダイアログボックスが開きます。スタックポインタの初期値(H'F000)を入力して'OK'をクリックするとセットされます。

ER7をダブルクリック

Register...	Register Value
ER0	H' 00000000
ER1	H' 00000000
ER2	H' 00000000
ER3	H' 00000000
ER4	H' 00000000
ER5	H' 00000000
ER6	H' 00000000
ER7	H' 00000010
PC	H' 000400

Register...	Register Value
ER0	H' 00000000
ER1	H' 00000000
ER2	H' 00000000
ER3	H' 00000000
ER4	H' 00000000
ER5	H' 00000000
ER6	H' 00000000
ER7	H' 0000F000
PC	H' 000400

レジスタ - [ER7]

値(V):

データ形式(S): レジスタ全体

OK

キャンセル

ここまでセットしたら、本格的なデバッグが可能になります。一例として'カーソルまで実行'の手順を説明します。
 ①プログラムを停止したい行にテキストカーソルをおく。②ツールバーの'カーソルまで実行'ボタンをクリックする。

The screenshot shows the IDE interface. The source code editor displays the following code:

```

/*****
メインプログラム
*****/
void main(void)
{
    P_PORT.PCR5.BYTE = 0xff;

    while(1){
        c = a + b;
        P_PORT.PDR5.BYTE = c;
    }
}
  
```

The memory window on the left shows the current execution address at 0x00000808, with a yellow callout pointing to the start of the while loop in the code.

プログラムが停止した

The screenshot shows the IDE after the program has stopped. The source code editor shows the same code as above, but the cursor is now at the beginning of the while loop. The memory window on the left shows the current execution address at 0x00000814, with a yellow callout pointing to the start of the while loop in the code.

あとは、トレース実行で1行ずつ実行していきます。‘P_PORT.PDR5.BYTE = c;’を実行すると、ポート5のLEDに計算結果が表示されます。

“E7”はその他にも、ステップ実行、ブレークポイント設定、メモリーリード・ライト、I/O リード・ライト、C のラベルによる変数のウォッチ機能など、本格的なデバッグに必要な機能を十分に備えています。詳細については、“E7”のユーザーズマニュアルをご覧ください。

“E7”でダウンロードした時点で‘H8/3687’のフラッシュメモリにプログラムが書き込まれていますので、“E7”を接続しないでTK-3687の電源をオンにすると書き込まれたプログラムを実行します。

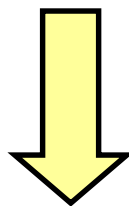


本章の最後に

以上が HEW におけるプログラム作成から動作確認までの手順です。ここでは、‘samp_01’を例にし説明をして来ましたが、これで一通りプログラムの作成手順が飲み込めたのではないのでしょうか。この後は、各自で簡単なプログラムにトライして見て下さい。

C の制御文(if, for, while など)が理解できれば、第1ステップ合格です。また、実習・応用プログラムを9章以降に載せてありますので、それらを参考にすれば、入力/出力ポートの制御やより高度なプログラミング・テクニックを身に付けることが可能です。

文中では、第2の関門は出てきませんでした。第1の関門をクリアしダウンロード・実行して、旨く動作しないこともあります。それが第2の関門です。プログラムの内容にも依りますが、第2の関門をクリアすることが極めて大変な辛い仕事になります。そこで机上デバッグやモニタを駆使してバグを探すわけですが、一番重要なことは‘何か変だ’とか‘何かちょっと違う’といった些細な疑問や変化を決して見逃さない事です。それを心がければ、バグを見つけることはそう大変なことではありません。

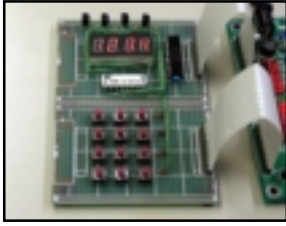


次章にお進みください。

付録 8, 9, 10, 11 は旧マニュアル(Ver2.xx)の5章から8章までをそのままコピーしたものです。「無償版コンパイラ」や「E7」をご利用のお客様にとっては必要な資料と考え、付録ではありますがマニュアルに残すことにしました。現在ご利用の開発環境に応じてご利用いただければ幸いです。

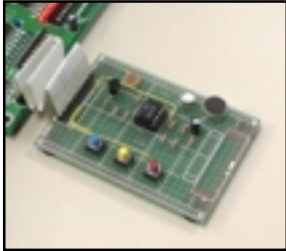
～～～ オプション品のご紹介 ～～～

★ ダイナミックスキャンによる 7 セグメント LED 表示&マトリックスキーキット: B6086(¥3,150-)



PIO を使用したダイナミックスキャンでの 7 セグメント LED 表示と 3×4 のマトリックスキー入力の学習キットです。学習内容はユニバーサル基板にハードを実装するところから始め、PIO の基本的な使い方・ダイナミックスキャンの考え方・マトリックスキーの入力方法、それらの応用として RTC を用いた時計プログラムを作成します。

★ 光センサとコンデンサマイク使用 A/D 変換キット: B6087(¥2,100-)



内蔵 A/D コンバータを使用した明るさと音を変換する学習キットです。学習内容はユニバーサル基板にハードを実装するところから始め、A/D コンバータの基本的な使い方・光センサでの直流信号の A/D 変換・コンデンサマイクを使用した交流信号の A/D 変換とパソコンへの送信をプログラムします。又、パソコン側の受信ソフトとして、Win32API による方法と Excel の VBA による方法を示します。

★ DC モータの回転制御: B6088(¥3,780-)



ツインモータギアボックスとドライバ IC・TA7279P/AP を用いた DC モータ制御の学習キットです。フォトフレクタによりタイヤの回転数を検出することが出来ます。学習内容はユニバーサル基板にハードを実装するところから始め、単純な PIO での制御、PWM での制御、応用として回転数一定制御をプログラムしていきます。

★ AC パワーコントローラキット: B6089(¥3,150-)

★ RS-485 実習キット: B6085(¥3,150-)

★ タイマ&LED ディスプレイキット: B6092(¥3,150-)

★ I/F トレーニングユニット: B6084(¥12,000～15,000-)

★ その他のオプション:

AC アダプタ(¥2,100-), RS232C ケーブル(9 ピン-9 ピン, ¥1,155-), ユニバーサル基板セット(¥1,050-, A8 版, ケーブルつき) 等

(価格は全て税込価格です)

★ご質問、お問い合わせはメール、または FAX で、

(株)東洋リンクス

〒102-0093 東京都千代田区平河町 1-2-2 朝日ビル

TEL: 03-3234-0559 / FAX: 03-3234-0549

URL: <http://www2.u-netsurf.ne.jp/~toyolinx>

E-Mail: toyolinx@va.u-netsurf.jp

※本書の内容は将来予告無しに変更することがあります(2006 年 1 月改定)