# 第22章 無償評価版 HEW を最大限に活用する三つの「裏技」

## —— オブジェクト・サイズ 64K バイト制約を克服する

**インストールして 60 日たつと 64K バイト以上のオブジェクト・サイズを生成できなくなる無償評価版 HEW の制約を突破する「裏技」を三つ紹介します.**

## 無償評価版 HEW の制約

### ● インストールして 60 日たつと 64K バイト以上のオブジェクト・サイズを生成できなくなる

無償評価版の C/C++ コンパイラ・パッケージには 60 日間という制限があります．それ以降も HEW を使うことはできますが，最適化リンカがオブジェクト・サイズで 64K バイト以上のファイルを作らなくなります．「使えね～」とお嘆きの方も多いかと思います．ルネサスが許可してくれないなら，テクニックで克服しましょう．

### ● 無償の HEW で制約を克服する三つの方法

無償の HEW を制約なしで使うには以下の三つの方法があります．

① 画像などデータが大きいことが原因で 64K バイト制約を超えてしまう場合
⇒配列ではなくポインタを使う
② プログラムが単に 64K バイトを超えてしまう場合
⇒ 64K バイト以下に分割してビルドし，アドレスを指定しながらフラッシュ・メモリに書き込む
③ ①と②両方
⇒制約のない GCC コンパイラを HEW から使うか，①と②を組み合わせる

## 裏技その 1 ：コード・サイズや期間の制限がない GCC コンパイラを HEW から使う

GNU ツールにはルネサス C/C++ コンパイラのように期間によるオブジェクト・サイズや機能に制限がありません．しかし HEW と異なり Eclipse や Cygwin などを使った環境の構築が難しい面があります．そこで，HEW を使って GNU を利用する方法を紹介します．デバッグにはシリアル・デバッガが使えます．

### ● GNU ツールを選択するだけ

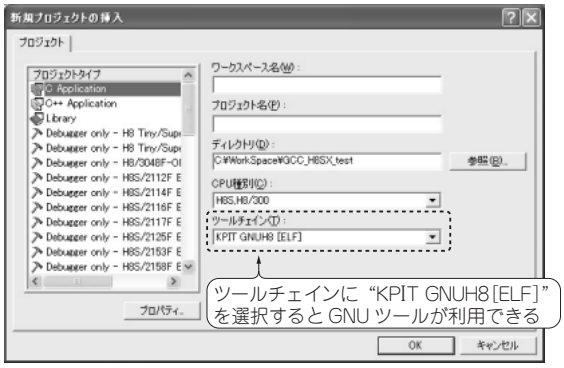
さあ，インストールしましょう．といっても第 3 章，第 4 章で HEW と KPIT GNU をインストールしていればほかにやることはありません[注]．HEW を起動すると GNU がすぐに使えます．

新規にプロジェクトを作成するには，**図 1** に示すように HEW を起動し，ツールチェインで「KPIT GNU H8[ELF]」を選択します．

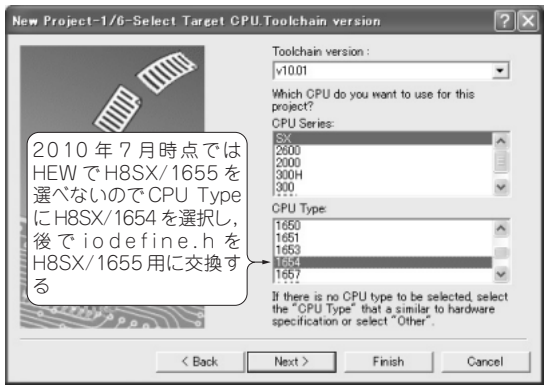
プロジェクト・タイプに「Application」を選択するとスタートアップも自動的に生成してくれます．残念なことに H8SX/1655 はメニューにないのでメモリ・サイズや周辺機能がほとんど同じ H8SX/1654（USB なし）を選択します．周辺機能レジスタを定義している iodefine.h は後で H8SX/1655 に変更すれば問題なくなります．

コンパイラ・パッケージが GNU に変更されただけなので HEW の使い方は変わりません．ビルドからシリアル・デバッガ，ROM 書き込みまでの手順を**図 2** に示します．

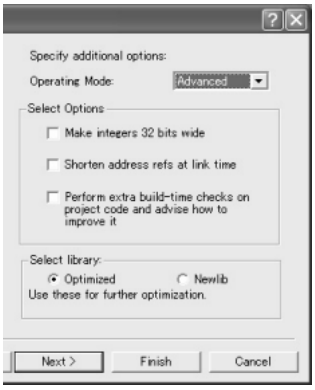
注：KPIT GNU をインストールする際に HEW への登録のチェックを ON しておかないとすぐには使えません．本書の手順通りに進めている場合はチェックを ON していないので，チェックを ON にして再インストールする必要があります．



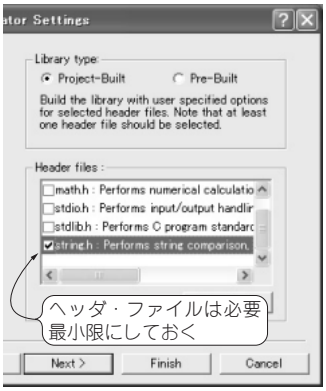

（a）KPIT GNUH8を選択




（b）H8SX/1654（H8SX/1655がないため）を選択

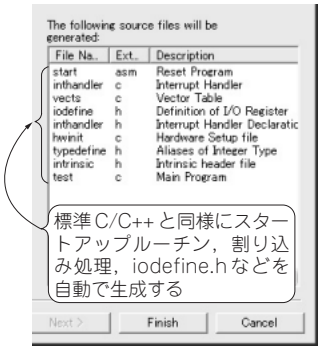



（c）オプション設定はデフォルトのまま

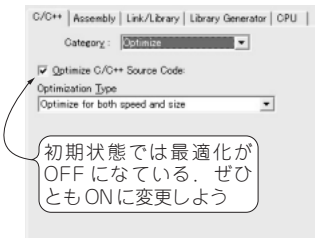



（d）ライブラリ選択

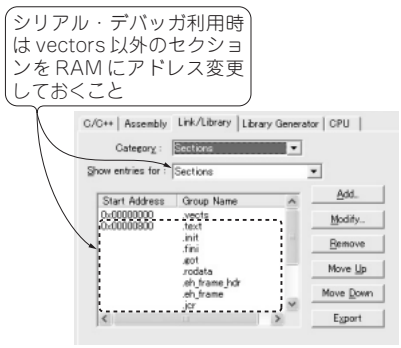

（e）デバッグ環境指定




（f）プロジェクト・ジェネレータ生成結果




（g）最適化オプションの変更




（h）セクション・アドレスの設定
（シリアル・デバッガ利用時は変更）

**図1　KPIT GNUツールチェインによる新規プロジェクト作成**

### ● GCCのデメリット：コード効率は最適ではない

ルネサスC/C++コンパイラはマイコン・メーカ純正なので，H8SX/1655マイコン用のコンパイルが行われます．しかし，GCCは汎用なのでコード効率などが悪くなる可能性があります．

また，GCCをHEWから操作できるといってもオプション設定はGCC流です．ルネサスC/C++コンパイラとすべて同じように使えるわけではありません．

付属CD-ROMのadd¥Workspaceフォルダにサンプルとして LED_GCC_serial_monitor プロジェクトが入っているので，C:¥Workspaceフォルダにコピーして試してみることができます．

## 裏技その2：配列ではなくポインタを使う

### ● LCD表示用データを配列に割り当てると64Kバイトを超える

まず無償評価版の制限にぶつかるのがLCD表示フレーム・バッファ(LCDの表示画面用メモリ)のアクセスです．タッチ・パネルLCD拡張基板TBのLCDは表示画面サイズがQVGA(320 × 240ドット，16ビット/ドット)です．これを(*x*，*y*)のunsigned short型の2次元配列で扱うとプログラムするときに直感的に分かりやすくなります．変数宣言は，

```
volatile unsigned short
```

最適化されないため必ず記述通りアクセスされる

配列の要素は16ビット

```
global_lcd_framebuffer[240][320];
```

320個の連続したアドレスの塊が240個ある

となります．

ここで問題なのは，QVGA表示画面が320 × 240 × 16/8 = 153.6 kバイトあることです．上記のように宣言すると配列はセクションB(初期値なし変数が割り当てられる領域)に割り当てられます．すると，評価版HEWでは60日間経過後，最適化リンカでサイズ・オーバと判定されロード・モジュールが作成されなくなります(**図3**)．

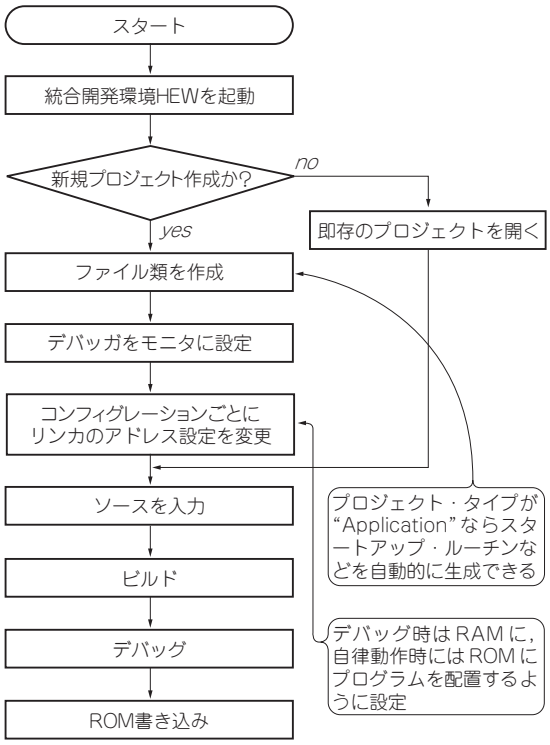
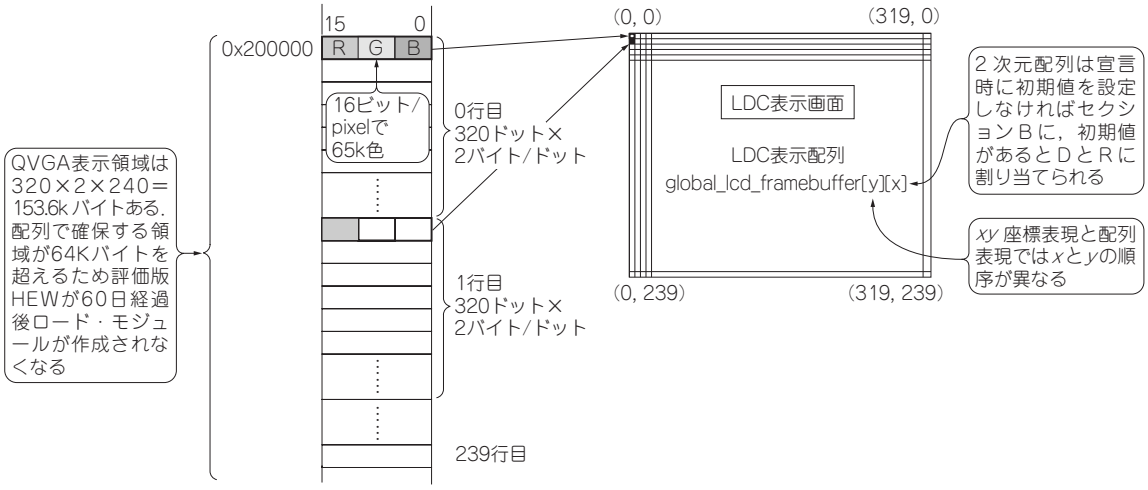



**図2　KPIT GNUツールチェインによる開発手順**

### ● 配列を使わずにポインタを使う

これを解消するには表示領域を変数に割り当てずに，メモリ・アクセスする必要があります．解決策は，配列をポインタ変数でアクセスすることです．もちろ




LCD表示用配列global_lcd_framebufferのメモリ・マップ

**図3　表示画面とプログラムの扱い**(配列とポインタ)
2次元配列は右に記述したインデックスが連続したメモリに割り当てられるので座標では*x*軸となる．つまり位置指定は，配列名[*y*位置][*x*位置]となり，数学表現とは異なるので注意が必要．

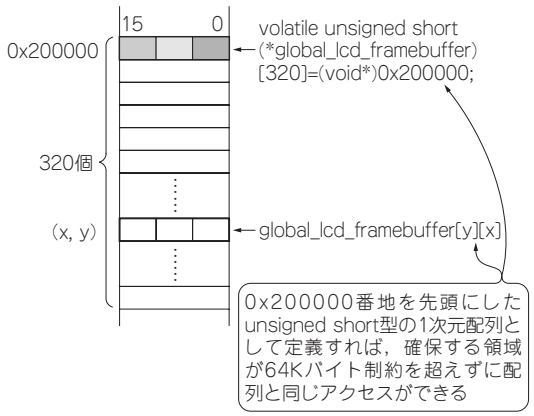



**図4　ポインタによる配列アクセス**

ん配列を宣言してはいけません．**リスト1**に示すように，ポインタ変数のみを宣言します．これでメモリの指定した番地に自由にアクセスできます．周辺機能レジスタにアクセスする際に用いる手法と同じです．

配列ではセクションBを実体としてメモリ確保しますが，ポインタは実体を伴いません．ポインタ変数が一つあるだけです．つまり，ポインタ変数がどこのどのくらいの大きさの範囲を指し示しているかをC/C++コンパイラは知りません，管理していないのです．だからQVGAサイズの画面であっても無償評価版HEWの64Kバイト制約を受けずにアクセスできます．**図4**にポインタを用いたアクセス方法を示します．宣言のみを変更するだけで，関数内の記述変更不要で簡単に評価版HEWの制限を超えられます．

### ● 自分でメモリを管理しないと暴走する

ただし，注意も必要です．ポインタ変数がどこを指し示しているかは誰も管理してくれません．何でもできますが逆に怖い存在でもあります．表示領域から外れたところもアクセスできますから，スタック領域や変数領域を壊しかねないのです．プログラムが暴走する可能性もあります．

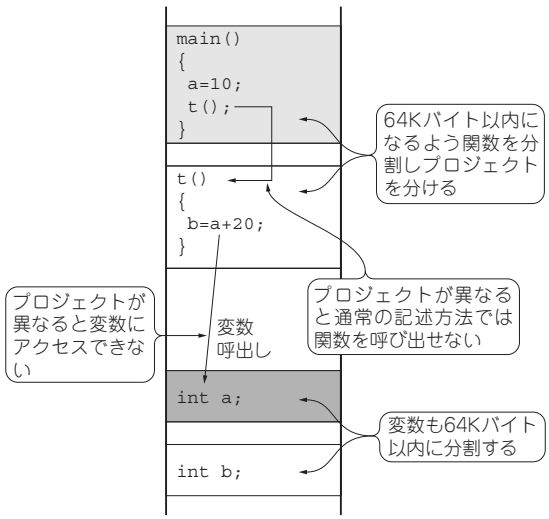



**図5　64Kバイト未満の複数のロード・モジュールに分割して制限を回避**

**リスト1　ポインタ変数を用いたアクセス**

```
unsigned short *p; //ポインタ変数を宣言
    //pが示す変数はunsigned short型
    p = (unsigned short *)0x200000;
    //ポインタ変数に0x200000を代入
    //pは0x200000番地を示す
    *p = 1234; //0x200000番地に1234を代入
    p++; //pをインクリメント
    //示している変数の型がunsigned shortのため，
    //インクリメントすると2番地分番地の大きい方へ
    //移動し，0x200002番地を示す
    *p = 5678; //0x200002番地に5678を代入
```

### ● プログラムのコツ：ポインタで直感的に分かりやすく座標を指定する

**リスト1**の方法でも構いませんが，配列のインデックスを用いたアクセスに比べ分かり難いですね．やはり*xy*座標系で位置指定した方が直感的で分かりやすいです．そこで次のように宣言を変更し，配列を利用しているときと同じ記述が利用できるようにします．

```
volatile unsigned short(*global_lcd_framebuffer)
[320] = (void *) 0x200000;
```

### ● アセンブリ言語では領域を確保せずに簡単にメモリ・アクセスできる

アセンブリ言語では領域を確保せずに，

```
MOV.W #1234,@H'200000
```

のように記述できます．

## 裏技その3：分割してビルド・書き込みする

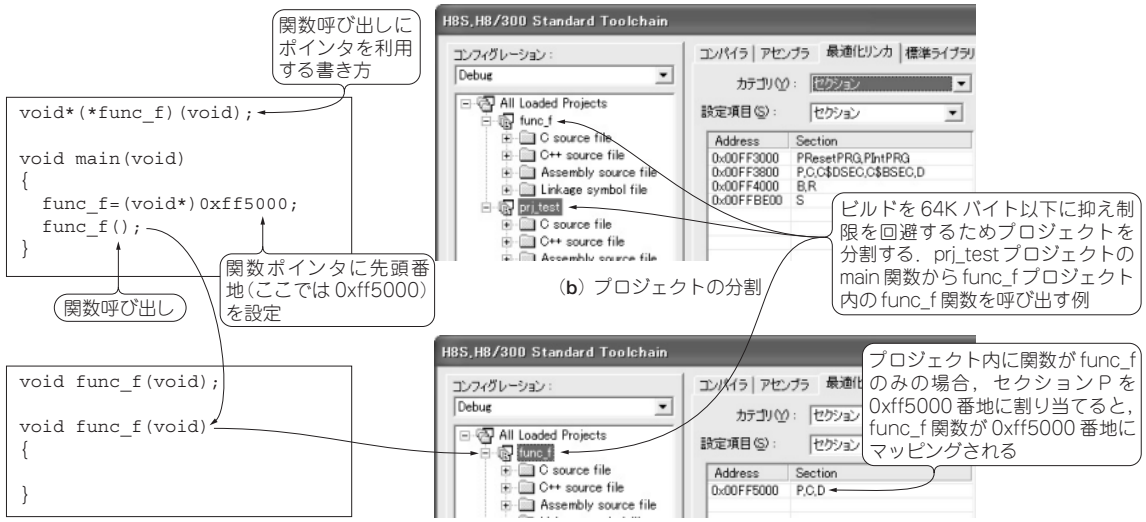
### ● プログラムを分割して64Kバイト以下にする

プログラム・サイズが64Kバイトを超えてしまったら，プロジェクトを64Kバイト未満にするよう分割し，複数のロード・モジュール(H8SXが実行可能なファイル)をダウンロードすることで制約を気にしなくてよくなります．

この方法では**図5**に示すように，通常の記述でプロジェクトをまたぐ関数呼び出しや変数アクセスは行えません．

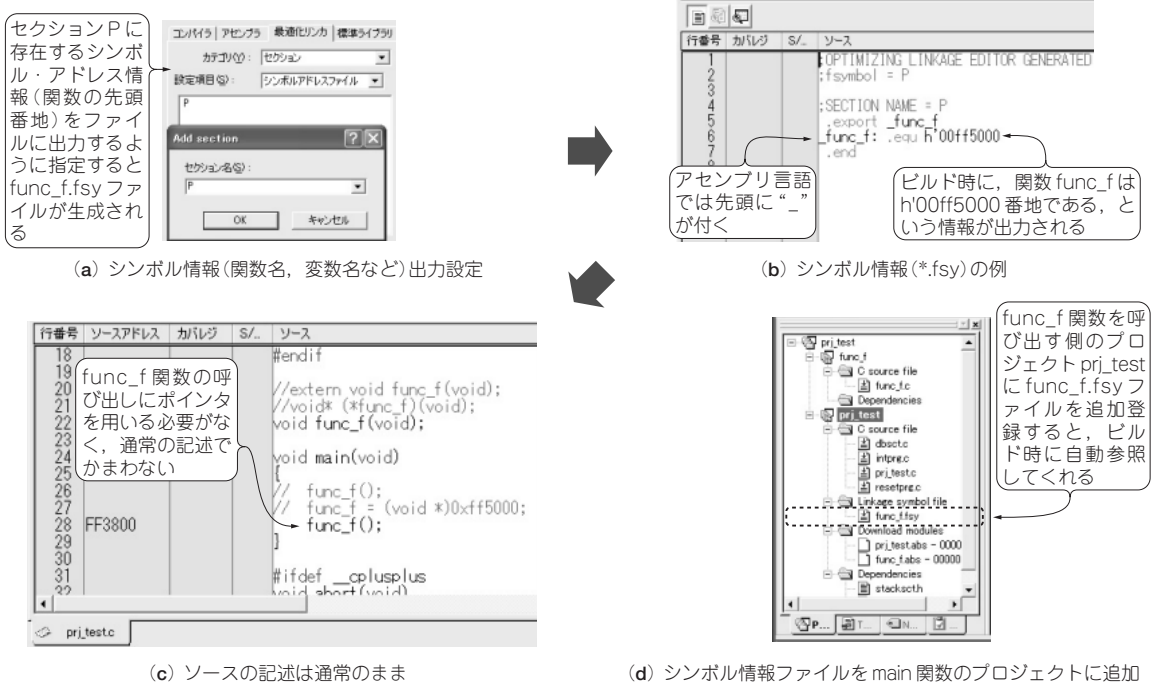
### ● 関数や変数は名前ではなくアドレスで呼び出さなければならない

複数に分かれたプロジェクト間で変数名や関数名を使って相互にアクセスするときは，番地の情報を使い




（a）ソース・プログラム　（b）プロジェクトの分割　（c）関数fのマッピング

**図6　ポインタを用いたプロジェクトをまたぐ関数や変数の呼び出し方法**




（a）シンボル情報（関数名，変数名など）出力設定　（b）シンボル情報（*.fsy）の例

（c）ソースの記述は通常のまま　（d）シンボル情報ファイルをmain関数のプロジェクトに追加

**図7　シンボル・ファイル生成によるプロジェクト間の関数呼び出し**

ます.つまり関数や変数の番地が分かればよいのです．変数や関数の番地はリンクすると確定します．片方のプロジェクトをビルドし，その結果を確認してもう一方のプロジェクトにその番地情報を設定，という具合にすれば解決できます．

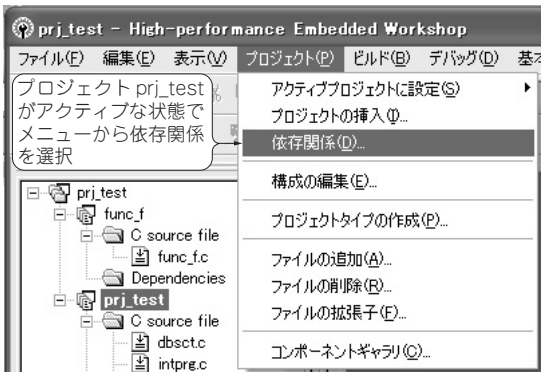
実際には，すべての関数や変数にこのような作業を行うことはやってられません．

### ● 関数や変数のアドレスが記されたファイルを出力して参照させる

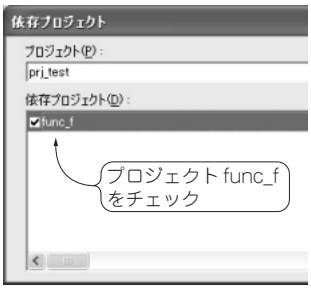
そこで片方のプロジェクトには，

#pragma section 新セクション名

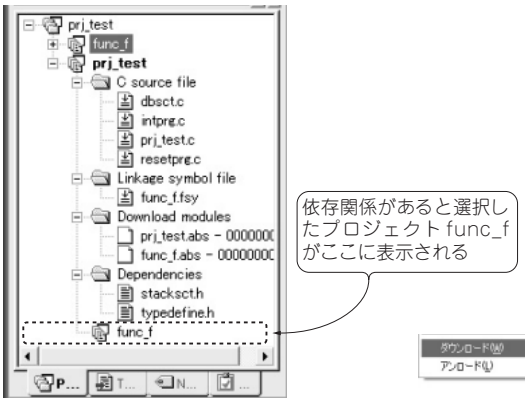
で新しいセクションを設定し，そのセクション名の先頭番地を最適化リンカのオプションに設定します．**図6**のように変数ごとにセクション名を付与することで




（a）依存関係を登録




（b）依存プロジェクトの指定




（c）設定後のプロジェクト・ウィンドウ


（d）ダウンロードはすべて（All Download Modules）を選択

**図 8　プロジェクトの依存関係設定で関連プロジェクトをまとめてビルド，ダウンロード**

解決できます．

しかしこの方法だとソースが複雑になります．もっとシンプルに，ソースに大きな手を加えないで済む方法があります．変数を宣言し，**図 7** の手順で実体のあるプロジェクトからシンボル情報を切り出しファイルにします．そのファイルをもう一方のプロジェクトが参照することで，ソースには手を加えなくても済みます．

これでプロジェクト間にまたがる関数呼び出しが行えます．プロジェクト間の変数アクセスも同様にシンボル・ファイルの生成と参照によって解決します．もう評価版の制限はないのと同じです．

### ● プログラムのコツ：ほかのプロジェクトの変更を自動的に反映させる

もう少し便利にしておきましょう．プロジェクト func_f の変更を別のプロジェクト prj_test に自動的に通知する機能を設定しておきましょう．**図 8** に示すように，この二つのプロジェクトには依存関係があることを HEW に設定しておくことで，prj_test をビルドすると func_f に変更があれば先にビルドしてから prj_test をビルドします．いちいちアクティブ・プロジェクトを変更しなくて済みますから時間の節約になります．

## ■ メモリに書き込む方法

### ● デバッグ時は RAM に「全てダウンロード」

これですべての問題が解決できました．しかし，プロジェクトごとにロード・モジュールが出力されますから，二つのロード・モジュールが存在します．ダウンロード指定を二つにし，シリアル・デバッガでダウンロードする場合は「全てダウンロード」を選択します．

### ● フラッシュ・メモリに書き込むときは mot ファイルを手で合体させる

デバッグが終わり，さあフラッシュ・メモリに書き込んで自律動作させようとすると問題があります．何と，FDT は書き込む mot ファイル（S フォーマット）を一つしか選べません．どうするか．

そこで拡張子 mot ファイルを結合します．MOT

**リスト 2　複数のプロジェクトから生成した mot ファイルを一つに合体すれば，FDT を使って H8 マイコンにプログラムを書き込める**

```
S00E000066756E635F6620206D6F74F0
S206FF50005470E6
S9030000FC
```

S0：ファイルの先頭
構成：行サイズ(2 文字)，ファイル名，チェックサム(2 文字)

S2：先頭アドレス(16M バイト空間用)と機械語コード
構成：行サイズ(2 文字)，アドレス(6 文字)，機械語コード，チェックサム(2 文字)

S9：ファイルの終了
構成：行サイズ(2 文字)，エントリ・アドレス(4 文字)，チェックサム(2 文字)

(**a**) func_f.mot ファイル

```
S00E000070726A5F746573746D6F7436
S107000000FF3000C9
S107001000FF3016A3
S107001400FF30189D
(途中省略)
S210FF38507B941B5C46FA1F9045E0542654
S210FF385C00FF387000FF387000FF4000CF
S20CFF386800FF400000FF4000D6
S804FF3000CC
```

S1：先頭アドレス(64K バイト空間用)と機械語コード
構成：行サイズ(2 文字)，アドレス(4 文字)，機械語コード，チェックサム(2 文字)

S9：ファイルの終了
構成：行サイズ(2 文字)，エントリ・アドレス(4 文字)，チェックサム(2 文字)

(**b**) prj_test.mot ファイル

```
S00E000070726A5F746573746D6F7436
S206FF50005470E6
S107000000FF3000C9
S107001000FF3016A3
S107001400FF30189D
(途中省略)
S210FF38507B941B5C46FA1F9045E0542654
S210FF385C00FF387000FF387000FF4000CF
S20CFF386800FF400000FF4000D6
S804FF3000CC
```

呼び出される側のファイル
func_f.mot の中身を追加する

(**c**) 結合した prj_test.mot ファイル

ファイルの S フォーマットは文字で構成されているので，テキスト・エディタで簡単に結合できます．**リスト 2**(**a**)を見てください，これが func_f.mot ファイルをテキスト・エディタで開いた状態です．

S フォーマットは S0 レコード(行)で始まり，S9 レコードで終了します．**リスト 2**(**b**)は prj_test.mot ファイルです．

二つのファイルを結合するには，**リスト 2**(**c**)のように先頭の S0 レコードを一つにし，最後の S8 または S9 レコードを一つにします．

これで FDT から H8 マイコン内蔵フラッシュ・メモリに書き込んで自律動作することができます．

＊

コンパイラはファイル単位でオブジェクト・ファイルを作成します．リンカはプロジェクト単位で，ファイルをまたいだ関数や変数の呼び出しを関連付けます．

プロジェクトをまたいだ関数や変数の呼び出しは HEW を使うと実現できます．ここでは 64K バイト制約を克服するためにプロジェクトを分割して関連付ける方法を紹介しましたが，ほかにもいろいろな場面で活用できると思います．応用範囲は広いので頑張ってみましょう．　**〈藤澤 幸穂〉**