ROM レス品にて、外部 FlashROM 品種を追加する場合の説明

Ｒｅｖ１．０１
ＤＥＦバージョン１０．１０Ａ仕様より

【対象ＣＰＵ】

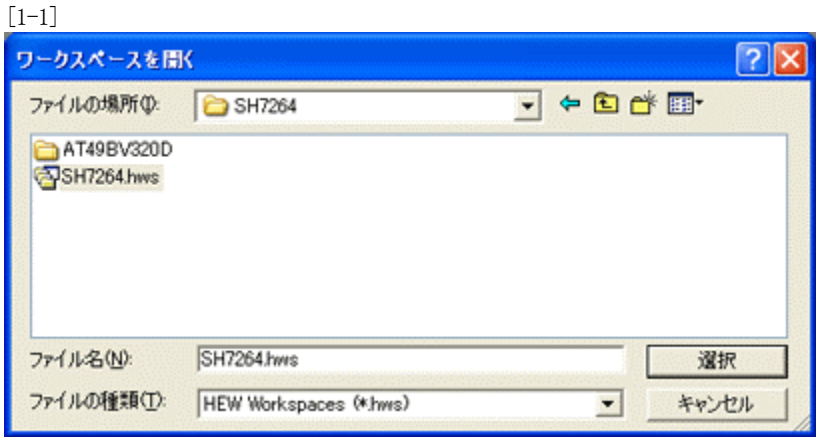
１．ＲＯＭレス品種が対象になります。（Ｈ８ＳＸ／１６５１、ＳＨ７２６４等）

【機能】

１．Ｈ８ＳＸ／１６５１は、サンプルで「ＴＣ５８ＦＶＭ５Ｔ」の対応ソフトを用意してあります。（ルネサスＣのみ対応）
２．ＳＨ７２６４は、サンプルで「ＡＴ４９ＢＶ３２０Ｄ」の対応ソフトを用意してあります。（ルネサスＣのみ対応）
３．ＨｅｗにてＦｌａｓｈＲＯＭの品種追加が出来るよう対応する。
４．ＦｌａｓｈＲＯＭ仕様にあわせたセクターイレーズとバイト書き込みをプログラミングするだけで追加が可能になります。
５．ＦｌａｓｈＲＯＭ対応ソフトをターゲット側に流し込み、Ｈ－デバッガと通信させるためのポートがＣＰＵ品種ごとに相違があるため、品種別に対応しています。

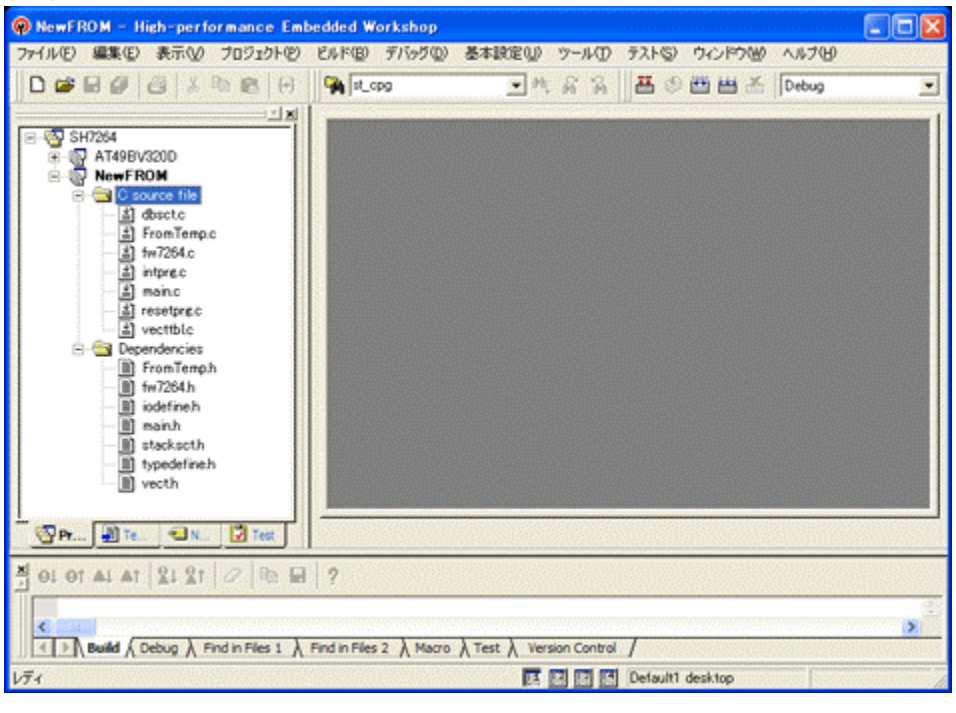
【品種追加前の準備】

１．ＡＨ７０００コントロールソフト（Ｖｅｒ１０．１０Ａ）のインストールＤＩＲにあるワークスペースを開きます。（ＳＨ７２６４での追加例）
［1-1］



＜デフォルトディレクトリ＞
"c:¥Program Files¥Aone¥DEF¥rom-custom¥SH7264"
になります。

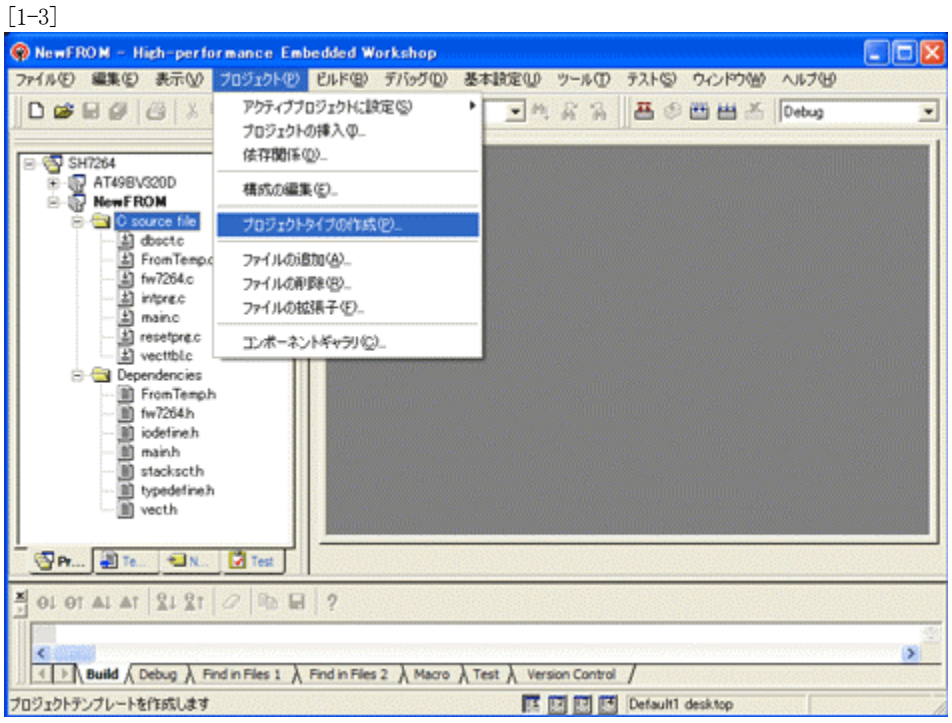
２．プロジェクト名「ＮｅｗＦＲＯＭ」をアクティブプロジェクトに設定します。
［1-2］



＜ＮｅｗＦＲＯＭ＞をマウスクリックし、右クリックのポップアップメニューから選択します。

3．「プロジェクトタイプの作成」をします。

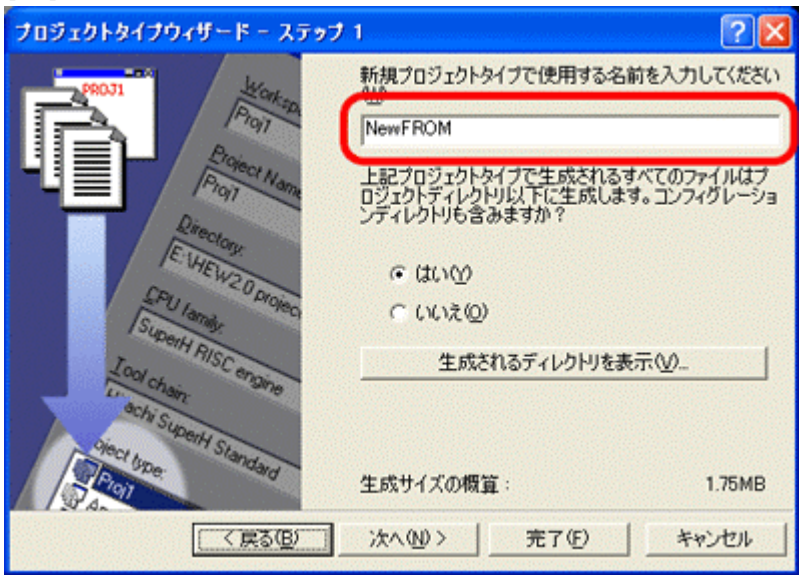
[1-3]



Ｈｅｗメニュー
＜プロジェクト＞ー＜プロジェクトタイプの作成＞をクリックします。

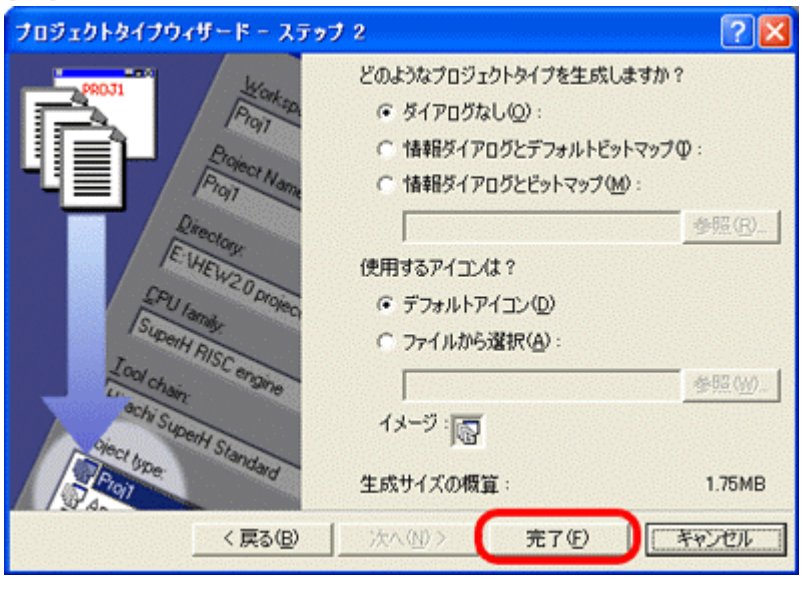
4．新規プロジェクトタイプで使用する名前を指定します。

[1-4]



＜ー「次へ」をクリックします。

[1-5]



＜ー完了をクリックします。

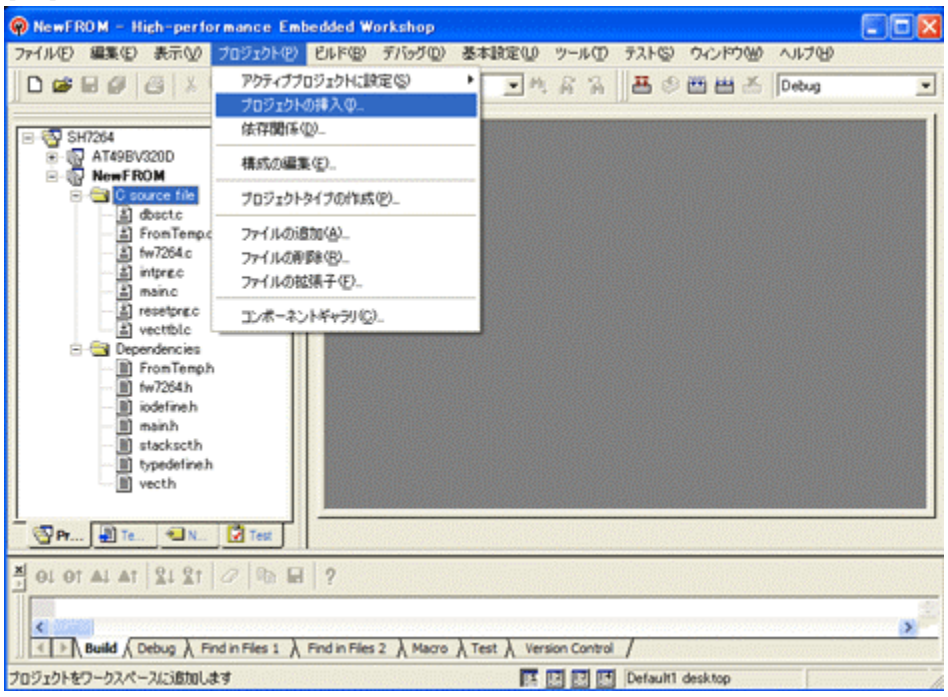
この操作で「品種追加前の準備」は完了です。

名前は、重複しないようにして下さい。他のＭＰＵ品種でも追加が必要な場合は、例として「ＮｅｗＦＲＯＭ＿７２６４」等が良いかもしれません。
ここの説明では「ＮｅｗＦＲＯＭ」としておきます。

【Ｆｌａｓｈ ＲＯＭ品種の追加】

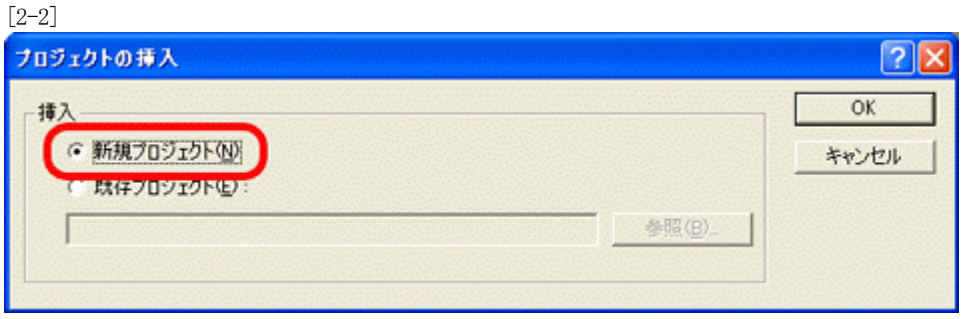
１．Ｆｌａｓｈ ＲＯＭ品種を追加するため、「プロジェクトの挿入」をします。

[2-1]



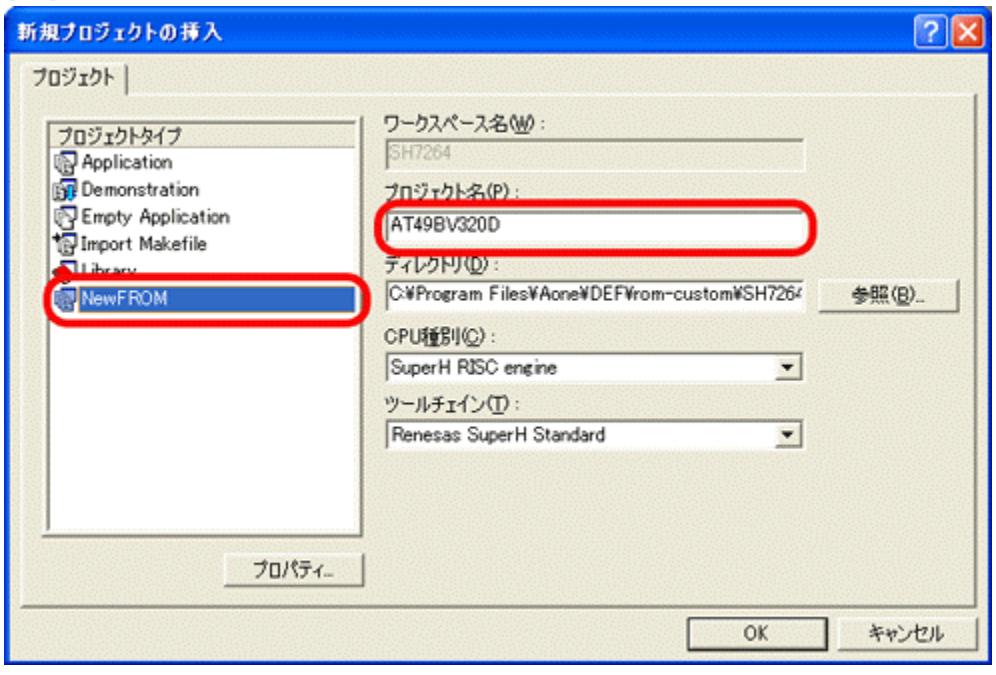
Ｈｅｗメニュー
＜プロジェクト＞ー＜プロジェクトの挿入＞をクリックします。

[2-2]



「新規プロジェクト」を指定して、「OK」をクリックします。

[2-3]

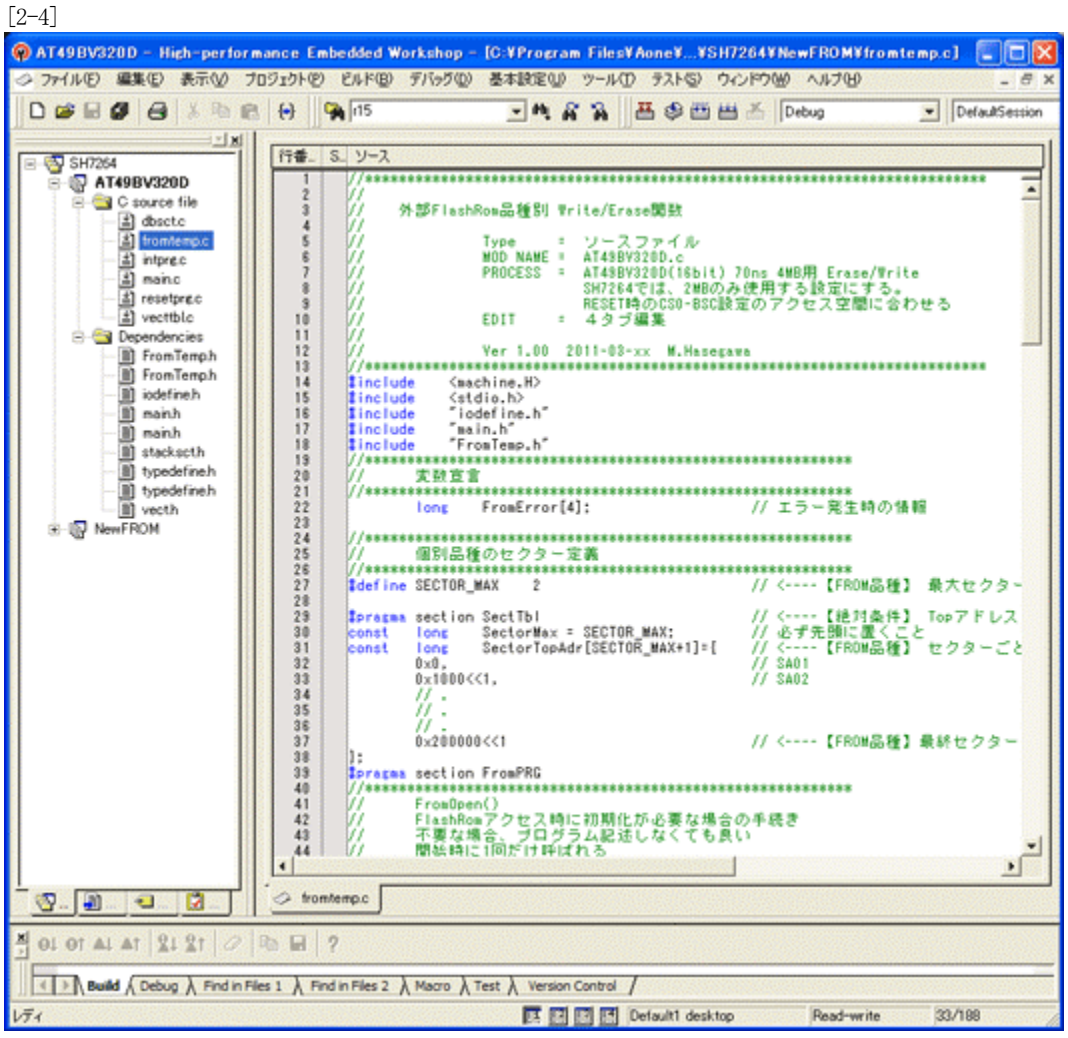


１）プロジェクトタイプを作成した「ＮｅｗＦＲＯＭ」に指定します。
２）プロジェクト名を指定します。推奨としてはFlashROM名が良いかと思います。

＜ー「OK」をクリックします。

２．新規プロジェクトに追加するＦｌａｓｈＲＯＭ品種の定義およびプログラムを作成します。

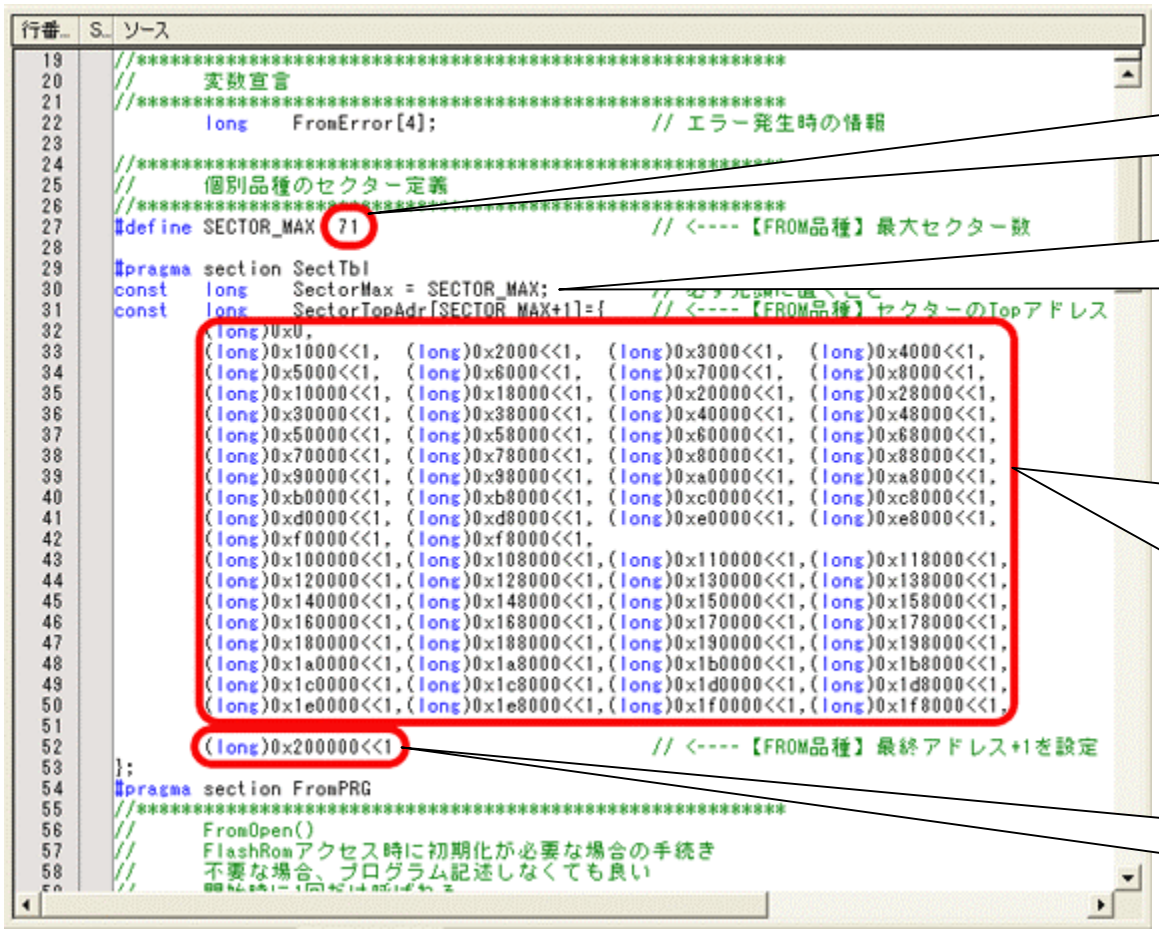
[2-4]



品種追加用テンプレートファイル「FromTemp.c」を開きます。

１）「FromTemp.c」に追加フラッシュＲＯＭのセクター情報を定義します。

[2-5]



「SECTOR_MAX」に使用するフラッシュＲＯＭのセクター（ブロック）数を定義します。

⚠
変数宣言の順番は変更しないで下さい。データ管理上、最初にセクター（ブロック）数が入るルールになっています。

変数「SectorTopAdr[]」に、
定義したセクター（ブロック）数ごとの先頭アドレスを定義します。
⚠
ＲＣのバージョンによっては、キャストなしで記述をすると「short」に丸められますので「(long)0x1000<<1」とキャストすることを推奨します。

最後にフラッシュＲＯＭの最終アドレス＋１したアドレスを定義します。

2）「FromTemp.c」に追加フラッシュＲＯＭの「オープン」「クローズ」関数を作成します。
[2-6]

```
行番… S… ソース
 50      };
 51      #pragma section FromPRG
 52      //********************************************************
 53      //      【FROM品種】
 54      //      FromOpen()
 55      //      FlashRomアクセス時に初期化が必要な場合の手続き
 56      //      不要な場合、プログラム記述しなくても良い
 57      //      開始時に1回だけ呼ばれる
 58      //********************************************************
 59      int FromOpen(void)
 60      {
 61          int     i;
 62          int     stat;
 63          Ushort  *from;                          // Word Access
 64
 65          CPG.FRQCR.WORD = 0x124;                 // デフォルト値
 66
 67          for(i = 0;i <= SectorMax-1;i++) {
 68              from = (Ushort *)SectorTopAdr[i];
 69              *from = 0x60;                       // Sector Unlock
 70              *from = 0xd0;
 71              stat = FullStatusCheck(from,'0');
 72              *from = 0xff;                       // Read
 73              if (stat == NG) return(NG);
 74          }
 75          return(OK);
 76      }
 77      //********************************************************
 78      //      【FROM品種】
 79      //      FromClose()
 80      //      FlashRom書き込み処理が全て終了した場合に必要な手続き
 81      //      不要な場合、プログラム記述しなくても良い
 82      //      終了時に1回だけ呼ばれる
 83      //********************************************************
 84      int FromClose(void)
 85      {
 86          return(OK);
 87      }
 88      //********************************************************
 89      //      【FROM品種】
 90      //      FromSectorEraseProc(char *cmd)
 91      //      伝送フォーム(in)
 92      //        cmd[0]    == %
 93      //            [1]   == 0
 94      //            [2]   == CommandSize[2]以降[Sum]も含む
 95      //            [3]   == Command('D')
 96      //            [4]   == Sector(0->255)
 97      //            [Sum] == [3]->[Sum-1]の加算
 98      //      return(out)
 99      //        0   -- 正常終了
```

関数「FromOpen()」は、ターゲット RAM に、このプログラムがダウンロードされた時、最初に１回コール(Call)されます。
このプログラム例は、全セクターをロック解除（アンロック）しています。必要無い場合は、「return(Ok);」のみ記述します。

関数「FromClose()」は、フラッシュ ROM に全データ書き込み終了後の最後に 1 回コール(Call)されます。ソフトウェアロックが必要な場合は、この関数にプログラムして下さい。必要無い場合は、「return(Ok);」のみ記述します。

3）「FromTemp.c」に追加フラッシュＲＯＭの「セクターイレーズ」関数を作成します。
[2-7]

```
行番… S… ソース
 98      }
 99      //********************************************************
100      //      FromSectorEraseProc(char *cmd)
101      //      伝送フォーム(in)
102      //        cmd[0]    == %
103      //            [1]   == 0
104      //            [2]   == CommandSize[2]以降[Sum]も含む
105      //            [3]   == Command('D')
106      //            [4]   == Sector(H) MAX(1024)
107      //            [5]   == Sector(L)
108      //            [Sum] == [3]->[Sum-1]の加算
109      //      return()
110      //        OK  == 正常終了
111      //        NG  == 異常終了
112      //********************************************************
113      int  FromSectorEraseProc(char *cmd)
114      {
115          Ushort  *from;                          // Word Access
116          short   sector;
117          int     stat;
118
119          sector  = (cmd[4] << 8) & 0xff00;
120          sector |= cmd[5] & 0xff;
121          if (sector < SectorMax) {
122              from = (Ushort *)SectorTopAdr[sector];
123              *from = 0x20;
124              *from = 0xd0;
125              stat = FullStatusCheck(from,'D');
126              *from = 0xff;                       // Read
127              if (stat == OK) return(OK);
128          }
129          return(NG);
130      }
131      //********************************************************
```

関数「FromSectorEraseProc(char *cmd)」には、セクターごとのイレーズプログラムを記述します。デバッガとの通信完了後、サムチェック正常の場合、この関数がコール(Call)されます。（サム異常時はコールされません）

＜引数の仕様＞
char　cmd[];
cmd[0]　==　'%'　//　ヘッダー
cmd[1]　==　0　//　予備
cmd[2]　==　4　//　コマンドサイズ　cmd[3]->[Sum]までのサイズ
cmd[3]　==　'D'　//　コマンド
cmd[4]　==　0xn　//　消去するセクター番号の H 数 最大 HIGH(1024)
cmd[5]　==　0xn　//　消去するセクター番号の L 数 最大 LOW(1024)
cmd[6]　==　0xn　//　サム　cmd[3]->[5]の加算値

＜備考＞
全イレーズ処理にしたい場合は、セクター番号ゼロ「０」のときのみ実施して、他ののセクターの場合は、「return(Ok);」にすれば代用できます。

4）「FromTemp.c」に追加フラッシュＲＯＭの「バイト／ワードプログラム（書き込み）」関数を作成します。
[2-8]

```
行番..  S..  ソース
 131     //*********************************************************
 132     //      FromWriteProc(char *cmd)
 133     //      伝送フォーム(in)
 134     //        cmd[0]      == %
 135     //            [1]     == 0
 136     //            [2]     == CommandSize[2]以降[Sum]も含む
 137     //            [3]     == Command('W')
 138     //            [4]     == WriteAddress(HH)
 139     //            [5]     == WriteAddress(MH)
 140     //            [6]     == WriteAddress(ML)
 141     //            [7]     == WriteAddress(LL)
 142     //            [8+0]   == WriteData[0]
 143     //             .      ==   "
 144     //             .      ==   "
 145     //            [8+127] == WriteData[127] 固定 余分エリア(
 146     //            [Sum] == [3]->[Sum-1]の加算
 147     //      return()
 148     //        OK  == 正常終了
 149     //        NG  == 異常終了
 150     //*********************************************
 151     int  FromWriteProc(char *cmd)
 152     {
 153         Ushort  *from;                          // Word Access
 154         Ushort  *verf;                          // Word Access
 155         Ushort  *data;
 156         int     i;
 157         int     stat;
 158
 159         from = (Ushort *)(                      // Address set
 160                (((long)cmd[4] << 24) & 0xff000000) |
 161                (((long)cmd[5] << 16) & 0x00ff0000) |
 162                (((long)cmd[6] << 8)  & 0x0000ff00) |
 163                 ((long)cmd[7]        & 0x000000ff)
 164                );
 165         verf = from;                           // Verify
 166         data = (Ushort *)&cmd[8];              // Data Adr set
 167
 168         for(i = 0;i < BLOCK/2;i++){            // Word Write
 169             *from = 0x40;
 170             *from = data[i];
 171             stat = FullStatusCheck(from,'W');
 172             if (stat == NG) return(NG);
 173             ++from;
 174         }
 175         verf[0] = 0xff;                        // Read
 176         for(i = 0;i < BLOCK/2;i++){            // VerifyTest
 177             if (verf[i] != data[i]) {          // VerifyError
 178                 FromErrorSet((long)'R',(long)&verf[i],(long)data[i]);
 179                 return(NG);
 180             }
 181         }
 182         return(OK);
 183     }
 184     //*********************************************************
```

関数「FromWriteProc(char *cmd)」には、128 バイト(64 ワード)ごとの書き込みプログラムを記述します。デバッガから書き込みデータ 128 バイト(64 ワード)固定でデータの受信完了後、サムチェック正常の場合、この関数がコール(Call)されます。（サム異常時はコールされません）

＜引数の仕様＞

```
char   cmd[];
cmd[0]  ==  '%'   //  ヘッダー
cmd[1]  ==  0     //  予備
cmd[2]  ==  134   //  コマンドサイズ  cmd[3]->[Sum]までのサイズ
cmd[3]  ==  'W'   //  コマンド
cmd[4]  ==  0xn   //  書き込みアドレス(HH)
cmd[5]  ==  0xn   //  書き込みアドレス(MH)
cmd[6]  ==  0xn   //  書き込みアドレス(ML)
cmd[7]  ==  0xn   //  書き込みアドレス(LL)
cmd[8]  ==  0xn   //  書き込みデータ[0]
"
"
cmd[135]==  0xn   //  書き込みデータ[127]  固定  余分エリアは(0xff)
cmd[136]==  0xn   //  サム  cmd[3]->[135]の加算値
```

＜備考＞
このプログラム例は、128 バイトの書き込み終了後、ベリファイテストも実施しています。

5）「FromTemp.c」に追加フラッシュＲＯＭの「フルステータスチェック」関数を作成します。
[2-9]

```
行番..  S..  ソース
 184     //*********************************************************
 185     //      FullStatusCheck()                       Word Access
 186     //      フルステータスをチェックエラー発生時は、エラー情報
 187     //      を[FromError]に残す
 188     //      FromErro[0] == command 'W'or'D'or'E' or 'O'==Open
 189     //              [1] == アクセスしたアドレス
 190     //              [2] == FROM ステータス
 191     //      return()
 192     //        OK  == 正常終了
 193     //        NG  == 異常終了
 194     //*********************************************************
 195     int     FullStatusCheck(Ushort *adr,char cmd)
 196     {
 197         FromErrorSet(0,0,0);                    // Error情報を消去
 198         while((*adr & SR7) == 0){               // Wait Ready
 199             if(*adr & (SR1 | SR3 | SR4 | SR5)){ // Error
 200                 FromErrorSet((long)cmd,(long)adr,(long)*adr);
 201                 return(NG);
 202             }
 203         }
 204         return(OK);
 205     }
 206     //*********************************************************
```

関数「FullStatusCheck(Ushort *adr,char cmd」には、フラッシュ ROM 特有のステータスチェックプログラムを記述します。この関数は、上記の作成関数からのみコール(Call)されます。よって、内部処理は自由に記述して下さい。

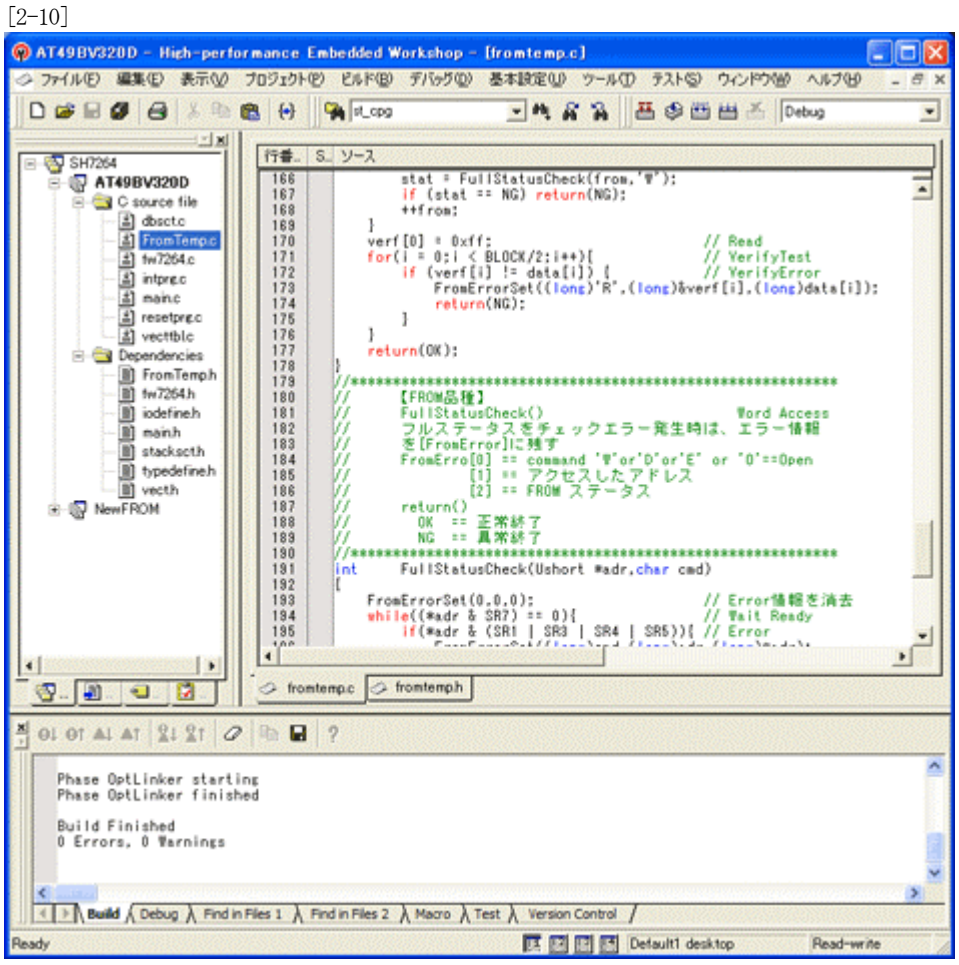
＜引数の仕様＞

```
Ushort   *adr;    // 処理しているフラッシュ ROM のアドレス
char     cmd;     // 処理しているコマンド
                  // "O' = オープンコマンド
                  // "C' = クローズコマンド
                  // "D' = イレーズコマンド
                  // "W' = 書き込みコマンド
                  // "R' = ベリファイコマンド
```

＜備考＞
関数「FromErrorSet(..)」は、関数デバッグ時のエラー情報を見るための一例としての関数です。仕様に関してはルールがありませんので、自由に作成して下さい。

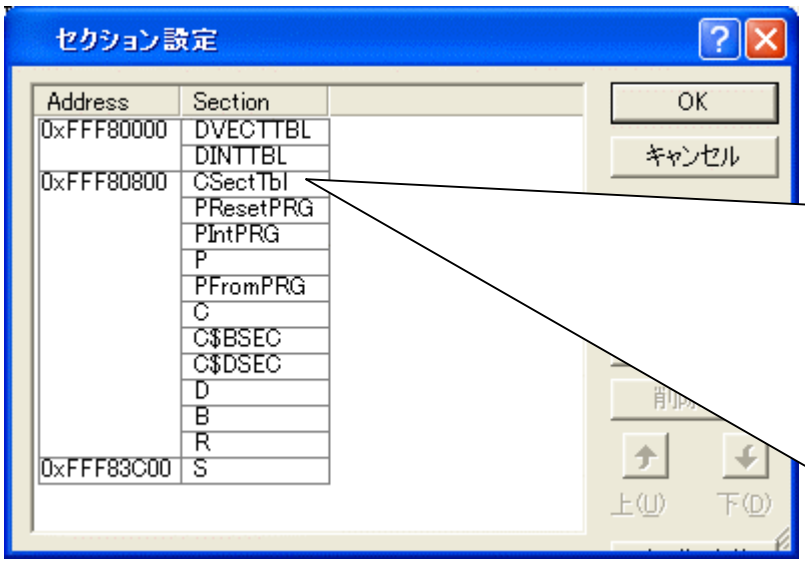
以上で、関数の作成は終了です。

6）新規プロジェクトのＦｌａｓｈROMプログラムをビルドします。
[2-10]



Ｈｅｗメニュー
＜ビルド＞－＜すべてをビルド＞
で、「0 Errors 0 Warnings」になったことを確認します。

7）新規プロジェクトのＦｌａｓｈROMプログラム作成上のルール
[2-11]



＜ルール１＞
セクター情報定義のセクション名「SectTbl」の、ロケートは必ず、実行プログラムアドレス(TopAdr)の「+0x800」に割付て下さい。

＜ルール２＞
セクション「SectTbl」のデータ配置にルールがあります。

```
const       long    SectorMax;      // セクター数
const       long    SectorTopAdr[SectorMax+1];// セクター毎 Top アドレス
```

この順番に配置されることが条件になります。

＜ルール３＞
最大セクター(ブロック)数は、「1024」です。

＜ルール４＞
スタックまで含めたオブジェクトサイズは「0x4000」16KB までです。

＜ルール５＞

```
#define OK      0               // 変更不可
#define NG      0xee            // 変更不可
```

数値変更しないで下さい。

＜ルール６＞
下記４関数は、名称・引数・帰り値等の仕様変更は不可です。

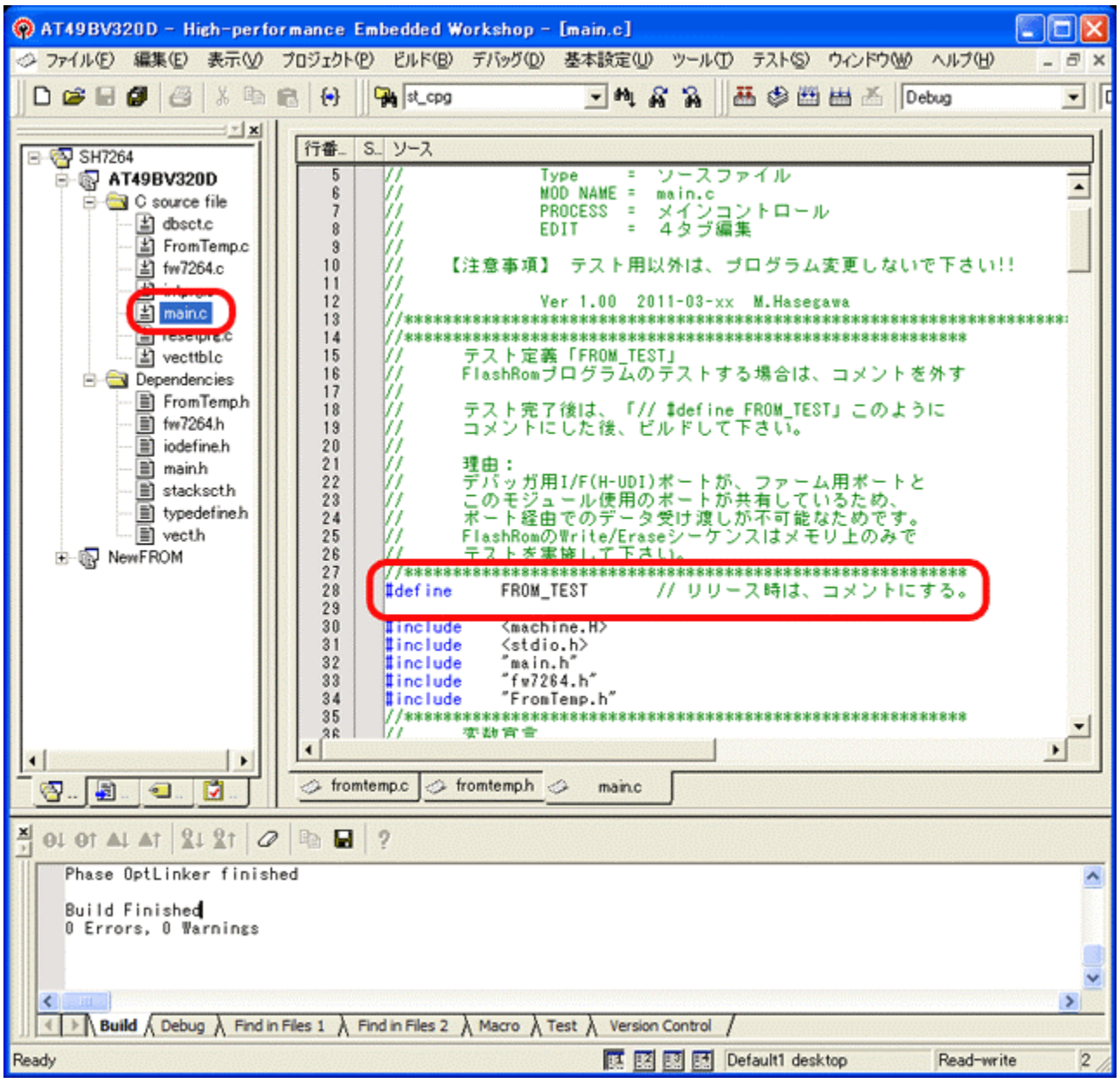
```
1)int       FromOpen(void);
2)int       FromClose(void);
3)int       FromSectorEraseProc(char *cmd);
4)int       FromWriteProc(char *cmd);
```

以上のルールは厳守して下さい。

【作成したＦｌａｓｈROMプログラムのデバッグ方法】

1．Ｈｅｗにてデバッグの準備をします。

[3-1]



1）モジュール「main.c」の「#define FROM_TEST」のコメントを外します。

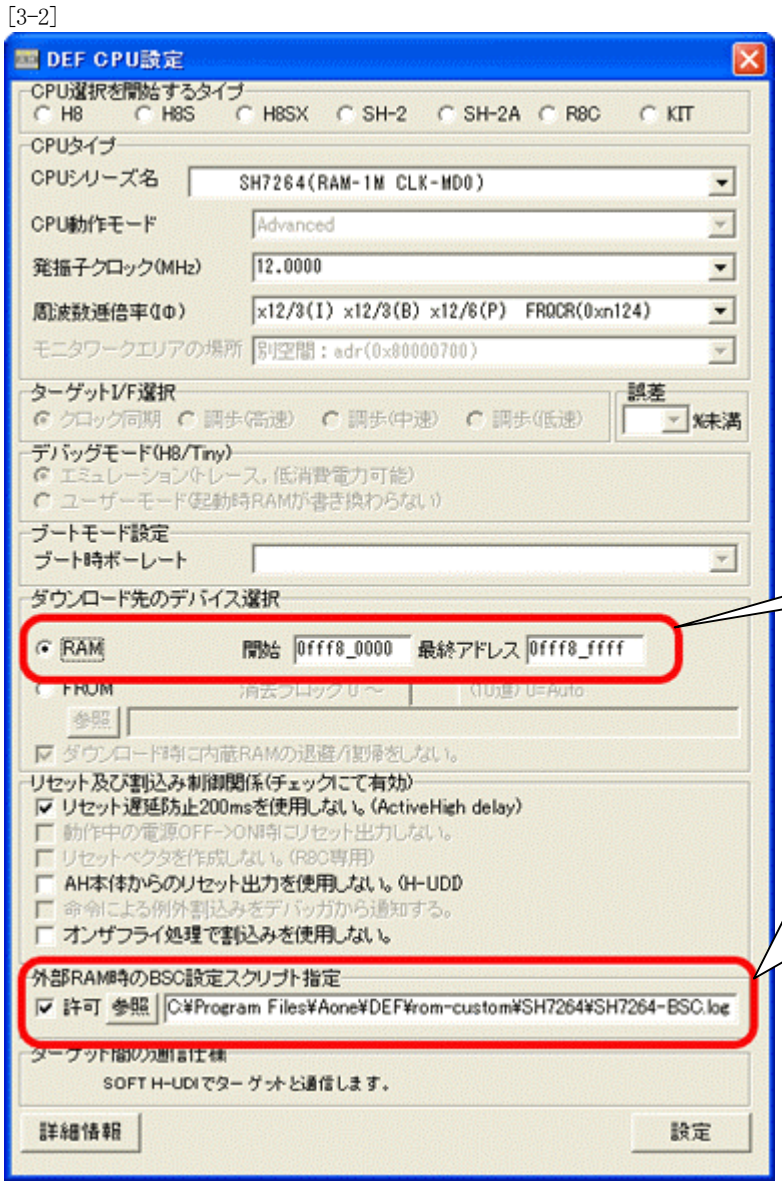
2）Ｈｅｗメニューの＜ビルド＞ー＜ビルド＞を指示します。

3）「0 Errors 0 Warning」を確認します。

4）デバッガ使用の通信ポートと、このＦｌａｓｈROMプログラム使用の通信ポートと兼用させているため、通常のデバッグ操作でのデバッグは出来ません。
よって、上記「#define FROM_TEST」のコメントを外すことにより、通信ポートを使用せず、内部メモリ操作のみで各関数のテストできるようにテスト用プログラムを用意しました。

２．デバッガ用コントロールソフト「ＤＥＦ」にてデバッグする為の設定をします。

[3-2]



＜ＣＰＵ設定＞

作成した「FlashROM」プログラムをターゲット側の内部ＲＡＭに転送して実行させますので、「ＲＡＭ」にチェック後、先頭と最終アドレスを指定します。

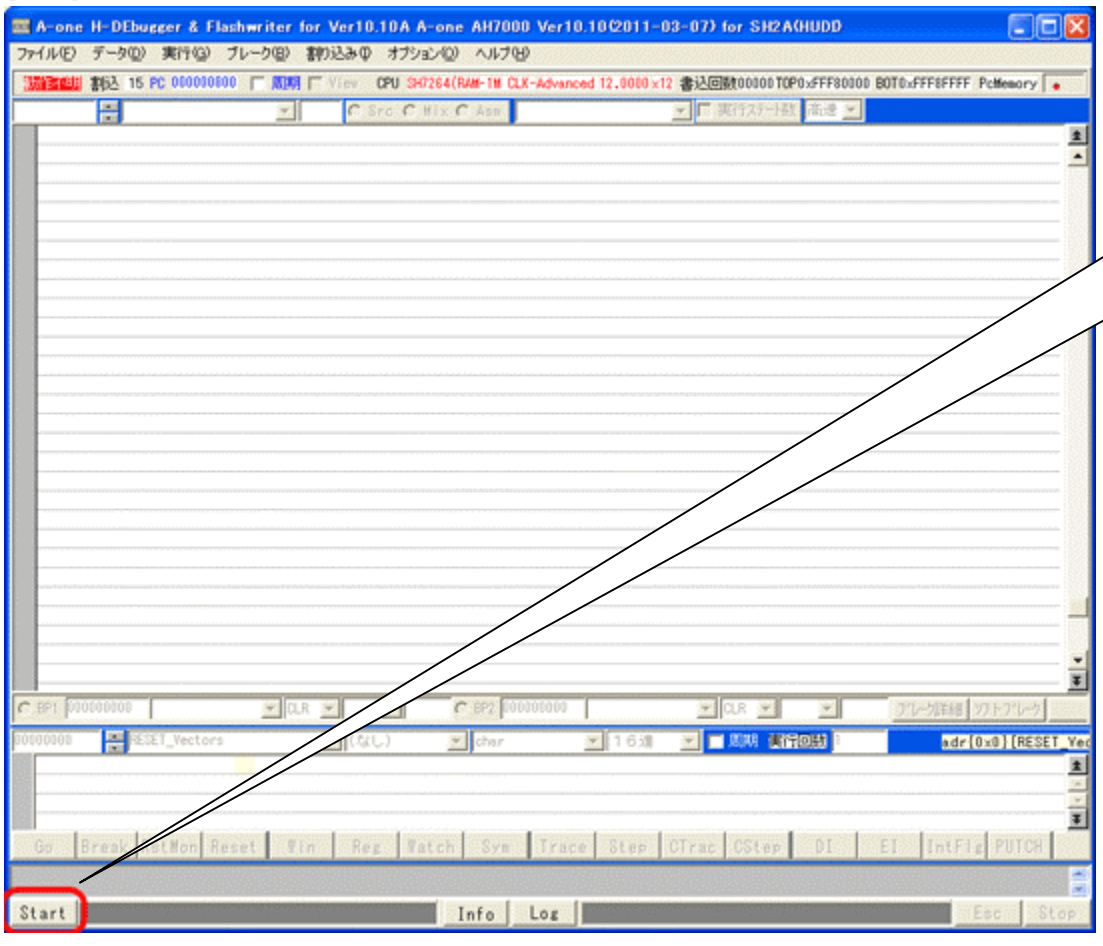
ＳＨ７２６４の場合
開始　0xfff8_0000　　最終　0xfff8_ffff
になります。

作成した「FlashROM」プログラムのデバッグにＢＳＣ設定が必要な場合は、スクリプトファイルを指定します。

ＳＨ７２６４の場合
インストールディレクトリ
"c:¥Program Files¥Aone¥DEF¥rom-custom¥SH7264"
に、例として「SH7264-BSC.log」が用意してありますので、目的ハードにカスタマイズして下さい。

＜ー「設定」をクリックします。

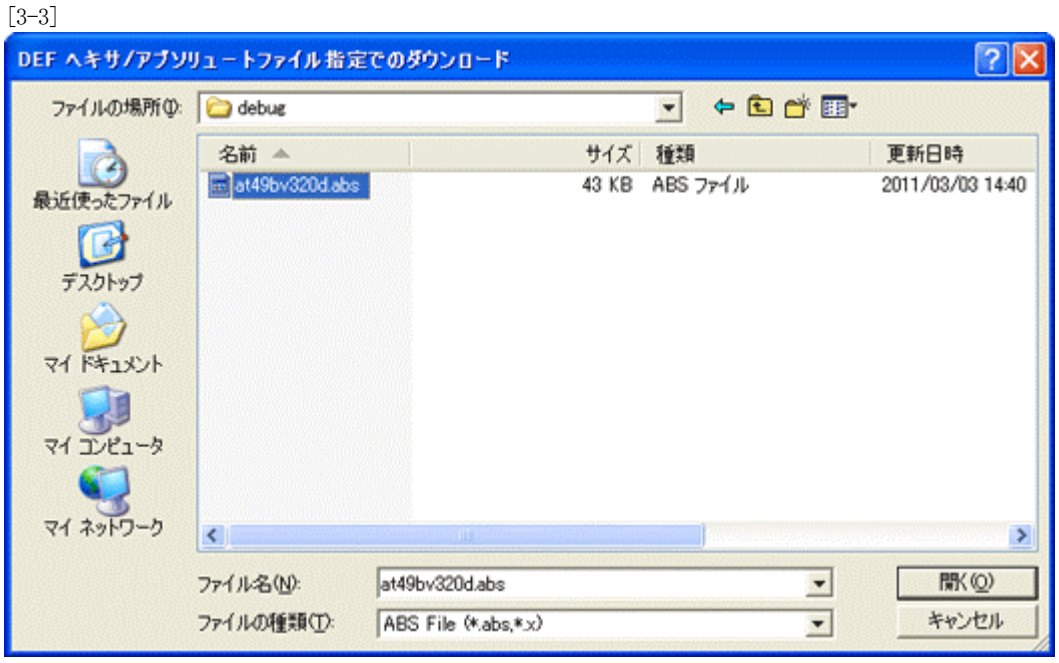
[3-2-1]



ＤＥＦ画面、左下隅の「Start」をクリックします。

3．作成したＦｌａｓｈROMソフトのデバッグを開始する準備をする。

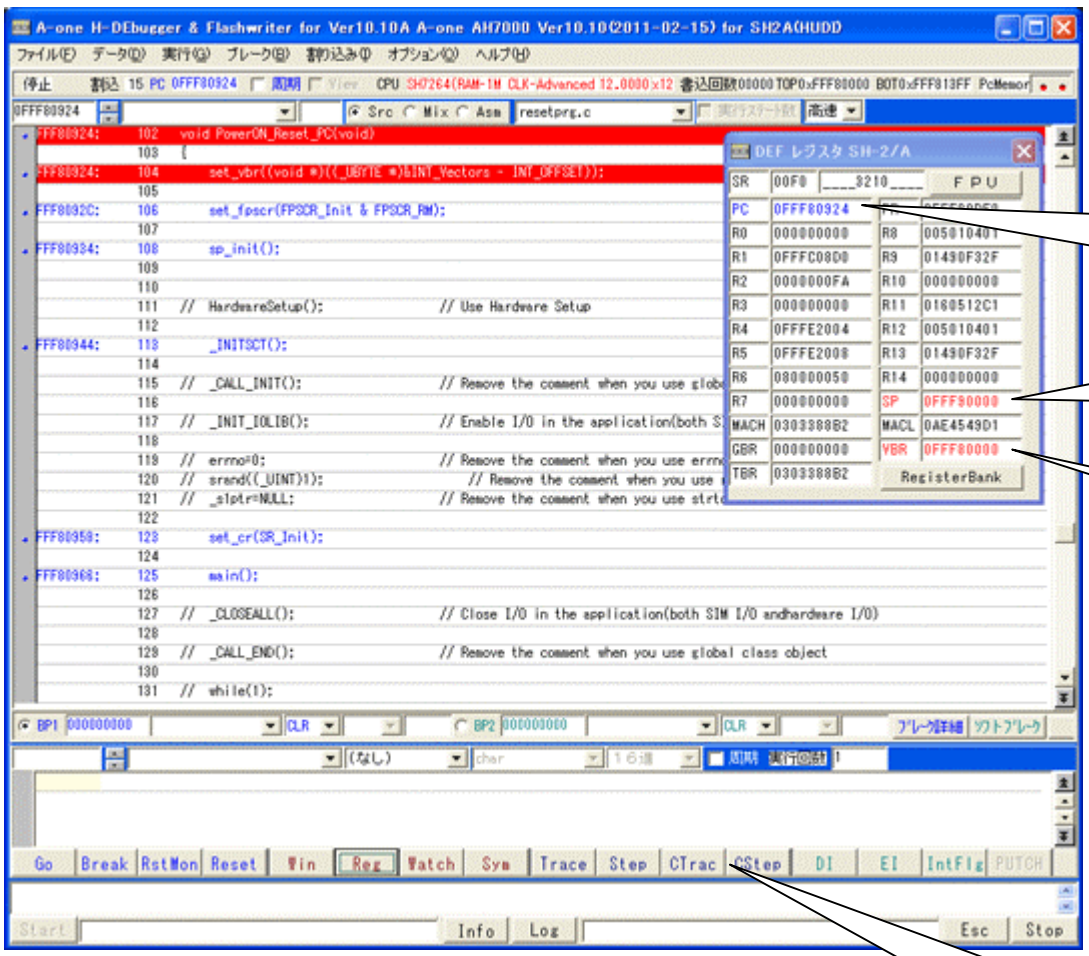
[3-3]



ＤＥＦメニュー
＜ファイル＞－＜ダウンロード＞
で、ダウンロードします。

インストールディレクトリ
"c:¥Program Files¥Aone¥DEF¥rom-custom¥SH7264"
下の"ProjectName¥Debug"
に作成したアブソリュートファイルがありますので指定します。
(例　ProjectName:AT49BV320D)

[3-4]



＜ダウンロードが成功した初期画面＞

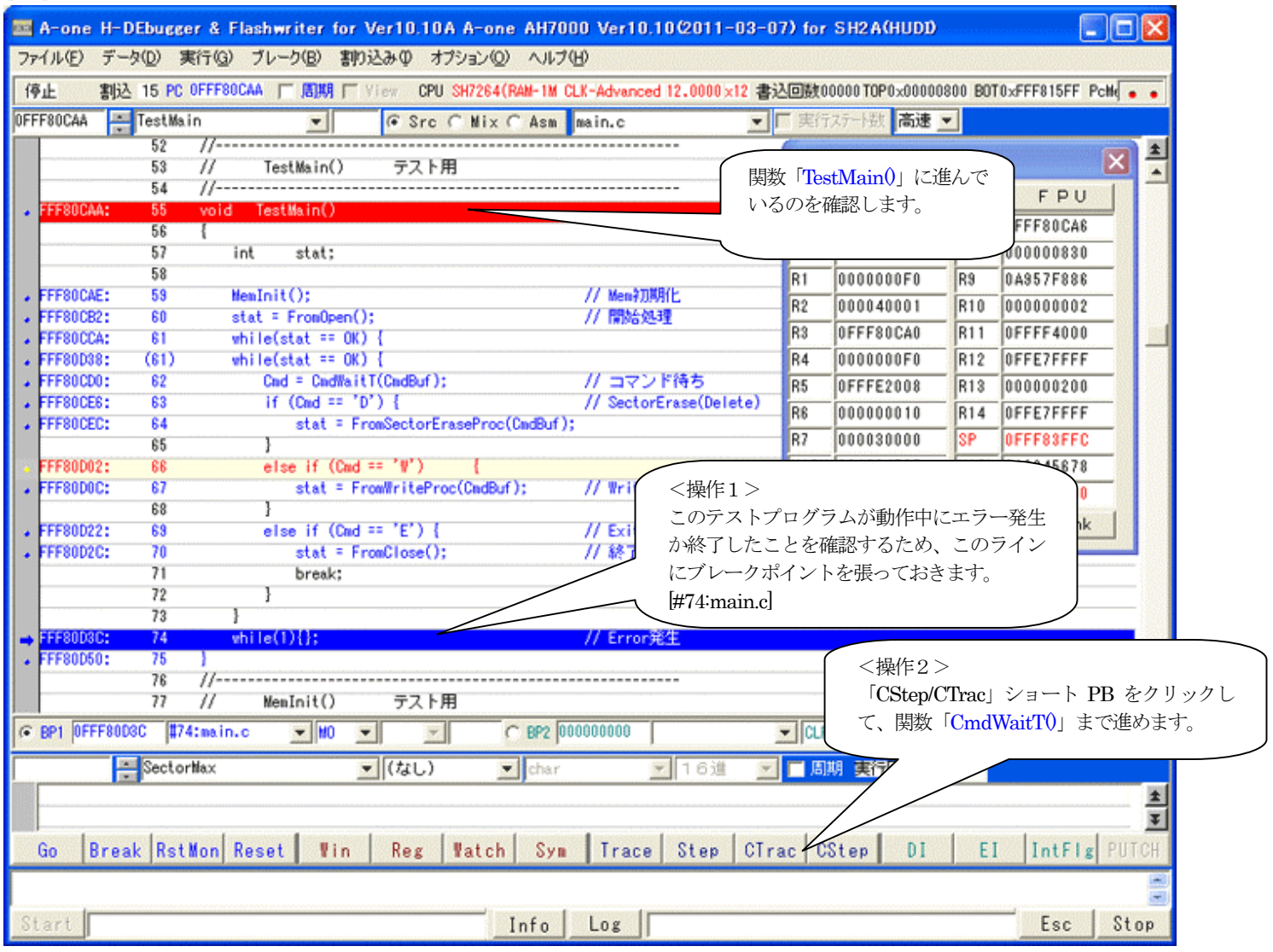
PC レジスタ値が
「PowerON_Reset_PC()」関数の
先頭アドレスになっていることが
確認出来ます。

SP レジスタ値が内蔵 RAM のボトムアドレスになっていることが確認できます。

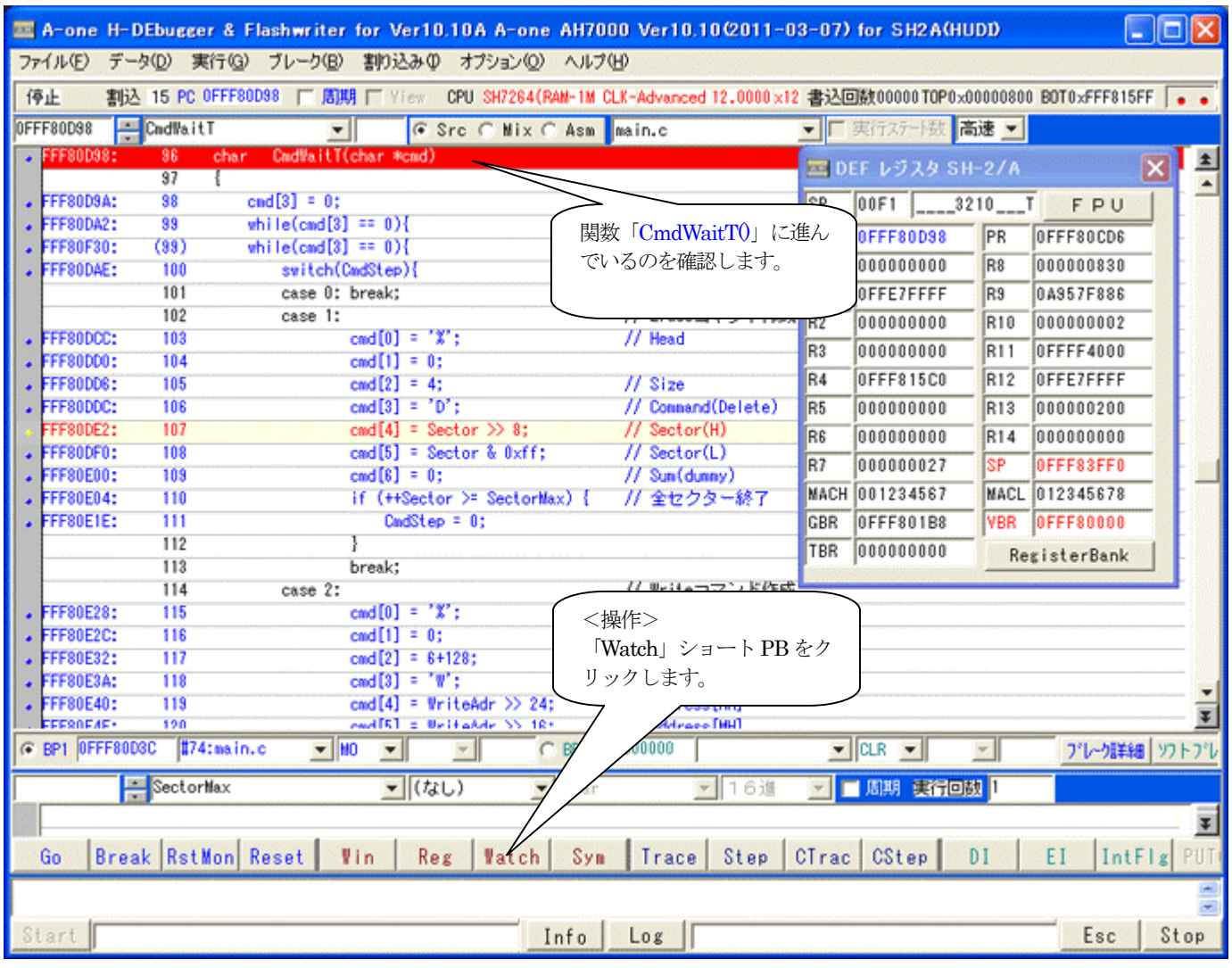
VBR レジスタ値が内蔵 RAM のトップアドレスになっていることが確認できます。

＜操作＞
「CStep/CTrac」ショート PB をクリックして、
関数「TestMain()」まで進めます。

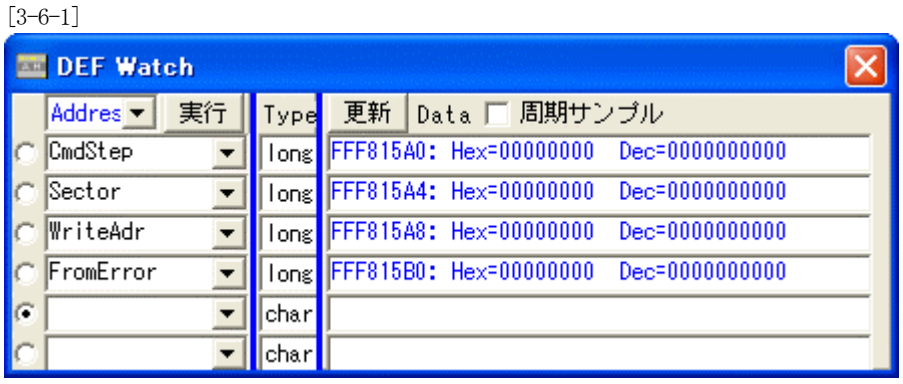
[3-5]


A-one H-DEbugger & Flashwriter for Ver10.10A A-one AH7000 Ver10.10(2011-03-07) for SH2A(HUDD)
関数「TestMain()」に進んでいるのを確認します。
＜操作１＞
このテストプログラムが動作中にエラー発生か終了したことを確認するため、このラインにブレークポイントを張っておきます。
[#74:main.c]
＜操作２＞
「CStep/CTrac」ショート PB をクリックして、関数「CmdWaitT()」まで進めます。
Go
Break
RstMon
Reset
Win
Reg
Watch
Sym
Trace
Step
CTrac
CStep
DI
EI
IntFlg
Start
Info
Log
Esc
Stop

[3-6]


A-one H-DEbugger & Flashwriter for Ver10.10A A-one AH7000 Ver10.10(2011-03-07) for SH2A(HUDD)
関数「CmdWaitT()」に進んでいるのを確認します。
＜操作＞
「Watch」ショート PB をクリックします。
Go
Break
RstMon
Reset
Win
Reg
Watch
Sym
Trace
Step
CTrac
CStep
DI
EI
IntFlg
Start
Info
Log
Esc
Stop

[3-6-1]



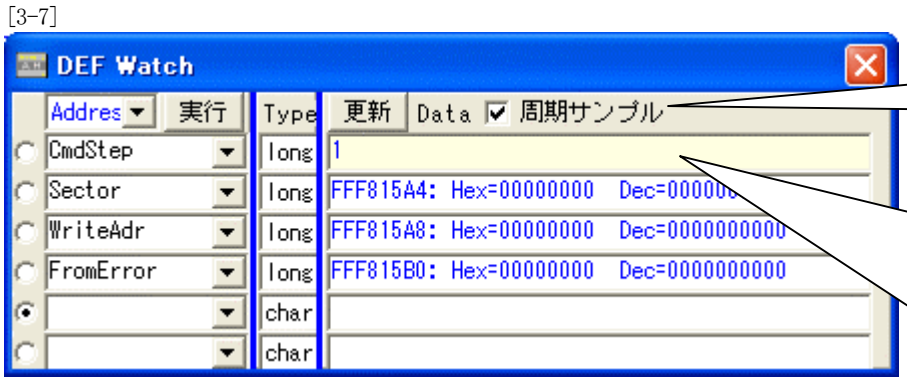
デバッグを進めるため、Watch 画面に変数を登録します。

1) CmdStep	関数「CmdWaitT」のコントロールステップ変数
2) Sector	セクターイレーズの進行カウンター
3) WriteAdr	FROM 書き込み中アドレス
4)FromError	エラー発生時情報

以上、４変数を登録します。

4．セクターイレーズ関数「FromSectorEraseProc(char *cmd)」をデバッグします。

[3-7]

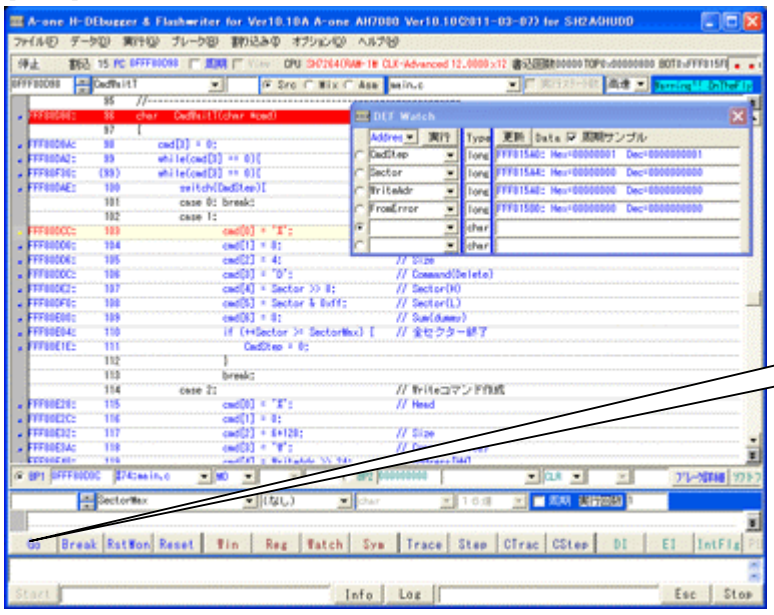


＜操作１＞
プログラム進行を確認するため、「周期サンプル」をチェックします。

＜操作２＞
変数「CmdStep」を数値「1」にします。
[Enter][1][Enter]で変更できます。

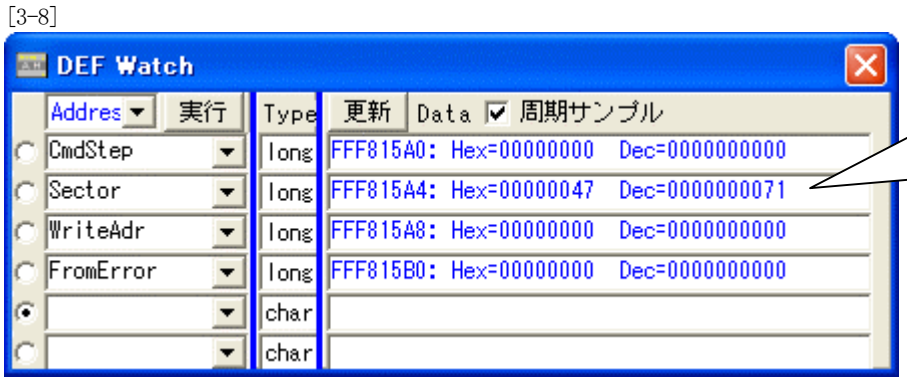
＜備考＞
関数「CmdWaitT」のソースを見てもらえれば理解できるかと思いますが、「CmdStep」を「1」にしますと、メモリー上でコマンドデータを作成して関数「FromSectorEraseProc(char *cmd)」に渡す仕組みになっています。
全て正常終了しますと、登録された全セクターをイレーズした後、変数「CmdStep」が数値「0」になります。

[3-7-1]



＜操作３＞
「Go」ショート PB をクリックします。

[3-8]



セクターイレーズが正常に動作していますと、変数「Sector」が、ゼロ「0」から登録セクター数までインクリメントしていきます。
最終セクターまで正常終了しますと、変数「CmdStep」がゼロ「0」になります。

異常終了しますと、[3-5]で設定したブレークポイントで停止します。

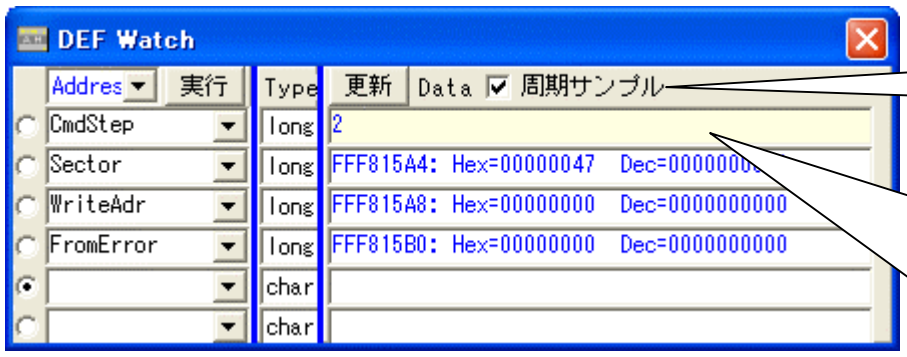
[3-9]

＜ダンプ画面＞

イレーズされているかダンプ画面で確認します。

5．書き込み関数「FromWriteProc(char *cmd)」をデバッグします。

[3-10]



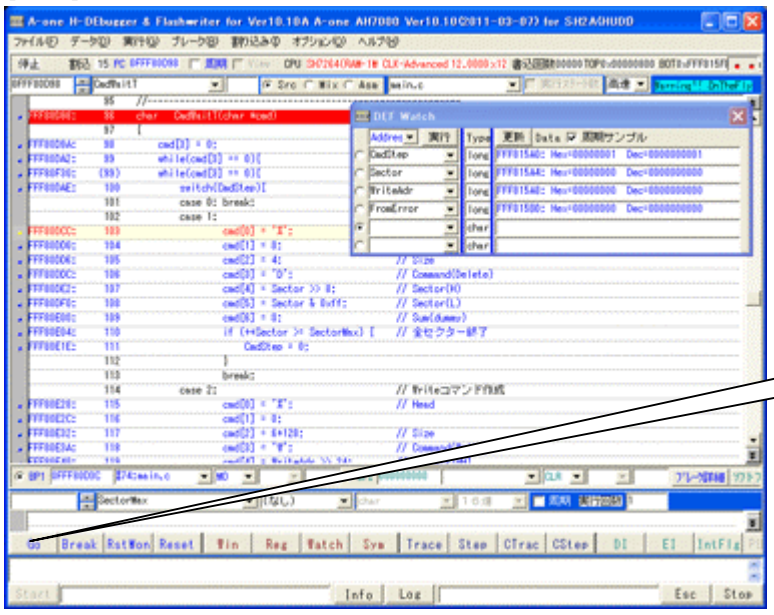
＜操作１＞
プログラム進行を確認するため、「周期サンプル」をチェックします。

＜操作２＞
変数「CmdStep」を数値「2」にします。
[Enter][2][Enter]で変更できます。

＜備考＞
関数「CmdWaitT」のソースを見てもらえれば理解できるかと思いますが、「CmdStep」を「2」にしますと、メモリー上でコマンドデータを作成して関数「FromWriteProc(char *cmd)」に渡す仕組みになっています。

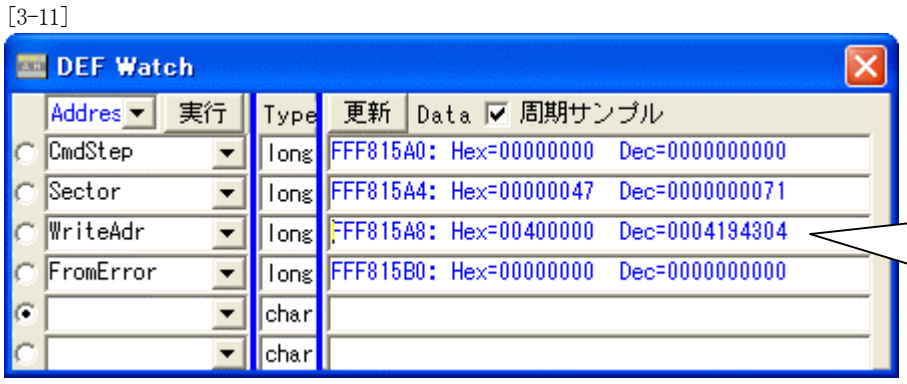
全て正常終了しますと、全エリア書き込み後、変数「CmdStep」が数値「0」になります。

[3-7-1]



＜操作３＞
「Go」ショート PB をクリックします。

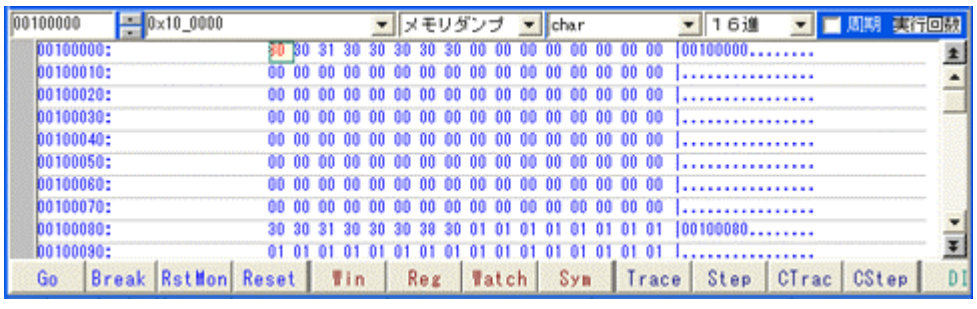
[3-11]



書き込みが正常に動作していますと、変数「WriteAdr」が、ゼロ「0」から最終アドレスまで「0x80」ごとに加算していきます。
最終アドレスまで正常終了しますと、変数「CmdStep」がゼロ「0」になります。

異常終了しますと、[3-5]で設定したブレークポイントで停止します。

[3-12]



＜ダンプ画面＞

正しくデータが書き込まれたかダンプ画面で確認します。

ゼロ番地以外は、0x80 番地ごとに先頭のアドレス位置を ASCII 文字データを書き、残りは、ゼロから始まるパターンを+1 ごとに埋めています。なお、ゼロ番地から 8 バイト分は、PC/SP レジスタの初期データになりますので、誤動作防止のため[0xff]にしています。

5．その他関数「FromOpen(void)/FromClose(void)」が必要な場合は、必要に応じてデバッグして下さい。

1）関数「FromOpen(void)」

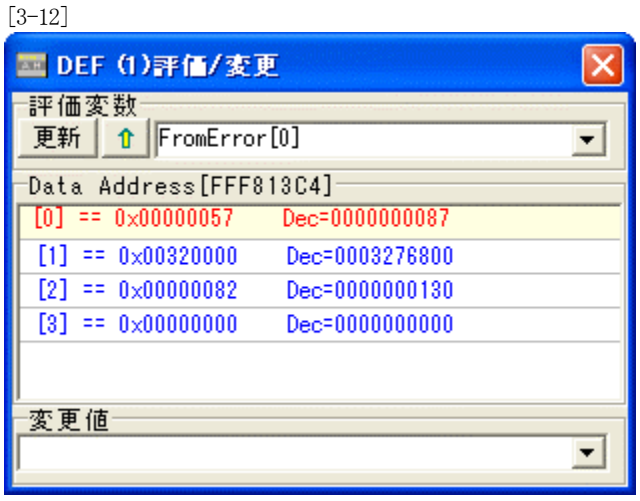
テスト用メイン関数「TestMain()」の 60 行でコール(Call)しています。

2）関数「FromClose(void)」

テスト用関数「CmdWaitT(char *cmd)」で、コントロールステップ変数「CmdStep」を「3」以上の数値をセットしますとコール(Call)されます。

6．異常終了時のエラー情報-「long　FromError[4]」の参照

[3-12]

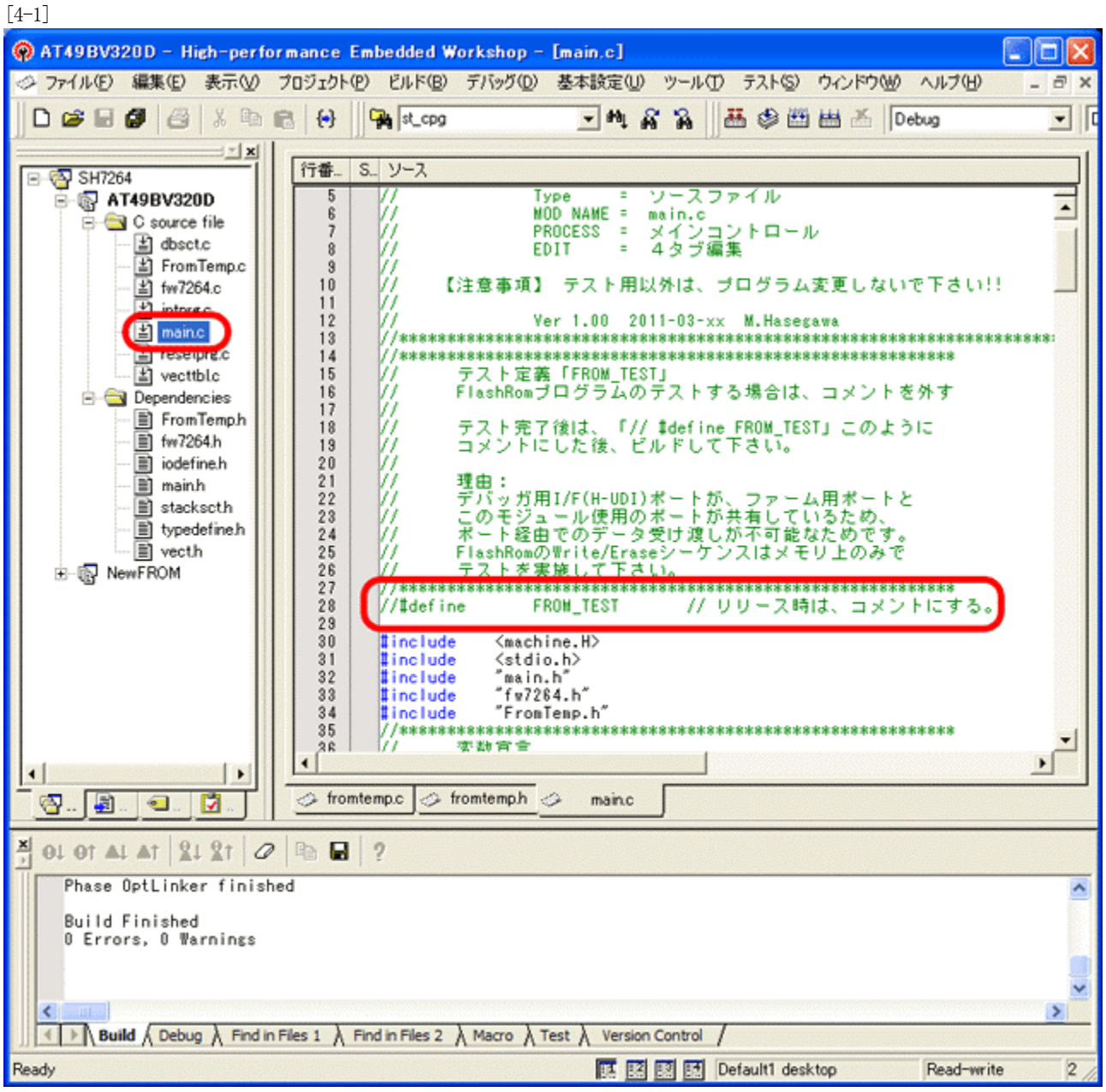


<評価・変更窓>

エラー発生にて異常終了した場合は、変数「FromError[4]」に情報が残りますので、プログラム修正の参考にして下さい。なお、この変数の仕様および名称等に規約はありませんので、自由に変更しても構いません。

【作成したＦｌａｓｈROMプログラムを正規リリース登録する】

1．ＦｌａｓｈROMプログラムのデバッグが終了しましたら、Ｈｅｗにて再コンパイルします。
[4-1]



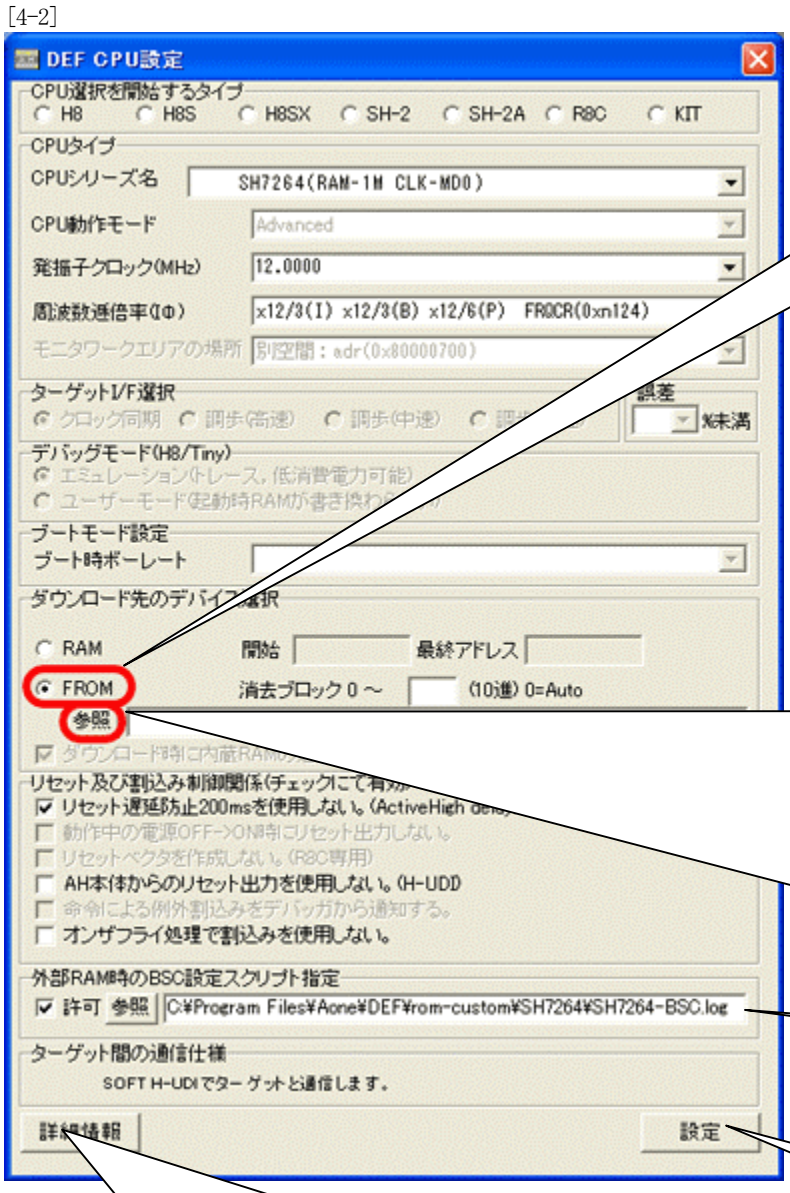
1）モジュール「main.c」の「// #define FROM_TEST」をコメントします。

2）Ｈｅｗメニューの＜ビルド＞ー＜ビルド＞を指示します。

3）「0 Errors 0 Warning」を確認します。

２．デバッガ用コントロールソフト「ＤＥＦ」にＦｌａｓｈROMプログラムの登録をします。

[4-2]

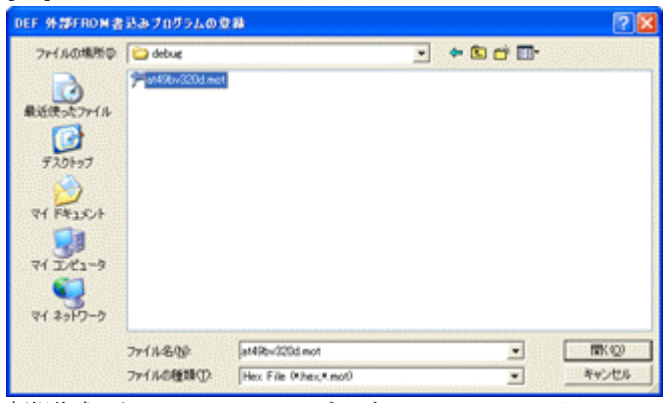


＜ＣＰＵ設定画面＞

＜操作１＞
FROM 側に「チェック」します。

＜操作２＞
「参照」ボタンをクリックします。

[4-3]



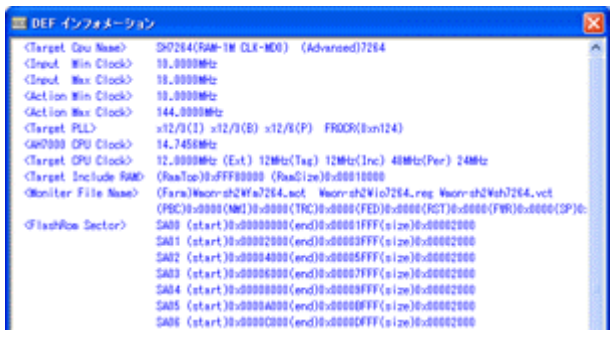
新規作成したＦｌａｓｈROMプログラムのＨＥＸファイル「Project.mot」を指定します。

⚠ ダウンロード時に使用するＲＡＭエリアの退避／復帰はできません。

⚠ FlashROM に Write する動作をさせるために BSC 設定が必要な CPU 品種の場合は、スクリプトファイルを登録して下さい。

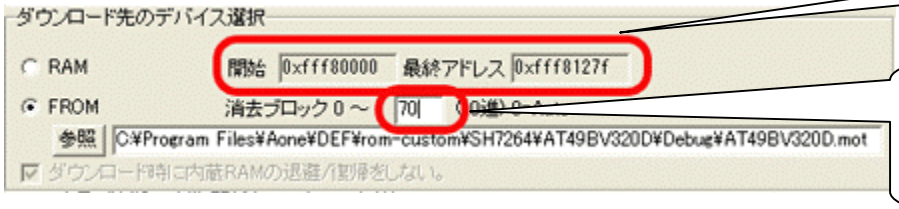
＜操作３＞
「詳細情報」ボタンをクリックしますと、＜インフォメーション画面＞に内部登録したセクターごとのトップアドレス情報を見ることができます。

[4-5-1]



＜操作４＞
最後に「設定」をクリックします。

[4-4-1]



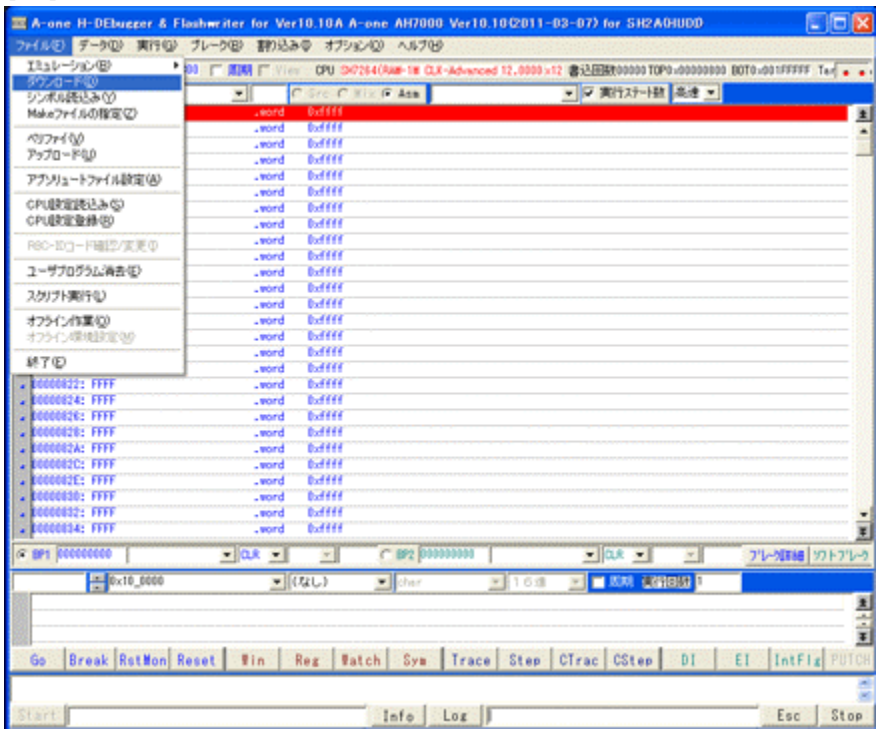
ＦｌａｓｈROMプログラムの先頭と最終アドレスを表示します。

ＦｌａｓｈROMプログラムに登録した最終セクター数を表示します。なお、登録時にここをゼロ「0」に設定しますと、ダウンロードするオブジェクトの大きさに応じたセクター数を自動で検出し、必要なセクターのみ消去する動作になります。(0=Auto)

【作成したＦｌａｓｈROMプログラムの最終確認をします】

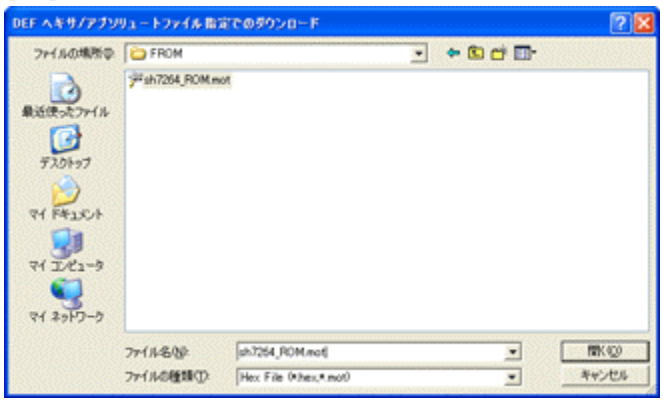
1．デバッガ用コントロールソフト「ＤＥＦ」を「Ｓｔａｒｔ」後、ＦｌａｓｈROMに、ダウンロードします。

[5-1]



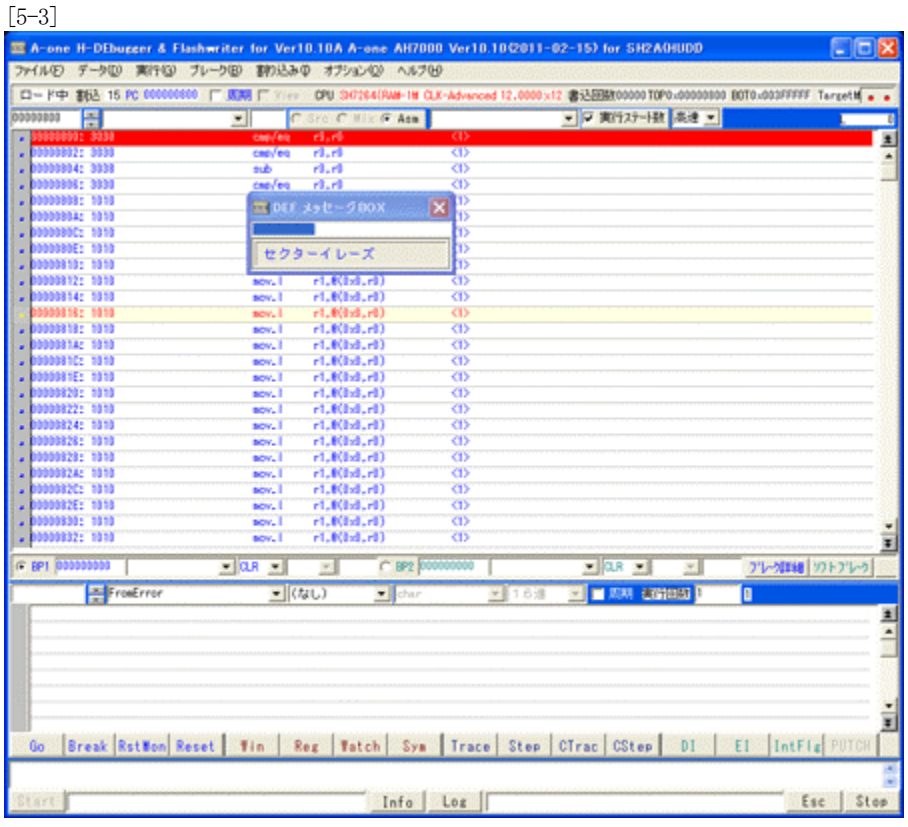
ＤＥＦメニュー
＜ファイル＞ー＜ダウンロード＞を指定します。

[5-2]



ＦｌａｓｈROM内にロケートされたプログラムを選択します。

[5-3]

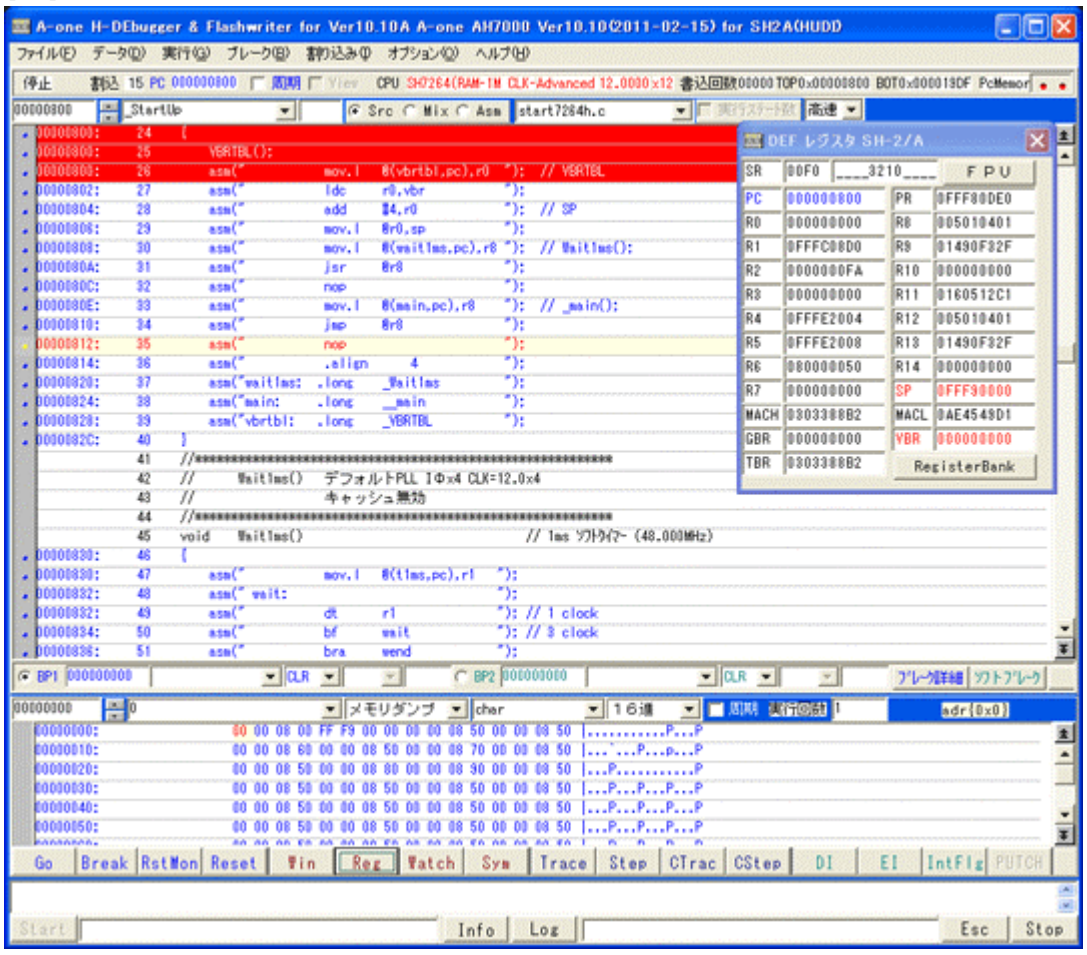


＜動作状況＞

1）【書き込みプログラムの転送】
2）【セクターイレーズ】
3）【ユーザーモードでダウンロード】

と、順次メッセージＢＯＸ表示され、処理が進行します。

[5-4]



＜最終確認画面＞

全て正常終了しますと、オブジェクトが外部Ｆｌａｓｈ ROMに登録され、このような初期画面になります。

⚠

２．御願い

本説の方法で、フラッシュROM品種を追加した場合、必ず、プロジェクトのバックアップすることを御願いします。

今回は「c:¥Program Files¥Aone¥DEF¥rom_custom」で作成する例で記述しましたが「¥rom_custom」をホルダごと別のディレクトリに貼り付けても作成できます。

つまり、ユーザーアプリのプロジェクトごとに管理するのも一案かもしれません。追加作成したプロジェクトは、ユーザー様の責任のもとで管理願います。

以上で、外付けＦｌａｓｈROMの品種追加作業が終了です。