シリアルフラッシュＲＯＭ対応のライタープログラムを追加する場合の説明（ブートモード１／３用）

Ｒｅｖ１．３０　２０１２／１０／４
ＤＥＦバージョン１１．００Ａより
ＤＥＦバージョン１２．１０Ａ変更
ＤＥＦバージョン１２．２０Ａ変更

【対象ＣＰＵ】

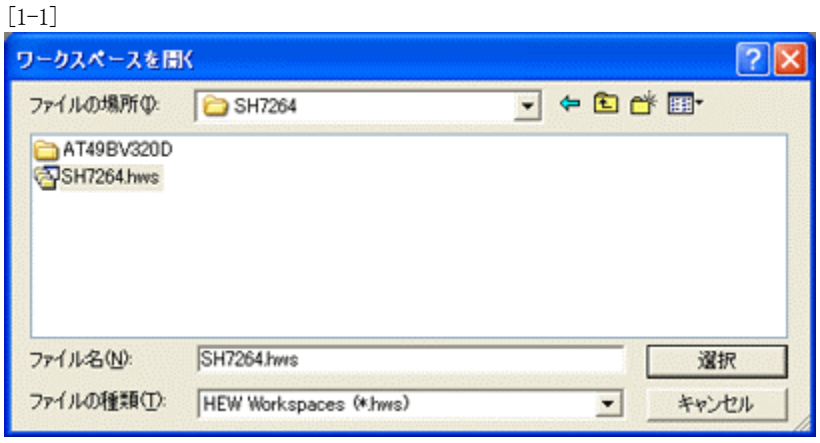
１．ＲＯＭレス品種が対象になります。（ＳＨ７２６２／４／６／７）

【機能】

１．サンプルで「ＡＴ２５ＤＦ０４１Ａ」の対応ソフトを用意してあります。（ルネサスＣのみ対応）【ＳＨ７２６２／４】
２．サンプルで「ＳＳＴ２５ＶＦ０１６Ｂ／Ｍ２Ｐ１６」の対応ソフトを用意してあります。（ルネサスＣのみ対応）【ＳＨ７２６６／７】
３．ＳＨ７２６２／４／６／７【ＲＳＰＩ０】に接続されたシリアルフラッシュＲＯＭが対象になります。
４．ＨｅｗにてＦｌａｓｈＲＯＭの品種追加が出来るよう対応する。
５．シリアル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。

【品種追加前の準備】

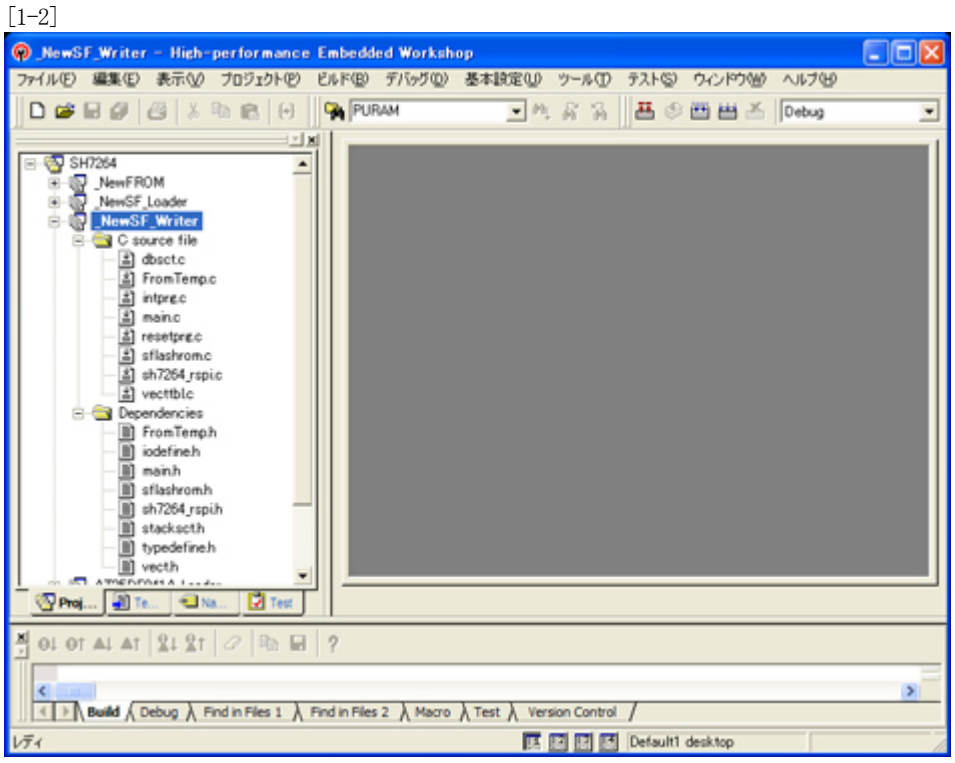
１．ＡＨ７０００コントロールソフトのインストールＤＩＲにあるワークスペースを開きます。（追加例）
［1-1］



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

＜ワークスペース＞
″SH7264.hws″

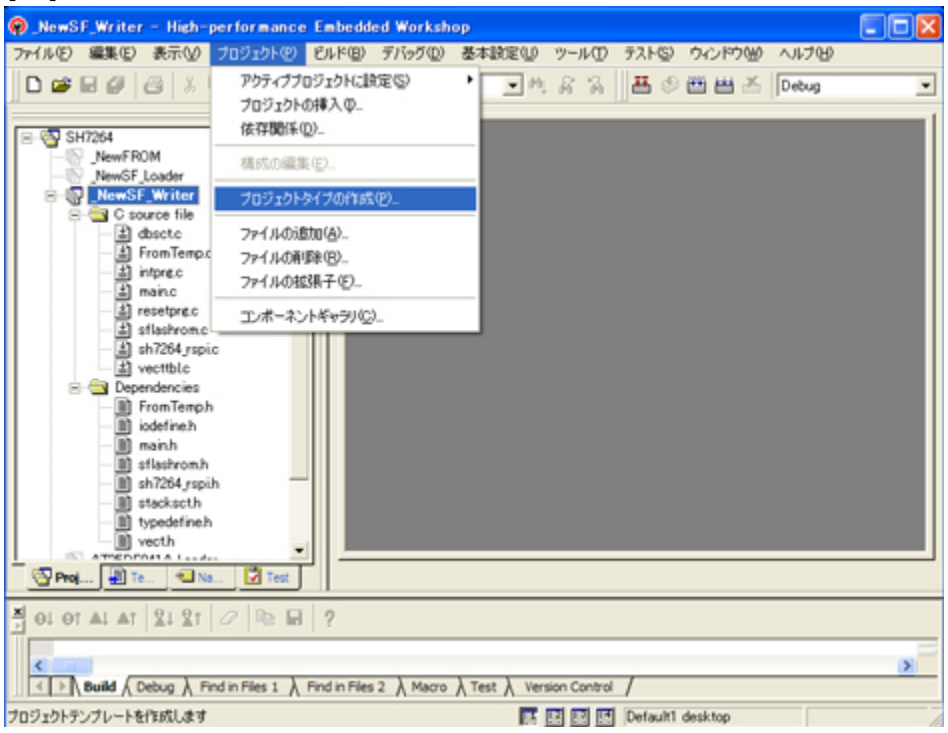
２．プロジェクト名「__ＮｅｗＳＦ__Ｗｒｉｔｅｒ」をアクティブプロジェクトに設定します。
［1-2］



＜__ＮｅｗＳＦ__Ｗｒｉｔｅｒ＞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。

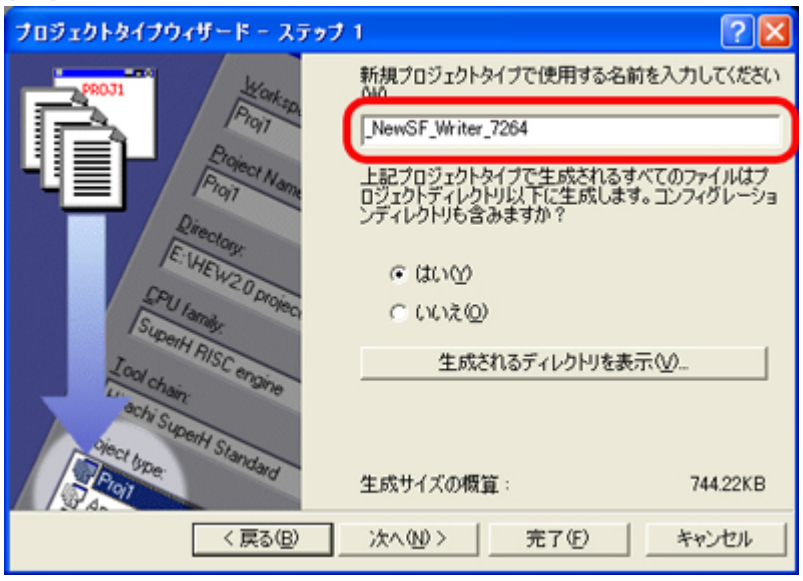
[1-3]



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。

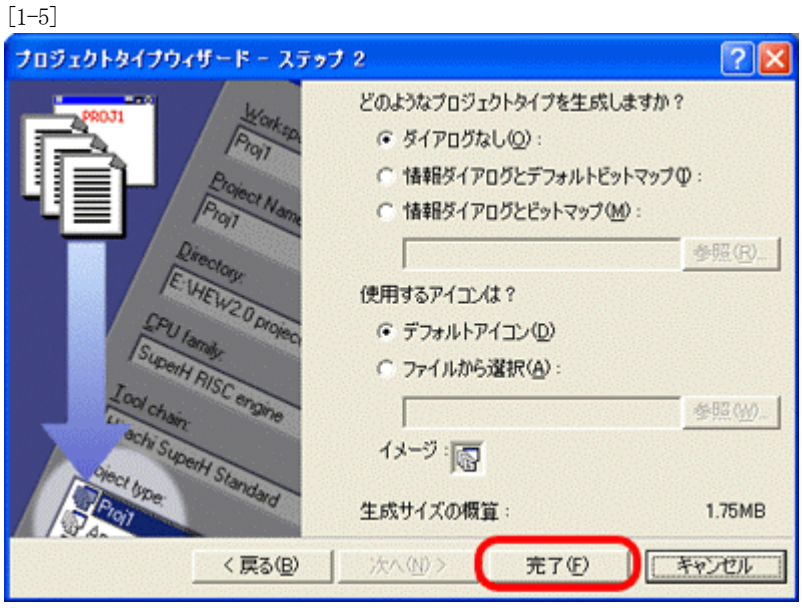
[1-4]



名前は、重複しないようにして下さい。
例として「ＮｅｗＳＦ＿Ｗｒｉｔｅｒ＿７２６４」としておきます。

＜ー「次へ」をクリックします。

[1-5]



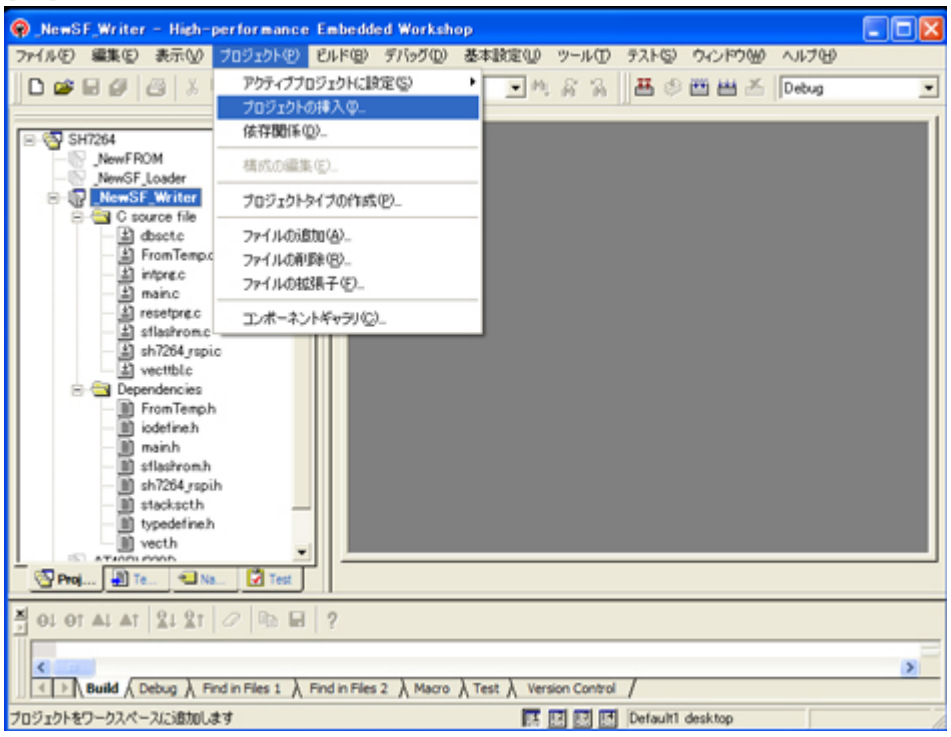
＜ー完了をクリックします。

この操作で「品種追加前の準備」は完了です。

【新規シリアルＦｌａｓｈROM用ライタの追加】

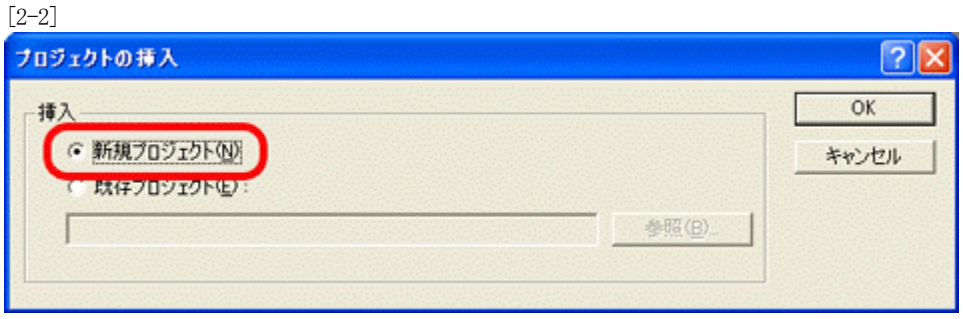
1．新規シリアルＦｌａｓｈROM用ライタを追加するため、「プロジェクトの挿入」をします。

[2-1]



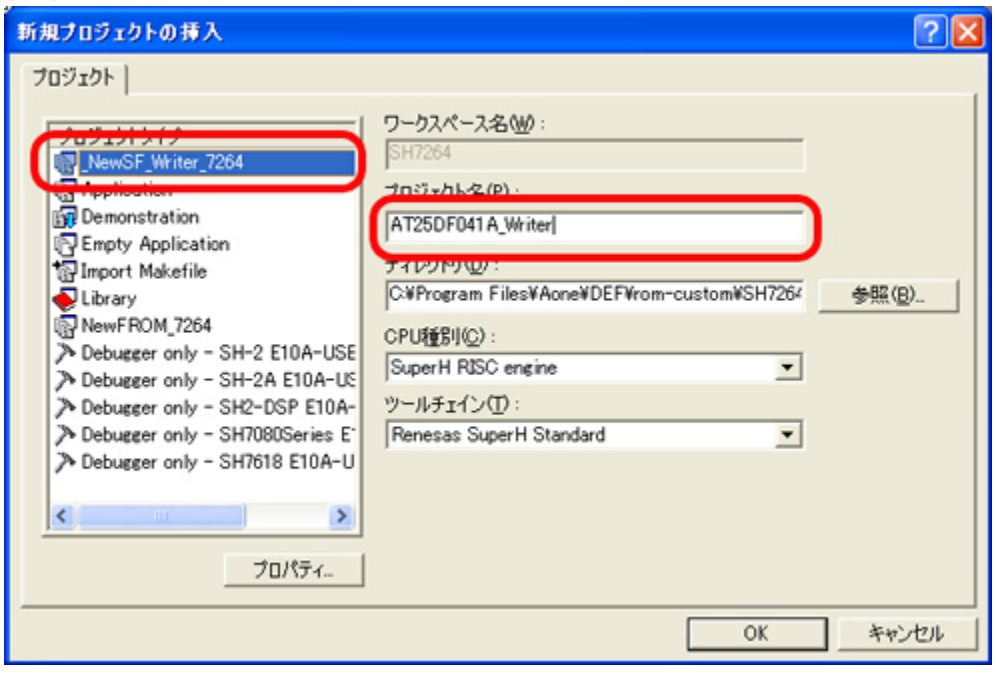
Ｈｅｗメニュー
＜プロジェクト＞ー＜プロジェクトの挿入＞をクリックします。

[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]



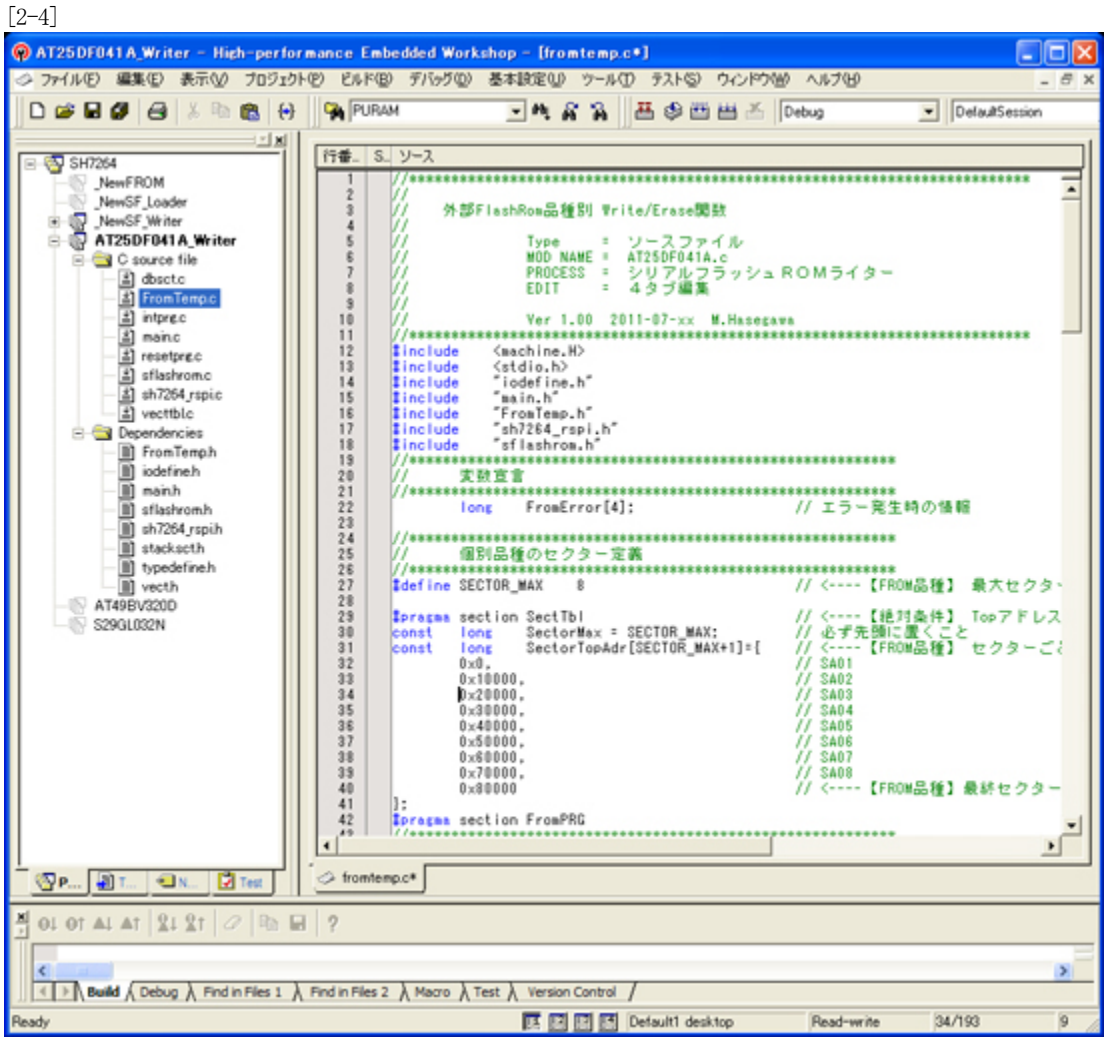
1）プロジェクトタイプに作成した
「ＮｅｗＳＦ＿Ｗｒｉｔｅｒ＿７２６４」を指定します。

2）プロジェクト名を指定します。FlashROM 名を追加した
「ＡＴ２５ＤＦ０４１Ａ＿Ｗｒｉｔｅｒ」にします。

＜ー「OK」をクリックします。

２．新規シリアルＦｌａｓｈＲＯＭ用ライタの定義および制御プログラムを作成します。

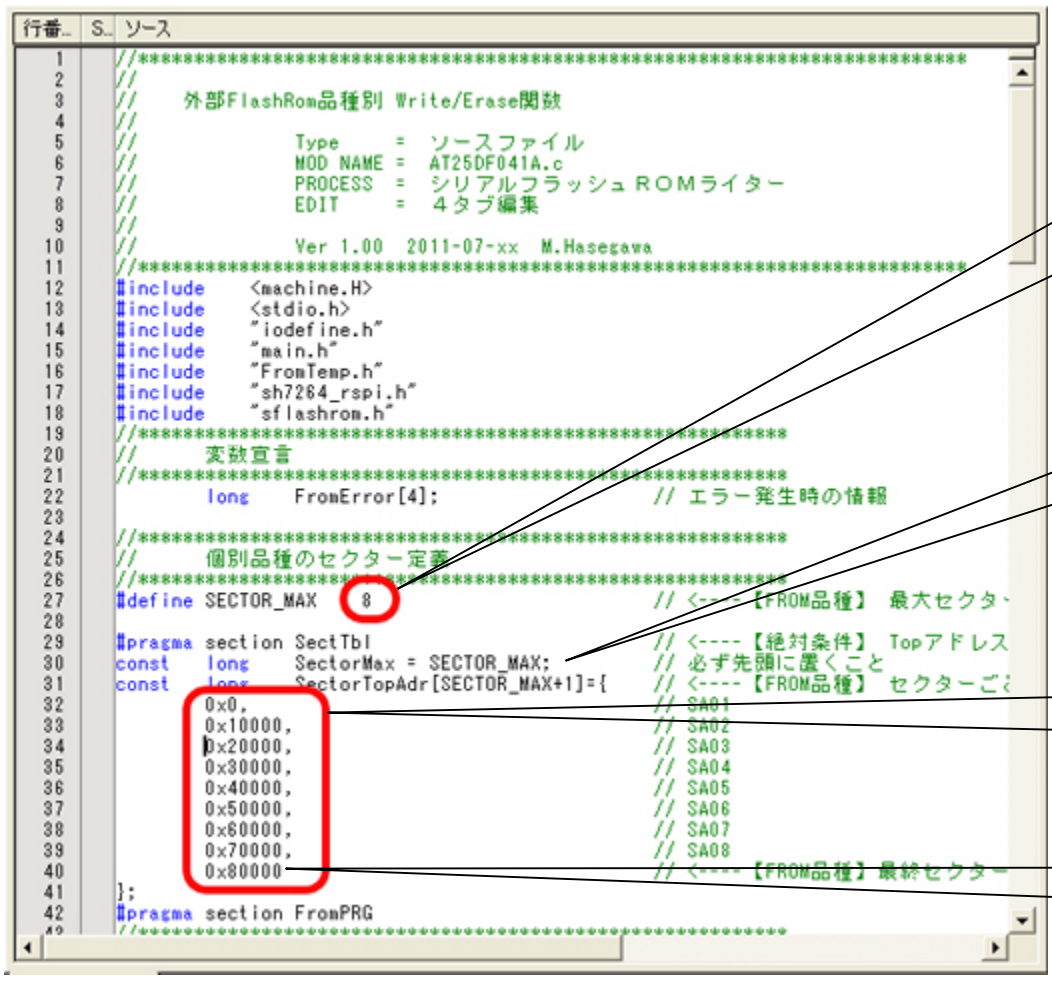
[2-4]



新規シリアルＦｌａｓｈＲＯＭ用の定義および制御プログラムファイル「FromTemp.c」を開きます。

１）「FromTemp.c」に新規シリアルＦｌａｓｈＲＯＭのセクター情報を定義します。

[2-5]



```
//*****************************************************************************
//
//    外部FlashRom品種別 Write/Erase関数
//
//              Type     =  ソースファイル
//              MOD NAME =  AT25DF041A.c
//              PROCESS  =  シリアルフラッシュＲＯＭライター
//              EDIT     =  ４タブ編集
//
//              Ver 1.00  2011-07-xx  M.Hasegawa
//*****************************************************************************
#include    <machine.H>
#include    <stdio.h>
#include    "iodefine.h"
#include    "main.h"
#include    "FromTemp.h"
#include    "sh7264_rspi.h"
#include    "sflashrom.h"
//*********************************************************
//      変数宣言
//*********************************************************
        long    FromError[4];                   // エラー発生時の情報

//*********************************************************
//      個別品種のセクター定義
//*********************************************************
#define SECTOR_MAX    8                         // <----【FROM品種】 最大セクタ
#pragma section SectTbl                         // <----【絶対条件】 Topアドレス
const   long    SectorMax = SECTOR_MAX;         // 必ず先頭に置くこと
const   long    SectorTopAdr[SECTOR_MAX+1]={    // <----【FROM品種】 セクターご
        0x0,                                    // SA01
        0x10000,                                // SA02
        0x20000,                                // SA03
        0x30000,                                // SA04
        0x40000,                                // SA05
        0x50000,                                // SA06
        0x60000,                                // SA07
        0x70000,                                // SA08
        0x80000                                 // <----【FROM品種】最終セクター
};
#pragma section FromPRG
```

「SECTOR_MAX」に使用するシリアルＦｌａｓｈＲＯＭ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。

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」に新規シリアルＦｌａｓｈＲＯＭ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
42      #pragma section FromPRG
43      //*********************************************************
44      //      FromOpen()
45      //      FlashRomアクセス時に初期化が必要な場合の手続き
46      //      不要な場合、プログラム記述しなくても良い
47      //      開始時に1回だけ呼ばれる
48      //      return()
49      //        OK  == 正常終了
50      //        NG  == 異常終了
51      //*********************************************************
52      int FromOpen(void)
53      {
54          IoInitRSPI0();                      // RSPIの初期化
55          sFlashUnProtect();                  // アンプロテクト
56
57          return(OK);
58      }
59      //*********************************************************
60      //      FromClose()
61      //      FlashRom書き込み処理が全て終了した場合に必要な手続き
62      //      不要な場合、プログラム記述しなくても良い
63      //      終了時に1回だけ呼ばれる
64      //      return()
65      //        OK  == 正常終了
66      //        NG  == 異常終了
67      //*********************************************************
68      int FromClose(void)
69      {
70          sFlashProtect();                    // プロテクト
71
72          return(OK);
73      }
74      //*********************************************************
```

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、プロテクト解除（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。
[IoInitRSPI0()¥sh7264_rspi.c] RSPI0の初期化関数
[sFlashUnProtect()¥sfflashrom.c] アンプロテクト関数

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。
このプログラム例は、プロテクト（ロック）しています。必要無い場合は、「return(Ok);」のみ記述します。
[sFlashProrect()¥sflashrom.c]プロテクト関数

3）「FromTemp.c」に新規シリアルＦｌａｓｈＲＯＭ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
73      }
74      //*********************************************************
75      //      FromSectorEraseProc(char *cmd)
76      //      伝送フォーム(in)
77      //        cmd[0]    == %
78      //            [1]   == 0
79      //            [2]   == CommandSize[2]以降[Sum]も含む
80      //            [3]   == Command('D')
81      //            [4]   == Sector(H) MAX(1024)
82      //            [5]   == Sector(L)
83      //            [Sum] == [3]->[Sum-1]の加算
84      //      return()
85      //        OK  == 正常終了
86      //        NG  == 異常終了
87      //*********************************************************
88      int  FromSectorEraseProc(char *cmd)
89      {
90          Ulong  adr;                         // セクターの先頭アドレ
91          short  sector;
92          int    stat;
93
94          sector  = (cmd[4] << 8) & 0xff00;
95          sector |= cmd[5] & 0xff;
96          if (sector < SectorMax) {
97              adr = SectorTopAdr[sector];
98              sFlashSectorErase(adr);         // セクターイレーズ
99              stat = FullStatusCheck(adr,'D');
100             if (stat == OK) return(OK);
101         }
102         return(NG);
103     }
104     //*********************************************************
105     //      FromWriteProc(char *cmd)
```

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。

[sFlashSectorErase()¥sflashrom.c]セクターイレーズ関数

4）「FromTemp.c」に新規シリアルＦｌａｓｈＲOMの「バイト（書き込み)」関数を作成します。
[2-8]

```
行番..  S..  ソース
104     //*************************************************************
105     //      FromWriteProc(char *cmd)
106     //      伝送フォーム(in)
107     //        cmd[0]      == %
108     //            [1]     == 0
109     //            [2]     == CommandSize[2]以降[Sum]も含む
110     //            [3]     == Command('W')
111     //            [4]     == WriteAddress(HH)
112     //            [5]     == WriteAddress(MH)
113     //            [6]     == WriteAddress(ML)
114     //            [7]     == WriteAddress(LL)
115     //            [8+0]   == WriteData[0]
116     //             .      ==   "
117     //             .      ==   "
118     //            [8+127] == WriteData[127] 固定 余分エリア(0xff
119     //            [Sum] == [3]->[Sum-1]の加算
120     //      return()
121     //        OK  == 正常終了
122     //        NG  == 異常終了
123     //*************************************************************
124     int  FromWriteProc(char *cmd)
125     {
126         Ulong   adr;                            // 先頭アドレス
127         Uchar   *data;
128         Uchar   verf[BLOCK];
129         int     i;
130         int     stat;
131
132         adr =  (((long)cmd[4] << 24) & 0xff000000) |
133                (((long)cmd[5] << 16) & 0x00ff0000) |
134                (((long)cmd[6] << 8)  & 0x0000ff00) |
135                 ((long)cmd[7]        & 0x000000ff);
136
137         data = (Uchar *)&cmd[8];                // Data Adr set
138         sFlashByteProgram(adr,data,BLOCK);      // 指定バイト数の書き
139         stat = FullStatusCheck(adr,'W');
140         if (stat == NG) return(NG);
141
142         sFlashByteRead(adr,verf,BLOCK);         // ベリファイテスト
143         for(i = 0;i < BLOCK;i++){               // VerifyTest
144             if (verf[i] != data[i]) {           // VerifyError
145                 FromErrorSet((long)'R',adr,(long)data[i]);
146                 return(NG);
147             }
148             ++adr;
149         }
150         return(OK);
151     }
152     //*************************************************************
```

関数「FromWriteProc(char *cmd)」には、128 バイト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)固定でデータを受信し完了後、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。

[sFlashByteProgram()¥sflashrom.c]バイト書き込み関数
[sFlashByteRead()¥sflashrom.c]バイト読み出し関数

5）「FromTemp.c」に新規シリアルＦｌａｓｈＲOM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
152     //*************************************************************
153     //      FullStatusCheck()                           Word Access
154     //      フルステータスをチェックエラー発生時は、エラー情報
155     //      を[FromError]に残す
156     //      FromErro[0] == command 'W'or'D'or'E' or 'O'==Open
157     //                  [1] == アクセスしたアドレス
158     //                  [2] == FROM ステータス
159     //      return()
160     //        OK  == 正常終了
161     //        NG  == 異常終了
162     //*************************************************************
163     int     FullStatusCheck(Ulong adr,char cmd)
164     {
165         Uchar   stat;
166
167         FromErrorSet(0,0,0);                    // Error情報を消去
168         stat = sFlashReadStatus();
169         if (stat & EPE) {                       // Error発生
170             FromErrorSet(cmd,adr,stat);
171             return(NG);
172         }
173         return(OK);
174     }
175     //*************************************************************
```

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。

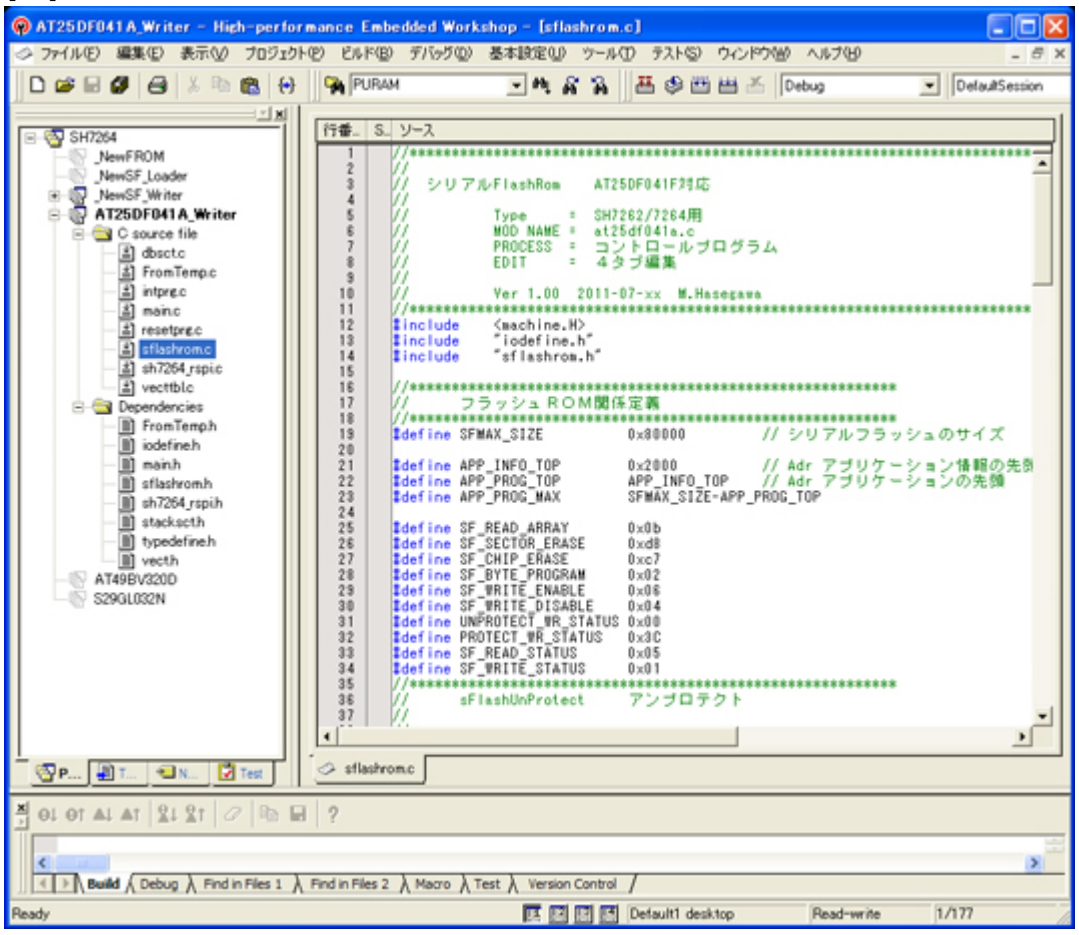
＜引数の仕様＞
Ushort　adr;　// 処理しているフラッシュ ROM のアドレス
char　cmd;　// 処理しているコマンド
// 'O' = オープンコマンド
// 'C' = クローズコマンド
// 'D' = イレーズコマンド
// 'W' = 書き込みコマンド
// 'R' = ベリファイコマンド

＜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。

[sFlashReadStatus()¥sflashrom.c]ステータスリード関数

３．新規シリアルＦｌａｓｈROMに依存した定義およびコマンドプログラムを作成します。

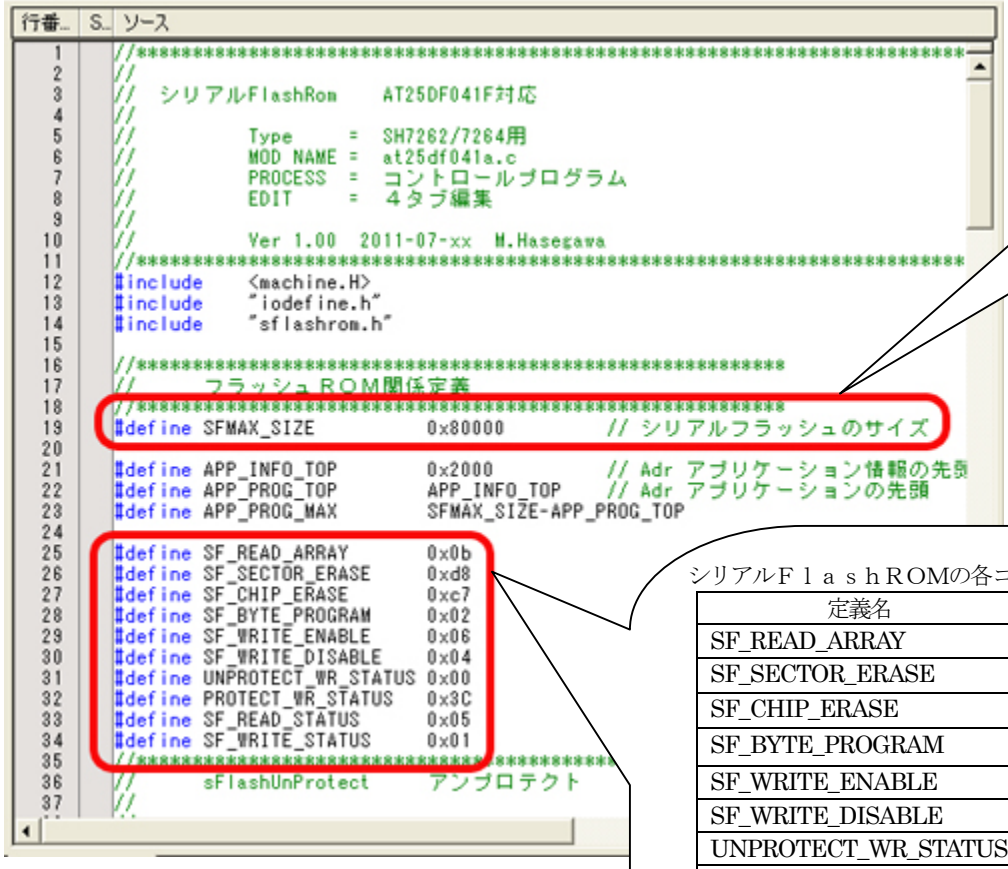
[3-1]



新規シリアルＦｌａｓｈROMに依存した定義およびコマンドプログラムファイル「sflashrom.c」を開きます。

１）「sflashrom.c」に新規シリアルＦｌａｓｈROMに依存したコマンドを定義します。

[3-2]



「SFMAX_SIZE」に使用するシリアルＦｌａｓｈROMの最大サイズ（バイト）を定義します。

シリアルＦｌａｓｈROMの各コマンドを定義します。

定義名	コマンド	機能
SF_READ_ARRAY	0x0B	データの読み出し
SF_SECTOR_ERASE	0xD8	セクターイレーズ
SF_CHIP_ERASE	0xC7	チップイレーズ
SF_BYTE_PROGRAM	0x02	データの書き込み（バイト）
SF_WRITE_ENABLE	0x06	書き込み／イレーズ／ライトステータス許可
SF_WRITE_DISABLE	0x04	書き込み／イレーズ／ライトステータス禁止
UNPROTECT_WR_STATUS	0x00	ライトプロテククの解除
PROTECT_WR_STATUS	0x3C	ライトプロテクトの設定
SF_READ_STATUS	0x05	ステータスレジスタの読み出し
SF_WRITE_STATUS	0x01	ステータスレジスタの書き込み

この例は、ATMEL 社の「AT25DF041A」のコマンドです。コマンド仕様は各デバイスメーカーによって異なります。使用するデバイスのデータシートを参照して下さい。

2）「sflashrom.c」に新規シリアルＦｌａｓｈＲＯＭに依存したコマンドプログラムを作成します。

⚠

下記の参考プログラム例は、ATMEL 社の「AT25DF041A」用コマンドプログラムです。コマンドパラメータ仕様は各デバイスメーカーによって異なります。使用するデバイスのデータシートを参照してから作成して下さい。

[3-3]

```
行番..  S..  ソース
 35      //************************************************************
 36      //      sFlashUnProtect     アンプロテクト
 37      //
 38      //************************************************************
 39      void    sFlashUnProtect(void)
 40      {
 41          sFlashWriteStatus(UNPROTECT_WR_STATUS);
 42      }
 43      //************************************************************
 44      //      sFlashProtect       プロテクト
 45      //
 46      //************************************************************
 47      void    sFlashProtect(void)
 48      {
 49          sFlashWriteStatus(PROTECT_WR_STATUS);
 50      }
```

関数「sFlashUnProtext(void)」は、デバイスのライトプロテクトを解除します。

関数「sFlashProtext(void)」は、デバイスをライトプロテクトします。

[3-4]

```
行番..  S..  ソース
 51      //************************************************************
 52      //      sFlashChipErase     チップイレーズ
 53      //
 54      //************************************************************
 55      void    sFlashChipErase(void)
 56      {
 57          Uchar   cmd;
 58
 59          sFlashWriteEnable();
 60
 61          cmd = SF_CHIP_ERASE;
 62          IoCmdExe(&cmd,1,0,0);
 63
 64          sFlashBusyWait();
 65      }
 66      //************************************************************
 67      //      sFlashSectorErase   セクターイレーズ
 68      //
 69      //************************************************************
 70      void    sFlashSectorErase(Ulong adr)
 71      {
 72          Uchar   cmd[4];
 73
 74          sFlashWriteEnable();
 75
 76          cmd[0] = SF_SECTOR_ERASE;
 77          cmd[1] = (Uchar)((adr >> 16) & 0xff);
 78          cmd[2] = (Uchar)((adr >> 8)  & 0xff);
 79          cmd[3] = (Uchar)( adr        & 0xff);
 80          IoCmdExe(cmd,4,0,0);
 81
 82          sFlashBusyWait();
 83      }
```

関数「sFlashChipErase(void)」は、デバイス全体をイレーズします。

[IoCmdExe()¥sh7264_rspi.c] RSPI0 コマンド（リードデータなしタイプ）送信関数です。

関数「sFlashSectorErase(void)」は、セクター単位でイレーズします。

＜引数の仕様＞
Ulong　adr ; // イレーズするデバイスセクター内のアドレス

[IoCmdExe()¥sh7264_rspi.c] RSPI0 コマンド（リードデータなしタイプ）送信関数です。

[3-5]

```
行番..  S..  ソース
 84      //************************************************************
 85      //      sFlashByteProgram   指定バイト数の書き込み
 86      //
 87      //************************************************************
 88      void    sFlashByteProgram(Ulong adr,Uchar *wdat,Ulong wsize)
 89      {
 90          Uchar   cmd[4];
 91
 92          sFlashWriteEnable();
 93
 94          cmd[0] = SF_BYTE_PROGRAM;
 95          cmd[1] = (Uchar)((adr >> 16) & 0xff);
 96          cmd[2] = (Uchar)((adr >> 8)  & 0xff);
 97          cmd[3] = (Uchar)( adr        & 0xff);
 98          IoCmdExe(cmd,4,wdat,wsize);
 99
100          sFlashBusyWait();
101      }
102      //************************************************************
103      //      sFlashByteRead      指定バイト数の読み込み
104      //
105      //************************************************************
106      void    sFlashByteRead(Ulong adr,Uchar *rdat,Ulong rsize)
107      {
108          Uchar   cmd[5];
109
110          cmd[0] = SF_READ_ARRAY;
111          cmd[1] = (Uchar)((adr >> 16) & 0xff);
112          cmd[2] = (Uchar)((adr >> 8)  & 0xff);
113          cmd[3] = (Uchar)( adr        & 0xff);
114          cmd[4] = 0;
115          IoCmdExeRead(cmd,5,rdat,rsize);
116      }
```

関数「sFlashByteProgram(....)」は、バイト書き込みします。

＜引数の仕様＞
Ulong　adr ; // 書き込みするデバイス内の先頭アドレス
Uchar *wdt; // 書き込みデータの先頭アドレス
Ulong　wsize; // 書き込みデータのサイズ（バイト）

[IoCmdExe()¥sh7264_rspi.c] RSPI0 コマンド（リードデータなしタイプ）送信関数です。

関数「sFlashByteRead(....)」は、バイト読み出しします。

＜引数の仕様＞
Ulong　adr ; // 読み出しするデバイス内の先頭アドレス
Uchar *rdt; // 読み出したデータを格納する先頭アドレス
Ulong　rsize; // 読み出しデータのサイズ（バイト）

[IoCmdExeRead()¥sh7264_rspi.c] RSPI0 コマンド（リードデータありタイプ）送信関数です。

[3-6]

```
行番… S… ソース
117     //*********************************************************
118     //      sFlashWriteStatus   sFlashROMステータスWrite
119     //
120     //*********************************************************
121     void    sFlashWriteStatus(Uchar status)
122     {
123         Uchar cmd[2];
124
125         sFlashWriteEnable();
126
127         cmd[0] = SF_WRITE_STATUS;
128         cmd[1] = status;
129         IoCmdExe(cmd,2,0,0);
130
131         sFlashBusyWait();
132     }
133     //*********************************************************
134     //      sFlashWriteEnable   sFlashROMへの書き込み許可
135     //
136     //*********************************************************
137     void    sFlashWriteEnable(void)
138     {
139         Uchar cmd;
140
141         cmd = SF_WRITE_ENABLE;
142         IoCmdExe(&cmd,1,0,0);
143     }
144     //*********************************************************
145     //      sFlashWriteDisable  sFlashROMへの書き込み禁止
146     //
147     //*********************************************************
148     void    sFlashWriteDisable(void)
149     {
150         Uchar cmd;
151
152         cmd = SF_WRITE_DISABLE;
153         IoCmdExe(&cmd,1,0,0);
154     }
155     //*********************************************************
```

関数「sFlashWriteStatus(...)」は、シリアルＦｌａｓｈＲＯＭのステータスレジスタに書き込む内部関数です。

＜引数の仕様＞
Uchar　status;// ステータスレジスタに書き込むデータ

[IoCmdExe()¥sh7264_rspi.c] RSPI0 コマンド（リードデータなしタイプ）送信関数です。

関数「sFlashWriteEnable(void)」は、シリアルＦｌａｓｈＲＯＭの書き込み／イレーズ／ステータスレジスタ書き込みを許可する内部関数です。

[IoCmdExe()¥sh7264_rspi.c] RSPI0 コマンド（リードデータなしタイプ）送信関数です。

関数「sFlashWriteDisable(void)」は、シリアルＦｌａｓｈＲＯＭの書き込み／イレーズ／ステータスレジスタ書き込みを禁止する内部関数です。

[IoCmdExe()¥sh7264_rspi.c] RSPI0 コマンド（リードデータなしタイプ）送信関数です。

[3-7]

```
行番… S… ソース
155     //*********************************************************
156     //      sFlashBusyWait      sFlashROMがBusy中は待つ
157     //
158     //*********************************************************
159     void    sFlashBusyWait(void)
160     {
161         while((sFlashReadStatus() & 0x01) != 0){}   // 0==RDY 0!=
162     }
163     //*********************************************************
164     //      sFlashReadStatus    sFlashROMステータスRead
165     //
166     //*********************************************************
167     Uchar   sFlashReadStatus(void)
168     {
169         Uchar buf;
170         Uchar cmd;
171
172         cmd = SF_READ_STATUS;
173         IoCmdExeRead(&cmd,1,&buf,1);
174
175         return(buf);
176     }
```

関数「sFlashBusyWait(void)」は、コマンド処理終了を待つ内部関数です。

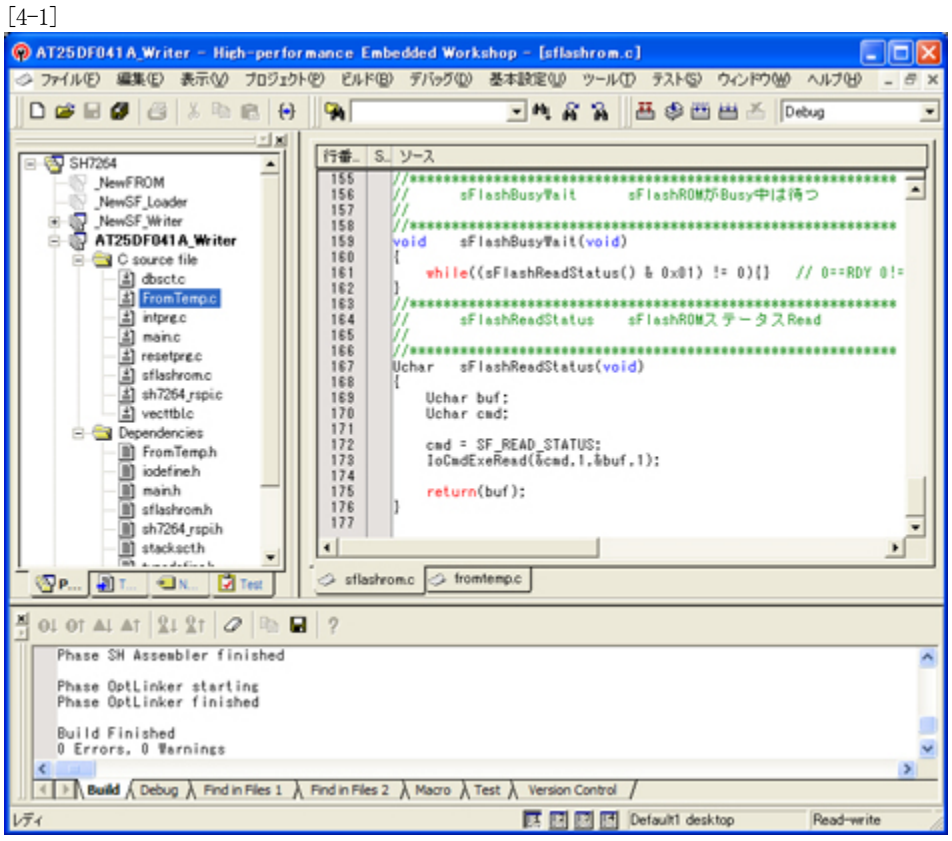
関数「sFlashReadStatus(...)」は、シリアルＦｌａｓｈＲＯＭのステータスレジスタを読み出す内部関数です。

[IoCmdExeRead()¥sh7264_rspi.c] RSPI0 コマンド（リードデータありタイプ）送信関数です。

以上で、関数の作成は終了です。

4．新規シリアルＦｌａｓｈROMライタ用プログラムをビルドします。

[4-1]



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。

1）新規シリアルＦｌａｓｈROMライタ用プログラム作成上のルール

[4-2]

＜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。

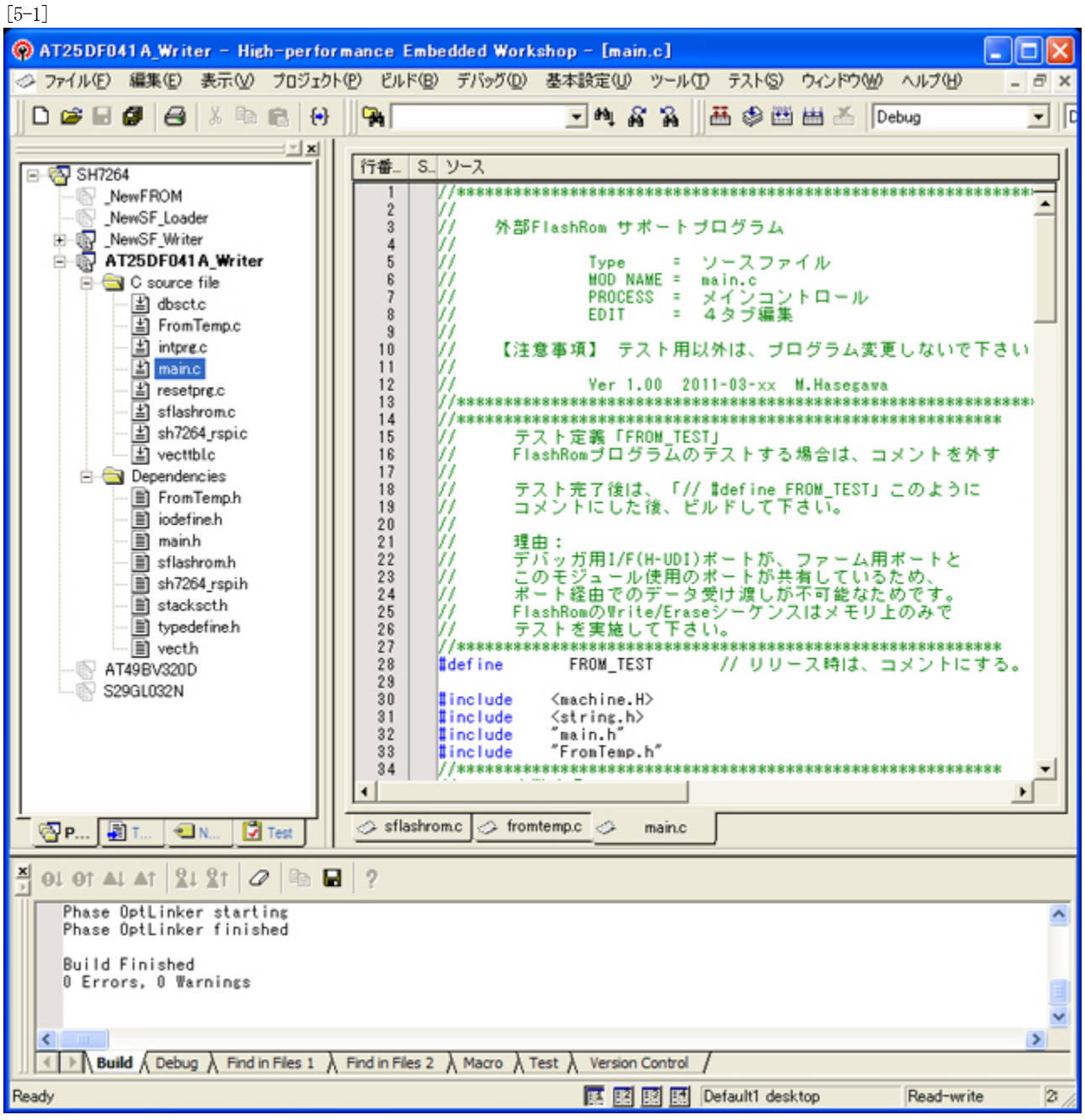
＜ルール６＞
下記４関数は、名称・引数・帰り値等の仕様変更は不可です。
1)int　　FromOpen(void);
2)int　　FromClose(void);
3)int　　FromSectorEraseProc(char *cmd);
4)int　　FromWriteProc(char *cmd);

以上のルールは厳守して下さい。

【作成したシリアルＦｌａｓｈＲＯＭライタプログラムのデバッグ方法】

⚠デバッグ中は、「ブートモード０」にするか、もしくは「ブートモード０」に出来ない回路構成（アドレス・データバスを汎用Ｉ／Ｏに使用）の場合は、プログラムの開始アドレスを「0xFFF8_2000」にロケートしなおしDEF 設定も「0xFFF8_2000」に設定してデバッグして下さい。デバッグ終了後は「0xFFF8_000」に戻して下さい。
理由：
「ブートモード１・３」のままですとリセット解除後、ブートプログラムが起動され内蔵高速ＲＡＭの先頭から「0x2000」エリアにローダプログラムがロードされてしまいます。

１．Ｈｅｗにてデバッグの準備をします。
［5-1］



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

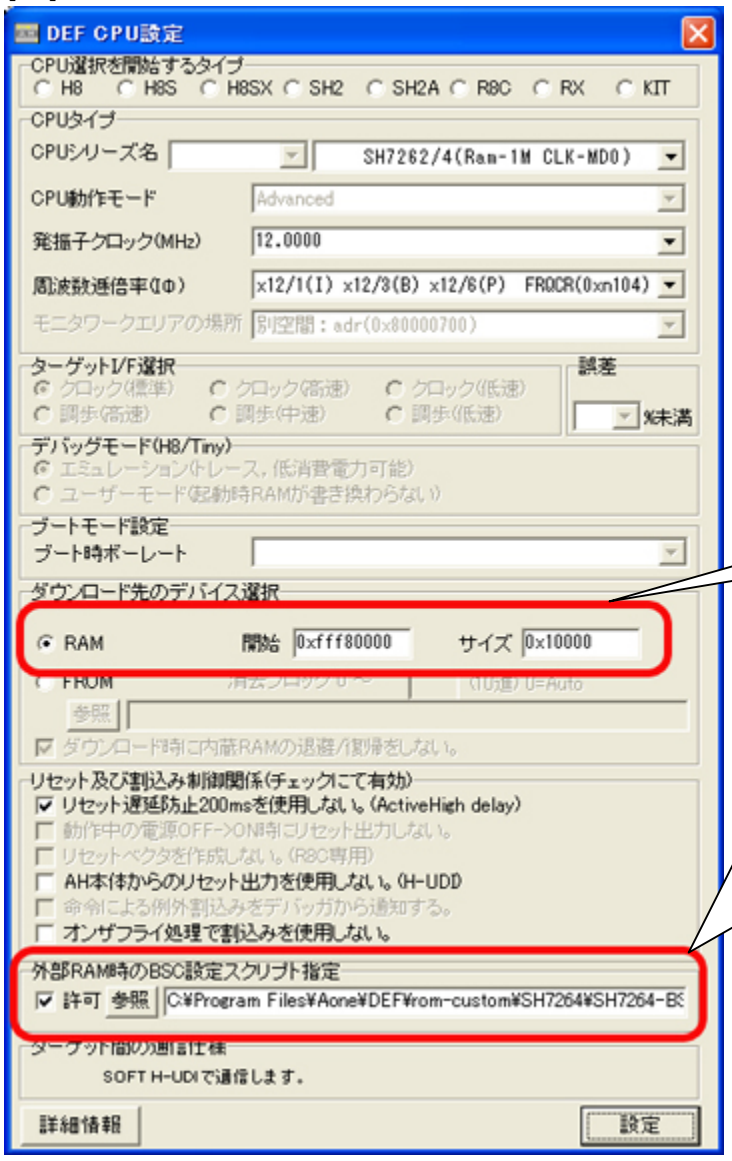
2）Ｈｅｗメニューの＜ビルド＞ー＜ビルド＞を指示します。

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ております。

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。

[5-2]



＜ＣＰＵ設定＞

作成した「シリアルFlashROMライタ」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、先頭アドレスとサイズを指定します。

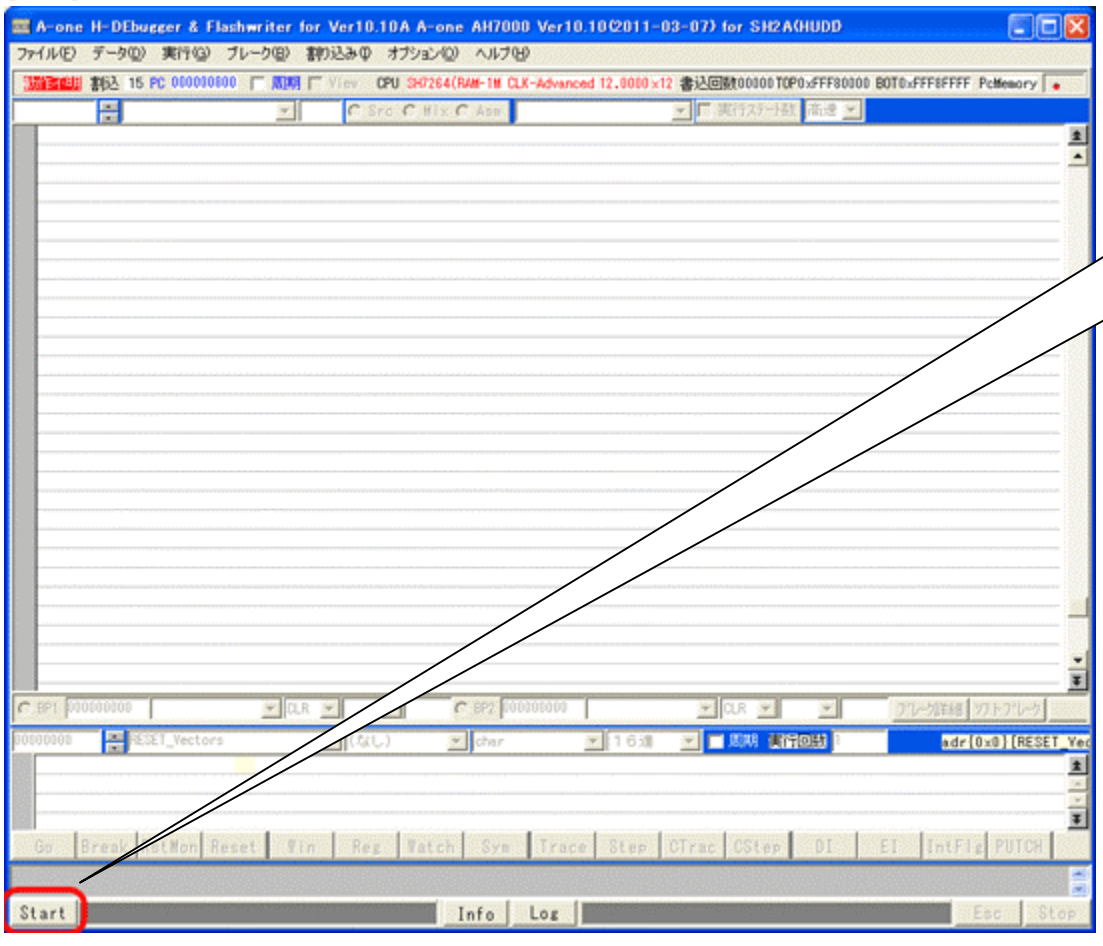
ＳＨ７２６４の場合
開始　0xfff80000　　サイズ　0x10000
になります。

作成した「シリアルFlashROMライタ」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-HSB.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。

＜－「設定」をクリックします。

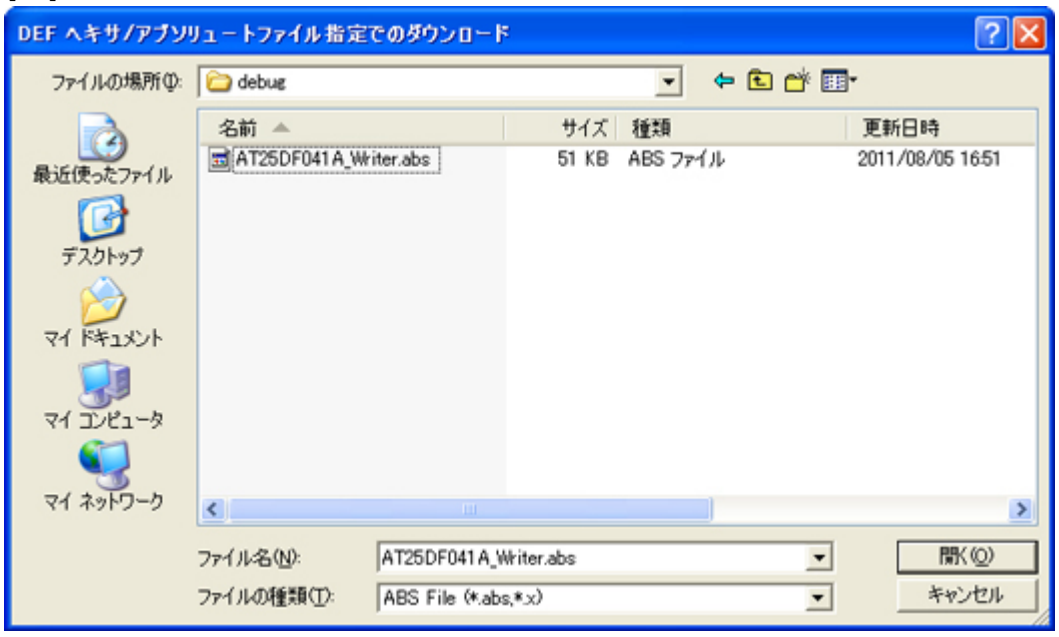
[5-2-1]



ＤＥＦ画面、左下隅の「Start」をクリックします。

3．作成したシリアルＦｌａｓｈROMライタソフトのデバッグを開始する準備をする。

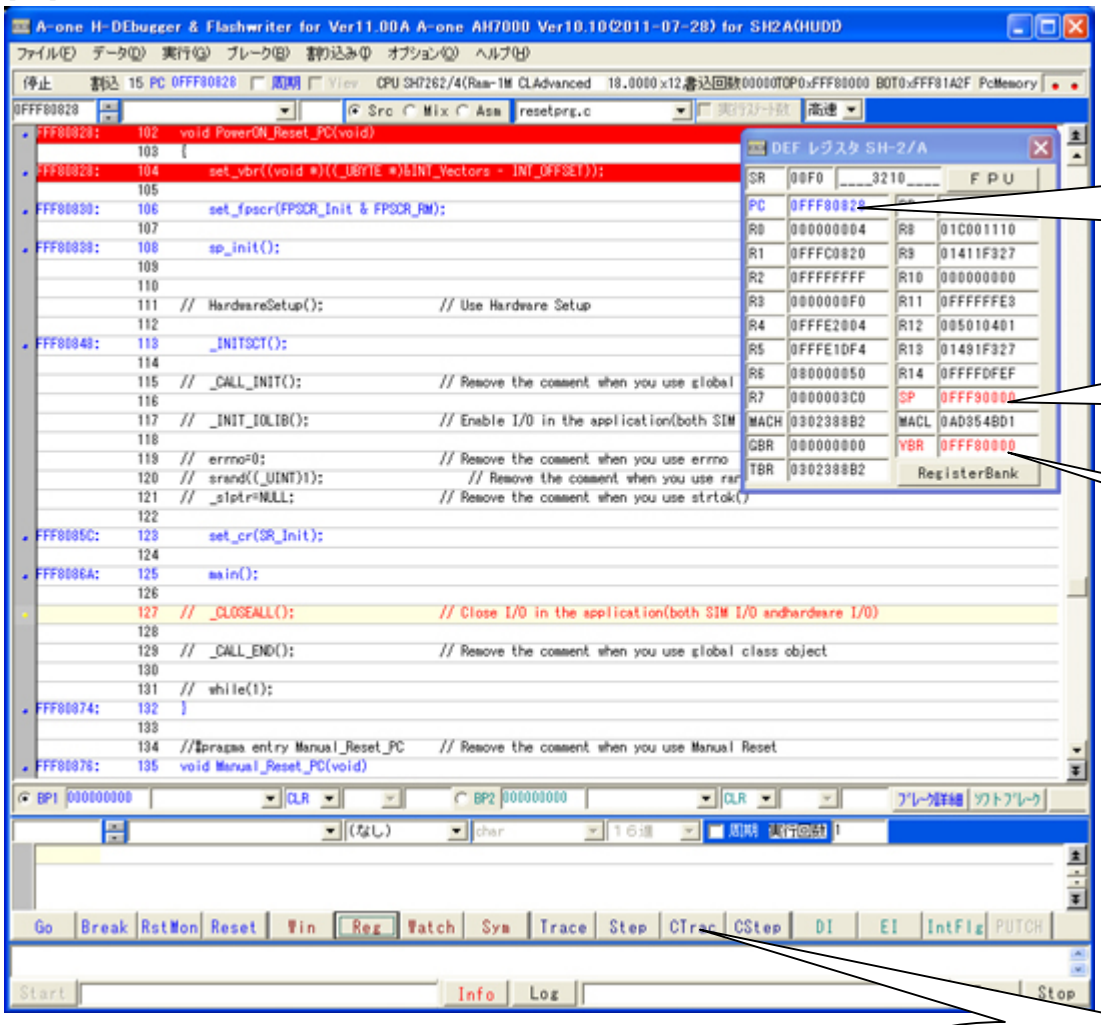
[5-3]



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):AT25DF041A_Writer
「AT25DF041A_Writer.abs」

[5-4]



＜ダウンロードが成功した初期画面＞

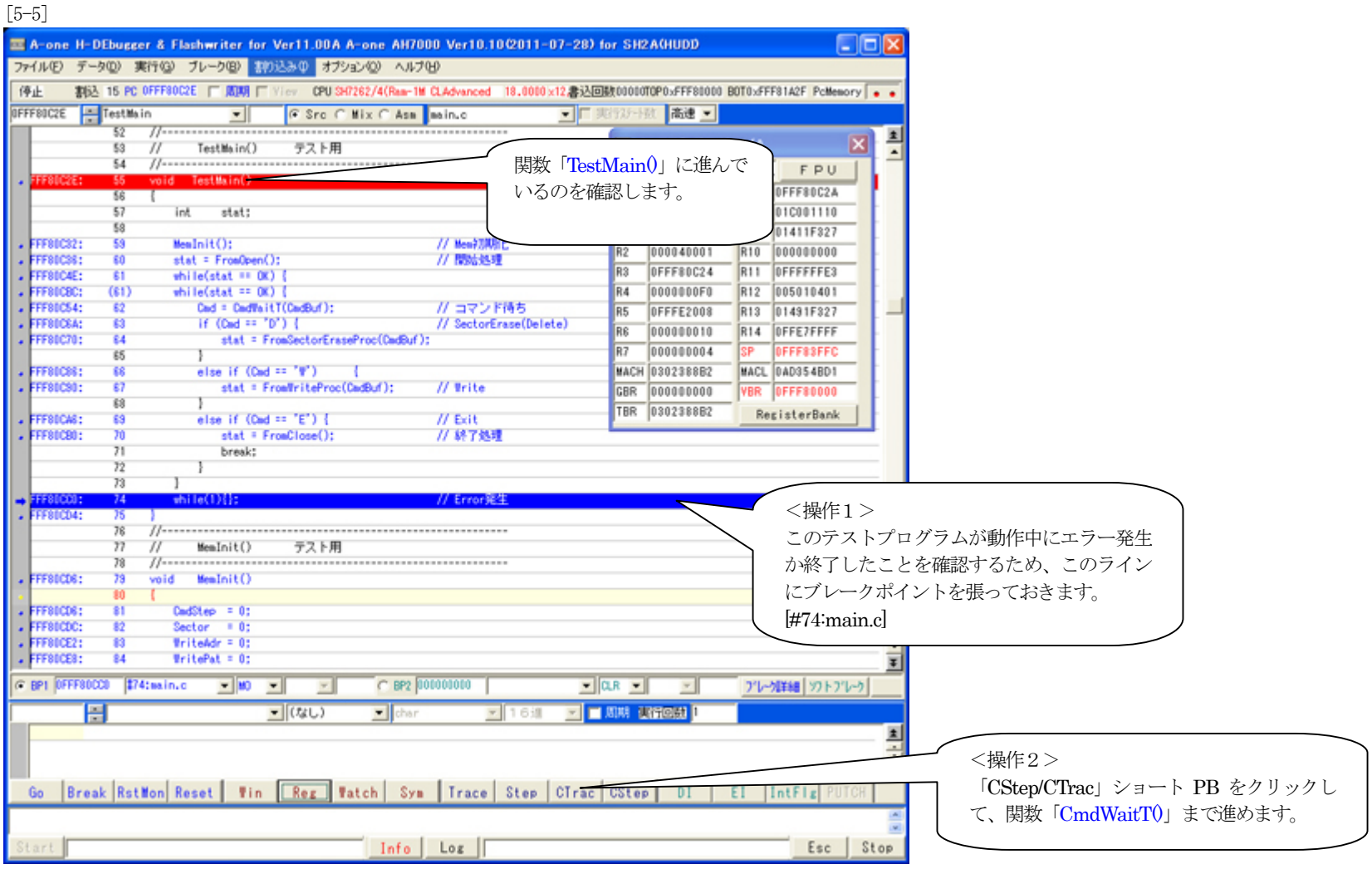
PC レジスタ値が「PowerON_Reset_PC()」関数の先頭アドレスになっていることが確認出来ます。

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。

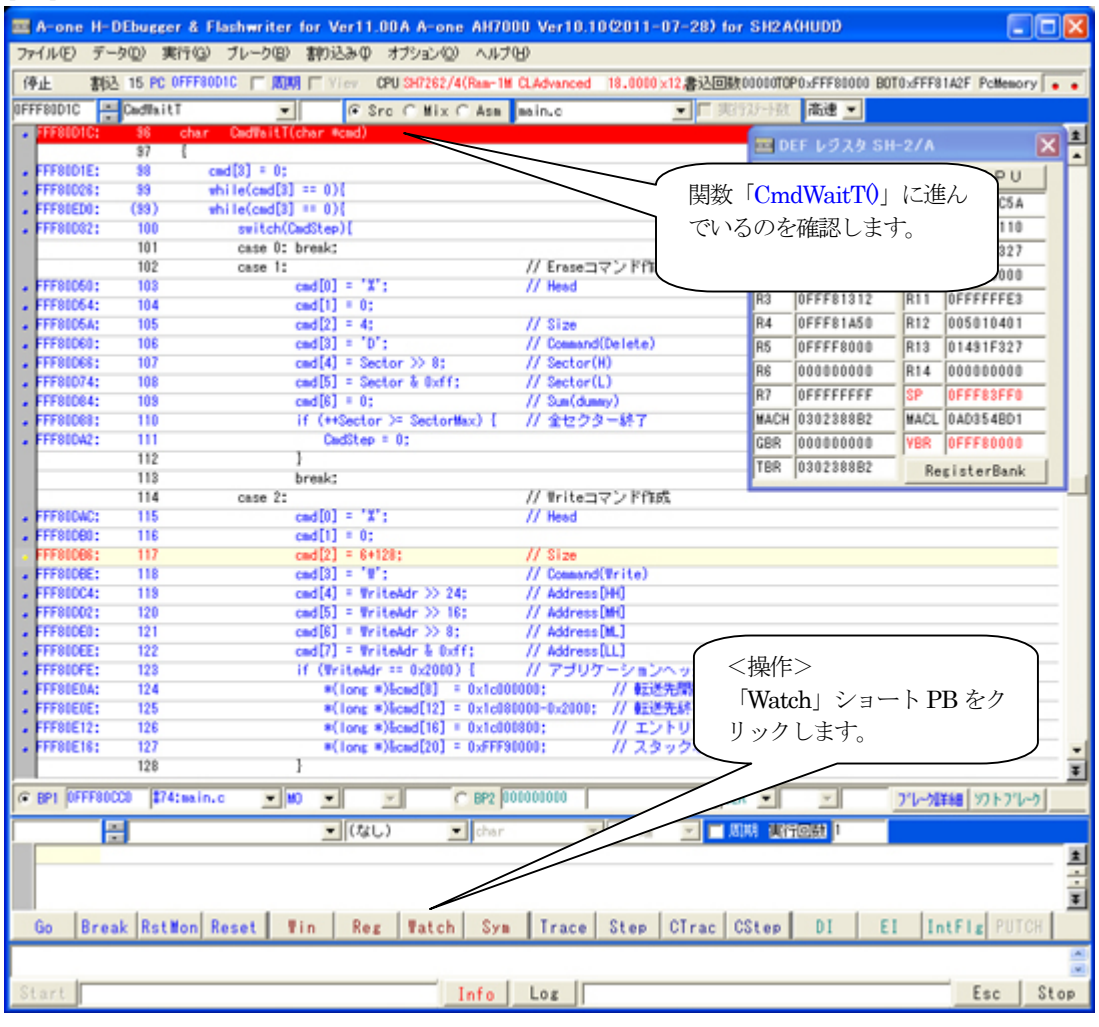
VBR レジスタ値が内蔵 RAM のトップアドレスになっていることが確認できます。

＜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。

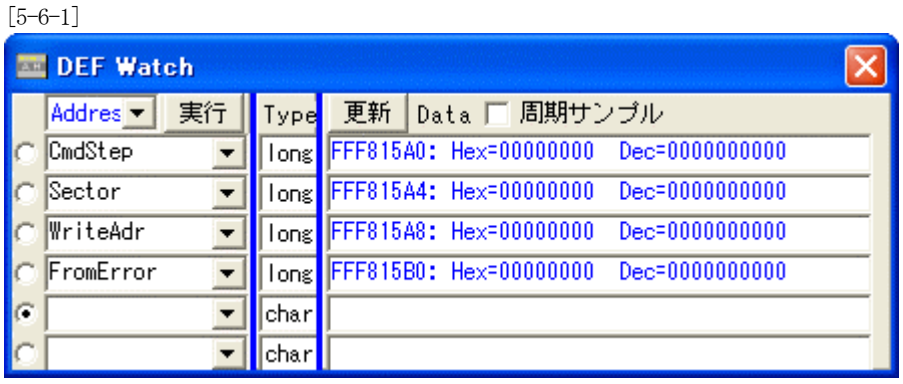
[5-5]


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。

[5-6]


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。

[5-6-1]



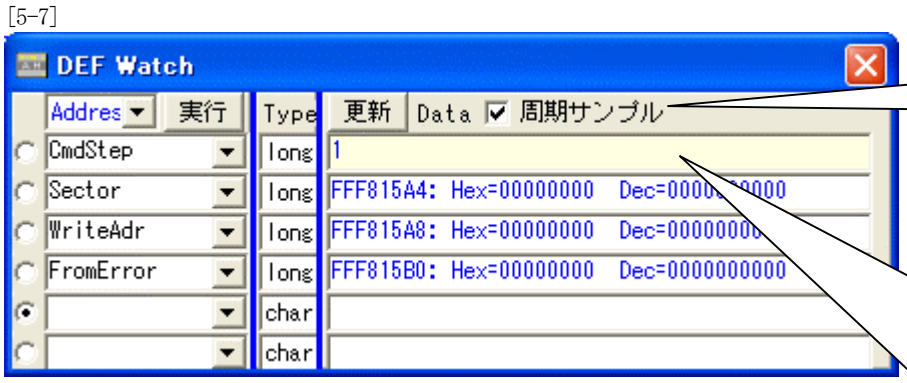
デバッグを進めるため、Watch 画面に変数を登録します。

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。

[5-7]

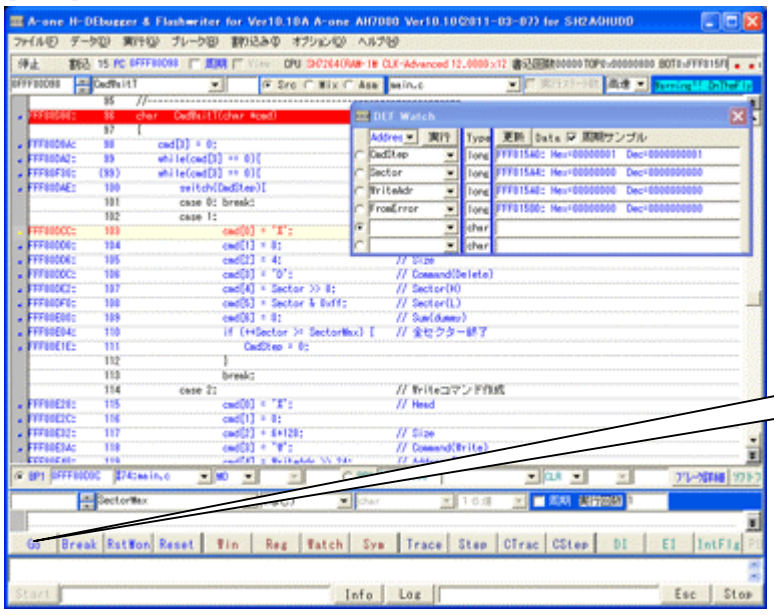


＜操作１＞
プログラム進行を確認するため、「周期サンプル」をチェックします。

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

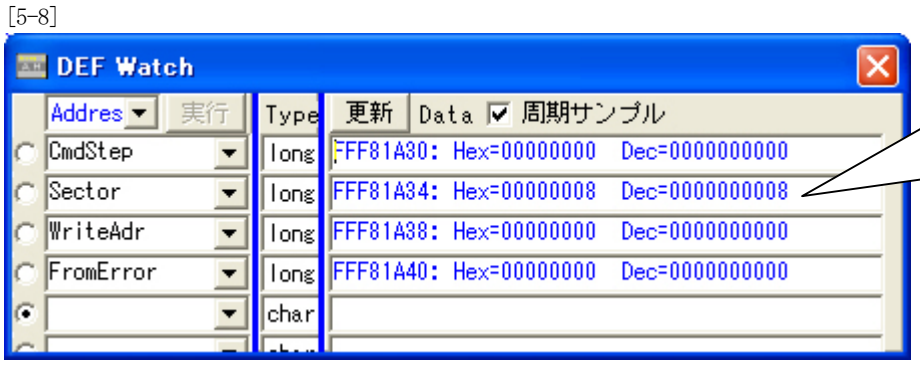
＜備考＞
関数「CmdWaitT」のソースを見てもらえれば理解できるかと思いますが、「CmdStep」を「1」にしますと、メモリー上でコマンドデータを作成して関数「FromSectorEraseProc(char *cmd)」に渡す仕組みになっています。
全て正常終了しますと、登録された全セクターをイレーズした後、変数「CmdStep」が数値「0」になります。

[5-7-1]



＜操作３＞
「Go」ショート PB をクリックします。

[5-8]

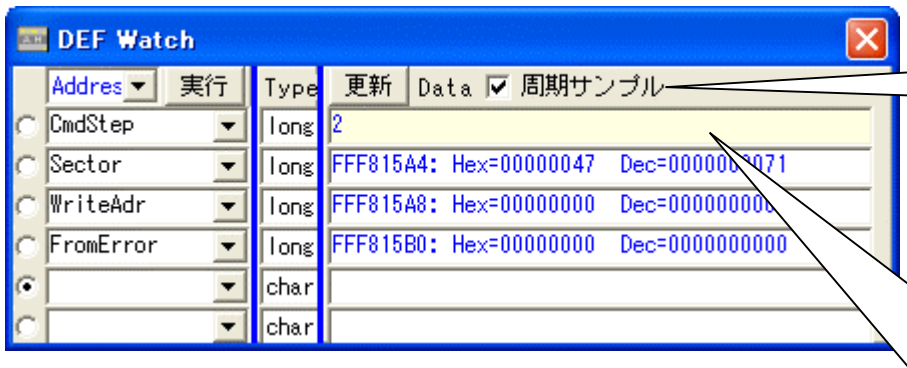


セクターイレーズが正常に動作していますと、変数「Sector」が、ゼロ「0」から登録セクター数までインクリメントしていきます。
最終セクターまで正常終了しますと、変数「CmdStep」がゼロ「0」になります。

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。

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

[5-10]



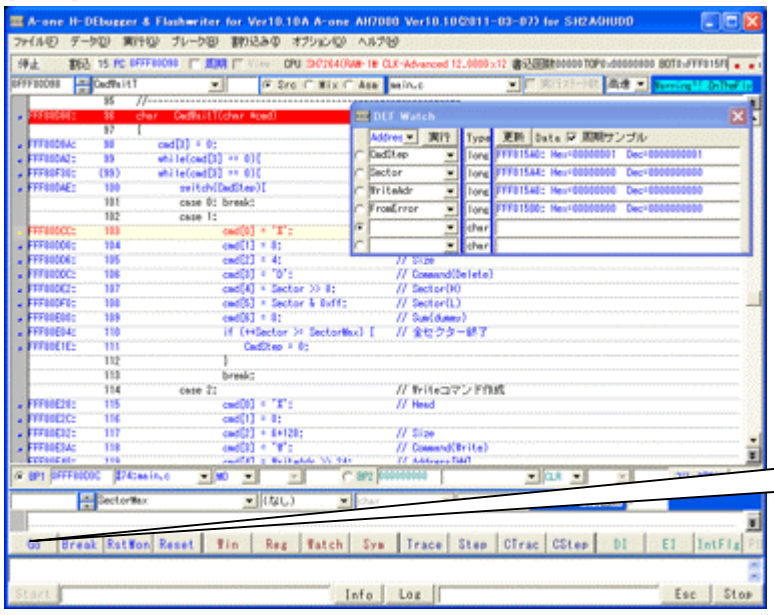
＜操作１＞
プログラム進行を確認するため、「周期サンプル」をチェックします。

＜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。

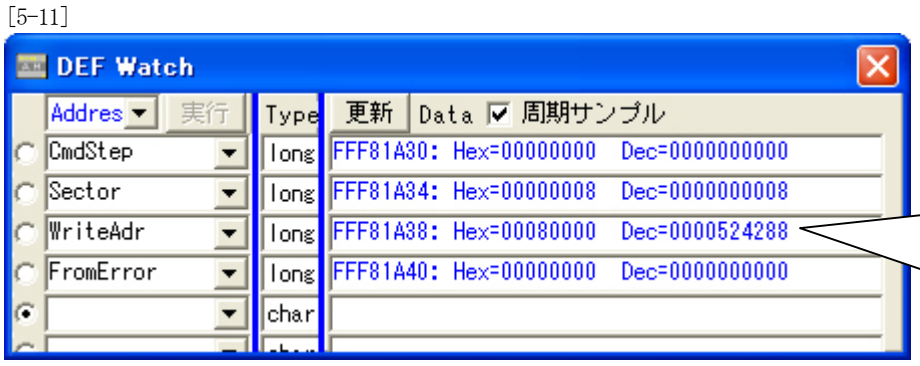
全て正常終了しますと、全エリア書き込み後、変数「CmdStep」が数値「0」になります。

[5-7-1]



＜操作３＞
「Go」ショート PB をクリックします。

[5-11]



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。

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)」

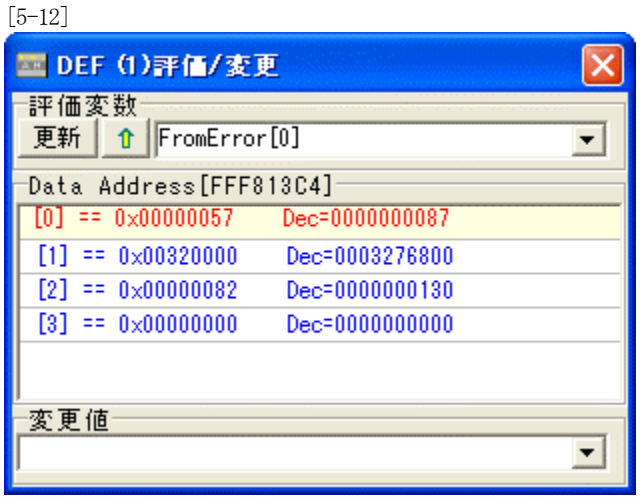
テスト用メイン関数「TestMain()」の 60 行でコール(Call)しています。

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

[5-12]

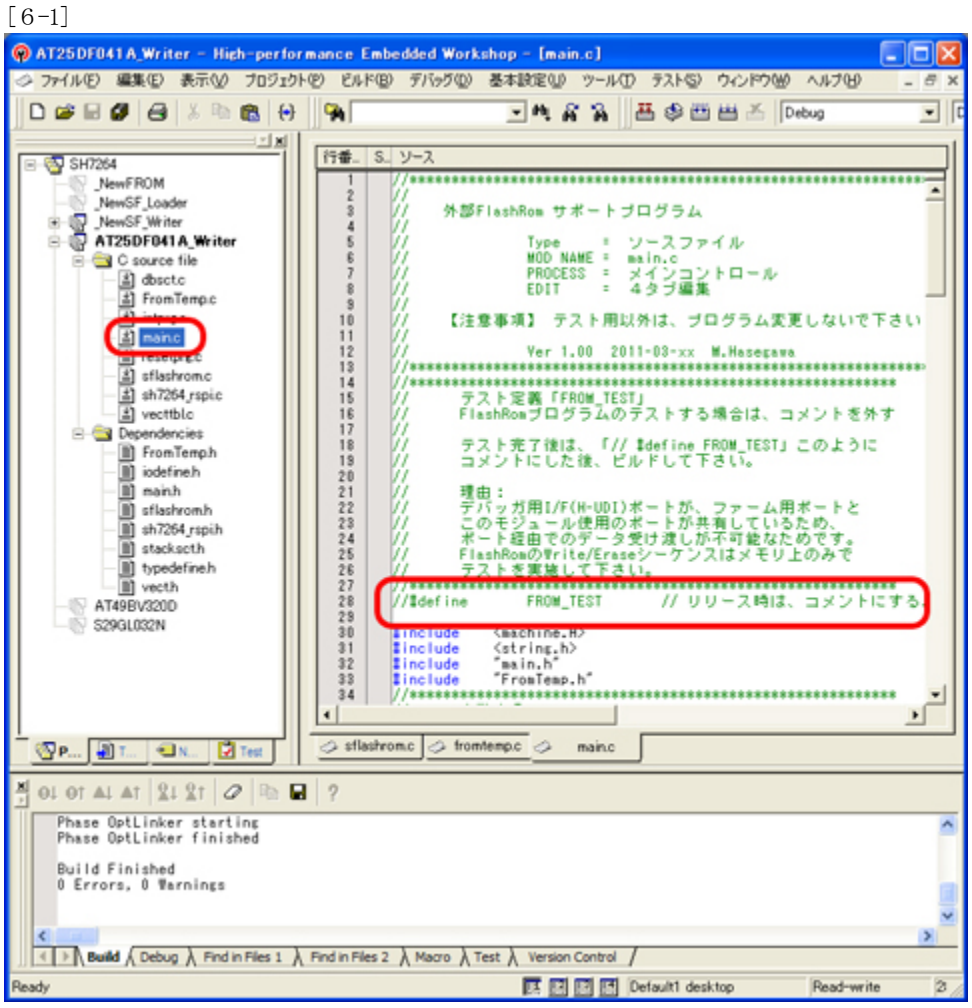


<評価・変更窓>

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。

【作成したシリアルＦｌａｓｈROMライタプログラムを正規リリース登録する】

1．シリアルＦｌａｓｈROMライタ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。
［6-1］



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＞－＜ビルド＞を指示
します。

3）「0 Errors 0 Warning」を確認し
ます。

２．御願い

本説の方法で、フラッシュROMライタプログラムを追加した場合、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、シリアルＦｌａｓｈROMライタプログラムの追加作業が終了です。