Rev 1.00.00

ROM レス品種にて、外付けシリアル FlashROM 品種を追加する場合の説明

（パケット通信対応版）

1．対象 MCU

・ROM レス品種が対象です。(SH7262 / 7264 / 7266 / 7267 / 7268 /7269）

2．機能

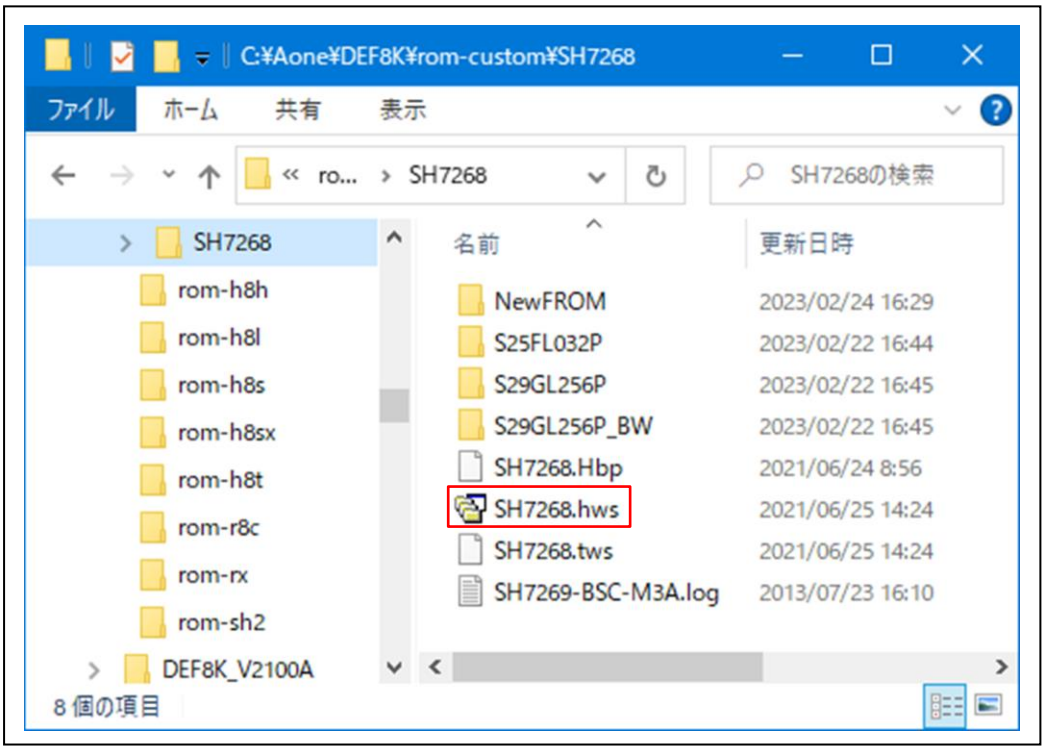
・Hew4 にてシリアル FlashROM の品種追加が出来るよう対応する。

・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るよう対応する。

・FlashROM 対応ソフトをターゲットの内臓 RAM に流し込むために、H-debugger と通信させるポートを MCU 品種ごとに用意する。

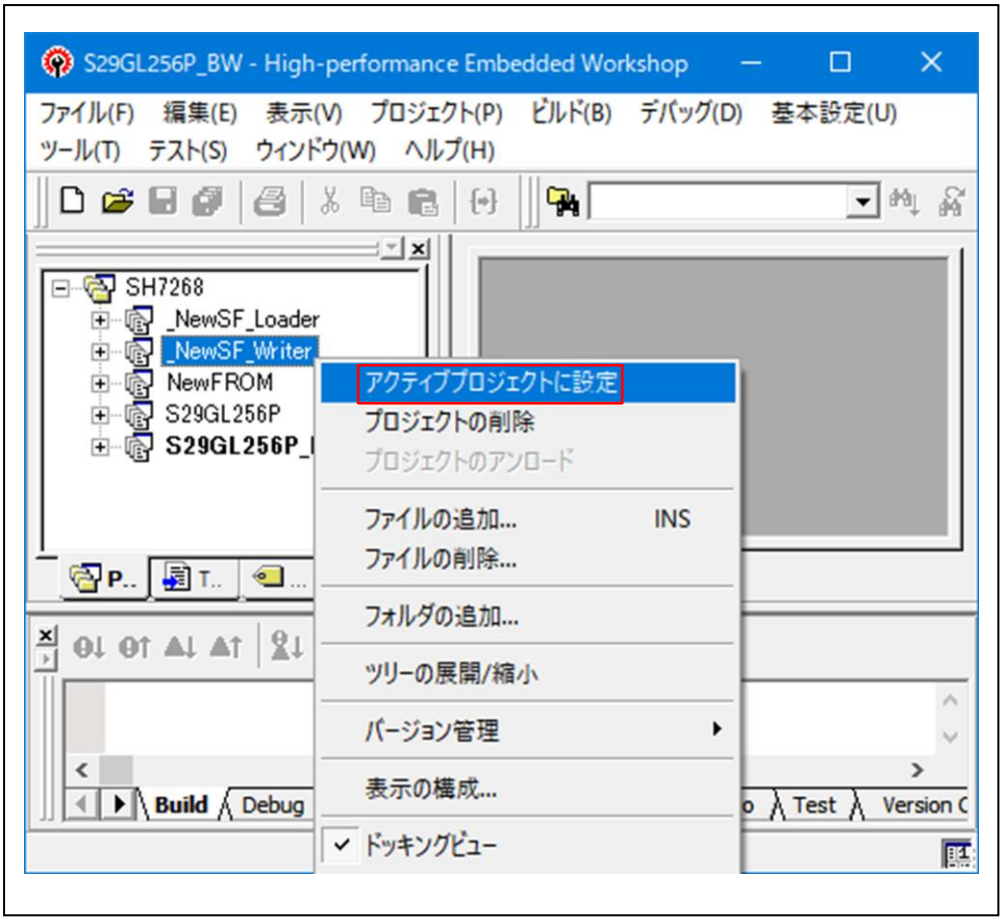
3．品種追加前の準備

３－１）AH8000 コントロールソフト(DEF8K)のインストール DIR にあるワークスペースを開く。
(SH7268 での追加例)

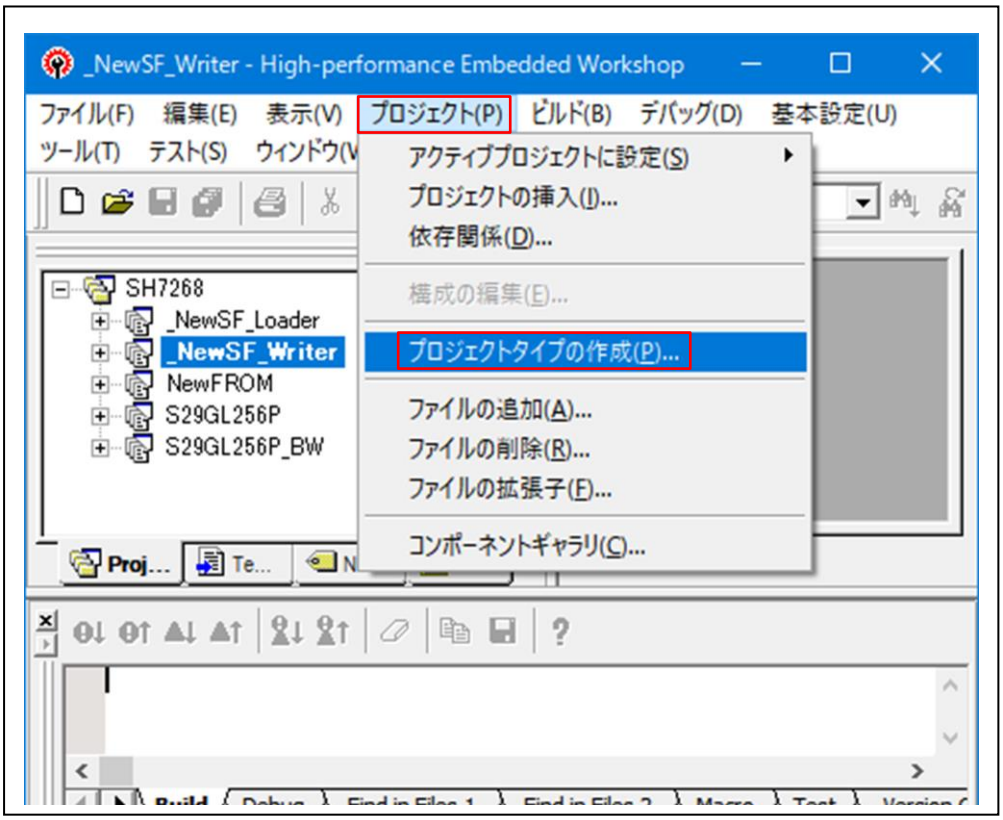


<default directory>
"c:¥Aone¥DEF8K¥rom-custom¥SH7268"

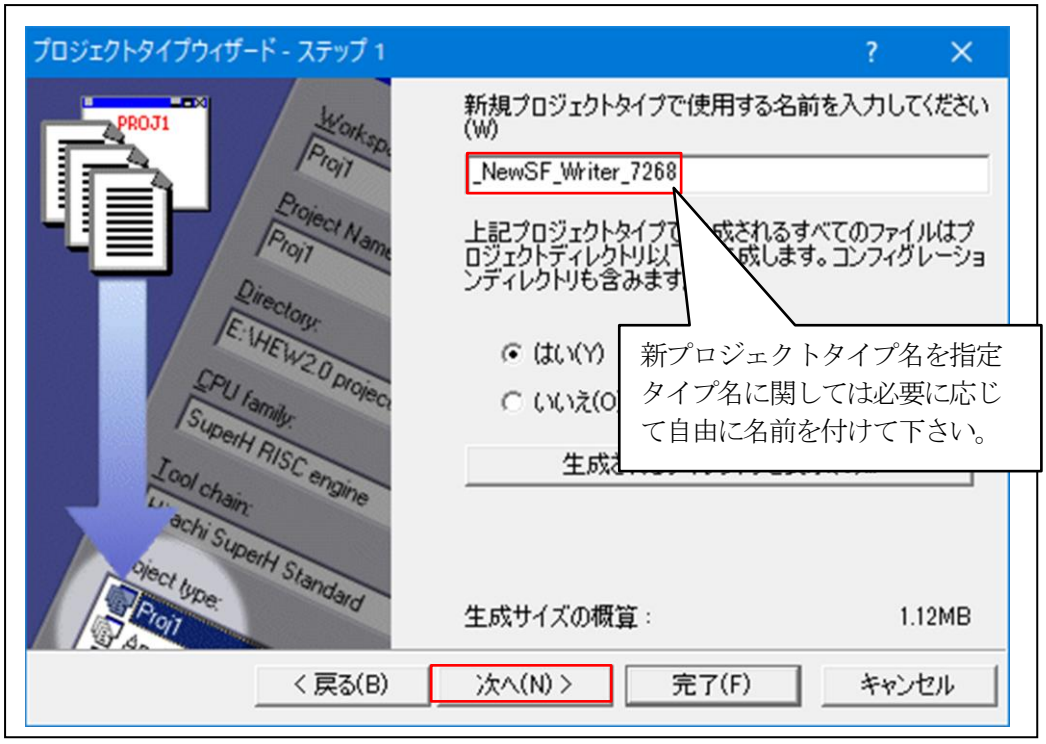
３－２）プロジェクト名「_NewSF_Writer」をアクティブプロジェクトにする。

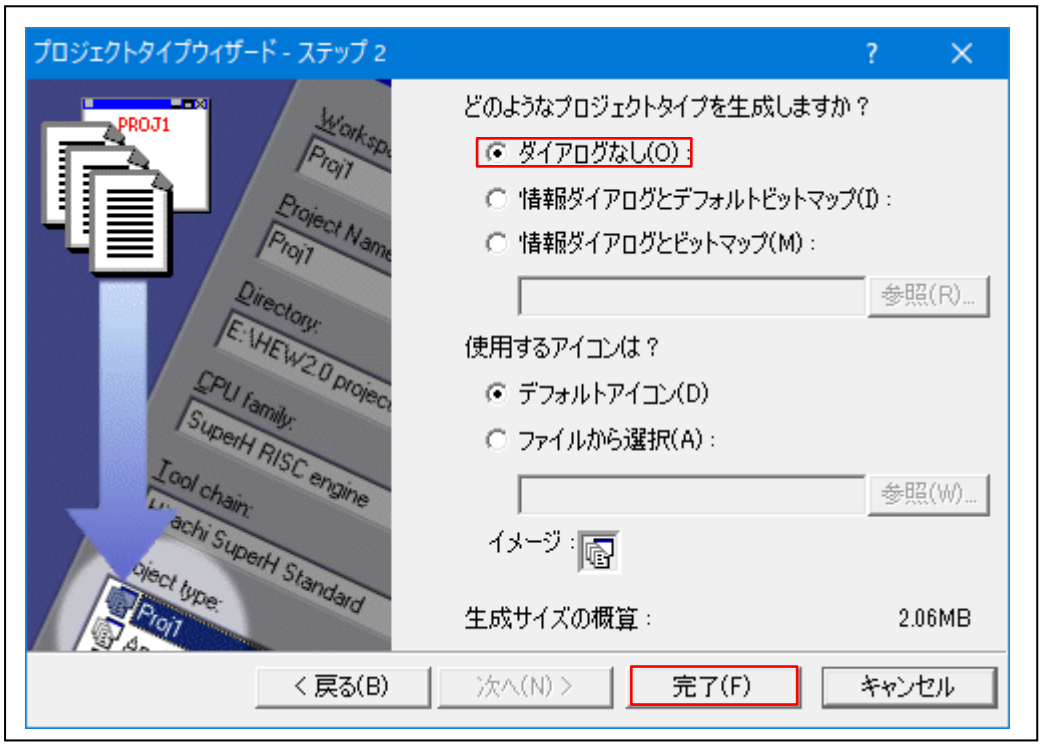


３－３）「プロジェクトタイプの作成」を指定する。



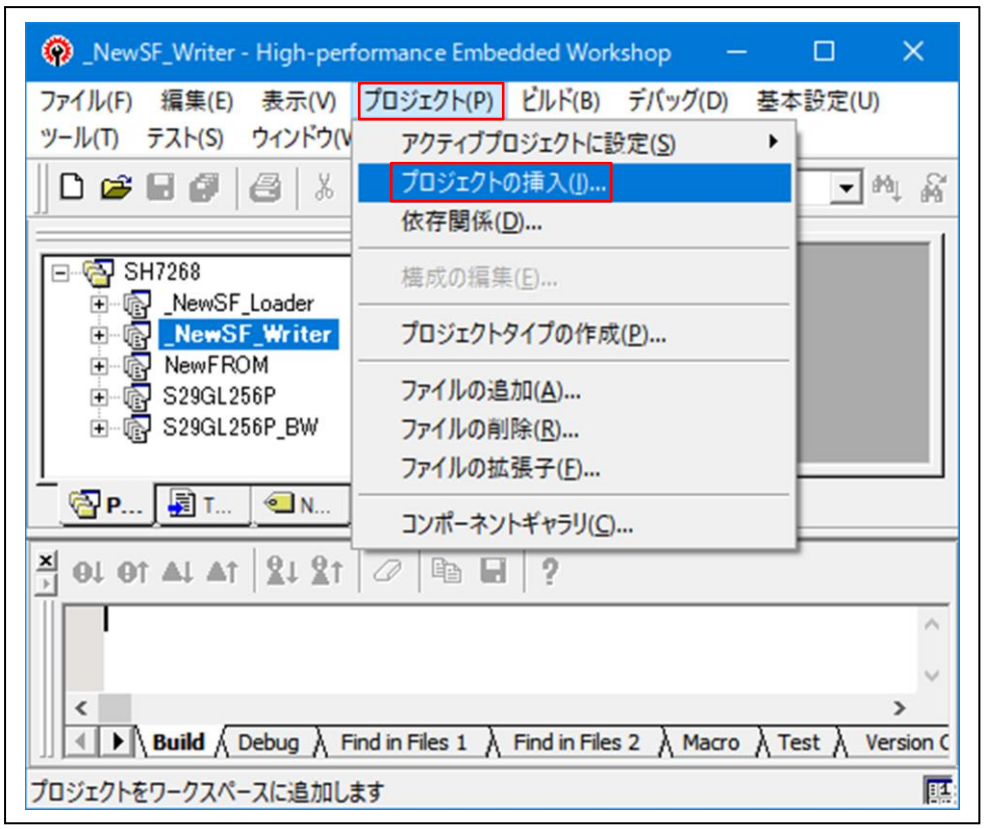
３－４）追加 FlashROM の新プロジェクトタイプを作成する。

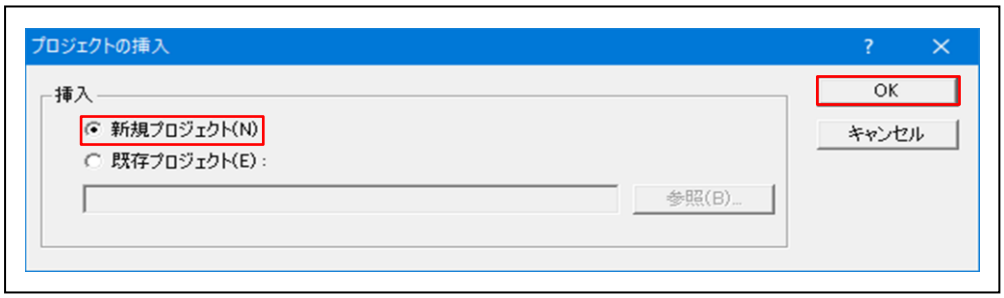


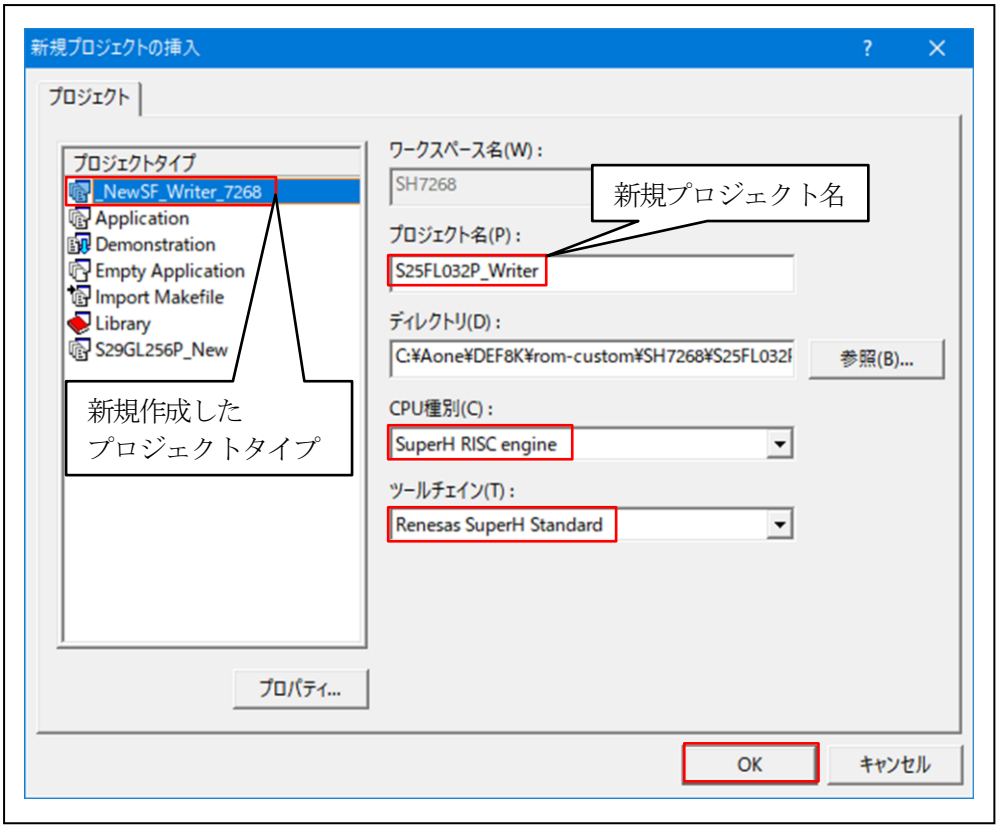


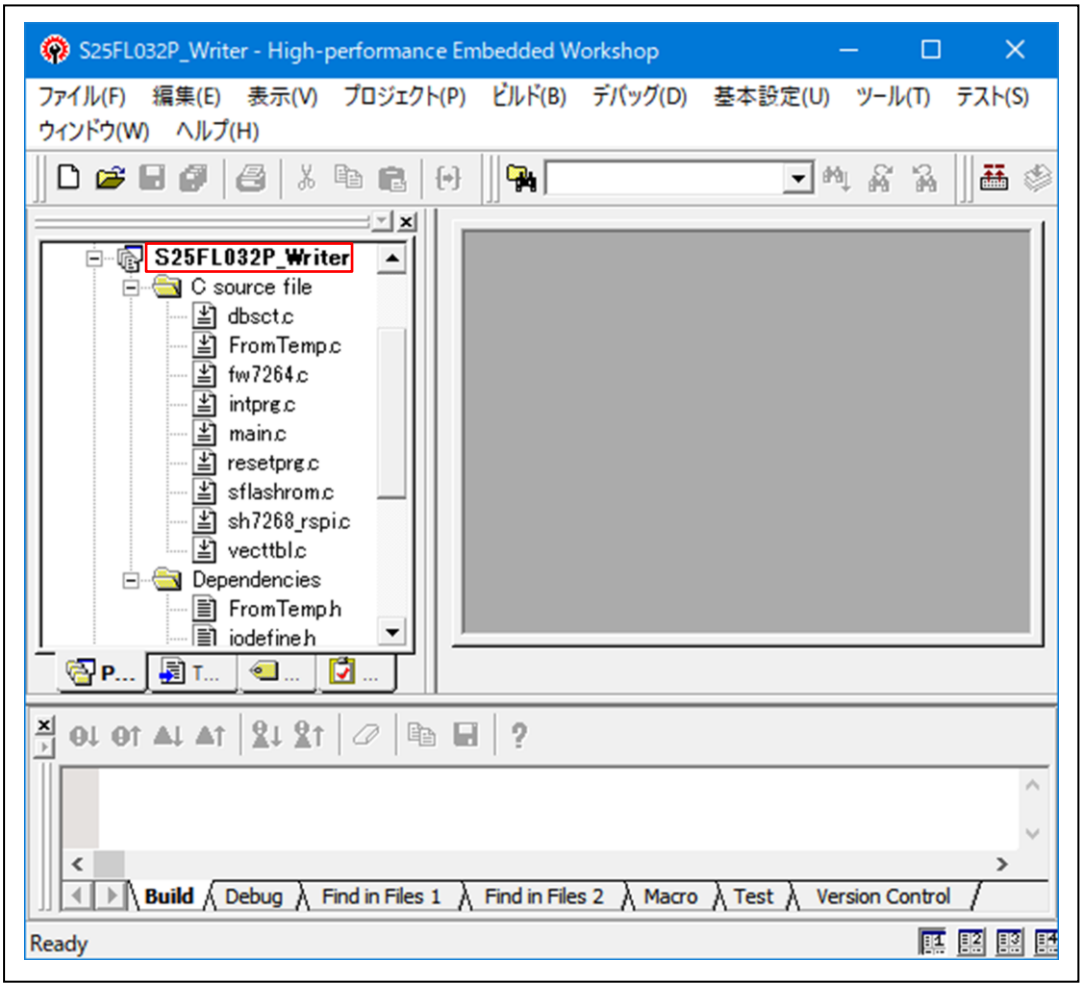
４．シリアル FlashROM 品種の追加

４－１）新規プロジェクトの挿入



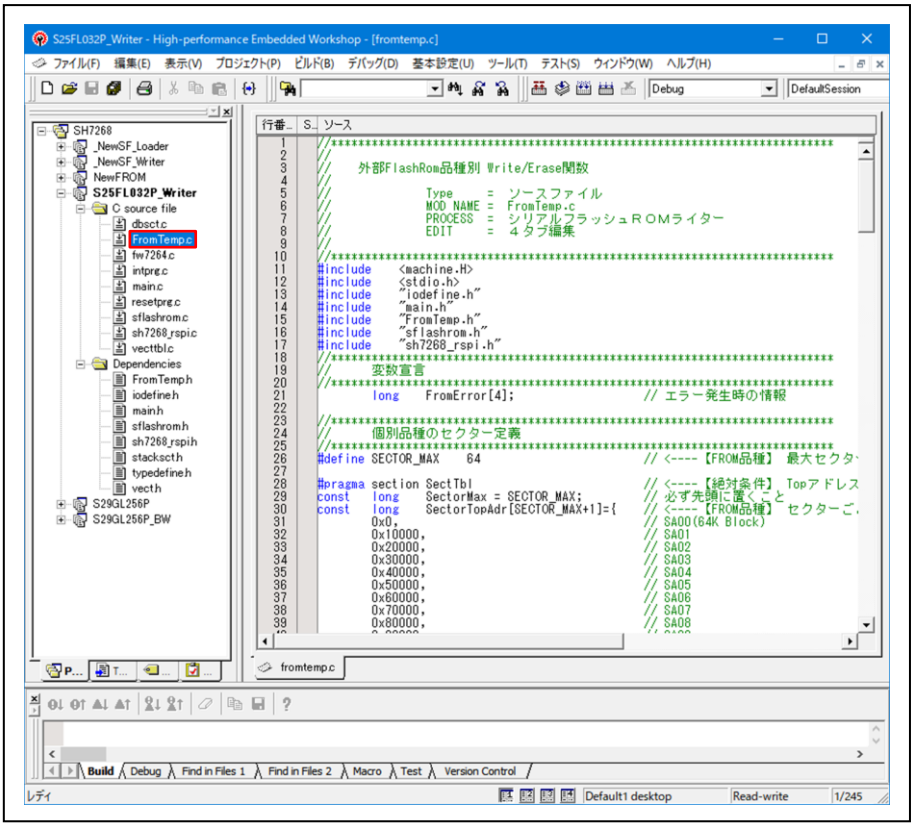



新規プロジェクトの挿入
プロジェクト
プロジェクトタイプ
NewSF_Writer_7268
Application
Demonstration
Empty Application
Import Makefile
Library
S29GL256P_New
新規作成した
プロジェクトタイプ
ワークスペース名(W):
SH7268
新規プロジェクト名
プロジェクト名(P):
S25FL032P_Writer
ディレクトリ(D):
C:¥Aone¥DEF8K¥rom-custom¥SH7268¥S25FL032
参照(B)...
CPU種別(C):
SuperH RISC engine
ツールチェイン(T):
Renesas SuperH Standard
プロパティ...
OK
キャンセル

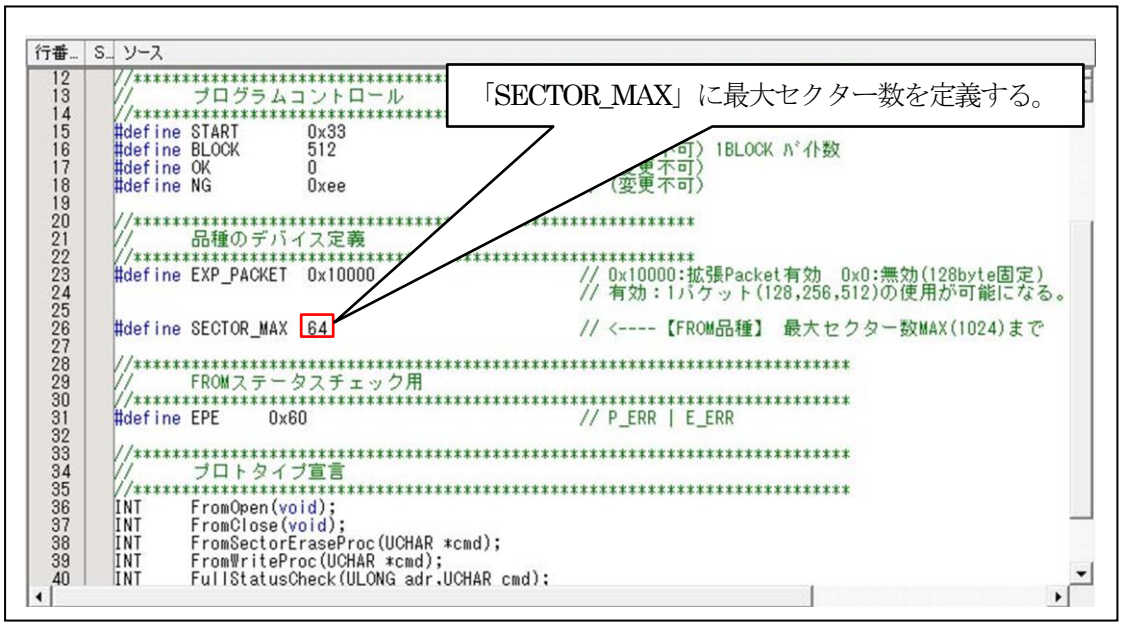

S25FL032P_Writer - High-performance Embedded Workshop
ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) 基本設定(U) ツール(T) テスト(S)
ウィンドウ(W) ヘルプ(H)
S25FL032P_Writer
C source file
dbsct.c
FromTemp.c
fw7264.c
intprg.c
main.c
resetprg.c
sflashrom.c
sh7268_rspi.c
vecttbl.c
Dependencies
FromTemp.h
iodefine.h
Build
Debug
Find in Files 1
Find in Files 2
Macro
Test
Version Control
Ready

４－２）シリアル FlashROM 品種の定義およびプログラムを作成する。

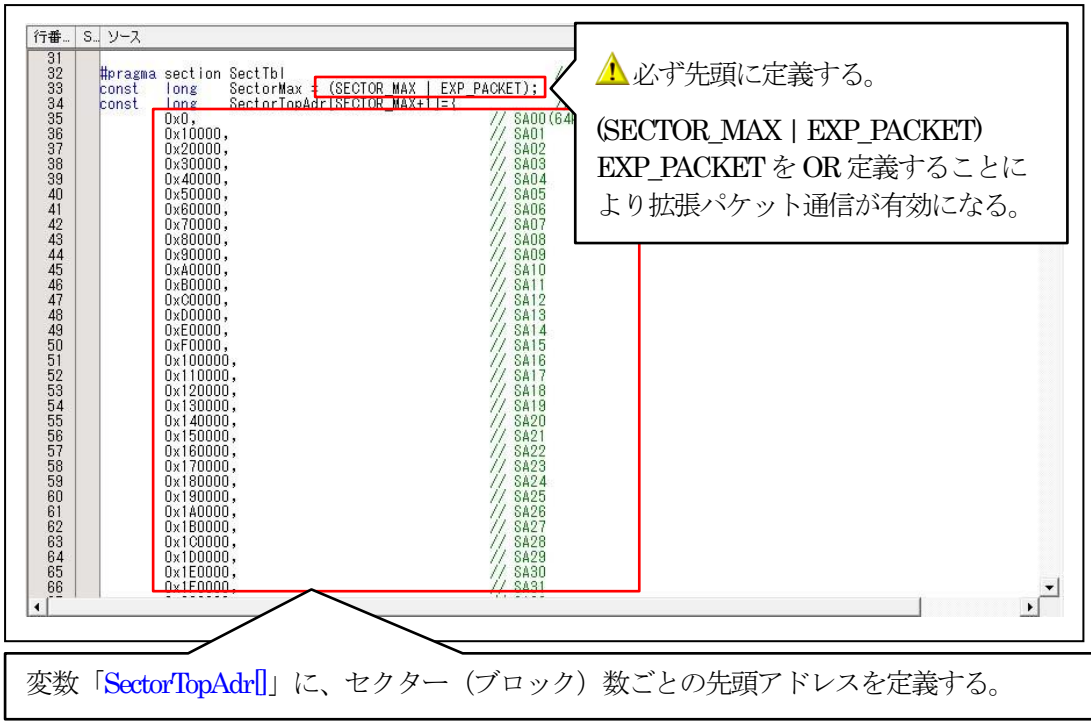
１）「FromTemp.c」を開く

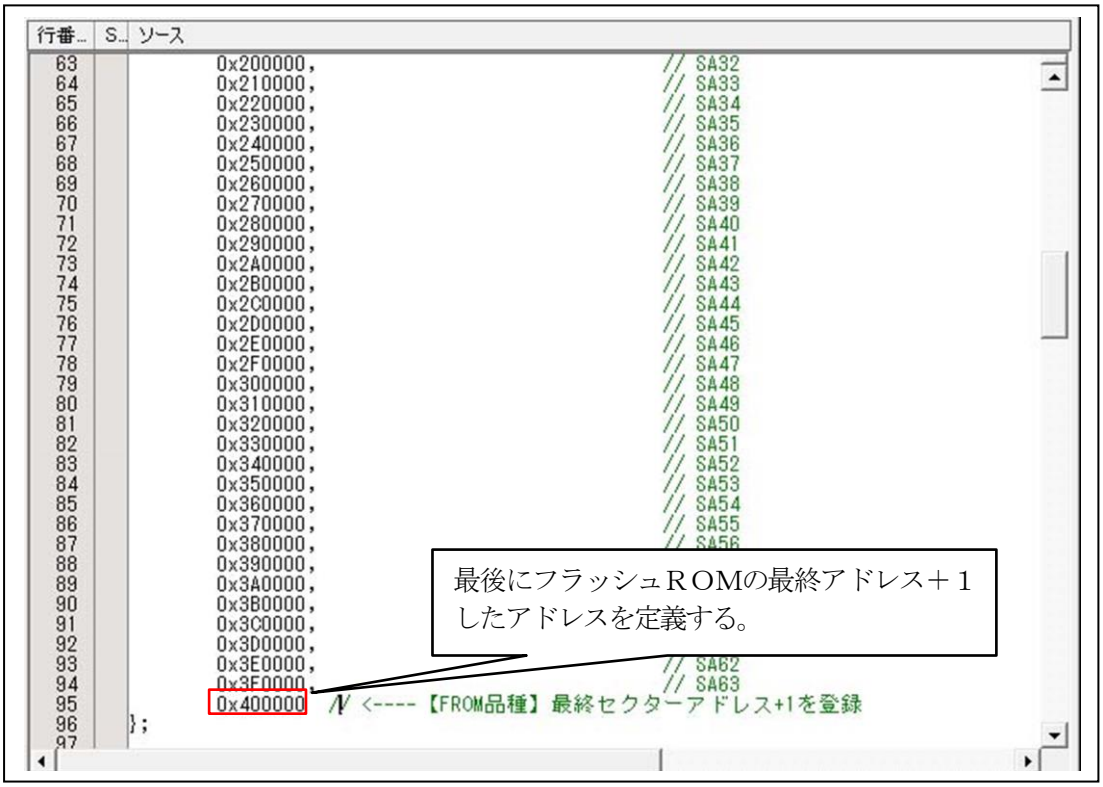


２）「FromTemp.h」の#define を定義する。



3）シリアル FlashROM のセクター情報を定義する。





4）「オープン」関数を確認または変更する。

```
行番... S... ソース
 99     //*****************************************************************************
100     //      FromOpen()
101     //      FlashRomアクセス時に初期化が必要な場合の手続き
102     //      不要な場合、プログラム記述しなくても良い
103     //      開始時に1回だけ呼ばれる
104     //      return()
105     //        OK  == 正常終了
106     //        NG  == 異常終了
107     //*****************************************************************************
108     INT     FromOpen(void)
109     {
110         CPG.FRQCR.WORD = 0x335;             // FRQCR set I=x5 B=x5 P1=x5 P0=x2.5
111         CCNT.CCR1.LONG = 0;                 // キャッシュ無効
112
113         IoInitSPIBSC();                     // RSPIの初期化
114
115         sFlashSetMode();                    // sFlashRom モードセット
116
117         sFlashUnProtect();                  // アンプロテクト
118
119         return(OK);
120     }
121     //*****************************************************************************
```

関数「FromOpen()」は、ターゲット RAM に、ダウンロード時の最初に１回コール(Call)する。このプログラム例は、
「FRQCR」の設定
「キャッシュ無効」を定義する。
「IoInitSPIBSC()」RSPI0 の初期化
「sFlashSetMode()」QUAD モード
「sFlashUnProtect()」プロテクト解除（アンロック）
必要無い場合は、「return(Ok);」のみ記述する。

5）「クローズ」関数を確認または変更する。

```
行番... S... ソース
121     //*****************************************************************************
122     //      FromClose()
123     //      FlashRom書き込み処理が全て終了した場合に必要な手続き
124     //      不要な場合、プログラム記述しなくても良い
125     //      終了時に1回だけ呼ばれる
126     //      return()
127     //        OK  == 正常終了
128     //        NG  == 異常終了
129     //*****************************************************************************
130     INT     FromClose(void)
131     {
132         sFlashProtect();                    // プロテクト
133
134         return(OK);
135     }
136     //*****************************************************************************
```

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)する。このプログラム例は
「sFlashProtect()」プロテクト（ロック）

必要無い場合は、「return(Ok);」のみ記述する。

6）「セクターイレーズ」関数を確認または変更する。

```
行番… S… ソース
136     //*****************************************************************************
137     //      FromSectorEraseProc(CHAR *cmd)
138     //      伝送フォーム(in)
139     //        cmd[0]   == %
140     //            [1]   == 0
141     //            [2]   == CommandSize[2]以降[Sum]も含む
142     //            [3]   == Command('D')
143     //            [4]   == Sector(H) MAX(1024)
144     //            [5]   == Sector(L)
145     //            [Sum] == [3]->[Sum-1]の加算
146     //      return()
147     //        OK  == 正常終了
148     //        NG  == 異常終了
149     //*****************************************************************************
150     INT     FromSectorEraseProc(UCHAR *cmd)
151     {
152         ULONG  adr;                                 // セクターの先頭アドレス
153         USHORT sector;
154         INT    stat;
155
156         sector  = (cmd[4] << 8) & 0xff00;
157         sector |= cmd[5] & 0xff;
158         if (sector < SectorMax) {
159             adr = SectorTopAdr[sector];
160             sFlashSectorErase(adr);                 // セクターイレーズ
161             stat = FullStatusCheck(adr,'D');
162             if (stat == OK) return(OK);
163         }
164         return(NG);
165     }
166     //*****************************************************************************
```

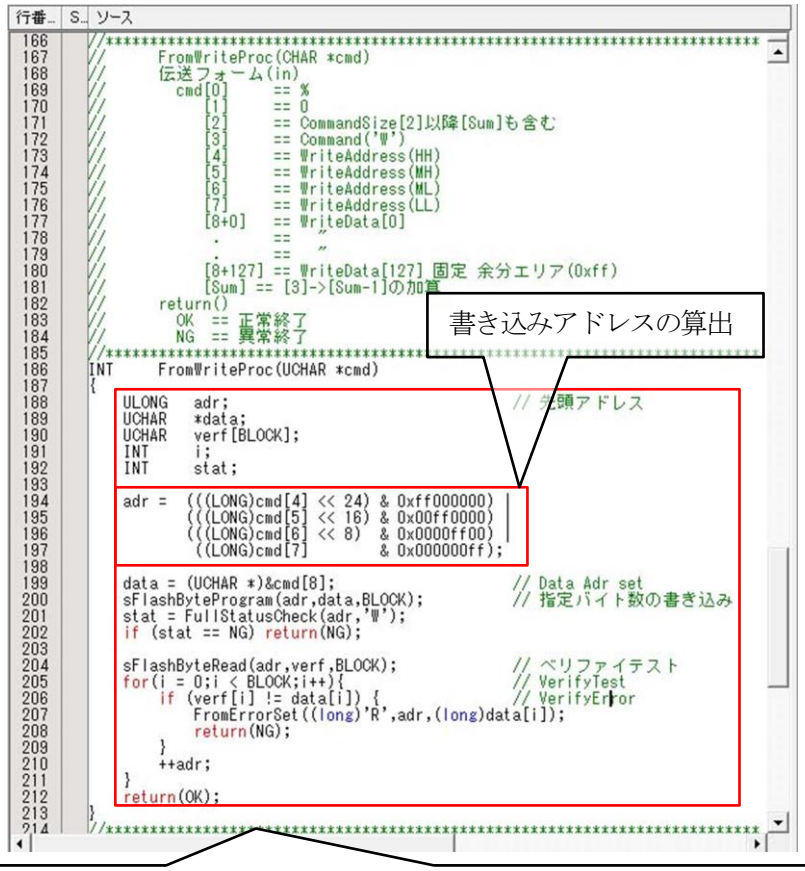
関数「FromSectorEraseProc(UCHAR *cmd)」には、セクターごとのイレーズプログラムを記述する。デバッガとの通信完了後、サムチェック正常の場合、この関数をコール(Call)する。（サム異常時はコールなし）

＜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);」にすれば代用できる。

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

7）「書き込みプログラム」関数を確認または変更する。




```
//*****************************************************************************
//      FromWriteProc(CHAR *cmd)
//      伝送フォーム(in)
//        cmd[0]      == %
//            [1]     == 0
//            [2]     == CommandSize[2]以降[Sum]も含む
//            [3]     == Command('W')
//            [4]     == WriteAddress(HH)
//            [5]     == WriteAddress(MH)
//            [6]     == WriteAddress(ML)
//            [7]     == WriteAddress(LL)
//            [8+0]   == WriteData[0]
//             .      ==  〃
//             .      ==  〃
//          [8+127] == WriteData[127] 固定 余分エリア(0xff)
//          [Sum] == [3]->[Sum-1]の加算
//      return()
//        OK  == 正常終了
//        NG  == 異常終了
//*****************************************************************************
INT     FromWriteProc(UCHAR *cmd)
{
    ULONG   adr;                                // 先頭アドレス
    UCHAR   *data;
    UCHAR   verf[BLOCK];
    INT     i;
    INT     stat;

    adr =  (((LONG)cmd[4] << 24) & 0xff000000) |
           (((LONG)cmd[5] << 16) & 0x00ff0000) |
           (((LONG)cmd[6] << 8)  & 0x0000ff00) |
            ((LONG)cmd[7]        & 0x000000ff);

    data = (UCHAR *)&cmd[8];                    // Data Adr set
    sFlashByteProgram(adr,data,BLOCK);          // 指定バイト数の書き込み
    stat = FullStatusCheck(adr,'W');
    if (stat == NG) return(NG);

    sFlashByteRead(adr,verf,BLOCK);             // ベリファイテスト
    for(i = 0;i < BLOCK;i++){                   // VerifyTest
        if (verf[i] != data[i]) {               // VerifyError
            FromErrorSet((long)'R',adr,(long)data[i]);
            return(NG);
        }
        ++adr;
    }
    return(OK);
}
//*****************************************************************************
```

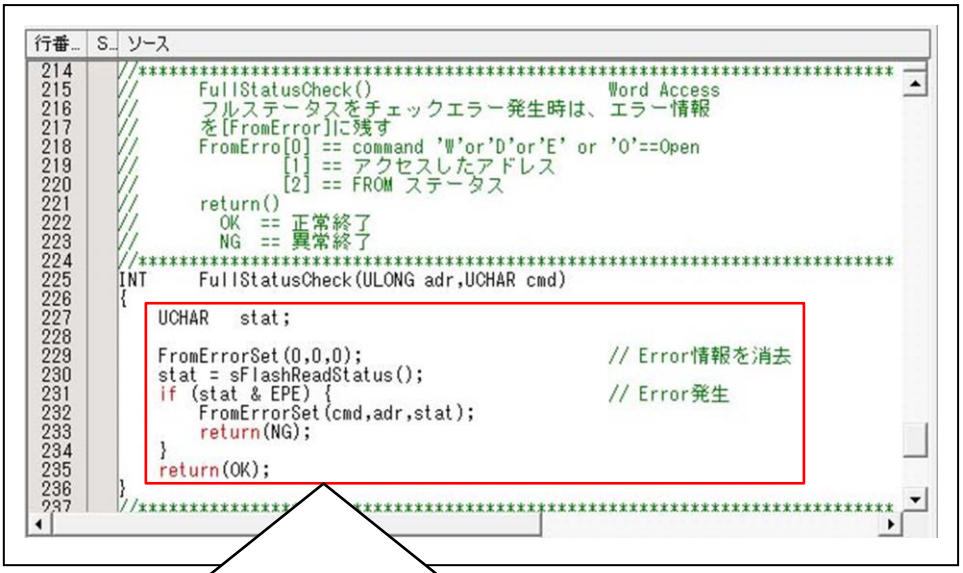
関数「FromWriteProc(UCHAR *cmd)」には、128 バイトごとの書き込みプログラムを記述する。デバッガから書き込みデータ 128 バイト)固定でデータを受信し完了後、サムチェック正常の場合、この関数をコール(Call)する。（サム異常時はコールなし）

＜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、ベリファイテストを実施している。

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

8）「フルステータスチェック」関数を確認または変更する。



```
行番… S… ソース
214  //*****************************************************************************
215  //      FullStatusCheck()                          Word Access
216  //      フルステータスをチェックエラー発生時は、エラー情報
217  //      を[FromError]に残す
218  //      FromErro[0] == command 'W'or'D'or'E' or 'O'==Open
219  //                  [1] == アクセスしたアドレス
220  //                  [2] == FROM ステータス
221  //      return()
222  //        OK  == 正常終了
223  //        NG  == 異常終了
224  //*****************************************************************************
225  INT     FullStatusCheck(ULONG adr,UCHAR cmd)
226  {
227      UCHAR   stat;
228
229      FromErrorSet(0,0,0);                        // Error情報を消去
230      stat = sFlashReadStatus();
231      if (stat & EPE) {                           // Error発生
232          FromErrorSet(cmd,adr,stat);
233          return(NG);
234      }
235      return(OK);
236  }
237  //*****************************************************************************
```

関数「FullStatusCheck(USHORT *adr,USHORT data,UCHAR 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する。この関数は、上記の作成関数からのみコール(Call)される。よって、内部処理は自由に記述可能です。

＜引数の仕様＞

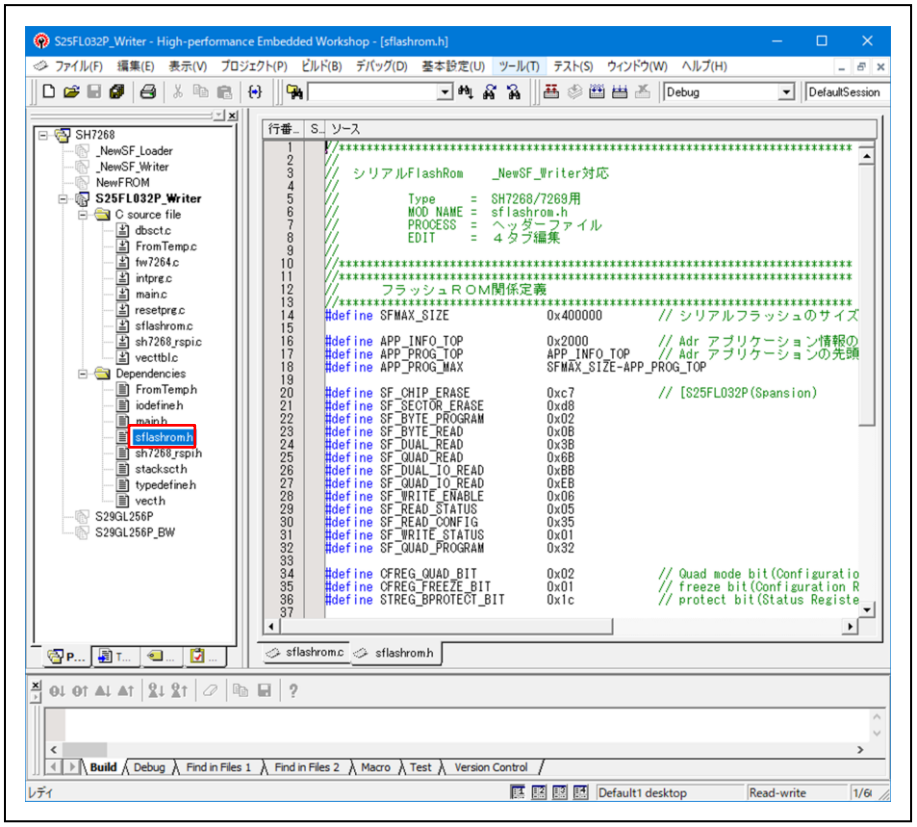
Ushort *adr;　//処理しているフラッシュROMのアドレス
char　cmd;　//処理しているコマンド
　　//"O"＝オープンコマンド
　　//"C"＝クローズコマンド
　　//"D"＝イレーズコマンド
　　//"W"＝書き込みコマンド
　　//"R"＝ベリファイコマンド

＜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のため、仕様に関してはルールが無いので、自由に作成して下さい。

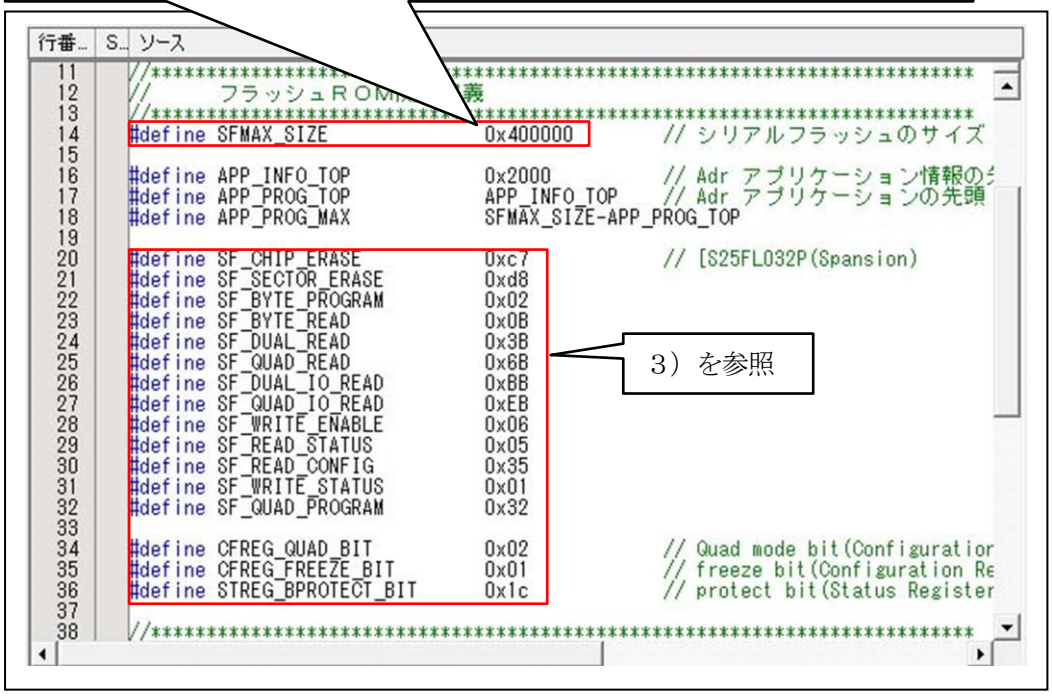
４－３）シリアル FlashROM に依存した定義およびコマンドプログラムを作成する。

１）「sflashrom.h」を開く



２）「sflashrom.h」の#define を定義する。

「SFMAX_SIZE」にシリアル FlashROM の最大サイズ（バイト）を定義する。



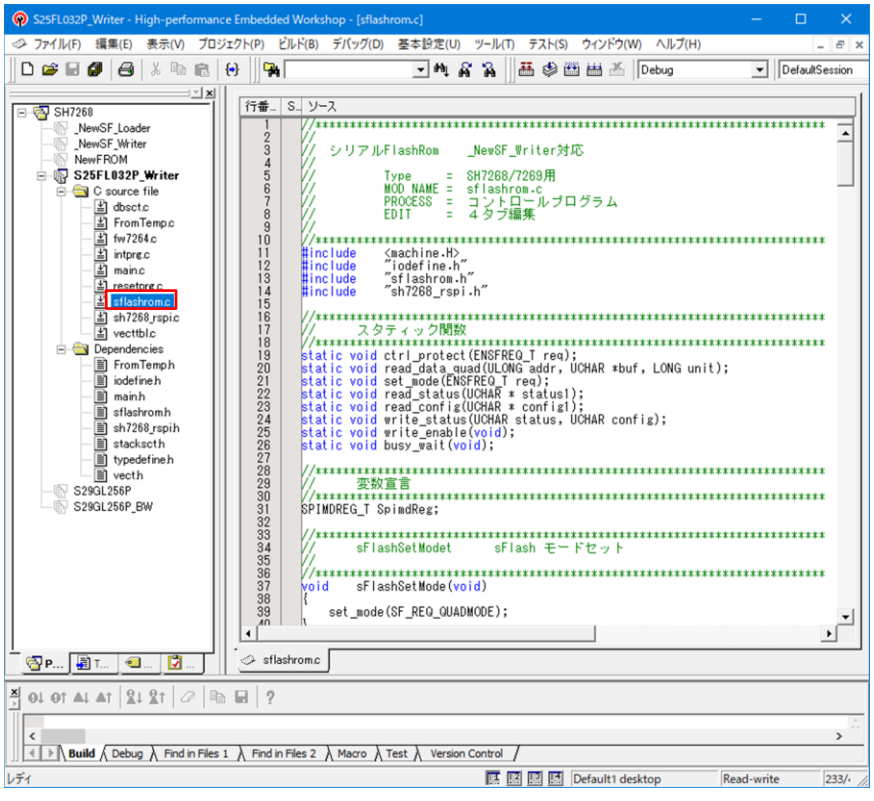
3）「sflashrom.h」の#define の定義説明。

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_CHIP_ERASE	0xC7	チップイレーズ
SF_SECTOR_ERASE	0xD8	セクターイレーズ
SF_BYTE_PROGRAM	0x02	データの書き込み（バイト）
SF_BYTE_READ	0x0B	データの読み出し（バイト）
SF_DUAL_READ	0x3B	データの読み出し（DUAL）
SF_QUAD_READ	0x6B	データの読み出し（QUAD）
SF_DUAL_IO_READ	0xBB	DUAL I/O 高性能読み取り
SF_QUAD_IO_READ	0xEB	QUAD I/O 高性能読み取り
SF_WRITE_ENABLE	0x06	Write 許可
SF_READ_STATUS	0x05	ステータスリード
SF_READ_CONFIG	0x35	コンフィグレーションリード
SF_WRITE_STATUS	0x01	ステータスライト
SF_QUAD_PROGRAM	0x32	データの書き込み（QUAD）
CFREG_QUAD_BIT	0x02	QUAD モードビット（Configuration）
CFREG_FREEZE_BIT	0x01	フリーズビット（Configuration）
STREG_BPROTECT_BIT	0x1C	プロテクトビット（Status）

この例は、Spansion の「S25FL032P」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。

4）「sflashrom.c」を開く



5）「sFlashSetMode()」開数を確認または変更する。

【機能】QUAD モードに設定する。

```
行番… S… ソース
33      //*****************************************************************************
34      //      sFlashSetModet      sFlash モードセット
35      //
36      //*****************************************************************************
37      void    sFlashSetMode(void)
38      {
39          set_mode(SF_REQ_QUADMODE);
40      }
41      //*****************************************************************************
```

6）「sFlashUnProtect()」開数を確認または変更する。

【機能】プロテクトを解除する。

```
行番… S… ソース
41      //*****************************************************************************
42      //      sFlashUnProtect     アンプロテクト
43      //
44      //*****************************************************************************
45      void    sFlashUnProtect(void)
46      {
47          ctrl_protect(SF_REQ_UNPROTECT);
48      }
49      //*****************************************************************************
```

7）「sFlashProtect()」開数を確認または変更する。

【機能】プロテクトする。

```
行番… S… ソース
49      //*****************************************************************************
50      //      sFlashProtect       プロテクト
51      //
52      //*****************************************************************************
53      void    sFlashProtect(void)
54      {
55          ctrl_protect(SF_REQ_PROTECT);
56      }
57      //*****************************************************************************
```

8）「sFlashReadStatus()」開数を確認または変更する。

【機能】ステータスをリードする。

```
行番… S… ソース
196     //*****************************************************************************
197     //      sFlashReadStatus    sFlashROMステータスRead
198     //
199     //*****************************************************************************
200     UCHAR   sFlashReadStatus(void)
201     {
202         UCHAR   stat;
203
204         read_status(&stat);
205
206         return(stat);
207     }
```

9）「sFlashChipErase()」開数を確認または変更する。

【機能】チップをイレーズする。

```
行番... S... ソース
 57    //*****************************************************************************
 58    //      sFlashChipErase     チップイレーズ
 59    //
 60    //*****************************************************************************
 61    void    sFlashChipErase(void)
 62    {
 63        write_enable();                             // WREN Command
 64
 65        SpimdReg.cdb   = SPIBSC_1BIT;               // Commmand bit-width = Single
 66        SpimdReg.cde   = SPIBSC_OUTPUT_ENABLE;      // Command Enable
 67        SpimdReg.ocde  = SPIBSC_OUTPUT_DISABLE;     // Optional-Command Disable
 68        SpimdReg.ade   = SPIBSC_OUTPUT_DISABLE;     // Address Disable
 69        SpimdReg.opde  = SPIBSC_OUTPUT_DISABLE;     // Option-Data Disable
 70        SpimdReg.spide = SPIBSC_OUTPUT_DISABLE;     // Disable
 71        SpimdReg.sslkp = SPIBSC_SPISSL_NEGATE;      // Negate after transfer
 72        SpimdReg.spire = SPIBSC_SPIDATA_DISABLE;    // Data Access (Read Disable)
 73        SpimdReg.spiwe = SPIBSC_SPIDATA_DISABLE;    // Data Access (Write Disable)
 74        SpimdReg.cmd   = SF_CHIP_ERASE;             // BE:Bulk Erase
 75
 76        IoTransferSPIBSC(&SpimdReg);
 77
 78        busy_wait();
 79    }
 80    //*****************************************************************************
```

１０）「sFlashSectorErase()」開数を確認または変更する。

【機能】セクターをイレーズする。

```
行番... S... ソース
 80    //*****************************************************************************
 81    //      sFlashSectorErase   セクターイレーズ
 82    //
 83    //*****************************************************************************
 84    void    sFlashSectorErase(ULONG adr)
 85    {
 86        write_enable();                             // WREN Command
 87
 88        SpimdReg.cdb   = SPIBSC_1BIT;               // Commmand bit-width = Single
 89        SpimdReg.adb   = SPIBSC_1BIT;               // Address bit-width = Single
 90        SpimdReg.cde   = SPIBSC_OUTPUT_ENABLE;      // Command Enable
 91        SpimdReg.ocde  = SPIBSC_OUTPUT_DISABLE;     // Optional-Command Disable
 92        SpimdReg.ade   = SPIBSC_OUTPUT_ADDR_24;     // Enable(Adr[23:0])
 93        SpimdReg.opde  = SPIBSC_OUTPUT_DISABLE;     // Option-Data Disable
 94        SpimdReg.spide = SPIBSC_OUTPUT_DISABLE;     // Disable
 95        SpimdReg.sslkp = SPIBSC_SPISSL_NEGATE;      // Negate after transfer
 96        SpimdReg.spire = SPIBSC_SPIDATA_DISABLE;    // Data Access (Read Disable)
 97        SpimdReg.spiwe = SPIBSC_SPIDATA_DISABLE;    // Data Access (Write Disable)
 98        SpimdReg.cmd   = SF_SECTOR_ERASE;           // SE:Sector Erase
 99        SpimdReg.addr  = adr;                       // dont care in dual mode
100                                                    // because address is calcurat
101        IoTransferSPIBSC(&SpimdReg);
102
103        busy_wait();
104    }
105    //*****************************************************************************
```

１１）「sFlashByteProgram()」開数を確認または変更する。

【機能】指定バイトを QUAD モードで書き込み（プログラミング）する。

```
行番… S… ソース
105     //*****************************************************************************
106     //      sFlashByteProgram   指定バイト数の書き込み
107     //*****************************************************************************
108     void    sFlashByteProgram(ULONG adr,UCHAR *wdat,LONG wsize)
109     {
110         LONG unit;
111
112         write_enable();                                 // WREN Command
113         SpimdReg.cdb   = SPIBSC_1BIT;                   // Commmand bit-width = Single
114         SpimdReg.adb   = SPIBSC_1BIT;                   // Address bit-width = Single
115         SpimdReg.cde   = SPIBSC_OUTPUT_ENABLE;          // Command Enable
116         SpimdReg.ocde  = SPIBSC_OUTPUT_DISABLE;         // Optional-Command Disable
117         SpimdReg.ade   = SPIBSC_OUTPUT_ADDR_24;         // Enable Adr[23:0]
118         SpimdReg.opde  = SPIBSC_OUTPUT_DISABLE;         // Option-Data Disable
119         SpimdReg.spide = SPIBSC_OUTPUT_DISABLE;         // Disable
120         SpimdReg.sslkp = SPIBSC_SPISSL_KEEP;            // Keep after transfer
121         SpimdReg.spire = SPIBSC_SPIDATA_DISABLE;        // Data Access (Read Disable)
122         SpimdReg.spiwe = SPIBSC_SPIDATA_DISABLE;        // Data Access (Write Disable)
123
124         SpimdReg.cmd   = SF_QUAD_PROGRAM;               // QPP: Quad Page Program
125         SpimdReg.addr  = adr;
126         IoTransferSPIBSC(&SpimdReg);                    // Command,Address
127                                                         // ---- Data ----
128         SpimdReg.spidb = SPIBSC_4BIT;                   // Quad
129         SpimdReg.cde   = SPIBSC_OUTPUT_DISABLE;         // Command Disable
130         SpimdReg.ocde  = SPIBSC_OUTPUT_DISABLE;         // Optional-Command Disable
131         SpimdReg.ade   = SPIBSC_OUTPUT_DISABLE;         // Disable Adr
132         SpimdReg.opde  = SPIBSC_OUTPUT_DISABLE;         // Option-Data Disable
133         SpimdReg.spire = SPIBSC_SPIDATA_DISABLE;        // Data Access (Read Disable)
134         SpimdReg.spiwe = SPIBSC_SPIDATA_ENABLE;         // Data Access (Write Ensable)
135
136         if((wsize % 4) == 0){                           // Enable(32bit)
137             SpimdReg.spide = SPIBSC_OUTPUT_SPID_32;
138             unit = 4;
139         } else if((wsize % 2) == 0){                        // Enable(16bit)
140             SpimdReg.spide = SPIBSC_OUTPUT_SPID_16;
141             unit = 2;
142         } else {                                            // Enable(8bit)
143             SpimdReg.spide = SPIBSC_OUTPUT_SPID_8;
144             unit = 1;
145         }
146         while(wsize > 0) {
147             if (unit >= 1) {                            // Data[31:24]
148                 SpimdReg.smwdr[0] = (ULONG)(((ULONG)(*wdat) << 24) & 0xff000000);
149                 wdat++;
150             }
151             if (unit >= 2) {                            // Data[23:16]
152                 SpimdReg.smwdr[0] |= (ULONG)(((ULONG)(*wdat) << 16) & 0x00ff0000);
153                 wdat++;
154             }
155             if (unit >= 4) {                            // Data[15:0]
156                 SpimdReg.smwdr[0] |= (ULONG)(((ULONG)(*wdat) <<  8) & 0x0000ff00);
157                 wdat++;
158                 SpimdReg.smwdr[0] |= (ULONG)(((ULONG)(*wdat)      ) & 0x000000ff);
159                 wdat++;
160             }
161             wsize -= unit;
162
163             if (wsize <= 0) {
164                 SpimdReg.sslkp = SPIBSC_SPISSL_NEGATE;
165             }
166             IoTransferSPIBSC(&SpimdReg);                // Data
167         }
168
169         busy_wait();
170     }
171     //*****************************************************************************
```

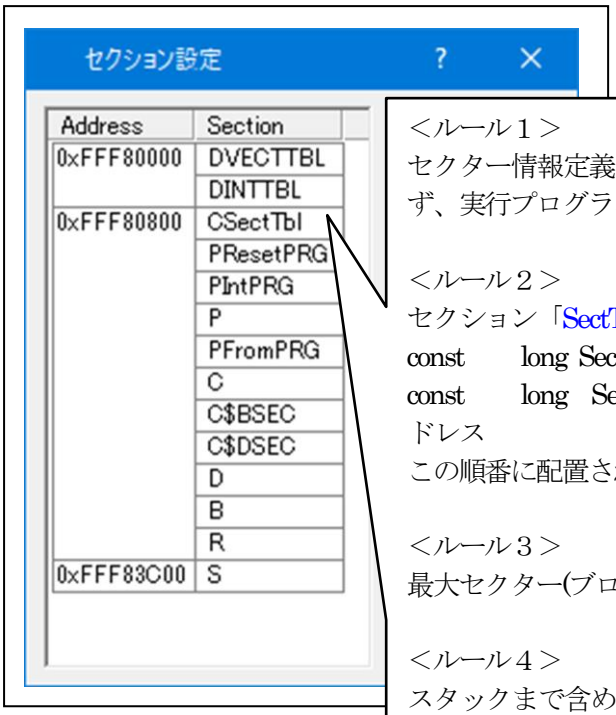
１２）「sFlashByteRead()」開数を確認または変更する。

【機能】指定バイトを QUAD モードで読み出しする。

```
行番... S... ソース
169     //*****************************************************************************
170     //      sFlashByteRead      指定バイト数の読み込み
171     //
172     //*****************************************************************************
173     void    sFlashByteRead(ULONG adr,UCHAR *rdat,LONG rsize)
174     {
175         LONG unit;
176
177         if ((rsize % 4) == 0)       {unit = 4;}
178         else if ((rsize % 2) == 0) {unit = 2;}
179         else                        {unit = 1;}
180
181         while(rsize > 0) {
182             read_data_quad(adr,rdat,unit);
183             adr   += unit;
184             rdat  += unit;
185             rsize -= unit;
186         }
187     }
188     //*****************************************************************************
```

４－４）作成したシリアル FlashROM プログラムをビルドする。

１）セクションの確認



＜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」に割付る。

＜ルール２＞
セクション「SectTbl」のデータ配置にルールがある。

```
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る。

＜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
```

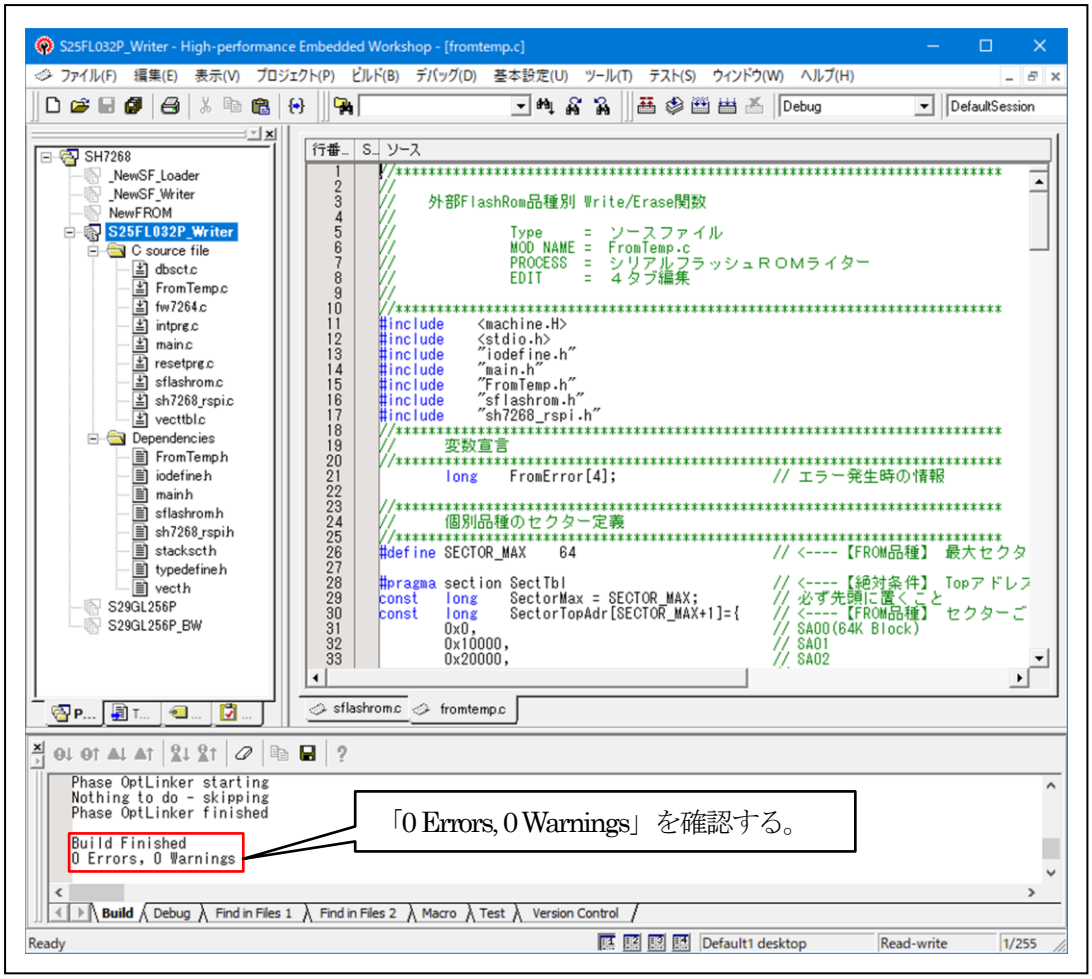
数値変更しないで下さい。

＜ルール６＞
下記４関数は、名称・引数・戻り値等の仕様変更は不可です。

```
1) INT      FromOpen(void);
2) INT      FromClose(void);
3) INT      FromSectorEraseProc(UCHAR *cmd);
4) INT      FromWriteProc(UCHAR *cmd);
```

以上のルールは厳守して下さい。

2）ビルドの実施

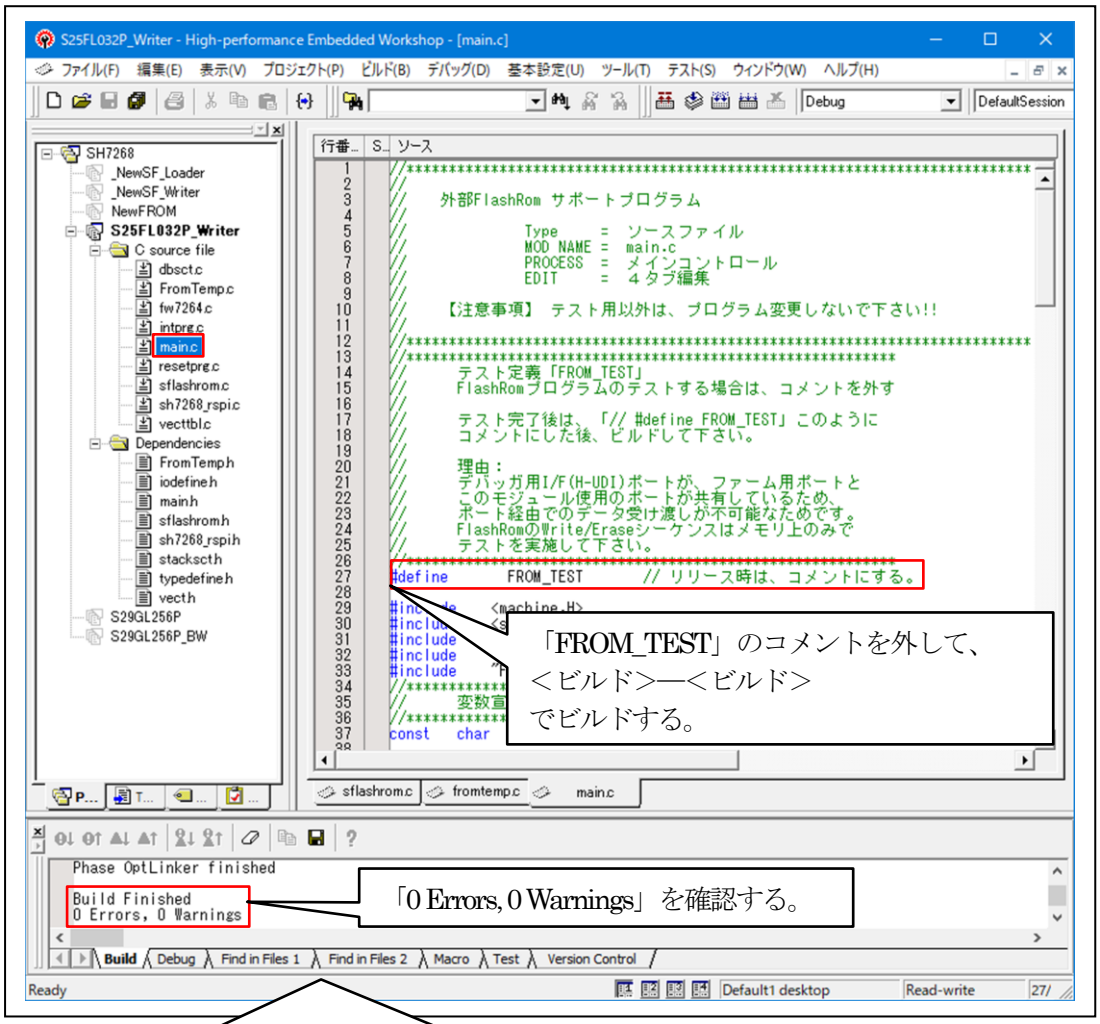

S25FL032P_Writer - High-performance Embedded Workshop - [fromtemp.c]
Phase OptLinker starting
Nothing to do - skipping
Phase OptLinker finished
Build Finished
0 Errors, 0 Warnings
「0 Errors, 0 Warnings」を確認する。

５．シリアル FlashROM プログラムのデバッグ方法

５－１）ターゲット基板側のモード設定

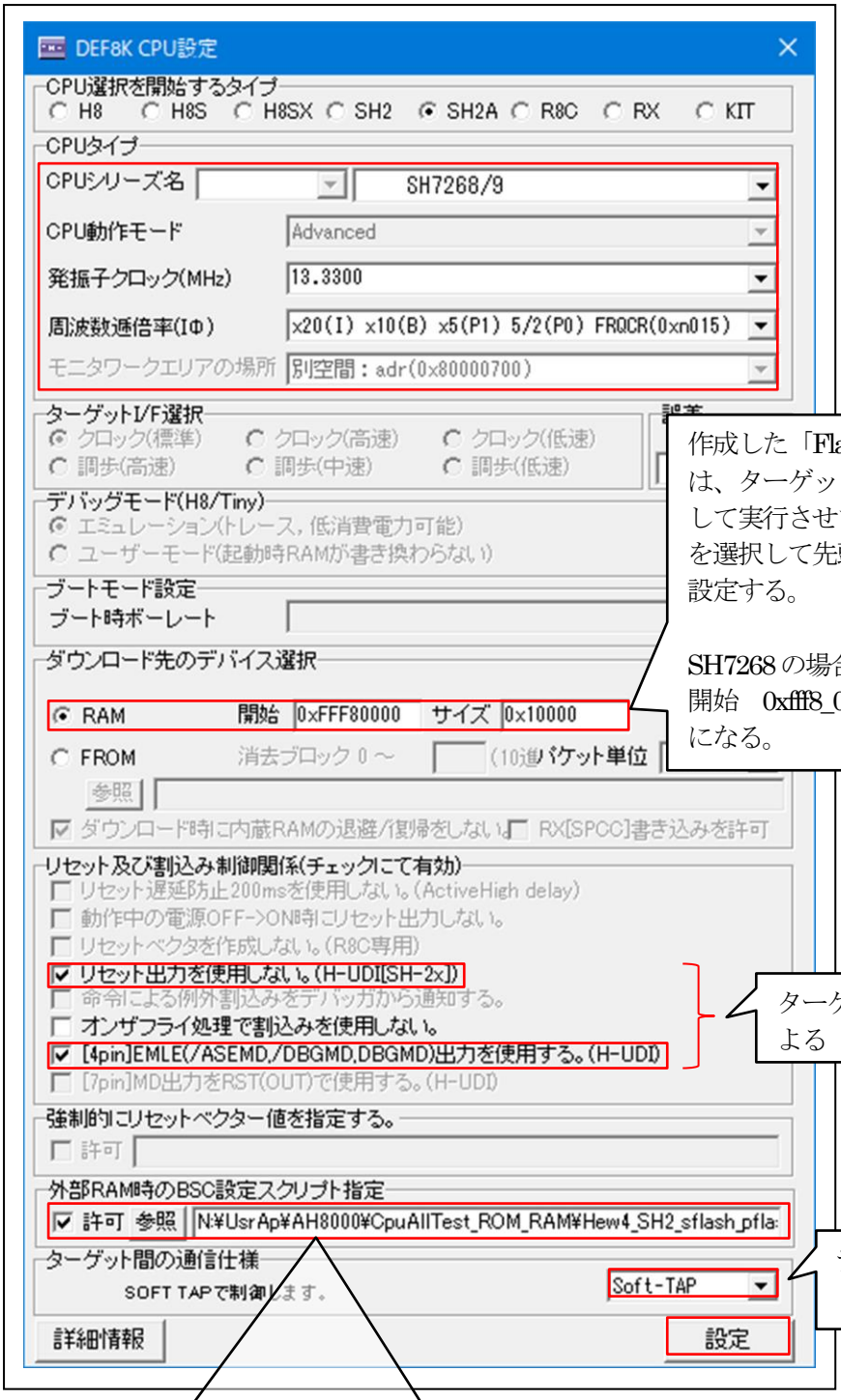
デバッグ中は、「ブートモード０」（CS0 空間に接続されたメモリからブートする）にして下さい。理由は、「ブートモード３」（シリアルフラッシュメモリからブートする）のままですとリセット解除後、ブートプログラムが起動され内蔵 RAM の先頭から「0x2000」エリアにローダプログラムがロードされてしまうからです。

５－２）デバッグの準備のため「main.c」を開きビルドする。



デバッガ使用の通信ポートと、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。
対策として、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用意してあります。

５－３）「DEF8K」の「CPU 設定」にデバッグ環境を設定する。



作成した「FlashROM」プログラムは、ターゲットの内臓 RAM に転送して実行させますので「RAM」側を選択して先頭アドレスとサイズを設定する。

SH7268 の場合
開始　0xfff8_0000 サイズ 0x10000 になる。

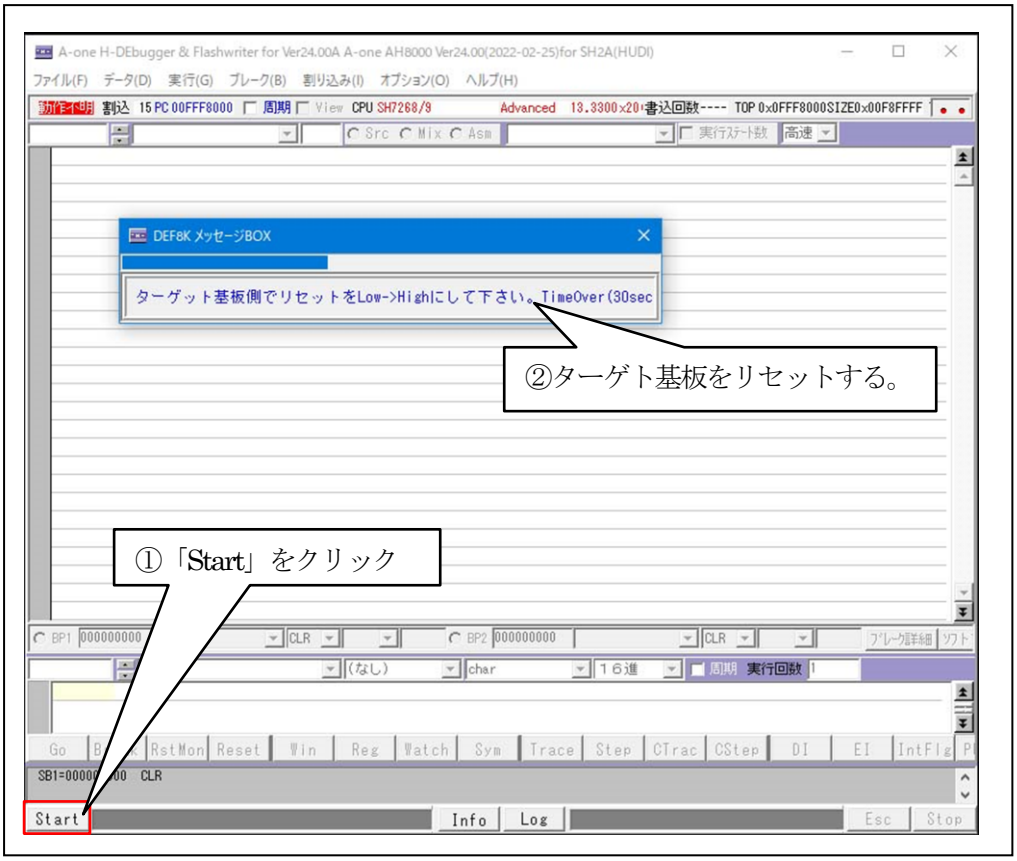
ターゲット側の基板構成による

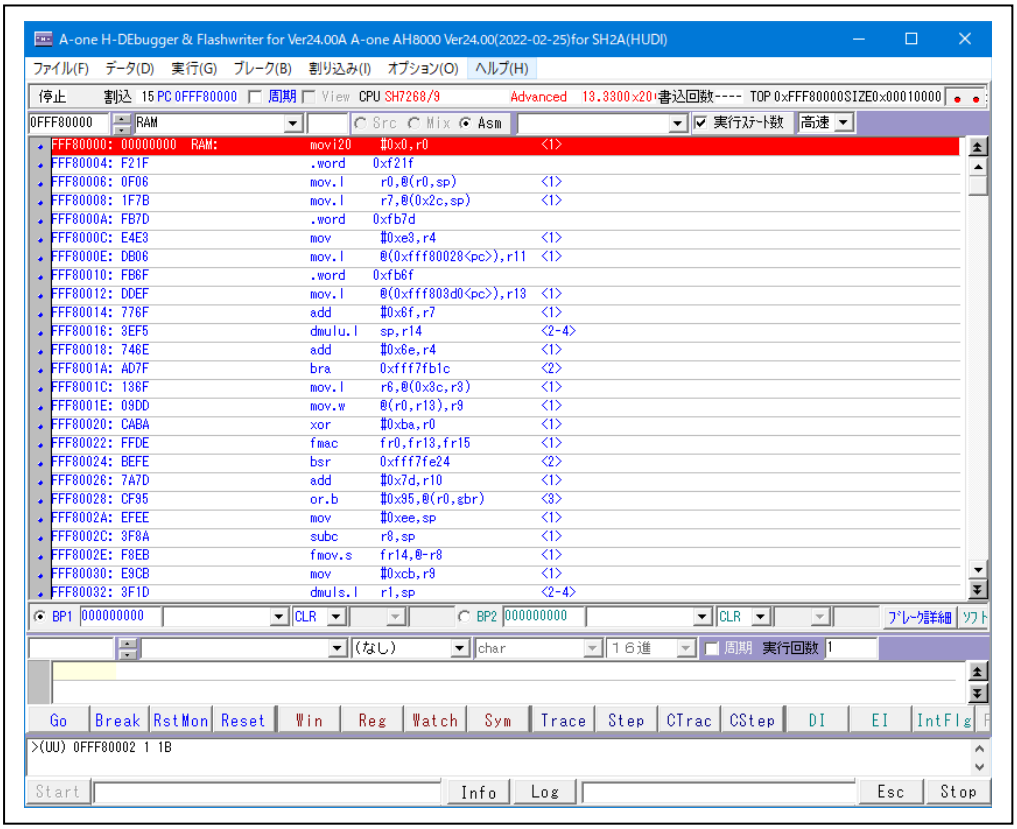
デバッグ時は、「Sof-TAP」を指定する。

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する。
SH7268 の場合
"c:s¥Aone¥DEF8K¥rom-custom¥SH7268"に、
例として「SH7268-BSC.log」を用意してあるので、目的ターゲット基板にカスタマイズして下さい。

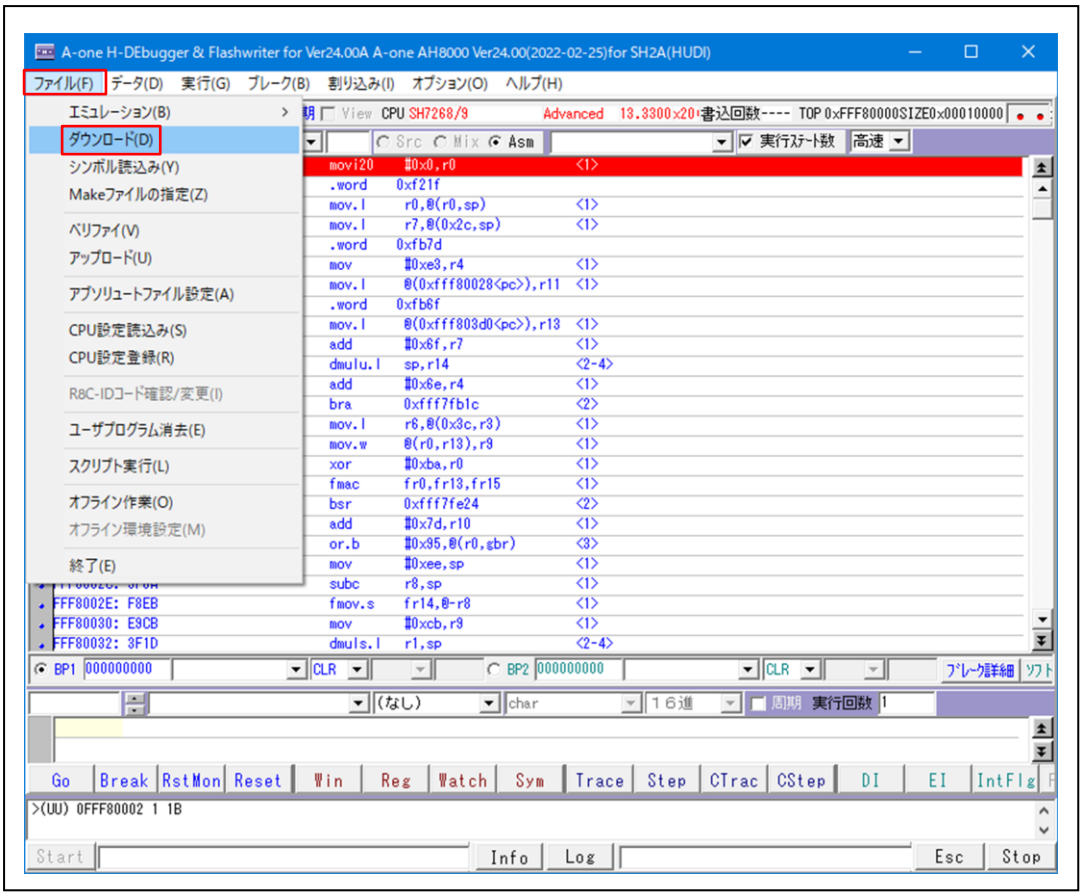
５－４）「DEF8K」でデバッグを開始する。

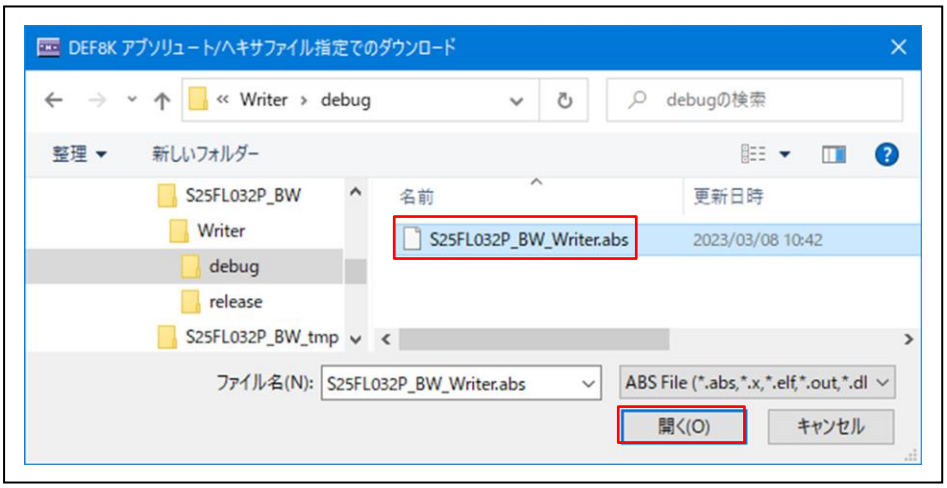
１）ターゲットと接続する。



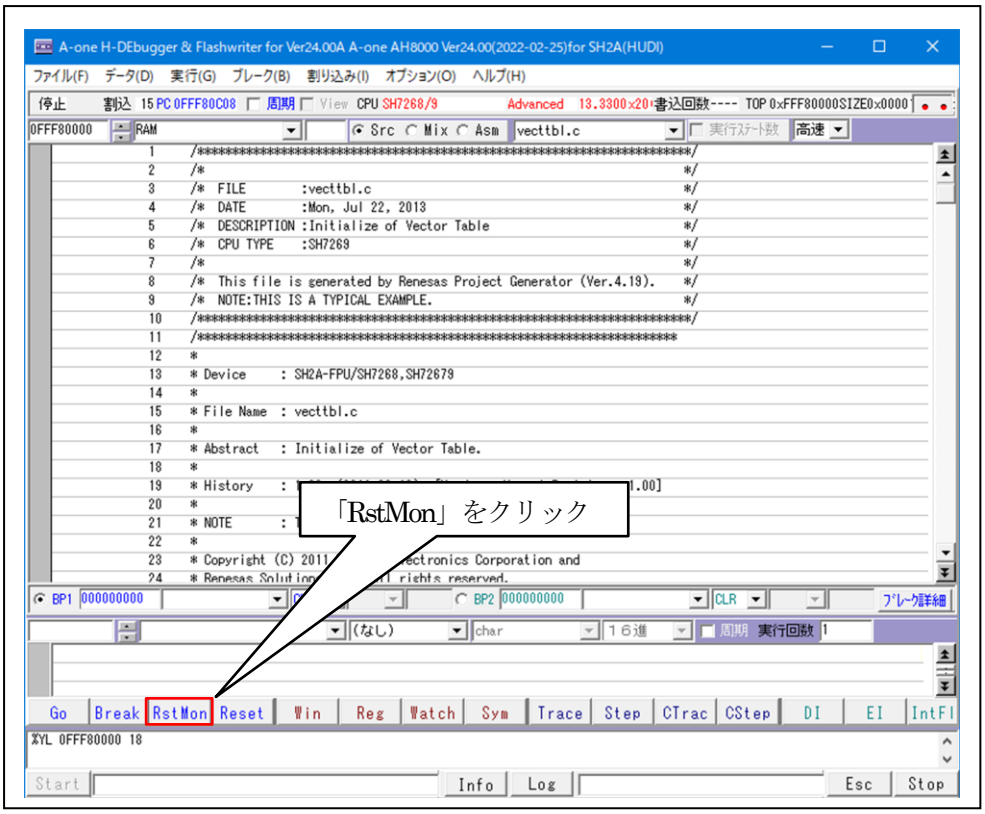


2）シリアル FlashROM プログラムをダウンロードする。

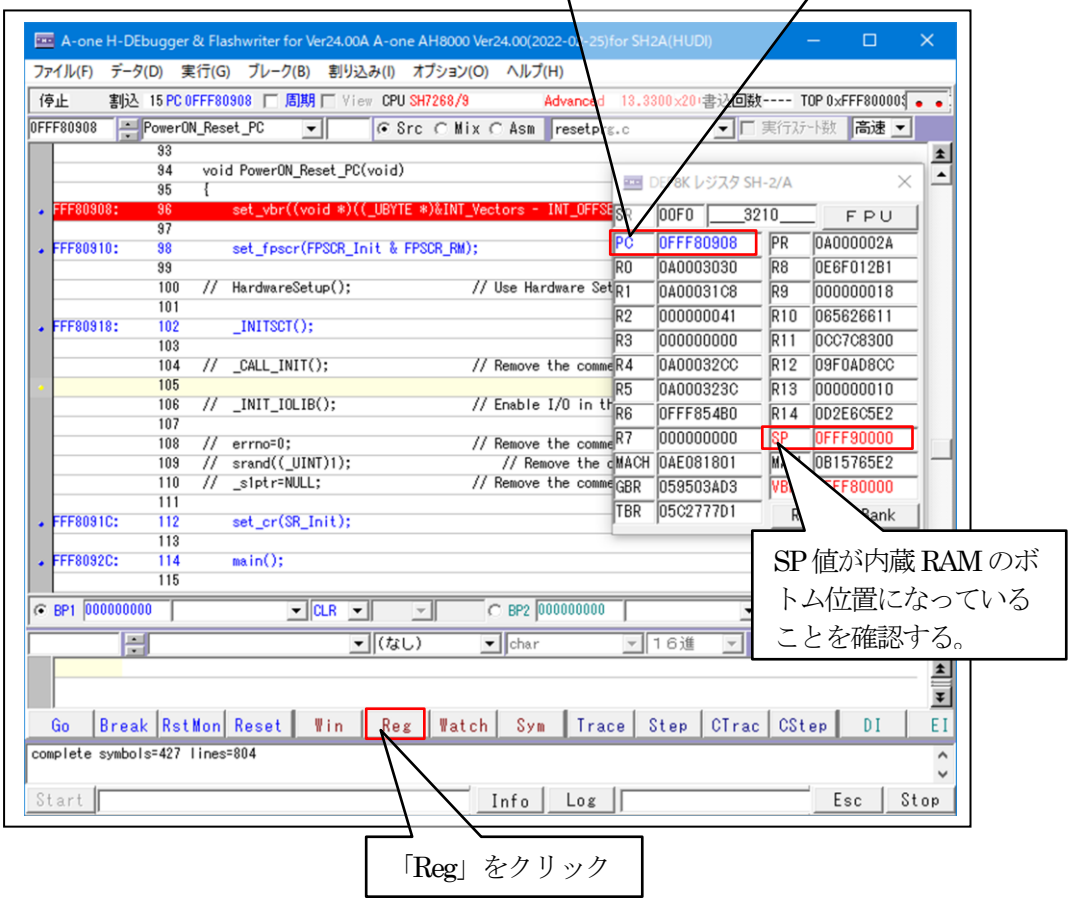




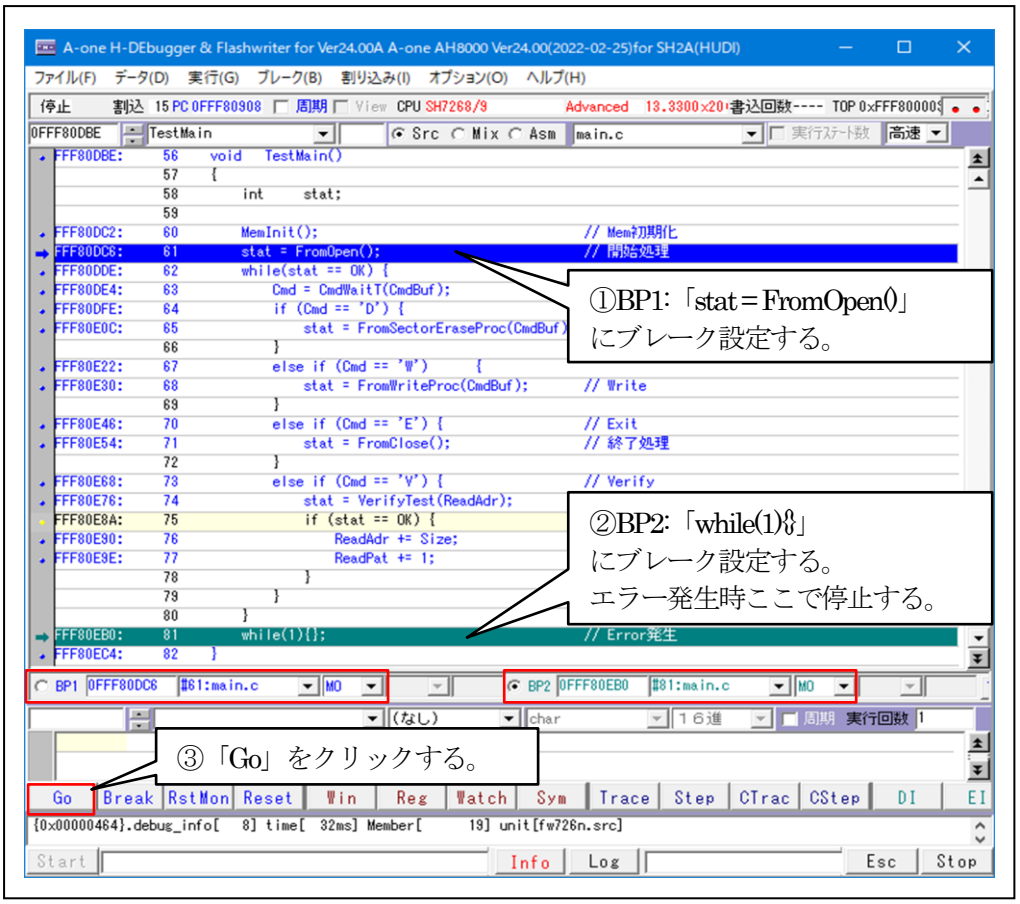
3）「RstMon」をクリックする。

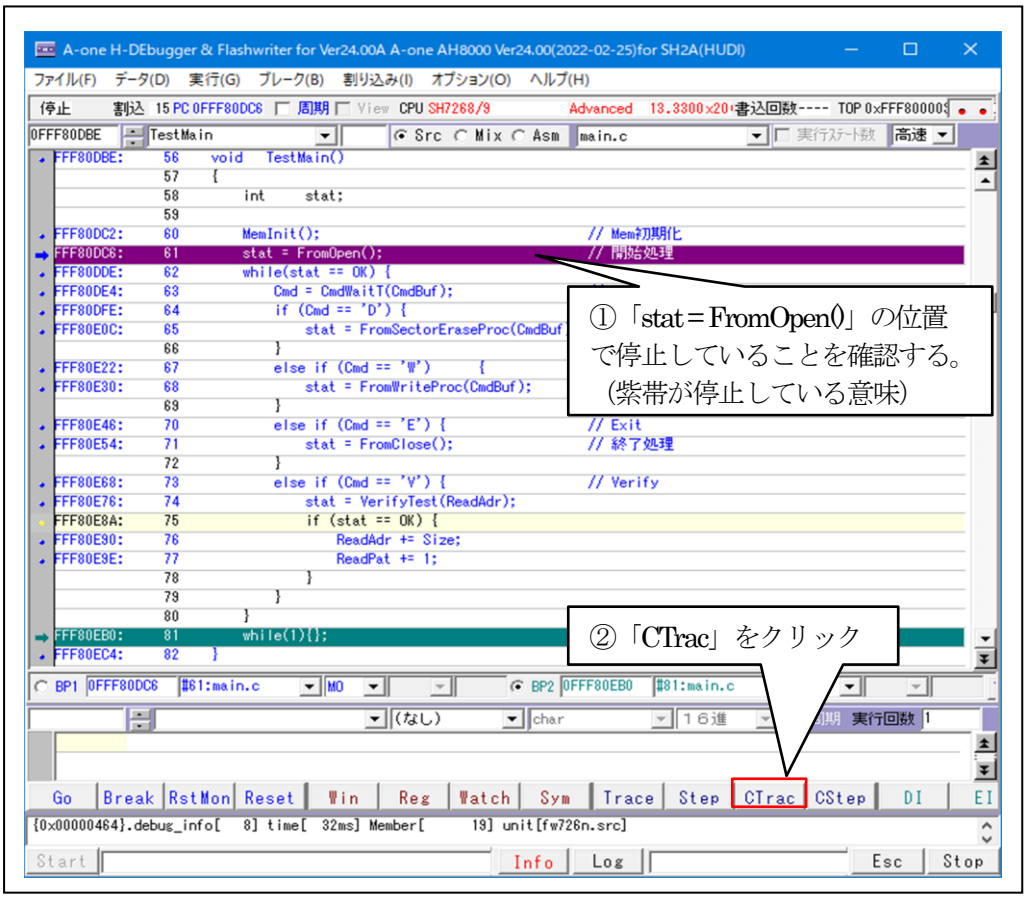


「void PowerON_Rest_PC」関数の先頭の PC 値になっている事を確認する。

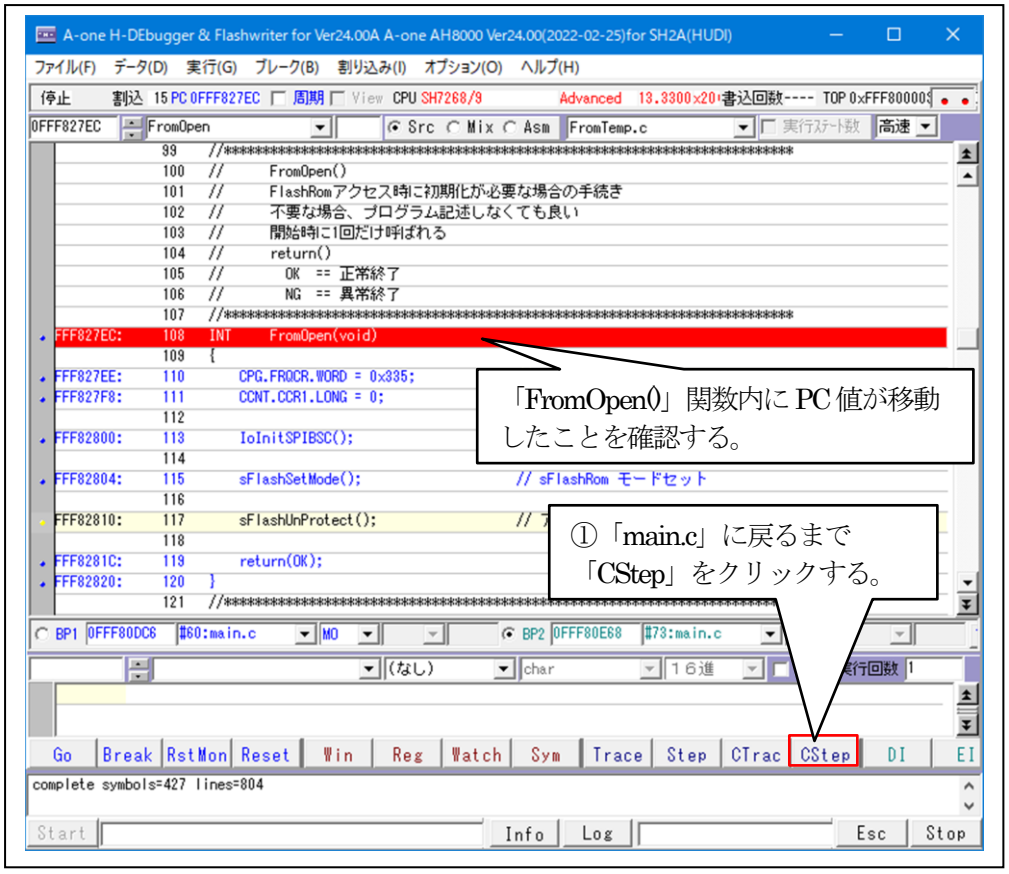


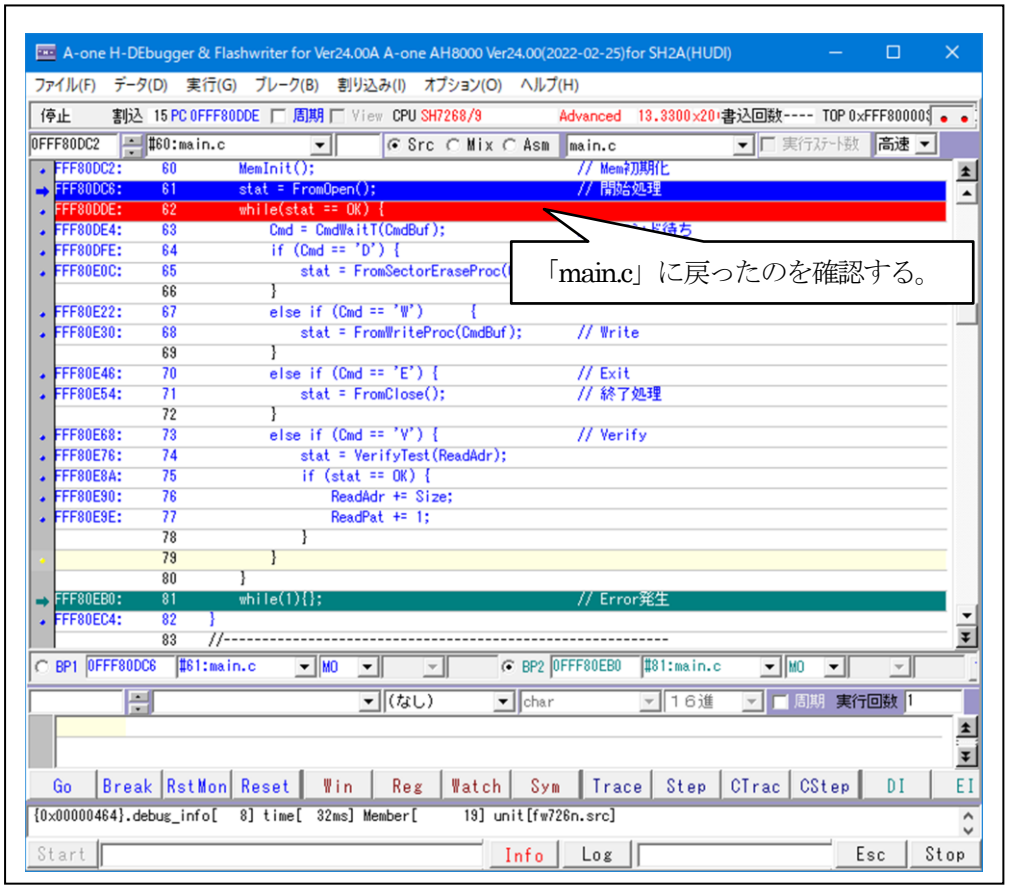
4）「main.c::TestMain()」にブレークポイントを設定後、実行する。



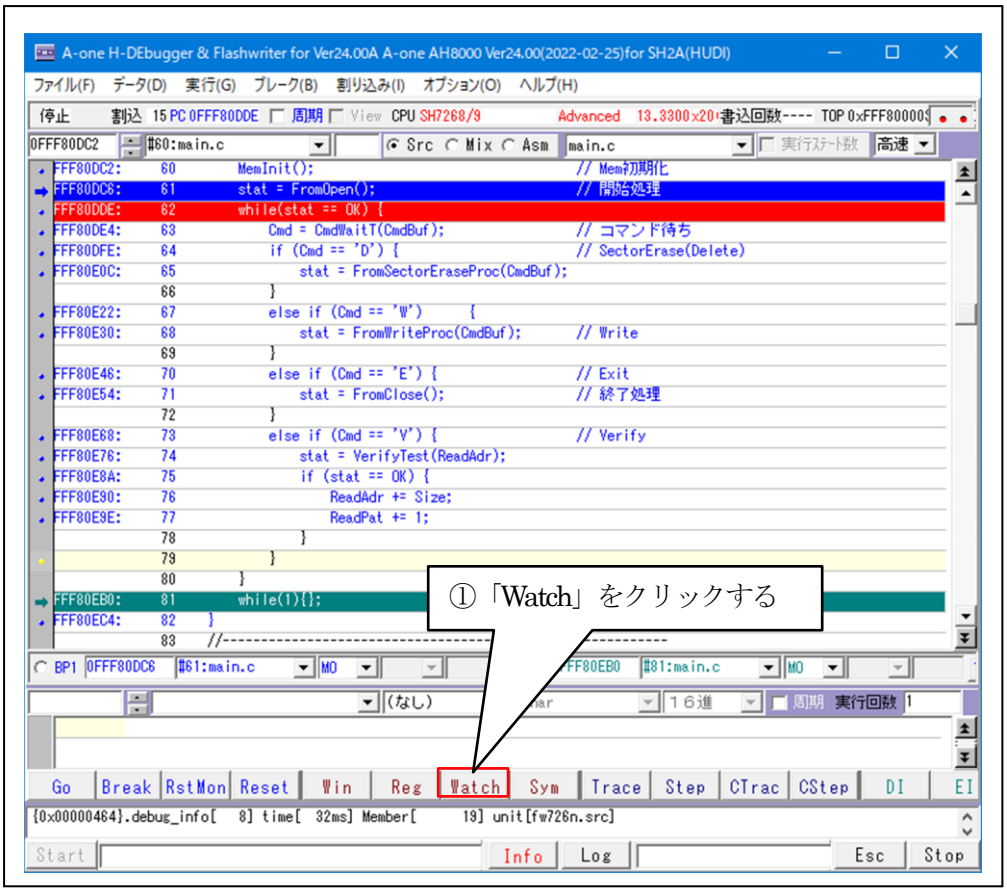


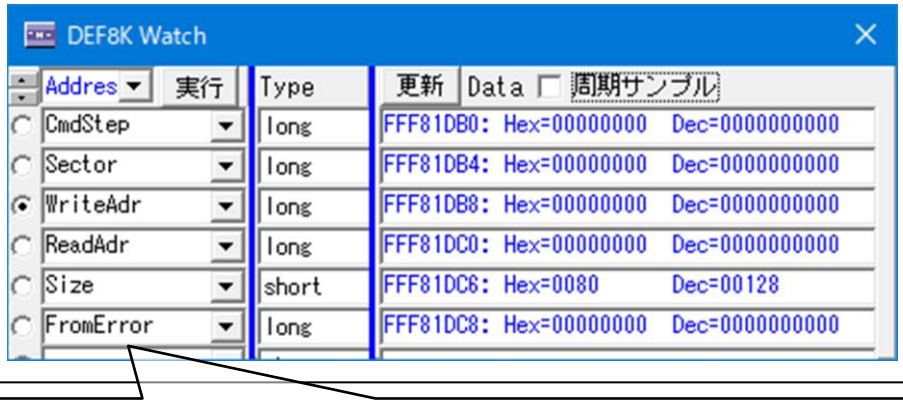
5）オープン「FromOpen()」を確認する。





6）デバッグを進めるため準備するため「Watch」画面を開く。



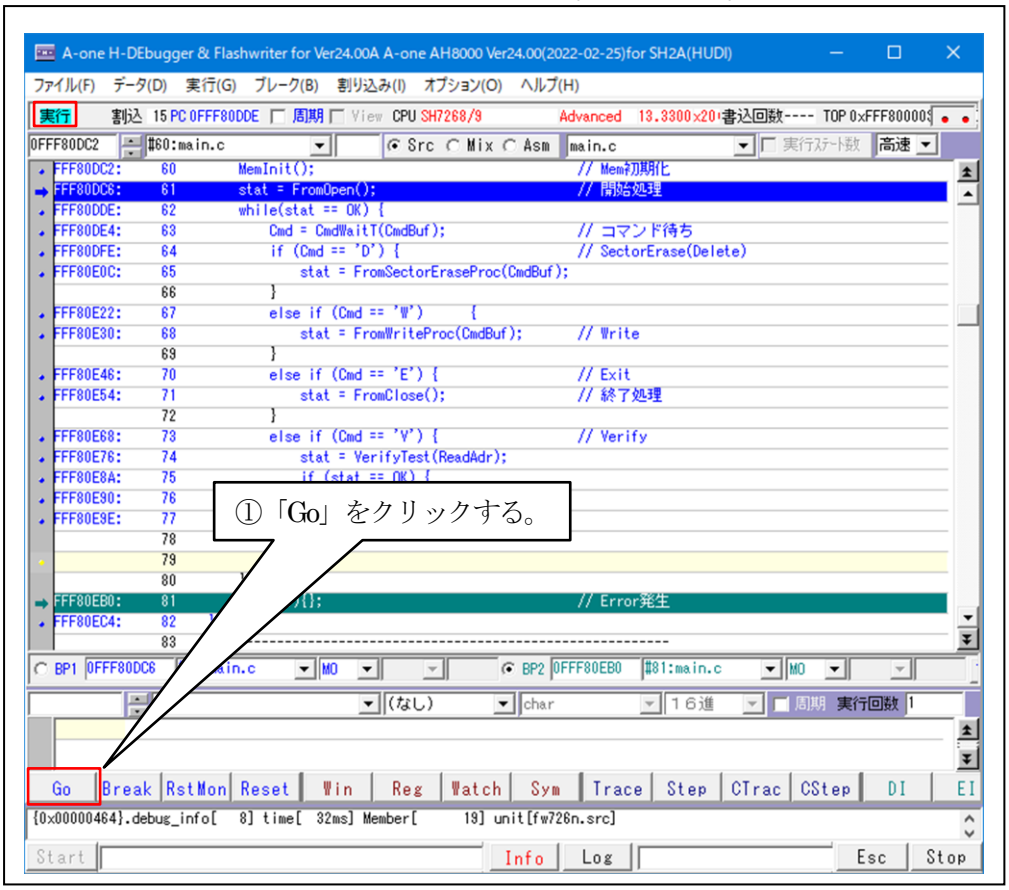


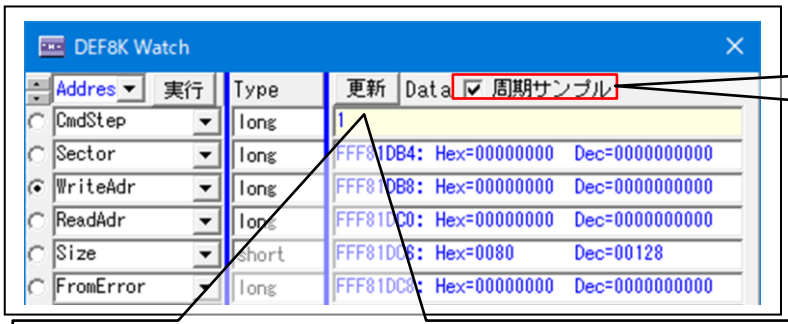
デバッグを進めるため、Watch 画面に変数を登録する。

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) ReadAdr　　Verify 中読み出しアドレス
5) Size　　パケットサイズ(128/256/512)
6) FromError　　エラー発生時情報

6 変数を登録します。

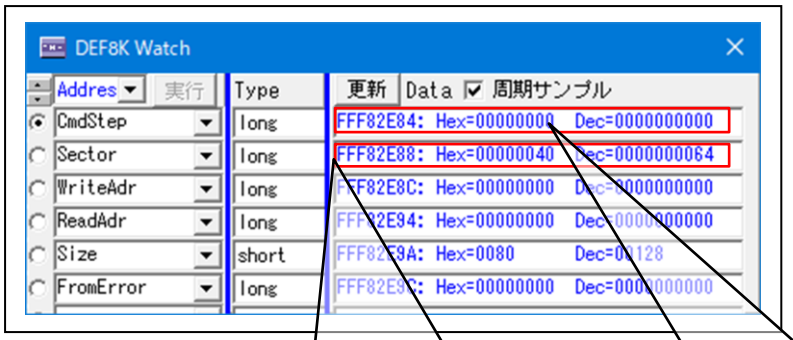
7）セクターイレーズ関数「FromSectorEraseProc(UCHAR *cmd)」を確認する。





②変数「CmdStep」を数値「1」にする。// ⏎１⏎で設定

＜備考＞
「main.c」の「CmdWaitT()」関数を参照。
疑似コマンドを作成して「FromSectorEraseProc(UCHAR *cmd)」をコールする。

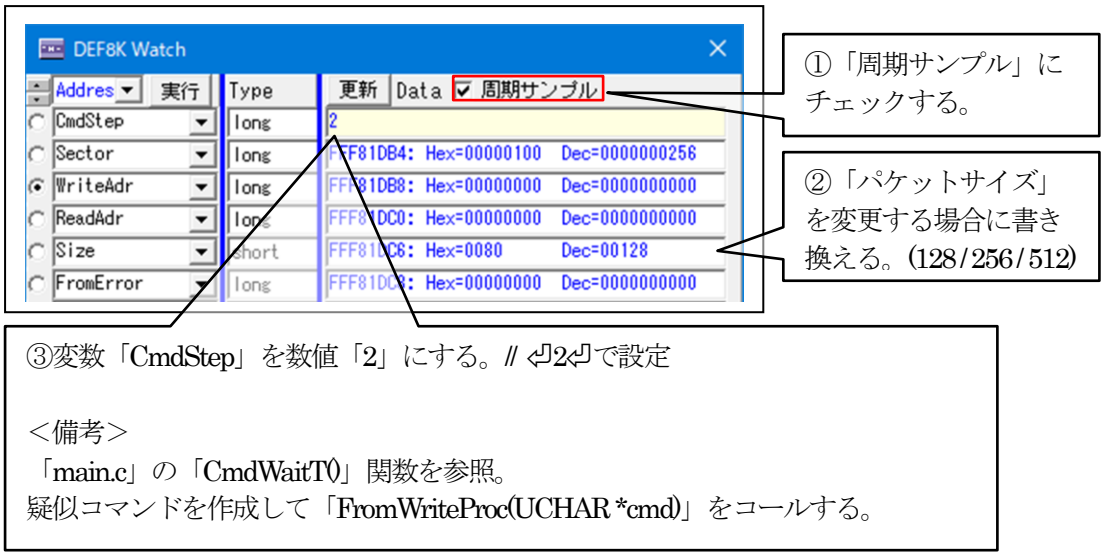


「Sector」はゼロ「0」から登録セクタ数までインクリメントする。

異常終了すると BP2 で停止する。

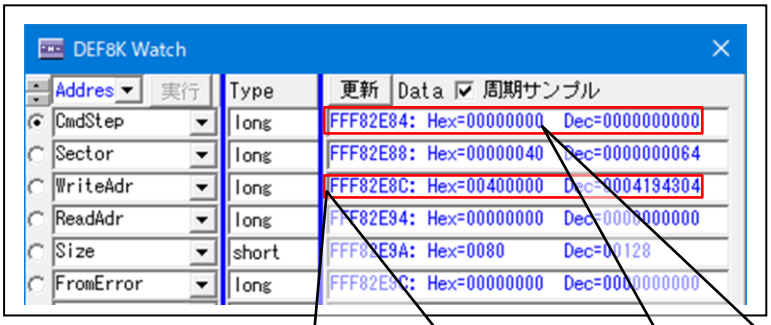
「CmdStep」は、全セクターイレーズが正常終了すると「1」から「0」に変化する。

8）書き込み関数「FromWriteProc(UCHAR *cmd)」を確認する。

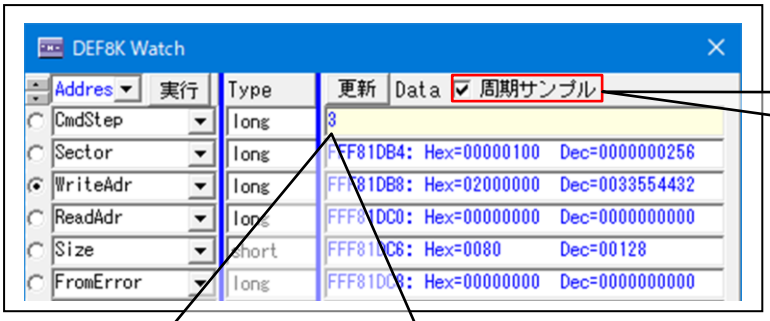


③変数「CmdStep」を数値「2」にする。// ⏎2⏎で設定

＜備考＞
「main.c」の「CmdWaitT()」関数を参照。
疑似コマンドを作成して「FromWriteProc(UCHAR *cmd)」をコールする。



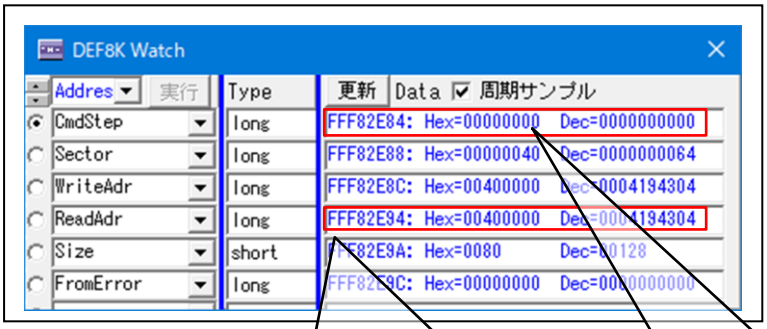
9）パラレル FlashROM 全エリアをベリファイする。



①「周期サンプル」にチェックする。

②変数「CmdStep」を数値「3」る。// ⏎3⏎で設定

＜備考＞
「main.c」の「CmdWaitT()」関数を参照。
書き込み時に使用した同等の疑似データを作成して全エリアのベリファイを実施する。

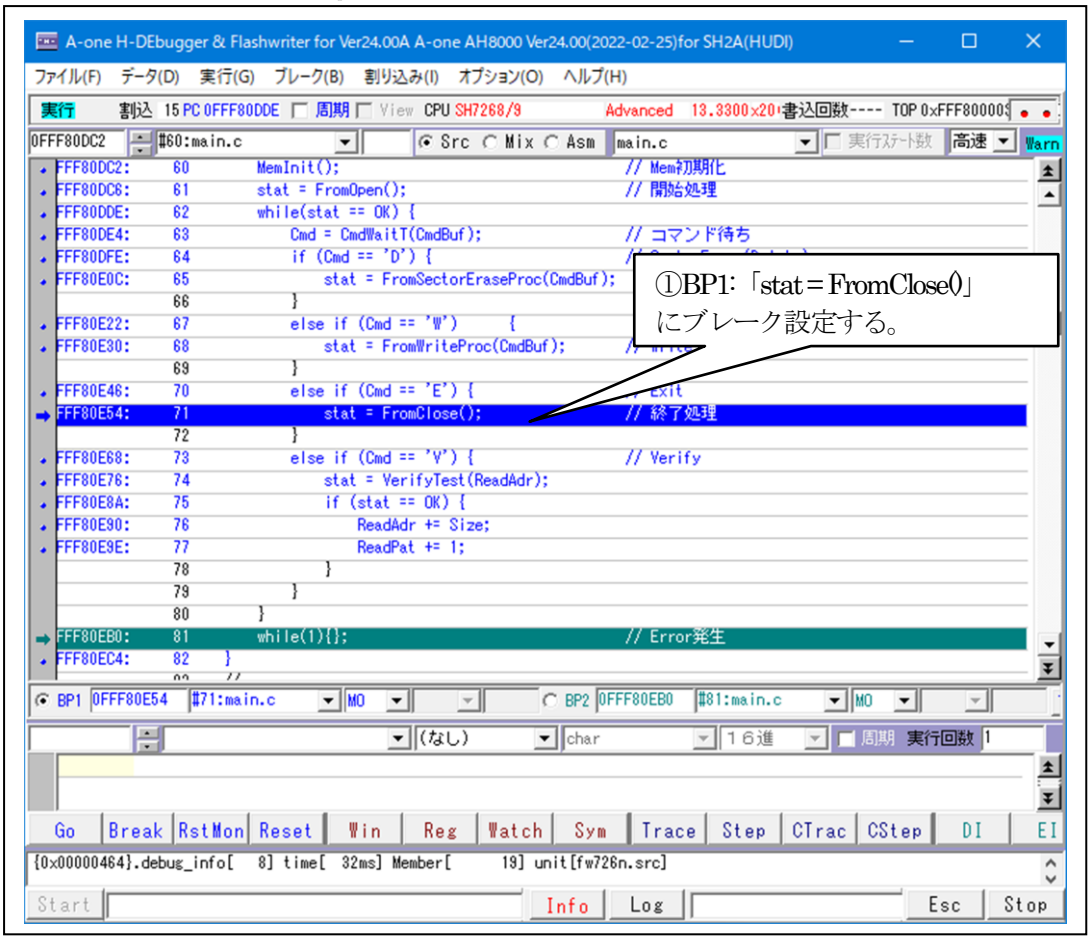


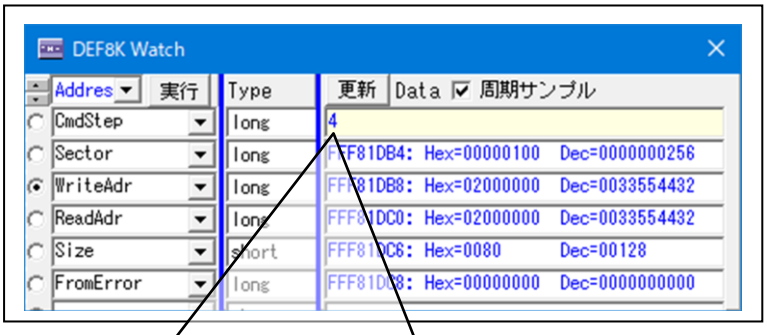
「ReadAdr」は、FlashROM の読み出し先頭アドレス「0x0」から登録最終アドレスまでインクリメントする。

異常終了すると BP2 で停止する。

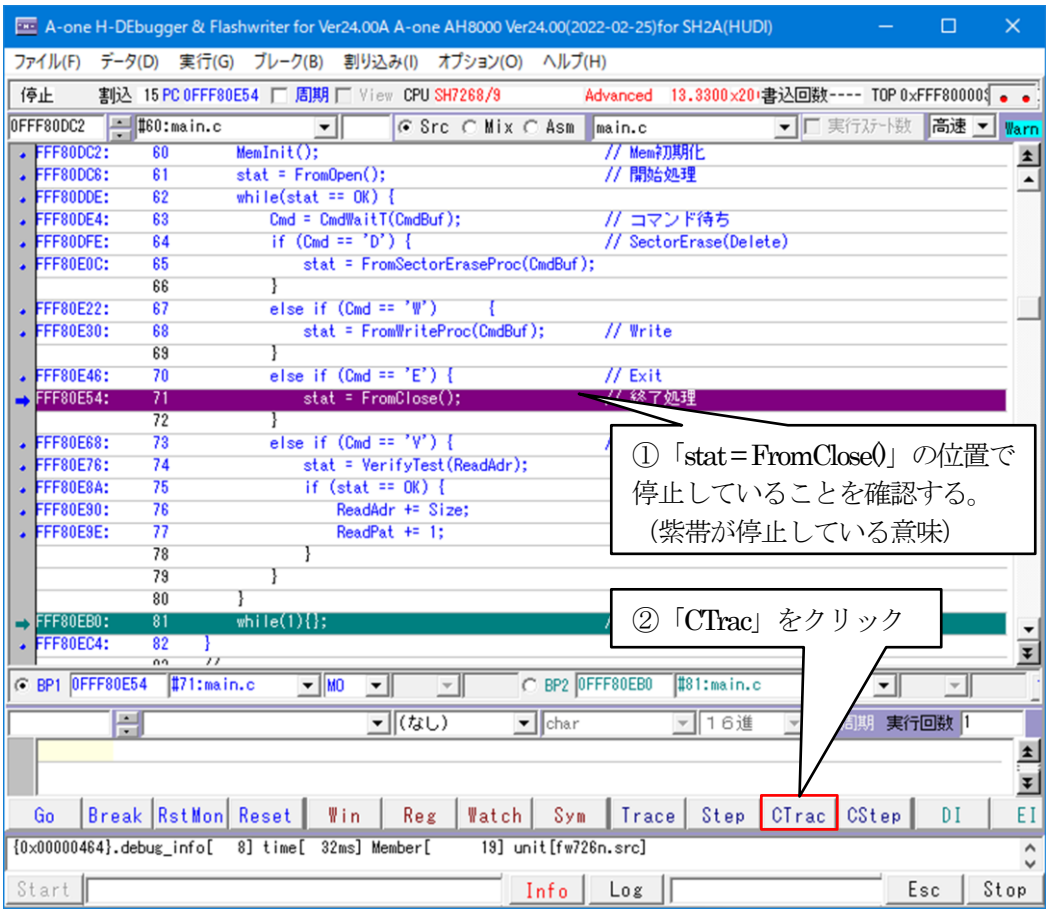
「CmdStep」は、全エリアの書き込みが正常終了すると「2」から「0」に変化する。

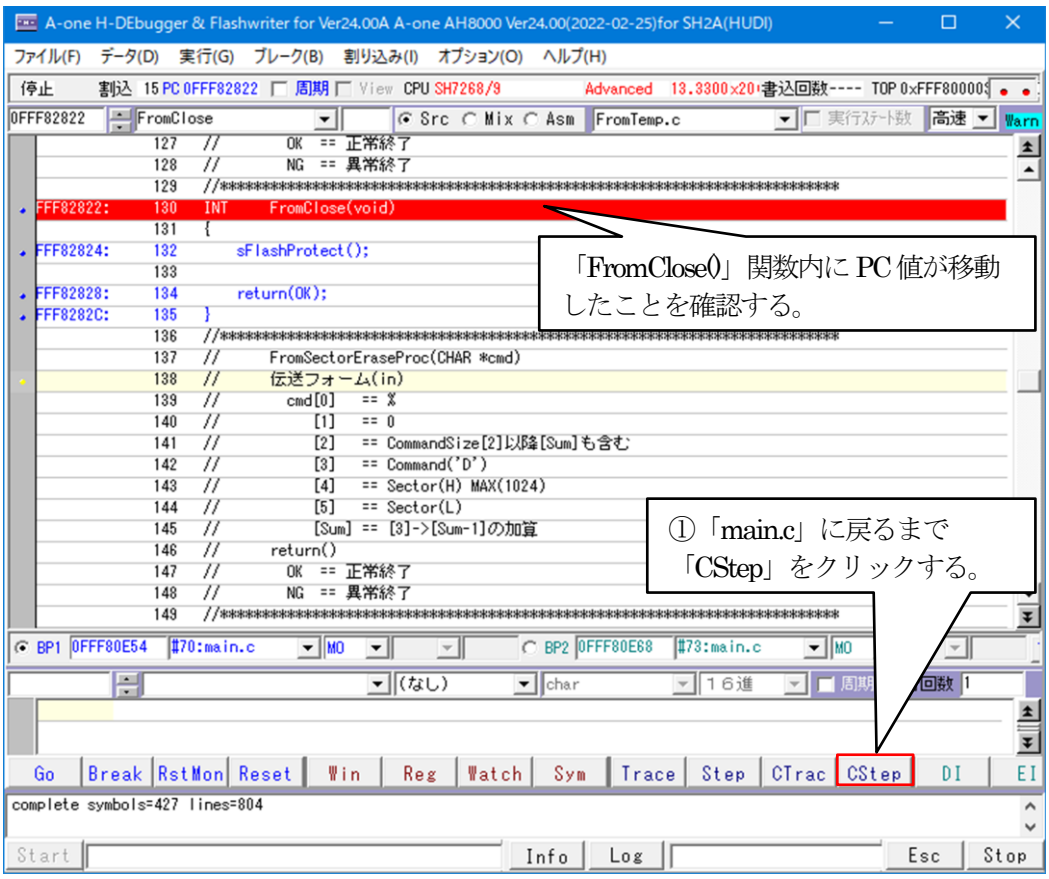
１０）クローズ「FromClose()」を確認する。

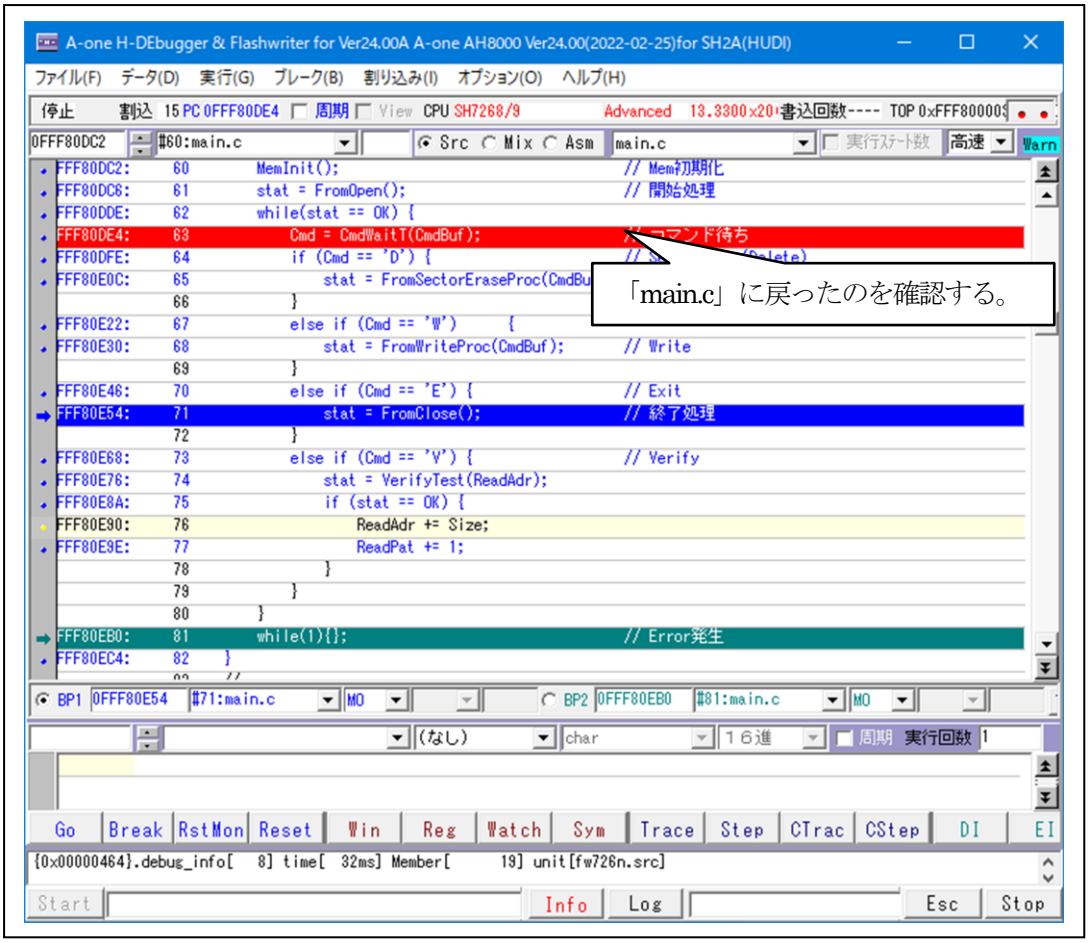




＜備考＞
「main.c」の「CmdWaitT()」関数を参照。
疑似コマンドを作成して「FromClose()」をコールする

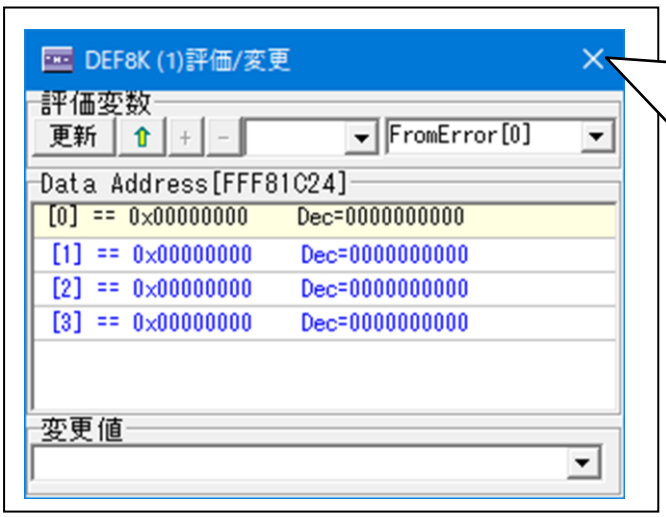

①「stat＝FromClose()」の位置で停止していることを確認する。
（紫帯が停止している意味）
②「CTrac」をクリック


「FromClose()」関数内に PC 値が移動したことを確認する。
①「main.c」に戻るまで「CStep」をクリックする。



以上で FlashROM プログラムの動作確認は終了です。

１１）異常終了時のエラー情報「long FromError[4]」の参照

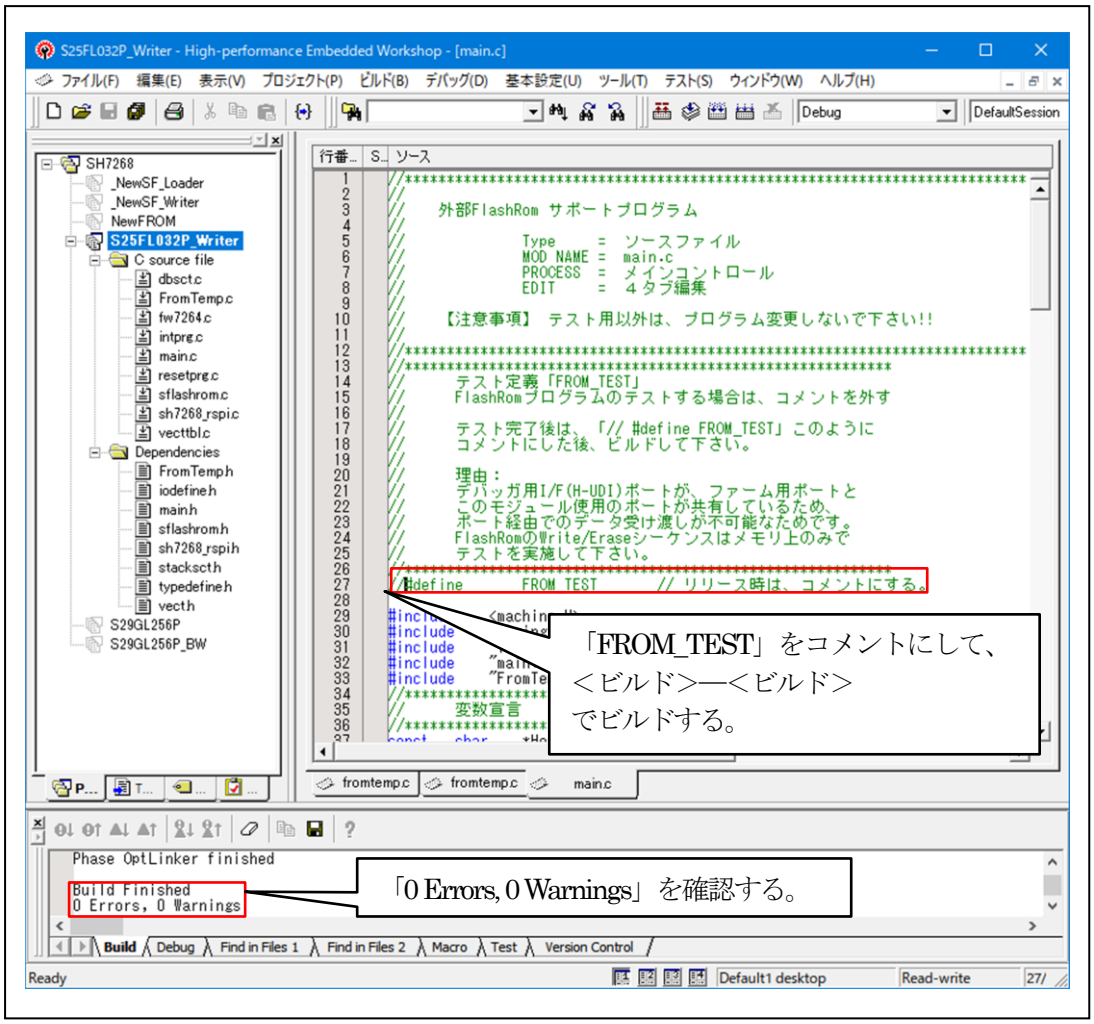


＜評価・変更窓＞

エラー発生にて異常終了した場合は、変数「FromError[4]」に情報が残しますのでプログラム修正の参考にする。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利用する。

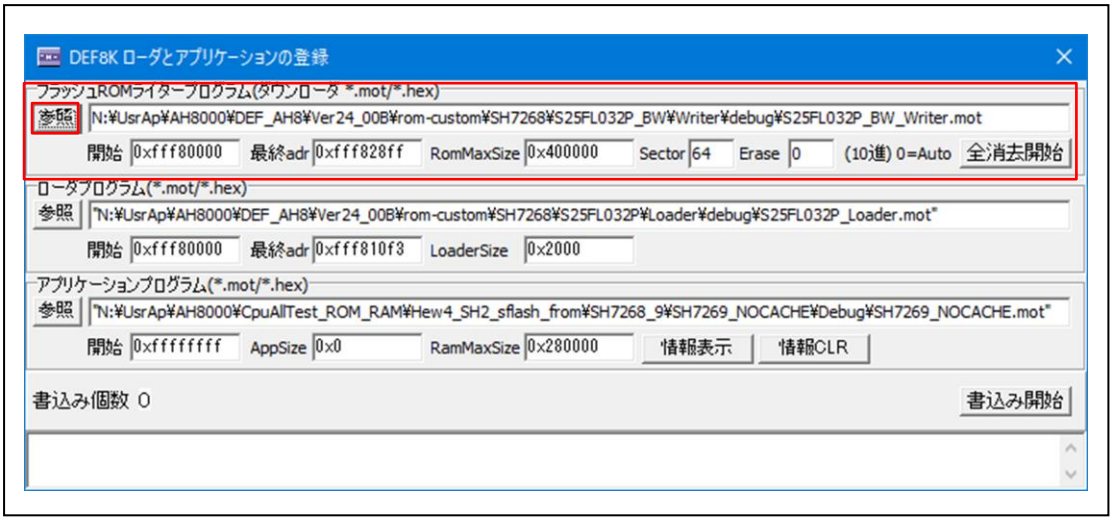
6．シリアル FlashROM プログラムを正規リリース登録する。

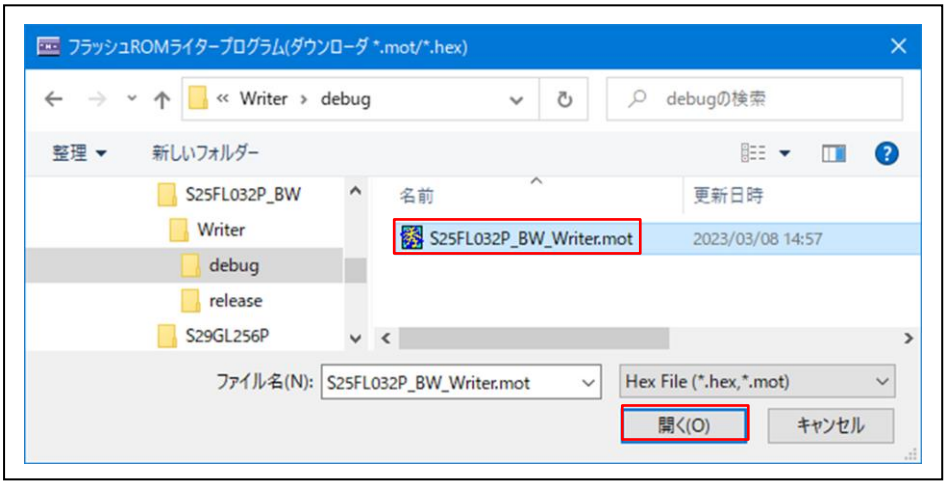
6－1）リリース登録のため Hew4 で「main.c」を開きビルドする。

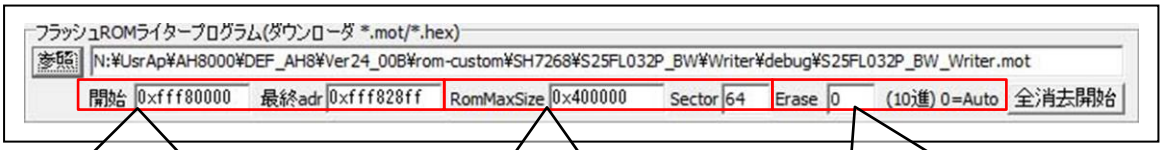


６－２）「DEF8K」の「ローダとアプリケーションの登録」に登録する。

１）フラッシュ ROM ライタープログラムの「参照」をクリックする。







シリアル FlashROM プログラムの開始アドレスと最終アドレスを表示する。

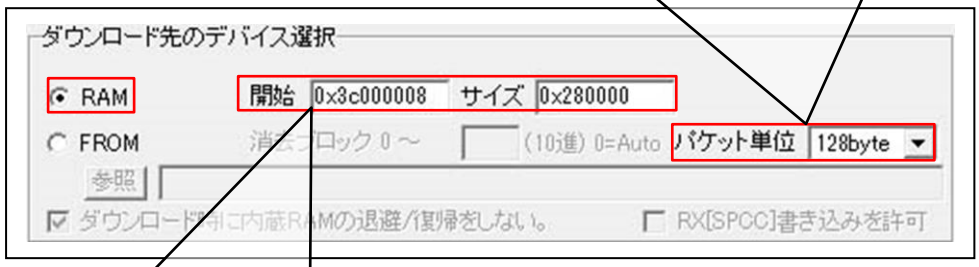
登録されているシリアル FlashROM の最大サイズとセクター数を表示する。

イレーズするセクター数を指定する。
ゼロ「0」指定の場合は、登録するアプリケーションサイズから自動で必要セクター数を算出してイレーズする。

6－3）「DEF8K」の「CPU 設定」にリリース環境を設定する。

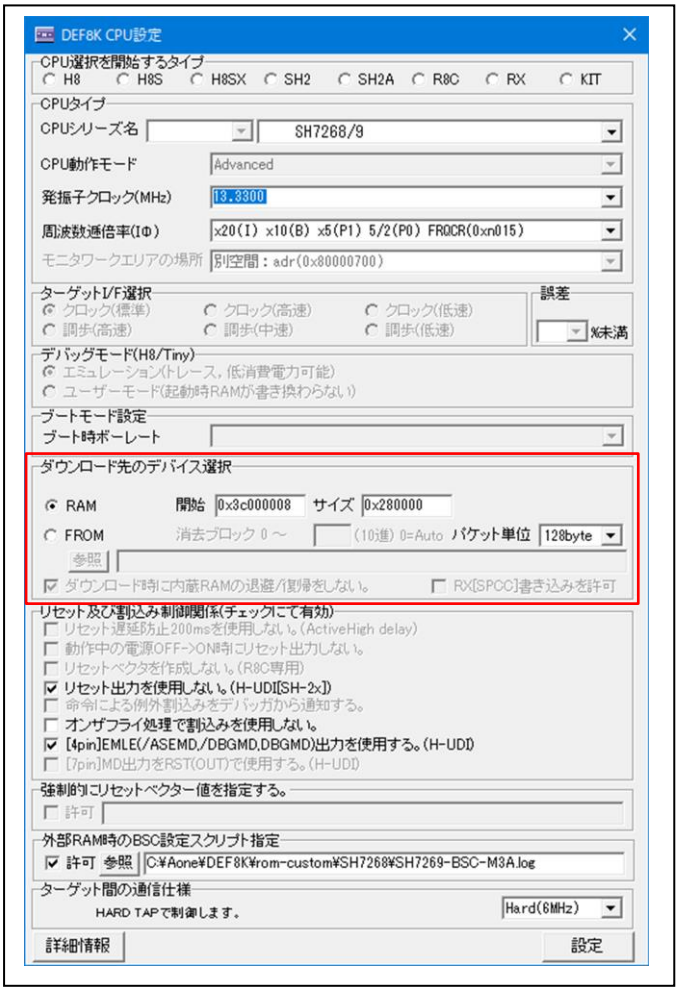
1）「RAM」を選択する。

「ローダとアプリケーションの登録」に登録した「シリアル FlashROM プログラム」がパケット通信対応の場合は選択が可能になる。(128/256/512byte)



開始： 内蔵 RAM の先頭アドレス＋８番地を設定する。
サイズ：内蔵 RAM の最大サイズを設定する。

＜備考＞
アプリケーションの開始アドレスが内蔵 RAM の先頭アドレスと終了アドレスがセットされているため。

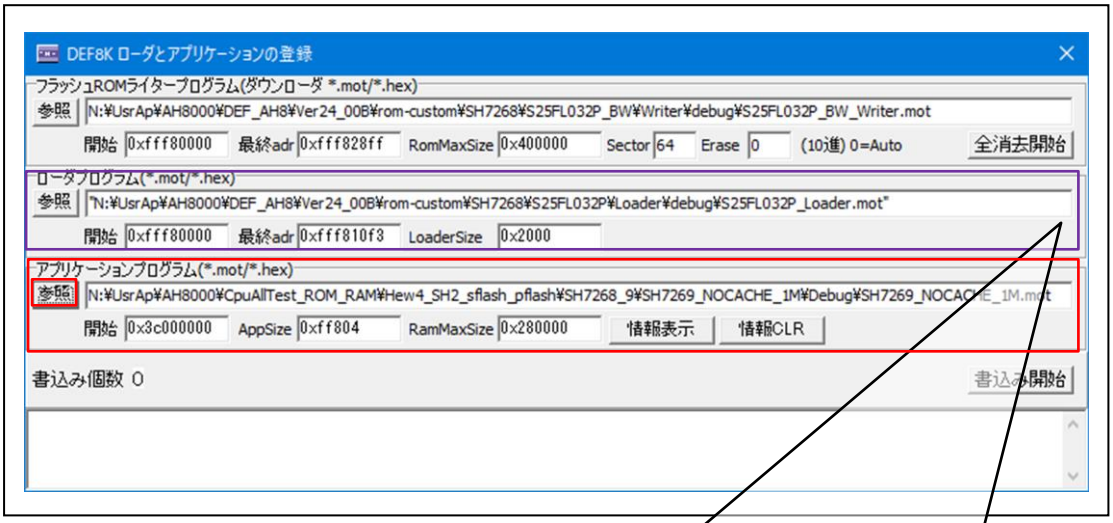


7．シリアル FlashROM プログラムの最終確認をする。

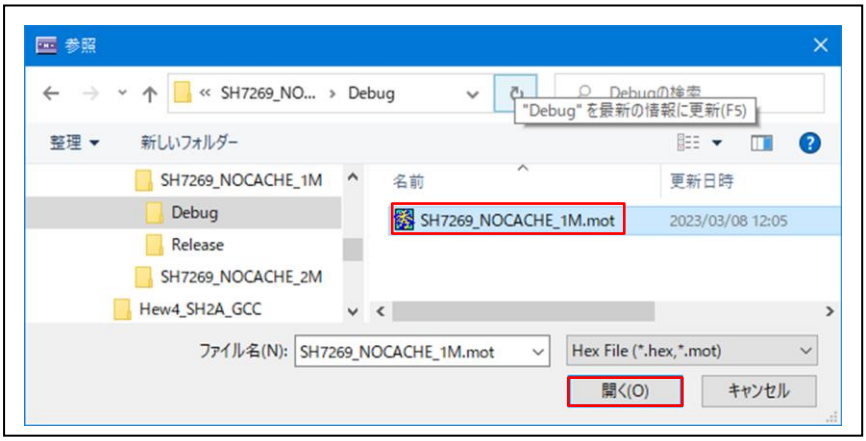
7－1）ターゲット基板側のモード設定
「ブートモード3」（シリアルフラッシュメモリからブートする）に設定する。

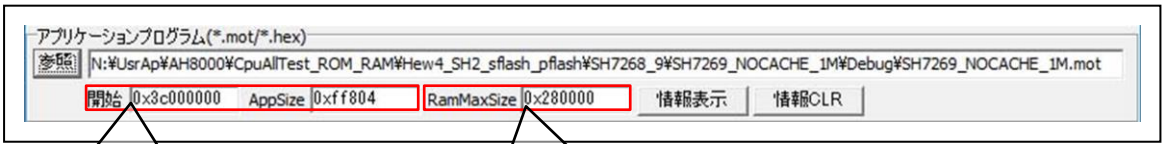
7－2）「ローダとアプリケーションの登録」で確認する。

1）アプリケーションプログラムの「参照」をクリックする。



追加シリアル FlashROM 用のローダプログラムが未完成の場合はデフォルトで用意してある「C:¥Aone¥DEF8K¥rom-custom¥SH7268¥S25FL032P¥Loader¥debug¥S25FL032P_Loader.mot」を一旦設定する。

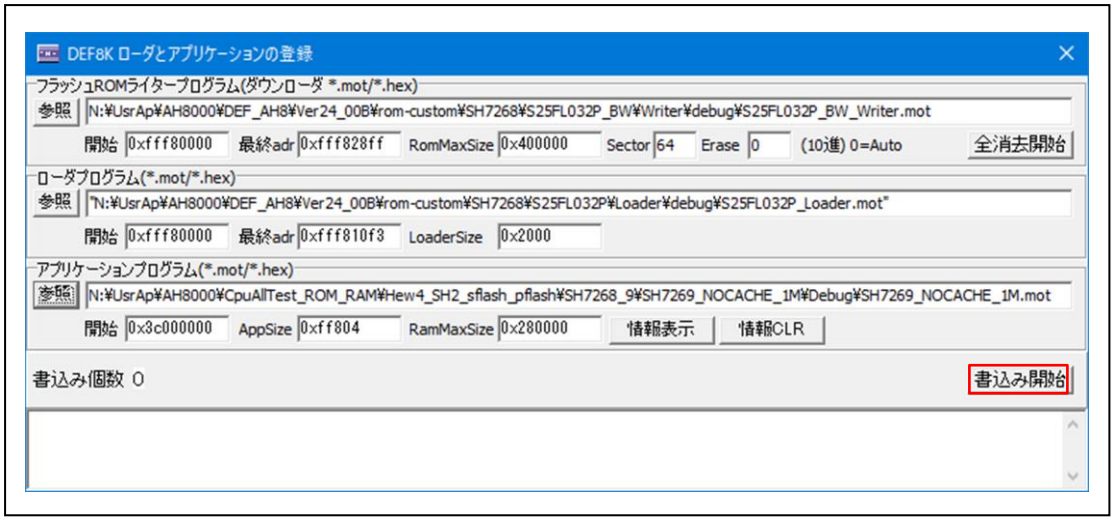


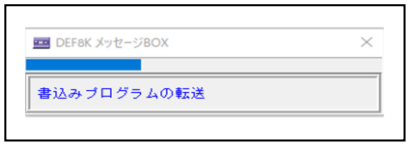


アプリケーションプログラムの開始アドレスとサイズを表示する。

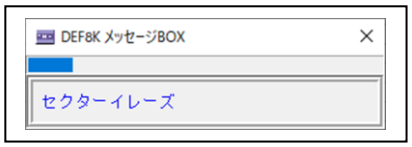
「CPU 設定」に設定した内蔵 RAM サイスを表示する。

2）「書込み開始」をクリックする。

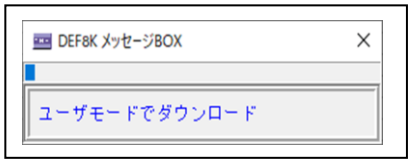




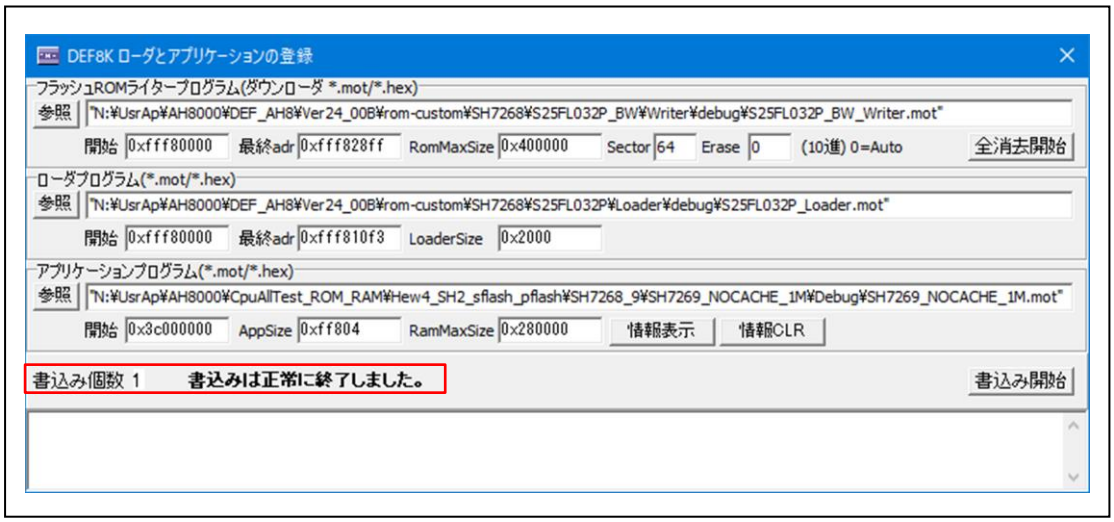
① FROM 書き込みプログラムの転送



② FROM のセクターイレーズ

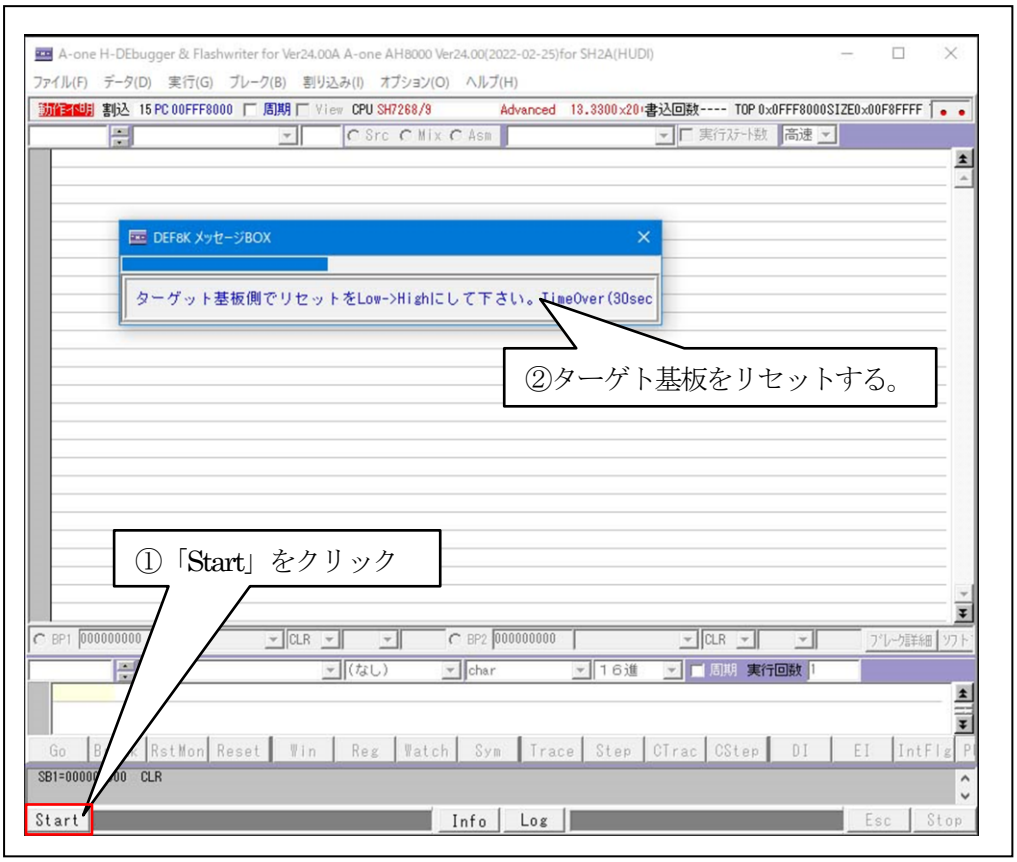


③ 拡張パケットでの FROM への書き込み

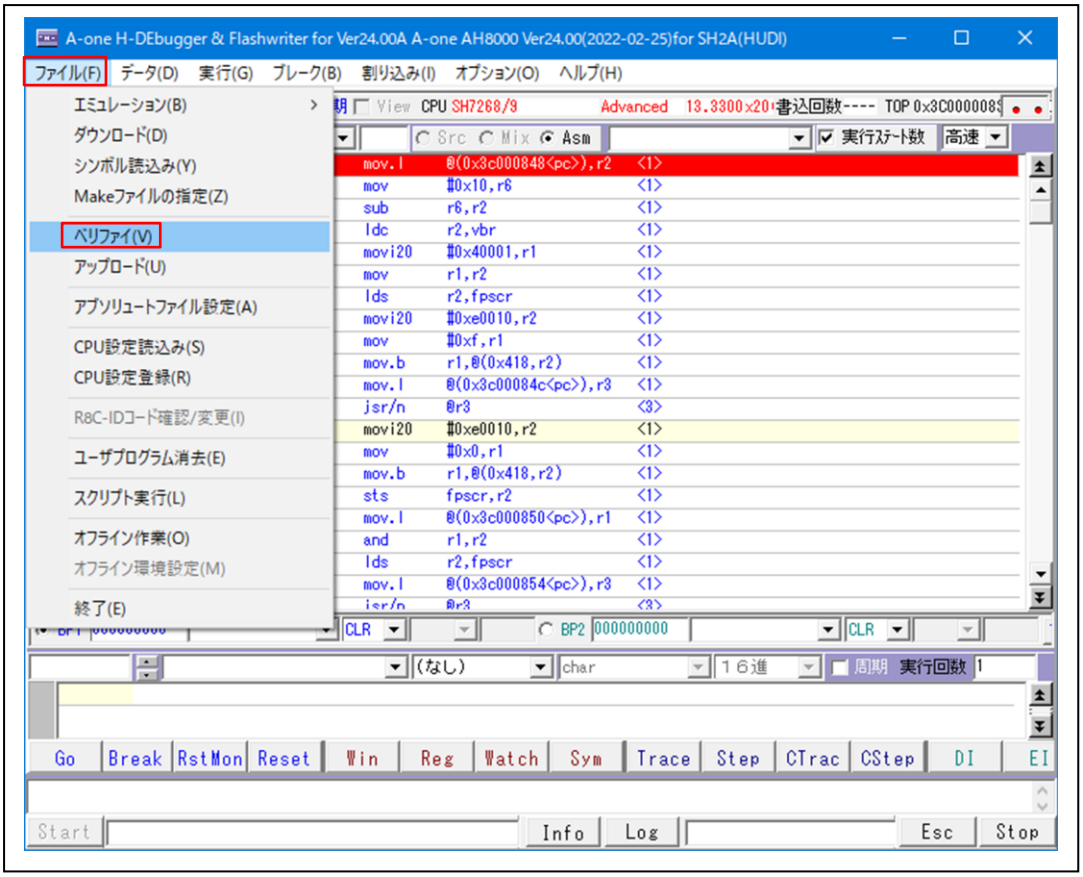


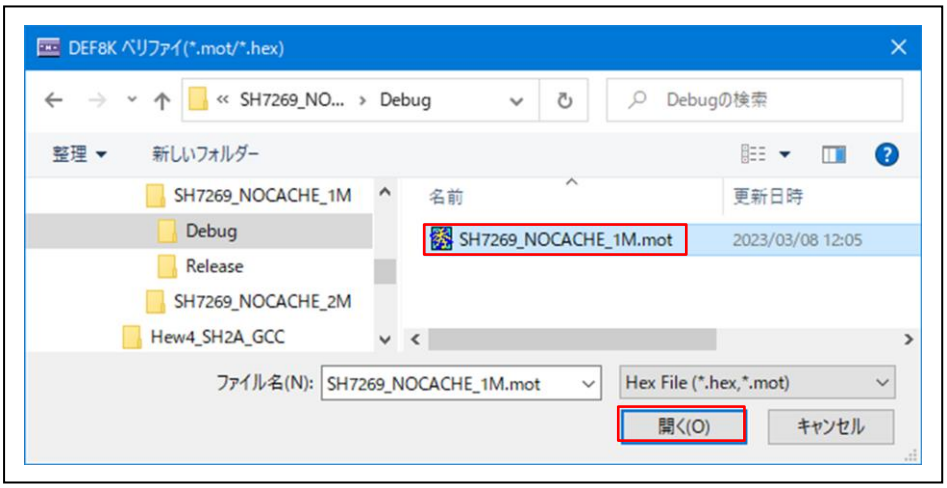
7－3）「アプリケーションプログラム」が正常に書き込みされたかベリファイで確認する。
【条件】この確認は、新規ローダープログラムが完成していることが条件になります。

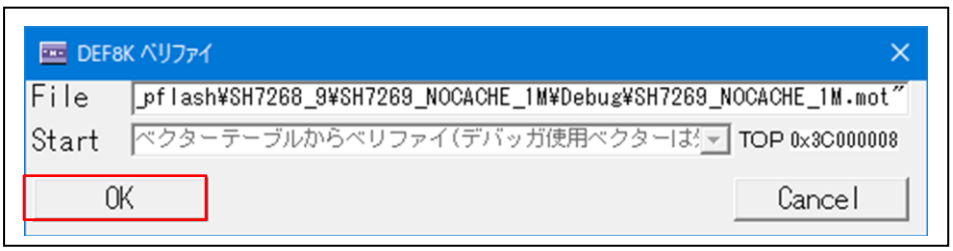
1）ターゲットの電源を再投入後、ロード時間を待ってから接続する。
（参考）1.0M バイトコードのロード時間は約 5 秒



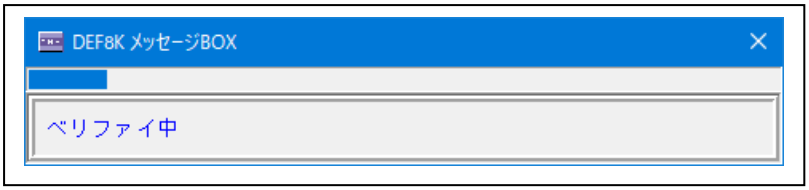
2）内蔵 RAM とベリファイする。



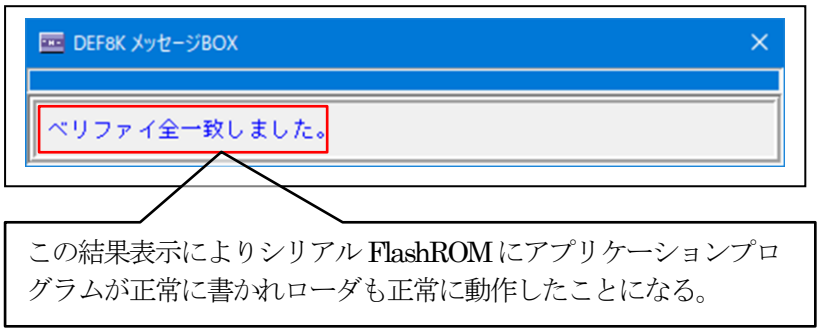




【ベリファイ中】



【ベリファイ結果】



この結果表示によりシリアル FlashROM にアプリケーションプログラムが正常に書かれローダも正常に動作したことになる。

8．注意事項

・本文書の著作権は、エーワン（株）が保有します。
・本文書を無断での転載は一切禁止します。
・本文書に記載されている内容についての質問やサポートはお受けすることが出来ません。
・本文章に関して、ルネサス　エレクトロニクス社への問い合わせは御遠慮願います。
・本文書の内容に従い、使用した結果、損害が発生しても、弊社では一切の責任は負わないものとします。
・本文書の内容に関して、万全を期して作成しましたが、ご不審な点、誤りなどの点がありましたら弊社までご連絡くだされば幸いです。
・本文書の内容は、予告なしに変更されることがあります。

〒486-0852
愛知県春日井市下市場町 6-9-20
エーワン株式会社
https://www.aone.co.jp