

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

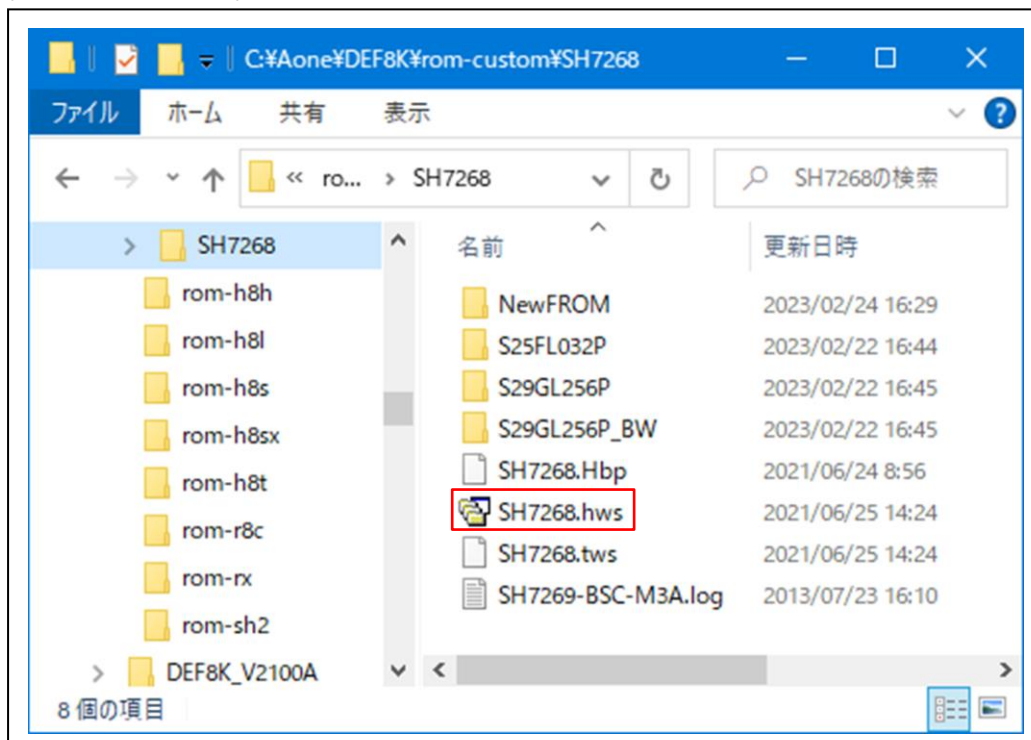
- ROM レス品種が対象です。(SH7262/7264/7266/7267/7268/7269)

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

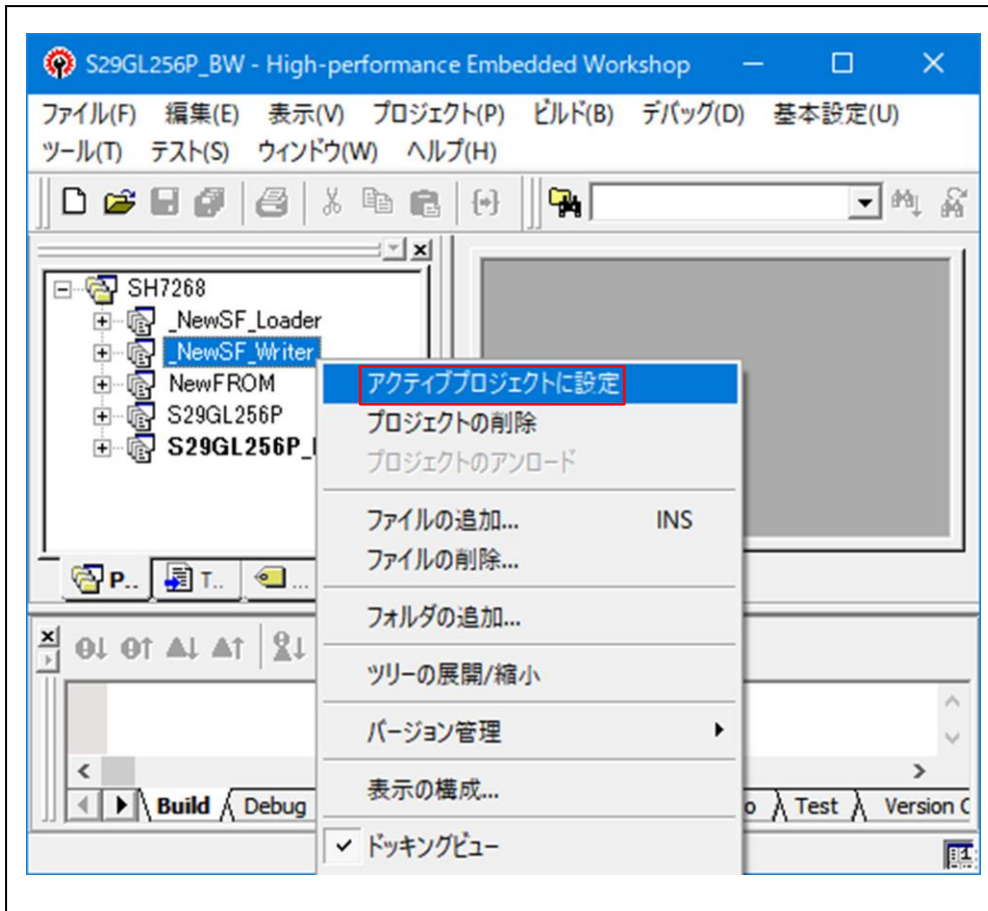
3-1) AH8000 コントロールソフト(DEF8K)のインストール DIR にあるワークスペースを開く。
(SH7268 での追加例)



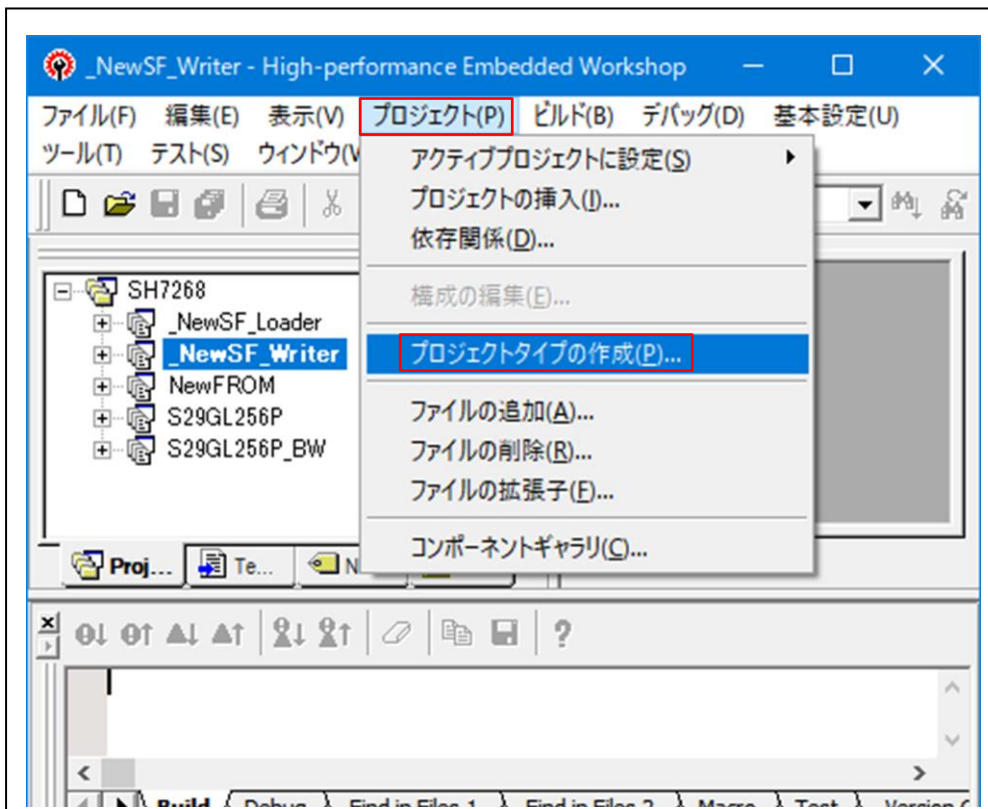
<default directory>

"c:\Aone\DEF8K\rom-custom\SH7268"

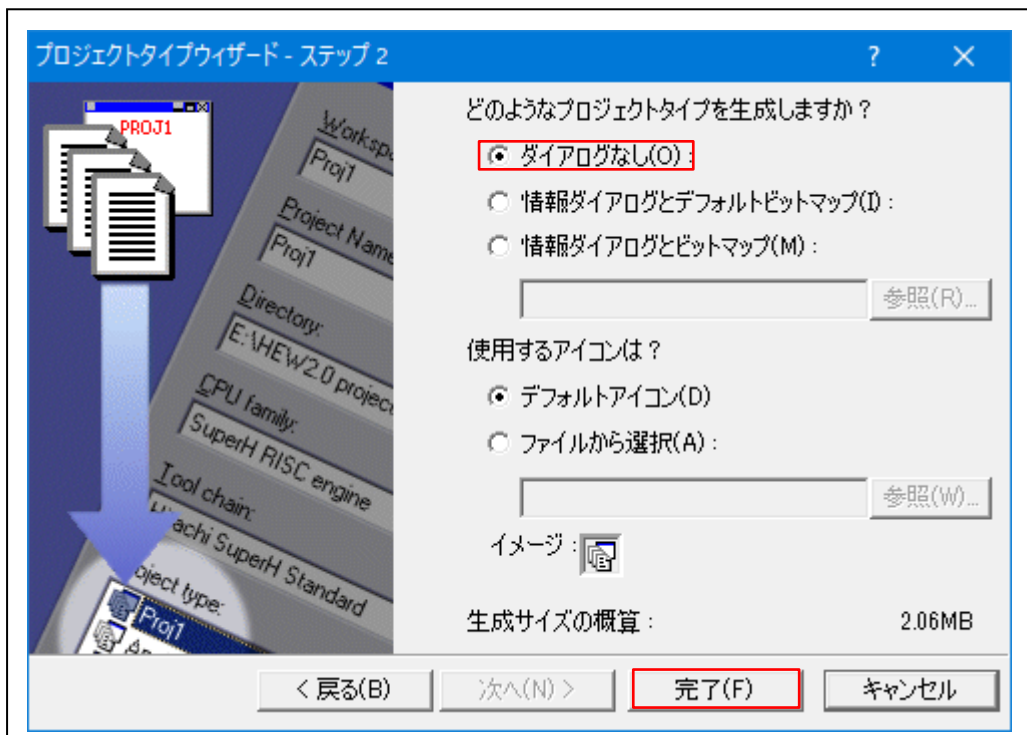
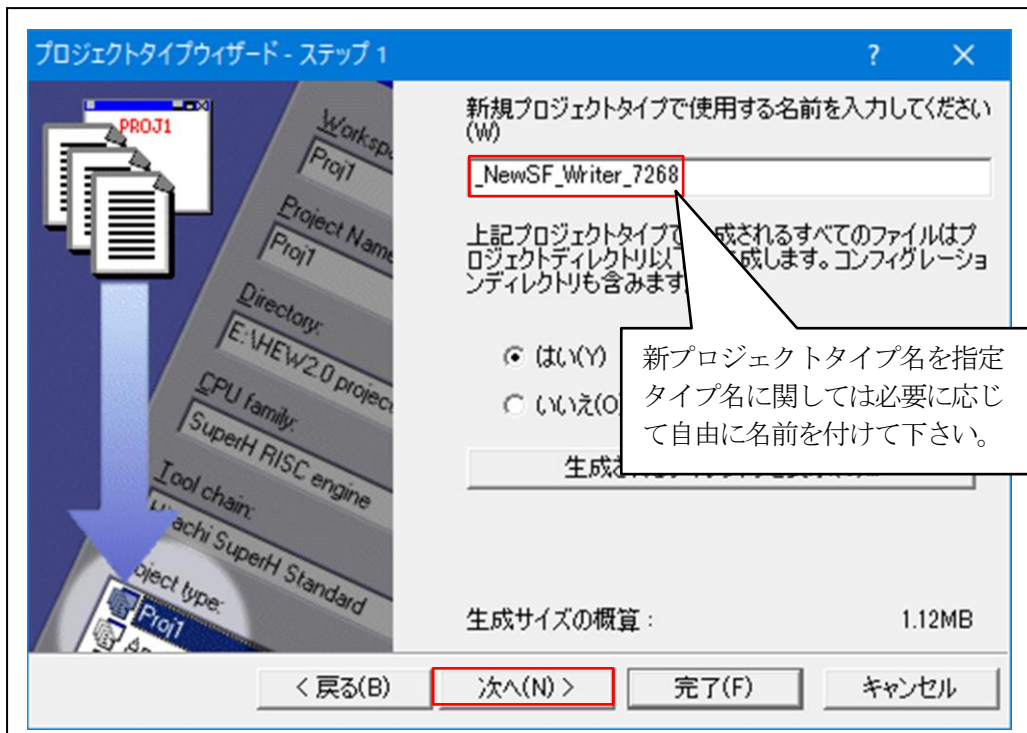
3-2) プロジェクト名「_NewSF_Writer」をアクティブプロジェクトにする。



3-3) 「プロジェクトタイプの作成」を指定する。

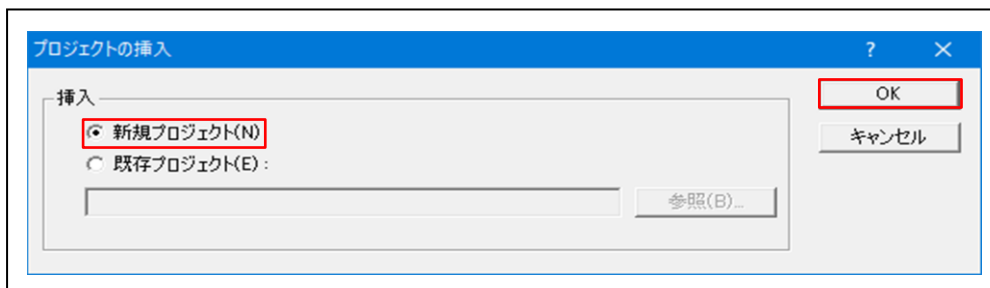
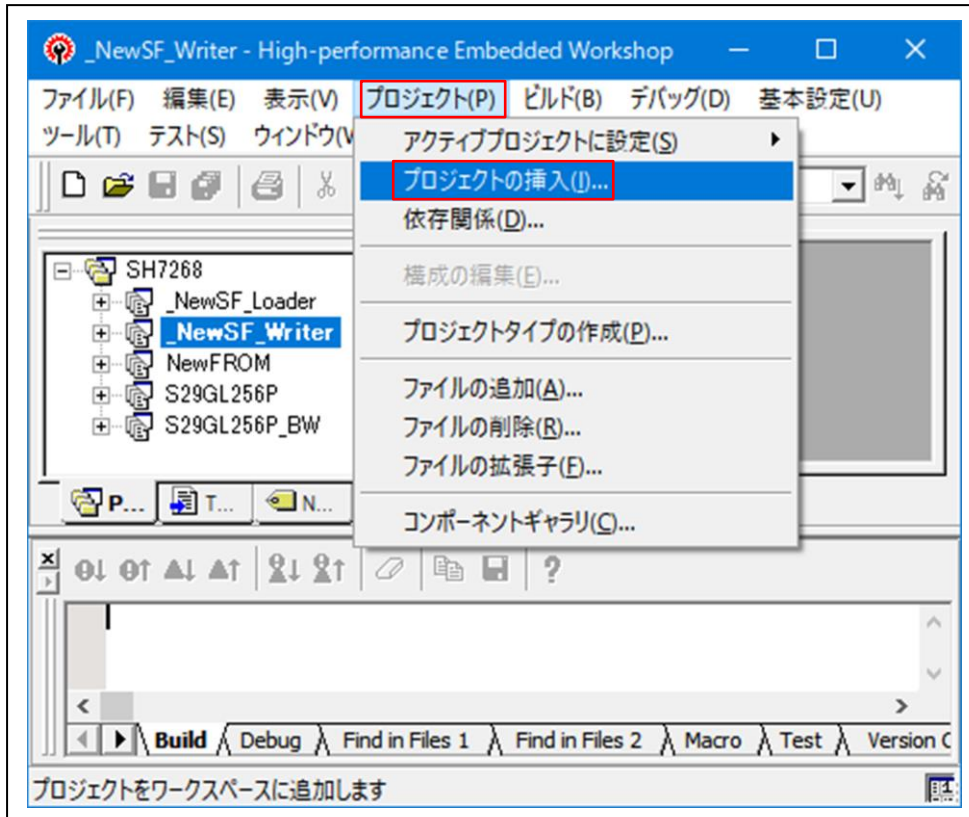


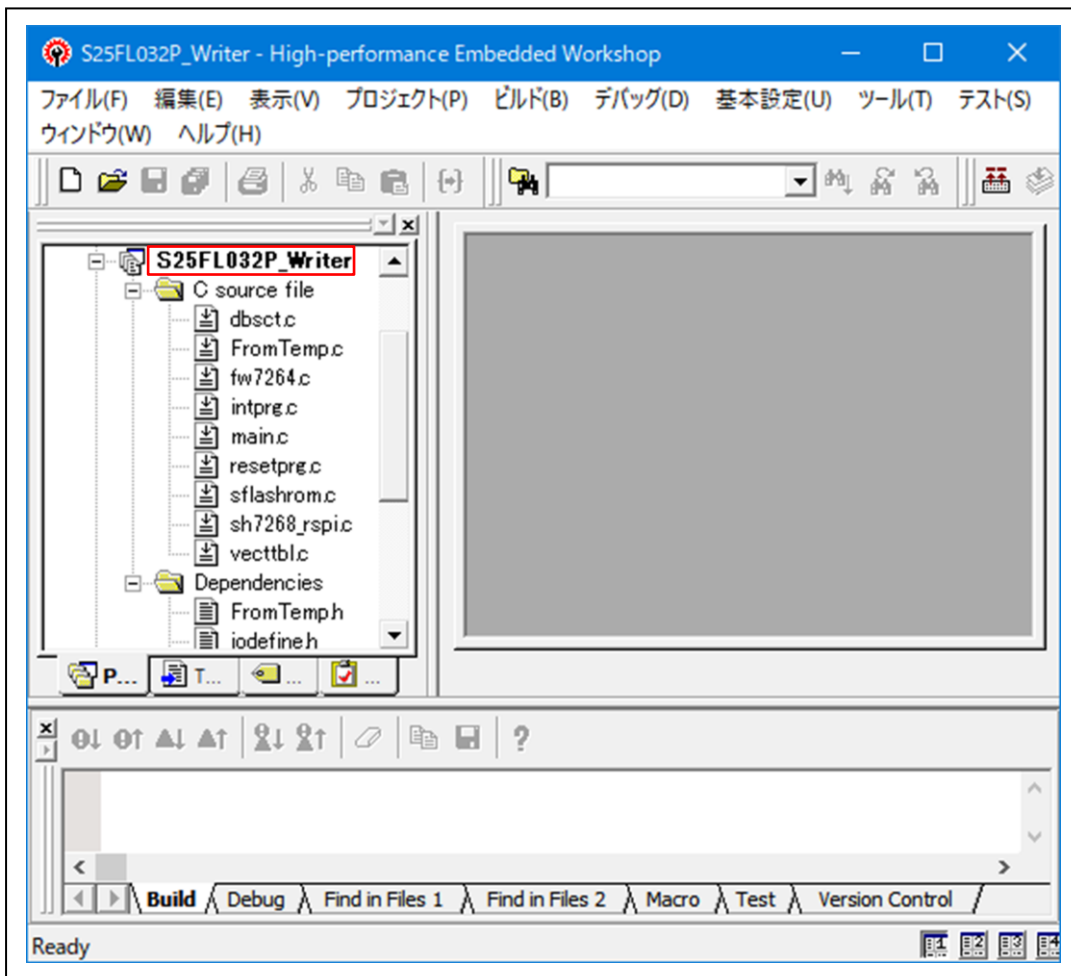
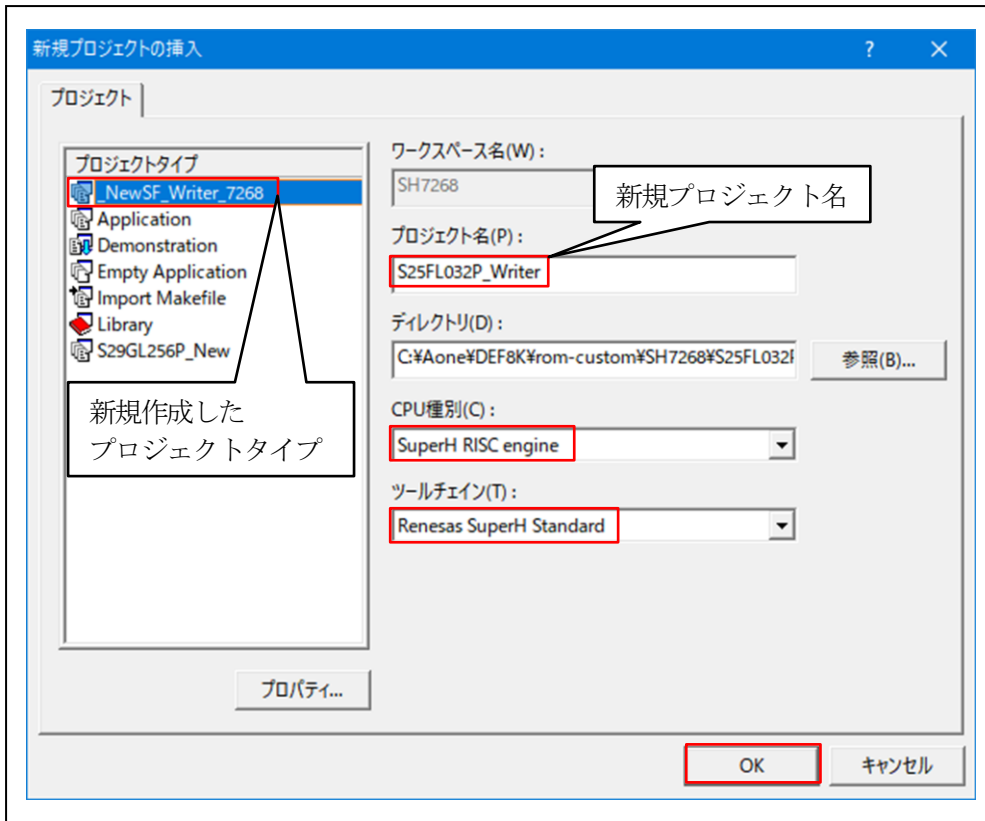
3-4) 追加FlashROMの新プロジェクトタイプを作成する。



4. シリアル FlashROM 品種の追加

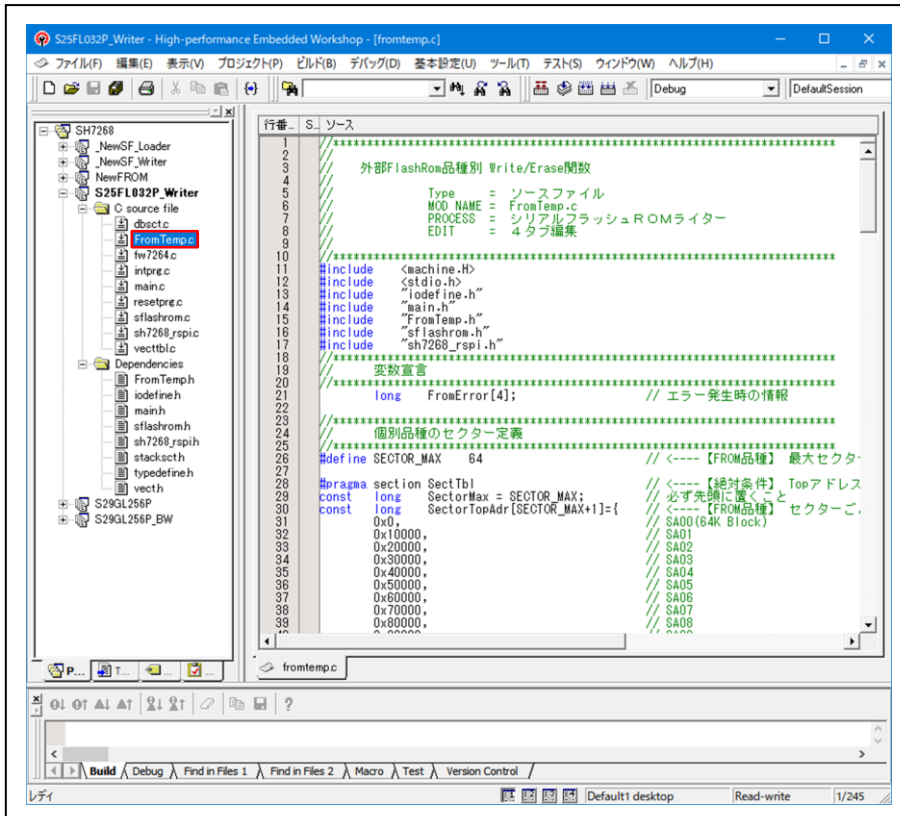
4-1) 新規プロジェクトの挿入



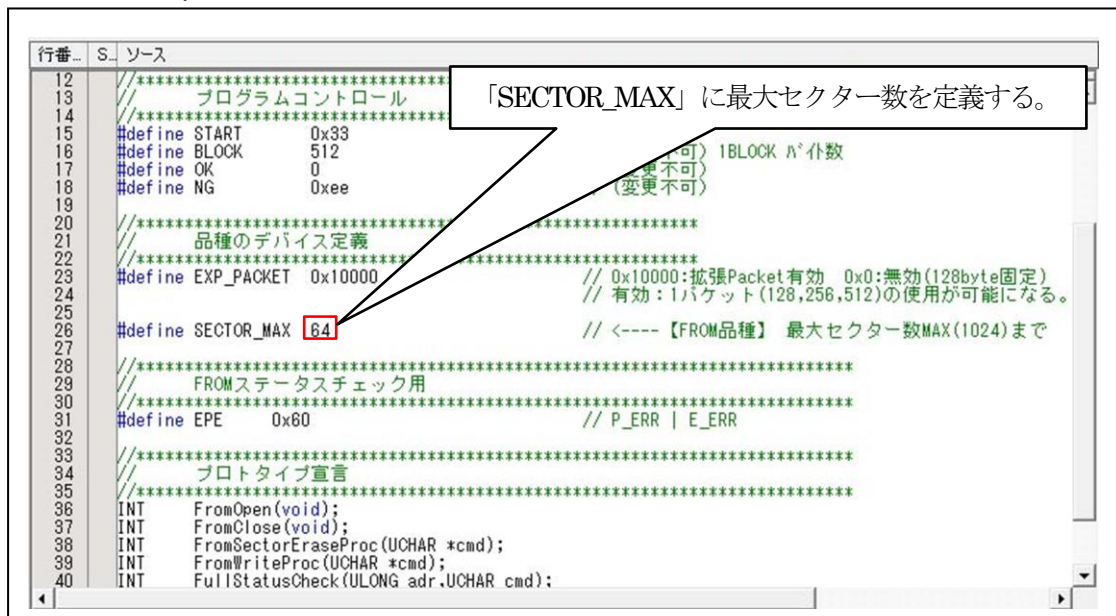


4-2) シリアル FlashROM 品種の定義およびプログラムを作成する。

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。

行番...	S...	ソース
31		
32		#pragma section SectTbl
33	const long	SectorMax = (SECTOR_MAX EXP_PACKET);
34	const long	SectorTopAddr[SECTOR_MAX+1]=
35		
36		0x0, // SA00 (64
37		0x10000, // SA01
38		0x20000, // SA02
39		0x30000, // SA03
40		0x40000, // SA04
41		0x50000, // SA05
42		0x60000, // SA06
43		0x70000, // SA07
44		0x80000, // SA08
45		0x90000, // SA09
46		0xA0000, // SA10
47		0xB0000, // SA11
48		0xC0000, // SA12
49		0xD0000, // SA13
50		0xE0000, // SA14
51		0xF0000, // SA15
52		0x100000, // SA16
53		0x110000, // SA17
54		0x120000, // SA18
55		0x130000, // SA19
56		0x140000, // SA20
57		0x150000, // SA21
58		0x160000, // SA22
59		0x170000, // SA23
60		0x180000, // SA24
61		0x190000, // SA25
62		0x1A0000, // SA26
63		0x1B0000, // SA27
64		0x1C0000, // SA28
65		0x1D0000, // SA29
66		0x1E0000, // SA30
67		0x1F0000, // SA31

⚠️ 必ず先頭に定義する。
(SECTOR_MAX | EXP_PACKET)
EXP_PACKETをOR定義することにより拡張パケット通信が有効になる。

変数「SectorTopAddr[]」に、セクター（ブロック）数ごとの先頭アドレスを定義する。

行番...	S...	ソース
63		0x200000, // SA32
64		0x210000, // SA33
65		0x220000, // SA34
66		0x230000, // SA35
67		0x240000, // SA36
68		0x250000, // SA37
69		0x260000, // SA38
70		0x270000, // SA39
71		0x280000, // SA40
72		0x290000, // SA41
73		0x2A0000, // SA42
74		0x2B0000, // SA43
75		0x2C0000, // SA44
76		0x2D0000, // SA45
77		0x2E0000, // SA46
78		0x2F0000, // SA47
79		0x300000, // SA48
80		0x310000, // SA49
81		0x320000, // SA50
82		0x330000, // SA51
83		0x340000, // SA52
84		0x350000, // SA53
85		0x360000, // SA54
86		0x370000, // SA55
87		0x380000, // SA56
88		0x390000, // SA57
89		0x3A0000, // SA58
90		0x3B0000, // SA59
91		0x3C0000, // SA60
92		0x3D0000, // SA61
93		0x3E0000, // SA62
94		0x3F0000, // SA63
95		0x400000, // <---- 【FROM品種】最終セクターアドレス+1を登録
96		};
97		

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()」RSPIO 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。

```

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。

```

166 *****
167 FromWriteProc(CHAR *cmd)
168  伝送フォーム(in)
169  cmd[0] == %
170  [1] == 0
171  [2] == CommandSize[2]以降[Sum]も含む
172  [3] == Command('W')
173  [4] == WriteAddress(HH)
174  [5] == WriteAddress(MH)
175  [6] == WriteAddress(ML)
176  [7] == WriteAddress(LL)
177  [8+0] == WriteData[0]
178  .
179  ==
180  [8+127] == WriteData[127] 固定 余分エリア(0xff)
181  [Sum] == [3]->[Sum-1]の加算
182  return()
183  OK == 正常終了
184  NG == 異常終了
185 *****
186 INT FromWriteProc(UCHAR *cmd)
187 {
188  ULONG adr; // 先頭アドレス
189  UCHAR *data;
190  UCHAR verf[BLOCK];
191  INT i;
192  INT stat;
193
194  adr = (((LONG)cmd[4] << 24) & 0xff000000) |
195        (((LONG)cmd[5] << 16) & 0x00ff0000) |
196        (((LONG)cmd[6] << 8) & 0x0000ff00) |
197        ((LONG)cmd[7] & 0x000000ff);
198
199  data = (UCHAR *)&cmd[8]; // Data Adr set
200  sFlashByteProgram(adr,data,BLOCK); // 指定バイト数の書き込み
201  stat = FullStatusCheck(adr,'W');
202  if (stat == NG) return(NG);
203
204  sFlashByteRead(adr,verf,BLOCK); // ベリファイテスト
205  for(i = 0;i < BLOCK;i++){ // VerifyTest
206    if (verf[i] != data[i]){ // VerifyError
207      FromErrorSet((long)'R',adr,(long)data[i]);
208      return(NG);
209    }
210    ++adr;
211  }
212  return(OK);
213
214 *****
  
```

書き込みアドレスの算出

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 FromError[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
222 return()
223 OK == 正常終了
224 NG == 異常終了
225 *****
226 INT FullStatusCheck(ULONG adr,UCHAR cmd)
227 {
228     UCHAR stat;
229     FromErrorSet(0,0,0); // Error情報を消去
230     stat = sFlashReadStatus(); // Error発生
231     if (stat & EPE) {
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

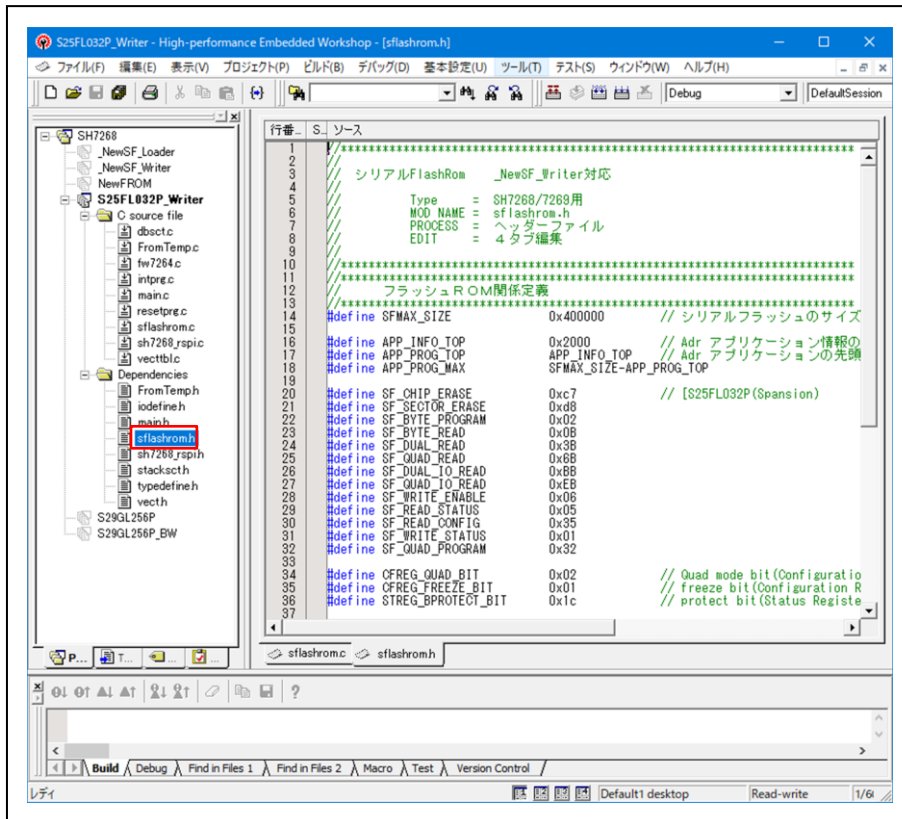
char cmd; //処理しているコマンド
 // 'O' = オープンコマンド
 // 'C' = クローズコマンド
 // 'D' = イレーズコマンド
 // 'W' = 書き込みコマンド
 // 'R' = ベリファイコマンド

<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。

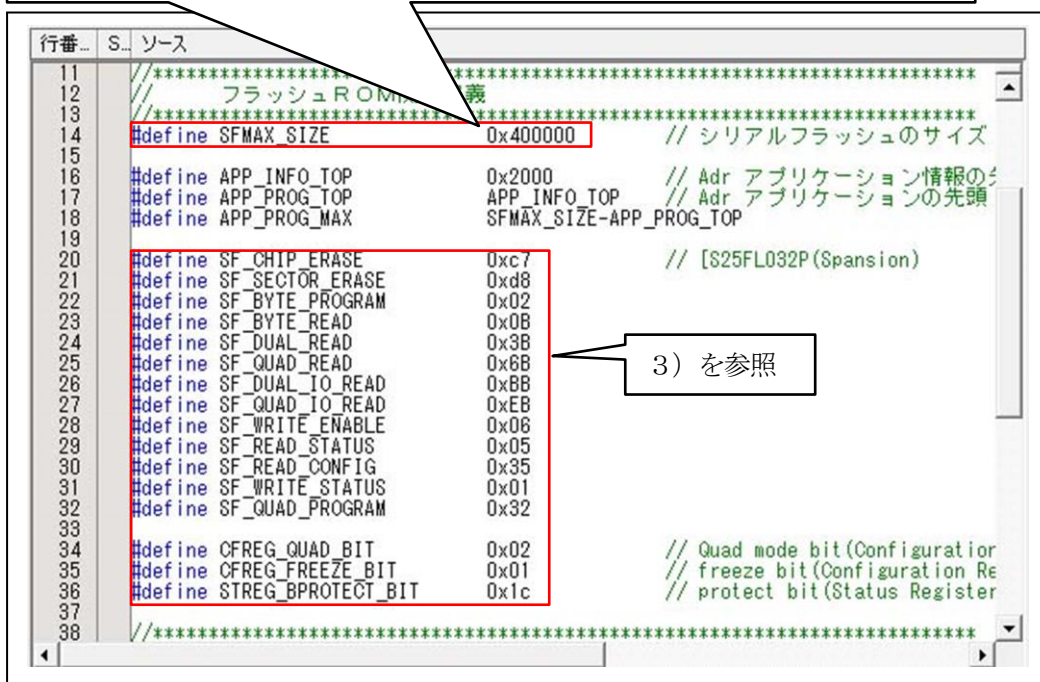
4-3) シリアル FlashROM に依存した定義およびコマンドプログラムを作成する。

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。



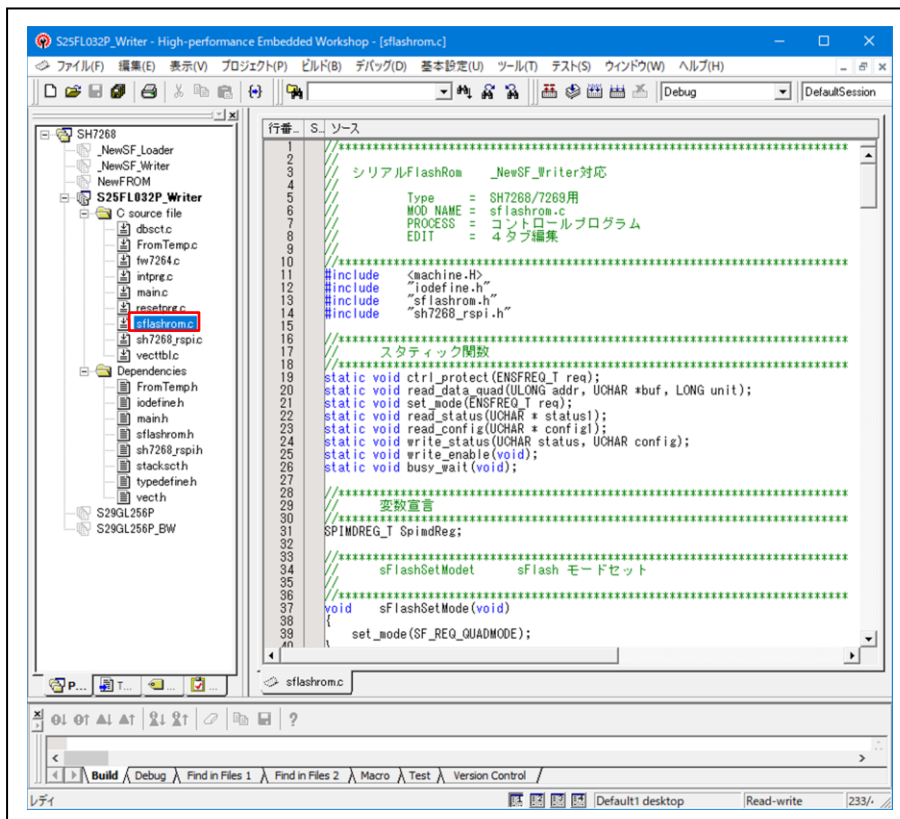
3) 「sflashrom.h」の#defineの定義説明。

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		read_status(&stat);
204		return(stat);
205	}	
206		
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 void  sFlashChipErase(void)
61 {
62     write_enable();                // WREN Command
63
64     SpimdReg.cdb = SPIBSC_1BIT;    // Commmand bit-width = Single
65     SpimdReg.cde = SPIBSC_OUTPUT_ENABLE; // Command Enable
66     SpimdReg.ocde = SPIBSC_OUTPUT_DISABLE; // Optional-Command Disable
67     SpimdReg.ade = SPIBSC_OUTPUT_DISABLE; // Address Disable
68     SpimdReg.opde = SPIBSC_OUTPUT_DISABLE; // Option-Data Disable
69     SpimdReg.spide = SPIBSC_OUTPUT_DISABLE; // Disable
70     SpimdReg.sskp = SPIBSC_SPISSL_NEGATE; // Negate after transfer
71     SpimdReg.spire = SPIBSC_SPIDATA_DISABLE; // Data Access (Read Disable)
72     SpimdReg.spive = SPIBSC_SPIDATA_DISABLE; // Data Access (Write Disable)
73     SpimdReg.cmd = SF_CHIP_ERASE; // BE: Bulk Erase
74
75     IoTransferSPIBSC(&SpimdReg);
76
77     busy_wait();
78 }
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 void  sFlashSectorErase(ULONG adr)
84 {
85     write_enable();                // WREN Command
86
87     SpimdReg.cdb = SPIBSC_1BIT;    // Commmand bit-width = Single
88     SpimdReg.adb = SPIBSC_1BIT;    // Address bit-width = Single
89     SpimdReg.cde = SPIBSC_OUTPUT_ENABLE; // Command Enable
90     SpimdReg.ocde = SPIBSC_OUTPUT_DISABLE; // Optional-Command Disable
91     SpimdReg.ade = SPIBSC_OUTPUT_ADDR_24; // Enable(Adr[23:0])
92     SpimdReg.opde = SPIBSC_OUTPUT_DISABLE; // Option-Data Disable
93     SpimdReg.spide = SPIBSC_OUTPUT_DISABLE; // Disable
94     SpimdReg.sskp = SPIBSC_SPISSL_NEGATE; // Negate after transfer
95     SpimdReg.spire = SPIBSC_SPIDATA_DISABLE; // Data Access (Read Disable)
96     SpimdReg.spive = SPIBSC_SPIDATA_DISABLE; // Data Access (Write Disable)
97     SpimdReg.cmd = SF_SECTOR_ERASE; // SE: Sector Erase
98     SpimdReg.addr = adr;           // dont care in dual mode
99                                     // because address is calculat
100
101     IoTransferSPIBSC(&SpimdReg);
102
103     busy_wait();
104 }
105 //*****
  
```

1) 「sFlashByteProgram 0」 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。

```

105 //*****
106 //      sFlashByteProgram   指定バイト数の書き込み
107 //*****
108 void   sFlashByteProgram(ULONG adr,UCHAR *wdat,ULONG wsize)
109 {
110     LONG unit;
111
112     write_enable();           // WREN Command
113     SpimdReg.cdb = SPIBSC_1BIT; // Command 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     // Quad
129     SpimdReg.spidb = SPIBSC_4BIT; // Command Disable
130     SpimdReg.cde = 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 Enable)
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 //*****
  
```

1 2) 「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

Address	Section
0xFFFF80000	DVECTTBL
	DINTTBL
0xFFFF80800	CSectTbl
	PRresetPRG
	PIntPRG
	P
	PFromPRG
	C
	C\$BSEC
	C\$DSEC
	D
	B
	R
0xFFFF83C00	S

<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

The screenshot shows the S25FL032P_Writer IDE interface. The left pane displays a project tree with the file 'S25FL032P_Writer' selected. The main editor displays the source code for 'fromtemp.c', which includes headers, defines, and constants for flashrom operations. The bottom pane shows the build output, indicating a successful build with no errors or warnings.

```

1  //*****
2  //
3  // 外部FlashRom品種別 Write/Erase関数
4  //
5  //      Type = ソースファイル
6  //      MOD_NAME = FromTemp.c
7  //      PROCESS = シリアルフラッシュROMライター
8  //      EDIT = 4タブ編集
9  //*****
10 //*****
11 #include <machine.H>
12 #include <stdio.h>
13 #include "iodefine.h"
14 #include "main.h"
15 #include "FromTemp.h"
16 #include "sflashrom.h"
17 #include "sh7268_rspi.h"
18 //*****
19 // 変数宣言
20 //*****
21 long FromError[4]; // エラー発生時の情報
22 //*****
23 // 個別品種のセクター定義
24 //*****
25 #define SECTOR_MAX 64 // <---- 【FROM品種】 最大セクタ
26 //*****
27 #pragma section SectTbl // <---- 【絶対条件】 Topアドレス
28 const long SectorMax = SECTOR_MAX; // <---- 必ず先頭に置くこと
29 const long SectorTopAdr[SECTOR_MAX+1]={ // <---- 【FROM品種】 セクターご
30     0x0, // SA00 (64K Block)
31     0x10000, // SA01
32     0x20000, // SA02
33 }

```

Build Output:

```

Phase OptLinker starting
Nothing to do - skipping
Phase OptLinker finished
Build Finished
0 Errors, 0 Warnings

```

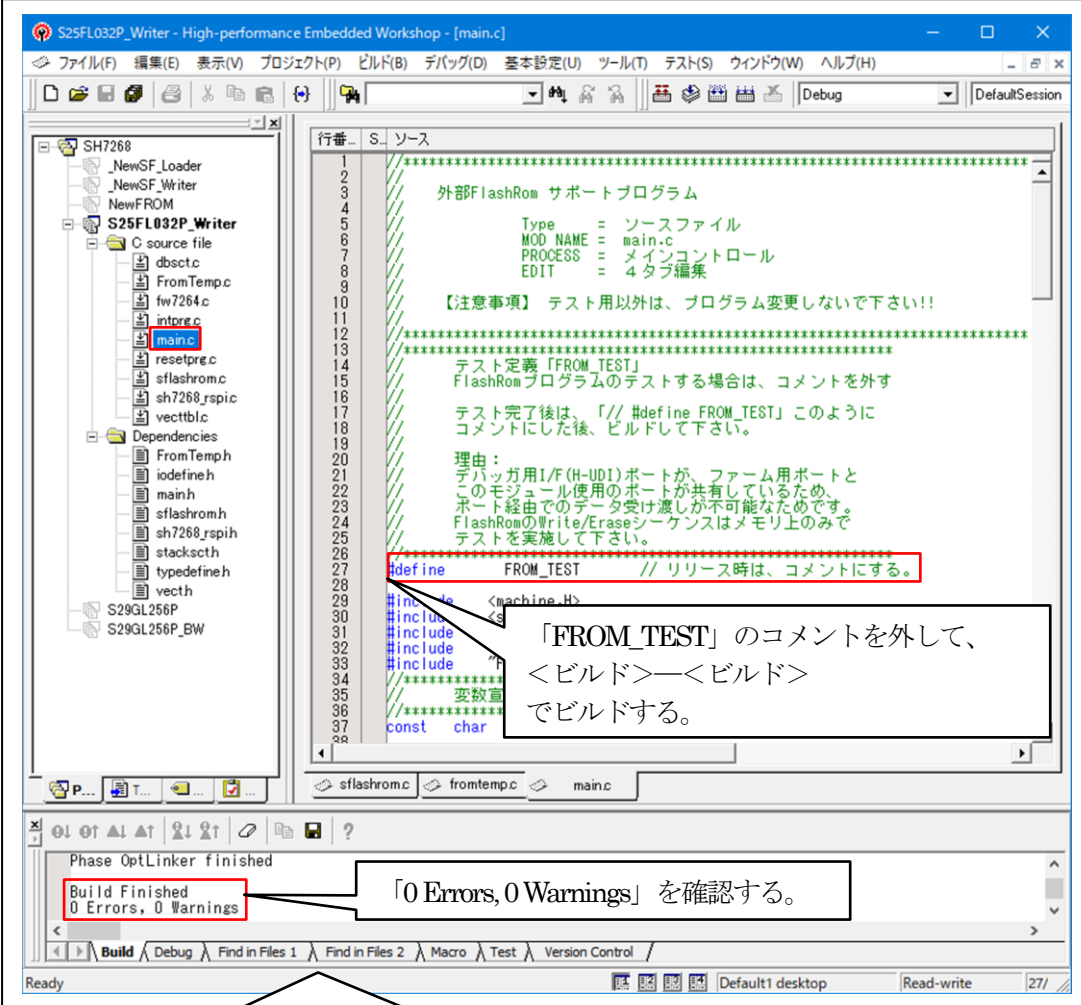
「0 Errors, 0 Warnings」を確認する。

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。



The screenshot shows the IDE interface for the S25FL032P_Writer project. The left pane displays the project tree with 'main.c' selected under the 'C source file' folder. The main editor shows the source code for 'main.c', which includes a preprocessor directive to define 'FROM_TEST' for testing purposes. A red box highlights the line: `#define FROM_TEST // リリース時は、コメントにする。`. A callout box points to this line with the text: 「FROM_TEST」のコメントを外して、<ビルド>—<ビルド>でビルドする。 The bottom status bar shows the build output: 'Phase OptLinker finished' and 'Build Finished 0 Errors, 0 Warnings', with a red box around the latter. A callout box points to this output with the text: 「0 Errors, 0 Warnings」を確認する。

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。

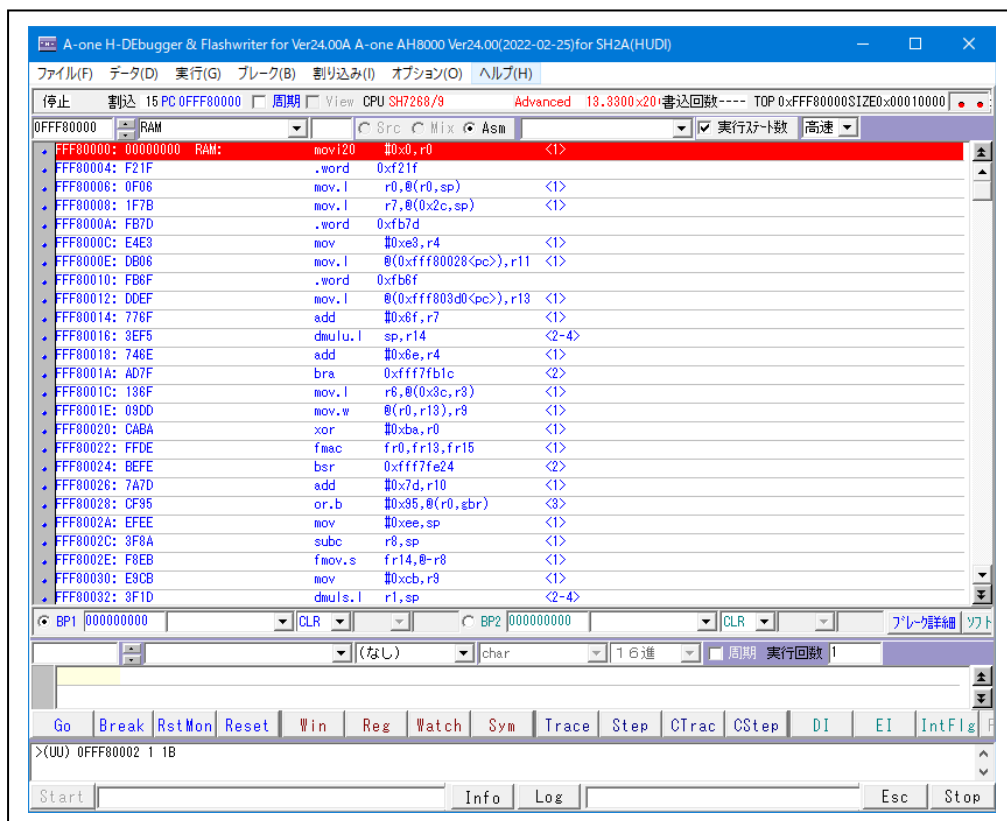
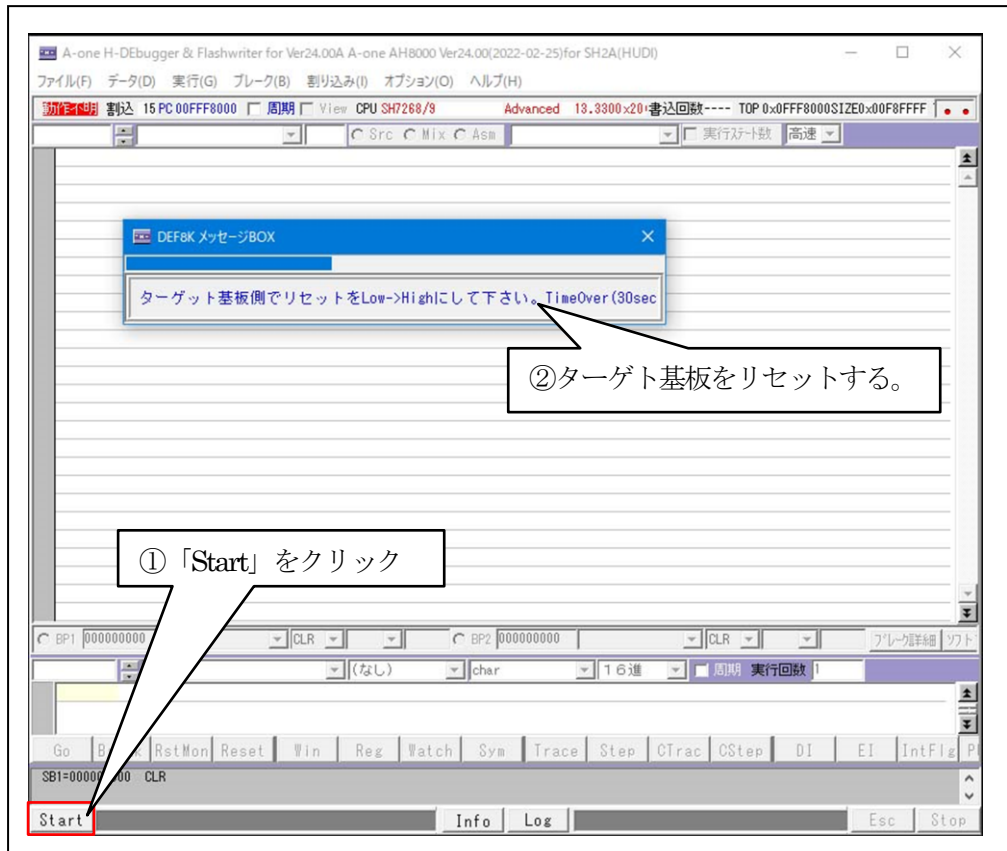
ターゲット側の基板構成による

デバッグ時は、「Soft-TAP」を指定する。

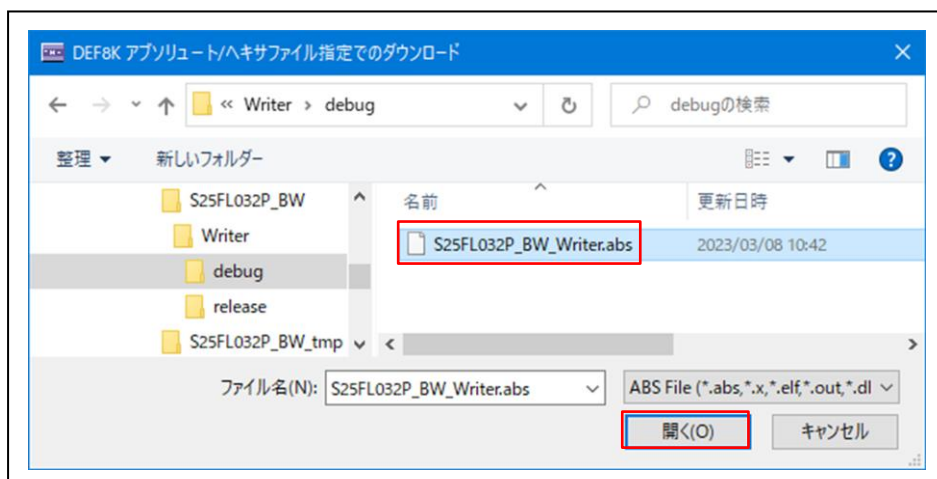
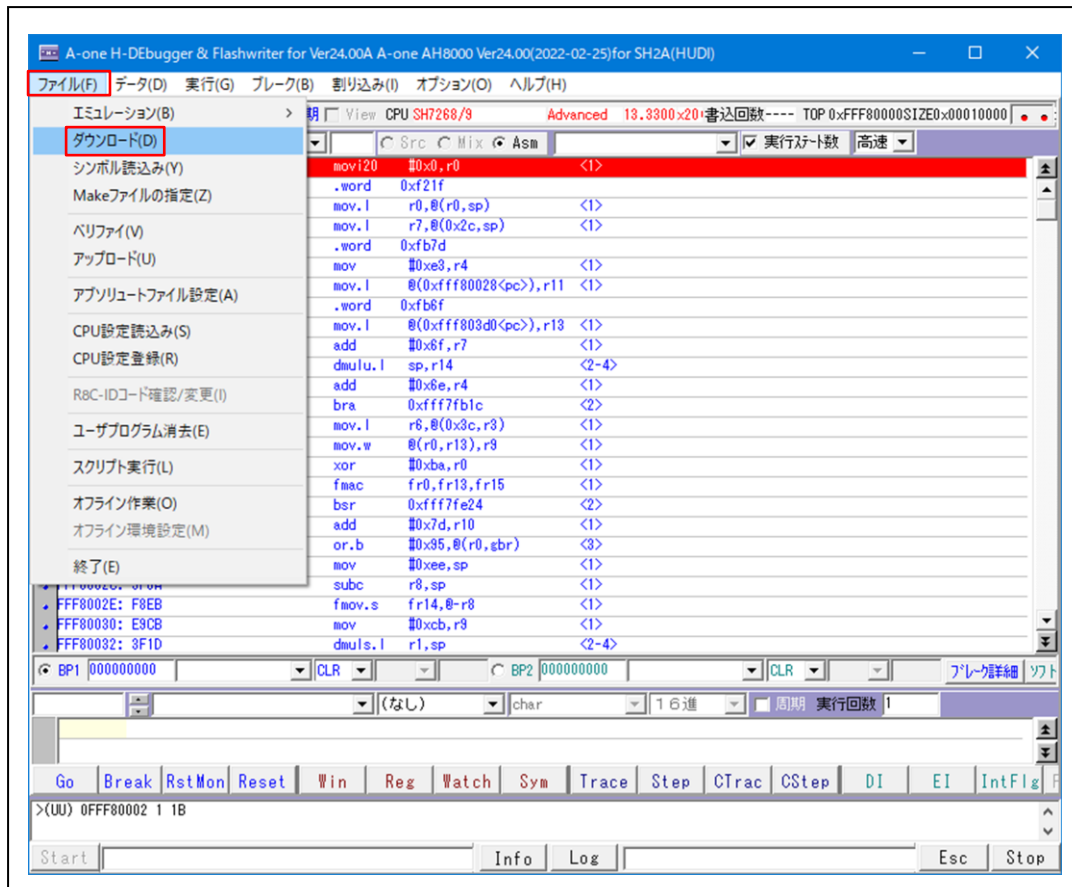
作成した「FlashROM」プログラムのデバッグにBSC設定が必要な場合は、スクリプトファイルを指定する。
SH7268の場合
"c:\s\Aone\DEF8K\rom-custom\SH7268"に、
例として「SH7268-BSC.log」を用意してあるので、目的ターゲット基板にカスタマイズして下さい。

5-4) 「DEF8K」 でデバッグを開始する。

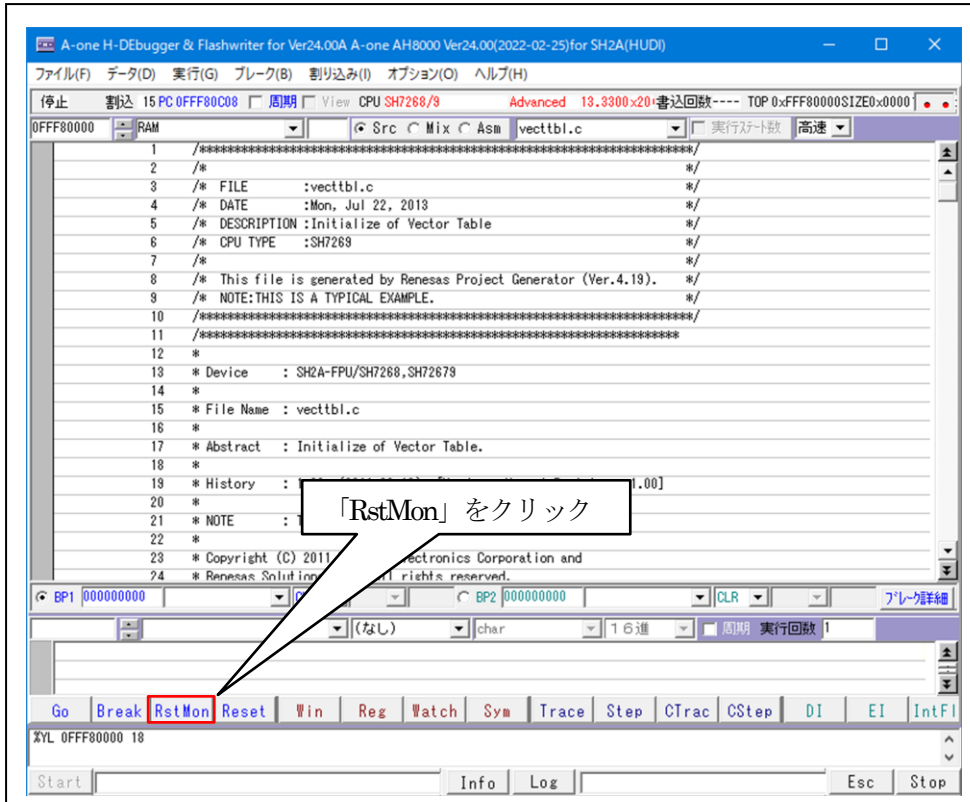
1) ターゲットと接続する。



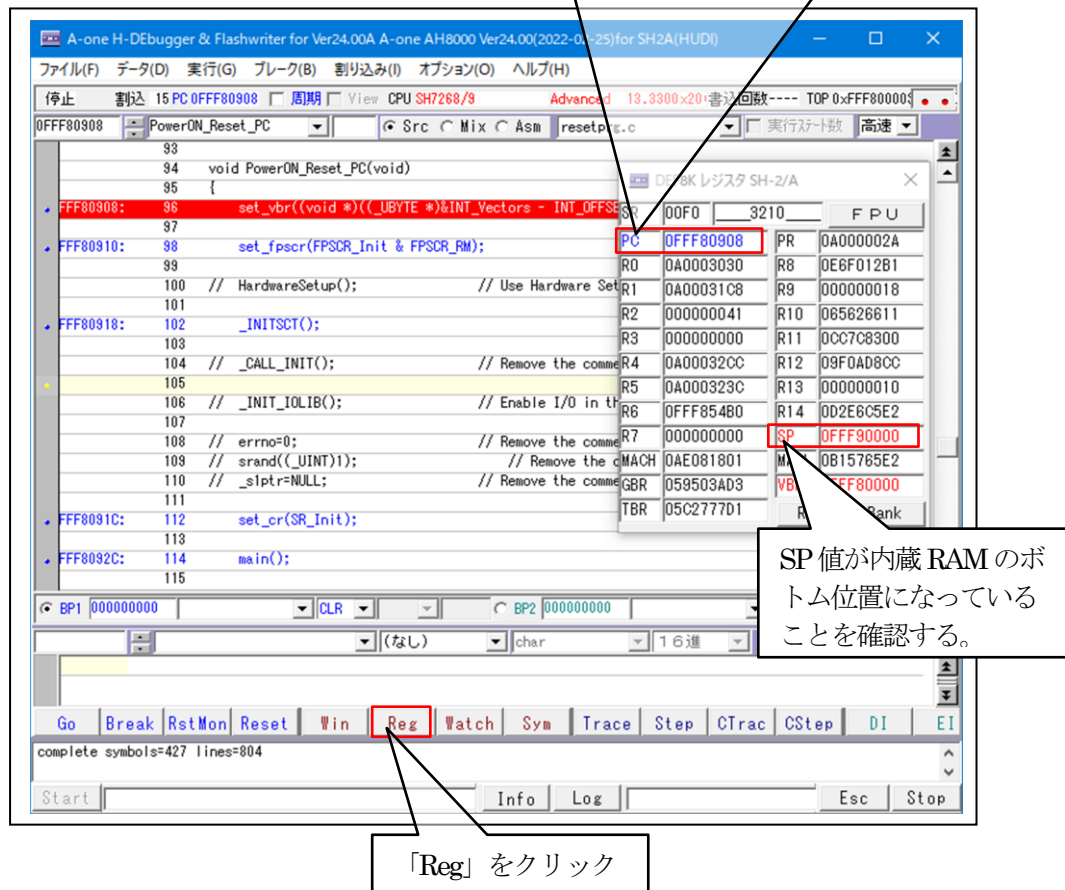
2) シリアルFlashROMプログラムをダウンロードする。



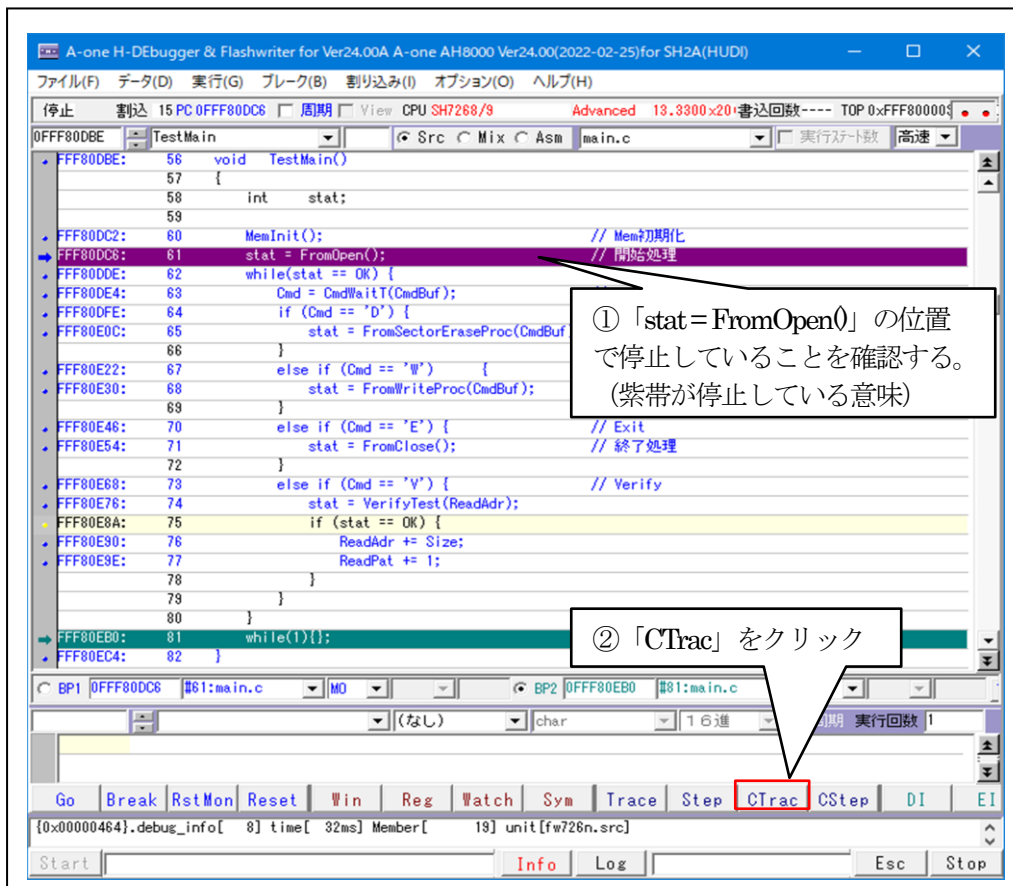
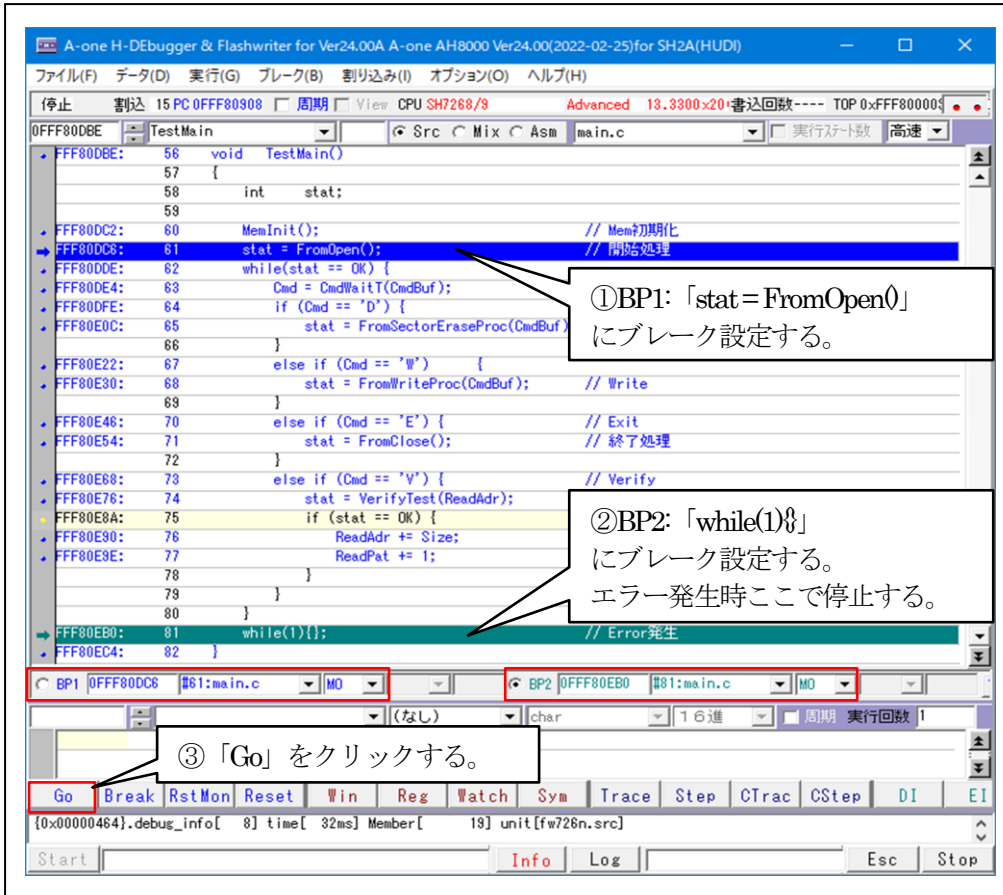
3) 「RstMon」 をクリックする。



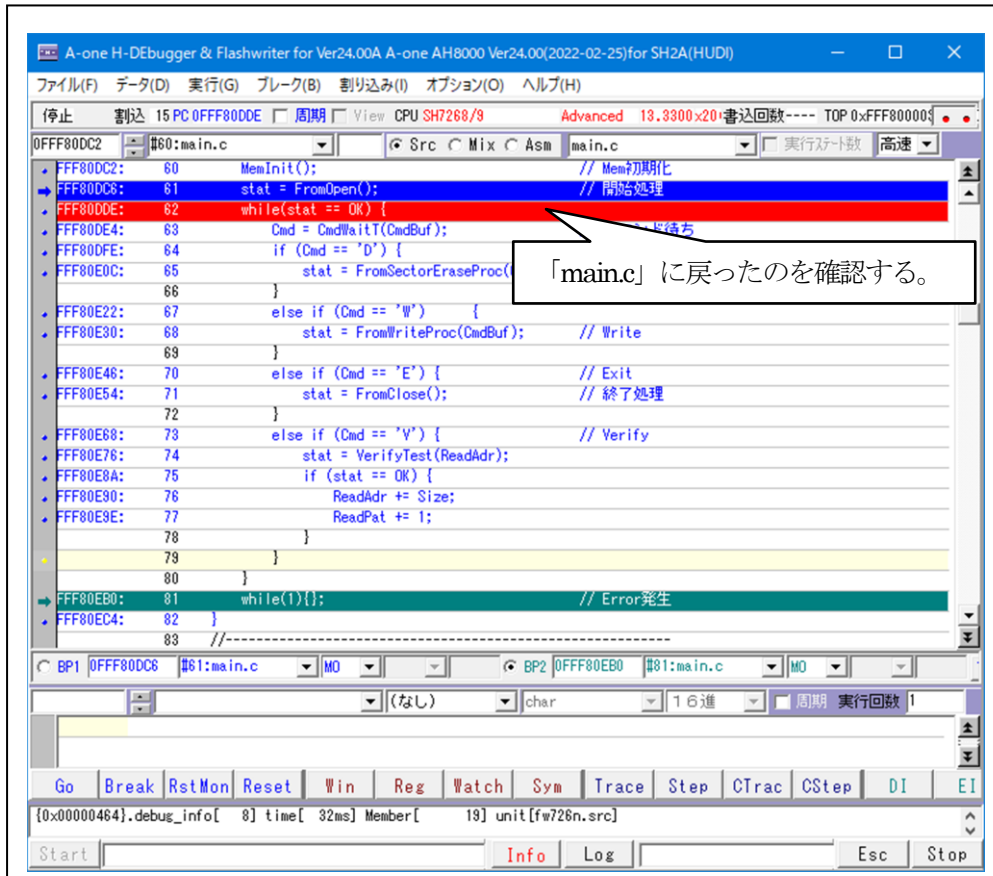
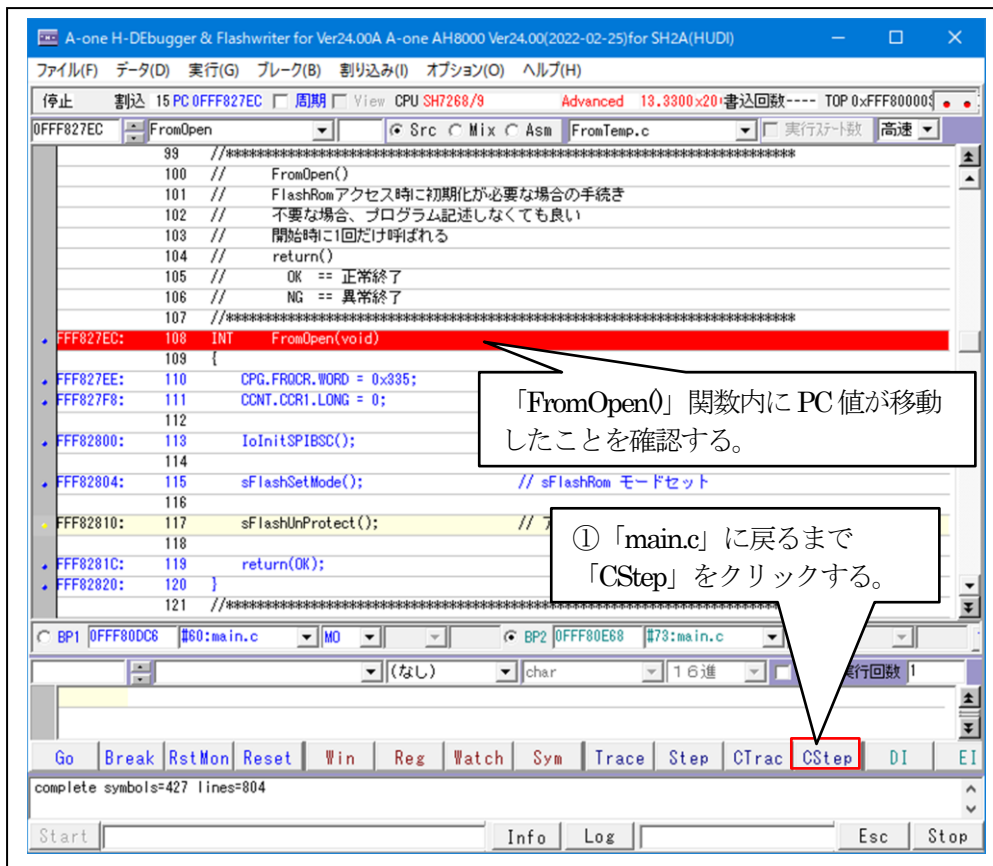
「void PowerON_Rest_PC」関数の先頭のPC値になっている事を確認する。



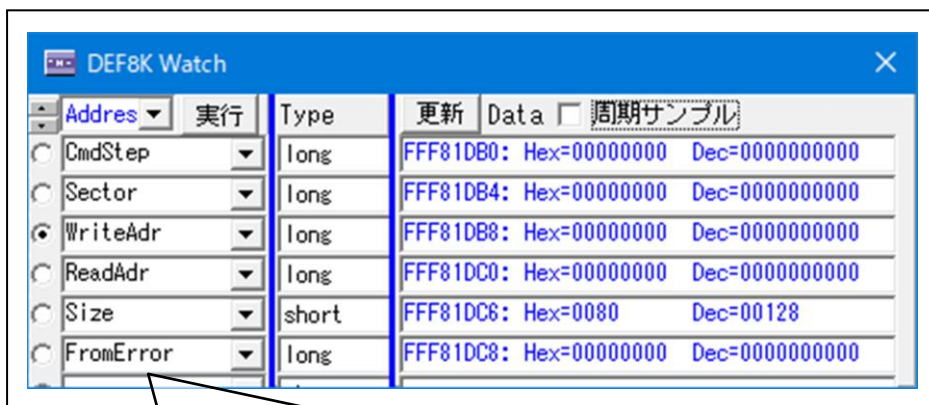
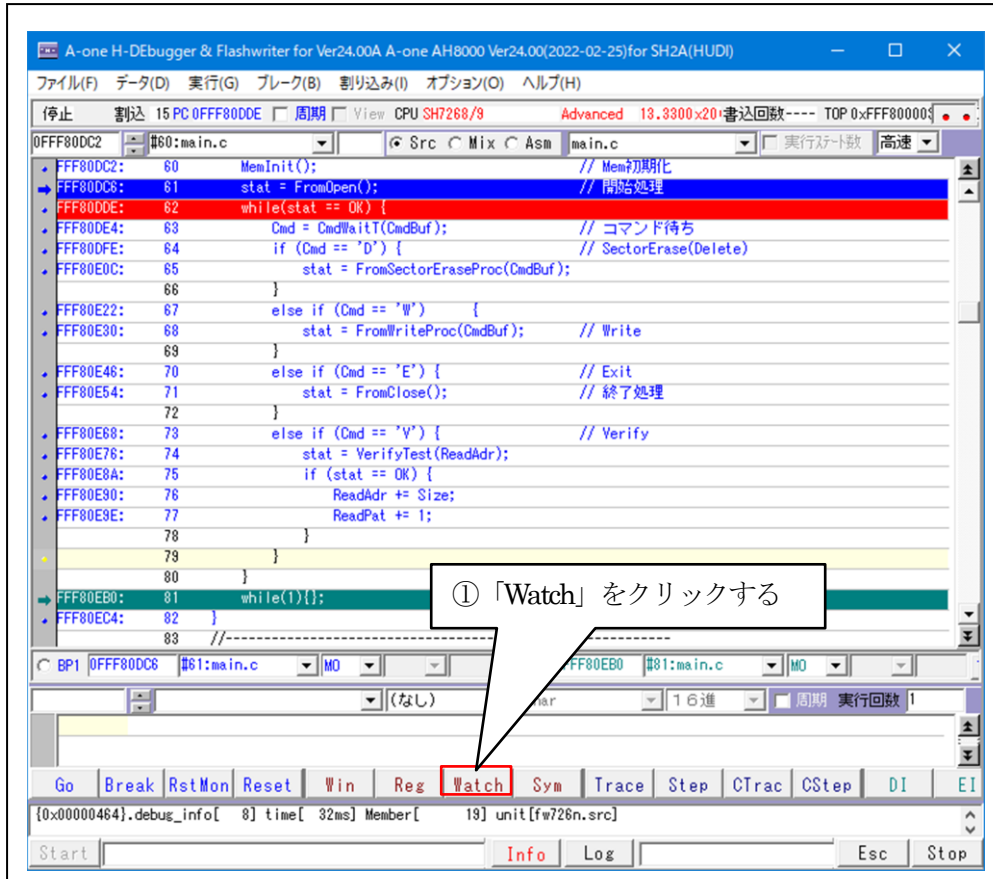
4) 「main.c:TestMain()」 にブレークポイントを設定後、実行する。



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。

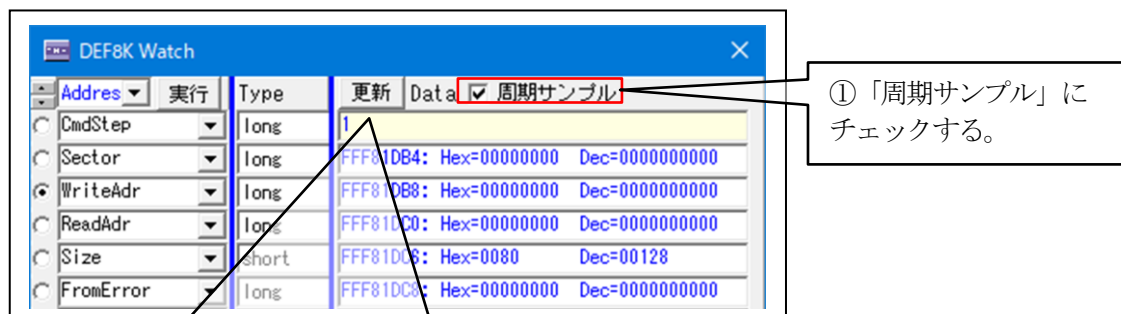
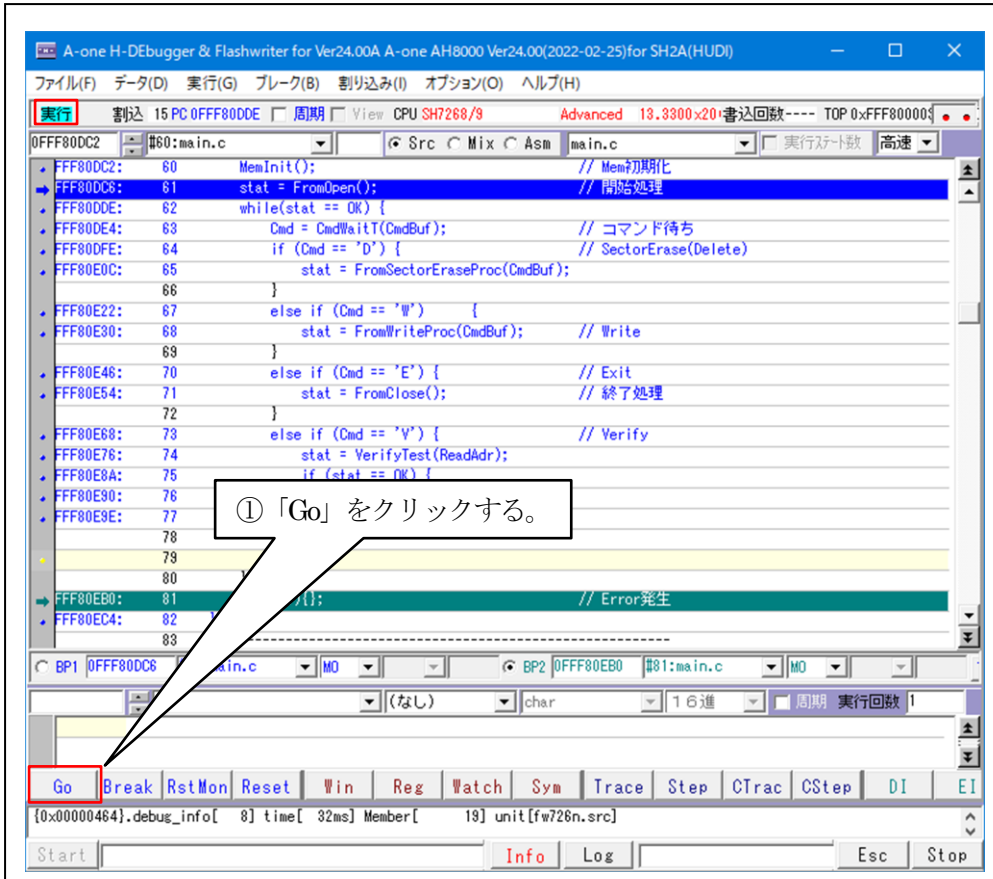


デバッグを進めるため、Watch 画面に変数を登録する。

- 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 パッケージサイズ(128/256/512)
- 6) FromError エラー発生時情報

6 変数を登録します。

7) セクターイレース関数「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「CmdWaitT0」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。

Address	実行	Type	更新	Data	周期サンプル
CmdStep	<input checked="" type="checkbox"/>	long	FFF82E84:	Hex=00000000	Dec=000000000
Sector	<input type="checkbox"/>	long	FFF82E88:	Hex=00000040	Dec=000000064
WriteAdr	<input type="checkbox"/>	long	FFF82E8C:	Hex=00000000	Dec=000000000
ReadAdr	<input type="checkbox"/>	long	FFF82E94:	Hex=00000000	Dec=000000000
Size	<input type="checkbox"/>	short	FFF82E9A:	Hex=0080	Dec=00128
FromError	<input type="checkbox"/>	long	FFF82E9C:	Hex=00000000	Dec=000000000

「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。

①「周期サンプル」にチェックする。

②「パケットサイズ」を変更する場合に書き換える。(128/256/512)

変数	Type	更新	Data
CmdStep	long	2	
Sector	long	FFF81DB4	Hex=00000100 Dec=0000000256
WriteAdr	long	FFF81DB8	Hex=00000000 Dec=0000000000
ReadAdr	long	FFF81DC0	Hex=00000000 Dec=0000000000
Size	short	FFF81DC6	Hex=0080 Dec=00128
FromError	long	FFF81DC8	Hex=00000000 Dec=0000000000

③変数「CmdStep」を数値「2」にする。// BP2で設定

<備考>

「main.c」の「CmdWaitT0」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。

変数	Type	更新	Data
CmdStep	long	FFF82E84	Hex=00000000 Dec=0000000000
Sector	long	FFF82E88	Hex=00000040 Dec=0000000064
WriteAdr	long	FFF82E8C	Hex=00400000 Dec=0004194304
ReadAdr	long	FFF82E94	Hex=00000000 Dec=0000000000
Size	short	FFF82E9A	Hex=0080 Dec=00128
FromError	long	FFF82E9C	Hex=00000000 Dec=0000000000

「WriteAdr」は、FlashROMの書き込み先頭アドレス「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。

9) パラレルFlashROM 全エリアをバリファイする。



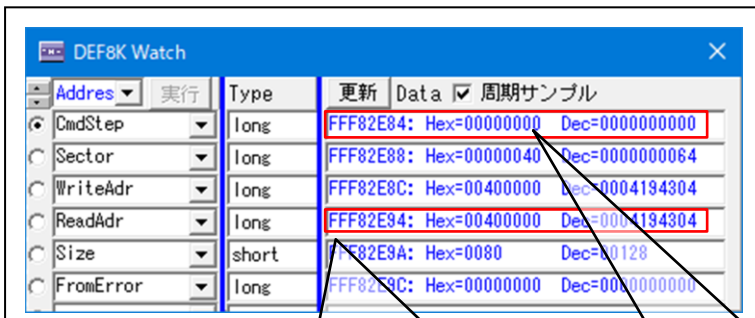
①「周期サンプル」にチェックする。

②変数「CmdStep」を数値「3」にする。// ⌘3⇩で設定

<備考>

「main.c」の「CmdWaitT0」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バリファイを実施する。

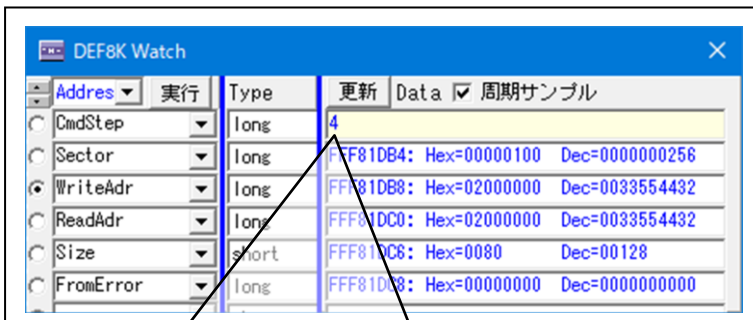
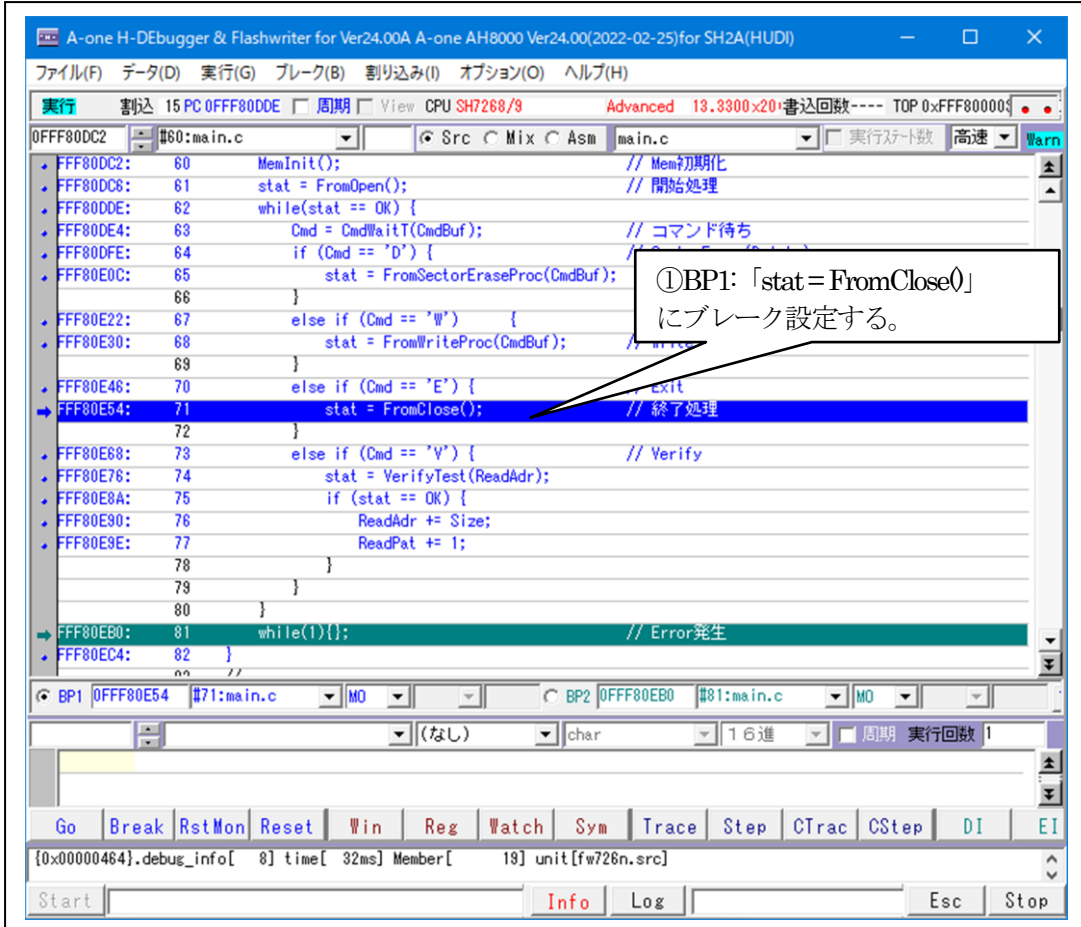


「ReadAdr」は、FlashROMの読み出し先頭アドレス「0x0」から登録最終アドレスまでインクリメントする。

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。

1 0) クローズ「FromClose()」を確認する。



②変数「CmdStep」を数値「4」にする。// ↵4↵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「FromClose()」をコールする

A-one H-DEbugger & Flashwriter for Ver24.00A A-one AH8000 Ver24.00(2022-02-25) for SH2A(HUDI)

ファイル(F) データ(D) 実行(G) ブレーク(B) 割り込み(I) オプション(O) ヘルプ(H)

停止 割込 15 PC:0FFF80E54 周期 View CPU SH7268/9 Advanced 13.3300x20 書込回数---- TOP 0x0FFF80000

0FFF80DC2 #80:main.c Src Mix Asm main.c 実行ステップ数 高速 Warn

```

FFF80DC2: 60 MemInit(); // Mem初期化
FFF80DC8: 61 stat = FromOpen(); // 開始処理
FFF80DDE: 62 while(stat == OK) {
FFF80DE4: 63 Cmd = CmdWaitT(CmdBuf); // コマンド待ち
FFF80DFE: 64 if (Cmd == 'D') { // SectorErase(Delete)
FFF80E0C: 65 stat = FromSectorEraseProc(CmdBuf);
66 }
FFF80E22: 67 else if (Cmd == 'W') {
FFF80E30: 68 stat = FromWriteProc(CmdBuf); // Write
69 }
FFF80E46: 70 else if (Cmd == 'E') { // Exit
FFF80E54: 71 stat = FromClose(); // 終了処理
72 }
FFF80E88: 73 else if (Cmd == 'V') {
FFF80E76: 74 stat = VerifyTest(ReadAdr);
FFF80E8A: 75 if (stat == OK) {
FFF80E90: 76 ReadAdr += Size;
FFF80E9E: 77 ReadPat += 1;
78 }
79 }
80 }
FFF80EB0: 81 while(1){};
FFF80EC4: 82 }

```

BP1 0FFF80E54 #71:main.c MO (なし) char 16進 初期 実行回数 1

Go Break RstMon Reset Win Reg Watch Sym Trace Step CTrac CStep DI EI

{0x00000464}.debug_info[8] time[32ms] Member[19] unit[fw726n.src]

Start Info Log Esc Stop

A-one H-DEbugger & Flashwriter for Ver24.00A A-one AH8000 Ver24.00(2022-02-25) for SH2A(HUDI)

ファイル(F) データ(D) 実行(G) ブレーク(B) 割り込み(I) オプション(O) ヘルプ(H)

停止 割込 15 PC:0FFF82822 周期 View CPU SH7268/9 Advanced 13.3300x20 書込回数---- TOP 0x0FFF80000

0FFF82822 FromClose Src Mix Asm FromTemp.c 実行ステップ数 高速 Warn

```

127 // OK == 正常終了
128 // NG == 異常終了
129 //*****
FFF82822: 130 INT FromClose(void)
131 {
FFF82824: 132 sFlashProtect();
133
FFF82828: 134 return(OK);
FFF8282C: 135 }
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 //*****

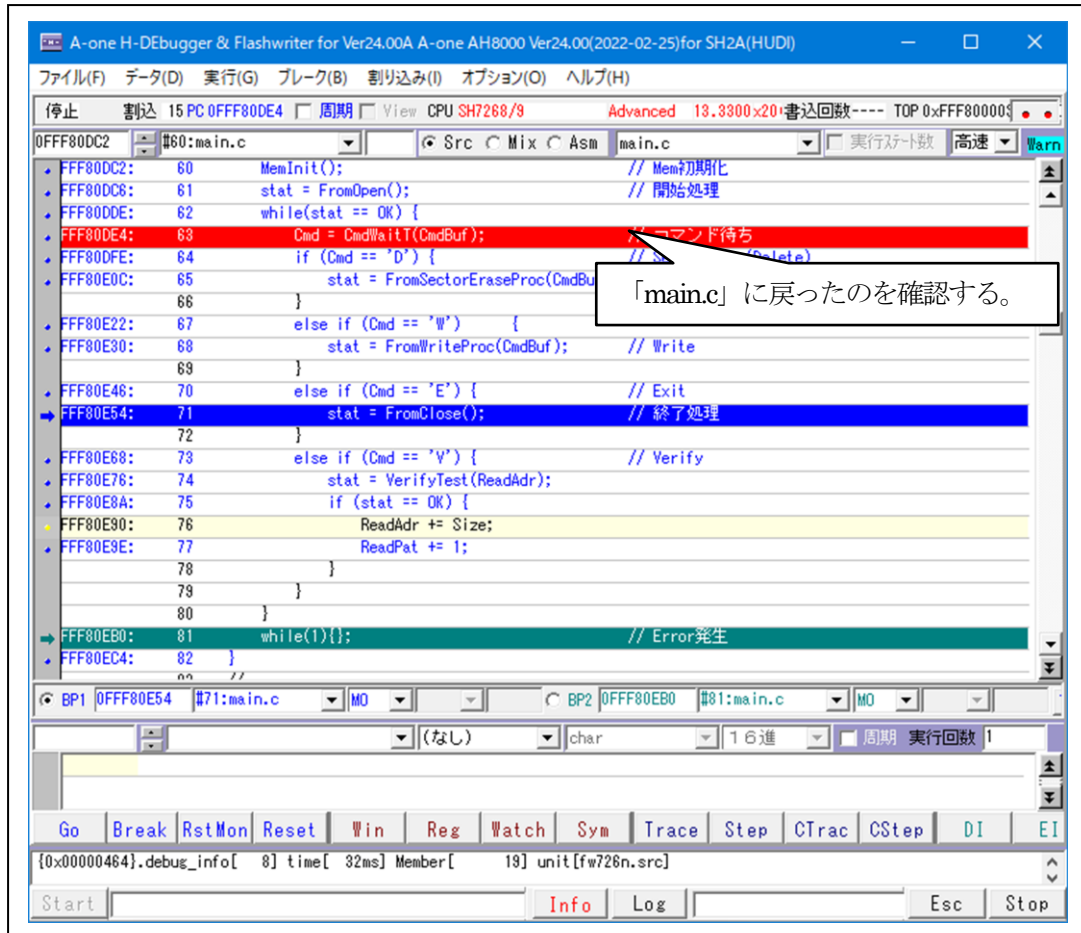
```

BP1 0FFF80E54 #70:main.c MO (なし) char 16進 周期 回数 1

Go Break RstMon Reset Win Reg Watch Sym Trace Step CTrac CStep DI EI

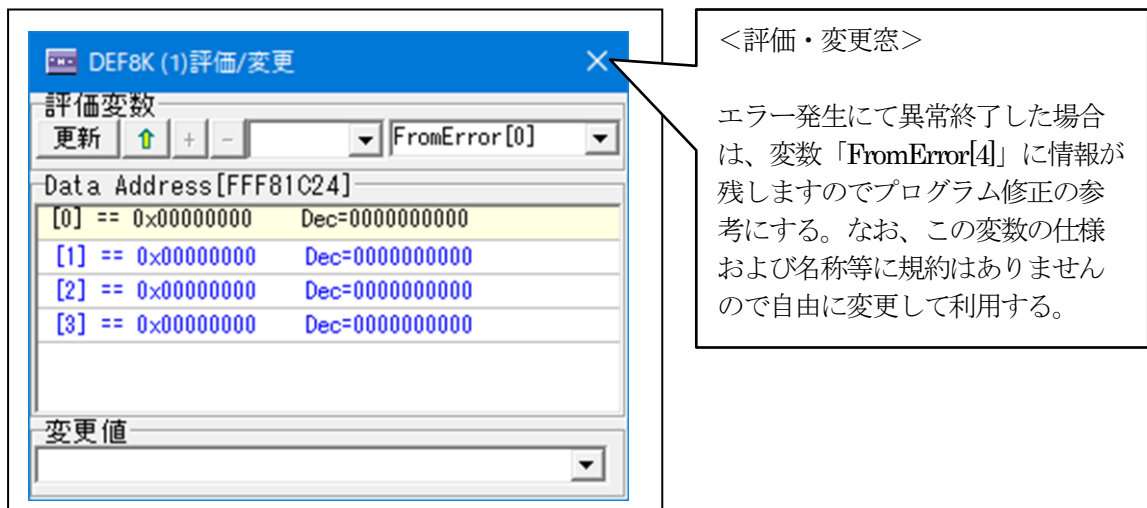
complete symbols=427 lines=804

Start Info Log Esc Stop



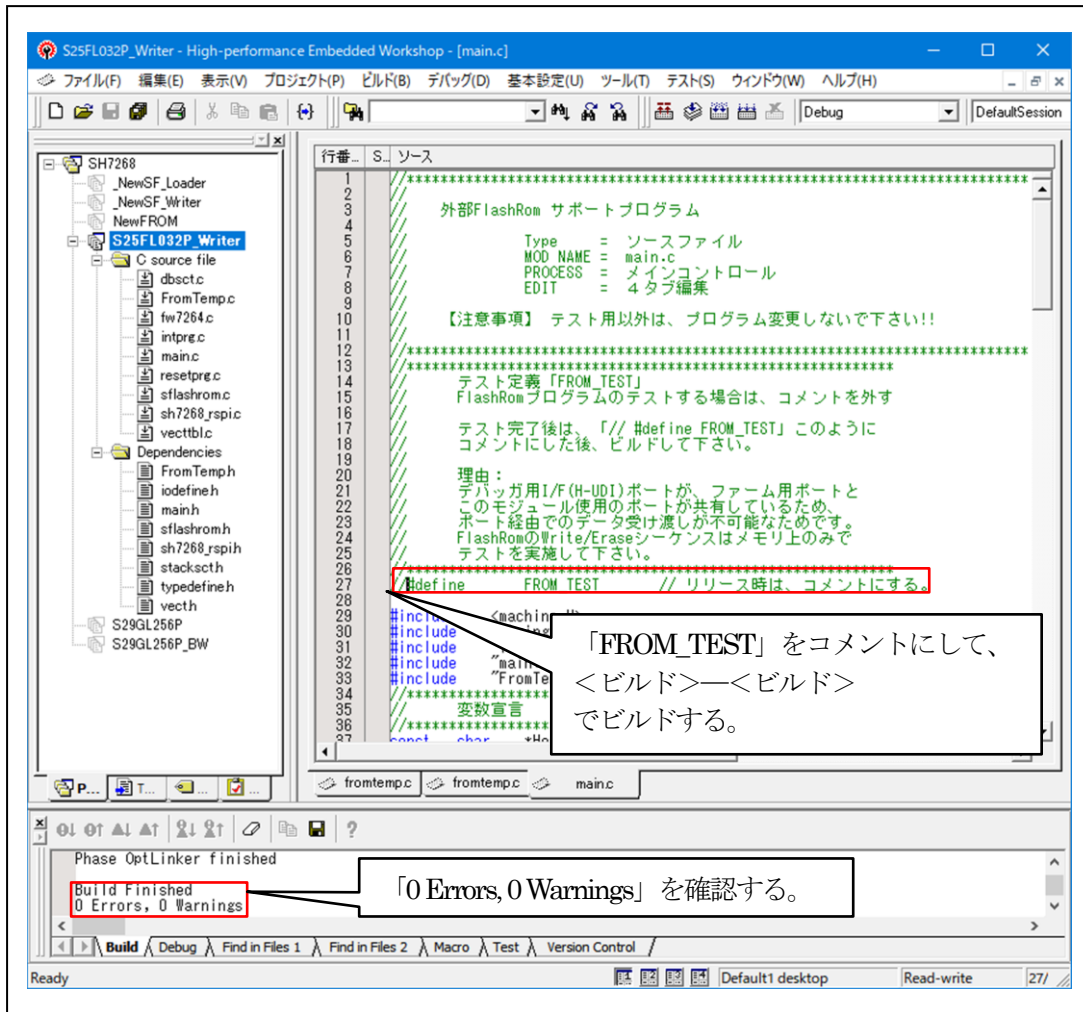
以上で FlashROM プログラムの動作確認は終了です。

1 1) 異常終了時のエラー情報「long FromError[4]」の参照



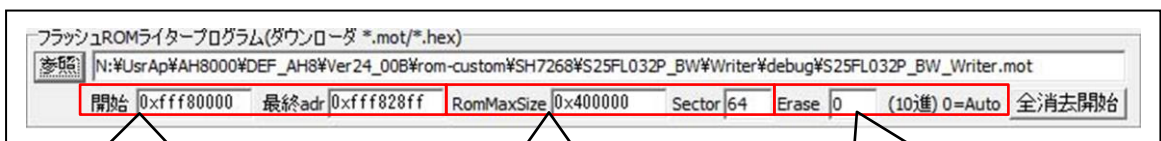
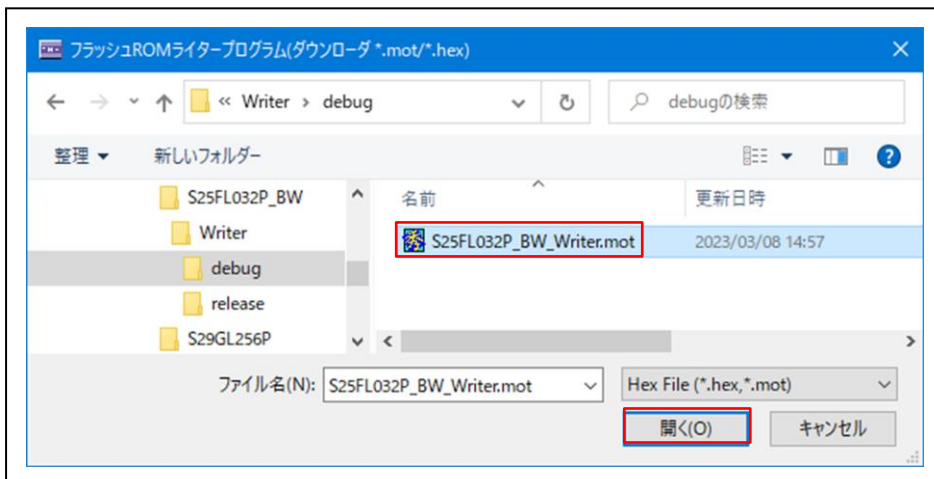
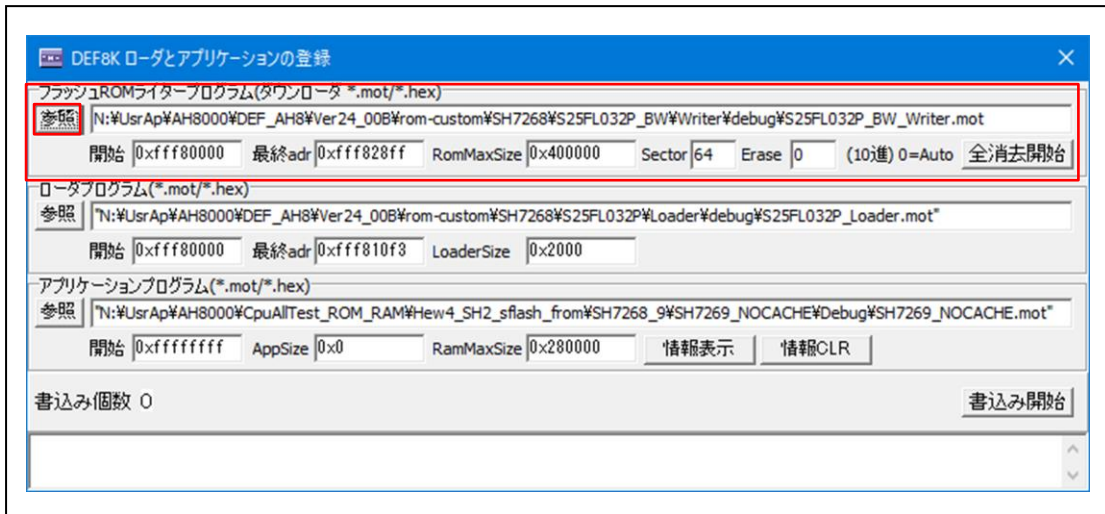
6. シリアル FlashROM プログラムを正規リリース登録する。

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。

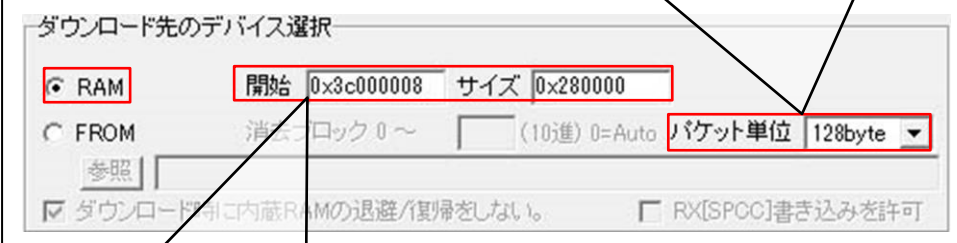
登録されているシリアル FlashROM の最大サイズとセクター数を表示する。

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。



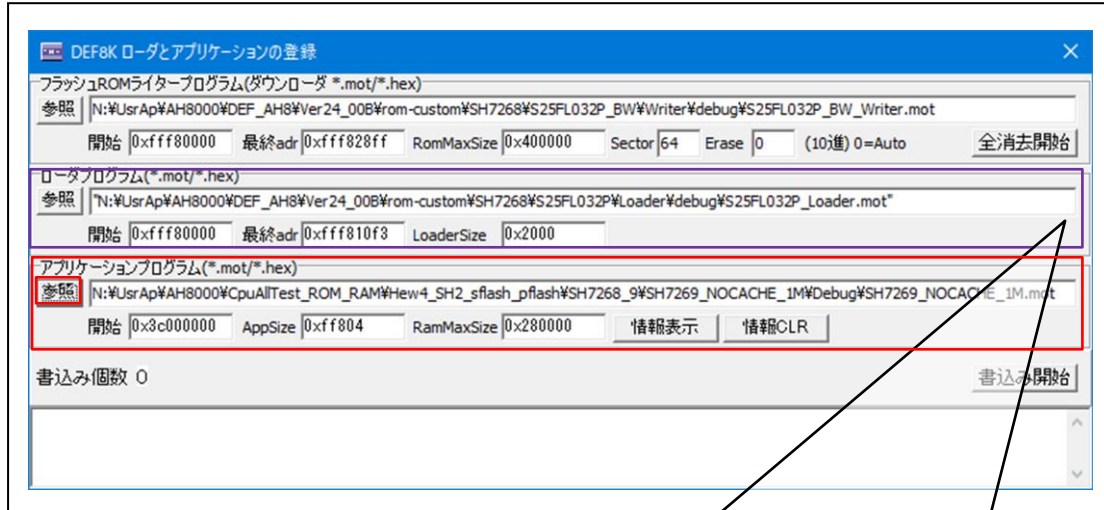
7. シリアル FlashROM プログラムの最終確認をする。

7-1) ターゲット基板側のモード設定

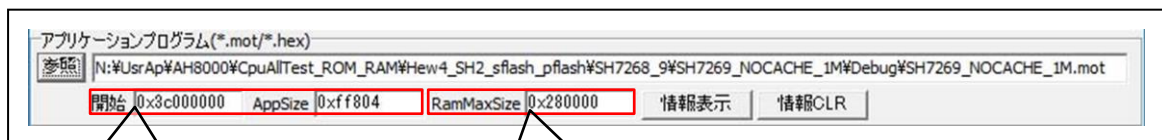
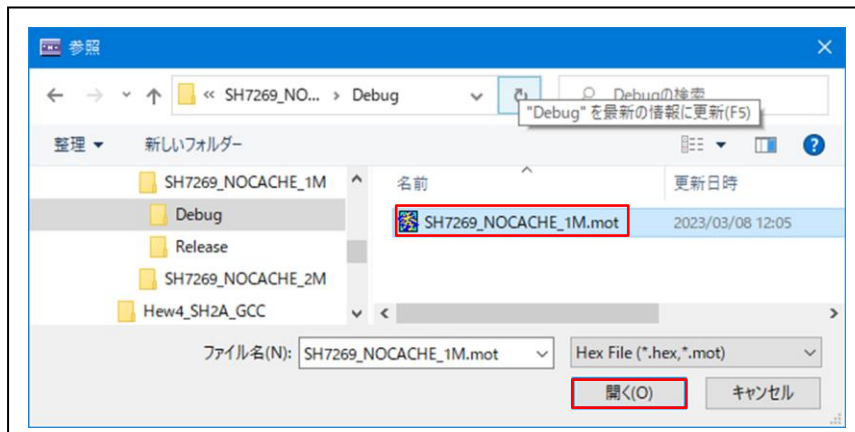
「ブートモード3」（シリアルフラッシュメモリからブートする）に設定する。

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。



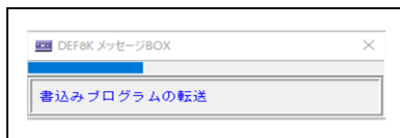
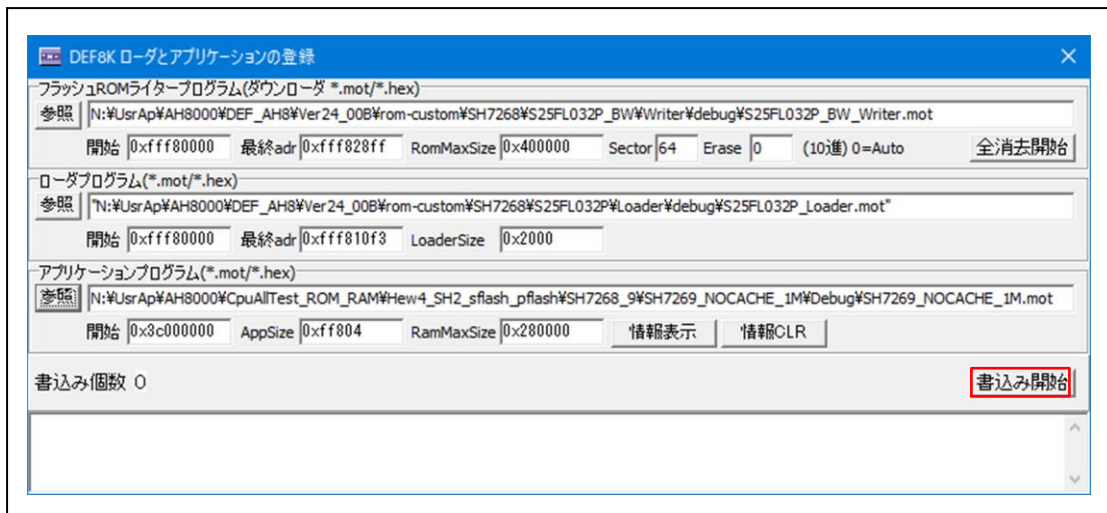
追加シリアル FlashROM 用のローダプログラムが未完成の場合はデフォルトで用意してある「C:\¥Aone¥DEF8K¥rom-custom¥SH7268¥S25FL032P¥Loader¥debug¥S25FL032P_Loader.mot」を一旦設定する。



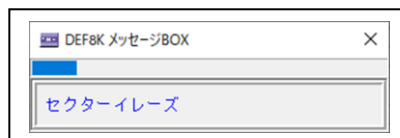
アプリケーションプログラムの開始アドレスとサイズを表示する。

「CPU 設定」に設定した内蔵 RAM サイズを表示する。

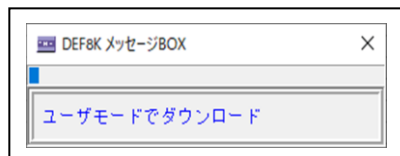
2) 「書き込み開始」をクリックする。



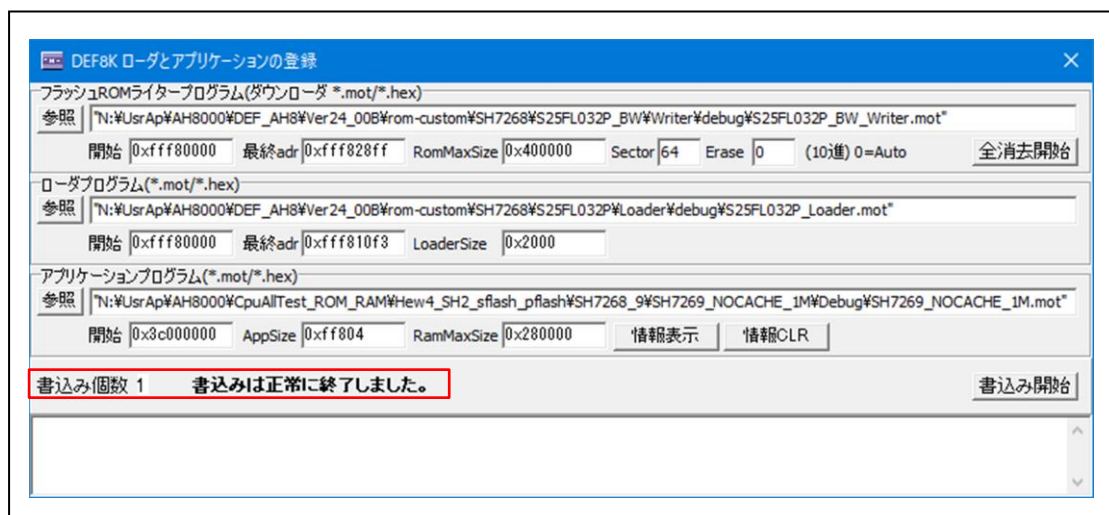
① FROM 書き込みプログラムの転送



② FROM のセクターイレーズ

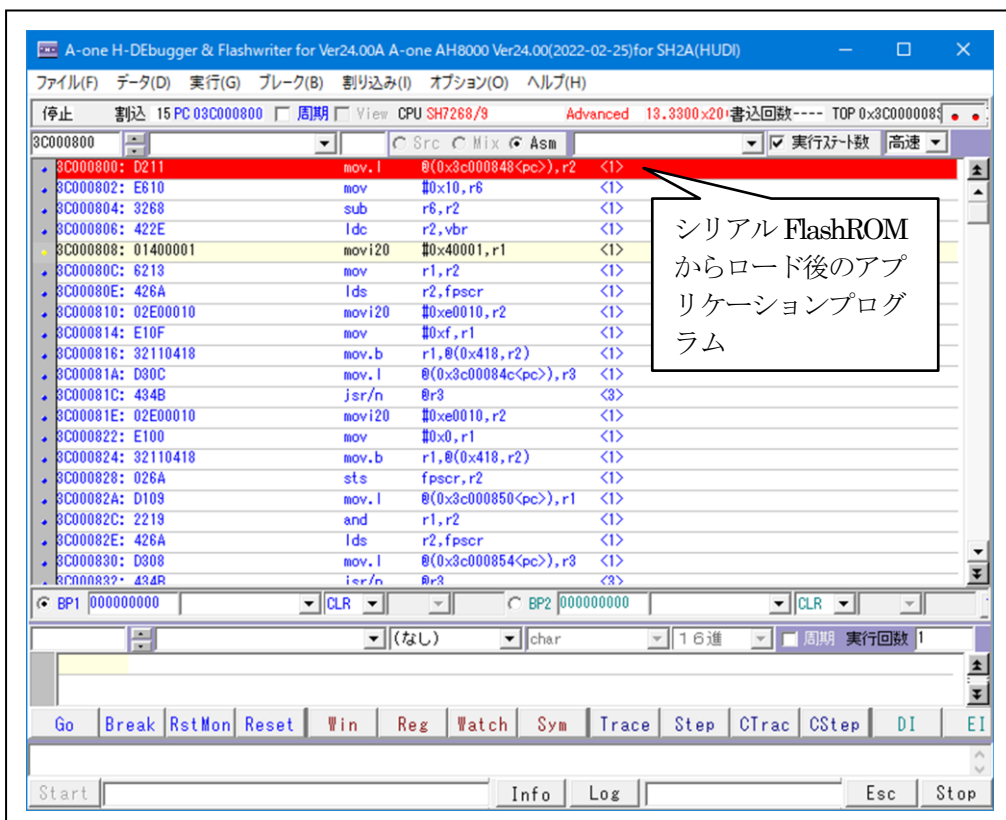
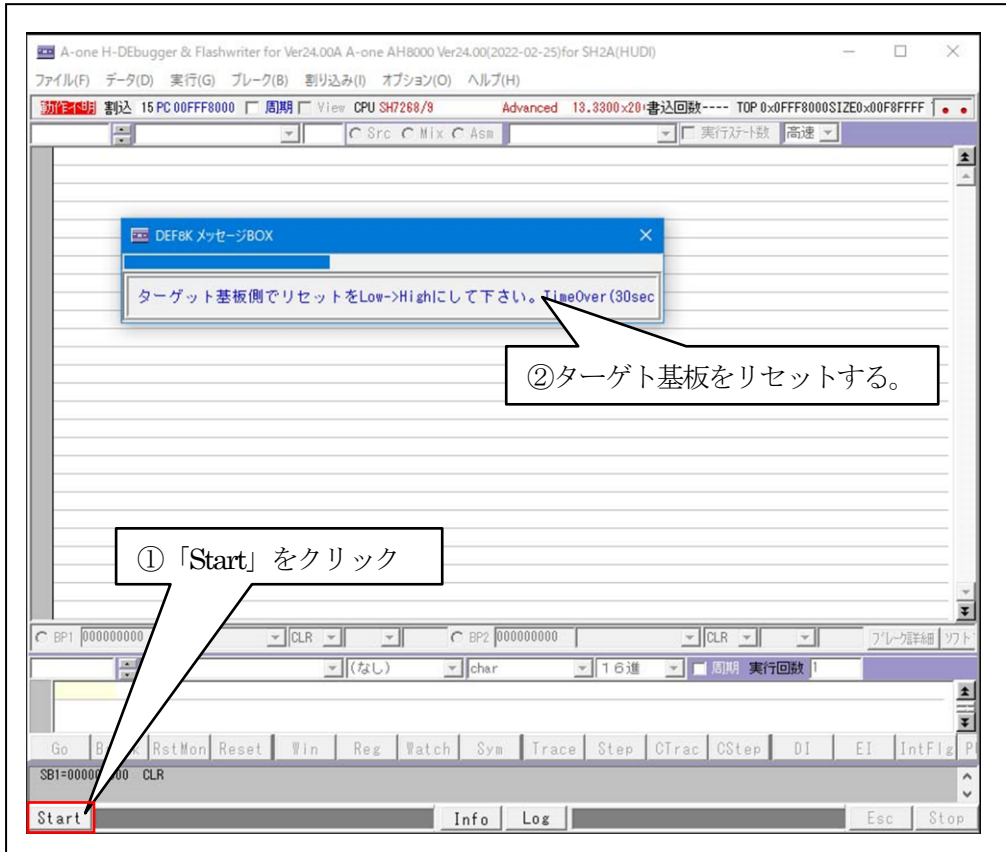


③ 拡張パケットでの FROM への書き込み

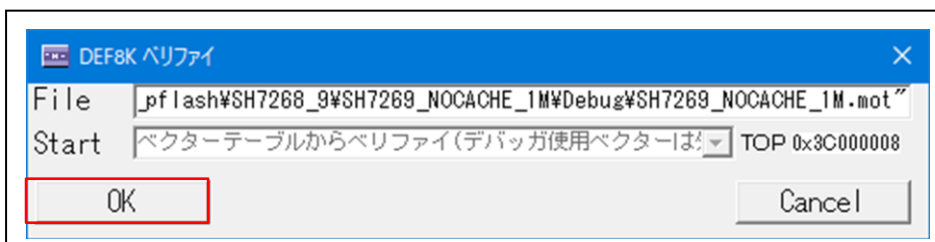
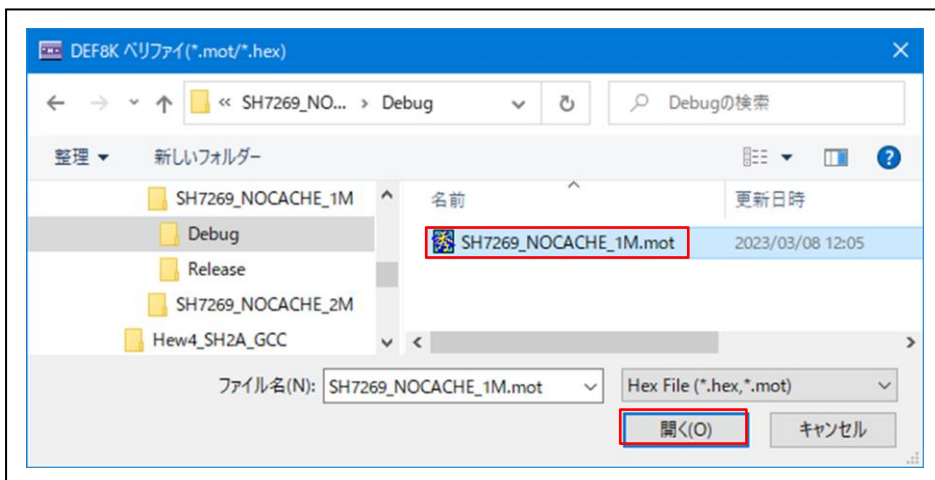
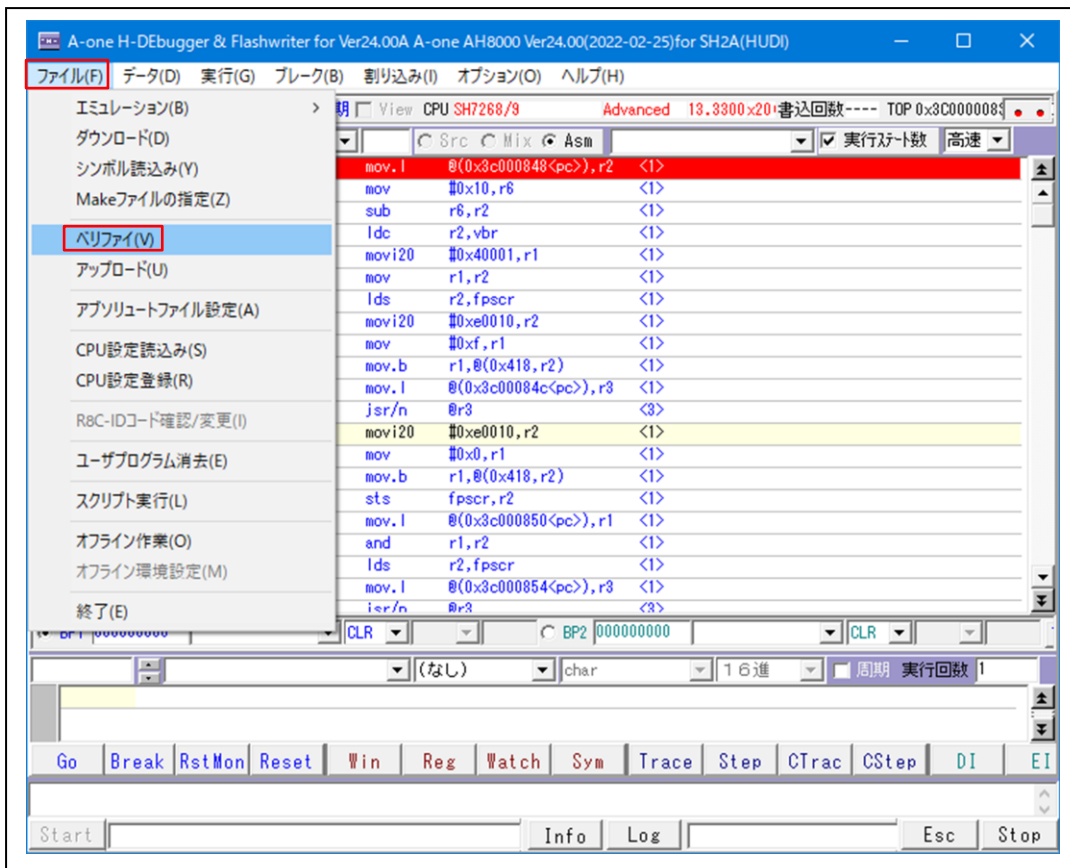


7-3) 「アプリケーションプログラム」が正常に書き込みされたかベリファイで確認する。
 【条件】この確認は、新規ローダープログラムが完成していることが条件になります。

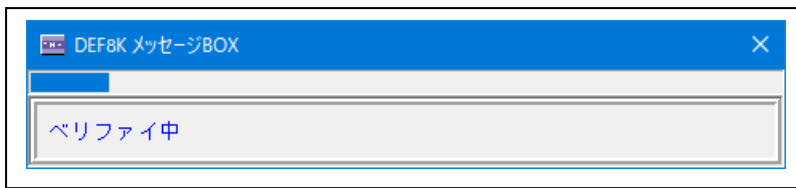
- 1) ターゲットの電源を再投入後、ロード時間を待ってから接続する。
 (参考) 1.0M バイトコードのロード時間は約 5 秒



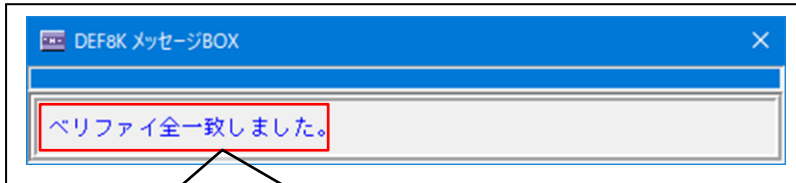
2) 内蔵RAM とベリファイする。



【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を負わないもの
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>

