

ROM レス品種にて、外付けパラレル 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

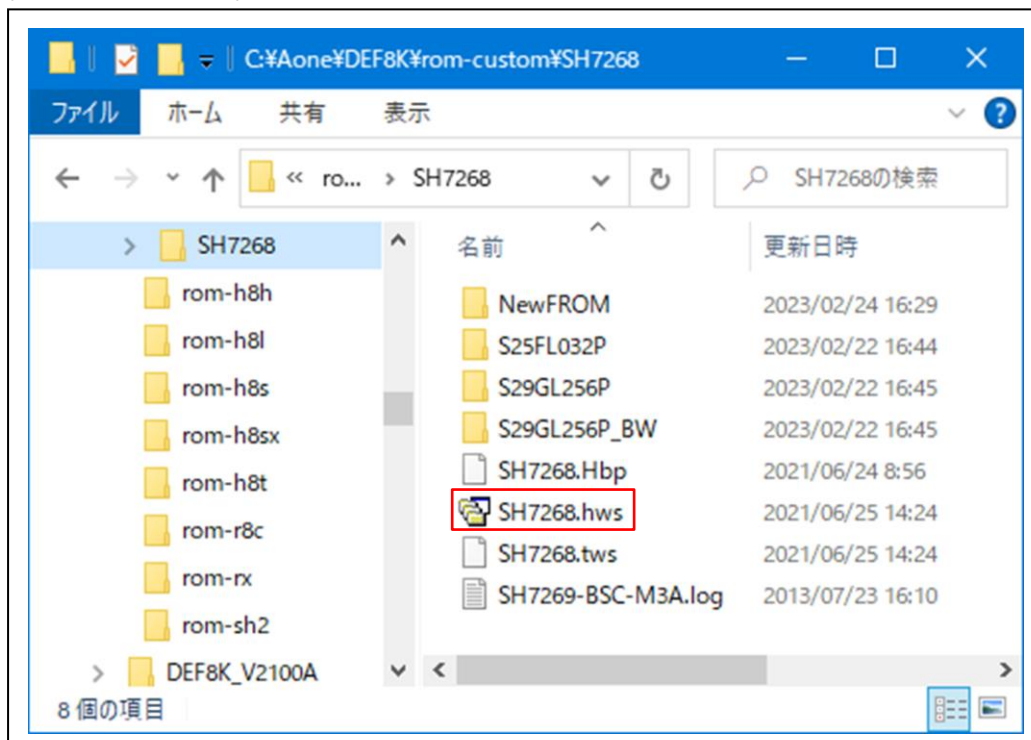
- ROM レス品種が対象です。(SH7262/7264/7266/7267/7268/7269)

2. 機能

- Hew4 にてパラレル 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

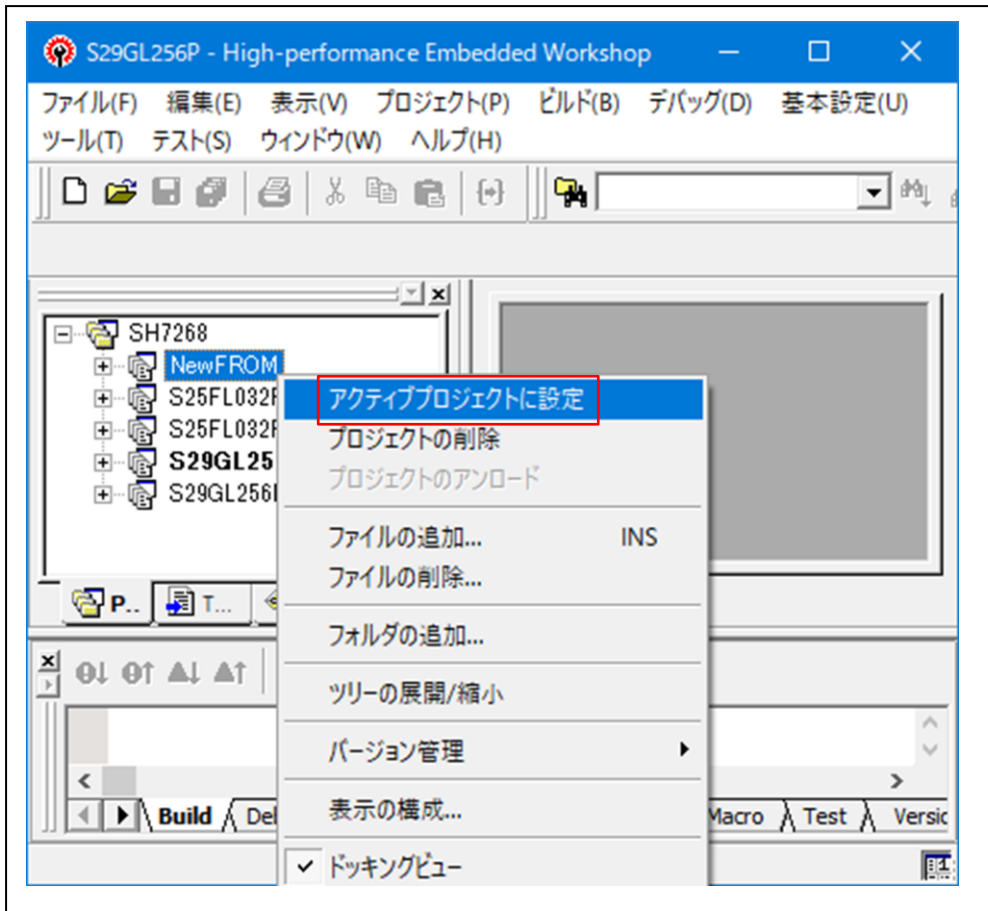
3-1) AH8000 コントロールソフト(DEF8K)のインストール DIR にあるワークスペースを開く。
(SH7268 での追加例)



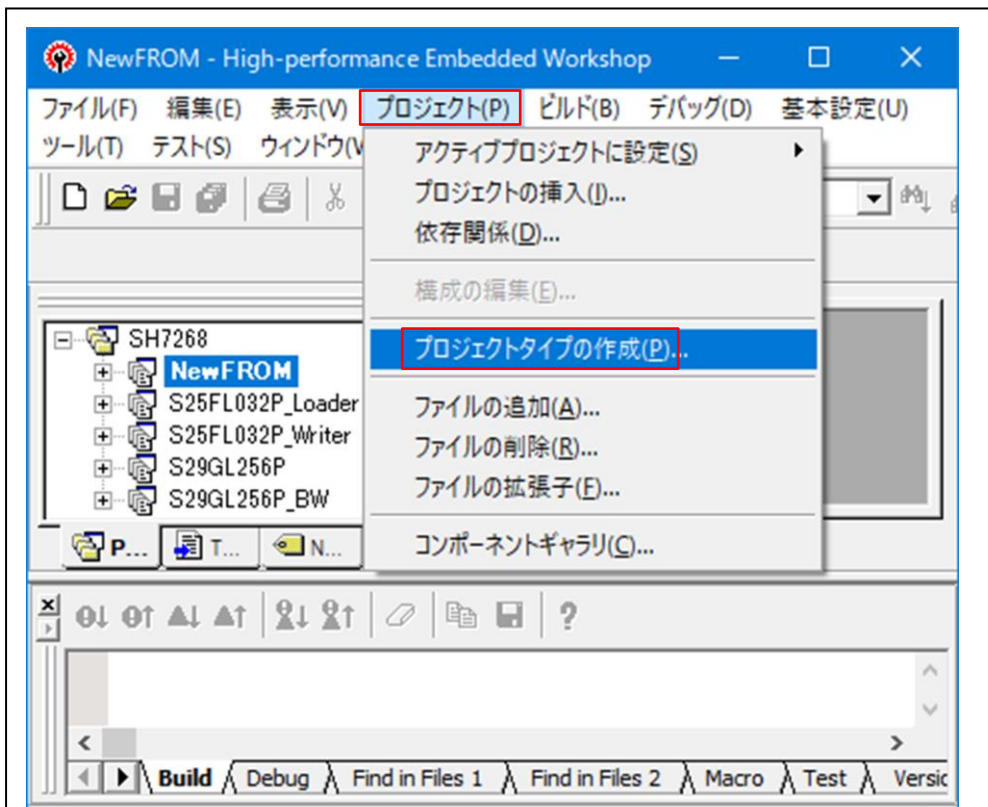
<default directory>

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

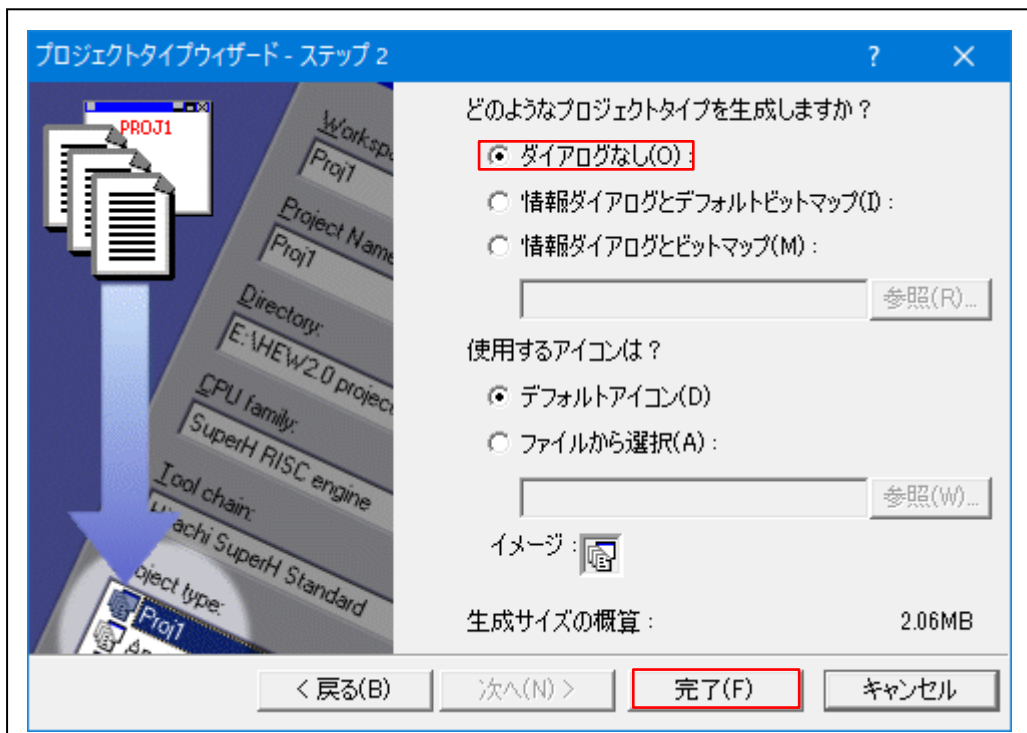
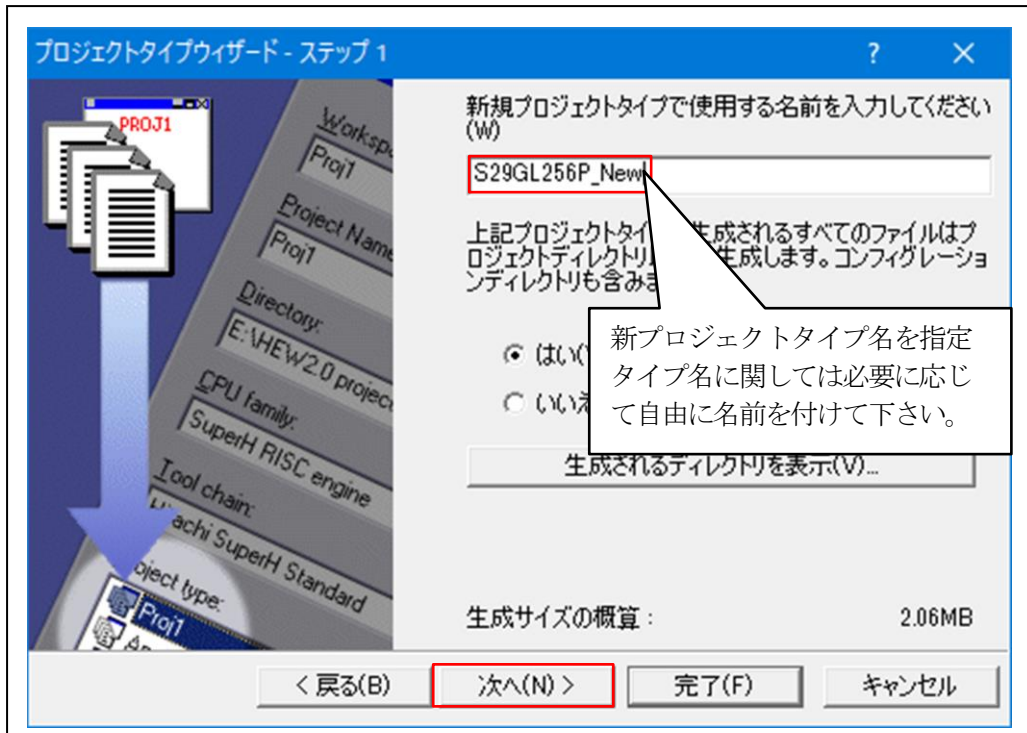
3-2) プロジェクト名「NewFROM」をアクティブプロジェクトにする。



3-3) 「プロジェクトタイプの作成」を指定する。

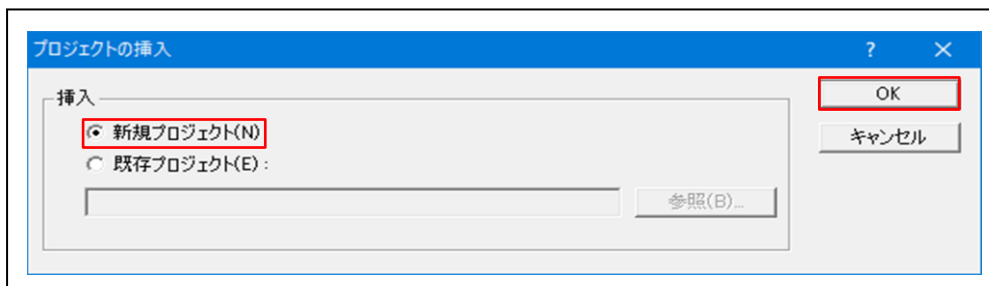
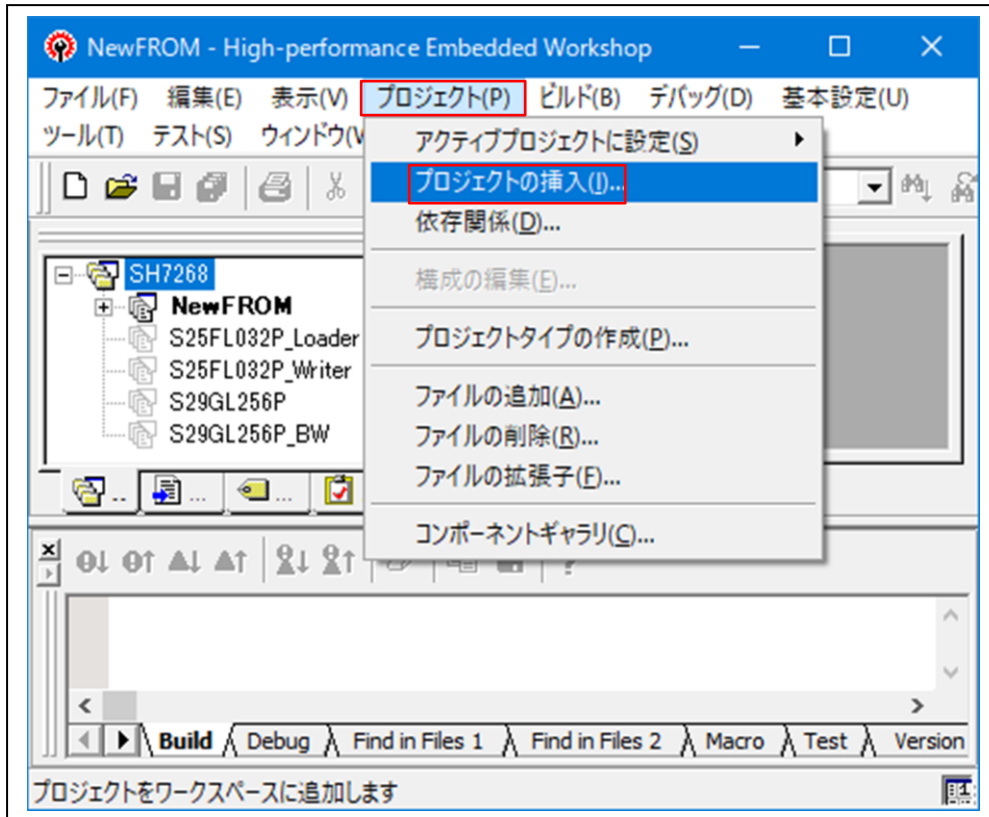


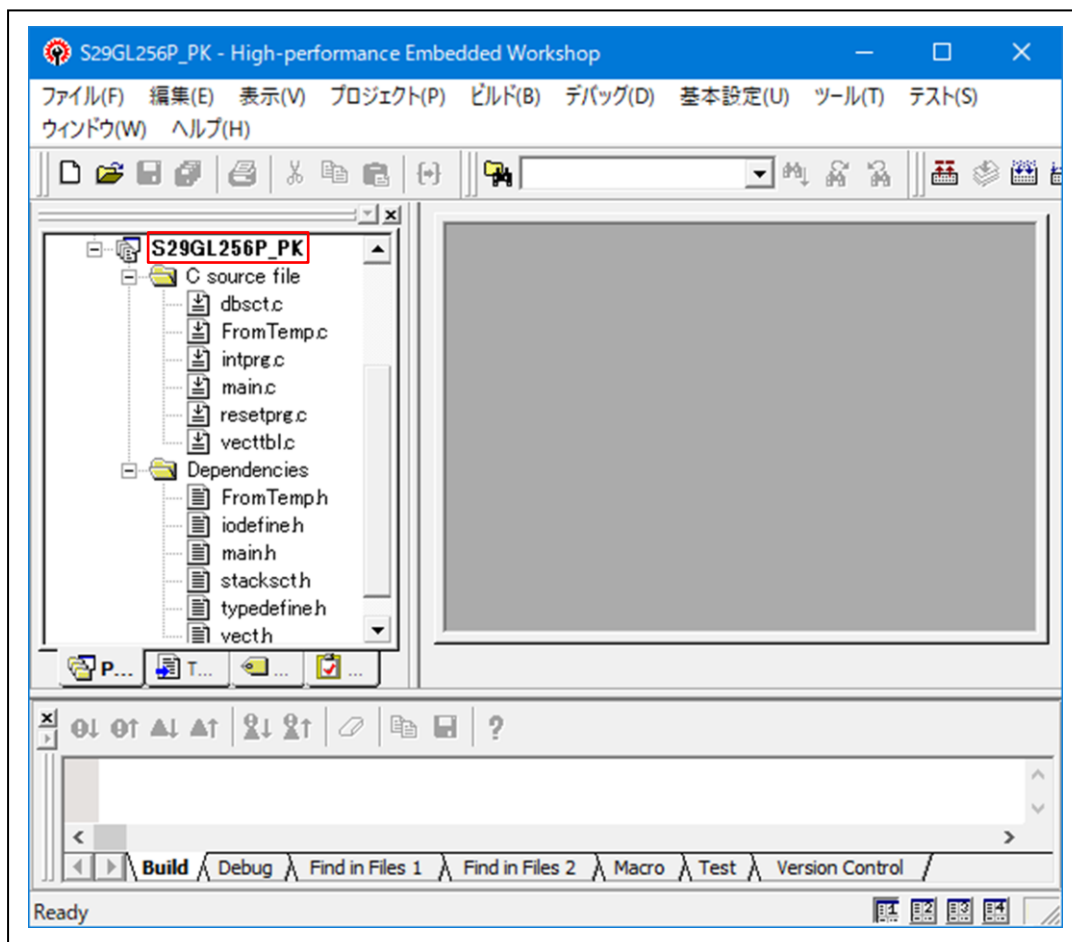
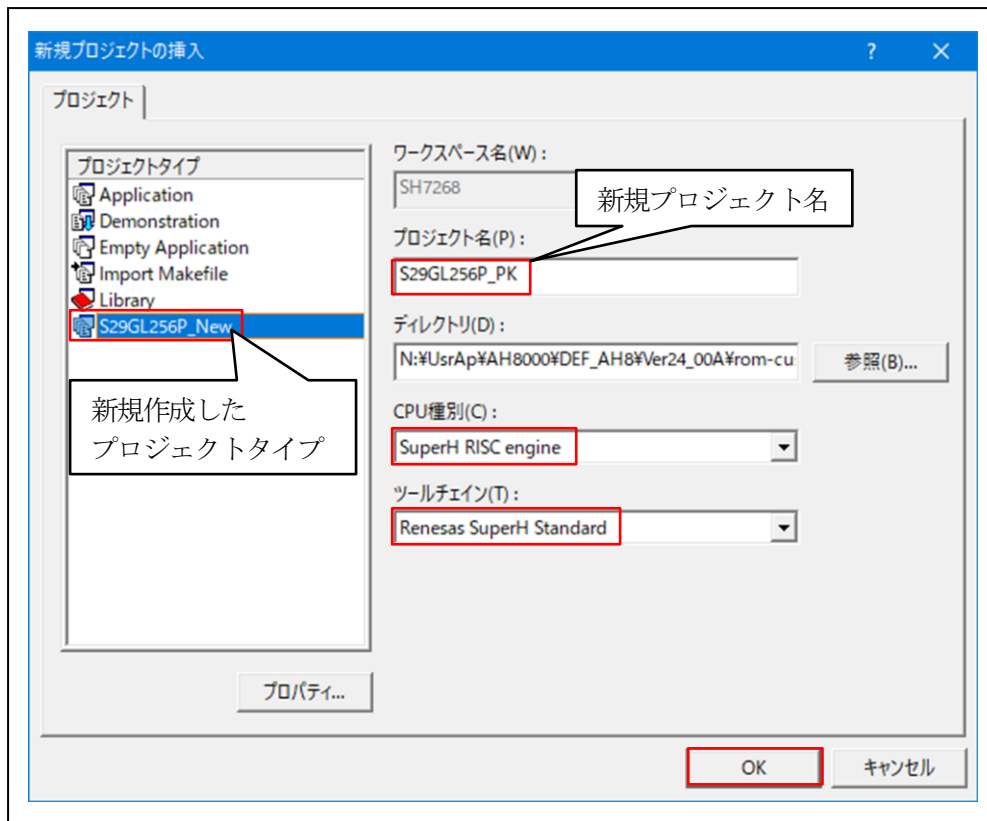
3-4) 追加FlashROMの新プロジェクトタイプを作成する。



4. パラレル FlashROM 品種の追加

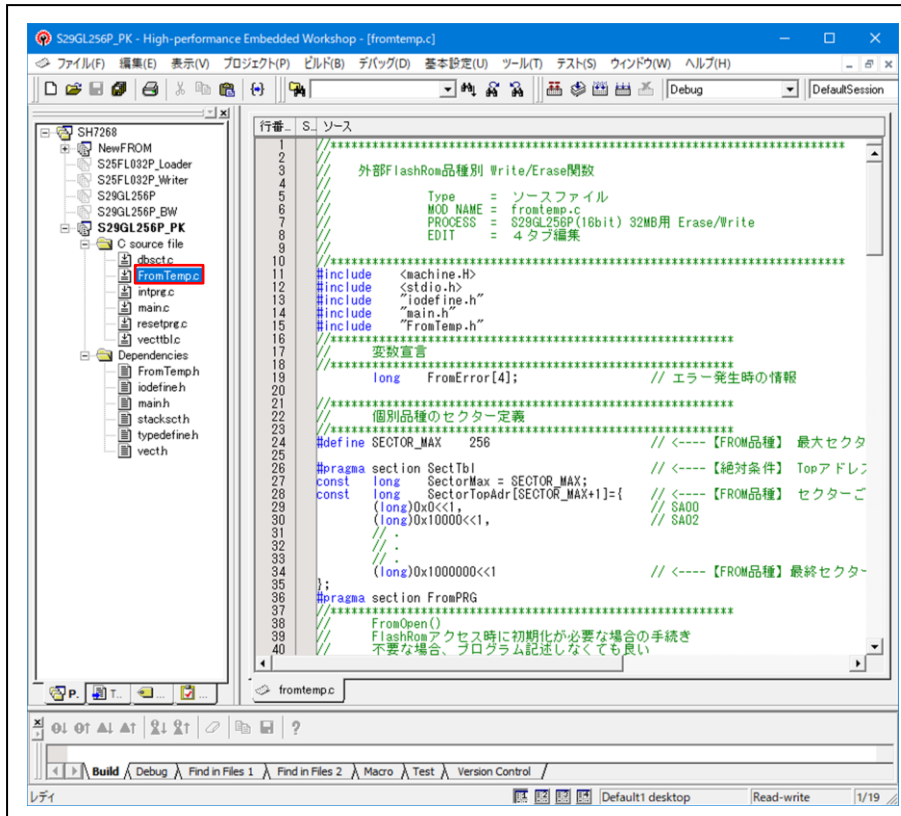
4-1) 新規プロジェクトの挿入



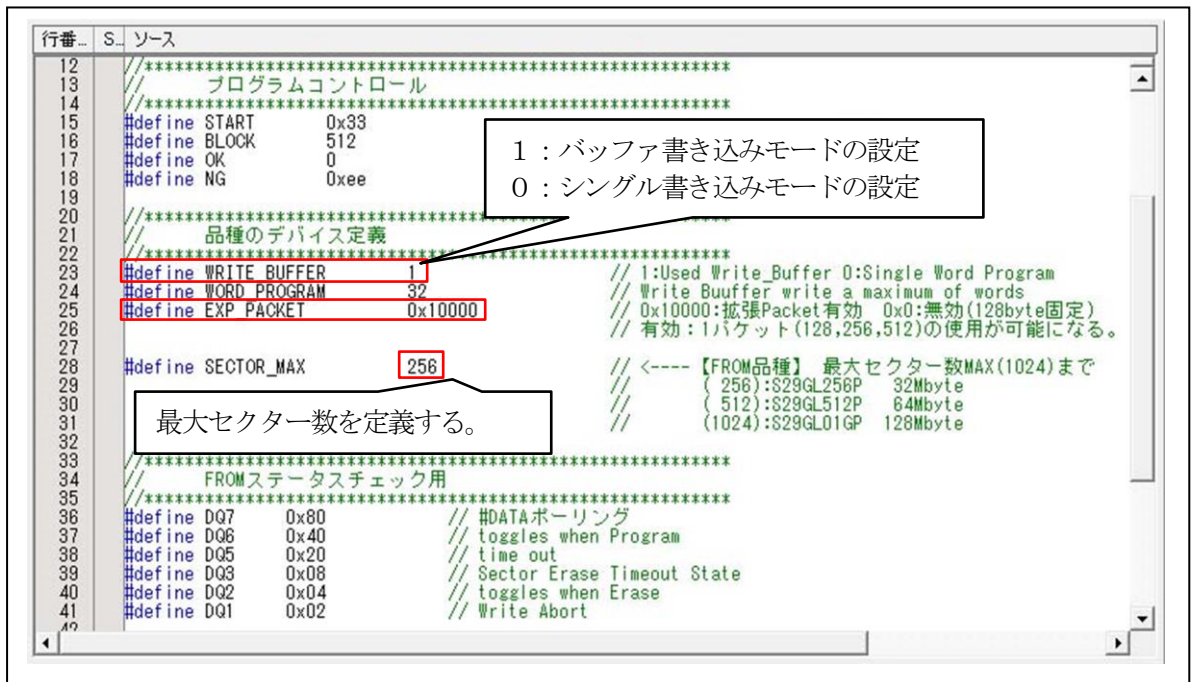


4-2) 新規プロジェクトに追加する FlashROM 品種の定義およびプログラムを作成する。

1) 「FromTemp.c」を開く



2) 「FromTemp.h」の#defineを定義する。



3) パラレルFlashROMのセクター情報を定義する。

```

16 ////////////////////////////////////////////////////
17 // 変数宣言
18 ////////////////////////////////////////////////////
19 long FromError[4]; // エラー発生回数
20 ////////////////////////////////////////////////////
21 // 個別品種のセクター定義
22 ////////////////////////////////////////////////////
23 ////////////////////////////////////////////////////
24 ////////////////////////////////////////////////////
25 #pragma section SectTbl
26 const long SectorMax = (SECTOR_MAX | EXP_PACKET);
27 const long SectorTopAddr = (SECTOR_MAX | EXP_PACKET);
28 ////////////////////////////////////////////////////
29 (long)0x0<<1, // SA00
30 (long)0x10000<<1, (long)0x20000<<1, (long)0x30000<<1, (long)0x40000<<1, (long)0x50000<<1,
31 (long)0x60000<<1, (long)0x70000<<1, (long)0x80000<<1, (long)0x90000<<1, (long)0xA0000<<1,
32 (long)0xB0000<<1, (long)0xC0000<<1, (long)0xD0000<<1, (long)0xE0000<<1, (long)0xF0000<<1,
33 (long)0x100000<<1, // SA16
34 (long)0x110000<<1, (long)0x120000<<1, (long)0x130000<<1, (long)0x140000<<1, (long)0x150000<<1,
35 (long)0x160000<<1, (long)0x170000<<1, (long)0x180000<<1, (long)0x190000<<1, (long)0x1A0000<<1,
36 (long)0x1B0000<<1, (long)0x1C0000<<1, (long)0x1D0000<<1, (long)0x1E0000<<1, (long)0x1F0000<<1,
37 (long)0x200000<<1, // SA32
38 (long)0x210000<<1, (long)0x220000<<1, (long)0x230000<<1, (long)0x240000<<1, (long)0x250000<<1,
39 (long)0x260000<<1, (long)0x270000<<1, (long)0x280000<<1, (long)0x290000<<1, (long)0x2A0000<<1,
40 (long)0x2B0000<<1, (long)0x2C0000<<1, (long)0x2D0000<<1, (long)0x2E0000<<1, (long)0x2F0000<<1,
41 (long)0x300000<<1, // SA48
42 (long)0x310000<<1, (long)0x320000<<1, (long)0x330000<<1, (long)0x340000<<1, (long)0x350000<<1,
43 (long)0x360000<<1, (long)0x370000<<1, (long)0x380000<<1, (long)0x390000<<1, (long)0x3A0000<<1,
44 (long)0x3B0000<<1, (long)0x3C0000<<1, (long)0x3D0000<<1, (long)0x3E0000<<1, (long)0x3F0000<<1,
45 (long)0x400000<<1, // SA64
46 (long)0x410000<<1, (long)0x420000<<1, (long)0x430000<<1, (long)0x440000<<1, (long)0x450000<<1,
47 (long)0x460000<<1, (long)0x470000<<1, (long)0x480000<<1, (long)0x490000<<1, (long)0x4A0000<<1,
48 (long)0x4B0000<<1, (long)0x4C0000<<1, (long)0x4D0000<<1, (long)0x4E0000<<1, (long)0x4F0000<<1,
49 (long)0x500000<<1, // SA80
50 (long)0x510000<<1, (long)0x520000<<1, (long)0x530000<<1, (long)0x540000<<1, (long)0x550000<<1,
51 (long)0x560000<<1, (long)0x570000<<1, (long)0x580000<<1, (long)0x590000<<1, (long)0x5A0000<<1,
52 (long)0x5B0000<<1, (long)0x5C0000<<1, (long)0x5D0000<<1, (long)0x5E0000<<1, (long)0x5F0000<<1,
53 (long)0x600000<<1, // SA96
54 (long)0x610000<<1, (long)0x620000<<1, (long)0x630000<<1, (long)0x640000<<1, (long)0x650000<<1,
55 (long)0x660000<<1, (long)0x670000<<1, (long)0x680000<<1, (long)0x690000<<1, (long)0x6A0000<<1,
56 (long)0x6B0000<<1, (long)0x6C0000<<1, (long)0x6D0000<<1, (long)0x6E0000<<1, (long)0x6F0000<<1,
57 (long)0x700000<<1, // SA112
58 (long)0x710000<<1, (long)0x720000<<1, (long)0x730000<<1, (long)0x740000<<1, (long)0x750000<<1,
59 (long)0x760000<<1, (long)0x770000<<1, (long)0x780000<<1, (long)0x790000<<1, (long)0x7A0000<<1,
60 (long)0x7B0000<<1, (long)0x7C0000<<1, (long)0x7D0000<<1, (long)0x7E0000<<1, (long)0x7F0000<<1,

```

⚠ 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。

```

65 (long)0x800000<<1, // SA128
66 (long)0x810000<<1, (long)0x820000<<1, (long)0x830000<<1, (long)0x840000<<1, (long)0x850000<<1,
67 (long)0x860000<<1, (long)0x870000<<1, (long)0x880000<<1, (long)0x890000<<1, (long)0x8A0000<<1,
68 (long)0x8B0000<<1, (long)0x8C0000<<1, (long)0x8D0000<<1, (long)0x8E0000<<1, (long)0x8F0000<<1,
69 (long)0x900000<<1, // SA144
70 (long)0x910000<<1, (long)0x920000<<1, (long)0x930000<<1, (long)0x940000<<1, (long)0x950000<<1,
71 (long)0x960000<<1, (long)0x970000<<1, (long)0x980000<<1, (long)0x990000<<1, (long)0x9A0000<<1,
72 (long)0x9B0000<<1, (long)0x9C0000<<1, (long)0x9D0000<<1, (long)0x9E0000<<1, (long)0x9F0000<<1,
73 (long)0xA00000<<1, // SA160
74 (long)0xA10000<<1, (long)0xA20000<<1, (long)0xA30000<<1, (long)0xA40000<<1, (long)0xA50000<<1,
75 (long)0xA60000<<1, (long)0xA70000<<1, (long)0xA80000<<1, (long)0xA90000<<1, (long)0xAA0000<<1,
76 (long)0xAB0000<<1, (long)0xAC0000<<1, (long)0xAD0000<<1, (long)0xAE0000<<1, (long)0xAF0000<<1,
77 (long)0xB00000<<1, // SA176
78 (long)0xB10000<<1, (long)0xB20000<<1, (long)0xB30000<<1, (long)0xB40000<<1, (long)0xB50000<<1,
79 (long)0xB60000<<1, (long)0xB70000<<1, (long)0xB80000<<1, (long)0xB90000<<1, (long)0xBA0000<<1,
80 (long)0xBB0000<<1, (long)0xBC0000<<1, (long)0xBD0000<<1, (long)0xBE0000<<1, (long)0xBF0000<<1,
81 (long)0xC00000<<1, // SA192
82 (long)0xC10000<<1, (long)0xC20000<<1, (long)0xC30000<<1, (long)0xC40000<<1, (long)0xC50000<<1,
83 (long)0xC60000<<1, (long)0xC70000<<1, (long)0xC80000<<1, (long)0xC90000<<1, (long)0xCA0000<<1,
84 (long)0xCB0000<<1, (long)0xCC0000<<1, (long)0xCD0000<<1, (long)0xCE0000<<1, (long)0xCF0000<<1,
85 (long)0xD00000<<1, // SA208
86 (long)0xD10000<<1, (long)0xD20000<<1, (long)0xD30000<<1, (long)0xD40000<<1, (long)0xD50000<<1,
87 (long)0xD60000<<1, (long)0xD70000<<1, (long)0xD80000<<1, (long)0xD90000<<1, (long)0xDA0000<<1,
88 (long)0xDB0000<<1, (long)0xDC0000<<1, (long)0xDD0000<<1, (long)0xDE0000<<1, (long)0xDF0000<<1,
89 (long)0xE00000<<1, // SA224
90 (long)0xE10000<<1, (long)0xE20000<<1, (long)0xE30000<<1, (long)0xE40000<<1, (long)0xE50000<<1,
91 (long)0xE60000<<1, (long)0xE70000<<1, (long)0xE80000<<1, (long)0xE90000<<1, (long)0xEA0000<<1,
92 (long)0xEB0000<<1, (long)0xEC0000<<1, (long)0xED0000<<1, (long)0xEE0000<<1, (long)0xEF0000<<1,
93 (long)0xF00000<<1, // SA240
94 (long)0xF10000<<1, (long)0xF20000<<1, (long)0xF30000<<1, (long)0xF40000<<1, (long)0xF50000<<1,
95 (long)0xF60000<<1, (long)0xF70000<<1, (long)0xF80000<<1, (long)0xF90000<<1, (long)0xFA0000<<1,
96 (long)0xFB0000<<1, (long)0xFC0000<<1, (long)0xFD0000<<1, (long)0xFE0000<<1, (long)0xFF0000<<1,
97 (long)0x1000000<<1 // <---- 【FROM品種】最終セクターアドレス+1を登録
98
99
100 #pragma section FromPRG
101 ////////////////////////////////////////////////////
102 FromOpen()
103 FlashRomアクセス時に初期化が必要な場合
104 不要な場合、プログラム記述しなくても良い
105 開始時に1回だけ呼び出す

```

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する。

4) 「オープン」関数を作成する。

```

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

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」の設定と「キャッシュ無効」を定義している。

必要無い場合は、「return(Ok);」のみ記述する。

5) 「クローズ」関数を作成する。

```

行番... S... ソース
117 //*****
118 FromClose()
119 FlashRom書き込み処理が全て終了した場合に必要な手続き
120 不要な場合、プログラム記述しなくても良い
121 終了時に1回だけ呼ばれる
122 return()
123 OK == 正常終了
124 NG == 異常終了
125 //*****
126 int FromClose(void)
127 {
128     return(OK);
129 }
130 //*****
131 //*****
  
```

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。

必要無い場合は、「return(Ok);」のみ記述する。

6) 「セクターイレース」関数を作成する。

```

行番... S... ソース
124 //*****
125 FromSectorEraseProc(char *cmd)
126 伝送フォーム(in)
127  cmd[0] == %
128  cmd[1] == 0
129  cmd[2] == CommandSize[2]以降[Sum]も含む
130  cmd[3] == Command('D')
131  cmd[4] == Sector(H) MAX(1024)
132  cmd[5] == Sector(L)
133  [Sum] == [3]->[Sum-1]の加算
134  return()
135  OK == 正常終了
136  NG == 異常終了
137 //*****
138 int FromSectorEraseProc(UCHAR *cmd)
139 {
140     USHORT *cmd555;
141     USHORT *cmdaaa;
142     USHORT *from; // Word Access
143     short sector;
144     int stat;
145
146     sector = (cmd[4] << 8) & 0xff00;
147     sector |= cmd[5] & 0xff;
148     cmd555 = (USHORT *)0x555;
149     cmdaaa = (USHORT *)0xaaa;
150     if (sector < SectorMax) {
151         *cmdaaa = 0xaa;
152         *cmd555 = 0x55;
153         *cmdaaa = 0x80;
154         *cmdaaa = 0xaa;
155         *cmd555 = 0x55;
156         from = (USHORT *)SectorTopAdr[sector];
157         *from = 0x30;
158         stat = FullStatusCheck(from,0x80,'D');
159         //
160         if (stat == OK) return(OK);
161     }
162     return(NG);
163 }
164 //*****
  
```

関数「FromSectorEraseProc(UCHAR *cmd)」には、セクターごとのイレース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>

全イレース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。

7) 「ワードプログラム (書き込み)」 関数を作成する。

(1) コマンドパラメータの算出と準備

```

164 *****
165 FromWriteProc(char *cmd)
166 伝送フォーム(in)
167  cmd[0] == %
168  [1] == len(H)
169  [2] == len(L) [3]から[Sum]までのLength
170  [3] == Command('W')
171  [4] == WriteAddress(HH)
172  [5] == WriteAddress(MH)
173  [6] == WriteAddress(ML)
174  [7] == WriteAddress(LL)
175  [8+0] == WriteData[0]
176  .
177  ==
178  [8+x] == WriteData[x] 固定 余分エリア(0xff)
179  x:512byte(max)
180  [Sum] == [3]->[Sum-1]の加算
181  return()
182  OK == 正常終了
183  NG == 異常終了
184 *****
185 int FromWriteProc(UCHAR *cmd)
186 {
187  USHORT *cmd555;
188  USHORT *cmdaaa;
189  USHORT *from;
190  USHORT *verf; // Word Access
191  USHORT *data; // Word Access
192  USHORT *sector;
193  USHORT len;
194  ULONG adr;
195  int s,i,n;
196  int wc;
197  int stat;
198
199  cmd555 = (USHORT *)0x555;
200  cmdaaa = (USHORT *)0xaaa;
201  len = (((USHORT)cmd[1] << 8) & 0xff00) |
202  ((USHORT)cmd[2] & 0x00ff);
203  len -= 8;
204
205  adr = (((long)cmd[4] << 24) & 0xff000000) |
206  (((long)cmd[5] << 16) & 0x00ff0000) |
207  (((long)cmd[6] << 8) & 0x0000ff00) |
208  ((long)cmd[7] & 0x000000ff);
209  from = (USHORT *)adr;
210  sector = (USHORT *) (adr & 0xFFFF0000);
211  data = (USHORT *)&cmd[8]; // Data Adr set

```

書き込み長の算出

書き込みアドレスの算出

書き込みデータのトップアドレスをセット

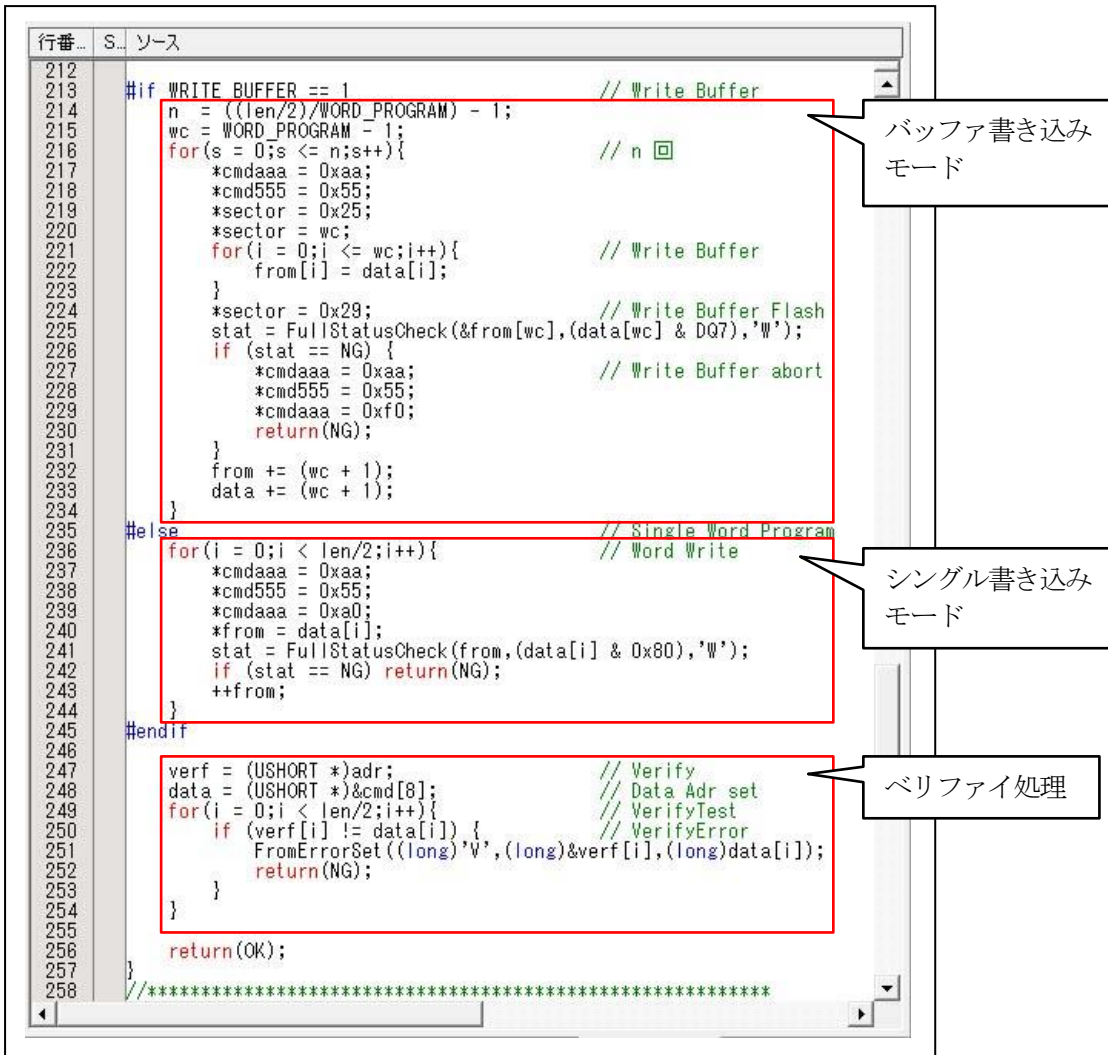
関数「FromWriteProc(UCHAR *cmd)」には、128~512 バイト(64~256 ワード)ごとの書き込みプログラムを記述する。受信完了後、サムチェック正常の場合、この関数をコール(Call)する。(サム異常時はコールなし)

<引数の仕様>

- char cmd[];
- cmd[0] = '%' // ヘッダー
- cmd[1] = len(H) // コマンドサイズ cmd[3]->[8+n]までのサイズ(H) 6+n(128/256/512)
- cmd[2] = len(L) // コマンドサイズ cmd[3]->[8+n]までのサイズ(L)
- 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[8+n] = 0xn // 書き込みデータ[n] 固定 余分エリアは(0xff) n=128/256/512
- cmd[9+n] = 0xn // サム cmd[3]->[8+n]の加算値

(2) 書き込みプログラムを作成



```

212
213 #if WRITE_BUFFER == 1 // Write Buffer
214     n = ((len/2)/WORD_PROGRAM) - 1;
215     wc = WORD_PROGRAM - 1;
216     for(s = 0; s <= n; s++){ // n回
217         *cmdaaa = 0xaa;
218         *cmd555 = 0x55;
219         *sector = 0x25;
220         *sector = wc;
221         for(i = 0; i <= wc; i++){ // Write Buffer
222             from[i] = data[i];
223         }
224         *sector = 0x29; // Write Buffer Flash
225         stat = FullStatusCheck(&from[wc], (data[wc] & 0x7), 'W');
226         if (stat == NG) {
227             *cmdaaa = 0xaa; // Write Buffer abort
228             *cmd555 = 0x55;
229             *cmdaaa = 0xf0;
230             return(NG);
231         }
232         from += (wc + 1);
233         data += (wc + 1);
234     }
235 #else // Single Word Program
236     for(i = 0; i < len/2; i++){ // Word Write
237         *cmdaaa = 0xaa;
238         *cmd555 = 0x55;
239         *cmdaaa = 0xa0;
240         *from = data[i];
241         stat = FullStatusCheck(from, (data[i] & 0x80), 'W');
242         if (stat == NG) return(NG);
243         ++from;
244     }
245 #endif
246
247     verf = (USHORT *)adr; // Verify
248     data = (USHORT *)&cmd[8]; // Data Adr set
249     for(i = 0; i < len/2; i++){ // VerifyTest
250         if (verf[i] != data[i]) { // VerifyError
251             FromErrorSet((long)'V', (long)&verf[i], (long)data[i]);
252             return(NG);
253         }
254     }
255
256     return(OK);
257 }
258 //*****
  
```

バッファ書き込みモード

シングル書き込みモード

バリファイ処理

8) 「フルステータスチェック」関数を作成する。

```

行番 S ソース
258 *****
259 FullStatusCheck()
260 フルステータスをチェックエラー発生時は、Word Access
261 を[FromError]に残す
262 FromError[0] == command 'W'or'D'
263 [1] == アクセスしたアドレス
264 [2] == FROM ステータス
265
266 return()
267 OK == 正常終了
268 NG == 異常終了
269 *****
270 int FullStatusCheck(UShort *adr,UShort data,UCHAR cmd)
271 {
272     UShort read1,read2,read3;
273     FromErrorSet(0,0,0); // Error情報を消去
274     while(1){
275         read1 = *adr;
276         read2 = *adr;
277         read3 = *adr;
278         if (cmd == 'D'){ // Erase
279             if((read1 & DQ3) == DQ3){
280                 if(((read1 ^ read2) & DQ2) == DQ2) && // toggles
281                    (((read2 ^ read3) & DQ2) == DQ2)){
282                     if((read1 & DQ5) == DQ5){ // Time Over
283                         FromErrorSet((long)cmd,(long)adr,(long)read1);
284                         return(NG);
285                     }
286                 }
287                 else {
288                     if((read1 & DQ7) == data) break; // Done
289                 }
290             }
291         }
292         else { // Programming
293             if(((read1 ^ read2) & DQ6) == DQ6) && // toggles
294                (((read2 ^ read3) & DQ6) == DQ6)){
295                 if(((read1 & DQ1) == DQ1) || // Write Abort
296                    ((read1 & DQ5) == DQ5)){ // Time Over
297                     FromErrorSet((long)cmd,(long)adr,(long)read1);
298                     return(NG);
299                 }
300                 else {
301                     if((read1 & DQ7) == data) break; // Done
302                 }
303             }
304         }
305     }
306     return(OK);
307 }
308 *****
309
  
```

Erase 処理の
フルステータス
チェック

Write 処理の
フルステータス
チェック

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
 //‘D’=イレーズコマンド
 //‘W’=書き込みコマンド

<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無いので、自由に作成して下さい。

9) 作成したパラレル 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>

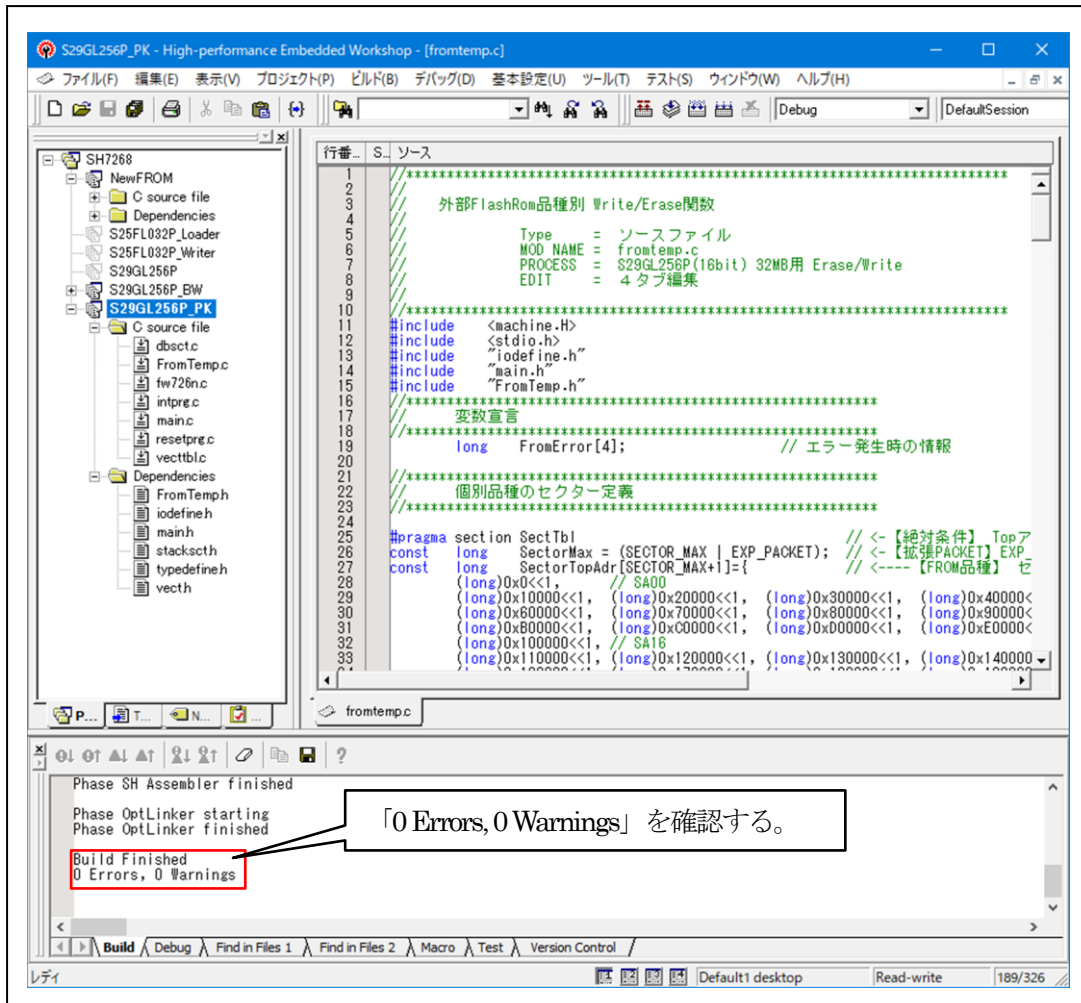
```
#define OK 0 // 変更不可
#define NG 0xee // 変更不可
```

 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);

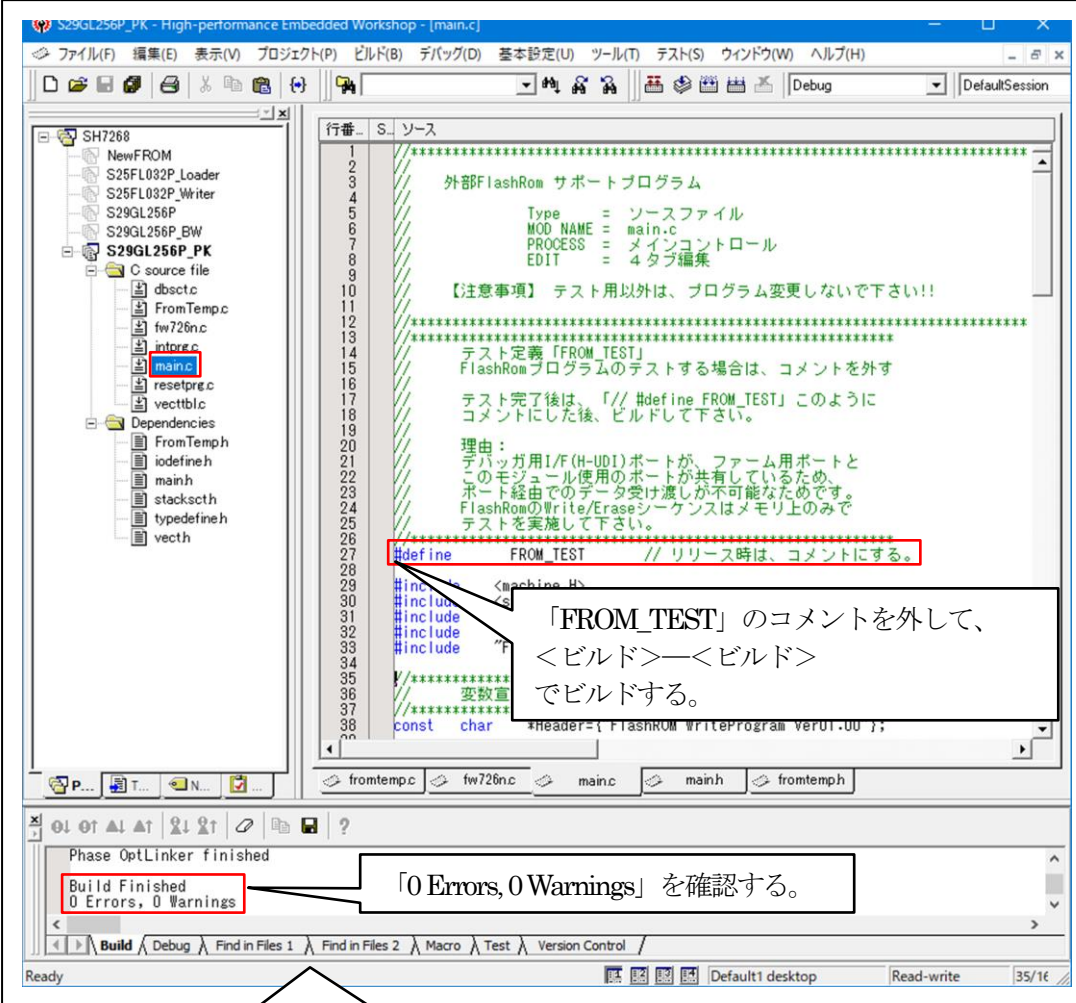
以上のルールは厳守して下さい。

(2) ビルドの実施



5. パラレル FlashROM プログラムのデバッグ方法

5-1) 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 S29GL256P_PK project. The left pane displays the project tree with 'main.c' selected. The main editor shows the source code of 'main.c', which includes a section for testing the external FlashROM. 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 pane shows the build output, with a red box highlighting the message: `Build Finished
0 Errors, 0 Warnings`. A callout box points to this message 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。

5-2) 「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、ターゲットの内蔵RAMに転送して実行させますので「RAM」側を指定して先頭終アドレスとサイズを指定する。

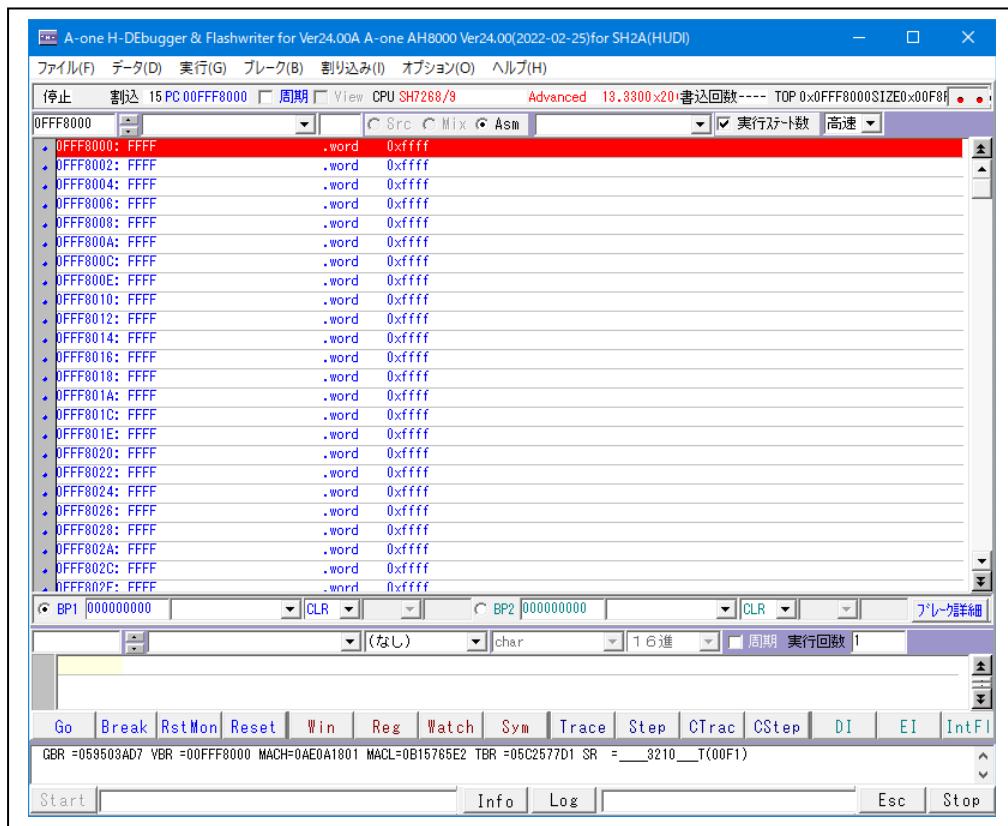
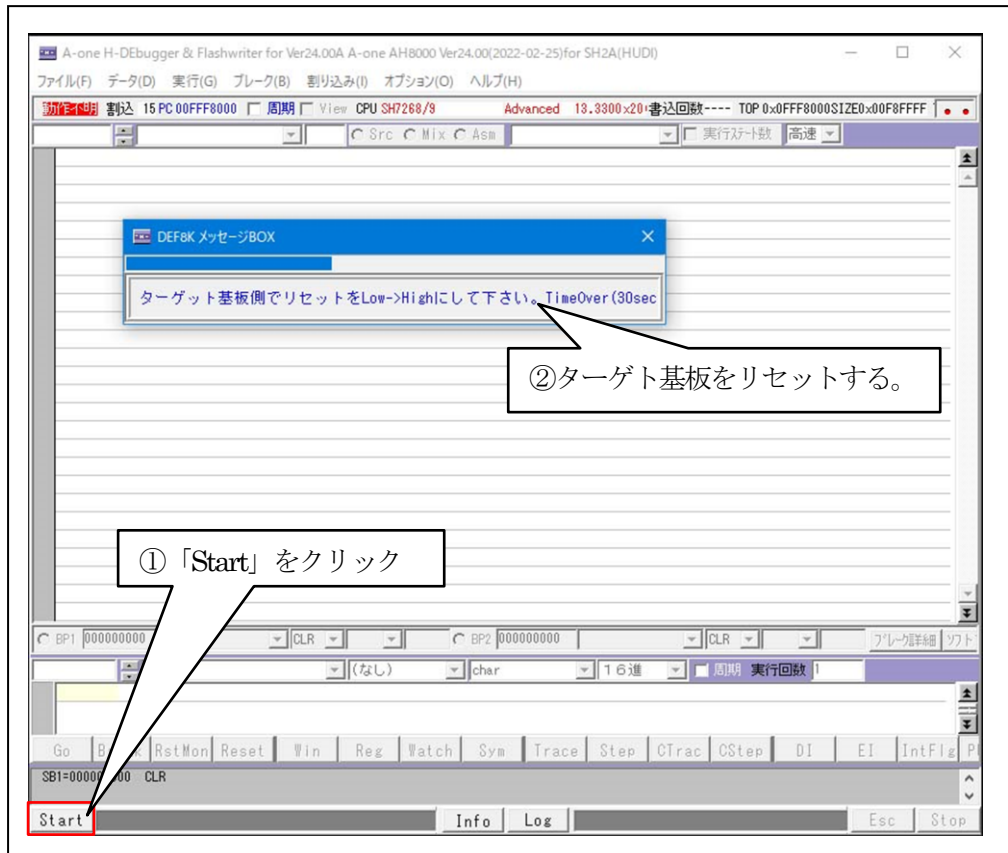
SH7268の場合
開始 0xffff8_0000 サイズ 0x10000になる。

ターゲット側の基板構成による

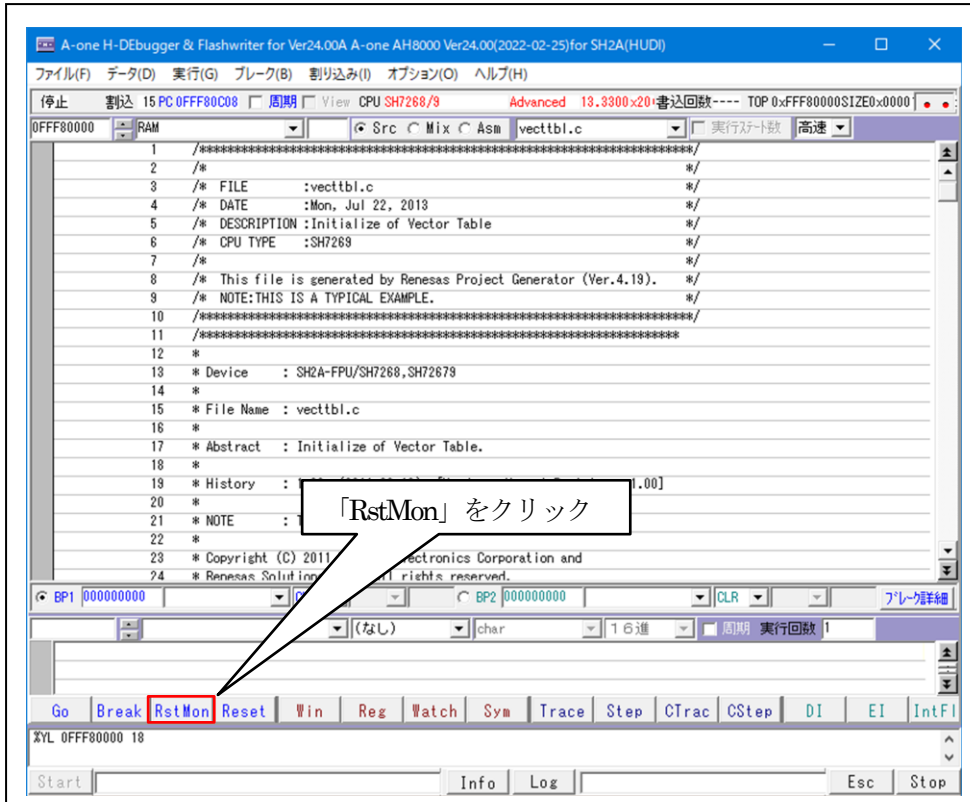
作成した「FlashROM」プログラムのデバッグにBSC設定が必要な場合は、スクリプトファイルを指定する。
SH7268の場合
"c:s\Aone\DEF8K\rom-custom\SH7268"に、
例として「SH7268-BSC.log」を用意してあるので、目的ターゲット基板にカスタマイズして下さい。

5-3) 「DEF8K」でデバッグを開始する。

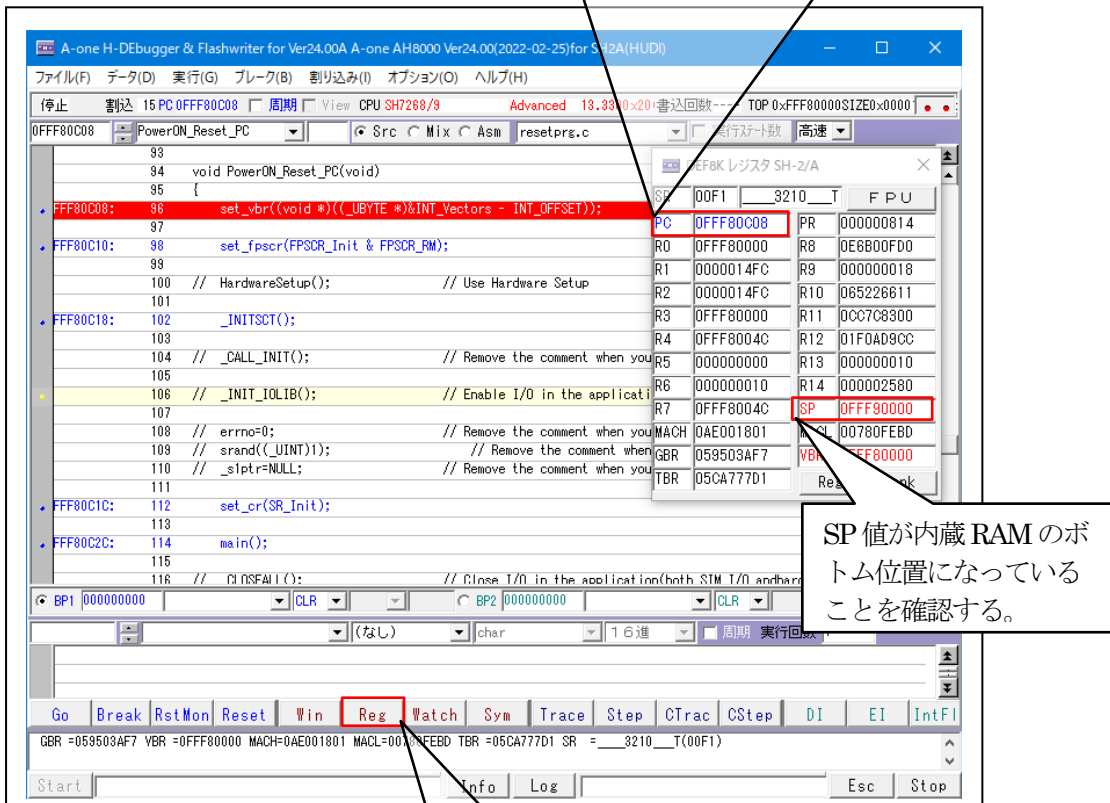
1) ターゲットと接続する。



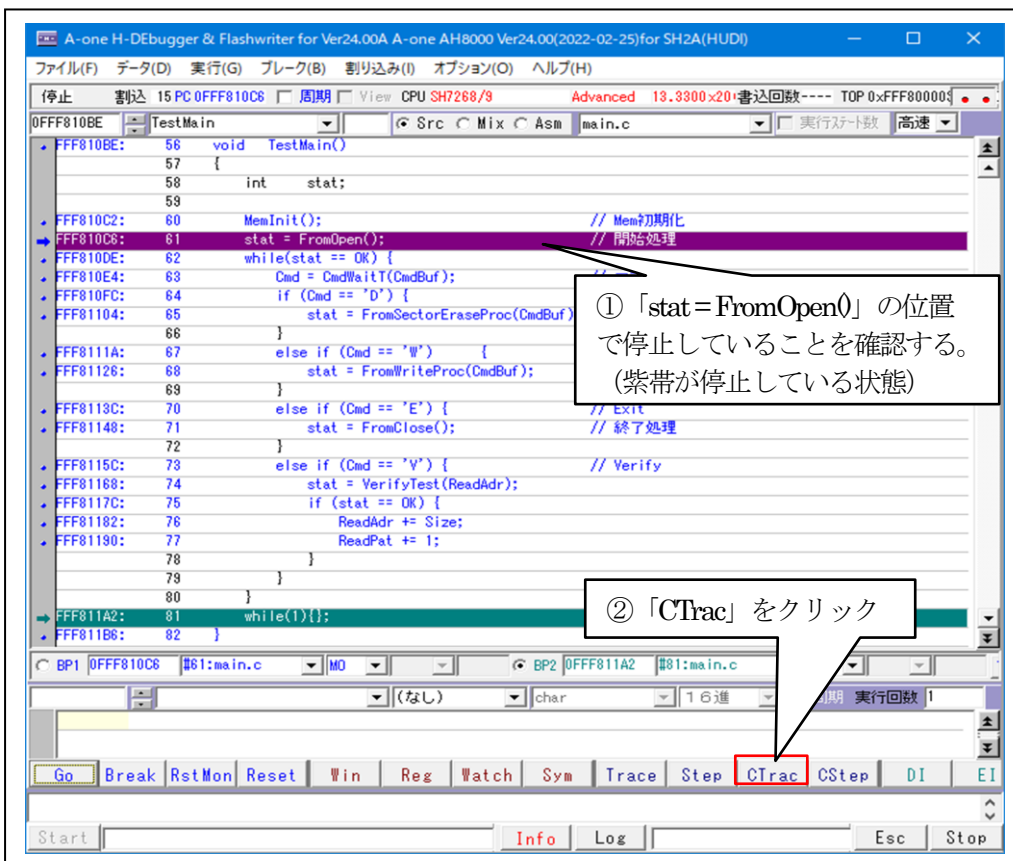
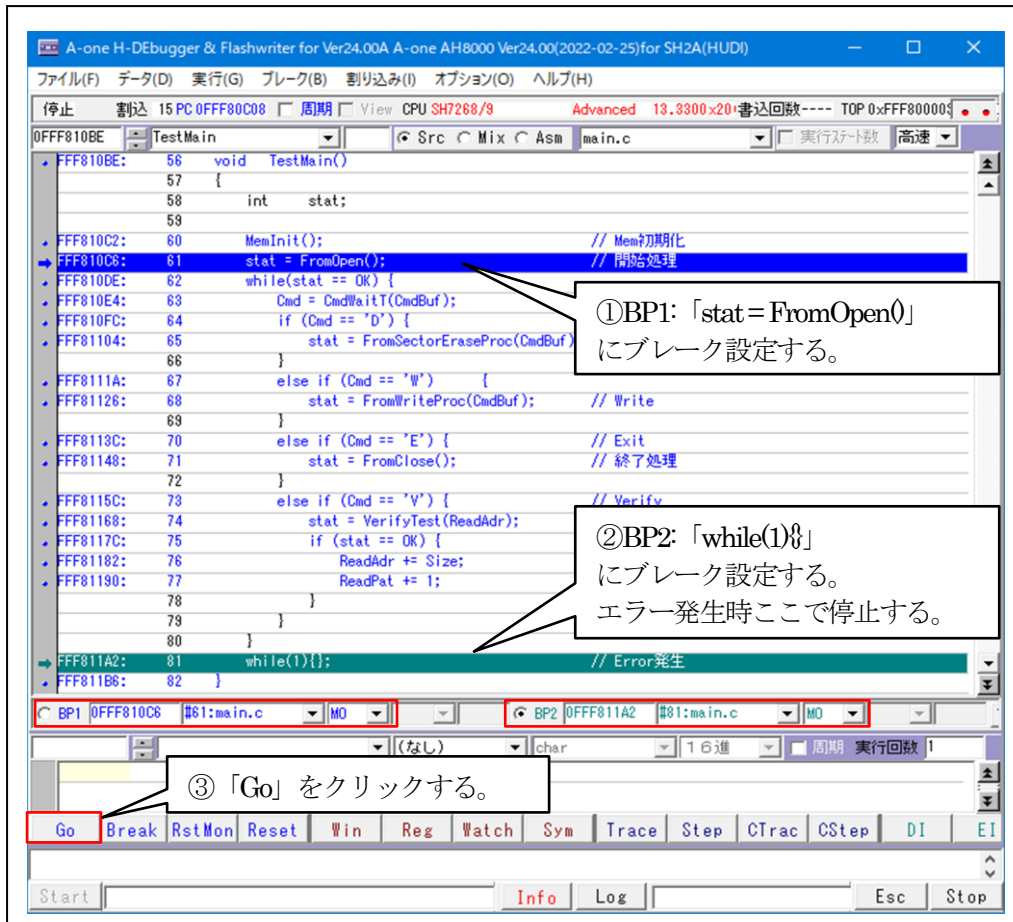
3) 「RstMon」 をクリックする。



「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。

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

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 0xFF8183C 周期 View CPU SH7268/9 Advanced 13.3300x20 書込回数----

0xFF8183C FromOpen Src Mix Asm FromTemp.c 実行スタート数

```

102 // NG == 異常終了
103 //*****
104 int FromOpen(void)
105 {
106     CPG.FROCR.WORD = 0x335; // FROCR set I=x5 B=x5 P1=x5 P0=x2
107     CCNT.CCR1.LONG = 0; // キャッシュ無効
108
109     return(OK);
110 }
111 //*****
112 // FromClose()
113 // FlashRom書き込み処理が全て終了した場合に必要な手続き
114 // 不要な場合、プログラム記述しなくても良い
115 // 終了時に1回だけ呼ばれる
116 // return()
117 // OK == 正常終了
118 // NG == 異常終了
119 //*****
120 int FromClose(void)
121 {

```

「FromOpen()」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「CTrac」をクリックする。

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

GBR =059503EF3 YBR =0xFF80000 MACH=0AE081801 MACL=046B75EC9 TBR =05CA777D1 SR = __3210__(00F0)

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 0xFF810DE 周期 View CPU SH7268/9 Advanced 13.3300x20 書込回数---- TOP 0xFF80000

0xFF810BE TestMain Src Mix Asm main.c 実行スタート数 高速

```

56 void TestMain()
57 {
58     int stat;
59
60     MemInit(); // Mem初期化
61     stat = FromOpen(); // 開始処理
62     while(stat == OK) {
63         Cmd = CmdWaitT(CmdBuf);
64         if (Cmd == 'D') {
65             stat = FromSectorEraseProc(CmdBuf);
66         }
67     }
68     else if (Cmd == 'W') {
69         stat = FromWriteProc(CmdBuf); // Write
70     }
71     else if (Cmd == 'E') {
72         stat = FromClose(); // 終了処理
73     }
74     else if (Cmd == 'V') {
75         stat = VerifyTest(ReadAdr);
76         if (stat == OK) {
77             ReadAdr += Size;
78             ReadPat += 1;
79         }
80     }
81     while(1){}; // Error発生
82 }

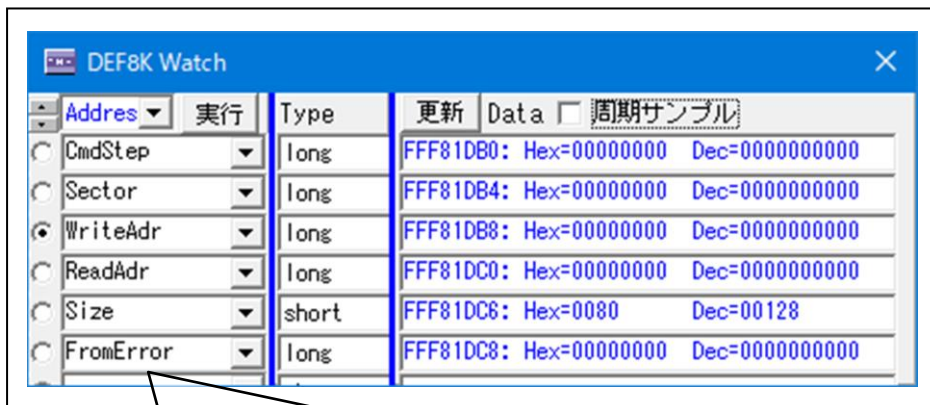
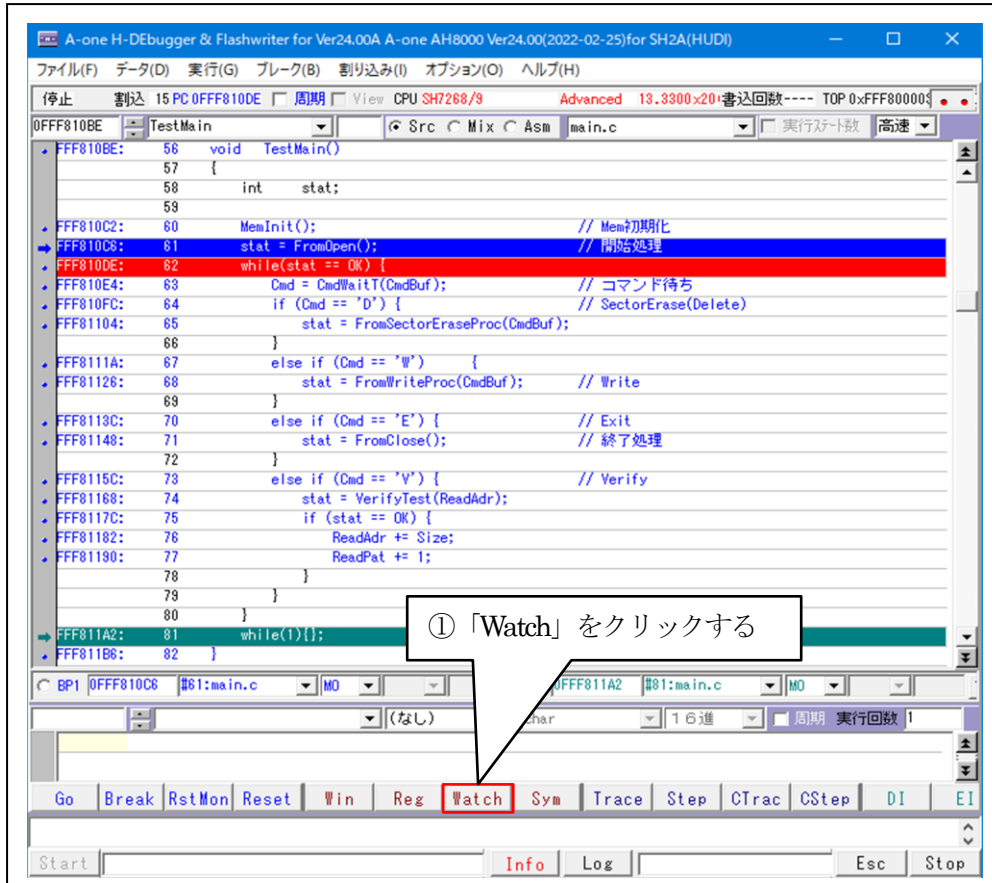
```

「main.c」に戻ったのを確認する。

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

Start Info Log Esc Stop

6) デバッグを進めるため準備するため「Watch」画面を開く。

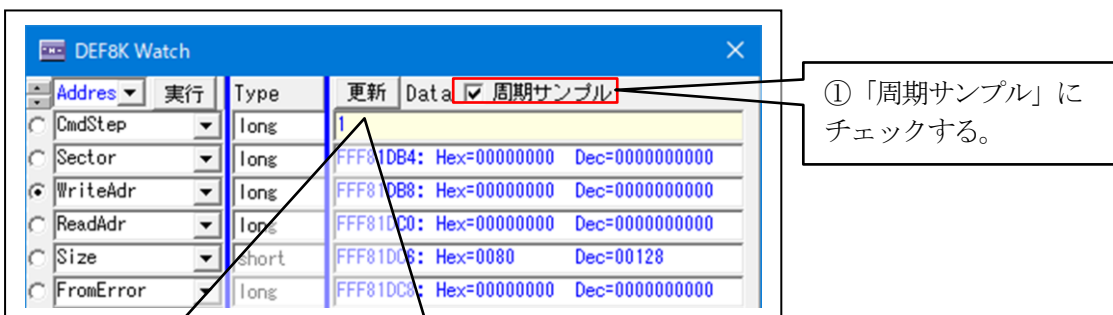
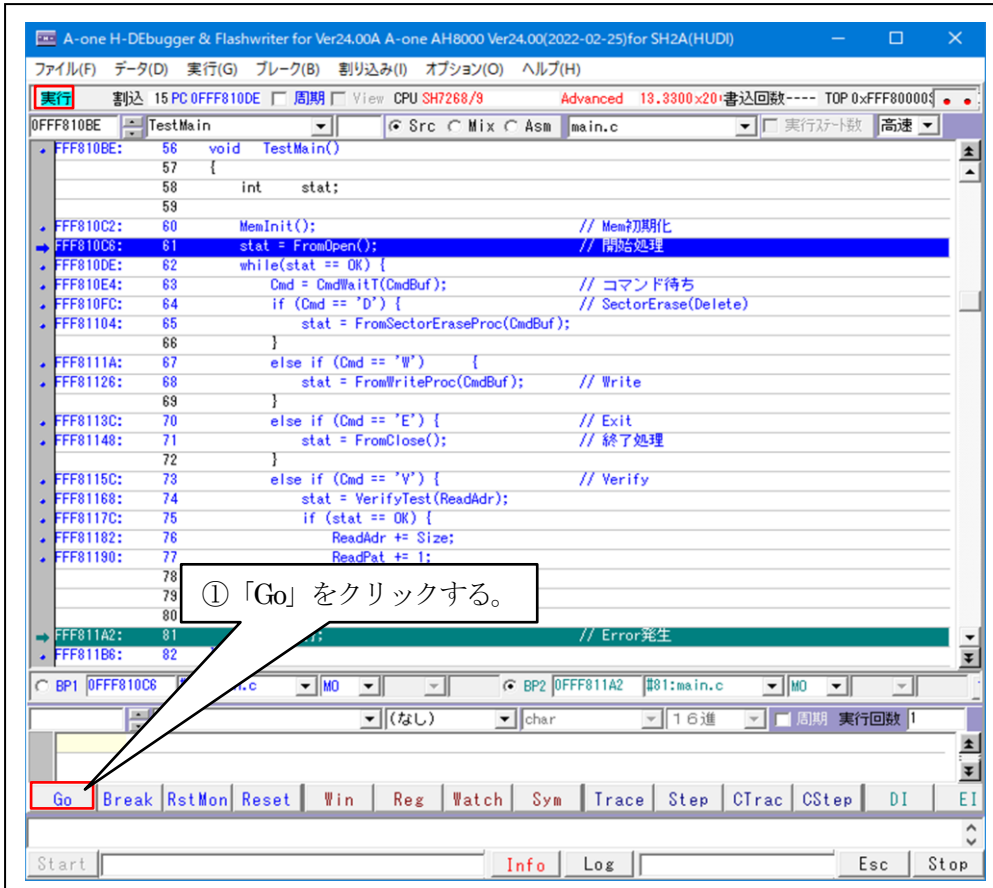


デバッグを進めるため、Watch 画面に変数を登録する。

- 1) CmdStep 関数「CmdWaitT」のコントロールステップ変数
- 2) Sector セクターイレースの進行カウンター
- 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「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。

Address	実行	Type	更新	Data	周期サンプル
CmdStep		long	FFF81DB0: Hex=00000000	Dec=000000000	
Sector		long	FFF81DB4: Hex=00000100	Dec=000000256	
WriteAdr		long	FFF81DB8: Hex=00000000	Dec=000000000	
ReadAdr		long	FFF81DC0: Hex=00000000	Dec=000000000	
Size		short	FFF81DC6: Hex=0080	Dec=00128	
FromError		long	FFF81DC8: 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。

【ダンプで確認する。】 FlashROM のアドレス(0x0~0x1FF_FFFF)

00FFFF00	0xFF_FF00	メモリダンプ	char	16進	周期	実行回数	adr {0xFFFF00}
00FFFF00:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFF10:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFF20:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFF30:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFF40:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFF50:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFF60:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFF70:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFF80:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFF90:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFA0:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					
00FFFB0:	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF					

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

Addr	実行	Type	更新	Data	周期サンプル
CmdStep		long	2		<input checked="" type="checkbox"/>
Sector		long	FFF81DB4: Hex=00000100 Dec=000000256		
WriteAdr		long	FFF81DB8: Hex=00000000 Dec=000000000		
ReadAdr		long	FFF81DC0: Hex=00000000 Dec=000000000		
Size		short	FFF81DC6: Hex=0080 Dec=00128		
FromError		long	FFF81DC8: Hex=00000000 Dec=000000000		

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)

③変数「CmdStep」を数値「2」にする。// ↵2↵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。

Addr	実行	Type	更新	Data	周期サンプル
CmdStep		long	FFF81DB0: Hex=00000000 Dec=000000000		<input checked="" type="checkbox"/>
Sector		long	FFF81DB4: Hex=00000100 Dec=000000256		
WriteAdr		long	FFF81DB8: Hex=02000000 Dec=033554432		
ReadAdr		long	FFF81DC0: Hex=00000000 Dec=000000000		
Size		short	FFF81DC6: Hex=0080 Dec=00128		
FromError		long	FFF81DC8: Hex=00000000 Dec=000000000		

「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。

【ダンプで確認する。】FlashROMのアドレス(0x0~0x1FF_FFFF)

```

01FFFF00: 30 31 46 46 46 30 30 FE FE FE FE FE FE FE FE FE FE |01FFFF00.....
01FFFF10: FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE |.....
01FFFF20: FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE |.....
01FFFF30: FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE |.....
01FFFF40: FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE |.....
01FFFF50: FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE |.....
01FFFF60: FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE |.....
01FFFF70: FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE |.....
01FFFF80: 30 31 46 46 46 38 30 FF FF FF FF FF FF FF FF FF |01FFFF80.....
01FFFF90: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |.....
01FFFA00: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |.....
01FFFB00: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |.....
    
```

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

9) パラレルFlashROM 全エリアをバリファイする。



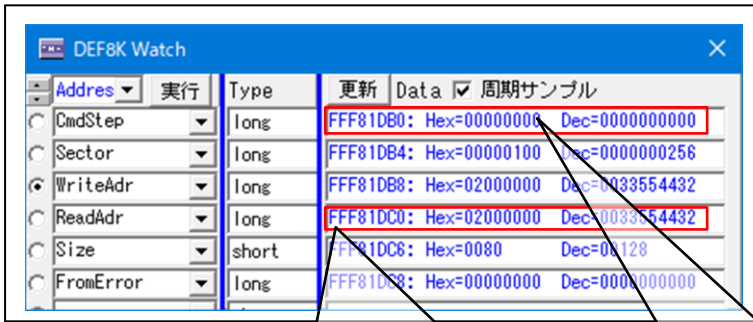
①「周期サンプル」にチェックする。

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。

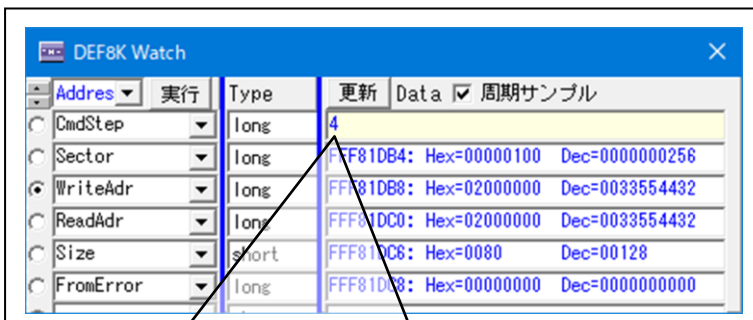
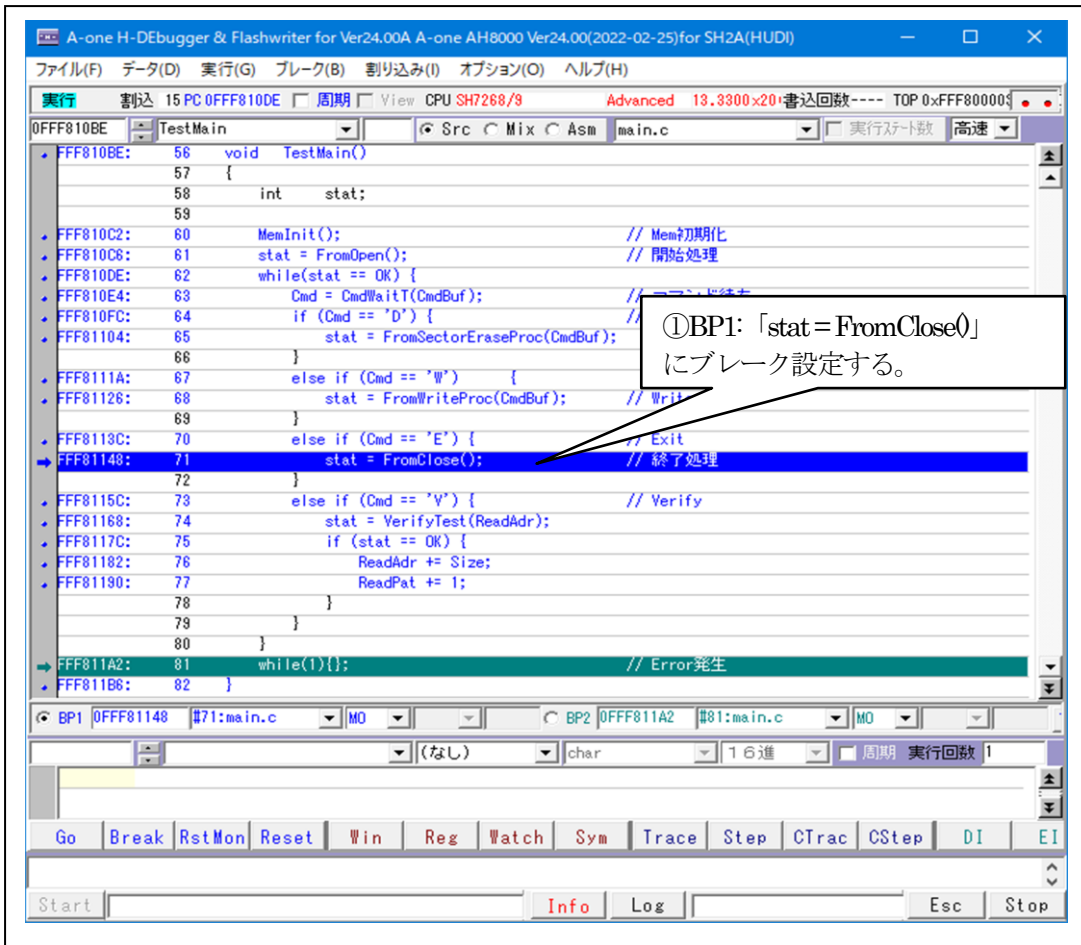


「ReadAdr」は、FlashROMの読み出し先頭アドレス「0x0」から登録最終アドレスまでインクリメントする。

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。

10) 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「CmdWaitT0」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 0FFF81148 周期 View CPU SH7268/9 Advanced 13.3300x20 書込回数---- TOP 0x0FFF80000

0FFF810BE TestMain Src Mix Asm main.c 実行スタート数 高速 Warn

```

56 void TestMain()
57 {
58     int stat;
59
60 MemInit(); // Mem初期化
61 stat = FromOpen(); // 開始処理
62 while(stat == OK) {
63     Cmd = CmdWaitT(CmdBuf); // コマンド待ち
64     if (Cmd == 'D') { // SectorErase(Delete)
65         stat = FromSectorEraseProc(CmdBuf);
66     }
67     else if (Cmd == 'W') {
68         stat = FromWriteProc(CmdBuf); // Write
69     }
70     else if (Cmd == 'E') { // Exit
71         stat = FromClose(); // 終了処理
72     }
73     else if (Cmd == 'V') {
74         stat = VerifyTest(ReadAdr);
75         if (stat == OK) {
76             ReadAdr += Size;
77             ReadPat += 1;
78         }
79     }
80 }
81 while(1){};
82 }
    
```

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状態)

② 「CTrac」をクリック

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

Start Info Log Esc Stop

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

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 0FFF81862 周期 View CPU SH7268/9 Advanced 13.3300x20 書込回数---- T

0FFF81862 FromClose Src Mix Asm FromTemp.c 実行スタート数

```

119 //*****
120 int FromClose(void)
121 {
122     return(OK);
123 }
124 //*****
125 // FromSectorEraseProc(char *
126 // 伝送フォーム(in)
127 // cmd[0] == %
128 // [1] == 0
129 // [2] == CommandSize[2]以降[Sum]も含む
130 // [3] == Command('D')
131 // [4] == Sector(H) MAX(1024)
132 // [5] == Sector(L)
133 // [Sum] == [3]->[Sum-1]の加算
    
```

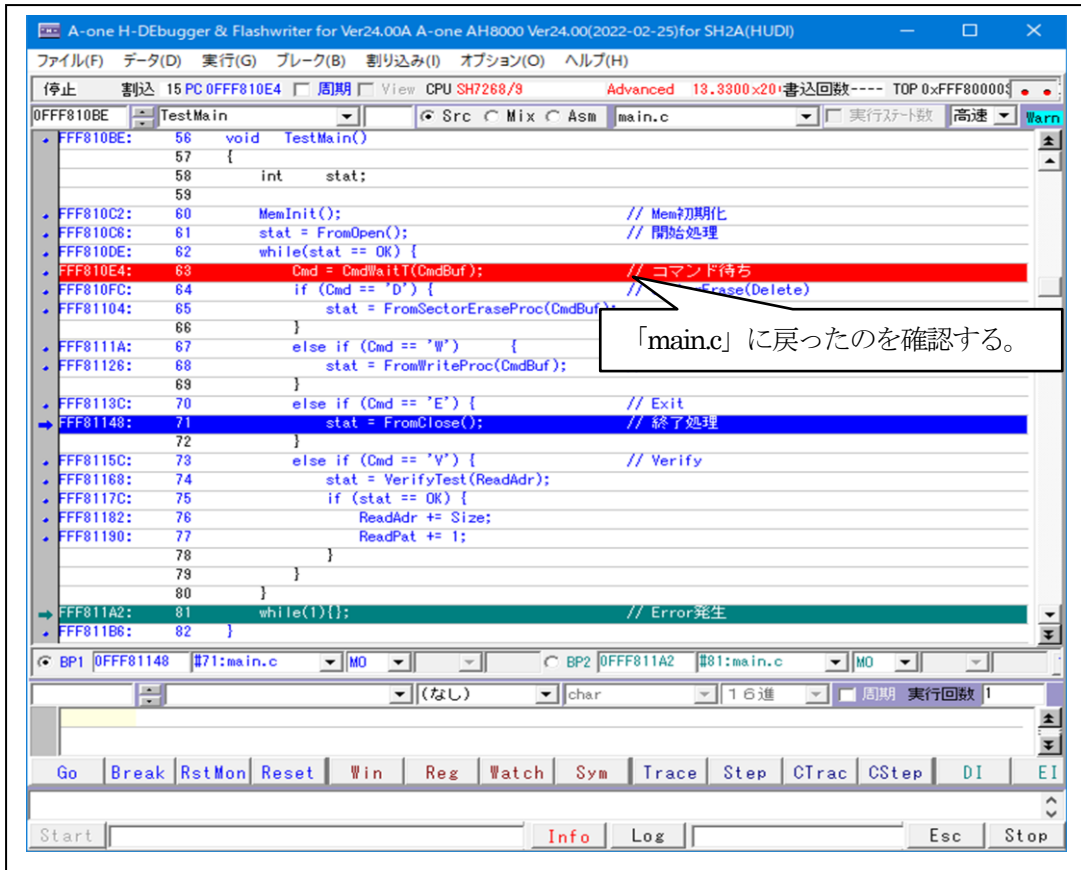
「FromClose()」関数内に PC 値が移動したことを確認する。

① 「main.c」に戻るまで「CTrac」をクリックする。

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

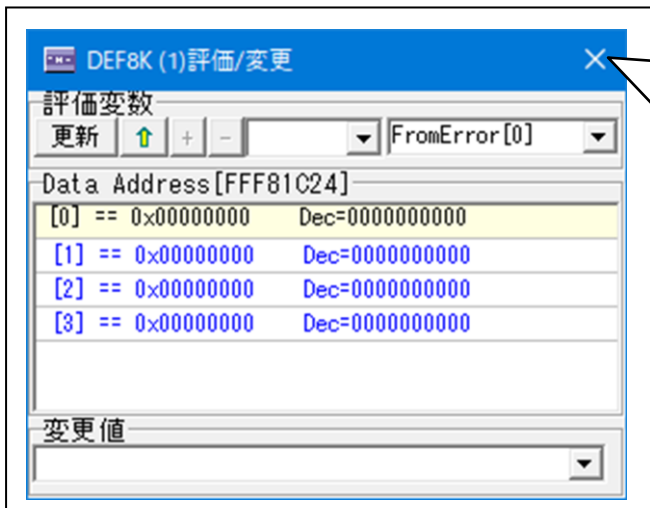
GBR =059503EF3 YBR =0FFF80000 MACH=0AE081801 MACL=046B75EC9 TBR =05CA777D1 SR = _3210_T(00F1)

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

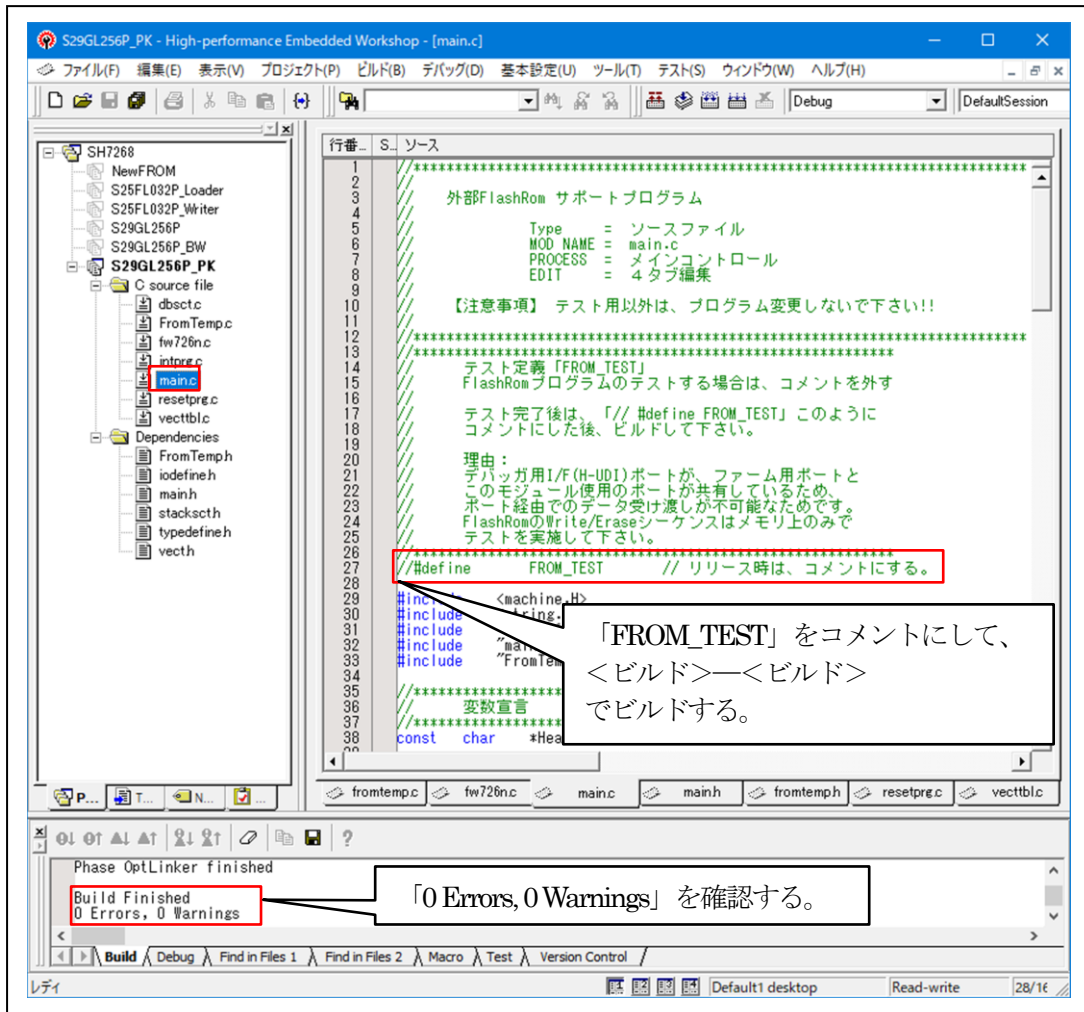


<評価・変更窓>

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無いので自由に変更して利用して下さい。

6. 平行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。

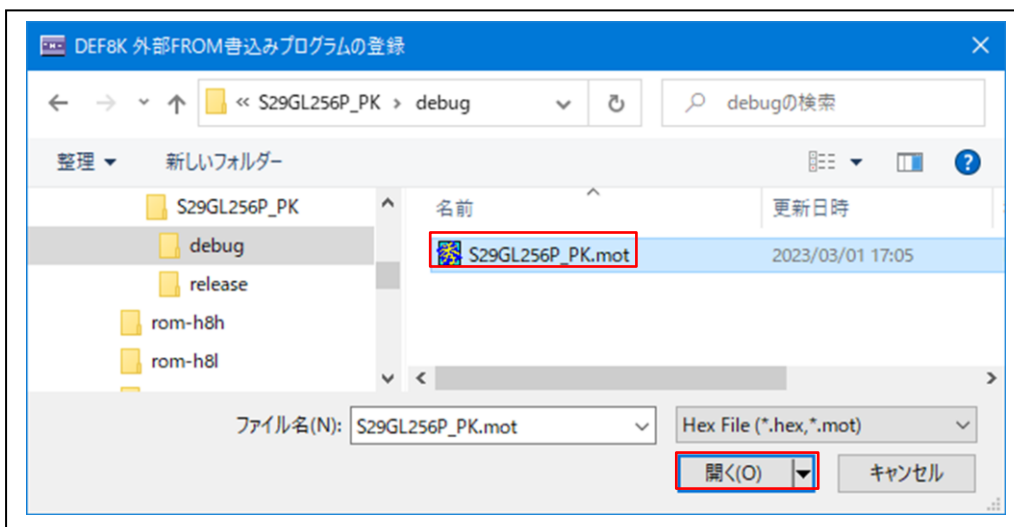


6-2) 「DEF8K」の「CPU設定」にリリース環境を設定する。

1) 「FROM」を選択する。



2) 「参照」をクリックして「FlashFROMプログラム」を登録する。



「FlashROMプログラム」の先頭アドレスとサイズを表示
また、登録した最終セクタを表示する。



指定した「FlashROMプログラム」の
Hex ファイル名を表示する。

パケット単位は
「128/256/512byte」の選択可

3) 「設定」をクリックする。

DEF8K CPU設定
✕

CPU選択を開始するタイプ
 H8 H8S H8SX SH2 SH2A R8C RX KIT

CPUタイプ
 CPUシリーズ名

CPU動作モード

発振子クロック(MHz)

周波数倍率(IΦ)

モニターワークエリアの場所

ターゲットI/F選択
 クロック(標準) クロック(高速) クロック(低速)
 調歩(高速) 調歩(中速) 調歩(低速) 誤差 %未満

デバッグモード(H8/Tiny)
 エミュレーション(トレース, 低消費電力可能)
 ユーザーモード(起動時RAMが書き換わらない)

ブートモード設定
 ブート時ボーレート

ダウンロード先のデバイス選択

RAM 開始 サイズ

FROM 消去ブロック 0 ~ (10進) 0=Auto パケット単位

参照

ダウンロード時に内蔵RAMの退避/復帰をしない。 RX[SPOC]書き込みを許可

リセット及び割込み制御関係(チェックにて有効)
 リセット遅延防止200msを使用しない。(ActiveHigh delay)
 動作中の電源OFF->ON時にリセット出力しない。
 リセットベクタを作成しない。(R8C専用)
 リセット出力を使用しない。(H-UDI[SH-2x])
 命令による例外割込みをデバッガから通知する。
 オンザフライ処理で割込みを使用しない。
 [4pin]EMLE(/ASEMD,/DBGMD,DBGMD)出力を使用する。(H-UDI)
 [7pin]MD出力をRST(OUT)で使用する。(H-UDI)

強制的にリセットベクター値を指定する。
 許可

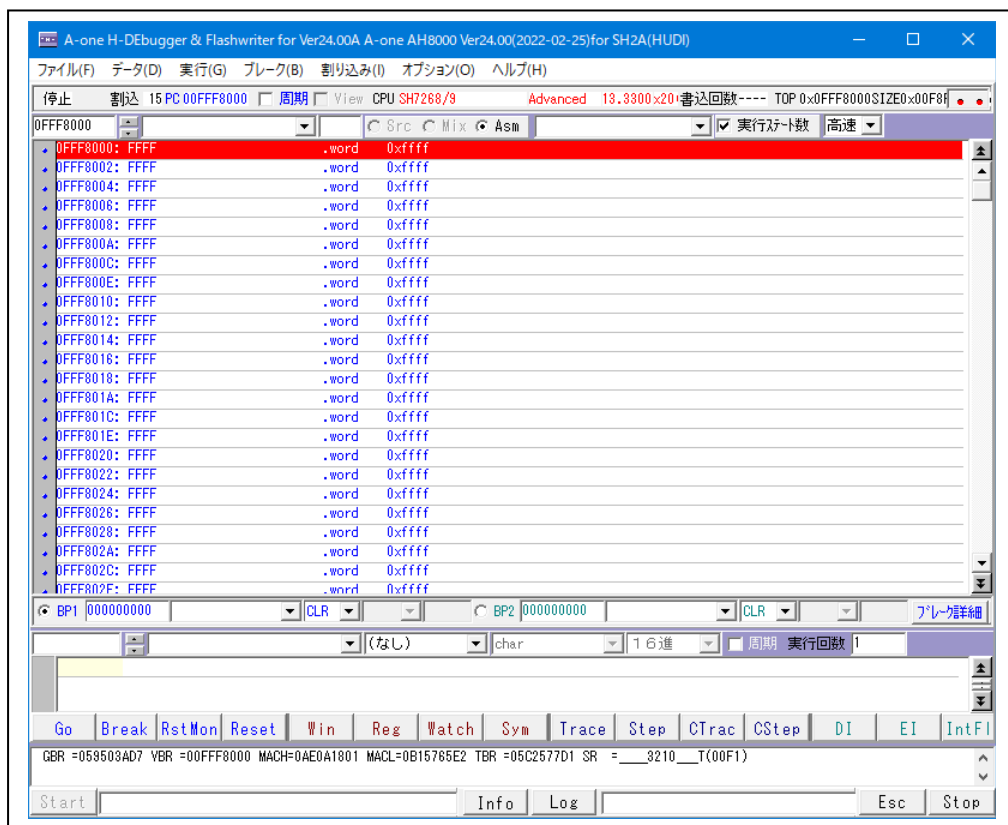
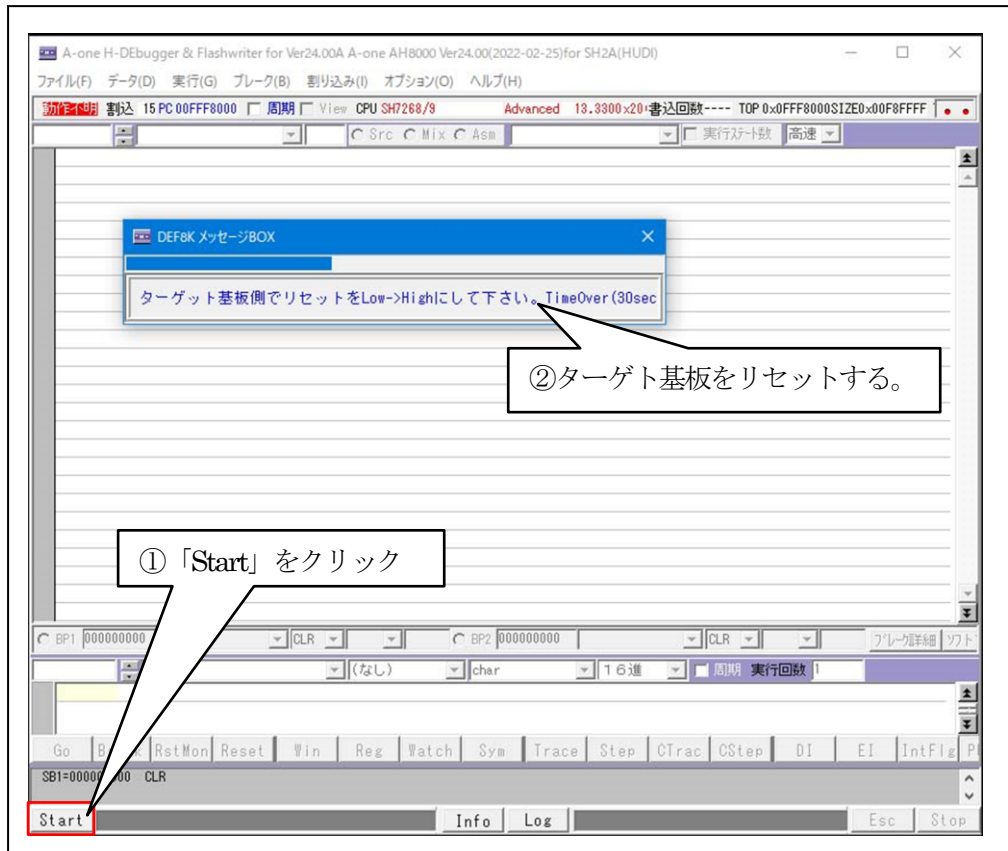
外部RAM時のBSC設定スクリプト指定
 許可 参照

ターゲット間の通信仕様
 HARD TAPで制御します。

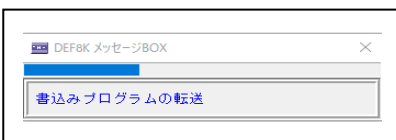
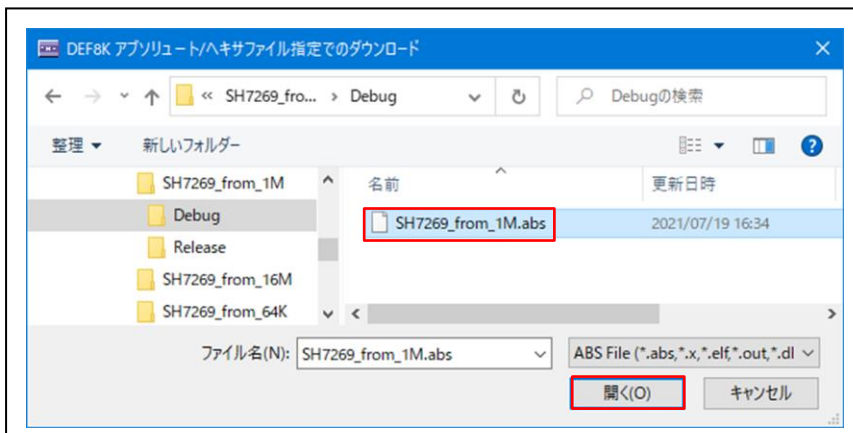
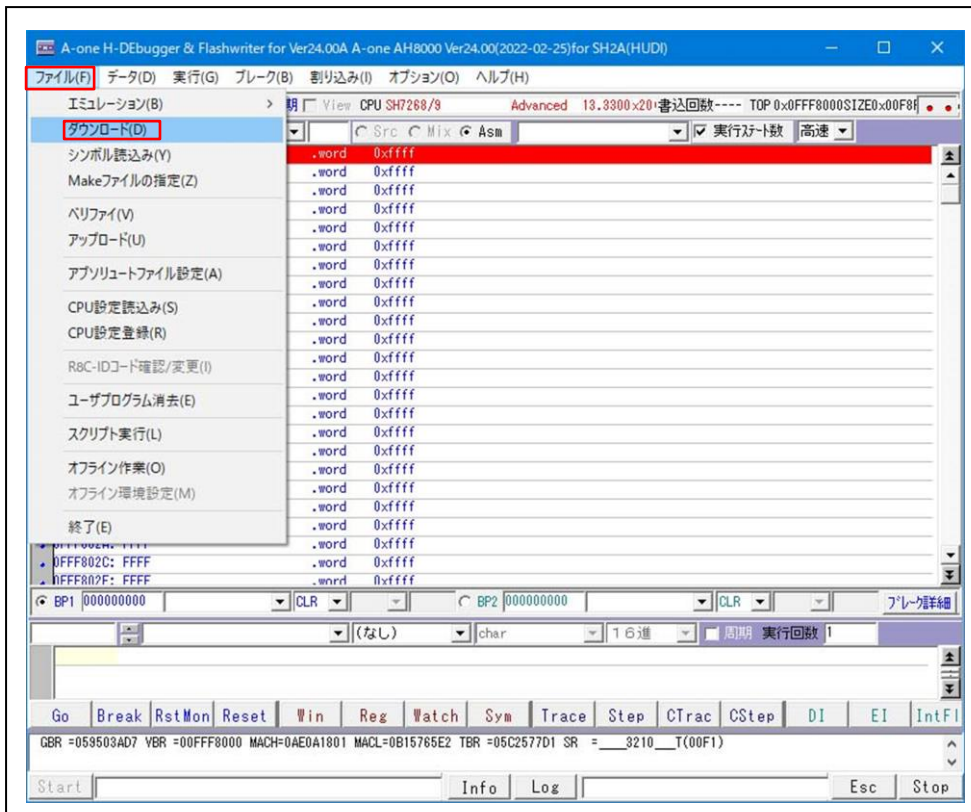
詳細情報
設定

7. パラレル FlashROM プログラムの最終確認をする。

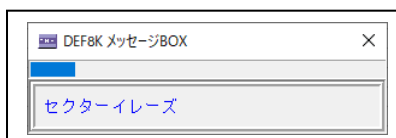
7-1) ターゲットと接続する。



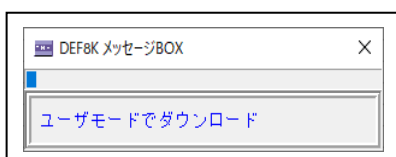
7-2) アプリケーションプログラムをダウンロードする。



① FROM 書き込みプログラムの転送



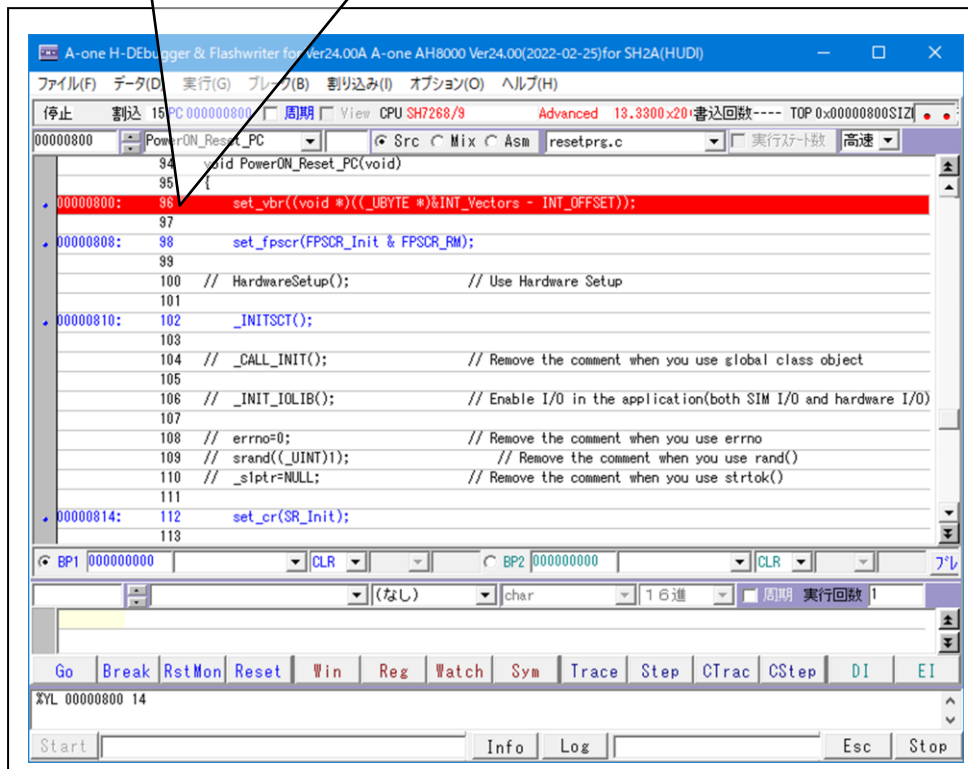
② FROM のセクターイレーズ



③ 拡張パケットでの FROM への書き込み

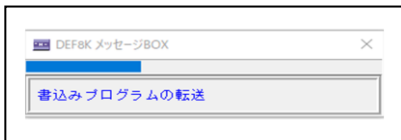
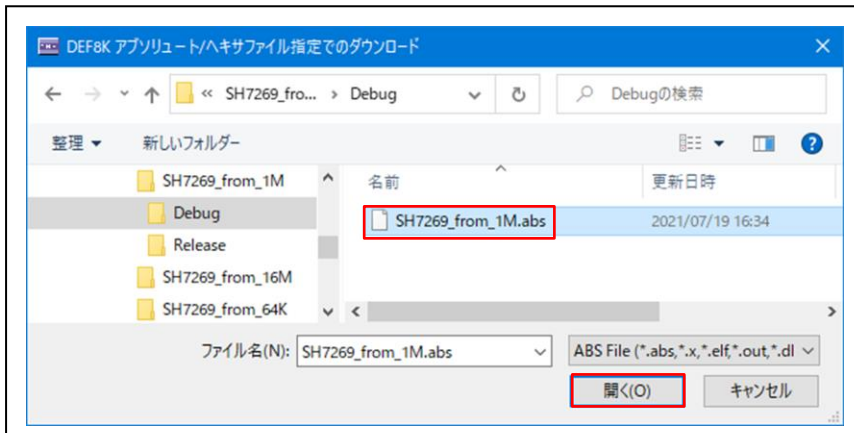
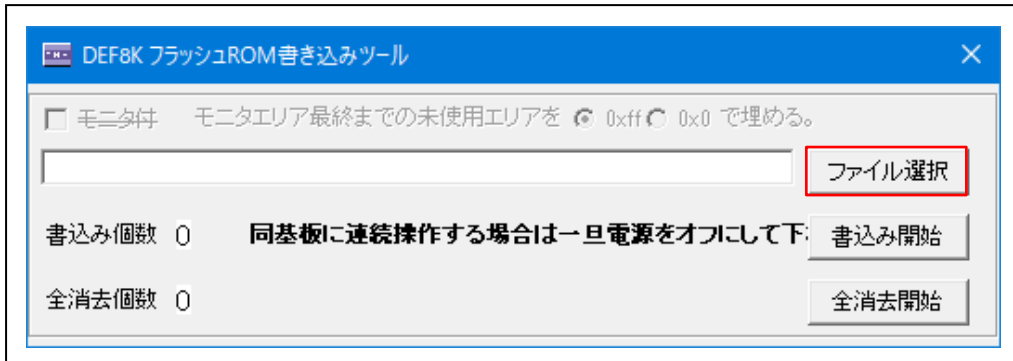
【ダウンロード正常終了画面】

「void PowerON_Rest_PC」関数の先頭で停止していることを確認する。

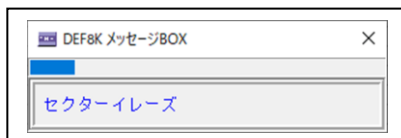


7-3) 「フラッシュ ROM 書き込みツール」で確認する。

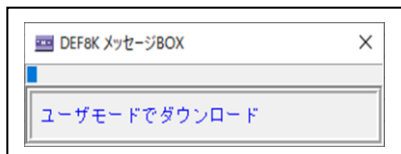
<オプション>-<フラッシュ ROM 書き込みツール>



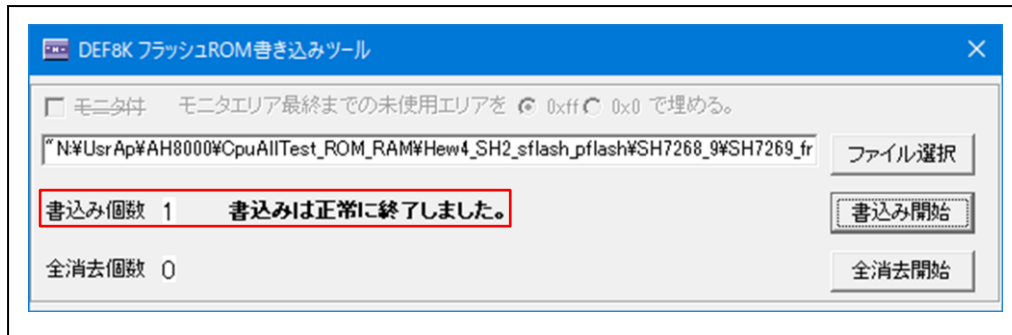
① FROM 書き込みプログラムの転送



② FROM のセクターイレース

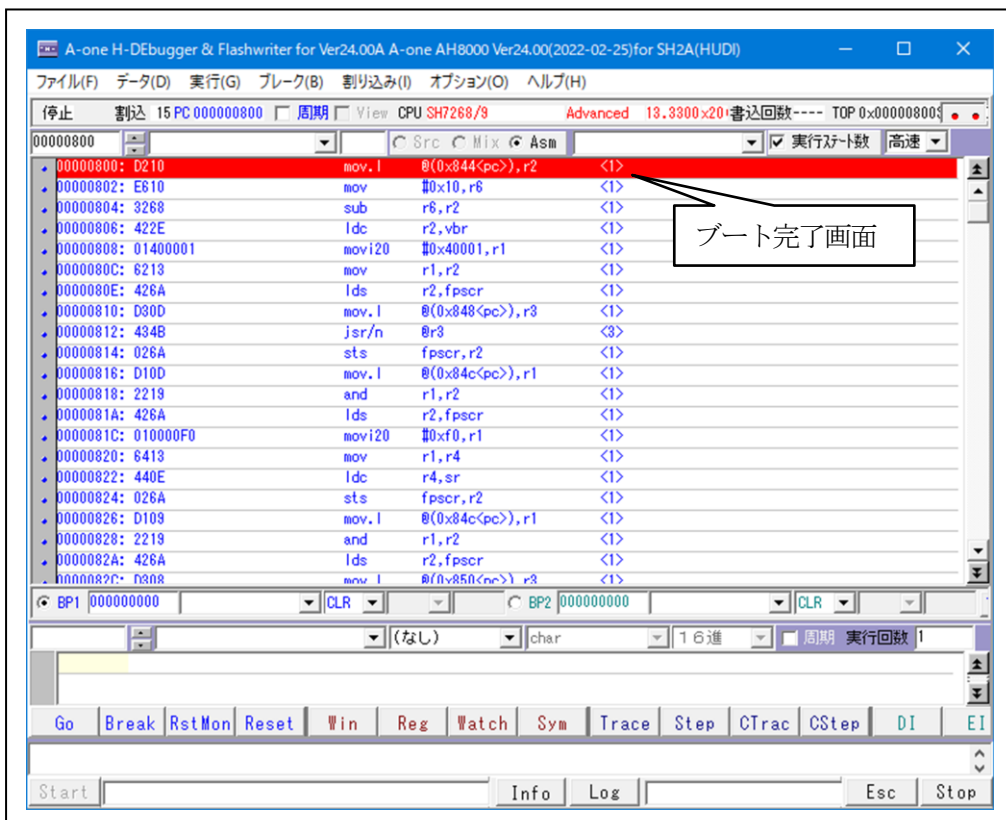
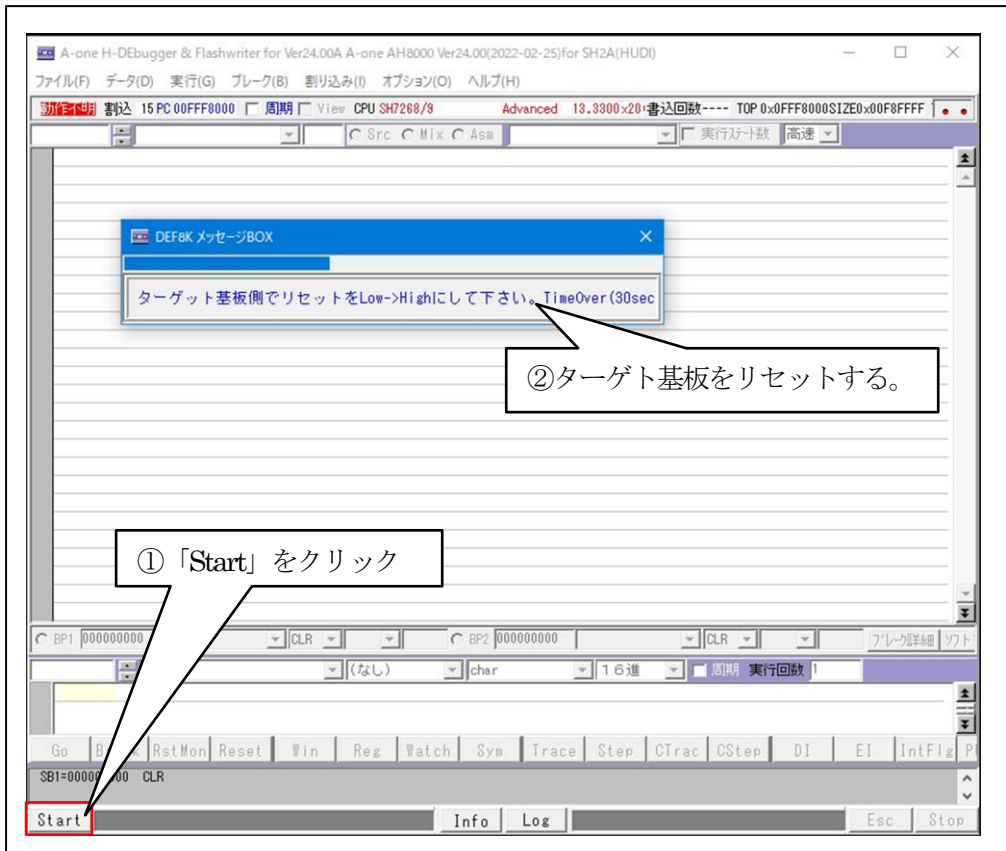


③ 拡張パケットでの FROM への書き込み

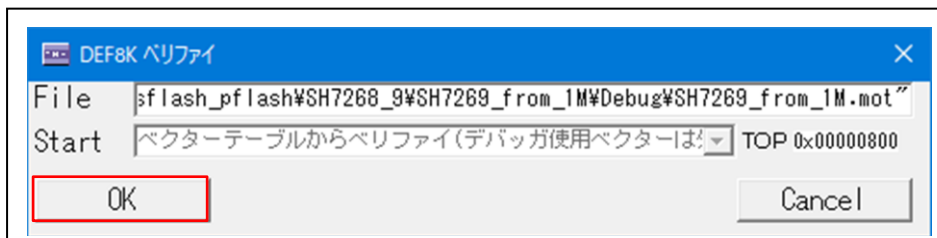
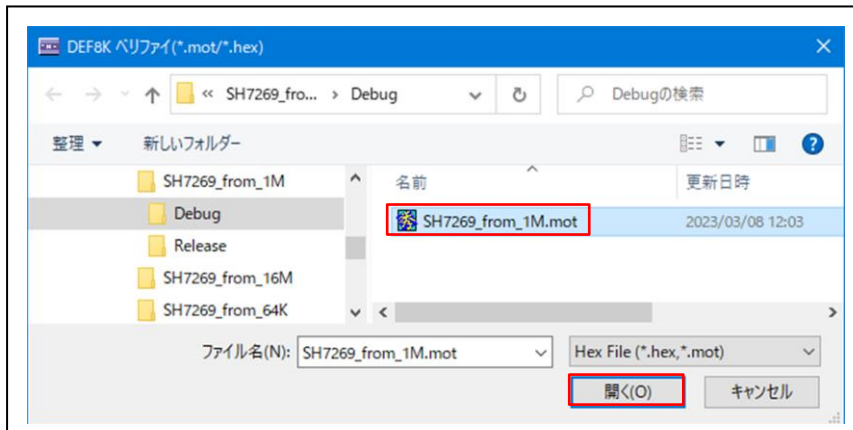
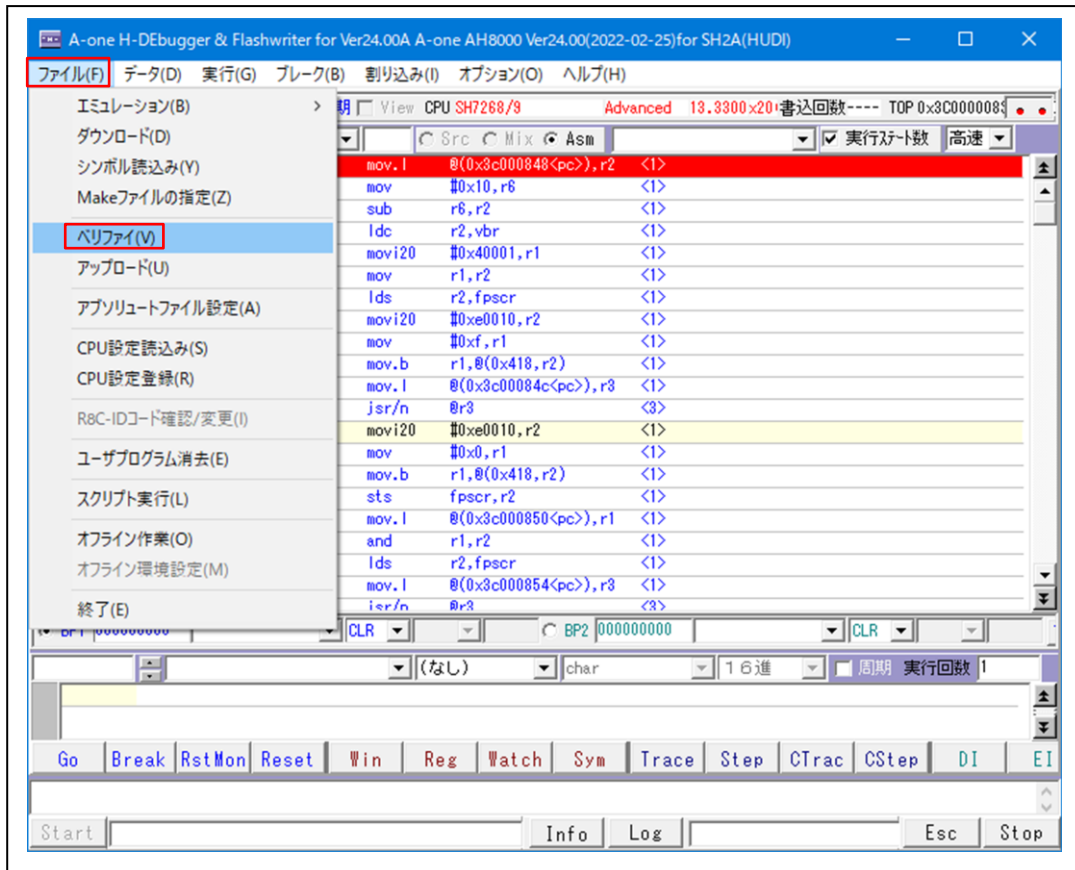


7-4) ベリファイで確認する。

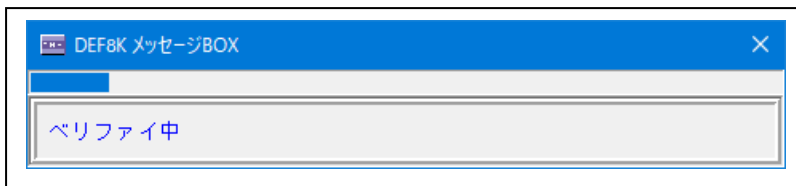
- 1) ターゲットの電源を再投入して接続する。



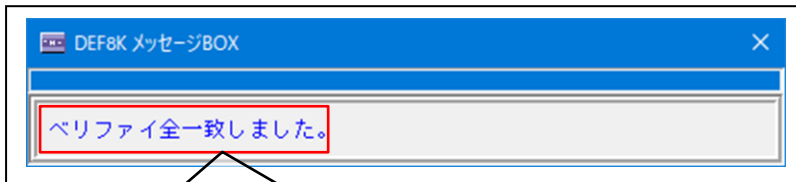
2) パラレルFlashROM とベリファイする。



【ベリファイ中】



【ベリファイ結果】



この結果表示によりパラレル FlashROM にアプリケーションプログラムが正常に書かれていることになる。

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>

