

## ROM 化プログラム (RAM に配置するプログラム) の持ち方

ROM 化するプログラムはロード・モジュールと HEX オブジェクトで持ち方が異なっています。

HEX オブジェクトの場合には, RAM に配置するプログラムは `_rcopy()` 関数の直後に出力されます。下図は RAM に配置するプログラムの中身のアセンブリ言語でのリストと出力された HEX オブジェクトの対比です。左側のリストの最初で `@@CODER` となっていることから RAM 配置であることが分かります。

このプログラムのオブジェクトコードを見ると, "C7", "20", "04", "FB", "F8", "FF" . . . . となっています。HEX オブジェクトで探してみると 00532 番地から格納されています。

ここで, 0049C 番地が `_rcopy()` 関数で, その後ろに `_rcopy()` 関数が対象にする RAM 配置するプログラムが入っていることがわかります。

RAM 配置であることを示す。

```
-----  
@@CODER CSEG↓  
00000 self_init:↓  
$DGL 1,19↓  
00000 C7          push    hl  
00001 2004         subw   sp,#04H  
00003 FBF8FF      movw   hl,sp  
00006        ??bf_self_init:↓  
: line 6↓  
$DGL 0,3↓  
00006 F6          clrw   ax  
00007 BB          movw   [hl],ax ; k  
: line 7↓  
$DGL 0,4↓  
00008 9C03        mov    [hl+3],a ; i  
0000A        ?L0003:↓  
0000A 8C03        mov    a,[hl+3] ; i  
0000C 4C64        cmp    a,#064H ; 100  
0000E DE0E        bnc   $?L0004  
: line 8↓  
00010        ??bb00_self_init:↓  
: line 9↓  
$DGL 0,6↓  
00010 8B          mov    a,[hl] ; k  
00011 0E03        add    a,[hl+3] ; i  
00013 70          mov    x,a  
00014 8C01        mov    a,[hl+1] ; k  
00016 1C00        addc  a,#00H ; 0  
00018 BB          movw   [hl],ax ; k
```

0049C 00C5C7148EFD1410011DB2805F643DD7480↓  
0049C 00AC00B145DD041345DC6C5100362A05F080C5DE↓  
0049C 004BC00C3C7C1C1F6B145C0DD06C1F145C0DF34CB↓  
0049C 0009EFD11AB1211AC021411AC04C111AC069F↓  
0049C 009EFD6C1118B089EFD081199A7F647C059↓  
0049C 0006118819EFD1A5F645C061F880C1B3F6E7↓  
0049C 0043C000DFC4C00408001661D885C2B3F66D↓  
0049C 004315C4D011C1F6B145C0DDA2C1F145C032↓  
0049C 000DF9CEF02F783C09EFD0C6C4D701002200DA↓  
0049C 000C4E932050F00C72004FBF8FF6BB9C039F↓  
0049C 0008C034C64DE0E8B0E03708C011C00BB61B3↓  
0049C 0005903EFEC1004C6D7BF↓  
0049C 020000020000FC↓

これを MAP ファイルで確認すると, 以下のようになっています。

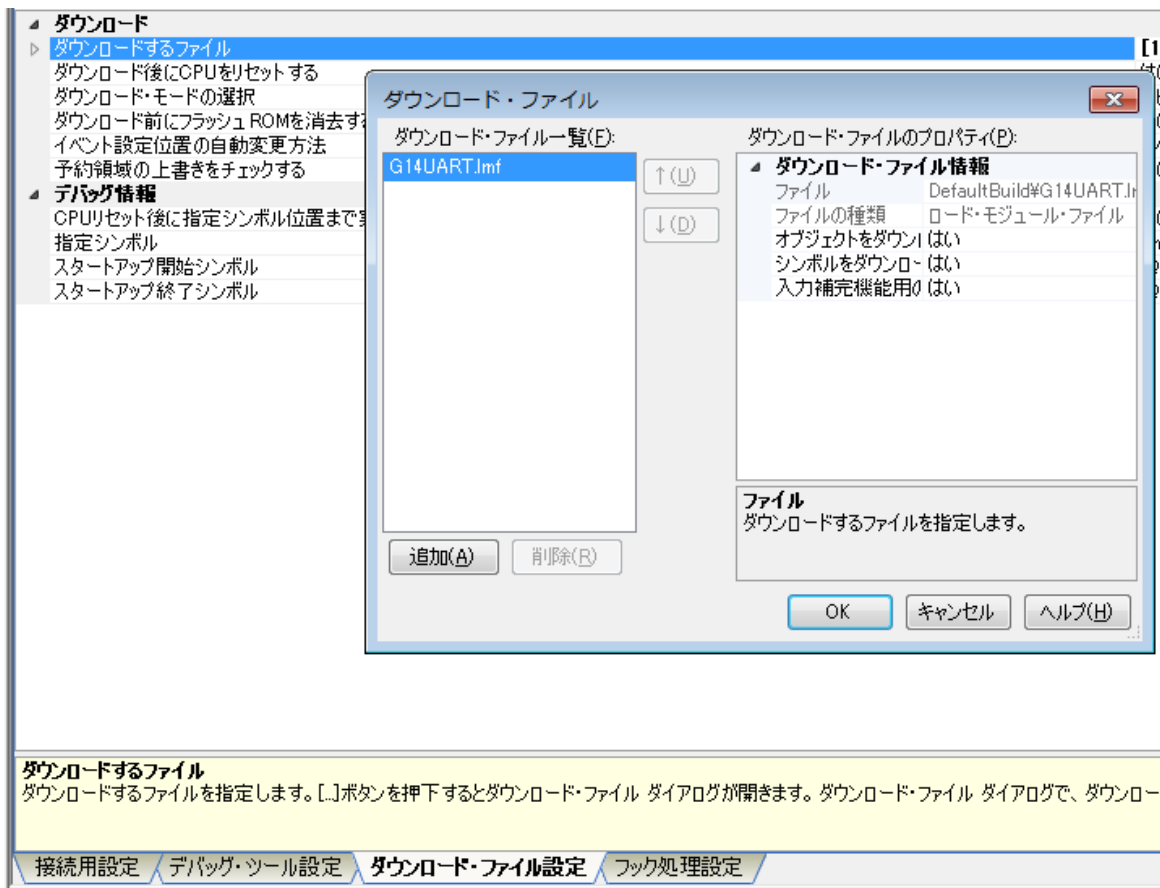
```
(@@ROMP ) 0049CH 000011H  
005E4H 0E04CH
```

RAM 配置のプログラムを MAP ファイルで確認すると, FE9C4 番地に配置されていることが分かります。`_rcopy()` 関数を実行することで, プログラムが実際の配置アドレス (FE9C4H) にコピーされます。

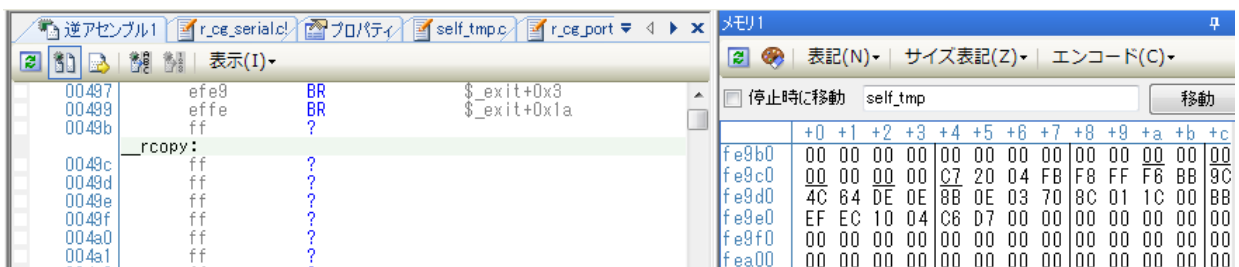
```
@@CODER FE9C4H 00022H CSEG↓  
@@CODER self tmp FE9C4H 00022H↓
```

では, 通常デバッグで使用するロード・モジュールではどうなっているかと言うと, これとは少し異なっています。

デバッガのプロパティで「ダウンロード・ファイル設定」タグの「ダウンロードするファイル」で確認すると, 次頁に示すようにロード・モジュールが一つ選択されて, そこで, 「オブジェクト」, 「シンボル」をダウンロードような設定になっています。




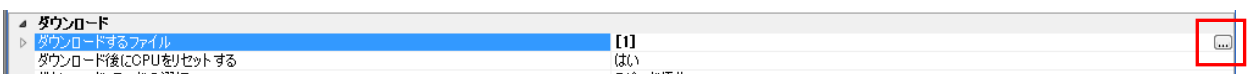
この状態でデバッガにダウンロードした結果を下に示します。049C 番地には何も入っていません。また、RAM の FE9C4H 番地からは RAM に配置したプログラムが既に入っています。

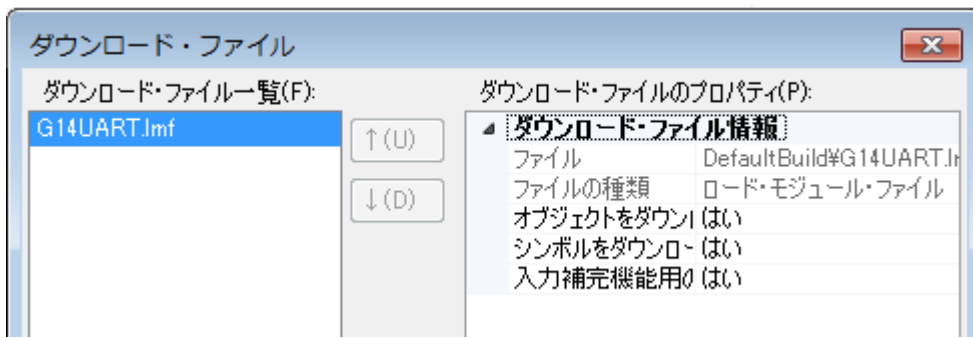


このまま単純に実行すると、`_rcopy()`関数のところで停止してしまいます。この状態を避けるには、デバッグ時には`_rcopy()`関数の呼び出しをコメントアウトしておくか、ダウンロードの設定を変更します。

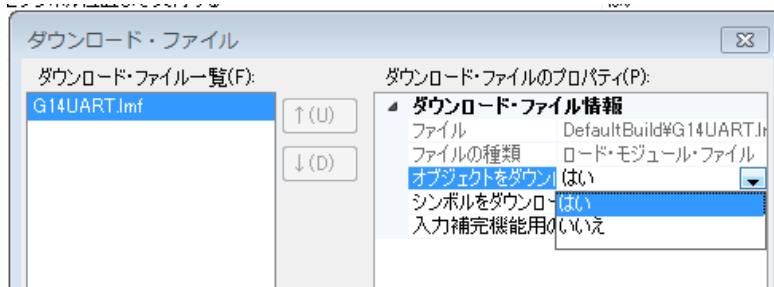
具体的には、オブジェクトはヘキサ・オブジェクトをダウンロードし、ロード・モジュールからはシンボルだけをダウンロードします。

デバッガのプロパティで「ダウンロード・ファイル設定」タグの「ダウンロードするファイル」画面を開きます。そこで、右の方の  (下図参照) をクリックして、次頁に示す設定画面を開きます。

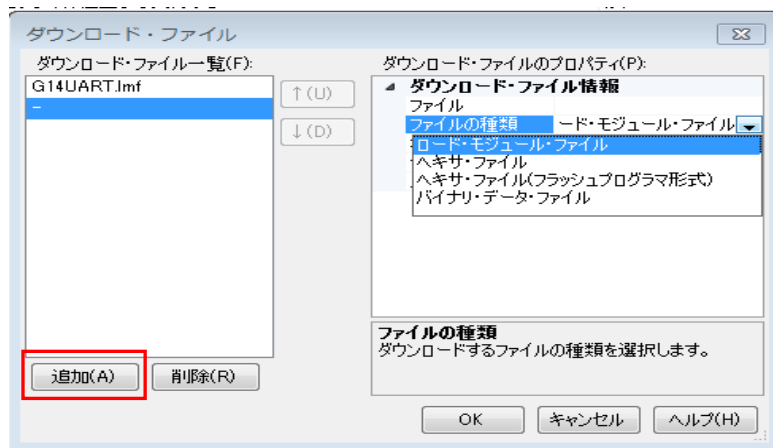




ここで、下の図のように、オブジェクトをダウンロードの部分が「はい」になっているのを「いいえ」に変更します。



次に左下の「追加(A)」をクリックしてダウンロード・ファイル一覧に「-」を追加します。「-」が選択された状態で、右側の「ファイルの種類」で「ヘキサ・ファイル」を選択します。



次に、右の「ファイル」でダウンロードする HEX オブジェクトを選択します。下の例では、G14UART.hex を選択しています。



この状態で、E1 にダウンロードさせてみます。その結果が次ページの図になります。main 関数の実行前の段階では RAM の内容は不定になっています。

