ILINK ガイド

基礎編

IAR Embedded Workbench

ILINK-2-j

内容

1　はじめに

1.1　本ガイドの目的

ILINK は、大きなサイズの再配置可能なマルチモジュールの C/C++プログラム、C/C++プログラムとアセンブラプログラムの混合リンク、サイズの小さい単一ファイルの絶対アドレスを持つアセンブラプログラムのリンクなど、幅広く組込みアプリケーションの開発に適した強力で柔軟性のあるリンカです。IAR Embedded Workbench では、選択したデバイス毎に標準的なメモリ割り付けができるように、リンカ設定ファイルがプロジェクトに登録されますが、より複雑なメモリ割り付けを行う場合は、このリンカ設定ファイル(.icf ファイル)の編集を行う必要があります。本ガイドでは、組み込みシステムで用いられる典型的なメモリ構成毎に、ソースファイルの追加記述、リンカ設定ファイルの編集のしかた解説します。

1.2　本ガイドの対象者

このガイドは、IAR Embedded Workbench の基本機能およびオペレーションを理解しており、リンカにて自由にコードやデータを配置する方法を習得されたい方を対象にしております。 IAR Embedded Workbench を初めて導入される方には、IAR Embedded Workbench 導入セミナーや資料を準備しておりますので、弊社営業までご相談ください。

1.3　対象製品

- EWARM V6.30.1 以降
- EWRX V2.10 以降
- EWSTM8

2　表記について

2.1　リンカ設定ファイル例

```
01 define memory mem with size = 4G;
02 define region RROM = mem:[from 0x00000000 to 0x0003ffff];
03 define region RRAM = mem:[from 0x20000000 to 0x2000ffff];
```

左の数字は行番号、右はリンカ設定ファイルの内容を表します。

2.2　ソースファイル例

```
01 int a[4];
02 int b[ ] = { 0x12345678, 0x23456789, 0x34567890, 0x45678901 };
03 void sub(int i, int j)
04 {
05    a[i] = b[j];
06 }
```

左の数字は行番号、右はソースファイルの内容を表します。

2.3　名前の付け方

No	名前	説明	備考
1	RMYREGION	領域名 RMYREGION	
2	BMYBLOCK	ブロック名 RMYBLOCK	
3	SMYSECTION	ユーザ定義のセクション名 SMYSECTION	
4	_MYSECTION	ユーザ定義のセクション名 _MYSECTION の __pragma()マクロ名。 (例) #define _SMYSECTION ¥ _Pragma(“location=¥”SMYSECTION¥””)	

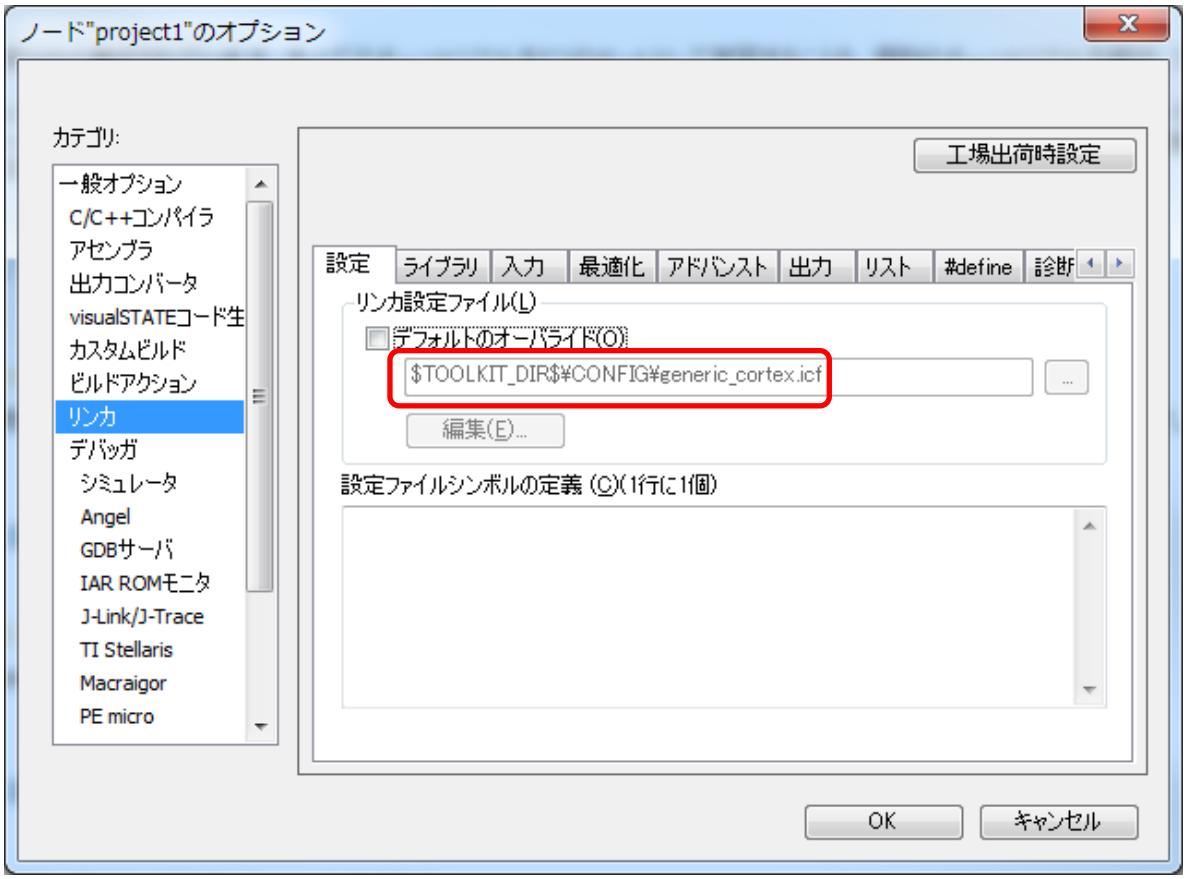
3　リンカオプションとリンカ設定ファイル

3.1　リンカ設定ファイルの選択

新規プロジェクトを作成したとき、各コア用もしくはデバイス用のデフォルトのリンカ設定ファイルが、自動的に選択されます。ARM Cortex コアを選択した場合、図のように generic_cotex.icf が選択されます。ここで **$TOOLKIT_DIR$** は、Embedded Workbench がインストールされているフォルダを意味しています。最新の IAR Embedded Workbench では、

プロジェクト > オプション > 一般オプション > ターゲット > デバイス

でデバイスを選択すると、そのデバイス用のリンカ設定ファイルが選択されます。



リンカ設定ファイルの設定内容を変更したい時は、この編集する必要がありますが、エディタで編集する場合は、別の場所にコピーして編集するか、編集結果をセーブする際に別の場所を指定してください。プロジェクトファイル（.ewp ファイル）と同じディレクトリに置く場合は、ファイルパスの指定を**$PROJ_DIR$¥generic_cotex.icf** のように変更します。ファイル名は任意で、**$PROJ_DIR$**は、プロジェクトファイルのパスを示します。

3.2　リンカ設定ファイルエディタ (EWARM のみ)

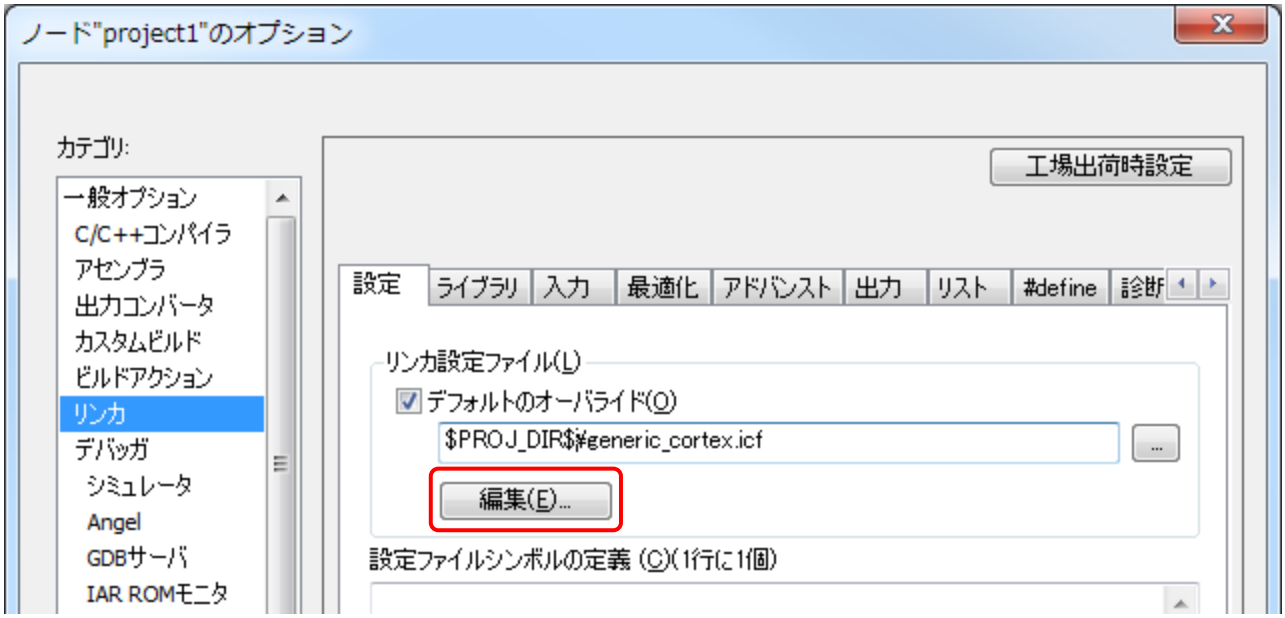
以下は、ARM Cortex-M シリーズ用リンカ設定ファイルのテンプレートの、最初の部分です。

```
01 /*###ICF### Section handled by ICF editor, don't touch! ****/
02 /*-Editor annotation file-*/
03 /* IcfEditorFile="$TOOLKIT_DIR$¥config¥ide¥IcfEditor¥cortex_v1_0.xml" */
04 /*-Specials-*/
05 define symbol __ICFEDIT_intvec_start__ = 0x00000000;
06 /*-Memory Regions-*/
07 define symbol __ICFEDIT_region_ROM_start__   = 0x00000000;
08 define symbol __ICFEDIT_region_ROM_end__     = 0x0007FFFF;
09 define symbol __ICFEDIT_region_RAM_start__   = 0x20000000;
10 define symbol __ICFEDIT_region_RAM_end__     = 0x2000FFFF;
11 /*-Sizes-*/
12 define symbol __ICFEDIT_size_cstack__   = 0x400;
13 define symbol __ICFEDIT_size_heap__     = 0x800;
14 /**** End of ICF editor section. ###ICF###*/
15
16
17 define memory mem with size = 4G;
18 define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to
19 __ICFEDIT_region_ROM_end__];
    :
   以下略
```

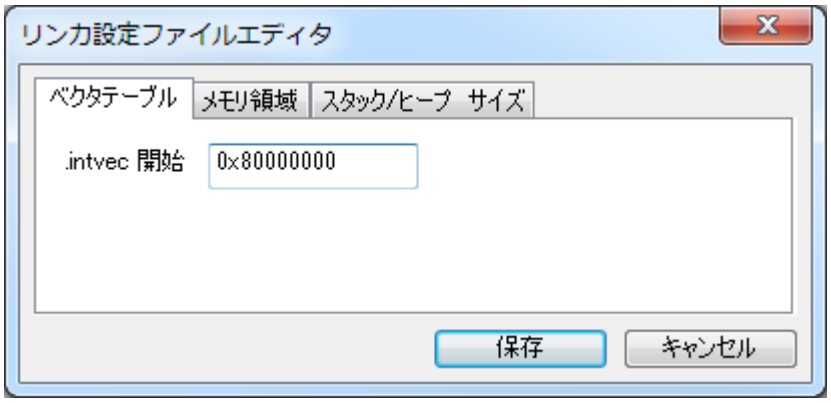
リンカ設定ファイルエディタを使用すると、

define symbol __ICFEDIT_xxxxxx__ = 0x00000000;

の行の、シンボルの値（赤字の部分）を簡単に修正することができます。リンカ設定ファイルエディタを起動するには、下の画面の編集ボタンをクリックします。



リンカ設定ファイルエディタが起動されると、Cortex-M シリーズのマイコンの場合、次の画面が表示されます。テキストボックス内の数値を以下のように変更すると、




```
05 define symbol __ICFEDIT_intvec_start__ = 0x00000000;
```

が

```
05 define symbol __ICFEDIT_intvec_start__ = 0x80000000;
```

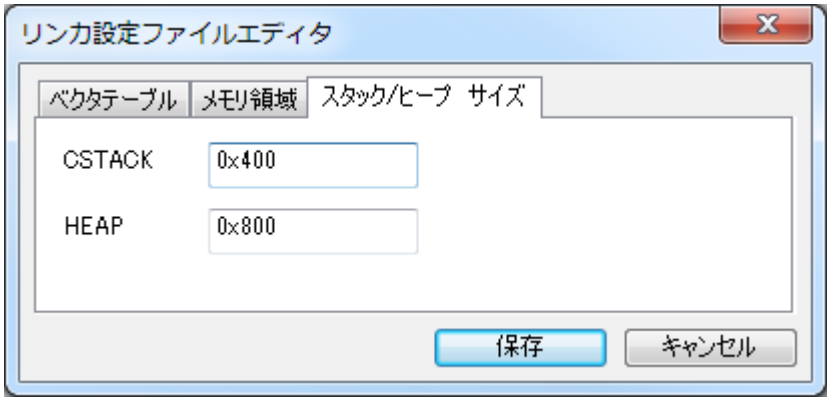
に変更されます。

同様に、メモリ領域タブは、


```
07 define symbol __ICFEDIT_region_ROM_start__   = 0x00000000;
08 define symbol __ICFEDIT_region_ROM_end__     = 0x0007FFFF;
09 define symbol __ICFEDIT_region_RAM_start__   = 0x20000000;
10 define symbol __ICFEDIT_region_RAM_end__     = 0x2000FFFF;
```

に対応し、

スタック／ヒープサイズタブは、




```
12 define symbol __ICFEDIT_size_cstack__   = 0x400;
13 define symbol __ICFEDIT_size_heap__     = 0x800;
```

に対応します。

変更が終わったら、保存ボタンをクリックしますが、デフォルトのリンカ設定ファイルを直接編集している場合は、ウィンドウズの保存のダイヤログが表示されますので、必要な場所に保存してください。

リンカ設定ファイルの以下の部分以外は、エディタで変更することができます。

```
01 /*###ICF### Section handled by ICF editor, don't touch! ****/
02 /*-Editor annotation file-*/

      :

13 define symbol __ICFEDIT_size_heap__     = 0x800;
14 /**** End of ICF editor section. ###ICF###*/
```

リンカ設定ファイルエディタを使用しない場合は、この部分を削除して構いません。また本ガイドでも、以降使用しません。

4　基礎編

4.1　セクションについて

関数や、データは全てセクションに所属しますが、デフォルトの所属セクションで基礎編で使用するものは下記の通りです。

No	略号	説明	備考
1	.bss	ゼロに初期化される静的/グローバル変数を保持	
2	.data	初期化される静的/ グローバル変数を保持	
3	.data_init	リンカディレクティブ initialize by copy の使用時に、.data セクションの初期値を保持	
4	.intvec	割り込みベクタテーブルを保持	
6	.noinit	__no_init を付加した、静的変数およびグローバル変数を保持します	
5	.rodata	定数変数、文字列、集合リテラルなどの、定数データを保持	
6	.text	システム初期化用のコードを除くプログラムコードを保持	
7	.textrw	__ramfunc により宣言されたプログラムコードを保持	
8	.textrw_init	.textrw で宣言されたセクションのイニシャライザを保持	
9	readwrite	• 読み書き可能セクションの総称 • セクション名またはブロック名の前にある場合は、そのセクションまたはブロックが読み書き可能であることを表す。 [例] readwrite section S_RW	
10	readonly	• 読み出し専用セクションの総称 • セクション名またはブロック名の前にある場合は、そのセクションまたはブロックが読み出し専用であることを表す。 [例] readonly section S_RO	
11	code	コードセクションの総称	
12	data	データセクションの総称	
13	rw	readwrite の省略表示	
14	ro	readonly の省略表示	

4.2　リンカディレクティブのステートメント例

セクションをメモリに配置するために、リンカ設定ファイルに必要なディレクティブを記述する必要がありますが、基礎編で使用するものは下記の通りです。

- define memory

mem という名前で 4G バイトのメモリ空間（memory）を定義

```
define memory mem with size = 4G;
```

- define region

mem という名前のメモリ空間に、ROM と言う名前の 0～0x3ffff 番地の範囲の領域（region）を定義

```
define region ROM = mem:[from 0x00000000 to 0x0003ffff];
```

- define block

CSTACK と言う名前の、アラインメント 8、大きさ 256 バイトのブロックを定義

```
define block  CSTACK with alignment = 8, size = 0x100{};
```

- initialize by copy

rw セクションをイニシャライザおよび初期化データ用のセクションに分割し、アプリケーション起動時に自動的に初期化する。

```
initialize by copy { rw };
```

- do not initialize

.noinit セクションは初期化しない

```
do not initialize  { section .noinit };
```

- place at address

読み出し専用（ro)である.intvec セクションをメモリ空間 mem の 0 番地に配置

```
place at address mem:0 { ro section .intvec };
```

- place in

ROM 領域に読み出し専用のセクションを配置

```
place in ROM { ro };
```

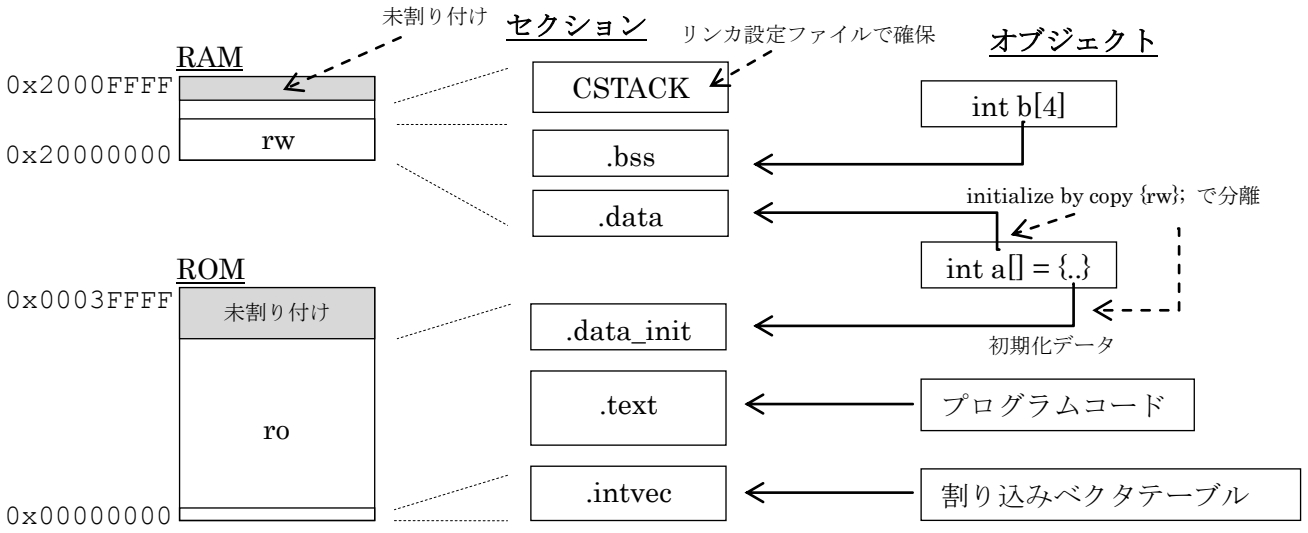
4.3 最も基本的な設定

- ソースプログラム例

```
01 int a[ ] = { 0x12345678, 0x23456789, 0x34567890, 0x45678901 };
02 int b[4];
03
04 void main(void)
05 {
06   for(int i = 0; i < 3; i++) b[i] = a[i];
07 }
```

行	説明	備考
01	配列 b は.data セクションに配置される。リンカ設定ファイルで、 initialize by copy (rw); が指定された場合 (次ページ参照)、初期化データを格納するセクション .data_init が分離される。	
02	配列 a は.bss セクションに配置される。	
	プログラムコードは .text セクションに配置される	

- メモリマップ



- リンカ設定ファイルの記述例

```
01 define memory mem with size = 4G;
02 define region ROM = mem:[from 0x00000000 to 0x0003ffff];
03 define region RAM = mem:[from 0x20000000 to 0x2000ffff];
04 define block  CSTACK with alignment = 8, size = 0x100{};
05
06 initialize by copy { rw };
07
08 place at address mem:0 { ro section .intvec };
09 place in ROM { ro };
```

行	説明	備考
01	mem という名の 4GB のメモリ空間を確保。	ARM/RX は 4GB stm8 は 16MB
02	mem 内に ROM と言う名の領域を 0x00000000～0x0003ffff に確保。	
03	mem 内に RAM と言う名の領域を 0x20000000～0x2000ffff に確保。	
04	CSTACK と言う名前で、アラインメント 8、大きさ 256 バイトのブロックを定義。	
06	rw セクションを、実体(.data)と初期化用のセクション(.data_init)に分割し、アプリケーションの起動時に自動的に初期化することを指示。	
08	ro である.intvec セクションを mem 空間の 0 番地に配置。	
09	ROM 領域に読み出し専用の領域を配置。	.text, .data_init
10	RAM 領域に読み書き可能の領域と、CSTACK ブロックを配置。	

4.4　初期化を行わない領域と const 付変数

- ソースプログラム例

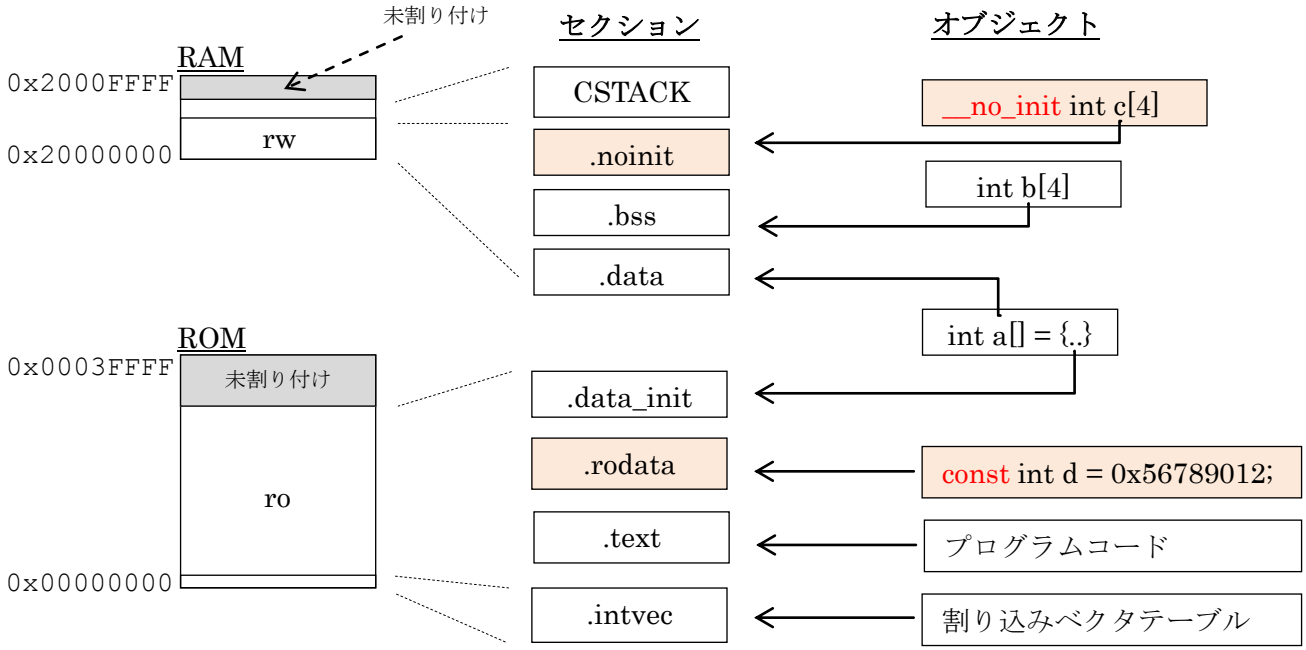
```
01  int a[ ] = { 0x12345678, 0x23456789, 0x34567890, 0x45678901 };
02  int b[4];
03  __no_init int c[4];
04  const int d = 0x56789012;

      : 以下略
```

行	説明	備考
03	配列 c は、.no_init セクションに配置される。	
04	変数 d は、.rodata セクションに配置される。	

- メモリマップ

固定データ用 .rodata セクションと、初期化を行わない.noinit セクションを追加



- リンカ設定ファイル例

```
01 define memory mem with size = 4G;
02 define region ROM = mem:[from 0x00000000 to 0x0003ffff];
03 define region RAM = mem:[from 0x20000000 to 0x2000ffff];
04 define block  CSTACK with alignment = 8, size = 0x100{};
05
06 initialize by copy { rw };
07 do not initialize  { section .noinit };
08
09 place at address mem:0 { ro section .intvec };
10 place in ROM { ro };
11 place in RAM { rw, block CSTACK };
```

行	説明	備考
07	.noinit セクションは初期化しないことを指定	

4.5　RAM 上で実行する関数

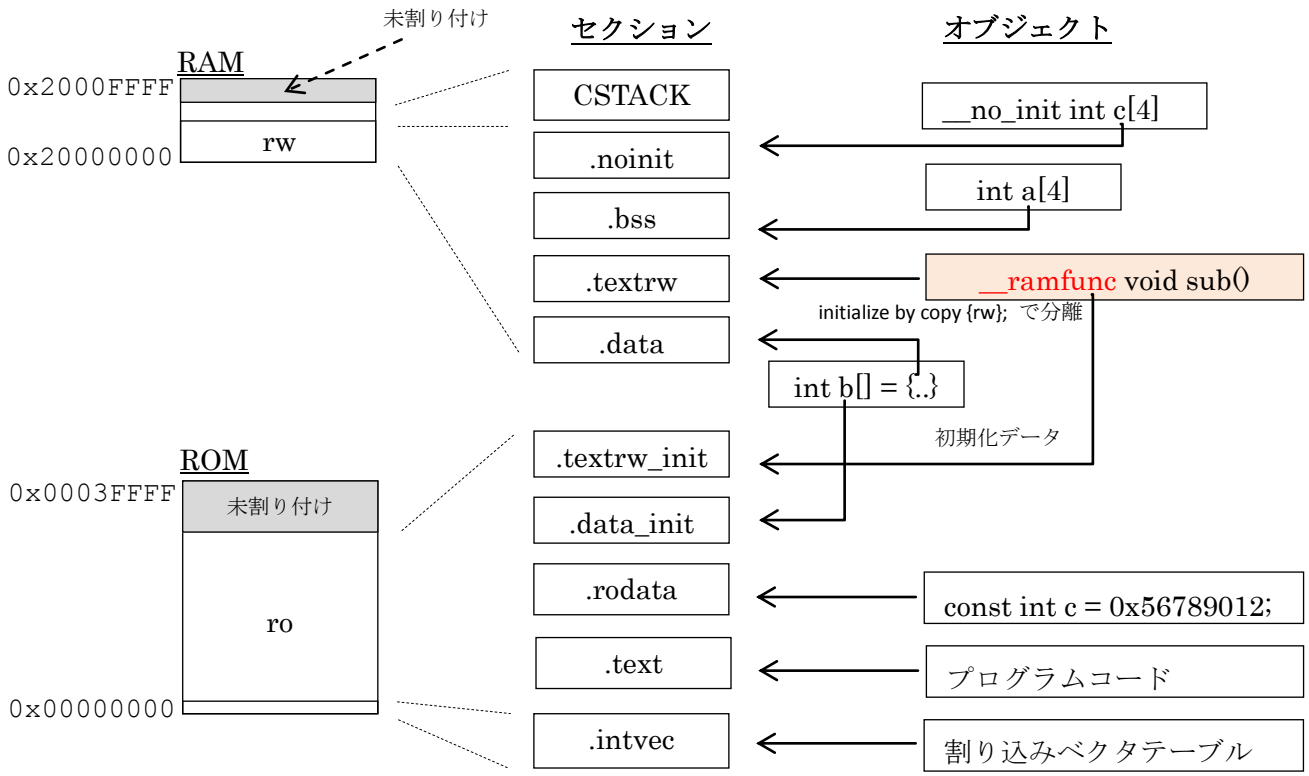
- ソースプログラム例

関数に__ramfunc を付加すると、コードが .text ではなく .textrw セクションに割り付けられます。ILINK は、.textrw を初期化付変数と同じに扱うので、リンカ設定ファイルには、RAM 上で実行する関数のための特別な記述は必要ありません。

```
01  int a[ ] = { 0x12345678, 0x23456789, 0x34567890, 0x45678901 };
02  int b[4];
03  __no_init int c[4];
04  #define d 0x56789012
05
06  __ramfunc void sub(int *i, int *j)
07  {
08    *i++ = *j++ + d;
09    *i++ = *j++ + d;
10    *i++ = *j++ + d;
11    *i   = *j   + d;
12  }
13
14  void main(void)
15  {
16    sub(&b, &a);
17    sub(&c, &b);
18  }
```

行	説明	備考
06	関数 sub は .textrw セクションに配置される。	

- メモリマップ



- リンカ設定ファイル例

```
01 define memory mem with size = 4G;
02 define region ROM = mem:[from 0x00000000 to 0x0003ffff];
03 define region RAM = mem:[from 0x20000000 to 0x2000ffff];
04 define block  CSTACK with alignment = 8, size = 0x100{};
05
06 initialize by copy { rw };
07 do not initialize  { section .noinit };
08
09 place at address mem:0 { ro section .intvec };
10 place in ROM { ro };
11 place in RAM { rw, block CSTACK };
```

行	説明	備考
06	配列 b は、.data セクションに配置される。関数 sub は、.textrw セクションに配置される。リンカ設定ファイルで、 initialize by copy (rw); が指定された場合、初期化データを格納するセクション .data_init と.textrw_init セクションが分離される。 この場合、.data と .textrw は rw、.data_init と .textrw_init は ro 属性になる。	

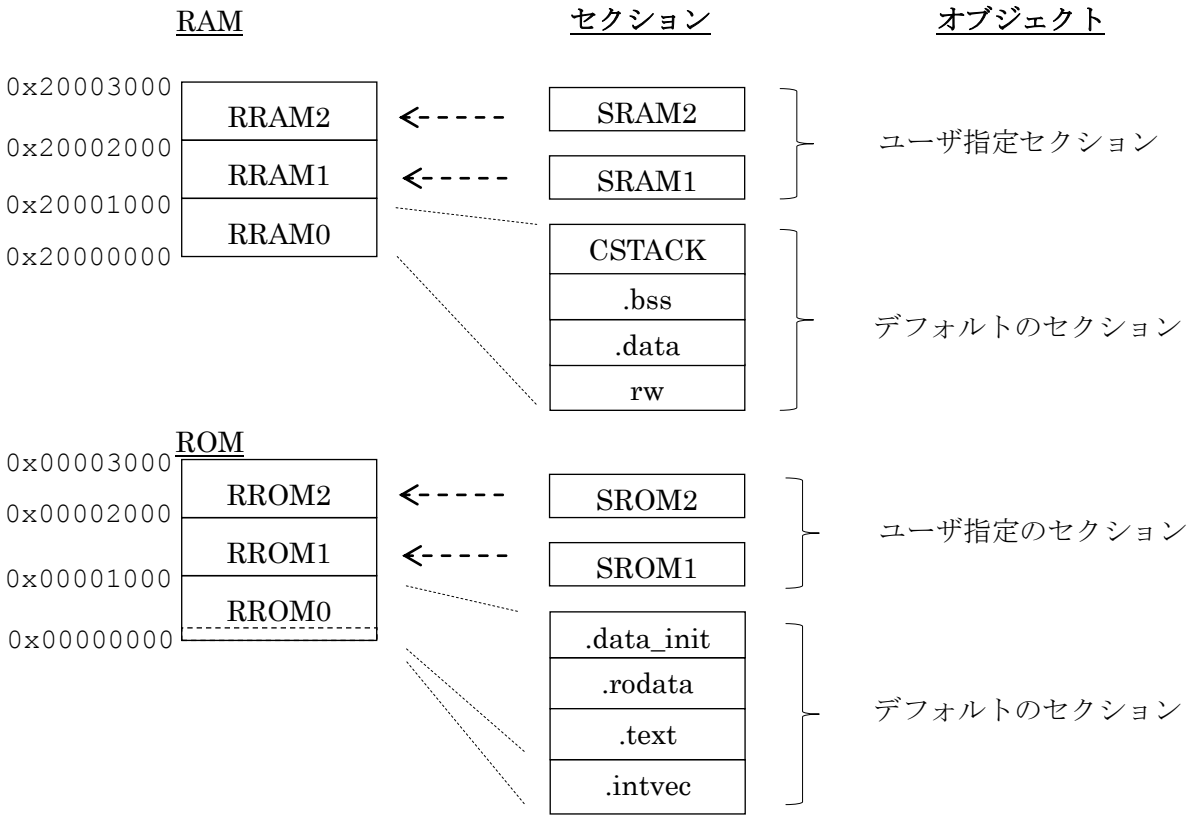
4.6　関数/データのセクション指定とメモリへの配置

関数やデータをデフォルトの所属セクション以外に割り付けたい場合は、#pragma または@演算子を使用して下記のように記述します。

- ソースプログラム例

```
01 // セグメントへの配置
02 #define _SRAM1 _Pragma("location=¥"SRAM1¥"")
03 #define _SROM1 _Pragma("location=¥"SROM1¥"")
04 // メモリ
05 #pragma location="SRAM1"
06        int a11,
07            a12[16];                    // 記法 1-1
08
09 _SRAM1 int a13;                        // 記法 1-2
10 _SRAM1 int a14[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
11
12        int a23     @"SRAM2";           // 記法 2
13        int a24[16] @"SRAM2"
14                    = {0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14,15 };
15 // 関数
16 #pragma location="SROM1"
17        int f11(void)          {return a11;} // 記法 1-1
18 _SROM1 int f12(void)          {return a13;} // 記法 1-2
19        int f2 (void)@"SROM2" {return a23;} // 記法 2
20 void main()
21 {
22   f11(); f12();
23   f2();
24   a12[0] = a14[0];
25   a14[0] = a24[0];
26   return;
27 }
```

行	説明	備考
02	この行以降_SRAM1 は、 #pragma location="SRAM1" と展開される。	
03	この行以降_SROM1 は、 #pragma location="SROM1" と展開される。	
05	データ a11,a12[16] の所属セクションは SRAM1	
09	データ a13 の所属セクションは SRAM1	
10	データ a14[] の所属セクションは SRAM1	
12	データ a23 の所属セクションは SRAM2	
13	データ a24[] の所属セクションは SRAM2	
16	関数 f11() の所属セクションは SROM1	
18	関数 f12() の所属セクションは SROM1	
19	関数 f2() の所属セクションは SROM2	


RAM
セクション
オブジェクト
0x20003000
RRAM2
0x20002000
RRAM1
0x20001000
RRAM0
0x20000000
SRAM2
SRAM1
ユーザ指定セクション
CSTACK
.bss
.data
rw
デフォルトのセクション
ROM
0x00003000
RROM2
0x00002000
RROM1
0x00001000
RROM0
0x00000000
SROM2
SROM1
ユーザ指定のセクション
.data_init
.rodata
.text
.intvec
デフォルトのセクション

- リンカ設定ファイル例

```
01 define memory mem with size = 4G;
02 define region RROM0 = mem:[from 0x00000000 to 0x0000ffff];
03 define region RROM1 = mem:[from 0x00010000 to 0x0001ffff];
04 define region RROM2 = mem:[from 0x00020000 to 0x0002ffff];
05 define region RRAM0 = mem:[from 0x20001000 to 0x20000fff];
06 define region RRAM1 = mem:[from 0x20002000 to 0x20001fff];
07 define region RRAM2 = mem:[from 0x20003000 to 0x20002fff];
08
09 define block  CSTACK with alignment = 8, size = 0x400{};
10
11 initialize by copy { rw };
12 do not initialize  { section .noinit };
13
14 "V0":place at address mem:0 { ro section .intvec };
15
16 "ROM0":place in RROM0 { ro };
17 "ROM1":place in RROM1 { section SROM1 };
18 "ROM2":place in RROM2 { section SROM2, section SRAM2_init };
19
20 "RAM0":place in RRAM0 { rw, block CSTACK };
21 "RAM1":place in RRAM1 { section SRAM1 };
22 "RAM2":place in RRAM2 { section SRAM2 };
```

行	説明	備考
14	”V0”、”RAMx” 、”ROMx” は map ファイルの対応箇所に表示されます。 16 [例] 17 "V0": 0x40 18 .intvec ro code 0x00000000 0x40 vector_table_M.o - 0x00000040 0x40 20 21 "ROM0": 0xf8 22 .text ro code 0x00000040 0x34 main_05.o [1] .text ro code 0x00000074 0x2a zero_init3.o [4] :	
17	書込み専用セクション SROM1 を領域 RROM1 に配置。	
18	書込み専用セクション SROM2 と、SRAM2 の初期化データを領域 RROM2 に配置。	
21	読み書き可専用セクション SRAM1 を領域 RRAM1 に配置。	
22	読み書き可専用セクション SRAM2 を領域 RRAM2 に配置。	

4.7　ファイル単位での配置

- ソースプログラム例

main.c

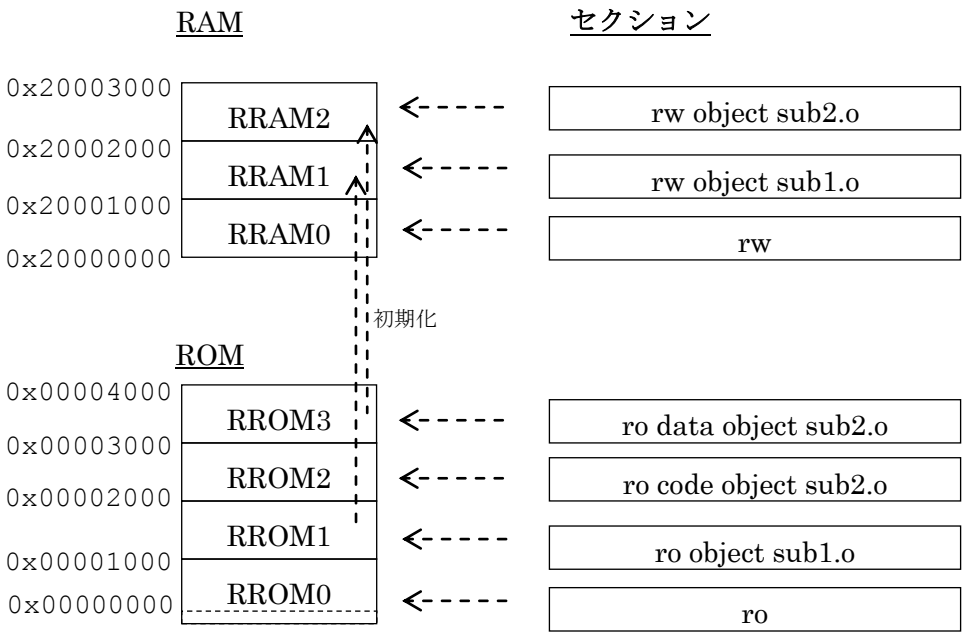
```
01 // メモリ
02 extern int a11, a12[];            // SRAM1
03 extern int a13, a14[];
04 extern int a23, a24[];            // SRAM2
05
06 // 関数
07 extern int f11(void), f11(void) ; // SROM1
08 extern int f2(void),  f12(void) ; // SROM2
09
10 void main()
11 {
12   f11(); f12();
13   f2();
14   a12[0] = a14[0];
15   a14[0] = a24[0];
16   return;
17 }
```

sub1.c

```
01 // データ
02 int a11, a12[16];
03 int a13, a14[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
04
05 // 関数
06 int f11(void){return a11;}
07 int f12(void){return a13;}
```

sub2.c

```
01 // データ
02 int a23;
03 int a24[16] = {0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14,15 };
04
05 // 関数
06 int f2 (void) {return a23;}
```


RAM
セクション
0x20003000
RRAM2
rw object sub2.o
0x20002000
RRAM1
rw object sub1.o
0x20001000
RRAM0
rw
0x20000000
初期化
ROM
0x00004000
RROM3
ro data object sub2.o
0x00003000
RROM2
ro code object sub2.o
0x00002000
RROM1
ro object sub1.o
0x00001000
RROM0
ro
0x00000000

- リンカ設定ファイル例

```
01 define memory mem with size = 4G;
02 define region RROM0 = mem:[from 0x00000000 to 0x0000ffff];
03 define region RROM1 = mem:[from 0x00010000 to 0x0001ffff];
04 define region RROM2 = mem:[from 0x00020000 to 0x0002ffff];
05 define region RROM3 = mem:[from 0x00030000 to 0x0003ffff];
06 define region RRAM0 = mem:[from 0x20001000 to 0x20000fff];
07 define region RRAM1 = mem:[from 0x20002000 to 0x20001fff];
08 define region RRAM2 = mem:[from 0x20003000 to 0x20002fff];
09
10 define block  CSTACK with alignment = 8, size = 0x400{};
11
12 initialize by copy { rw, ro data object sub2.o };
13 do not initialize  { section .noinit };
14
15 "V0":place at address mem:0 { ro section .intvec };
16
17 "ROM0":place in RROM0 { ro };
18 "ROM1":place in RROM1 { ro object sub1.o };
19 "ROM2":place in RROM2 { ro code object sub2.o };
20 "ROM3":place in RROM3 { ro data object sub2.o };
21
22 "RAM0":place in RRAM0 { rw, block CSTACK };
22 "RAM1":place in RRAM1 { rw object sub1.o };
23 "RAM2":place in RRAM2 { rw object sub2.o };
```

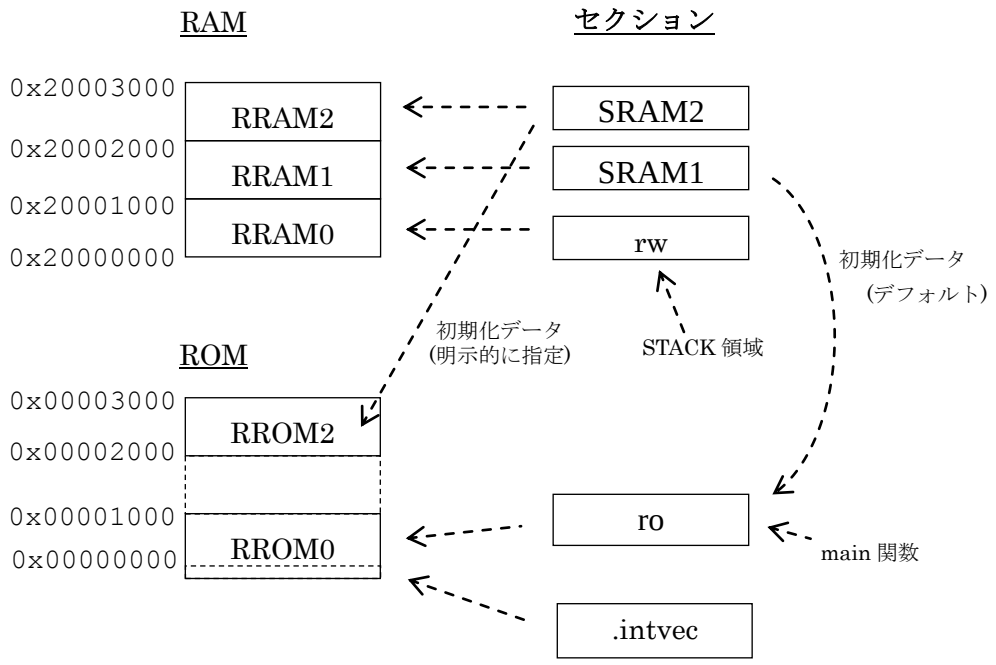
行	説明	備考
18	ファイル sub1.o (sub1.c をコンパイルしたもの、以下同じ)で、読み出し専用セクションを領域 RROM1 に配置	
19	ファイル sub2.o 読み出し専用の code セクションを領域 RROM2 に配置	
20	ファイル sub2.o 読み出し専用の data セクションを領域 RROM2 に配置	データの初期化指定子
22	ファイル sub1.o の、rw セクションを領域 RRAM1 に配置	
23	ファイル sub2.o の、rw セクションを領域 RRAM2 に配置	

4.8　RAM 上で実行する関数 (__ramfunc を使用しないバージョン)

- ソースプログラム例

```
01 #pragma location="SRAM1"
02 int add(int i, int j)
03 {
04   return i + j;
05 }
06
07 int sub(int i, int j) @"SRAM2"
08 {
09   return i - j;
10 }
11
12 int main(void)
13 {
14   return add(1, 2) + sub(3, 4);
15 }
```

行	説明	備考
01	関数 add を SRAM1 セクションに配置	
07	関数 sub を SRAM2 セクションに配置	



- リンカ設定ファイル例

```
01 define memory mem with size = 4G;
02 define region RROM0 = mem:[from 0x00000000 to 0x0000ffff];
03 define region RROM2 = mem:[from 0x00020000 to 0x0002ffff];
04
05 define region RRAM0 = mem:[from 0x20000000 to 0x20000fff];
06 define region RRAM1 = mem:[from 0x20001000 to 0x20001fff];
07 define region RRAM2 = mem:[from 0x20002000 to 0x20002fff];
08
09 define block  CSTACK with alignment = 8, size = 0x400{};
10
11 initialize by copy { rw, ro section SRAM2 };
12 do not initialize  { section .noinit };
13
14 "V0":place at address mem:0 { ro section .intvec };
15
16 "ROM0":place in RROM0 { ro };
17 "ROM2":place in RROM2 { section SRAM2_init };
18
19 "RAM0":place in RRAM0 { rw, block CSTACK };
20 "RAM1":place in RRAM1 { section SRAM1 };
21 "RAM2":place in RRAM2 { section SRAM2 };
```

行	説明	備考
17	セクション SRAM2 の初期化データ(SRAM2_init)を領域 RROM2 に配置。(SRAM1 の初期化データは、SRAM1_init での配置指定がないため領域 RROM0 に配置される)	

5　チップ集

5.1　セクション選択について

セクション選択が重複する場合は、より狭い選択が優先されます。place ディレクティブが次のように指定されてる場合、SROM1 セクションは領域 RROM1 に配置されますが、他の ro セクションは領域 RROM0 に配置されます。詳細は、マニュアル「IAR C/C++ コンパイラ コンパイルおよびリンク」の、セクションセレクタの項をご参照ください。

```
place in RROM0 { ro };
place in RROM1 { section SROM1 };
```

5.2　セクション配置順序の指定

セクションを固定順序で配置したい場合は、define block ディレクティブで固定順序でセグメントを並べたブロックを定義してから配置しますます。詳細は、マニュアル「IAR C/C++ コンパイラ コンパイルおよびリンク」のリンカ設定ファイルの章の、define block ディレクティブの項をご参照ください。

[例]

```
define block BLOCKNAME with fixed order (segment1, segment2, ...)
place in RROM0 { block BLOCKNAME };
```

5.3　外部メモリの使用

領域を物理メモリの領域に合わせて定義し、セグメントやブロックを配置しますが、一般に外部メモリは、メモリの種類を問わず、メモリコントローラ側の初期設定が必要です。初期設定プログラムは、内部フラッシュメモリに配置・起動し、メモリコントローラを初期化してから、外部メモリの使用を開始することが基本になります。ただし、C-SPY デバッガのマクロを使用して、この役割を代替させ、外部 SDRAM のみを使用してデバッグすることなどが可能になります。詳細は、マニュアル「IAR Embedded Workbench® C-SPY® デバッガガイド」の C-SPY マクロの章をご参照ください。

5.4　バイナリファイルの取り込み

バイナリファイルの取り込みは、リンカオプションで行います。最後の数字はアラインメントです。

[例]

--image_input=filename1.bin,symbol1,SECTION1,8

--image_input=filename2.bin,symbol2,SECTION2,4

バイナリファイルが 1 つの場合は、

プロジェクトメニュー > オプション > リンカ > 入力タブ

ローバイナリイメージのところに同じ情報を書き込むことで、同じオプションを指定したことになります。

バイナリファイルの配置は、place コマンドで行うことができます。

place at address 0x30000000 {section SECTION1};

また、symbol1/2 はプログラムから参照することができます。

5.5　セクションアドレスやサイズの取り出し

外部/静的変数の初期化をマニュアルで行いたい場合や、オーバレイなどでセクション全体のアドレスやサイズをプログラムから知りたい場合は、下記のようなセクション演算子を使用します。

void * __section_begin(char const * section) // セクションの開始アドレス

void * __section_end(char const * section) // セクションの終了アドレス

size_t * __section_size(char const * section) // セクションのサイズ

詳細は、マニュアル「IAR C/C++ コンパイラ コンパイルおよびリンク」のセクション演算子の章をご参照ください。