お客様各位

資料中の「ラピスセミコンダクタ」等名称の ラピステクノロジー株式会社への変更

2020 年 10 月 1 日をもって、ラピスセミコンダクタ株式会社の LSI 事業部門は、ラピステクノロジー株式会社に分割承継されました。従いまして、本資料中にあります「ラピスセミコンダクタ株式会社」、「ラピスセミ」、「ラピス」といった表記に関しましては、全て「ラピステクノロジー株式会社」に読み替えて適用するものとさせていただきます。なお、会社名、会社商標、ロゴ等以外の製品に関する内容については、変更はありません。以上、ご理解の程よろしくお願いいたします。

2020年10月1日
ラピステクノロジー株式会社

Dear customer

LAPIS Semiconductor Co., Ltd. ("LAPIS Semiconductor"), on the 1st day of October, 2020, implemented the incorporation-type company split (shinsetsu-bunkatsu) in which LAPIS established a new company, LAPIS Technology Co., Ltd. (“LAPIS Technology”) and LAPIS Technology succeeded LAPIS Semiconductor’s LSI business.

Therefore, all references to "LAPIS Semiconductor Co., Ltd.", "LAPIS Semiconductor" and/or "LAPIS" in this document shall be replaced with "LAPIS Technology Co., Ltd."

Furthermore, there are no changes to the documents relating to our products other than the company name, the company trademark, logo, etc.

Thank you for your understanding.

LAPIS Technology Co., Ltd.
October 1, 2020

ROHM GROUP
LAPIS
SEMICONDUCTOR

FJXT_MCU_HOWTO_ALLOC-01

関数･変数等の特定領域への配置方法

初版　発行日 2019 年 7 月 22 日

ROHM
SEMICONDUCTOR

ご注意

1) 本資料の記載内容は改良などのため予告なく変更することがあります。

2) ラピスセミコンダクタは常に品質・信頼性の向上に取り組んでおりますが，半導体製品は種々の要因で故障・誤作動する可能性があります。
万が一，本製品が故障・誤作動した場合であっても，その影響により人身事故，火災損害等が起こらないようご使用機器でのディレーティング，冗長設計，延焼防止，バックアップ，フェイルセーフ等の安全確保をお願いします。定格を超えたご使用や使用上の注意書が守られていない場合，いかなる責任もラピスセミコンダクタは負うものではありません。

3) 本資料に記載されております応用回路例やその定数などの情報につきましては，本製品の標準的な動作や使い方を説明するものです。したがいまして，量産設計をされる場合には，外部諸条件を考慮していただきますようお願いいたします。

4) 本資料に記載されております技術情報は，本製品の代表的動作および応用回路例などを示したものであり，それをもって，当該技術情報に関するラピスセミコンダクタまたは第三者の知的財産権その他の権利を許諾するものではありません。したがいまして，上記技術情報の使用に起因して第三者の権利にかかわる紛争が発生した場合，ラピスセミコンダクタはその責任を負うものではありません。

5) 本製品は，一般的な電子機器（AV機器，OA機器，通信機器，家電製品，アミューズメント機器など）および本資料に明示した用途への使用を意図しています。

6) 本資料に掲載されております製品は，耐放射線設計はなされておりません。

7) 本製品を下記のような特に高い信頼性が要求される機器等に使用される際には，ラピスセミコンダクタへ必ずご連絡の上，承諾を得てください。
・ 輸送機器（車載，船舶，鉄道など），幹線用通信機器，交通信号機器，防災・防犯装置，安全確保のための装置，医療機器，サーバー，太陽電池，送電システム

8) 本製品を極めて高い信頼性を要求される下記のような機器等には，使用しないでください。
・ 航空宇宙機器，原子力制御機器，海底中継機器

9) 本資料の記載に従わないために生じたいかなる事故，損害もラピスセミコンダクタはその責任を負うものではありません。

10) 本資料に記載されております情報は，正確を期すため慎重に作成したものですが，万が一，当該情報の誤り・誤植に起因する損害がお客様に生じた場合においても，ラピスセミコンダクタはその責任を負うものではありません。

11) 本製品のご使用に際しては，RoHS 指令など適用される環境関連法令を遵守の上ご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して，ラピスセミコンダクタは一切の責任を負いません。本製品の RoHS 適合性などの詳細につきましては，セールス・オフィスまでお問合せください。

12) 本製品および本資料に記載の技術を輸出又は国外へ提供する際には，「外国為替及び外国貿易法」，
「米国輸出管理規則」など適用される輸出関連法令を遵守し，それらの定めにしたがって必要な手続を行ってください。

13) 本資料の一部または全部をラピスセミコンダクタの許可なく，転載・複写することを堅くお断りします。

ラピスセミコンダクタ株式会社
〒222-8575 神奈川県横浜市港北区新横浜 2-4-8
http://www.lapis-semi.com

目次

1. はじめに

本書は，関数・変数等を特定領域に配置する方法について記載しています。
以下のようなことを実現したいときに本書をご参照ください。

実現したいこと	参照箇所
関数や変数を特定の領域に配置したい	「2.1. 関数や変数の特定領域への配置」を参照してください。
スタック領域を特定の領域に配置したい	「2.2. スタック領域の特定領域への配置」を参照してください。
関数や変数をソースファイルに記述した順番に配置したい	「2.3. 関数や変数を記述順に配置」を参照してください。
関数名や変数名を変えても，それぞれの配置が変わらないようにしたい	「2.3. 関数や変数を記述順に配置」を参照してください。

1.1. 関連するマニュアル

本書に関連するマニュアル類を以下に示します。併せて参照してください。

マニュアル名称	説明	備考
MACU8 アセンブラパッケージ ユーザーズマニュアル	アセンブラ，リンカ，オブジェクトコンバータ，ライブラリアンの使用方法を記載したマニュアルです。 アセンブリファイルの疑似命令やマップファイルの見方などを確認したい場合に参照ください。	Windows のスタートメニュー [U8 Tools] > [V2_xx_xx] > [ドキュメント] から参照いただけます。*1
CCU8 ユーザーズマニュアル	CCU8 C コンパイラの使用方法を記載したマニュアルです。 プラグマや組み込み関数の記述の仕方などを確認したい場合に参照ください。	
LEXIDE-U16 ユーザーズマニュアル	統合開発環境 LEXIDE-U16 の使用方法を記載したマニュアルです。 LEXIDE-U16 のオプションとコンパイラなど各ツールのオプションの対応を確認したい場合に参照ください。	Windows のスタートメニュー [U8 Tools] > [nX-U8 ドキュメント] から参照いただけます。

*1： U8/U16 Development Tools Release 2.0.0 以降を対象としています。

2. 特定領域への配置方法

2.1. 関数や変数の特定領域への配置

関数や変数を特定領域に配置するには，プラグマを使用します。
使用するプラグマを以下に示します。

特定の領域に割り付ける対象	使用するプラグマ
関数（割込み関数を除く）	SEGCODE プラグマ
割込み関数	SEGINTR プラグマ
初期化付き変数	SEGINIT プラグマまたは ABSOLUTE プラグマ
初期化なし変数	SEGNOINIT プラグマまたは ABSOLUTE プラグマ
const 変数	SEGCONST プラグマまたは ABSOLUTE プラグマ

上記のプラグマの詳細については，『CCU8 ユーザーズマニュアル』の「5.4 ABSOLUTE プラグマ」および「5.16 SEGMENT プラグマ」を参照してください。

2.1.1. 関数（割込み関数を除く）の特定領域への配置

関数を特定の領域に配置するには，SEGCODE プラグマを使用します。
方法としては，以下の 2 種類があります。

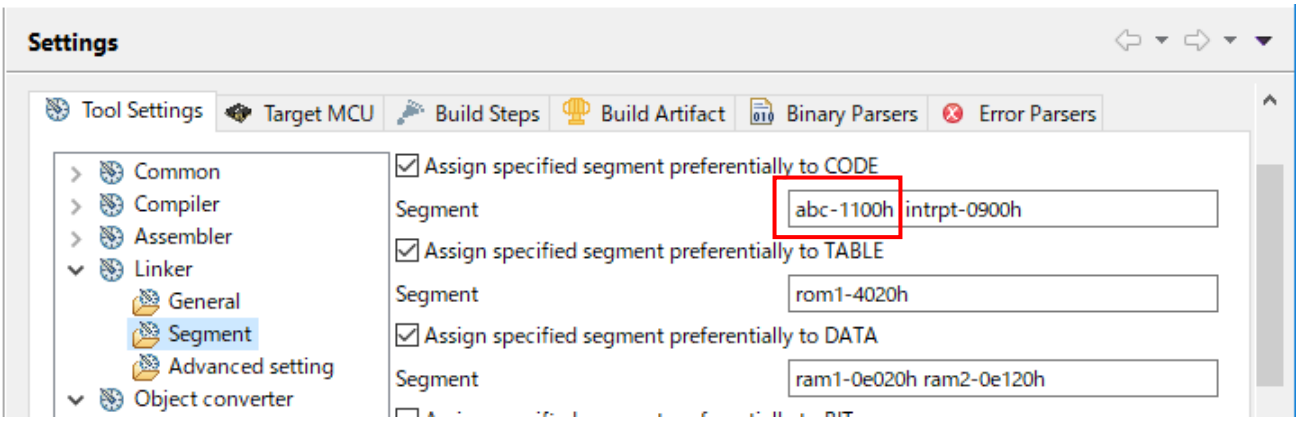
- 直接アドレスを指定する（記述例 1）
- 論理セグメントに配置してリンカでアドレスを指定する（記述例 2）

以下に記述例を示します。

【記述例】

```
#pragma SEGCODE 0x1000          /* 記述例 1 */
void func1() {                  /* 関数 funcf1 は 0x1000 番地に配置されます */
    …
}
#pragma SEGCODE "abc"           /* 記述例 2 */
void func2() {                  /* 関数 funcf2 は論理セグメント abc に配置されます */
   …                            /* 配置アドレスは，リンカのオプションで指定できます */
}
#pragma SEGCODE                 /* SEGCODE プラグマの終了 */
```

上記の記述例 2 で指定したセグメントの配置アドレスを指定する場合，LEXIDE-U16 の Linker の設定（[Tool Settings] > [Linker] > [Segment]）で，[Assign specified segment preferentially to CODE]の[Segment]フィールドに指定します。
以下の例では，セグメント abc を 0x1100 番地に配置するよう指定しています。



実際に配置されたアドレスを確認するには，「2.5. 配置結果の確認方法」を参照してください。

2.1.2. 割込み関数の特定領域への配置

割込み関数を特定の領域に配置するには，SEGINTR プラグマを使用します。
方法としては，以下の 2 種類があります。

- 直接アドレスを指定する（記述例 1）
- 論理セグメントに配置してリンカでアドレスを指定する（記述例 2）

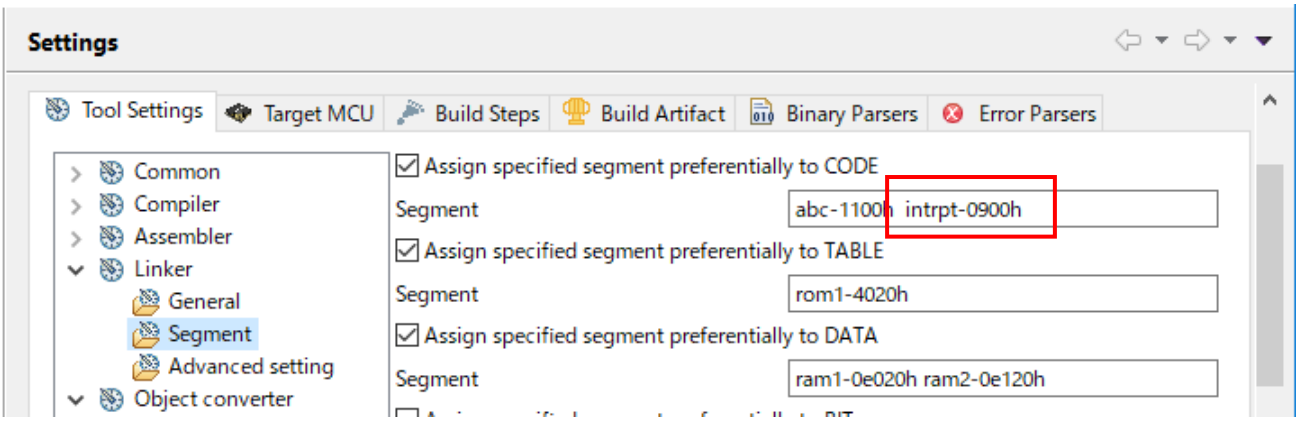
以下に記述例を示します。

【記述例】

```
static void intr_func1(void);
static void intr_func2(void);
#pragma interrupt intr_func1 0x10
#pragma interrupt intr_func2 0x20

#pragma SEGINTR 0x0800          /*記述例 1 */
void intr_func1 () {            /* 関数 intr_funcf1 は 0x0800 番地に配置されます*/
    …
}
#pragma SEGINTR ”intrpt”        /*記述例 2 */
void intr_func2 () {            /* 関数 intr_funcf2 は論理セグメント intrpt に配置されます*/
    …                           /* 配置アドレスは，リンカのオプションで指定できます   */
}
#pragma SEGINTR                 /* SEGINTR プラグマの終了 */
```

上記の記述例2で指定したセグメントの配置アドレスを指定する場合，LEXIDE-U16のLinkerの設定（[Tool Settings] > [Linker] > [Segment]）で，[Assign specified segment preferentially to CODE]の[Segment]フィールドに指定します。
以下の例では，セグメント intrpt を 0x0900 番地に配置するよう指定しています。



実際に配置されたアドレスを確認するには，「2.5. 配置結果の確認方法」を参照してください。

2.1.3. 初期化付き変数の特定領域への配置

初期化付き変数を特定の領域に配置するには，SEGINIT プラグマまたは ABSOLUTE プラグマを使用します。
方法としては，以下の 2 種類があります。

- 直接アドレスを指定する（記述例 1，記述例 3）
- 論理セグメントに配置してリンカでアドレスを指定する（記述例 2）

以下に記述例を示します。

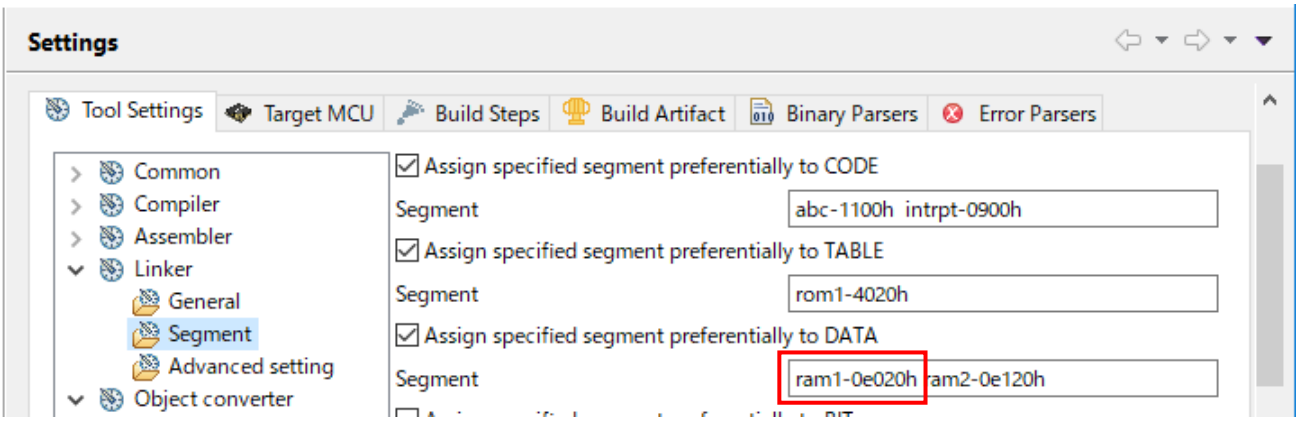
【記述例】

```
#pragma SEGINIT 0xE000        /*記述例 1 */
int var1 = 0x1234;            /*var1 は 0xE000 番地に配置されます */
int var2 = 0x5678;            /*var2 は 0xE002 番地に配置されます */

#pragma SEGINIT "ram1"        /*記述例 2 */
int var3 = 0x90AB;            /* var3,var4 は論理セグメント ram1 に配置されます*/
int var4 = 0xCDEF;            /* 配置アドレスは，リンカのオプションで指定できます */
#pragma SEGINIT               /* SEGINIT プラグマの終了 */

#pragma ABSOLUTE var5 0xE010 /*記述例 3 */
int var5 = 10;                /*var5 は 0xE010 番地に配置されます */
```

上記の記述例2で指定したセグメントの配置アドレスを指定する場合，LEXIDE-U16のLinkerの設定（[Tool Settings] > [Linker] > [Segment]）で，[Assign specified segment preferentially to DATA]の[Segment]フィールドに指定します。
以下の例では，セグメント ram1 を 0xE020 番地に配置するよう指定しています。



実際に配置されたアドレスを確認するには，「2.5. 配置結果の確認方法」を参照してください。

2.1.4. 初期化なし変数の特定領域への配置

初期化なし変数を特定の領域に配置するには，SEGNOINIT プラグマまたは ABSOLUTE プラグマを使用します。
方法としては，以下の 2 種類があります。

- 直接アドレスを指定する（記述例 1，記述例 3）
- 論理セグメントに配置してリンカでアドレスを指定する（記述例 2）

以下に記述例を示します。

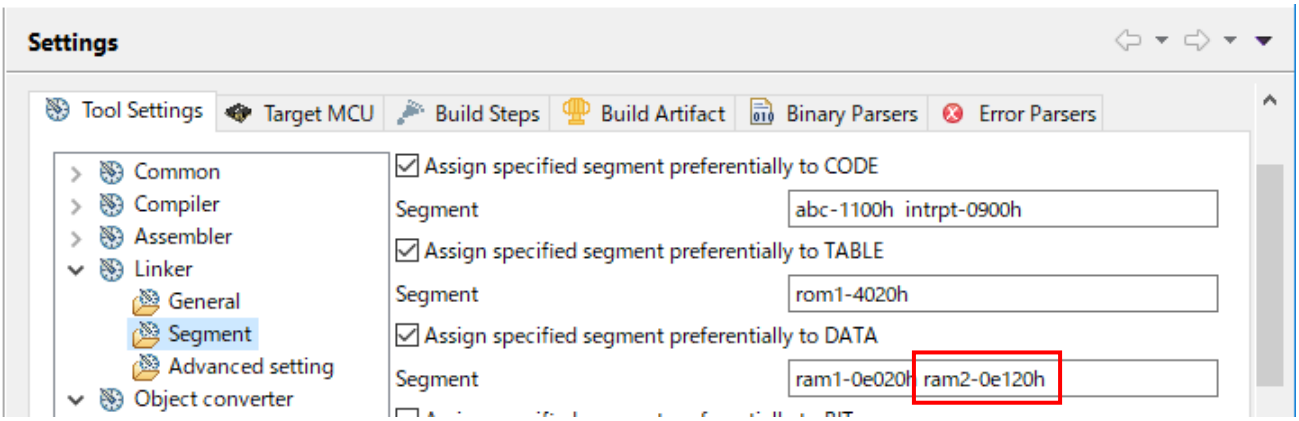
【記述例】

```
#pragma SEGNOINIT 0xE100      /*記述例 1 */
int ni_var1;                  /*ni_var1 は 0xE100 番地に配置されます */
int ni_var2;                  /*ni_var2 は 0xE102 番地に配置されます */

#pragma SEGNOINIT "ram2"      /*記述例 2 */
int ni_var3;                  /* ni_var3,ni_var4 は論理セグメント ram2 に配置されます*/
int ni_var4;                  /* 配置アドレスは，リンカのオプションで指定できます */
#pragma SEGNOINIT             /* SEGNOINIT プラグマの終了 */

#pragma ABSOLUTE ni_var5 0xE110 /*記述例 3 */
int ni_var5;                  /*ni_var5 は 0xE110 番地に配置されます */
```

上記の記述例2で指定したセグメントの配置アドレスを指定する場合，LEXIDE-U16のLinkerの設定（[Tool Settings] > [Linker] > [Segment]）で，[Assign specified segment preferentially to DATA]の[Segment]のフィールドに指定します。
以下の例では，セグメント ram2 を 0xE120 番地に配置するよう指定しています。



実際に配置されたアドレスを確認するには，「2.5. 配置結果の確認方法」を参照してください。

2.1.5. const 変数の特定領域への配置

const 変数を特定の領域に配置するには，SEGCONST プラグマまたは ABSOLUTE プラグマを使用します。
方法としては，以下の 2 種類があります。

- 直接アドレスを指定する（記述例 1，記述例 3）
- 論理セグメントに配置してリンカでアドレスを指定する（記述例 2）

以下に記述例を示します。

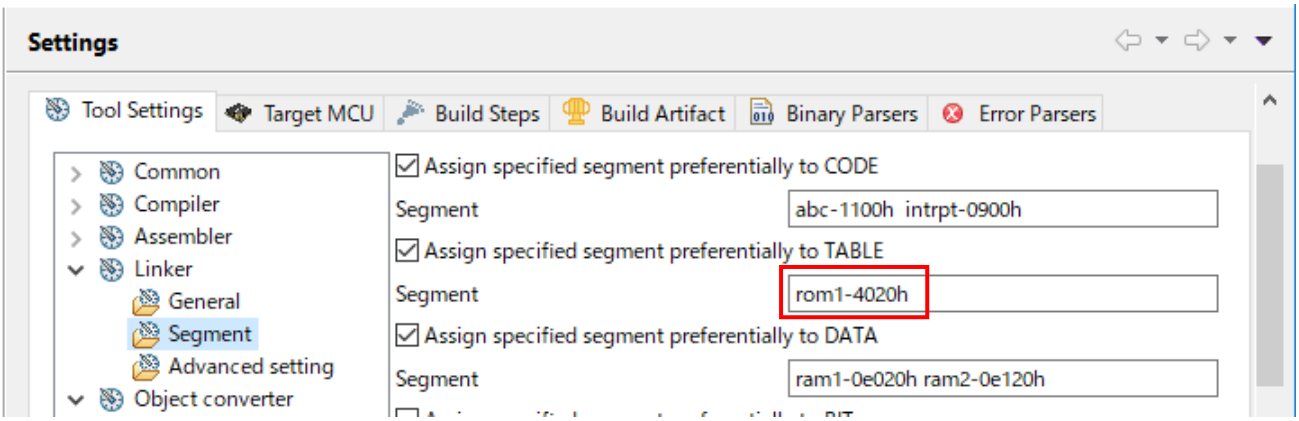
【記述例】

```
#pragma SEGCONST 0x4000       /*記述例 1 */
const int tvar1 = 10;         /* tvar1 は 0x4000 番地に配置されます */
const int tvar2 = 20;         /* tvar2 は 0x4002 番地に配置されます */

#pragma SEGCONST "rom1"       /*記述例 2 */
const int tvar3 = 30;         /* tvar3,tvar4 は論理セグメント rom1 に配置されます*/
const int tvar4 = 40;         /* 配置アドレスは，リンカのオプションで指定できます */
#pragma SEGCONST              /* SEGCONST プラグマの終了 */

#pragma ABSOLUTE tvar5 0x4010 /*記述例 3 */
const int tvar5 = 50;         /* tvar5 は 0x4010 番地に配置されます */
```

上記の記述例2で指定したセグメントの配置アドレスを指定する場合，LEXIDE-U16のLinkerの設定（[Tool Settings] > [Linker] > [Segment]）で，[Assign specified segment preferentially to TABLE]の[Segment]フィールドに指定します。
以下の例では，セグメント rom1 を 0x4020 番地に配置するよう指定しています。

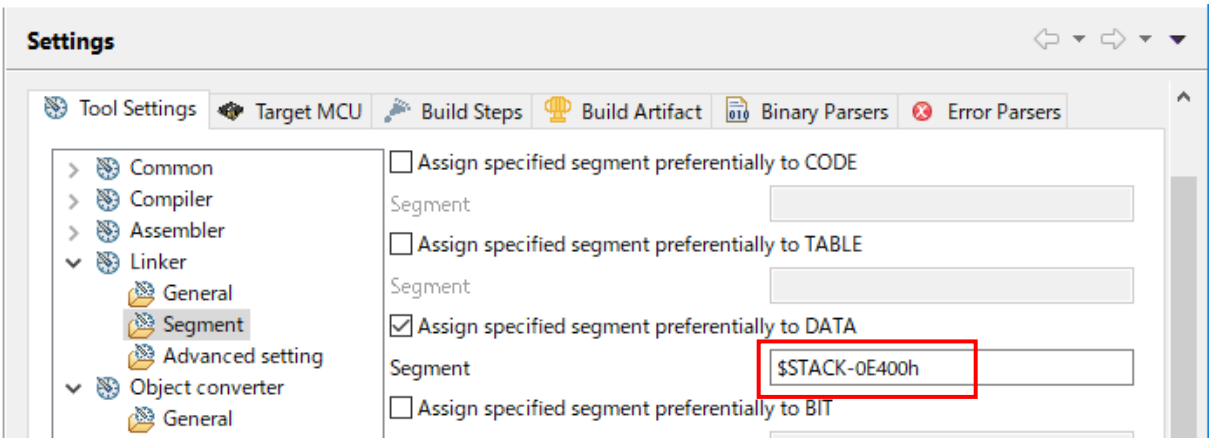


実際に配置されたアドレスを確認するには，「2.5. 配置結果の確認方法」を参照してください。

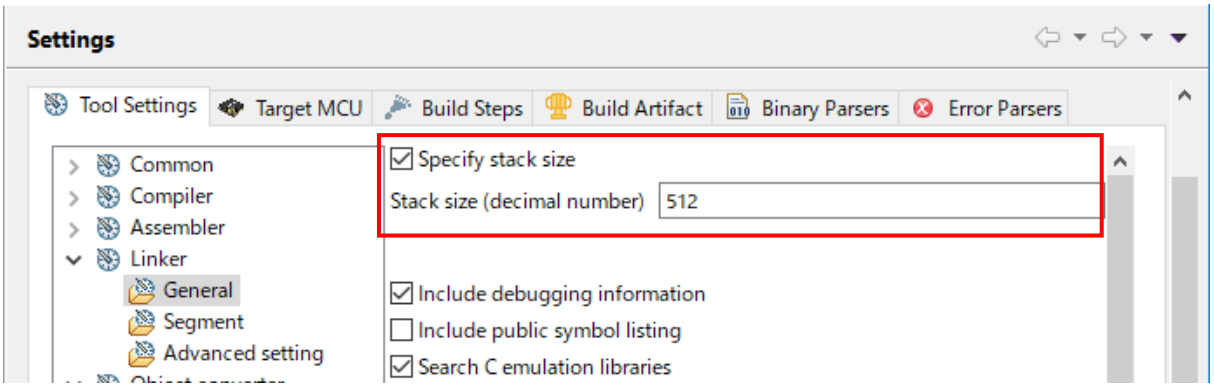
2.2. スタック領域の特定領域への配置

スタック領域には予め$STACK というセグメント名が付けられています。
スタック領域を特定の領域に配置するには，LEXIDE-U16 の Linker の設定（[Tool Settings] > [Linker] > [Segment]）で，[Assign specified segment preferentially to DATA]の[Segment]フィールドに$STACK を指定します。

以下の例では，スタック領域のセグメント$STACK を 0xE400 番地に配置するよう指定しています。



なお，スタック領域のサイズは LEXIDE-U16 の Linker の設定（[Tool Settings] > [Linker] > [Gneral]）で変更できます。
以下の例では，スタック領域のサイズを 512 バイトに設定しています。



スタックポインタの初期値は，（$STACK の先頭アドレス）＋（スタック領域のサイズ）に設定されます。
$STACK の配置アドレスが 0xE400，スタック領域のサイズが 512 バイト（0x200 バイト）の場合，スタックポインタの初期値（スタートアップファイルに記述されている_$$SP）は 0xE600 となります。

実際に配置されたアドレスを確認するには，「2.5. 配置結果の確認方法」を参照してください。

2.3. 関数や変数を記述順に配置

関数や変数をソースファイルの記述順に配置したい場合には，SEGCODE プラグマや SEGNOINIT プラグマなどで複数の関数や変数を囲んでください。このようにすることで，一つのセグメントに複数の関数や変数が記述順に配置されます。また，この場合，関数名や変数名を変更しても配置順が変わることはありません。

以下に記述例を示します。

【記述例】

```
#pragma SEGNOINIT "DATA1"     /* この行以降に記述された初期化なしの変数は，記述した順に*/
                              /* セグメント DATA1 内に配置されます */
unsigned short ni_var1;
unsigned short ni_var2;
unsigned short ni_var3;
unsigned short ni_var4;
unsigned short ni_var5;
unsigned short ni_var6;
#pragma SEGNOINIT             /* SEGNOINIT プラグマの有効範囲はここまでになります */

#pragma SEGCODE "CODE1"       /* この行以降に記述された関数は，記述した順に*/
                              /* セグメント CODE1 内に配置されます */
void func_ABC() {
    …
}

void func_DEF() {
   …
}

void func_GHI() {
   …
}
#pragma SEGCODE               /* SEGCODE プラグマの有効範囲はここまでになります */
```

【注意】

- 関数や変数の割り付け単位となるセグメントは，デフォルトでは関数単位および変数単位に分割され，ソースファイルに記述した順番には配置されません。配置の順番は，リンカによってセグメント名をもとにランダムに並べ替えられるためです。なお，デフォルトで生成されるセグメントの名前は，関数名，変数名，およびファイル名がベースとなります。このため，関数名や変数名，ファイル名のいずれかを変えると，それに伴いセグメント名が変わることでセグメントの配置が変わり，結果的に HEX コードが変わる場合があります。

- SEGCODE プラグマ，または SEGCONST プラグマを使って，複数の関数または複数の const 変数を一つのセグメントにまとめた場合，以下の理由により ROM サイズが増加する場合があります。
  - SEGCODE プラグマを使って複数の関数を一つのセグメントにまとめた場合，そのセグメント内の関数のうち，一つでも参照される関数が含まれていれば，参照されない関数が含まれていても，参照・未参照に関わらずすべて配置されます。
  - SEGCONST プラグマを使って複数の const 変数を一つのセグメントにまとめた場合，そのセグメント内の const 変数のうち，一つでも参照される const 変数が含まれていれば，参照されない const 変数が含まれていても，参照・未参照に関わらずすべて配置されます。

2.4. 【参考】リンカ用応答ファイルの利用

LEXIDE-U16 の Linker の設定（[Tool Settings] > [Linker] > [Segment]）で指定するセグメントのアドレス配置指定が多くなってくると，セグメント指定用のフィールドにすべてが表示しきれなくなり，全体がつかみにくくなってきます。
このようなときに，応答ファイルを作成することでオプションの確認がしやすくなります。
応答ファイルは，LEXIDEU16 の Linker の設定（[Tool Settings] > [Linker] > [General]）にて指定できます。

リンカ用の応答ファイルの記述例を以下に示します。
この例は，「2.1. 関数や変数の特定領域への配置」および「2.2. スタック領域の特定領域への配置」の LEXIDE-U16 の Linker の設定（[Tool Settings] > [Linker] > [Segment]）を応答ファイルに置き換えた例を示しています。オプション記号はスラッシュ‘/’ではなく，ハイフン‘-’を記述してください。

【記述例】

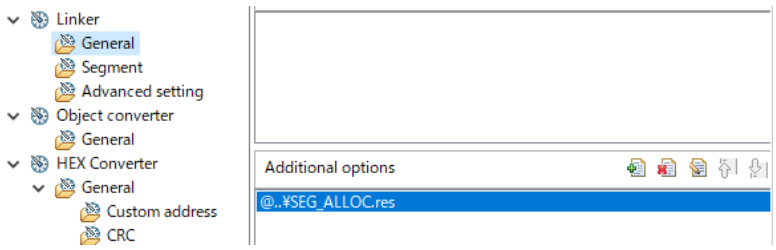
```
//応答ファイル名:SEG_ALLOC.res

-CODE(abc-1100h intrpt-0900h)
-TABLE(rom1-4020h)
-DATA(ram1-0E020h ram2-0E120h)
-DATA($STACK-0E400h)
```

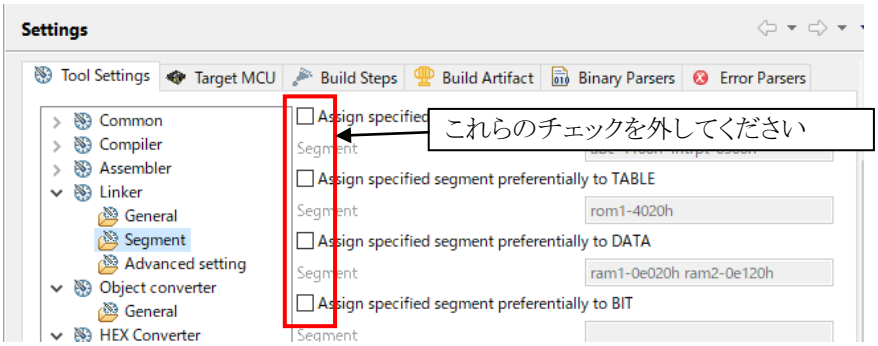
リンカ用の応答ファイルの詳細については，『MACU8 アセンブラパッケージ ユーザーズマニュアル』の「7.2.2.2. 応答ファイルによる入力の指定」を参照してください。
オプションの詳細については，『MACU8 アセンブラパッケージ ユーザーズマニュアル』の「7.5 オプション」を参照してください。

応答ファイルを LEXIDE-U16 で指定する場合，以下を設定してください。

[Tool Settings] > [Linker] > [General]の[Additional options]に応答ファイルをファイルを指定します。上記例の応答ファイル SEG_ALLOC.res を，プロジェクトファイル（.project/.cproject）が保存されているフォルダに保存した場合，[Additional options]のフィールドに“@..¥SEG_ALLOC.res”を指定してください。



[Tool Settings] > [Linker] > [Segment]で指定していた Assign specified segment preferenctially to *XXXX* のチェックは外してください。



2.5. 配置結果の確認方法

関数や変数等の配置結果は，マップファイルで確認できます。マップファイルについては，『MACU8 アセンブラパッケージ ユーザーズマニュアル』の「7.7 マップファイル」を参照してください。

以下に，「2.1. 関数や変数の特定領域への配置」および「2.2. スタック領域の特定領域への配置」で指定した場合の，マップファイルの出力例を示します。
ソースファイルにてアドレスを直接指定したセグメントは，(absolute)として出力されます。名前付きで指定したセグメントは，そのセグメント名がマップファイルに出力されます。

```
                        ------------------
                         Segment Synopsis
                        ------------------

Link Map - Program memory space #0 ( ROMWINDOW: 0000 - DFFF )

      Type      Start       Stop        Size                Name
     ------------------------------------------------------------
      S CODE    00:0000     00:0001     0002(2)             (absolute)
      S CODE    00:0002     00:0003     0002(2)             (absolute)
      ...
>GAP<           00:00E8     00:07FF     0718(1816)          (ROM)
      S CODE    00:0800     00:0803     0004(4)             (absolute)   <- 「2.1.2」の記述例 1 に該当
>GAP<           00:0804     00:08FF     00FC(252)           (ROM)
      S CODE    00:0900     00:0903     0004(4)             intrpt       <- 「2.1.2」の記述例 2 に該当
>GAP<           00:0904     00:0FFF     06FC(1788)          (ROM)
      S CODE    00:1000     00:1001     0002(2)             (absolute)   <- 「2.1.1」の記述例 1 に該当
>GAP<           00:1002     00:10FF     00FE(254)           (ROM)
      S CODE    00:1100     00:1101     0002(2)             abc          <- 「2.1.1」の記述例 2 に該当
>GAP<           00:1102     00:3FFF     2EFE(12030)         (ROM)
      S TABLE   00:4000     00:4001     0002(2)             (absolute)   <- 「2.1.5」の記述例 1 に該当
      S TABLE   00:4002     00:4003     0002(2)             (absolute)
>GAP<           00:4004     00:400F     000C(12)            (ROM)
      S TABLE   00:4010     00:4011     0002(2)             (absolute)   <- 「2.1.5」の記述例 3 に該当
>GAP<           00:4012     00:401F     000E(14)            (ROM)
      S TABLE   00:4020     00:4023     0004(4)             rom1         <- 「2.1.5」の記述例 2 に該当
>GAP<           00:4024     00:FFBF     BF9C(49052)         (ROM)
      ...

Link Map - Data memory space #0

      Type      Start       Stop        Size                Name
     ------------------------------------------------------------
      Q ROMWIN  00:0000     00:DFFF     E000(57344)         (ROMWIN)
      S DATA    00:E000     00:E001     0002(2)             (absolute)   <- 「2.1.3」の記述例 1 に該当
      S DATA    00:E002     00:E003     0002(2)             (absolute)
>GAP<           00:E004.0   00:E00F.7   000C.0(12.0)        (RAM)
      S DATA    00:E010     00:E011     0002(2)             (absolute)   <- 「2.1.3」の記述例 3 に該当
>GAP<           00:E012.0   00:E01F.7   000E.0(14.0)        (RAM)
      S DATA    00:E020     00:E023     0004(4)             ram1         <- 「2.1.3」の記述例 2 に該当
>GAP<           00:E024.0   00:E0FF.7   00DC.0(220.0)       (RAM)
      S DATA    00:E100     00:E101     0002(2)             (absolute)   <- 「2.1.4」の記述例 1 に該当
      S DATA    00:E102     00:E103     0002(2)             (absolute)
>GAP<           00:E104.0   00:E10F.7   000C.0(12.0)        (RAM)
      S DATA    00:E110     00:E111     0002(2)             (absolute)   <- 「2.1.4」の記述例 3 に該当
>GAP<           00:E112.0   00:E11F.7   000E.0(14.0)        (RAM)
      S DATA    00:E120     00:E123     0004(4)             ram2         <- 「2.1.4」の記述例 2 に該当
>GAP<           00:E124.0   00:E3FF.7   02DC.0(732.0)       (RAM)
      S DATA    00:E400     00:E5FF     0200(512)           $STACK       <- 「2.2」の例に該当
>GAP<           00:E600.0   00:EFFF.7   0A00.0(2560.0)      (RAM)
      Q SFR     00:F000     00:FFFF     1000(4096)          (SFR)
```

各関数および各変数のアドレスは，“Symbol Tabe Synopsis”にモジュールごとに出力されます。

```
          ----------------------
           Symbol Table Synopsis
          ----------------------

Module          Value     Type          Symbol
------          -----     ----------    ------
main
                00:0012   Pub CODE      _main
                00:1000   Pub CODE      _func1
                00:1100   Pub CODE      _func2
                00:4000   Pub TABLE     _tvar1
                00:4002   Pub TABLE     _tvar2
                00:4020   Pub TABLE     _tvar3
                00:4022   Pub TABLE     _tvar4
                00:4010   Pub TABLE     _tvar5
                00:E000   Pub DATA      _var1
                00:E002   Pub DATA      _var2
                00:E020   Pub DATA      _var3
                00:E022   Pub DATA      _var4
                00:E010   Pub DATA      _var5
                00:E100   Pub DATA      _ni_var1
                00:E102   Pub DATA      _ni_var2
                00:E120   Pub DATA      _ni_var3
                00:E122   Pub DATA      _ni_var4
                00:E110   Pub DATA      _ni_var5
                00:0900   Loc CODE      _intr_func2
...
```

【注意】

static 修飾された関数および変数を，ソースファイルにてアドレスを直接指定した場合には，それらのアドレス情報は出力されません。static 修飾された関数および変数アドレス情報を出力させたい場合は，ソースファイルには直接アドレスを指定せずに，セグメント名を指定するようにしてください。

改版履歴

ドキュメント No.	発行日	ページ		変更内容
		改版前	改版後	
FJXT_MCU_HOWTO_ALLOC -01	2019.7.22	ー	ー	初版発行