注意: この日本語版文書は参考資料としてご利用ください。
最新情報は必ずオリジナルの英語版をご参照願います。

Hexmate ユーザガイド

お客様へのご注意

どのような文書でも内容は時間と共に古くなります。本書も例外ではありません。Microchip 社のツールとマニュアルはお客様のニーズを満たすために常に改良を重ねており、実際のダイアログやツールの説明が本書に記載されているものと異なる場合があります。最新文書は Microchip 社のウェブサイト(www.microchip.com)をご覧ください。

文書は「DS」番号によって識別されます。この識別番号は、各ページのフッタ部分、ページ番号の前に記載しています。DS 番号「DSXXXXXA」の「XXXXX」は文書番号、「A」はリビジョンレベルを表します。

開発ツールの最新情報は MPLAB® IDE のオンラインヘルプでご覧になれます。[Help]メニューから[Topics]を選択すると、オンラインヘルプ ファイルのリストが表示されます。

© 2023 Microchip Technology Inc.
and its subsidiaries

目次

1. 序章

1.1 本書の表記規則

本書には以下の表記規則を適用しています。

表 1-1. 本書の表記規則

表記	意味	例
Arial、MS ゴシックフォント:		
二重かぎカッコ: 『』 太字	参考資料	Hexmate ユーザガイド
	テキストの強調	...は**唯一の**コンパイラです...
角カッコ: [ ]	ウィンドウ名	[Output]ウィンドウ
	ダイアログ名	[Settings]ダイアログ
	メニューの選択肢	[File]、[Save]を順に選択します。
かぎカッコ: 「 」	ウィンドウまたはダイアログのフィールド名	「Save project before build」
右山カッコ(>)で区切り、角カッコ([ ])で囲んだ下線付きテキスト	メニューパス	[File] > [Save]
角カッコ([ ])で囲んだ太字のテキスト	ダイアログのボタン	**[OK]**をクリックする
	タブ	**[Power]**タブをクリックする
N ‘Rnnnn	Verilog 形式の数値(N は総桁数、R は基数、n は各桁の値)	4 ‘b0010, 2 ‘hF1
山カッコ(< >)で囲んだテキスト	キーボードのキー	<Enter>、<F1>を押す
Courier New フォント		
標準書体の Courier New	サンプル ソースコード	`#define START`
	ファイル名	`autoexec.bat`
	ファイルパス	`C:\Users\User1\Projects`
	キーワード	`static, auto, extern`
	コマンドライン オプション	`-Opa+, -Opa-`
	ビット値	`0, 1`
	定数	`0xFF, ‘A’`
斜体 Courier New	変数の引数	`file.o` (`file` は有効な任意のファイル名)
角カッコ: [ ]	オプションの引数	`xc8 [options] files`
中カッコとパイプ文字: { | }	どちらかの引数を選択する場合 (OR 選択)	`errorlevel {0|1}`
省略記号: ...	繰り返されるテキスト	`var_name [, var_name...]`

	ユーザが定義するコード	`void main (void)`
		`{ ...`
		`}`

1.2 推奨参考資料

この文書には、Hexmate アプリケーションの使い方と機能を記載しています。参考資料として、Microchip 社が提供する以下の文書を推奨します。

MPLAB® XC8 C コンパイラ ユーザガイド
Hexmate は MPLAB XC8 C コンパイラに付属するアプリケーションで、同コンパイラによって自動的に実行されます。同コンパイラを使ってプロジェクトをビルドする場合、Hexmate のほとんどの機能にコンパイラから直接アクセスできます。本書には Hexmate の関数を呼び出すコンパイラ オプションを記載します。

PIC® MCU 向け MPLAB® XC8 C コンパイラ リリースノート
Hexmate に加えられた変更点はコンパイラのインストール フォルダの Docs サブフォルダにある『MPLAB® XC8 C コンパイラ リリースノート』(HTML ファイル)に記載されています。リリースノートには、本書に記載できなかった最新情報と既知の問題を記載しています。

開発ツールのリリースノート
各種開発ツールの最新情報については、MPLAB X IDE インストール フォルダの「docs」サブフォルダ内にある各ツールの Readme ファイルを参照してください。

© 2023 Microchip Technology Inc.
and its subsidiaries

2. Hexmate

Hexmate アプリケーションは Intel HEX ファイルのマージ、再フォーマット、有用な情報の挿入のために設計されたポストリンク段階のユーティリティです。

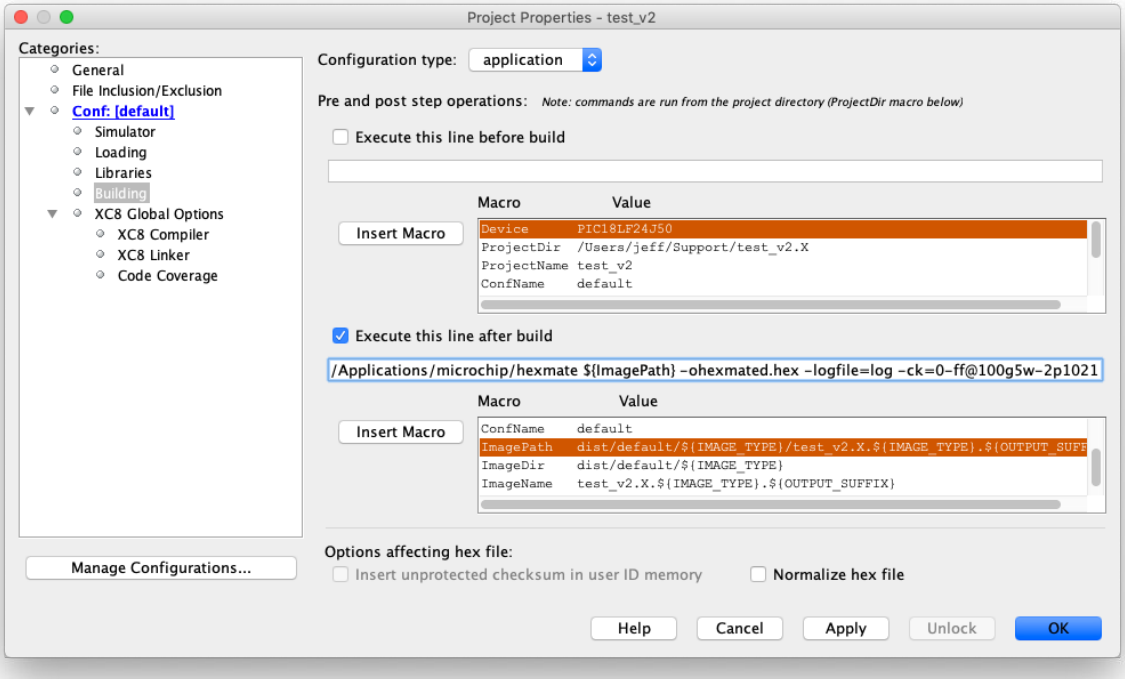
Hexmate は MPLAB XC8 C コンパイラに付属しており、同コンパイラによって自動的に実行されますが、任意のコンパイラで生成された Intel HEX ファイルを処理するためにスタンドアロン アプリケーションとして実行する事もできます。MPLAB X IDE にも Hexmate が別途付属しており、ロード可能プロジェクトをビルドする際に HEX ファイルをマージするのに使われます。

本書には、Hexmate で実行できるタスクと、アプリケーションを制御するコマンドライン オプションについての情報を記載します。

Hexmate の機能の一部は `xc8-cc` コマンドライン ドライバまたは MPLAB XC8 プロジェクトに関連するプロジェクト プロパティより直接アクセス可能ですが、それほど頻繁に利用されない機能についてはプロジェクトのビルド後に Hexmate を明示的に呼び出す必要があります。MPLAB XC16 または XC32 プロジェクトから生成された HEX ファイルに対して使う場合も、Hexmate を明示的に実行する必要があります。

Hexmate は MPLAB X IDE 内でリンク後のビルドコマンドとして追加できるため、ターミナル アプリケーションにアクセスしなくても実行できます。以下に示す MPLAB X IDE の[Project Properties]ダイアログのスクリーン ショットでは、ビルド後のステップとして Hexmate を実行し、プロジェクトが生成する HEX ファイルのデータの一部に対して CRC を計算しています。プロジェクトのビルドプロパティを設定する方法の詳細は『MPLAB® X IDE ユーザガイド』を参照してください。

図 2-1. MPLAB X IDE のビルド後ステップとして Hexmate を実行



2.1 Hexmate の用途

Hexmate は Intel HEX ファイルに関連する各種作業に使えます。これには以下が含まれます。

- 複数の Intel HEX ファイルを 1 つの Intel HEX ファイルにマージする。
- 可変長のハッシュ値(CRC または SHA 等)を計算し保存する。

- 未使用のメモリ位置を既知のデータシーケンスで埋める。
- INHX32 ファイルをその他の INHX フォーマット(INHX8M 等)に変換する。
- HEX ファイル内の特定のオペコード シーケンス(またはその一部)を検出する。
- 特定のオペコード シーケンス(またはその一部)を検索/置換する。
- HEX ファイルで使われているアドレスのマップを提供する。
- HEX ファイルのデータ レコードの長さを変更または固定する。
- Intel HEX ファイル内のチェックサムを検証する。

Hexmate の一般的な応用例を以下に示します。

- ビルド時にブートローダやデバッグ モジュールをメイン アプリケーションにマージする。
- プログラムメモリのレンジ全体でハッシュ値を計算し、その値をプログラムメモリまたは EEPROM に格納する。
- 何らかの問題により、未使用メモリ位置にプログラムカウンタが設定された場合に既知の位置に設定する命令で未使用領域を埋める。
- シリアル番号を固定アドレスに保存する。
- 文字列(タイムスタンプ等)を固定アドレスに保存する。
- 初期値を特定のメモリアドレスに保存する(EEPROM の初期化等)。
- バグのある命令または制限されている命令のシーケンスを検索する。
- 特定のブートローダの要件に合わせて HEX ファイルを調整する。

© 2023 Microchip Technology Inc.
and its subsidiaries

3. Hexmate のコマンドライン オプション

以下のコマンド フォーマットで Hexmate を直接実行します。

```
hexmate [specs,]file1.hex [... [specs,]fileN.hex] [options]
```

ここで、`file1.hex`～`fileN.hex` は Hexmate を使ってマージする入力 Intel HEX ファイルのリストです。`options` はコマンドラインの任意の場所に配置できるオプションで、下表にまとめます。

HEX ファイルを 1 つだけ指定した場合、マージは実行せずにオプションで指定した他の操作をその HEX ファイルに対して実行できます。

Hexmate では一般的な 8 ビット、16 ビット、32 ビットの名前付きフォーマットを読み書きできます。これには、レコードタイプの特定のサブセットのみが含まれます。フォーマットについては 3.16 Format オプションで説明します。

表 3-1. Hexmate のコマンドライン オプション

オプション (説明セクションにリンク)	影響
--edf=file	メッセージ記述ファイルを指定します。
--emax=n	終了までに許容される最大エラー数を設定します。
--msgdisable=number	指定した番号のメッセージを無効にします。
--sla=address	タイプ 5 レコードの開始リニアアドレスを設定します。
--ssa=address	タイプ 3 レコードの開始セグメント アドレスを設定します。
--ver	バージョンとビルドの情報を表示してから終了します。
-addressing=units	全ての Hexmate オプションのアドレス フィールドでワードアドレスまたはその他の方法を使うように設定します。
-break	連続データを分断して、設定したアドレスから新しいレコードが開始するようにします。
-ck=spec	ハッシュ値を計算し、保存します。
-fill=spec	未使用の位置に既知の値をプログラムします。
-find=spec	特定のコードシーケンスを検索し、検出した場合に通知します。
-find=spec,delete	コードシーケンスを検出した場合、削除します(注意してお使いください)。
-find=spec,replace=spec	コードシーケンスを新しいコードシーケンスに置き換えます。
-format=type	最大データレコード長を指定する、または INHX の種類を選択します。
-help	全てのオプションを表示します。または、特定のオプションのヘルプメッセージを表示します。
-logfile=file	Hexmate の解析の出力と各種結果をファイルに保存します。
-mask=spec	メモリレンジをビットマスクで論理的に AND します。
-ofile	出力ファイルの名前を指定します。
-serial=spec	シリアル番号またはコードシーケンスを固定アドレスに保存します。
-size	結果の HEX イメージに含まれるデータのバイト数を報告します。
-string=spec	ASCII 文字列を固定アドレスに保存します。
-strpack=spec	文字列パッキングを使って ASCII 文字列を固定アドレスに保存します。
-wlevel	警告の感度を調整します。

© 2023 Microchip Technology Inc.
and its subsidiaries

+ オーバーライド接頭辞	必要に応じて任意のオプションに接頭辞として付加し、そのアドレスレンジ内の他のデータを上書きします。

オプションに関連する値のフォーマットや想定される基数は各オプションの説明に記載されています。`-addressing` オプションを使って変更していない限り、これらのオプションに指定するアドレス フィールドは HEX ファイルアドレスとして入力する事に注意してください。

3.1 指定とファイル名

Hexmate は INHX32 または INHX8M フォーマットの Intel HEX ファイルを処理できます。コマンドラインにリスト表示された各 HEX ファイルに追加の指定を適用し、そのファイルの処理方法に制限または条件を課す事ができます。

指定を使う場合、ファイル名の前に配置する必要があります。ファイル名とカンマで区切って指定のリストを記述します。

レンジの制限は指定 `rStart-End` で適用できます。`Start` と `End` では 16 進値が想定されます。Hexmate はこのアドレスレンジ制限内のデータのみを処理します。以下に例を示します。

```
r100-1FF,myfile.hex
```

上記は `myfile.hex` を入力としますが、そのファイルのうちアドレスレンジが 0x100-1FF(指定アドレスを含む)のデータのみを処理します。

アドレスシフトは指定 `sOffset` で適用できます。ここで、`Offset` は非修飾の 16 進値が想定されます。アドレスシフトを使うと、その HEX ファイルから読み込まれたデータは出力ファイルの新しいアドレスに(指定されたオフセットだけ)シフトされます。正または負のオフセットが可能です。以下に例を示します。

```
r100-1FFs2000,myfile.hex
```

上記は `myfile.hex` の 0x100-1FF のデータブロックを出力のアドレスレンジ 0x2100-21FF にシフトします。

実行可能コードのセクションをシフトする場合は注意が必要です。プログラムコードは位置独立である場合のみシフトさせるようにします。

3.2 オーバーライド接頭辞

入力ファイルまたはオプションの前に+接頭辞を付けると、そのファイルから取得したデータ、またはそのオプションによって生成されたデータを優先して強制的に出力ファイルに書き出します。同じアドレスに他のデータが存在した場合、通知なしで上書きされます。

この接頭辞を付けるとデータの上書きに関する警告が抑制されるため、注意してお使いください。この接頭辞を付けずに 2 つのソースが同じ場所に異なるデータを保存しようとした場合、Hexmate はエラーを出します。

例えば、`input.hex` のアドレス 0x1000 にデータが存在している場合に、以下のコマンド(+と`-string` オプションの組み合わせ)を実行したとします。

```
hexmate input.hex +-string@1000="My string"
```

この場合、`-string` オプションで指定された文字列が出力ファイルのアドレス 0x1000 に配置され、入力ファイルのそのアドレスにあったデータは出力されなくなります。一方、以下のコマンド(+接頭辞なし)を使った場合、

```
hexmate input.hex -string@1000="My string"
```

エラーがトリガーされ、データの競合について警告が出力されます。

また、オーバーライド接頭辞をファイルと合わせて使うと、複数のファイルの同じアドレスに競合するデータが含まれている場合にあるファイルを他のファイルより優先するよう指定できます。例えば、2 つのファイルをマージする場合、以下のようなコマンドが使えます。

© 2023 Microchip Technology Inc. and its subsidiaries

```
hexmate +base.hex auxillary_data.hex
```

これは、base.hex の内容が優先される事を示します。

3.3 Edf

--edf=file は、警告メッセージまたはエラーメッセージの表示に使われるメッセージ記述ファイルを指定します。このオプションの引数にはメッセージ ファイルのフルパスを指定します。ほとんどのアプリケーションにはメッセージ ファイルの内部コピーが含まれているため、通常、このオプションは必要ありません。このオプションは、内容が更新された別のファイルを指定する場合に使います。

MPLAB XC8 C コンパイラにはメッセージ記述ファイルが同梱されており、コンパイラの pic/dat フォルダにあります。同梱されているファイルの名前は build_en.msgs です。ファイル名にビルド日(yyyymmddhhmmss フォーマット)が含まれているため、より新しい日付のファイルを見つけるのに役立ちます。

3.4 Emax

--emax=num オプションは Hexmate が実行を終了するまでに表示するエラーの最大数を設定します。

例えば、--emax=25 等です。既定値では、最大で 20 件のエラーメッセージが表示されます。

3.5 Msgdisable

--msgdisable=number オプションは Hexmate 実行中のエラー、警告、またはアドバイザリ メッセージを無効化できます。

このオプションには無効にするメッセージ番号のカンマ区切りのリストを渡します。その後に:off 引数が続く場合を除き、このリストに含まれるエラーメッセージは無視されます。以下に例を示します。

```
--msgdisable=2031,963
```

メッセージリストに 0 を指定すると、全ての警告が無効化されます。

3.6 Sla Hexmate オプション

--sla=address オプションを使うと、INHX32 または INHX032 出力ファイルのタイプ 5 レコードに SLA(リニア開始アドレス)を指定できます。例えば、--sla=0x10000 とすると、出力 HEX ファイルにペイロードが 0x10000 のタイプ 5 レコードが含まれます。例を以下に示します。

```
:040000050010000F6
```

このオプションを使うと、入力ファイルに存在する全ての入力 SLA レコードについて、構文とチェックサムが正しいかどうかがチェックされます。正しくない場合はエラーが発生します。これらの SLA レコードは警告**なし**で破棄されます。出力ファイル フォーマットが INHX32 または INHX032 でない場合、警告が発生し、SLA レコードは書き込まれません。INHX32 または INHX032 の場合、オプションで指定した値を含む 1 つの SLA レコードのみが出力に現れます。

このオプションを使わない場合、入力ファイルに存在する全ての入力 SLA レコードについて、構文とチェックサムが正しいかチェックされます。これらのレコードで指定されたアドレスに相違がない場合、そのアドレスを持つ 1 つの SLA レコードのみが出力に書き込まれます。入力ファイルに存在する SLA レコード間に相違がある場合、警告メッセージが表示され、SLA レコードは出力に**現れません**。入力ファイルに SLA レコードが存在しない場合、SLA レコードは出力に書き込まれません。

3.7 Ssa Hexmate オプション

`--ssa=address` オプションは、INHX16 出力ファイルのタイプ 3 レコードにセグメント開始アドレス(SSA)を指定できるようにします。例えば、`--ssa=0x10000` の場合、出力 HEX ファイルにペイロードが 0x10000 のタイプ 3 レコードが含まれるようになります。例を以下に示します。

```
:040000030010000F8
```

このオプションを使うと、入力ファイルに存在する全ての SLA レコードについて、構文とチェックサムが正しいかどうかがチェックされます。正しくない場合はエラーが発生します。これらの SSA レコードは警告**なし**で破棄されます。出力ファイル フォーマットが INHX16 でない場合、警告が発生し、SSA レコードは書き込まれません。INHX16 の場合、オプションで指定した値を含む 1 つの SSA レコードのみが出力に現れます。

このオプションを使わない場合、入力ファイルに存在する全ての SSA レコードについて、構文とチェックサムが正しいかチェックされます。これらのレコードで指定されたアドレスに相違がない場合、そのアドレスを持つ 1 つの SSA レコードのみが出力に書き込まれます。入力ファイルに存在する SSA レコード間に相違がある場合、警告メッセージが表示され、SSA レコードは出力に**現れません**。入力ファイルに SSA レコードが存在しない場合、SSA レコードは出力に書き込まれません。

3.8 Ver

`--ver` オプションはバージョンとビルドの情報を出力してから終了するように Hexmate に要求します。

3.9 Addressing

`-addressing=units` オプションを使うと、Hexmate のコマンドライン オプションにある任意のアドレスのアドレス指定単位を既定値の 1 から最大で 4 に変更できます。

既定値では、Hexmate のオプションで指定される全てのアドレス引数を Intel HEX ファイルで使われているバイトアドレスとみなします。例えば、オプション`-mask=0F@0-FF` を指定すると、アドレス 0x0 からアドレス 0xFF までの任意の HEX ファイルデータに対してマスクが実行されます。デバイス アーキテクチャによっては、ネイティブなアドレス指定フォーマットがバイトアドレス指定ではない場合があります。例えば、HEX ファイルのアドレス 0x200 に 0x0F、アドレス 0x201 に 0x55 が含まれているとします。ところが、この HEX ファイルがミッドレンジの PIC デバイスに読み込まれると、これらのバイトはデバイス内のアドレス 0x100 で 1 ワードを構成します。この場合、各デバイスアドレスのワード値は HEX ファイルの別々のアドレスの 2 バイト値に拡張されます。Hexmate オプションでデバイスアドレスを使うには、このオプションにより HEX ファイルアドレスとデバイスアドレス間の割り当てを指定します。

このオプションは 1 つのパラメータを取ります。このパラメータは各デバイスアドレス位置に格納される HEX ファイルのバイト数を示します。このパラメータは 1～4 の値を取る事ができます。8 ビット AVR デバイス、ベースライン、ミッドレンジ、24 ビット PIC デバイスの場合、必要に応じてアドレス指定単位を 2 に指定できます(例: `-addressing=2`)。その後は、全ての Hexmate オプションにデバイスアドレスを指定できます。Microchip 社のその他全てのデバイスでは、通常、既定値のアドレス指定単位である 1 バイトを使う(例: `-addressing=1`)か、このオプション全体を省略できます。これらのデバイスでは HEX ファイルアドレスとデバイスアドレスは全ての位置で同じであるため、割り当ては不要です。

3.10 Break

`-break` オプションは非修飾の 16 進アドレスのカンマ区切りリストを指定します。指定したアドレスのいずれかが HEX ファイル内に見つかった場合、現在のデータレコードを終了させ、指定されたアドレスから新しいデータレコードを再開します。

© 2023 Microchip Technology Inc.
and its subsidiaries

例えば、Hexmate の通常の出力内容が以下であるとします。

```
:100000000EEF0AF03412ADDEADDEADDEADDEADDEFC
:10001000ADDEADDEADDEADDEADDEADDEADDEADDE88
:10002000ADDEADDEADDEADDEADDEADDEADDEADDE78
...
```

この時、`-break=16` オプションを使うと、出力は以下のようになります。

```
:100000000EEF0AF03412ADDEADDEADDEADDEADDEFC
:06001000ADDEADDEADDE49
:10001600ADDEADDEADDEADDEADDEADDEADDEADDE82
:10002600ADDEADDEADDEADDEADDEADDEADDEADDE72
...
```

データ レコードを分断する事で、プログラム領域のうち機能的に異なる領域を区別できます。このようなレコード配置を必要とする HEX ファイルリーダーも存在します。

3.11 Ck Hexmate オプション

`-ck` オプションはハッシュ値の計算に使います。このオプションは以下のように使います。

```
-ck=start-end@dest[+offset][wWidth][tCode[.Base]][gAlgorithm][pPolynomial][rRevWidth]
[sSkipWidth[.SkipBytes]][oXORvalue]
```

- `start` と `end` は、ハッシュ計算の対象とする 16 進アドレスレンジを指定します。指定したアドレスがチェックサムおよびフレッチャー アルゴリズムのデータ幅の倍数でない場合、欠けている該当入力ワード位置に値 0 がパディングされます。
- `dest` はハッシュの結果を格納する 16 進アドレスです。このアドレスはハッシュ計算の対象とするアドレスレンジ内であってはなりません。
- `offset` はハッシュの計算に使う 16 進数の初期値で、必須ではありません。SHA アルゴリズムでは使いません。
- `Width` は必須ではなく、結果の 10 進数の幅を指定します。ほとんどのアルゴリズムでは 1～4 バイトのバイト幅で結果を計算できますが、SHA アルゴリズムではビット幅を表します。正の幅を要求した場合、結果はビッグ エンディアンのバイトオーダで格納されます。負の幅を指定した場合、結果はリトル エンディアンのバイトオーダで格納されます。幅を指定しない場合、結果は 2 バイト幅でリトル エンディアンのバイトオーダで格納されます。この幅の引数は固定幅であるフレッチャー アルゴリズムでは必要ありませんが、結果の既定値のエンディアンを変更するために使えます。
- `Code` は結果の各バイトに後続する 16 進コードシーケンスで、必須ではありません。この機能は、ハッシュ結果の各バイトを命令内に埋め込む必要がある場合、または実行時にデバイスで読み込むのにハッシュ値をパディングする必要がある場合に使います。例えば、`t34` と指定すると、ミッドレンジ PIC デバイスの `retlw` 命令(ビットシーケンス 0x34xx)に結果の各バイトを埋め込みます。コードシーケンスに複数のバイトを指定した場合、ハッシュバイトの後にビッグ エンディアンのバイトオーダで格納されます。例えば、`tAABB` と指定するとハッシュバイトの直後に 0xAA が、その次のアドレスに 0xBB が付加されます。後続コードとして `t0000` を指定した場合、2 つの 0x00 バイトがハッシュの各バイトの後に格納されます。オプションで、コードシーケンス引数の後に`.Base` を指定できます。ここで、`Base` は後続コードシーケンスを付加する前に出力するハッシュのバイト数です。例えば、`t11.2` と指定すると、ハッシュ結果の各 2 バイトの後にバイト 0x11 が出力されます。
- `Algorithm` は結果の計算に使う Hexmate ハッシュ アルゴリズムを選択する 10 進整数です。選択可能なアルゴリズムの一覧を下表に示します。指定しない場合、既定値で使われるアルゴリズムは 8 ビット チェックサム加算(アルゴリズム 1)です。
- `Polynomial` は CRC アルゴリズムを選択した場合に使う多項式の 16 進値です。
- `RevWidth` は逆順ワード幅で、必須ではありません。0 以外を指定した場合、ハッシュ値を計算する際に各ワード内のバイトを逆順に読み出します。ワードは HEX ファイルのアドレスに整列されます。現時点では、幅は 0 または 2 にする必要があります。幅を 0 にした場合、逆順バイト機能が無効になり、`r` サブオプションが存在しない場合と同じように扱われます。このサブオプションは、スキャナ モジュールを使ってデータをストリーミングする PIC ハードウェア CRC モジュールで生成される CRC と一致させるために Hexmate を使う状

況を想定しています。この機能は全てのハッシュタイプで動作しますが、チェックサム アルゴリズム (アルゴリズム-4～4)を使う場合は効果がありません。

- `SkipWidth` はスキップワード幅で、必須ではありません。0 以外を指定した場合、ハッシュ値を計算するために各ワード内の最上位アドレスのバイトがスキップされます。ワードは HEX ファイルのアドレスに整列されます。現時点では、幅は 0 にする(`s` サブオプションが存在しない場合と同様)、または 1 より大きくする必要があります。オプションで、スキップ幅引数の後に`.SkipBytes` を指定できます。ここで、`SkipBytes` は各ワード内でスキップするバイト数を表す数値です。例えば、`s4.2` の場合、各 4 バイトワードの最上位アドレスの 2 バイトをスキップします。ハッシュ計算時に MPLAB XC16 C コンパイラによって HEX ファイルに追加される「ファントム」の 0x00 バイトを処理しないようにするには、`s4` を使います。
- `XORvalue` は、保存前にハッシュ結果と XOR される 16 進値です。

1 文字の引数トークンにおいて大文字と小文字は区別されません。例えば w2 と W2 はどちらもこのオプションの有効な幅の引数です。

このオプションを使ってチェックサムを計算する典型的な例を以下に示します。

```
-ck=0-1FFF@2FFE+2100w-2g2
```

この場合、0～0x1FFF のレンジでチェックサム(16 ビット加算)を計算し、アドレス 0x2FFE にチェックサムの結果をプログラムします。チェックサム値は 0x2100 だけオフセットされます。結果は 2 バイト幅で、リトル エンディアン形式で保存されます。

逆順機能とスキップ機能は、処理中のシーケンスのデータの開始バイトではなく、HEX ファイルのアドレスに整列されたワードに対して機能する事に注意してください。つまり、ワードの位置は`-ck` オプションで指定された開始アドレスと終了アドレスの影響を受けません。以下のオプションを考えてみます。

```
-ck=0-5@100w2g5p1021s2
```

上記はハッシュ値を計算する時、HEX アドレス 0～5 まで 2 バイトおきにスキップする事(s2)を示します。HEX レコードに対して機能する場合、以下のようになります(下線部のデータ)。

```
:10000000<u>64002500030A75007600770078006400</u>1C
```

この時、ハッシュは(16 進数の)バイト 0x64、0x25、0x03 から計算されます。同じ HEX レコードを、異なる開始終了アドレスレンジ(1～6)を使ったオプションで処理すると、以下のようになります。

```
-ck=1-6@100w2g5p1021s2
```

この時、ハッシュは(16 進数の)バイト 0x25、0x03、0x75 から計算されます。これらの機能はデバイス上で実行されるコードのデータ読み出しの制限を模倣しようとするものです。そのため、使うワードはデバイスアドレスに整列され、そのアドレスは HEX ファイルアドレスに整列されます。

ハッシュ アルゴリズムの 1 回の反復で 16 ビットまたは 32 ビットのデータを処理するアルゴリズムで `skipWidth` にゼロ以外の値を指定した場合、スキップされるバイトは 0 バイトでパディングされます。例えば、HEX レコードが以下であるとします(下線部のデータ)。

```
:0400000<u>012345678</u>E8
```

この時、このデータに 16 ビット加算チェックサム (アルゴリズム 2)を適用すると、通常、0x3412 と 0x7856 が加算されます。`skipWidth` に 2 を指定した場合、入力が 2 バイトおきにスキップされるため、アルゴリズムは 0x0012 と 0x0056 を加算します。ただし、入力配列を 1 バイトずつ処理する CRC(アルゴリズム 5)が選択されている場合、バイト 0x12 と 0x56 は CRC アルゴリズムで処理されます。

表 3-2. Hexmate Hash アルゴリズムの選択

セレクタ	アルゴリズムの説明
-5	反転 CRC(巡回冗長検査)
-4	初期値から 32 ビット値を減算
-3	初期値から 24 ビット値を減算

-2	初期値から 16 ビット値を減算
-1	初期値から 8 ビット値を減算
1	初期値から 8 ビット値を加算
2	初期値から 16 ビット値を加算
3	初期値から 24 ビット値を加算
4	初期値から 32 ビット値を加算
5	CRC(巡回冗長検査)
7	フレッチャーのチェックサム(8 ビット計算、2 バイト結果幅)
8	フレッチャーのチェックサム(16 ビット計算、4 バイト結果幅)
10	SHA-2(現在は SHA256 のみサポート)。

ハッシュ計算に使われるアルゴリズムの詳細は 5. ハッシュ値の計算を参照してください。

3.12 Fill オプション

`-fill` オプションは HEX ファイル内の未使用(未指定)のメモリ位置を既知の値で埋めるために使います。このオプションは以下のように使います。

```
-fill=[wwidth:]fill_expr@address[:end_address]
```

各引数の意味を以下に示します。

wwidth

`fill_expr` 内の各定数の 10 進数幅を表し、1〜9 のレンジで指定できます。この幅を指定しない場合、既定値は 2 バイトです。例えば、`-fill=w1:0x55@0:0xF` はアドレス 0〜0xF の全ての未使用バイトを値 0x55 で埋めます。例えば、以下のようになります。

```
:10000000FBEF3FF05555555555555555555555555DB
```

一方、`-fill=w2:0x55@0:0xF` は同じアドレス間の全ての未使用バイトを値 0x0055 で埋めます。例えば、以下のようになります。

```
:10000000FBEF3FF05500550055005500550055005500D9
```

fill_expr

埋める値を定義します。これは最初のメモリ位置に配置する基準値である `const` と、この基準値が使うたびにどのように変化するかを示す任意指定の `increment` によって構成されます。基準値で複数のバイトを指定した場合、これらはリトル エンディアン バイトオーダで格納されます。フィルの可能な表現を以下に示します。

- `const` は定数の繰り返しでメモリを埋めます。すなわち、`-fill=0xBEEF@0:0x1FF` はアドレス 0 で始まる未使用位置を値 0xBEEF、0xBEEF、0xBEEF、0xBEEF(以下略)で埋めます。例を以下に示します。

  ```
  :10000000FBEF3FF0EFBEEFBEEFBEEFBEEFBEEFBEC9
  ```

- `const+=increment` はインクリメントする定数でメモリを埋めます。すなわち、`-fill=0xBEEF+=1@0:0x1FF` は値 0xBEEF、0xBEF0、0xBEF1、0xBEF2(以下略)で埋めます。例を以下に示します。

  ```
  :10000000FBEF3FF0F1BEF2BEF3BEF4BEF5BEF6BEAE
  ```

  `const` は、その位置が実装されているか未使用かを問わず、スキャンされる位置ごとにインクリメントされる事に注意してください。

- `const-=increment` はデクリメントする定数でメモリを埋めます。すなわち、`-fill=0xBEEF-=0x10@0:0x1FF` は値 0xBEEF、0xBEDF、0xBECF、0xBEBF(以下略)で埋めます。例を以下に示します。

```
:10000000FBEF3FF0CFBEBFBEAFBE9FBE8FBE7FBE79
```

ただし、const は、その位置が実装されているか未使用かを問わず、スキャンされる位置ごとにインクリメントされる事に注意してください。

- const,const,...,const は繰り返す定数のリストでメモリを埋めます。すなわち、-fill=0xDEAD,0xBEEF@0:0x1FF は 0xDEAD、0xBEEF、0xDEAD、0xBEEF で埋めます。例を以下に示します。

```
:10000000FBEF3FF0ADDEEFBEADDEEFBEADDEEFBE2F
```

@address 特定のアドレスを fill_expr で埋めます。例えば、-fill=0xBEEF@0x1000 は、アドレス指定値を 1 に設定した場合、アドレス 0x1000 にバイト値 0xEF を配置します。

```
:01100000EF00
```

上記の例でさらに-addressing=2 オプションを使った場合、fill オプションはアドレス 0x2000 と 0x2001 に 2 バイトを配置します。

```
:02200000EFBE31
```

:end_address は必須ではなく、fill_expr で埋めるメモリの終了アドレスを指定します。例えば、-fill=0xBEEF@0xF0:0xFF は 0～0xFF(指定アドレスを含む)の未使用アドレスに 0xBEEF を配置します。

```
:1000F000EFBEEFBEEFBEEFBEEFBEEFBEEFBEEFBE98
```

アドレスレンジ(-addressing 値を乗じた値)がフィル値の幅の倍数でない場合、最終位置でフィル値の一部のみが使われ、警告が出力されます。

ベースライン、ミッドレンジ、全ての 24 ビットデバイスでは、-addressing オプションに値 2 を指定します。その他全てのデバイスについては、既定値のアドレス値を使うか、アドレス値 1 を指定します。

フィル値はワード境界に整列されているため、フィル幅の倍数のアドレスから開始されます。フィル値が命令オペコードである場合、この整列によって命令が正しく実行されるようになります。同様に、フィルシーケンスの総長が 1 を上回る場合(指定された幅が 1 であっても)、フィルシーケンスはその総長に合わせて整列されます。例えば、以下の fill オプションは、2 バイトのフィルシーケンスと 2 の倍数でない開始アドレスを指定します。

```
-fill=w1:0x11,0x22@0x1100:0x1100c
```

結果の HEX レコードは以下のようになります。この整列により開始アドレスがフィルシーケンスの 2 バイト目で埋められています。

```
:0C100100221122112211221122112211B1
```

これを開始アドレスが 2 の倍数に指定されている-fill=w1:0x11,0x22@0x1100**0**:0x1100c オプションの場合と比較します。

```
:0D10000011221122112211221122112211A0
```

全てのフィル定数(幅の指定を除く)は、通常の C 言語の構文に従って(符号なしの)2 進数、8 進数、10 進数、16 進数で表現できます。例えば、1234 は 10 進値、0xFF00 は 16 進値、FF00 は不正です。

3.13 Find

-find=opcode オプションはオペコードまたはコードシーケンスの発生を検出し、ログに記録するために使われます。このオプションは以下のように使います。

```
-find=Findcode[mMask]@Start-End[/Align][w][t”Title”]
```

- Findcode は検索する 16 進数のコードシーケンスです。例えば、オペコードが 0x01F1 の clrf 命令を検索す

るには、シーケンスとして 01F1 を使います。これは HEX ファイルでは F1 01 というバイトシーケンスとして現れます。この時、HEX アドレス 0 が 0xF1、HEX アドレス 1 が 0x01 となります。

- Mask は必須ではありません。より限定的でない検索を可能にするために Findcode 値に適用するビットマスクを指定します。リトル エンディアン バイトオーダで入力されます。
- Start と End は検索するアドレスレンジを限定します。
- Align は指定しなくてもかまいません。コードシーケンスがこの値の倍数のアドレスから始まる場合にのみ一致するよう指定します。
- w はコードシーケンスが検出されるたびに Hexmate が警告を発するようにします(指定した場合)。
- Title は指定しなくてもかまいません。このコードシーケンスにタイトルを付けられるようになります。タイトルを定義すると、ログレポートやメッセージがより具体的になり、読みやすくなります。タイトルは実際の検索結果に影響を与えません。

全ての数値引数は 16 進数とみなされます。

例を以下に示します。

オプション-find=1234@0-7FFF/2w は 0h～7FFFh の間の 2 バイトのアドレス境界に整列されている時、コードシーケンス 1234h（HEX ファイルに 34 12 として格納）を検出します。w はこのシーケンスが見つかるたびに警告を発する事を示します。

この次の例-find=1234M0F00@0-7FFF/2wt”ADDXY”では、オプションは前の例と同じですが、マッチするコードシーケンスが 000Fh でマスキングされているため、Hexmate はオペコード 123xh (x は任意の数)を検索します。バイトマスクを使う場合、適用先のオペコードと同じバイト幅にする必要があります。Hexmate が生成するメッセージまたはレポートでは、この検索で定義されたタイトルである名前 ADDXY によってこのオペコードを表します。

要求された場合、全ての検索結果がログファイルに記録されます。-find オプションは HEX データの 1～8 バイト長の全バイトを受け入れます。オプションで、-find を replace または delete(別途説明)と組み合わせて使う事ができます。

3.14 検索して削除

-find オプションの delete 形式を使うと、一致したシーケンスが出力ファイルから削除されます。これは、データをゼロバイトで置き換えるのではなく、完全に削除するという意味です。

-find オプションで削除を実行するには、オプションの後に,delete を付加します。以下に例を示します。

```
-find=ff@7fe0-7fff,delete
```

この機能は細心の注意を払って使う必要があり、通常は実行可能コードの削除には推奨されません。

3.15 検索して置換

-find オプションの replace 形式を使うと、一致したシーケンスが新しいコードに置換または部分的に置換されます。

-find オプションで置換を実行するには、以下に示すようにオプションの後に,replace=spec を付加します。

```
-find=spec,replace=Code[mMask]
```

- Code は-find 条件に一致するシーケンスを置き換える 16 進コードシーケンスです。
- Mask は一致したコードシーケンスを Code 内のどのビットで置き換えるかを指定するビットマスクで、必須ではありません。例えば、16 ビット命令のうちの 4 ビットのみを変更する必要がある場合に役立ちます。残りの 12 ビットをマスクして変更しないようにする事ができます。

以下に例を示します。

```
-find=ff@7fe0-7fff,replace=55
```

この機能は細心の注意を払って使う必要があります。

© 2023 Microchip Technology Inc. and its subsidiaries

3.16 Format オプション

-format=type[,length]は出力する INHX ファイルのフォーマットを指定します。このオプションの簡単なユースケースは HEX ファイルのフォーマットの変更です(INHX16 から INHX32 フォーマットへの変更等)。HEX ファイルマージ後、データに値を挿入または調整した後の出力ファイルタイプの指定にも使えます。また、このオプションでは出力ファイルの最大レコード長を調整する事もできます。

type 引数には、下表に示すように生成する名前付きの INHX フォーマット(INHX16 等)を指定します。ファイルのフォーマットは HEX ファイル内のレコードタイプによってのみ決定される事に注意してください。例えば、HEX ファイルにレコードタイプ 0 とレコードタイプ 1 のみが存在する場合、そのファイルは INHX8M フォーマットに準拠します。タイプ 4 またはタイプ 5 のレコードも存在する場合、そのファイルは INHX32 フォーマットに準拠します。HEX ファイルにタイプ 2 またはタイプ 3 およびタイプ 4 またはタイプ 5 のレコードが存在する場合、ファイルはどの名前付きフォーマットにも準拠しません。このようなファイルを Hexmate に渡すと、このオプションで指定された出力フォーマットで許可されないレコードは出力される HEX ファイルから削除されます。入力ファイルにタイプ 2、タイプ 3、タイプ 4、またはタイプ 5 のレコードが存在する場合、INHX8 ファイルに変換する事はできません。

length は 1 データレコードあたりの最大バイト数を設定する引数で、必須ではありません。有効な長さは 1〜16 の 10 進数で、既定値は 16 です。

このオプションでサポートされる利用可能なフォーマットを下表に示します。INHX032 の選択は実際の名前付き INHX フォーマットではない事に注意してください。このフォーマットを指定した場合、INHX32 ファイルが生成されますが、拡張リニアアドレス (タイプ 4)レコードを使って上位アドレス情報がゼロに初期化されます。これを要件とするデバイス プログラマも存在します。

表 3-3. HEX ファイル フォーマット

フォーマット	有効なレコードタイプ	内容
INHX8M	0, 1	16 ビット幅のアドレス フィールド
INHX16	0, 1, 2, 3	20 ビット幅のアドレス フィールド
INHX32	0, 1, 4, 5	32 ビット幅のアドレス フィールド
INHX032	0, 1, 4, 5	INHX32、かつ上位アドレスを 0 に初期化

3.17 Help

-help を使うと、全ての Hexmate オプションの一覧が表示されます。-help のパラメータとして他の Hexmate オプションを入力すると、指定したオプションの詳細なヘルプメッセージが表示されます。以下に例を示します。

```
-help=string
```

これは-string Hexmate オプションについて追加のヘルプ情報を表示します。

3.18 Logfile

-logfile オプションは HEX ファイルの統計情報を指定したファイルに保存します。以下に例を示します。

```
-logfile=output.hxl
```

これは Hexmate が生成中の HEX ファイルを解析し、レポートを output.hxl という名前のファイルに保存します。

3.19 Mask

-mask=spec オプションを使うと、特定のビットマスクを使ってメモリレンジの論理積(AND)を取ります。プログラムワードに未実装ビットがある場合、空白にするために使われます。このオプションの使い方は以下の通りです。

© 2023 Microchip Technology Inc. and its subsidiaries

```
-mask=hexcode@start-end
```

ここで、hexcode は start〜end のアドレスレンジ内のデータと AND を取る値です。全ての値は 16 進数とみなします。マルチバイトのマスク値は、リトル エンディアン バイトオーダで入力できます。

3.20 O: 出力ファイルの指定

-ofile オプションを使うと、指定したファイル内に生成された Intel HEX 出力が作成されます。以下に例を示します。

```
-oprogram.hex
```

これは結果の出力を program.hex に保存します。出力ファイルは入力ファイルと同じ名前でもかまいませんが、その場合、出力ファイルによって入力ファイルが完全に置き換えられます。

ファイル名を指定せずにこのオプションを使った場合、何も出力されません。これは Hexmate を使って HEX ファイルのサイズのみを表示する場合等に役立ちます。このオプション自体を使わない場合、出力 HEX ファイルの内容は標準出力ストリームに出力されます。

3.21 Serial

-serial=specs オプションは、特定の HEX 値シーケンスを固定アドレスに格納します。このオプションは以下のように使います。

```
-serial=Code[+/-Increment]@Address[+/-Interval][rRepetitions]
```

- Code は格納する 16 進数のシーケンスです。指定された最初のバイトが最下位アドレスに格納されます。
- Increment は反復のたびに Code の値を指定した値分だけ変化させます(要求された場合)。指定は必須ではありません。
- Address はこのコード(またはその最初の反復)を格納する場所です。
- Interval はこのコードの反復ごとのアドレスシフトを指定するもので、必須ではありません。
- Repetitions はこのコードを何回反復するかを指定するもので、必須ではありません。

Repetitions 引数の既定値は 10 進数ですが、それ以外の全ての数値引数は 16 進数が想定されます。

以下に例を示します。

```
-serial=000001@EFFE
```

これは HEX コード 0x00001 をアドレス 0xEFFE に格納します。

以下に別の例を示します。

```
-serial=0000+2@1000+10r5
```

これはアドレス 0x1000 で値 0000 から始まる 5 つのコードを格納します。以降のコードはアドレス間隔+0x10 で現れ、コード値は+0x2 ずつ変化します。

3.22 Size オプション

-size オプションを使うと、結果の HEX イメージ内のデータ (レコード ペイロード)の総バイト数が標準出力に報告されます。ログファイルが要求されている場合、このサイズはログにも記録されます。以下に例を示します。

```
> hexmate -ooutput.hex float.hex -size
Total data: 988 (3DCh) bytes
> hexmate -ooutput.hex float.hex -string@9000="More data" -size
Total data: 998 (3E6h) bytes
```

3.23 String

-string オプションは ASCII 文字列を固定アドレスに埋め込みます。このオプションは以下のように使います。

```
-string@Address[tCode]=”Text”
```

- Address は、文字列を格納するアドレスを表す 16 進値とします。
- Code は文字列の各バイトに後続するバイトシーケンスを可能にするもので、必須ではありません。これにより、命令内に文字列のバイトをエンコードできます。
- Text は ASCII を変換して埋め込む文字列です。

以下に例を示します。

```
-string@1000=”My favorite string”
```

これはアドレス 0x1000 に「My favorite string」という文字列の ASCII データ（NULL 終端文字を含む)を格納します。

別の例:

```
-string@1000t34=”My favorite string”
```

これは同じ文字列を格納し、文字列の各バイトに HEX コード 0x34 を後続させます。

3.24 Strpack

-strpack=spec オプションは-string と同じ機能を実行しますが、重要な違いが 2 つあります。

-string が各文字に対応する全バイトを格納するのに対し、-strpack は各文字の下位 7 ビットのみを格納します。7 ビットの文字のペアを連結し、別々のバイトとしてではなく 14 ビットのワードとして格納します。これを文字列パッキングと呼びます。これは、プログラムメモリが 14 ビットワードでアドレス指定されているミッドレンジの PIC デバイスで役に立ちます。このオプションを使う場合、エンコードされた文字が実行時に完全に読み出し可能で、正しく解釈される事を確認する必要があります。

これら 2 つのオプションの 2 つ目の違いは、-string で使える t 指定子が-strpack オプションでは使えない点です。

3.25 W: 警告レベルの指定

-wlevel オプションは警告レベルのしきい値を設定します。level 値には、例えば-w5 のように-9～9 の数字を指定できます。

警告レベルは Hexmate が不審な要求やファイルの内容に対してどの程度厳密に対応するかを決定します。各警告には警告レベルが設定されており、警告レベルが高いほど警告メッセージの重要度が上がります。警告メッセージの警告レベルがこのオプションで設定されたしきい値を超えると、警告が出力されます。既定値の警告レベルしきい値は 0 です。この時、通常の警告メッセージが全て有効になります。

© 2023 Microchip Technology Inc. and its subsidiaries

4. 内部動作

Hexmate は処理中の HEX ファイルに対して各種操作を実行できます。以下の操作は、実行される順番に記載されています。順番は出力に影響を与えるため、順番を認識する事は重要です。

- 入力された全ての HEX ファイルに含まれるデータをコマンドラインに表示される順番で、各ファイルに関連付けられた読み出し指定に従って処理し、内部マスターイメージを作成します。
- `-find` オプションを使って指定した値をマスターイメージから検索し、要求に応じて、オプション発行時とは逆順でこれらの値を削除および置換します。
- `-serial` オプションを使って指定したシリアルデータを、オプション発行時とは逆順でマスターイメージに挿入します。
- `-string` オプションと`-strpack` オプションを使って指定した文字列を、オプション発行時とは逆順でマスターイメージに埋め込みます。
- `-ck` オプションを使って要求されたハッシュと後続コード(`-ck` の `t` 引数)のための空間をマスターイメージ内に予約します。
- `-fill` オプションを使って指定したマスターイメージ内の未使用の位置を、オプションの発行順に埋めます。
- 指定された各ハッシュをマスターイメージから計算し、これらのハッシュと後続コードをオプションの発行順にマスターイメージに挿入します。
- 最終的なマスターイメージを Intel HEX ファイル フォーマットにエンコードし、`-o` オプションで指示されたファイルに出力します。

`-mask` オプションで指定されたデータのマスキングは、入力 HEX ファイルからのデータの書き込みと Hexmate 操作のマスターイメージへの保存のどちらであっても、マスターイメージが変更されるたびに行われます。

4.1 障害の考えられる原因

ここでは、実行の失敗や実行時の障害につながる可能性があるプロジェクト コードやコンパイラ オプションの設定ミス、Hexmate オプションの誤った使用のケースを示します。

データ上書きエラー、`(944) data conflict at address *h between * and *`のよくある原因は以下の通りです。

- ブートローダ/アプリケーション プロジェクトを記述する時、アプリケーション プロジェクトとブートローダ プロジェクトで異なるコンフィグレーション ワードを指定した場合。通常、1 つのプロジェクトで 1 回だけ指定します。
- ブートローダ/アプリケーション プロジェクトを記述する時、2 つのプロジェクト間でコンパイル時のプログラムメモリが分割されていなかった場合。コンパイラ オプションを使い、あるプロジェクトで使われるメモリが他のプロジェクトで使われないようにします。
- ブートローダ/アプリケーション プロジェクトを記述する時、あるプロジェクトにおいて(`__at()` または `address` 属性を使って)別のプロジェクトで使われているプログラムメモリのレンジ内にある絶対アドレスで関数が定義された場合。別のアドレスを選択する、または絶対関数を使わないようにします。
- XC32 ブートローダ/アプリケーション プロジェクトを記述する時、デバッグ例外ハンドラがリンカスクリプトから破棄されなかった場合。以下に例を示します。

```
SECTION
{
  /DISCARD/ : { *(._debug_exception) }
}
```

- 実行可能コードまたは他のデータによって使われる場所にハッシュ、シリアル番号、または文字列が埋め込まれている場合。別の場所を選択します。
- Hexmate が生成したハッシュ値が、ハッシュ自体の計算に使われるメモリ領域に格納された場合。例えば、アドレス 0～0xFF のデータに対して計算されたハッシュが位置 0x7E に格納される等。ハッシュを生成するために使う値のレンジを変更する、またはハッシュの結果を新しい場所に移動させます。

© 2023 Microchip Technology Inc. and its subsidiaries

Hexmate が生成したハッシュ値が実行時に計算されるハッシュ値と一致しない場合の一般的な理由を以下に挙げます。

- 複数ワード内のデータのバイトオーダが、例えば MSb 先頭と LSb 先頭等、異なるオーダで処理されている場合。コードのアルゴリズムを変更する、または-CK Hexmate オプションで `revWidth` 値を指定します。
- ハッシュ計算に含まれる未使用のメモリ領域が既知の値が埋められていなかった場合。お使いのコンパイラまたは Hexmate の fill オプションを使うか、ハッシュの計算に使うアドレスのレンジを調整します。

多数の Hexmate オプションの障害に共通する原因は以下の通りです。

- 誤ったアドレス指定値を使った事により、指定したアドレスが正しく解釈されなかった場合。8 ビット AVR デバイス、ベースライン、ミッドレンジ PIC デバイス、24 ビット PIC デバイスを使う場合、`-addressing` Hexmate オプションに値 2 を指定します。その他のデバイスでは、既定値の 1 で問題ありません。

© 2023 Microchip Technology Inc.
and its subsidiaries

5. ハッシュ値の計算

ハッシュ値とは、任意のサイズのデータブロックに含まれる全ての値から計算され、その値を表現するために使われる小さな固定サイズの値です。そのデータブロックをコピーした場合、新しいブロックから再計算したハッシュを元のハッシュと比較できます。2 つのハッシュが一致すれば、そのコピーが正当であるという確証が高いレベルで得られます。ハッシュ アルゴリズムには多くの種類があります。アルゴリズムが複雑であるほどより信頼性の高い検証が可能ですが、組み込み環境、特に小型のデバイスで使う場合には計算負荷が高過ぎる場合があります。

Hexmate を使って HEX ファイルに含まれるプログラム イメージのハッシュを計算できます。このハッシュを同じ HEX ファイルに埋め込み、プログラム イメージと合わせてターゲット デバイスに書き込む事が可能です。実行時に、ターゲット デバイスのメモリに保存されているプログラム イメージに対して同様のハッシュ アルゴリズムを実行できます。格納されているハッシュと計算されたハッシュが一致すれば、組み込みプログラムは正当なプログラム イメージを実行可能だとみなせます。

Hexmate にはチェックサムや巡回冗長検査等の複数のハッシュ アルゴリズムが実装されており、アルゴリズムを選択してハッシュ値を計算できます。

Hexmate を明示的に実行する場合、3.11 Ck Hexmate オプションで説明した通り、`-ck` オプションでハッシュの計算を要求します。

未使用のメモリ位置を含むメモリに対してハッシュ値を計算する場合はいくらか注意が必要です。Hexmate を明示的に実行する場合、`-fill` オプション(3.12 Fill オプション参照)を使ってこれらの位置に既知の値をプログラムする事を検討してください。

Hexmate は、ファイルを作成したコンパイラの種類とそのファイルでプログラムするデバイスの種類を問わず、あらゆる Intel HEX ファイルからハッシュ値を生成できます。ただし、ターゲット デバイスのアーキテクチャによっては、実行時に読み込み可能なメモリ位置が制限される場合があります。その場合、2 つのハッシュが同じように計算されて一致するように、Hexmate でハッシュ計算を行う方法を変更する必要があります。また、コンパイラによっては、デバイスメモリに存在しないパディングまたはファントムバイトを HEX ファイルに挿入する事があります。これらのバイトは、Hexmate がハッシュ値を計算する際に無視する必要があるかもしれません。以下に考えられる解決策を述べます。

全てのデバイスがプログラムメモリの全幅を読み出せるとは限りません。例えば、ベースラインとミッドレンジの PIC デバイスでは各プログラムメモリ位置の下位バイトしか読み出せません。これに対し、HEX ファイルにはプログラムメモリの各ワードに対して 2 バイトが含まれるため、通常、この 2 バイトは Hexmate がハッシュ値を計算する時に処理されます。Hexmate を明示的に実行する場合、-ck オプションの `s2` サブオプションを使って各 2 バイトワードの MSb をスキップします。ただし、このような検証プロセスでは各プログラムワードの MSb の破損が考慮されていない事に注意してください。

MPLAB XC16 C コンパイラでは、dsPIC および PIC24 デバイスの 24 ビット(3 バイト) プログラムメモリのワードサイズに対応するため、各 3 バイト命令の後に 0x00 ファントムバイトを挿入して 4 バイトワードを構成しています。Hexmate は通常、ハッシュ値を計算する際にこれらのファントムバイトを見て処理しますが、デバイス上で実行される同じ計算を行うコードはこれを処理できない可能性があります。Hexmate を明示的に実行する場合、`-ck` オプションの `s4` サブオプションを使い、これらのデバイスで各 4 バイトワードの MSb をスキップさせます。

デバイスによっては、CRC ハッシュ値を計算できるハードウェア CRC モジュールを持つものもあります。必要に応じて、スキャナ モジュールを使ってプログラムメモリのデータをこのモジュールにストリーミングし、計算を自動化できます。スキャナ モジュールは、各プログラム メモリワードの MSb を最初に読み込むため、Hexmate でも命令ワード内の HEX ファイルバイトを逆順で処理する必要があります。Hexmate を明示的に実行する場合、`-ck` オプションの `r2` サブオプションを使い、Hexmate が 2 バイトワード内のバイトを逆順に処理するようにします。

また、HEX ファイルにエンコードされた Hexmate のハッシュ値を実行時に読み出す方法についても考慮が必要です。

ベースラインとミッドレンジの PIC デバイスでは、`retlw` 命令を使ってプログラムメモリにデータを格納する必要があります。そのため、Hexmate が計算したハッシュ値の各バイトを格納するのに 1 つの命令が必要になります。Hexmate を明示的に実行する場合、`-ck` オプションの `t34` サブオプションを使い、Hexmate プロセスが `retlw` 命令の一部として各バイトを保存するようにします。

dsPIC および PIC24 デバイスは、プログラムメモリの全幅を読み出す事ができないため、そのメモリに読み込むデータは通常、HEX ファイルデータの 4 バイトごとに 2 バイトのチャンクで格納されます。Hexmate を明示的に実

© 2023 Microchip Technology Inc. and its subsidiaries

行する場合、-ck オプションの t0000.2 サブオプションを使い、Hexmate が HEX ファイルの 4 バイトの下位に 2 バイトのハッシュ値を格納し、上位バイトは 0 に設定するようにします。

5.1 ハッシュ アルゴリズム

ここでは、Hexmate で使っており、実行時に対応するハッシュ値を計算できるアルゴリズムの例を示します。なお、以下の例は対象とするデバイスや状況によって変更が必要な場合があります。

5.1.1 加算アルゴリズム

Hexmate はプログラム イメージのあるレンジ内のデータ値の総和を計算するシンプルなチェックサム アルゴリズムを複数備えています。これらのアルゴリズムはアルゴリズム サブオプションのセレクタ値 1、2、3、4 に対応しており、それぞれ、プログラム イメージのデータを 1 バイト、2 バイト、3 バイト、4 バイトのサイズとして読み出します。この総和は、同じオプションでアルゴリズムに提供される初期値(オフセット)に加算されます。最終的なチェックサムを切り詰める幅もこのオプションで指定され、1 バイト、2 バイト、3 バイト、4 バイトのいずれかを選択できます。Hexmate は、チェックサム オプションで指定されたアドレスの HEX ファイルに自動的にチェックサムを格納します。

以下に示す関数は、データサイズ(read_t)とチェックサム幅(result_t)の任意の組み合わせで動作するようカスタマイズできます。

```
#include <stdint.h>
typedef uint8_t read_t;        // size of data values read and summed
typedef uint16_t result_t;     // size of checksum result

// add to offset, n additions of values starting at address data,
// truncating and returning the result
// data: the address of the first value to sum
// n:    the number of sums to perform
// offset: the intial value to which the sum is added
result_t ck_add(const read_t *data, unsigned n, result_t offset)
{
    result_t chksum;
    chksum = offset;
    while(n--) {
        chksum += *data;
        data++;
    }
    return chksum;
}
```

read_t の型定義はデータ読み出し/総和の幅に合わせて、result_t の型定義はチェックサム結果の幅に合わせて調整する必要があります。オフセットを使わない場合、そのパラメータを削除し、ループの前で chksum に 0 を代入できます。

この関数の使い方について例を挙げて説明します。例えば、オフセット 0x20 から始まるアドレスレンジ 0x100～0x7fd に対して 1 バイト幅の値を加算して、2 バイト幅のチェックサムを計算するとします。チェックサムはリトルエンディアン形式で 0x7fe と 0x7ff に格納されるものとします。

Hexmate を明示的に実行する場合、以下のオプションを使います。

```
-ck=100-7fd@7fe+20g1w-2
```

以下の MPLAB XC8 コードスニペットを PIC デバイスに合わせて調整します。これは ck_add() を呼び出し、実行時のチェックサムを Hexmate によってコンパイル時に格納されたチェックサムと比較します。

```
extern const read_t ck_range[0x6fe/sizeof(read_t)] _at(0x100);
extern const result_t hexmate _at(0x7fe);
result_t result;

result = ck_add(ck_range, sizeof(ck_range)/sizeof(read_t), 0x20);
if(result != hexmate)
    ck_failure(); // take appropriate action
```

このコードはプレースホルダ配列 ck_range を使ってチェックサムの計算を行うメモリを表現します。変数

hexmate は Hexmate がチェックサム結果を格納する位置にマップされます。extern で絶対的なため、これらのオブジェクトはどちらもデバイスメモリ消費を増加させません。これらのオブジェクトのアドレスやサイズは、Hexmate に渡すオプションに合わせて調整します。

Hexmate はどのようなアドレスレンジでもチェックサムを計算できますが、テスト関数 ck_add()は積算されるレンジの開始アドレスと終了アドレスが read_t 幅の倍数であると仮定します。これは read_t のサイズが 1 であれば問題ありません。チェックサム仕様はこの仮定に従う事が推奨されます。そうでない場合、開始データ値/終了データ値の部分読み出しを実行するようにテストコードを修正する必要があります。その場合、コードが大幅に複雑化します。

5.1.2 減算アルゴリズム

Hexmate はプログラム イメージのあるレンジ内でデータ値を減算するチェックサム アルゴリズムを複数備えています。これらのアルゴリズムはアルゴリズム サブオプションのセレクタ値-1、-2、-3、-4 に対応しており、それぞれ、プログラム イメージのデータを 1 バイト、2 バイト、3 バイト、4 バイトのサイズとして読み出します。それ以外は、減算アルゴリズムは加算アルゴリズムと全く同じです。減算アルゴリズムの詳細は 5.1.1 加算アルゴリズムを参照してください。

以下に示す関数は、データサイズ(read_t)とチェックサム幅(result_t)の任意の組み合わせで動作するようカスタマイズできます。

```
#include <stdint.h>
typedef uint8_t read_t;        // size of data values read and subtracted
typedef uint16_t result_t;     // size of checksum result

// add to offset n subtractions of values starting at address data,
// truncating and returning the result
// data: the address of the first value to subtract
// n:    the number of subtractions to perform
// offset: the intial value to which the subtraction is added
result_t ck_sub(const read_t *data, unsigned n, result_t offset)
{
    result_t chksum;
    chksum = offset;
    while(n--) {
        chksum -= *data;
        data++;
    }
    return chksum;
}
```

この関数の使い方について例を挙げて説明します。例えば、オフセット 0x0 から始まるアドレスレンジ 0x0～0x7fd に対して 2 バイト幅の値を加算して、4 バイト幅のチェックサムを計算するとします。チェックサムはリトル エンディアン形式で 0x7fe と 0x7ff に格納されるものとします。

Hexmate を明示的に実行する場合、以下のオプションを使います。

```
-ck=0-7fd@7fe+0g-2w-4
```

以下の MPLAB XC8 コードスニペットを PIC デバイスに合わせて調整します。これは ck_sub()を呼び出し、実行時のチェックサムを Hexmate によってコンパイル時に格納されたチェックサムと比較します。

```
extern const read_t ck_range[0x7fe/sizeof(read_t)] _at(0x0);
extern const result_t hexmate _at(0x7fe);
result_t result;

result = ck_sub(ck_range, sizeof(ck_range)/sizeof(read_t), 0x0);
if(result != hexmate)
    ck_failure(); // take appropriate action
```

5.1.3 フレッチャー アルゴリズム

Hexmate にはフレッチャーのチェックサムを実装するアルゴリズムが複数あります。これらのアルゴリズムはより複雑で、巡回冗長検査に近い高信頼性を少ない計算量で提供します。このアルゴリズムにはアルゴリズム サブオプションのセレクタ値 7 と 8 に対応する 2 つの形式があります。セレクタ値 7 のアルゴリズムは 1 バイトの計算と 2 バイトの結果を実装し、セレクタ値 8 のアルゴリズムは 2 バイトの計算と 4 バイトの結果を実装します。Hexmate

© 2023 Microchip Technology Inc. and its subsidiaries

は、チェックサム オプションで指定されたアドレスの HEX ファイルに自動的にチェックサムを格納します。

以下に示す関数は 1 バイト幅の加算を行って 2 バイトの結果を生成します。

```
#include <stdint.h>
typedef uint16_t result_t;     // size of fletcher result

result_t
fletcher8(const unsigned char * data, unsigned int n )
{
    result_t sum = 0xff, sumB = 0xff;
    unsigned char tlen;
    while (n) {
        tlen = n > 20 ? 20 : n;
        n -= tlen;
        do {
            sumB += sum += *data++;
        } while (--tlen);
        sum = (sum & 0xff) + (sum >> 8);
        sumB = (sumB & 0xff) + (sumB >> 8);
    }
    sum = (sum & 0xff) + (sum >> 8);
    sumB = (sumB & 0xff) + (sumB >> 8);
    return sumB << 8 | sum;
}
```

この関数の使い方について例を挙げて説明します。例えば、オフセット 0x20 から始まるアドレスレンジ 0x100～0x7fb に対して 2 バイト幅のフレッチャー ハッシュを計算するとします。チェックサムはリトル エンディアン形式で 0x7fc～0x7ff に格納されるものとします。

Hexmate を明示的に実行する場合、以下のオプションを使います。`w` サブオプションで幅を制御する事はできませんが、このサブオプションの引数の符号を使って結果に求められるエンディアンを示します。

```
-ck=100-7bd@7fc+20g8w-4
```

このコードは、加算アルゴリズムに示したのと同様の方法で呼び出せます(5.1.1 加算アルゴリズム参照)。

4 バイトの結果を生成する 2 バイト加算フレッチャー アルゴリズムのコードを以下に示します。

```
#include <stdint.h>
typedef uint32_t result_t;     // size of fletcher result

result_t
fletcher16(const unsigned int * data, unsigned n)
{
    result_t sum = 0xffff, sumB = 0xffff;
    unsigned tlen;
    while (n) {
        tlen = n > 359 ? 359 : n;
        n -= tlen;
        do {
            sumB += sum += *data++;
        } while (--tlen);
        sum = (sum & 0xffff) + (sum >> 16);
        sumB = (sumB & 0xffff) + (sumB >> 16);
    }
    sum = (sum & 0xffff) + (sum >> 16);
    sumB = (sumB & 0xffff) + (sumB >> 16);
    return sumB << 16 | sum;
}
```

5.1.4 CRC アルゴリズム

Hexmate は信頼性の高い CRC(巡回冗長検査)を実装する複数のアルゴリズムを備えています。アルゴリズム サブオプションのセレクタ値 5 と-5 に対応する 2 つのアルゴリズムから選択でき、セレクタ値 5 のアルゴリズムは CRC 計算、セレクタ値-5 のアルゴリズムは反転 CRC 計算を実装します。反転アルゴリズムはデータの最下位ビットを最初に処理します。

使う多項式と初期値をオプションで指定できます。Hexmate は、チェックサム オプションで指定されたアドレスの HEX ファイルに自動的に CRC 結果を格納します。

デバイスによっては、CRC を実行時に計算するのに使う事のできる CRC モジュールがハードウェアに実装されて

いるものもあります。これらのモジュールは、スキャナ モジュールを使ってプログラムメモリから読み出したデータをストリーミングできます。Hexmate と CRC/スキャナ モジュールが処理するバイトの順番を同じにするには、逆順ワード幅を 2 に指定する必要があります。この場合、HEX ファイルの 2 バイトの各ワードは順番に読み出されますが、各ワード内のバイトは逆順に処理されます。Hexmate を明示的に実行する場合、`-ck` の `r2` サブオプションを使います。

以下に示す関数は、任意の結果の幅(`result_t`)で動作するようカスタマイズできます。POLYNOMIAL マクロで指定された多項式を使って CRC ハッシュ値を計算します。

```
#include <stdint.h>
typedef uint16_t result_t;     // size of CRC result
#define POLYNOMIAL     0x1021
#define WIDTH   (8 * sizeof(result_t))
#define MSb     ((result_t)1 << (WIDTH - 1))

result_t
crc(const unsigned char * data, unsigned n, result_t remainder)
    { unsigned pos;
    unsigned char bitp;
    for (pos = 0; pos != n; pos++) {
        remainder ^= ((result_t)data[pos] << (WIDTH - 8));
        for (bitp = 8; bitp > 0; bitp--) {
            if (remainder & MSb) {
                remainder = (remainder << 1) ^ POLYNOMIAL;
            } else {
                remainder <<= 1;
            }
        }
    }
    return remainder;
}
```

`result_t` の型定義は結果の幅に合わせて調整する必要があります。

この関数の使い方について例を挙げて説明します。例えば、初期値 0xFFFF から始まるアドレスレンジ 0x0～0xFF に対して 2 バイト幅の CRC ハッシュ値を計算するとします。結果はリトル エンディアン形式で 0x100 と 0x101 に格納されるものとします。

Hexmate を明示的に実行する場合、以下のオプションを使います。

```
-ck=0-ff@100+ffffg5w-2p1021
```

以下の MPLAB XC8 コードスニペットを PIC デバイスに合わせて調整します。これは `crc()` を呼び出し、実行時のハッシュ結果を Hexmate によってコンパイル時に格納されたチェックサムと比較します。

```
extern const unsigned char ck_range[0x100] _at(0x0);
extern const result_t hexmate _at(0x100);
result_t   result;

result = crc(ck_range, sizeof(ck_range), 0xFFFF);
if(result != hexmate){
    // something's not right, take appropriate action
    ck_failure();
}
// data verifies okay, continue with the program
```

反転 CRC 結果は入力データと最終結果を反転させる、または多項式を反転させる事によって計算できます。以下に示す関数は、任意の結果の幅(`result_t`)で動作するようカスタマイズできます。関数 `crc_reflected_IO()` は、データストリームのビット位置を反転させる事によって反転 CRC ハッシュ値を計算します。また、関数 `crc_reflected_poly()` はデータストリームを調整する代わりに多項式(どちらの関数でも `POLYNOMIAL` マクロで指定される)を反転させます。どちらの関数も関数 `reflect()` を使ってビット反転を実行します。

```
#include <stdint.h>
typedef uint16_t result_t;     // size of CRC result
typedef unsigned char read_t;
typedef unsigned int reflectWidth;
// This is the polynomial used by the CRC-16 algorithm we are using.
#define POLYNOMIAL     0x1021
#define WIDTH   (8 * sizeof(result_t))
#define MSb     ((result_t)1 << (WIDTH - 1))
#define LSb     (1)
#define REFLECT_DATA(X)       ((read_t) reflect((X), 8))
#define REFLECT_REMAINDER(X) (reflect((X), WIDTH))

reflectWidth
reflect(reflectWidth data, unsigned char nBits)
{
    reflectWidth reflection = 0;
    reflectWidth reflectMask = (reflectWidth)1 << nBits - 1;
    unsigned char bitp;
    for (bitp = 0; bitp != nBits; bitp++)
        { if (data & 0x01) {
            reflection |= reflectMask;
        }
        data >>= 1;
        reflectMask >>= 1;
    }
    return reflection;
}

result_t
crc_reflected_IO(const unsigned char * data, unsigned n, result_t remainder) {
     unsigned pos;
    unsigned char reflected;
    unsigned char bitp;
    for (pos = 0; pos != n; pos++)
        { reflected =
        REFLECT_DATA(data[pos]);
        remainder ^= ((result_t)reflected << (WIDTH - 8));
        for (bitp = 8; bitp > 0; bitp--) {
            if (remainder & MSb) {
                remainder = (remainder << 1) ^ POLYNOMIAL;
            } else {
                remainder <<= 1;
            }
        }
    }
    remainder = REFLECT_REMAINDER(remainder);
    return remainder;
}

result_t
crc_reflected_poly(const unsigned char * data, unsigned n, result_t remainder) {
     unsigned pos;
    unsigned char bitp;
    result_t rpoly;
    rpoly = reflect(POLYNOMIAL, WIDTH);
    for (pos = 0; pos != n; pos++) {
        remainder ^= data[pos];
        for (bitp = 8; bitp > 0; bitp--)
            { if (remainder & LSb) {
                remainder = (remainder >> 1) ^ rpoly;
            } else {
                remainder >>= 1;
            }
        }
    }
    return remainder;
}
```

この関数の使い方について例を挙げて説明します。例えば、初期値 0xFFFF から始まるアドレスレンジ 0x0～0xFF に対して 2 バイト幅の反転 CRC ハッシュ結果を計算するとします。結果はリトル エンディアン形式で 0x100 と 0x101 に格納されるものとします。

Hexmate を明示的に実行する場合、代わりに以下のオプションを使います。選択されたアルゴリズムが-5 である事に注意してください。

```
-ck=0-ff@100+ffffg-5w-2p1021
```

上記の通り、プロジェクトで関数 `crc_reflected_IO()` または `crc_reflected_poly()` を呼び出します。

5.1.5 SHA アルゴリズム

Hexmate は SHA(セキュアハッシュ アルゴリズム)を実装しています。セレクタ値 10 は SHA-2 アルゴリズムの 256 ビット版である SHA256 アルゴリズムを選択します。

SHA256 を実装するコードは Hexmate がサポートする他のアルゴリズムよりも複雑で、名前からも分かるようにハッシュの結果は 256 ビット(32 バイト)幅となります。github.com/B-Con/crypto-algorithms 等のサードパーティ ウェブサイトから、このアルゴリズムのパブリック ドメイン実装をダウンロードできます。

Hexmate の使用例を以下に示します。例えば、アドレスレンジ 0x0～0x1FF に対して SHA256 ハッシュ値を計算するとします。結果はリトル エンディアン形式で開始アドレス 0x1000 に格納されるものとします。

```
-ck=0-1ff@1000g10w-256
```

© 2023 Microchip Technology Inc.
and its subsidiaries

6. 改訂履歴

リビジョン A (2020 年 9 月)

- 本書の初版です。『MPLAB® XC8 C コンパイラ ユーザガイド』(DS 50002053)を基に一部編集。

リビジョン B (2022 年 12 月)

- INHX32 ファイルで使われる`--ssa` オプションの説明を追加。
- メッセージ記述ファイルで使われる新しいファイル名フォーマットを記載。
- よく行われる操作について考えられる障害の原因について説明する新しいセクションを追加。
- INHX16 ファイルで使われる`--sla` オプションの動作の説明を更新。
- `-ck` オプションの動作の説明を明確化し、新しい XOR サブオプションの説明を追加。
- `-fill` オプションの説明を改善。
- `-format` オプションの説明を改善。

© 2023 Microchip Technology Inc.
and its subsidiaries

Microchip 社ウェブサイト

Microchip 社はウェブサイト(www.microchip.com)を通してオンライン サポートを提供しています。当ウェブサイトでは、お客様に役立つ情報やファイルを提供しています。以下を含む各種の情報をご覧になれます。

- **製品サポート** - データシートとエラッタ、アプリケーション ノートとサンプル プログラム、設計リソース、ユーザガイドとハードウェア サポート文書、最新のソフトウェアと過去のソフトウェア
- **技術サポート** - FAQ(よく寄せられる質問)、技術サポートのご依頼、オンライン ディスカッション グループ、Microchip 社のデザイン パートナー プログラムおよびメンバーリスト
- **ご注文とお問い合わせ** - 製品セレクタと注文ガイド、最新プレスリリース、セミナー/イベントの一覧、お問い合わせ先(営業所/正規代理店)の一覧

製品変更通知サービス

Microchip 社の製品変更通知サービスは、お客様に Microchip 社製品の最新情報をお届けする配信サービスです。ご興味のある製品ファミリまたは開発ツールに関する変更、更新、リビジョン、エラッタ情報をいち早くメールにてお知らせします。

http://www.microchip.com/pcn にアクセスし、登録手続きをしてください。

お客様サポート

Microchip 社製品をお使いのお客様は、以下のチャンネルからサポートをご利用頂けます。

- 正規代理店
- 技術サポート

サポートは正規代理店にお問い合わせください。本書の最後のページに各国の営業所の一覧を記載しています。

技術サポートは以下のウェブページからもご利用頂けます。
www.microchip.com/support

Microchip 社のデバイスコード保護機能

Microchip 社製品のコード保護機能について以下の点にご注意ください。

- Microchip社製品は、該当するMicrochip 社データシートに記載の仕様を満たしています。
- Microchip社では、通常の条件ならびに動作仕様書の仕様に従って使った場合、Microchip 社製品のセキュリティレベルは、現在市場に流通している同種製品の中でも最も高度であると考えています。
- Microchip社はその知的財産権を重視し、積極的に保護しています。Microchip 社製品のコード保護機能の侵害は固く禁じられており、デジタル ミレニアム著作権法に違反します。
- Microchip社を含む全ての半導体メーカーで、自社のコードのセキュリティを完全に保証できる企業はありません。コード保護機能とは、Microchip 社が製品を「解読不能」として保証するものではありません。コード保護機能は常に進化しています。Microchip 社では、常に製品のコード保護機能の改善に取り組んでいます。

法律上の注意点

本書および本書に記載されている情報は、Microchip 社製品を設計、テスト、お客様のアプリケーションと統合する目的を含め、Microchip 社製品に対してのみ使う事ができます。それ以外の方法でこの情報を使う事はこれらの条項に違反します。デバイス アプリケーションの情報は、ユーザの便宜のためにのみ提供されるものであり、更新によって変更となる事があります。お客様のアプリケーションが仕様を満たす事を保証する責任は、お客様にあります。その他のサポートはMicrochip 社正規代理店にお問い合わせ頂くか、https://www.microchip.com/en-us/support/design-help/client-support-servicesをご覧ください。

Microchip 社は本書の情報を「現状のまま」で提供しています。Microchip 社は明示的、暗黙的、書面、口頭、法定のいずれであるかを問わず、本書に記載されている情報に関して、非侵害性、商品性、特定目的への適合性の暗黙的保証、または状態、品質、性能に関する保証をはじめとするいかなる類の表明も保証も行いません。

いかなる場合もMicrochip 社は、本情報またはその使用に関連する間接的、特殊的、懲罰的、偶発的または必然的損失、損害、費用、経費のいかんにかかわらず、またMicrochip 社がそのような損害が生じる可能性について報告を受けていた場合あるいは損害が予測可能であった場合でも、一切の責任を負いません。法律で認められる最大限の範囲を適用しようとも、本情報またはその使用に関連する一切の申し立てに対するMicrochip 社の責任限度額は、使用者が当該情報に関連してMicrochip 社に直接支払った額を超えません。

Microchip 社の明示的な書面による承認なしに、生命維持装置あるいは生命安全用途にMicrochip社の製品を使う事は全て購入者のリスクとし、また購入者はこれによって発生したあらゆる損害、クレーム、訴訟、費用に関して、Microchip 社は擁護され、免責され、損害をうけない事に同意するものとします。特に明記しない場合、暗黙的あるいは明示的を問わず、Microchip社が知的財産権を保有しているライセンスは一切譲渡されません。

商標

Microchip 社の名称とロゴ、Microchip ロゴ、Adaptec、AVR、AVRロゴ、AVR Freaks、BesTime、BitCloud、CryptoMemory、CryptoRF、dsPIC、flexPWR、HELDO、IGLOO、JukeBlox、KeeLoq、Kleer、LANCheck、LinkMD、maXStylus、maXTouch、MediaLB、megaAVR、Microsemi、Microsemi ロゴ、MOST、MOST ロゴ、MPLAB、OptoLyzer、PIC、picoPower、PICSTART、PIC32 ロゴ、PolarFire、Prochip Designer、QTouch、SAM-BA、SenGenuity、SpyNIC、SST、SST ロゴ、SuperFlash、Symmetricom、SyncServer、Tachyon、TimeSource、tinyAVR、UNI/O、Vectron、XMEGA は米国とその他の国におけるMicrochip TechnologyIncorporated の登録商標です。

AgileSwitch、APT、ClockWorks、The Embedded Control SolutionsCompany、EtherSynch、Flashtec、Hyper Speed Control、HyperLightLoad、Libero、motorBench、mTouch、Powermite 3、Precision Edge、ProASIC、ProASIC Plus、ProASIC Plus ロゴ、Quiet-Wire、SmartFusion、SyncWorld、Temux、TimeCesium、TimeHub、TimePictra、TimeProvider、TrueTime、ZL は米国におけるMicrochip Technology Incorporated の登録商標です。

Adjacent Key Suppression、AKS、Analog-for-the-Digital Age、Any Capacitor、AnyIn、AnyOut、Augmented Switching、BlueSky、BodyCom、Clockstudio、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、Espresso T1S、EtherGREEN、GridTime、IdealBridge、In-Circuit Serial Programming、ICSP、INICnet、Intelligent Paralleling、IntelliMOS、Inter-Chip Connectivity、JitterBlocker、Knob-on-Display、KoD、maxCrypto、maxView、memBrain、Mindi、MiWi、MPASM、MPF、MPLAB Certified ロゴ、MPLIB、MPLINK、MultiTRAK、NetDetach、Omniscient Code Generation、PICDEM、PICDEM.net、PICkit、PICtail、PowerSmart、PureSilicon、QMatrix、REAL ICE、RippleBlocker、RTAX、RTG4、SAM-ICE、Serial Quad I/O、simpleMAP、SimpliPHY、SmartBuffer、SmartHLS、SMART-I.S.、storClad、SQI、SuperSwitcher、SuperSwitcher II、Switchtec、SynchroPHY、TotalEndurance、Trusted Time、TSHARC、USBCheck、VariSense、VectorBlox、VeriPHY、ViewSpan、WiperLock、XpressConnect、ZENAは米国とその他の国におけるMicrochip Technology Incorporated の商標です。

SQTP は米国におけるMicrochip Technology Incorporated のサービスマークです。

Adaptec ロゴ、Frequency on Demand、Silicon Storage Technology、Symmcom はその他の国におけるMicrochip Technology Incorporatedの登録商標です。

GestIC は、その他の国におけるMicrochip Technology Germany II GmbH & Co. KG (Microchip Technology Incorporated の子会社) の登録商標です。

その他の商標は各社に帰属します。

© 2023, Microchip Technology Incorporated and its subsidiaries.

All Rights Reserved.

ISBN: 978-1-6683-1787-7

品質管理システム

Microchip社の品質管理システムについてはwww.microchip.com/qualityをご覧ください。

各国の営業所とサービス

南北アメリカ	アジア/太平洋	アジア/太平洋	欧州
本社 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 技術サポート: http://www.microchip.com/support URL: www.microchip.com	**オーストラリア - シドニー** Tel: 61-2-9868-6733	**インド - バンガロール** Tel: 91-80-3090-4444	**オーストリア - ヴェルス** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393
アトランタ Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455	**中国 - 北京** Tel: 86-10 -8569-7000	**インド - ニューデリー** Tel: 91-11-4160-8631	**デンマーク - コペンハーゲン** Tel: 45-4485-5910 Fax: 45-4485-2829
オースティン、TX Tel: 512-257-3370	**中国 - 成都** Tel: 86-28-8665-5511	**インド - プネ** Tel: 91-20-4121-0141	**フィンランド - エスポー** Tel: 358-9-4520-820
ボストン Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088	**中国 - 重慶** Tel: 86-23-8980-9588	**日本 - 大阪** Tel: 81-6-6152-7160	**フランス - パリ** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79
シカゴ Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075	**中国 - 東莞** Tel: 86-769-8702-9880	**日本 - 東京** Tel: 81-3-6880-3770	**ドイツ - ガーヒンク** Tel: 49-8931-9700
ダラス Addison, TX Tel: 972-818-7423 Fax: 972-818-2924	**中国 - 広州** Tel: 86-20-8755-8029	**韓国 - 大邱** Tel: 82-53-744-4301	**ドイツ - ハーン** Tel: 49-2129-3766400
デトロイト Novi, MI Tel: 248-848-4000	**中国 - 杭州** Tel: 86-571-8792-8115	**韓国 - ソウル** Tel: 82-2-554-7200	**ドイツ - ハイルブロン** Tel: 49-7131-72400
ヒューストン、TX Tel: 281-894-5983	**中国 - 香港SAR** Tel: 852-2943-5100	**マレーシア - クアラルンプール** Tel: 60-3-7651-7906	**ドイツ - カールスルーエ** Tel: 49-721-625370
インディアナポリス Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380	**中国 - 南京** Tel: 86-25-8473-2460	**マレーシア - ペナン** Tel: 60-4-227-8870	**ドイツ - ミュンヘン** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44
ロサンゼルス Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800	**中国 - 青島** Tel: 86-532-8502-7355	**フィリピン - マニラ** Tel: 63-2-634-9065	**ドイツ - ローゼンハイム** Tel: 49-8031-354-560
ローリー、NC Tel: 919-844-7510	**中国 - 上海** Tel: 86-21-3326-8000	**シンガポール** Tel: 65-6334-8870	**イスラエル - ラーナナ** Tel: 972-9-744-7705
ニューヨーク、NY Tel: 631-435-6000	**中国 - 瀋陽** Tel: 86-24-2334-2829	**台湾 - 新竹** Tel: 886-3-577-8366	**イタリア - ミラノ** Tel: 39-0331-742611 Fax: 39-0331-466781
サンノゼ、CA Tel: 408-735-9110 Tel: 408-436-4270	**中国 - 深圳** Tel: 86-755-8864-2200	**台湾 - 高雄** Tel: 886-7-213-7830	**イタリア - パドヴァ** Tel: 39-049-7625286
カナダ - トロント Tel: 905-695-1980 Fax: 905-695-2078	**中国 - 蘇州** Tel: 86-186-6233-1526	**台湾 - 台北** Tel: 886-2-2508-8600	**オランダ - ドリューネン** Tel: 31-416-690399 Fax: 31-416-690340
	中国 - 武漢 Tel: 86-27-5980-5300	**タイ - バンコク** Tel: 66-2-694-1351	**ノルウェー - トロンハイム** Tel: 47-7288-4388
	中国 - 西安 Tel: 86-29-8833-7252	**ベトナム - ホーチミン** Tel: 84-28-5448-2100	**ポーランド - ワルシャワ** Tel: 48-22-3325737
	中国 - 厦門 Tel: 86-592-2388138		**ルーマニア - ブカレスト** Tel: 40-21-407-87-50
	中国 - 珠海 Tel: 86-756-3210040		**スペイン - マドリッド** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91
			スウェーデン - ヨーテボリ Tel: 46-31-704-60-40
			スウェーデン - ストックホルム Tel: 46-8-5090-4654
			イギリス - ウォーキンガム Tel: 44-118-921-5800 Fax: 44-118-921-5820