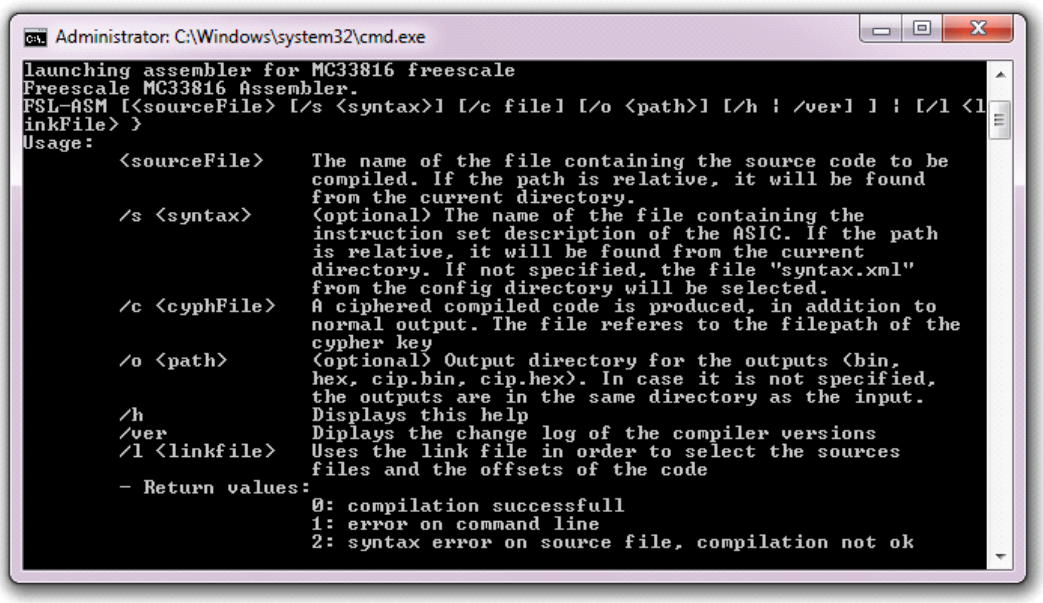
Document Number: MC33816ASM_APPSPUG
Rev. 1.0, 1/2014

MC33816 Assembler User Guide

An Introduction to Programming the MC33816 Device



Contents

1 Introduction

The MC33816 Assembler is designed to compile MC33816 assembler code into its binary equivalent for the KIT33816AEEVM evaluation board or any other hardware based on the MC33816. The tool is provided as a command line application.

2 Jump Start

The MC33816 Assembler (in addition to related software and documentation) can be downloaded from the Freescale website by following the steps listed below:

- Go to www.freescale.com/analogtools
- Under the column with the heading “Kit Number”, click on the KIT33816AEEVM entry.
- The link opens a Tool Summary Page for the evaluation board that features the MC33816 device.
- Look on that page for the following heading

Jump Start Your Design

- Clicking on the link opens a pop-up window that lists downloads for the MC33816 device. The MC33816 Assembler download has the name MC33816ASM_APPSP.zip.

3 Contents of MC33816 Assembler Download

The MC33816 assembler zip file contains the following files:

- MC33816ASM.bat (the executable file)
- delivery (folder)
  - bin (folder)
    - Assembler_exec.jar
  - cipher (folder)
    - key4.key
  - config (folder)
    - log4j.properties
    - syntax.xml
    - syntax.xsd
  - verif (folder)
    - (empty - for future usage)
- MC33816 Assembler Release Notes

4 System Functionality

4.1 Prerequisites

The MC33816 Assembler has been evaluated on the Windows 7 32-bit operating system only. The assembler is based on Java. It should run smoothly on any Java-compatible platform. The minimum requisite for Java is version 1.6 or later. Please download and install Java from the following link: http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html.

The assembler has most recently been validated using Java 1.6u35. It is highly recommended to use this version of Java. The assembler tests for the presence of Java and issues an error if Java is not detected.

4.2 Installation

The package is delivered as a zip file. Installation is performed by unzipping the file into an empty directory.

4.2.1 Directories

The standard directories created by the installation are:

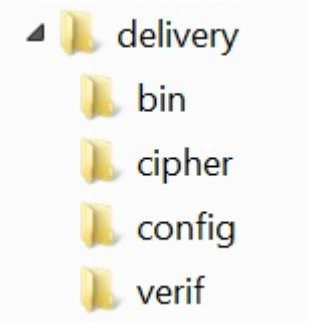


Figure 1. Assembler Directory Hierarchy

- The *delivery* directory groups all system directories and holds the start command for the assembler (*MC33816ASM.bat*).
- The *bin* directory contains different executable programs. The user must not modify this directory.
- The *config* directory contains the configuration files. The user must not modify this directory.
- The *cipher* directory is provided to the end user to store the cipher files.
- The *verif* directory is empty. This folder is reserved for future usage.

4.3 Assembler Program

The assembler is a command line program. The program start command must be entered with the suitable options.

4.3.1 Help Menu

Open a command console within the *delivery* directory and type `MC33816asm` <enter> to see the options for that command.

Figure 2 shows the command line help listing, which can be obtained by running the program either without options or with the `/h` option.

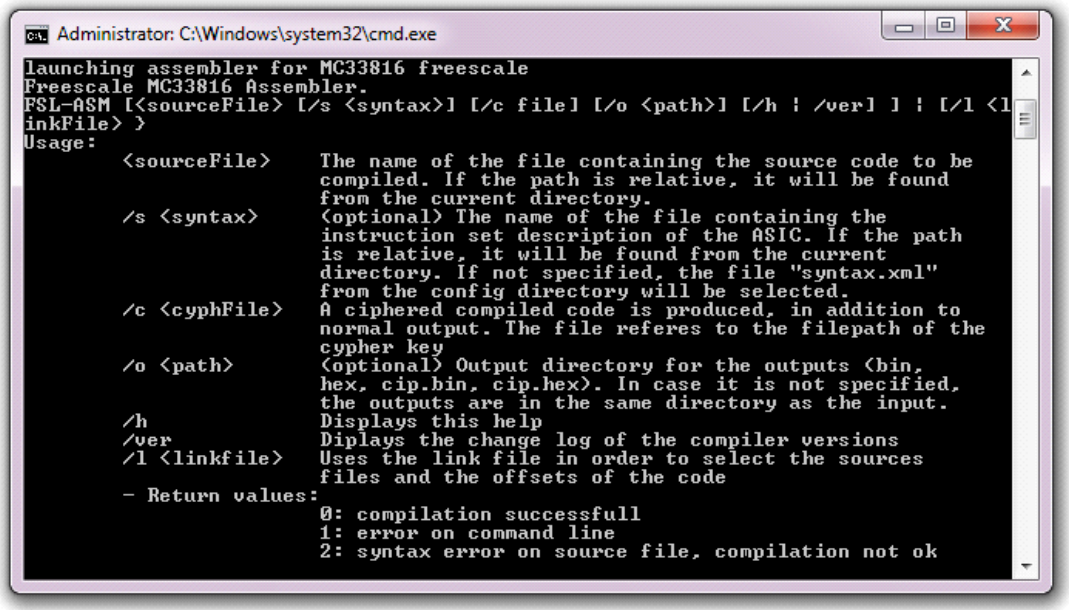


Figure 2. MC33816 Assembler Help Menu

The command `MC33816asm` is followed by the required options and source file names.

The options <*`sourceFile`*>, `/h`, and `/ver` must be first in the command line in order for them to be taken into account. Other options may follow in any order.

The <*`sourceFile`*> and `/l` options may only be used together if the link file contains 0 source files; otherwise, they are mutually exclusive. This combination is allowed in order to restrict the maximum number of instructions that can be assembled.

4.4 Input Files

4.4.1 Input Files Extension

In order to perform source code compilation, the following files listed in Table 1 must be provided to the assembler as options.

Table 1. List of MC33816 Assembler Input Files and Extensions

Extension	Description
`*.dfi (or *.psc)`	The microcode source file
`*.link`	The source files link file
`*.xml`	The instruction syntax library file
`*.key`	The cipher key file

4.4.2 Syntax File

A custom syntax file may be created based on the included default syntax.xml file. The custom syntax file must be valid with respect to the schema which is included in the *config* directory. Custom syntax files may be used through the use of the */s* option detailed in the command help.

4.4.3 Cipher Key File

The output binary data may be ciphered using a key file. The key file must have the following format:

```
NFSR=0123456789ABCDEF0123
LFSR=0123456789ABCDEF0123
```

Cipher files may be used through the use of the */c* option detailed in the command help.

4.4.4 Source Files Link Option

The assembler can perform code assembly using two modes:

- Single source code assembly
- Linked files assembly

In single source code assembly, the source code file is provided as an assembler <*sourceFile*> option. In the link file assembly mode, a link file is provided to the assembler by means of the */l <linkfile>* option. Link file assembly makes it possible to assemble several source code files to predefined memory locations. In this mode, the link file source code files and definition files must all be located in the same input folder. The link file format is as follows:

```
LinkFileName = 'link.link'
LinkFileVersion = 'LinkFileVersionNumber'
ConcatenateMaxLine='MaximumNumberOfConcatenatedLines'
SourceFileNumber = '<n>'
Source1File = 'Source1FileName'
Offset1 = 'OffsetSize'
Source2File = 'Source2FileName'
Offset2 = 'OffsetSize'
...
Source<n>File = 'Source2FileName'
Offset<n> = 'OffsetSize'
```

Below is a specific example:

```
LinkFileName = 'Concat_ch1_ch2.link'
LinkFileVersion = '1.0'
ConcatenateMaxLine= '1023'
SourceFileNumber = '2'
Source1File = 'MC33816_ch1.dfi'
Offset1 = '0'
Source2File = 'MC33816_ch2.dfi'
Offset2 = '100'
```

4.5 Output Files

4.5.1 Extension

Following successful compilation, the assembler produces the series of files described in Table 2.

Table 2. List of Assembler Output Files and Extensions

Extension	Description
`*.hex`	The hexadecimal result of the assembled file
`*.bin`	The binary result of the assembled file
`*.asm`	The complete assembler file corresponding to the expected Code RAM memory (processor memory). This pre-processed file based on the source code file (`*.dfi`) does not contain comments, labels, `define` or `include`.
`*.log`	The compilation report
`*.rep`	The CRC in bin and Hex format, all labels of the source code, code size
`*.cip.bin`	The encrypted binary file
`*.cip.hex`	The encrypted hexadecimal file

Note: If the output folder already contains one or more files with any of the extensions listed in the above table, the compilation is aborted. The output folder must be cleared before running the assembler.

5 Specific Assembler Language

The MC33816 requires a microcode to enable most of its functions. The main benefit is the large flexibility in setting the device. This microcode is defined by the software engineer in a source file, coding the 33816 specific instructions in assembler language.

The extension `*.dfi` or `*.psc` is generally used for the source file. Any other extension can be used as a source code extension, with the exception of assembler's input file extensions (`*. link`, `*.xml`, `*.key`) or output file extensions. (`*.cip`, `*.hex`, `*.bin`, `*.asm`, `*.log`, `*.reg`, `*.cip.bin`, `*.cip.hex`).

The assembler language coding rules are defined in the following sections.

5.1 Writing an Instruction

Instructions allow the software designer to define the behavior that is executed independently by each microcore.

The allowed instructions and parameters are only the ones defined in the default instructions library (syntax.xml) or the custom syntax files.

All the instructions must be followed by the mandatory parameters. All the instructions must terminate with the character ';'. The instruction syntax is as follows:

```
InstructionName Parameter1NameOrValue Parameter2NameOrValue...;
```

The instruction and parameter descriptions are provided in the MC33816 Data Sheet. The instructions and the associated parameters are case sensitive. They can only be placed:

- At the beginning of the line
- Or after the end-of-comment field character '*'
- Or after a valid Label

One instruction per source file line or per `include` line is allowed. Below is an example:

```
stf low ErrorFlag;
```

5.2 Inserting a Comment Field

The source code file supports the addition of comments. The comment fields are identified with:

- Two '*' characters, one placed before and the other after the comment text
- One '*' symbol placed before comment text. In this case, all characters up to the end of the line are considered as part of the comment.

The comment field syntax is as follows:

```
*Comment*
*Comment
```

Below are some examples:

```
*Comments*
*Comments
*Put the channel in error stat* SWInterruptRoutine: stf low ErrorFlag;
SWInterruptRoutine: stf low ErrorFlag; *Put the channel in error stat
```

5.3 Defining a Constant

The software designer can use a constant value label, instead of using a number as an instruction parameter (`define`). This constant definition helps to make the source code more readable. The `define` function is used for a constant value definition that is used locally in the source code. The constant definitions must terminate in the character '`;`'. The `define` syntax is as follows:

```
#define SymbolName SymbolValue;
```

The constant definitions are placed:

- At the beginning of the line
- Or after the final comment field character '*'

The constant definition must be placed in an instruction line. No other item, such as an instruction, label or `include` statement, is allowed in a constant definition line. All `define` statements must be unique, that is not already used as the *`SymbolName`* of a line label, and cannot have the same name as an instruction or a parameter. The `define` name (*`SymbolName`*) cannot start with a number but can contain one or several numbers. It must not include spaces. The `Define` arguments *`SymbolName`* and *`SymbolValue`* are mandatory. See the example below:

```
#define ErrorFlag 10;
```

5.4 Including a Data RAM Address Definition File

The assembler has the ability to manage a nested file structure. The sub files called in the main source file are known as definition files. These definition files are commonly used for variable definition dedicated to device Data RAM. However any instruction or label can be used in definition files.

All the `include` statements must terminate with the character '`;`'. A valid file name must be placed between two apostrophe characters (‘*`text`*’).

The `include` declaration must be placed:

- At the beginning of the line
- Or after the end comment character '*'

The `include` syntax is as follows:

```
#include ‘Filename.def’;
```

Use of nested *`include`* is not permitted.

Note that `*.def` or any other extension can be used as a definition file extension, with the exception of assembler’s input file extensions (`*.dfi`, `*. link`, `*.xml`) or output file extensions. (`*.cip`, `*.hex`, `*.bin`, `*.asm`, `*.log`, `*.reg`, `*.cip.bin`, `*.cip.hex`).

See the example below:

```
#include ‘Source1.def’; *include variable to the source.dfi defined in the
*definition file
```

5.5 Using a Line label

The assembler can manage line labels. This kind of label is used to replace a line number called as an instruction parameter. The assembler immediately replaces the label with the corresponding line number at the time of the source code assembly.

The label syntax is as follows:

```
LabelName:
```

The label refers to a line code where the label is set. For example, if the label *Init* is located on line 7 any instruction using this label refers to line 7.

Labels must be placed at the beginning of the line or after the final character of a comment field. These labels end with the symbol ':'. All labels must be followed by an instruction. They must be unique, must not already have been used as a *SymbolName* in a `Define` statement, and cannot be an instruction or parameter name. The *LabelName* can contain numbers, but cannot fit the number format and must not include spaces. An example follows:

```
SWInterruptRoutine:
```

5.6 Numbering Convention

A parameter can either be a parameter name associated with a value defined in the syntax file or it can be a numeric value. When a numeric value is used, the parameter is decimal. Three formats are possible:

- By default, the value is decimal (no suffix)
- A 'h' specifies that the value is hexadecimal
- A 'b' suffix specifies that the value is binary

An example follows:

```
ldirl 10 _rst;       * number 10
ldirl 10h _rst;      * number 16
ldirl 10b _rst;      * number 2
```

5.7 Conditional Assembly

The assembler includes a basic IF function (conditional assembly). This function is 'static' so the IF function branch is considered at the time of assembly. The IF function syntax is as follows:

```
#IF Condition
Instructions1
#ELSEIF
Instructions2
#ENDIF
```

The conditional assembly considers a branch only if its parameter is defined (whatever its value may be). Using an ELSE branch is optional. All the instructions placed before the #IF label and after the #ENDIF label are excluded from the conditional code block and are assembled. Only one level of condition assembly is supported, so an IF function cannot be nested within another IF function. Consider the example below:

```
#define DDI 1; *define optional in this case.
#IF DDI
jmpr DDI_Init; *DDI constant is defined so this condition is met
*In this case the program counter jumps to the DDI_Init label line
#ELSE
jmpr GDI_Init; *GDI constant is not defined so this condition is never met
#ENDIF
```

6 References

Following are URLs where you can obtain information on related Freescale products and application solutions:

Document Number and Description		URL
MC33816	Data Sheet	http://cache.freescale.com/files/analog/doc/data_sheet/MC33816.pdf
Freescale.com Support Pages		**URL**
MC33816 Product Summary Page		http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MC33816
KIT33816AEEVM Tool Summary Page		http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=KIT33816AEEVM
Analog Home Page		http://www.freescale.com/analog
Automotive Home Page		http://www.freescale.com/automotive

7 Revision History

Revision	Date	Description of Changes
1.0	1/2014	• Initial Release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: MC33816ASM_APPSPUG
Rev. 1.0
1/2014