Document ID: ETPUASMCVTR
Rev. 0.1, 08/2010

eTPU Assembly Converter

Overview

The eTPU Assembly Converter is a tool that converts Byte Craft eTPU assembly language code to Freescale eTPU assembly language code. The tool can be used either to convert instructions that are input manually or to convert entire .asm files, .c files, or directories of these files.

The tool converts only assembly language instructions (either in .asm files or in inline assembly sections of .c files).

With this tool, it is easy to convert files previously compiled using the Byte Craft compiler to the new, Freescale C compiler for the eTPU.

Contents

1 Command-Line Options

The following usage appears when running the converter with the `-h` switch. A detailed explanation appears in the sub sections below.

```
etpu_asm_converter [-options] (-c <file> | -a
<file> | -m | -mc)
```

Convert Modes — exactly one of these flags must appear:

- `-m`: manual mode, converts assembly language instructions from stdin.
- `-mc`: manual mode, converts C language text from stdin.
- `-c <c_file>  [out_file]`: converts all assembly language code in <*c_file*>.
- `-a <asm_file> [out_file]`: converts the assembly language file <*asm_file*>.

Options — if used, these flags must appear before the convert mode flag:

- `-nowarn`: suppress warnings
- `-h`: display usage message
- `-pp`: preprocess. Replaces macros in assembly language sections before performing conversion.

`-debugE`: debug errors. When this option is used, more information is printed if errors occur. Please note this option was created for development purposes — the information refers to lex token names and is not always helpful.

2 Convert Mode

The converter can run either in manual mode (`-m`, `-mc`) or in directory/file mode (`-c`, `-a`). In both modes, the converter can run in assembly mode (`-a`, `-m`) or c mode (`-c`, `-mc`).

2.1 Convert Mode — Language

2.1.1 Assembly Mode

In assembly mode (`-a`, `-m`), the entire file (or manually inserted input) will be treated as Byte Craft assembly language and converted immediately.

2.1.2 C Mode

In c mode (`-c`, `-mc`), the entire file (or manually inserted input) will be treated as a C language file. The file is copied line-by-line and only the inline assembly language instructions are converted. Inline assembly language instructions in Byte Craft begin with the `#asm` directive and end with `#endasm` directive. One line inline assembly instructions in Byte Craft can also begin with the `#asm` directive and an open parenthesis and end with a closing parenthesis.

2.2 Convert Mode — Input Type

2.2.1 Manual Mode

In manual mode (`-m`, `-mc`), the user inserts text manually to the standard input and the converter prints the converted text to the standard output. This mode is useful for checking specific instructions. Notice that the output is usually printed only after a full instruction has been analyzed (including the ending `'.'` character, which is essential in Byte Craft).

2.2.2 Directory/File Mode

When the converter runs in directory/file mode (`-a`, `-c`), a specific file or directory must be specified immediately after the `-c` or `-a` mode switch.

If the argument after the mode switch is a directory, the converter converts all the files in the specified directory that end with the `.c` extension (for mode switch `-c`) or the `.asm` extension (for mode switch `-a`). Each generated file will be named <*file_name*>`.converted.c` or <*file_name*>`.converted.asm` (depending on the current language mode).

If the argument after the mode switch is a file name (as opposed to a directory name), the specified file is converted. In this case, another argument may appear immediately after the file name that specifies the name to assign to the converted file. If this argument does not appear, the converted file is named <*file_name*>`.converted.c` or <*file_name*>`.converted.asm`, depending on the working language mode.

2.3 Examples

2.3.1 Manual Assembly Mode

```
etpu_asm_converter.exe -m

->alu c=b+a.
add c,b,a
->ram p = by_diob.
ld p,*diob
```

2.3.2 Manual C Mode

```
etpu_asm_converter.exe -mc

->/* All c information is copied line-by-line */
/* All c information is copied line-by-line */
->#asm (alu c=c+1.)
asm{ addi c,c,1 }
->callExampleCFunc();
callExampleCFunc();
->#asm
asm{
->ram p = (diob++).
ld p,*diob++
->alu c=b.
move c,b
->#endasm
}
->callExampleCFunc();
callExampleCFunc();
```

2.3.3 File Assembly Mode

```
etpu_asm_converter.exe -a asm_input.txt
```

Table 1. File Assemble Mode

asm_input.txt:	**asm_input.txt.converted.asm:**
chan write_mera; ram p -> (diob). alu a = a - p. alu p =<< mach + 0x0.	**erw1 ; st p,*diob** **sub a,a,p** **addi.shl p,mach,0x0**

2.3.4 File C Mode

```
etpu_asm_converter.exe -c c_input.c out.txt
```

Table 2. File C Mode

c_input.c	**out.txt**

Main() {
#asm
alu c = diob << 2.
alu c=17.
/* this is a note*/
#endasm
callFunc()
#asm (alu c = b + a.);
}
	main() {
	asm{
	shli c,diob,2
	movei c,0x11
	// this is a note
	}
	callFunc()
	asm{ add c,b,a };
	}

3 Converter options

3.1 2.2.1 No Warnings

When the `-noWarn` option is used, warnings are not reported. This option is useful when converting many files and only the conversion errors are important. For more about warnings, see the *Limitations* section.

3.2 Preprocessing

When the `-pp` option is used, the file is preprocessed before being converted. The preprocessing is executed using the ccetpu compiler. In the preprocessing stage, all the macros are analyzed and replaced with their values. Only after preprocessing the file completes does conversion start. This option is useful with C language files that use macros in their inline assembly language sections. The converter does not recognize these macros without preprocessing the file first; therefore the option is necessary in such cases.

NOTE

> Using the `-pp` option causes white space and new line modifications in the file. Spaces and new lines may be added or removed during the preprocessing stage, and macros are replaced with their values; however, the actual content of the file is not modified.
> When using this option, the input file is passed through the ccetpu C compiler. Therefore, this file must follow eTPU C language rules. For example, the file must end with the "`.c`" extension.

Example of the `-pp` option:

```
etpu_asm_converter.exe -pp -c c_input.c out.txt
```

Table 3. Preprocessing Options

c_input.c	def.h	out.txt

#include "def.h"
#asm
MY_ALU c = REG_B+ VALUE.
MY_CHAN pdcm= sm_dt.
#endasm
	#define MY_ALU alu
	#define REG_B b
	#define VALUE 3
	#define MY_CHAN chan
		#include "def.h"
		
		asm{
		addi c,b,3
		chmode.sm_dt
		
		}

3.3 Debug Information

When the `-debugE` option is used, debug information is printed in the case of errors. This information may be useful in finding Byte Craft syntax errors. Note that this option was created for development purposes; therefore, in many cases, the provided information may not be clear or helpful.

This option is mostly useful for the `chan` instruction. For example:

```
etpu_asm_converter.exe -m -debugE
chan set flag2.
error in line 3: syntax error, unexpected FREE_TEXT, expecting FLAG0 or FLAG1.
```

4 Converter Operation

The eTPU assembly converter is given a file (or manually input data) as input and generates a new file, with all assembly language instructions converted from Byte Craft to Freescale assembly language. If the original file compiled correctly according to the standard Byte Craft assembly language architecture, the asm converter should convert the file without error and produce a new, Freescale compile-ready file.

4.1 Context

The converter is a context-independent tool. It analyzes each assembly language instruction separately and converts it without reference to any other instructions. The only exception to this rule involves macros. The converter has the ability to preprocess C language files before converting them and therefore can recognize macros in instructions even if the macros were defined elsewhere in the file (or outside the file). More information about this option in available in the command-line section.

4.2 Error Handling

When the converter encounters an unrecognized instruction, it reports an error along with the line number of the problematic instruction. Since the converter is context independent, it can continue converting instructions immediately after the error. Therefore, when the converter finishes, it is only necessary to manually fix the reported error; there is no need to convert the whole file again.

In addition, during conversion, the converter may report warnings. In these cases, the converter is able to convert the specified instruction, but there a chance that the conversion was not perfect. See the *Limitations* section for more information.

4.3 Variables

In C language files, variables may appear in inline assembly language instructions. The converter does not recognize these variables (since it is context independent). In order to handle variables in inline assembly language, the converter copies any text that appears in assembly language operands where variables are an option. It is the user’s responsibility to verify that this text refers to a real variable. If no such variable exists, the compiler reports an error when it compiles the converted file.

4.4 Parallel Instructions

The eTPU assembly converter supports parallel instructions. The sub-instructions in parallel instructions are separated by semicolons in Byte Craft assembly language.

5 Limitations

This section lists possible limitations to the eTPU assembly converter.

5.1 Complex expressions

The converter does not analyze complex expressions (for example, alu `c=3+6-1+2`); instead, the converter copies them as is. (In the example above, the converted text is: `movei c,3+6-1+2`). This approach should deal correctly with most expressions, but some complex expressions may cause issues when converted.

5.2 Negative Numbers

The Byte Craft approach to negative numbers is not consistent: They are treated as 8-bit variables in some cases, as 24-bit variables in some cases, ignored in some cases, and not allowed in other cases. Therefore, the conversion of these numbers is also not completely consistent.

5.3 Macros

Macros in inline assembly language are supported, but the `-pp` switch must be used for them to be converted correctly. See the command-line section for more information.

5.4 Byte Craft Errors

In certain cases, the Byte Craft compiler does not create code in consistent fashion and according to its own documentation. In these cases, the eTPU assembly converter “fixes” Byte Craft's errors. This solution ensures that the converted Freescale assembly language instruction matches Byte Craft's original instruction; however, in contrast to most cases, the generated binary code of the original instruction and the converted instruction are different (because Byte Craft's instruction does not generate the expected code).

For example:

The instruction "`alu c =>> b+a+1.`" generates the binary code `0x3D330F95` when assembled by the Byte Craft compiler, even though the meaning of this code is to shift `b+a+1` left instead of right. (This seems like a Byte Craft error.) The eTPU assembly converter converts this instruction to "`add.shr.one c,b,a`". As a result, the Freescale and Byte Crafts instructions match — their syntax has the same meaning — but the code each instruction generates is different, since the converted instruction shifts `b+a+1` right, while Byte Craft's original instruction shifts them left.

5.5 Unsupported features

All Byte Craft directives besides `%hex` are not supported. Local Byte Craft labels are not supported either.

5.6 Parallel Instructions

The converter supports parallel instructions. However, there are certain rare parallel instructions that are supported in Byte Craft but not by the Freescale assembler. The Byte Craft compiler supports `jmp` and `end` sub-instructions as part of the same parallel instruction, but the Freescale compiler does not support such a combination since both `jmp` and `end` change the program flow. If the original code contains such a parallel instruction, it will be converted without error, but the assembler will report an error when the converted instruction is assembled.

5.7 Warnings

5.7.1 ld/ldm

When loading variables that are allocated on the channel, the `ldm` operator should be used, and when loading global variables, the `ld` operator should be used. The converter does not distinguish between these variables and therefore always uses the `ldm` operator when converting instructions that load C variables. In such cases, this warning is printed: "`Instruction has been converted by default to ldm but might be ld depending on the label's value`".

5.7.2 Hex Directives

The converter supports hex directives that begin with `%hex` in Byte Craft assembly language; they are converted to `.word` directives in Freescale assembly language. However, a warning is still printed when these directives appear because the converter does not analyze their meaning. Since there is usually a reason that the original instructions appears in `hex` instead of in Byte Craft assembly language, it is recommended to review the converted directive and possibly rewrite the instruction in Freescale assembly language.

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 2666 8080
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

ETPUASMCVTR
Rev. 0.1
08/2010