AN3349
Application note

eTPU assembly converter

Introduction

The eTPU assembly converter is a tool used to convert Byte Craft eTPU assembly code for SPC563Mxx and SPC564Axx devices to eTPU compiler assembly code. The tool can be used either to convert specific instructions manually or to convert entire .asm files, .c files or directories. The tool only converts assembly instructions (either in .asm files or in inline assembly sections in c files). With this tool it is possible to easily convert files which were previously compiled in Byte Craft to the new compiler for 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 the following flags must appear):

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

Options (if used, must appear before convert mode):

- `-nowarn`: suppress warnings
- `-h`: display usage message
- `-pp`: preprocess. Replaces macros in assembly sections before converting.
- `-debugE`: debug errors. When this option is used more information is printed if errors occur (please note this option was created for developing purposes, the information refers to lex token names and isn't 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 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 will be copied line by line and only the inline assembly instructions will be converted. Inline assembly 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 in order to check 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 right 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 which end with the `.c` extension (if the switch was `-c`) or the `.asm` extension (if the switch was `-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 (and not a directory) the specified file will be converted. In this case another argument may appear right after the file name to specify the name of the new generated converted file. If such an argument does not appear the new file will be called `<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 No warnings

When the `-noWarn` option is used warnings aren't reported. This option is useful when converting many file and only the conversion errors are important. For more details about warnings, see the *Chapter 5: Limitations*.

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, its conversion starts. This option is useful with c files that use macros in their inline assembly sections. The converter will not recognize the macros without preprocessing the file first and therefore the option is necessary in such cases.

Note: Using this option will cause white space and new line modifications in the file. Spaces and new lines might be added or removed during the preprocessing stage, and macros will be replaced with their values, but the actual content of the file will not be modified.
When using this option, the input file is passed through the ccetpu C compiler. Therefore, it has to follow eTPU c language rules, for example the file must end with a "`.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 case of errors. This information may be useful in order to find Byte Craft syntax errors. Note that this option was created for development purposes and therefore in many cases the provided information will probably not be clear or helpful.

This option is mostly useful in 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 manual data) as an input and produces a new file with all assembly instruction converted from Byte Craft to eTPU Compiler assembly. If the original file compiled correctly according to standard Byte Craft assembly architecture, the converted should convert the file without errors and create a new eTPU compile-ready file.

4.1 Context

The converter is a context independent tool. It analyzes each assembly instruction separately and converts it regardless of any other instructions. The only exception to that rule is macros. The converter has the ability to preprocess c files before converting them and therefore recognize macros in instruction even if they 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 converting, it is necessary only to manually fix the reported error and there is no need to convert the whole file again.

In addition during the conversion, the converter may report warnings. In these cases, the converter was able to convert the specified instruction but there a chance that the conversion wasn't perfect, more information is available at the *Chapter 5: Limitations*.

4.3 Variables

In c files, variables may appear in inline assembly instructions. The converter does not recognize these variables (since it is context independent). In order to allow variables in inline assembly, the converter copies any text that appears in assembly operands where variables are optional. It is the users' responsibility to verify that this text refers to a real variable. If no such variable exists the compiler will report an error if it attempts to compile 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.

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`), and instead copies them as is (in the above example 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 isn't consistent: They are treated as 8 bit variables or 24 bit variables in some cases, ignored in other cases and not allowed all together in others. Therefore, the conversion of these numbers is also not completely consistent.

5.3 Macros

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

5.4 Byte Craft errors

In certain specific cases the Byte Craft compiler doesn't create code in consistent fashion and according to its own documentation. In these cases the converter "fixes" Byte Craft's errors. This solution ensures that the converted eTPU compiler assembly instruction will match Byte Craft's original instruction, however in contrast to most cases the generated binary code of the original instruction and converted instruction will be different (because Byte Craft's instruction doesn't generate the expected code).

For example:

The instruction "`alu c =>> b+a+1`." generates the binary code `0x3D330F95` when assembled in 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 converter will convert the instruction to "`add.shr.one c,b,a`". As a result the eTPU compiler and Byte Crafts's instructions match (their syntax has the same meaning) but the code each instruction will generate is different (since the converted instruction shifts `b+a+1` right, and 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 as well.

5.6 Parallel instructions

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

5.7 Warnings

5.7.1 ld/ldm

When loading variables which 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 those variable and therefore always uses the `ldm` operator when converting instructions which load C variables. In such cases the following 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 which begin with `%hex` in Byte Craft, and are converted to `.word` directives in eTPU compiler assembly. However, a warning is still printed when these directives appear because the converter doesn't analyze their meaning. Since there is usually a reason that the original instructions appear in `hex` instead of Byte Craft assembly, it is recommended to review the converted directive and possibly rewrite the instruction in eTPU compiler assembly.

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
28-Apr-2011	1	Initial release.
18-Sep-2013	2	Updated Disclaimer.