Application Note

AN_342

FT90x Assembly Language Programming Guide

Version 1.1

Issue Date: 2016-09-19

This document provides a brief guide on how to develop firmware using FT32 Assembly language.

Future Technology Devices International Limited (FTDI)
Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
Web Site: http://ftdichip.com

Table of Contents

1 Introduction

In firmware development, it is almost impossible to avoid coding in assembly language completely. In this document, the fundamental assembly language coding using the FT32 assembler is provided.

AN_341 FT90x Technical Manual is available under NDA for the complete Programming Model.

2 Basic assembly syntax for the FT90x assembler

The most common way to develop software using assembly language is to draft source code according to the FT90x assembly language syntax, store it as text file, assemble it via the assembler to generate object code and link the object code with other modules to form the final executable firmware image.

One of the typical use cases for applying this development method is to draft the bootstrap code for the FT90x based SoC.

In this section, basic syntax for the FT90x assembly language is discussed.

2.1 Comments

Comments are very important for readability of assembly code, for FT90x assembly; there are two kinds of comments available, multi-line comments and single line comments.

2.1.1 Multi-line comments

If a long comment is required, such as a copyright declaration, a multi-line comment could be used. Multiline comments in the FT90x assembly language are the same as in C language:

```
/*
  The only way to include a newline ('\n') in a comment
  is to use this sort of comment.
*/

/* This sort of comment does not nest. */
```

Anything from /* through the next */ is a comment. This means you may not nest these comments

2.1.2 Single line comments

To add some simple one line comments, apply a "#" and the rest of the line is considered as a comment:

```
# Disable Global Interrupt here!
```

Or

```
    reti          # IRQ 3
```

The latter sample shows the mixing of instruction and one line comments.

2.2 Symbols

Symbol names begin with a letter or with ".", "_", or "$". That character may be followed by any string of digits, letters, dollar signs, and underscores. The case of letters is significant: e.g. foo is a different symbol name to Foo. Though the FT90x assembler supports symbol names starting with "$", using "$" as the starting letter for symbol name is not a recommended practice as registers also start with "$".

2.3 Constants

For better readability, it is better to use constants instead of hard-coded values. This could be by using the ".equ" directive as follows:

```
.equ PMBASE      , 0x1fc80
```

The .equ directive above will define the PMBASE symbol as 0x1fc80.

2.4 Labels

Labels are very important for FT90x assembly language. The labels define the offset which could be used for branch instructions or function call instructions etc.

2.4.1 Defining a label

To define a label state the symbol name or identifier and terminate with a colon ":" e.g.

```
label:
```

2.4.2 Global label

If a label needs to be accessed from other files, a global label is needed. In this case, apply the ".global" directive to define a global symbol first, as shown below:

```
.global _start
_start:
```

2.5 Statements

A statement ends at a newline character (`\n') or at a semicolon (;). The newline or semicolon is considered part of the preceding statement. Newlines and semicolons within character constants are an exception: they do not end statements.

It is an error to end any statement with end-of-file: the last character of any input file should be a newline. An empty statement is allowed, and may include whitespace. It is ignored.

A statement begins with zero or more labels, optionally followed by a key symbol which determines what kind of statement it is. The key symbol determines the syntax of the rest of the statement. If the symbol begins with a dot, then the statement is an assembler directive. If the symbol begins with a letter the statement is an assembly language instruction: it assembles into a machine language instruction. A label is a symbol immediately followed by a colon (:). Whitespace before a label or after a colon is permitted, but you may not have whitespace between a label's symbol and its colon.

2.6 Registers

The FT90x core has 32 registers, in the assembly code, each of the registers may be accessed by "$rn", where "n" is the index of the register, such as "$r1", "$r2", "$r18" etc.

3 Inline assembly

Quite often, using inline assembly embedded in C source code is much more convenient than drafting an independent assembly source code file and linking it together with other modules. For the FT90x, this is especially true for streaming instructions.

The format of basic inline assembly is:

```
asm("assembly code");
```

Example:

```
asm(“move.l $r18, $r19”);
__asm__(“move.l $r18, $r19”);
```

Both “asm” and “__asm__” are valid for FT90x if there is more than one instruction, write one per line in double quotes, and also suffix a ’\n’ and ’\t’ to the instruction. This is required as gcc sends each instruction as a string to the assembler (GAS) and by using the newline/tab, correctly formatted lines are sent to the assembler.

To make inline assembly really useful for development, variables must be associated with C code to the assembly code as operands. This is done using the following template:

```
asm (
    : output operands                   /* optional */
    : input operands                    /* optional */
    : list of clobbered registers       /* optional */
    );
```

The assembler template consists of assembly instructions. Each operand is described by an operand-constraint string followed by the C expression in parentheses. A colon separates the assembler template from the first output operand and another separates the last output operand from the first input, if any. Commas separate the operands within each group. The total number of operands is limited to ten or to the maximum number of operands in any instruction pattern in the machine description, whichever is greater.

If there are no output operands but there are input operands, place two consecutive colons surrounding the place where the output operands would go.

For a better description of the template above, refer to a real source code as below:

```
1 #include <stdio.h>
2
3 int main()
```

```
 4 {
 5   int i = 18, j = 30, k = 0;
 6
 7   __asm__("move.l $r18, $r19");
 8
 9   printf("i, j, k is %d, %d, %d  \n", i, j, k);
10
11   asm volatile
12   (
13           "add.l $r18, %2, 0x01;\n\t"
14           "move.l %0,$r18;      \n\t"
15           "add.l %1, %3, 5;     \n\t"
16           : "=r"(j), "=r"(k)
17           : "r"(i), "r"(k)
18           : "$r18"
19   );
20
21   printf("i, j, k is %d, %d, %d after execution \n", i, j, k);
22 }
```

There are three integer variables defined in the sample code, i, j and k with initial value as 18, 30 and 0 respectively.

Variable j is defined as an output operand which is represented in the inline assembly as %0, variable i is defined as an input operand which is represented in the inline assembly as %2, and variable k is defined as both an input and output operand and is represented in the inline assembly code as %1 and %3 respectively. “r” and “=r” are constraints which means these variables shall be stored in the General Purpose Registers.

As an example, Register r18 is defined as a clobber register which informs GCC that the user code will control and modify r18 in this code segment.

What this inline assembly code segment will do is: extract value of variable i and increase by 1 and store the result into register r18 (line 13); after that, put the value in the register r18 to variable j (line 14), and value of variable k will be used to add to a constant immediate with value 5 and store the result into variable k itself (line 15).

After execution of this code segment, the value of variables i, j, k will become 18, 19, 5 respectively.

4 Contact Information

Head Quarters – Singapore

Bridgetek Pte Ltd
178 Paya Lebar Road, #07-03
Singapore 409030
Tel: +65 6547 4827
Fax: +65 6841 6071

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Branch Office – Taipei, Taiwan

Bridgetek Pte Ltd, Taiwan Branch
2 Floor, No. 516, Sec. 1, Nei Hu Road, Nei Hu District
Taipei 114
Taiwan, R.O.C.
Tel: +886 (2) 8797 5691
Fax: +886 (2) 8751 9737

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Branch Office - Glasgow, United Kingdom

Bridgetek Pte. Ltd.
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales.emea@brtichip.com
E-mail (Support) support.emea@brtchip.com

Branch Office – Vietnam

Bridgetek VietNam Company Limited
Lutaco Tower Building, 5th Floor, 173A Nguyen Van Troi,
Ward 11, Phu Nhuan District,
Ho Chi Minh City, Vietnam
Tel : 08 38453222
Fax : 08 38455222

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Web Site

http://brtchip.com/

Distributor and Sales Representatives

Please visit the Sales Network page of the Bridgetek Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A – References

Document References

http://www.ftdichip.com/Products/ICs/FT90x.html

AN_341 FT90x Technical Manual (available under NDA. Contact FTDI for more information)

Acronyms and Abbreviations

Terms	Description
FT90x	FTDI Proprietary 32 bit Core
SoC	System on Chip
ASM	Assembly
GCC	GNU Compiler Collection
GAS	GNU Assembler
NDA	Non-Disclosure Agreement

Appendix B – Revision History

Document Title: AN_342 FT90x Assembly Language Programming Guide

Document Reference No.: BRT_000035

Clearance No.: BRT#040

Product Page: http://brtchip.com/product

Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	2015-10-06
1.1	Added NDA info to AN_341 references Updated contact details Dual branding to reflect the migration of the product to the Bridgetek name – logo changed, copyright changed, contact information changed	2016-09-19