Micromega Corporation

uM-FPU V3.1 Support Software

MPLAB C Compiler for dsPIC DSC or PIC24

Introduction

This document describes the support software for using the uM-FPU V3.1 floating point coprocessor with Microchip dsPIC DSC or PIC24 microcontrollers and the MPLAB C Compiler. For a full description of the uM-FPU V3.1 chip, please refer to the *uM-FPU V3.1 Datasheet* and *uM-FPU V3.1 Instruction Reference*. Application notes are also available on the Micromega website.

uM-FPU V3.1 Support Software

Several files are provided for interfacing the uM-FPU V3.1 chip with the MPLAB C Compiler.

fpu_spi.c
This file contains all of the low-level support routines for interfacing with the uM-FPU V3.1 chip using the master SPI serial interface on the dsPIC DSC or PIC24 microcontroller. Descriptions of all of the C callable routines are provided below.

fpu_print.c
This file contains various print utility routines for sending data to stdout. Descriptions of all of the C callable routines are provided below.

fpu.h
This header file includes *fops.h* to define all of the uM-FPU V3.1 opcodes, matrix operations and FFT operations. It also defines the FPU status bits and provides function prototypes for all of the C callable routines in *fpu_spi.c* and *fpu_print.c*. It should be included in any C program that calls the FPU support routines.

fops.h
This header file is included by *fpu.h* and defines symbols for all uM-FPU V3.1 opcodes, matrix operations and FFT operations.

template.c
This file provides an example of the initializing the FPU, and can be used as a starting point for new programs.

Various sample programs are also provided with the support software.

Configuring the Support Software

The following definitions are located in the *fpu_spi.c* file. They must be modified as required to match the target configuration. The SPI_PPRE and SPI_SPRE symbols define the primary and secondary pre-scale value for the SPI clock frequency. They should be set so that the maximum SPI clock frequency doesn't exceed 5 MHz.

```
//-------------------- SPI definitions ---------------------------------------
// *** change these definitions to match target configuration ***

#define FCY 29480000UL                  // clock frequency: 29.48 MHz
#include <libpic30.h>

#define SPISTAT     SPI1STATbits        // SPI status register
#define SPICON      SPI1CONbits         // SPI control register
#define SPIBUF      SPI1BUF             // SPI data buffer

#define SPI_SCLK    PORTBbits.RB6       // SPI SCLK pin
#define SPI_SDI     PORTBbits.RB5       // SPI SDI pin
#define SPI_SDO     PORTBbits.RB4       // SPI SDO pin

#define SPI_PPRE    3                   // SPI clock primary pre-scale
#define SPI_SPRE    2                   // SPI clock secondary pre-scale
```

Function Descriptions for fpu_spi.c

fpu_reset

```
unsigned char fpu_reset(void);
```

To ensure that the microcontroller and the FPU are synchronized, a reset call must be done at the start of every program. The `fpu_reset` routine resets the FPU, confirms communications, and returns the sync character (0x5C) if the reset is successful. A sample reset call is included in the *template.c* file.

fpu_wait

```
void fpu_wait(void);
```

The FPU must have completed all instructions in the instruction buffer, and be ready to return data, before sending an instruction to read data from the FPU. The `fpu_wait` routine checks the ready status of the FPU and waits until it is ready. The print routines check the ready status, so calling `fpu_wait` before calling a print routine isn't required, but if your program reads directly from the FPU using one of the `fpu_write` functions, a call to `fpu_wait` is required prior to sending the read instruction. An example of reading a byte value is as follows:

```
fpu_wait();
fpu_write(LREADBYTE);
dataByte = fpu_readByte();
```

Description:

- wait for the FPU to be ready
- send the LREADBYTE instruction
- wait for the read setup delay
- read a byte value and store it in the variable dataByte

The uM-FPU V3.1 chip has a 256 byte instruction buffer. In most cases, data will be read back before 256 bytes have been sent to the FPU. If a long calculation is done that requires more than 256 bytes to be sent to the FPU, an `Fpu_Wait` call should be made at least every 256 bytes to ensure that the instruction buffer doesn't overflow.

fpu_write

```
void fpu_write(unsigned char bval1);
void fpu_write2(unsigned char bval1, unsigned char bval2);
void fpu_write3(unsigned char bval1, unsigned char bval2, unsigned char bval3);
void fpu_write4(unsigned char bval1, unsigned char bval2,
                unsigned char bval3, unsigned char bval4);
```

These routines are used to send instructions and data to the FPU. Each parameter specifies an 8-bit value to be sent to the FPU.

fpu_writeWord

```
void fpu_writeWord(int wval);
```

This routine is used to send a 16-bit value to the FPU.

fpu_writeLong

```
void fpu_writeLong(long lval);
```

This routine is used to send a 32-bit long integer value to the FPU.

fpu_writeFloat

```
void fpu_writeFloat(float fval);
```

This routine is used to send a 32-bit floating point value to the FPU

fpu_writeString

```
void fpu_writeString(char *s);
```

This routine is used to write a zero-terminated string to the FPU.

fpu_read

```
char fpu_read(void);
```

This routine reads an 8-bit value from the FPU with no initial read delay. This routine is used by the support routines and is not normally called directly by the user program. The initial read delay is not included. User programs would normally use the `fpu_readByte` function.

fpu_readByte

```
char fpu_read(void);
```

This routine reads an 8-bit value from the FPU. The initial read delay is included.

fpu_readWord

```
int fpu_readWord(void);
```

This routine reads an 16-bit value from the FPU. The initial read delay is included.

fpu_readLong

```
long fpu_readLong(void);
```

This routine reads an 32-bit long integer value from the FPU. The initial read delay is included.

fpu_readFloat

```
float fpu_readFloat(void);
```

This routine reads an 32-bit floating point value from the FPU. The initial read delay is included.

fpu_readString

```
char *fpu_readString(char *s);
```

This routine is used to read a zero-terminated string from the FPU and store it at the location pointed at by the passed parameter. The initial read delay is included and a pointer to the string is returned.

fpu_readStatus

```
unsigned char fpu_readStatus(void);
```

This routine reads the status byte from the FPU. An `fpu_wait` call is done internally, before the `READSTATUS` instruction is sent. The initial read delay is included.

fpu_readDelay

```
void fpu_readDelay(void);
```

After a read instruction is sent, and before the first data is read, a setup delay is required to ensure that the FPU is ready to send data. Note: All of the `fpu_read` routines include an `fpu_readDelay` call, so this function is not not usually called directly the user program.

Function Descriptions for fpu_print.c

print_version

```
void print_version(void);
```

This routine sends the FPU version string to the serial port.

print_float

```
void print_float(char format);
```

The value in register A is sent to *stdout* as a floating point string. The `format` parameter is used to specify the desired format. If the `format` parameter is zero, the length of the displayed value is variable and can be from 3 to 12 characters in length. Up to eight significant digits will be displayed if required, and very large or very small numbers are displayed in exponential notation. The special cases of NaN (Not a Number), +Infinity, -Infinity, and -0.0 are handled. Examples of the display format are as follows:

```
1.0           NaN          0.0
1.5e20        Infinity     -0.0
3.1415927     -Infinity    1.0
-52.333334    -3.5e-5      0.01
```

If the `format` parameter is non-zero, it determines the display format. The tens digit specifies the total number of characters to display and the ones digit specifies the number of digits after the decimal point. If the value is too large for the format specified, then asterisks will be displayed. If the number of digits after the decimal points is zero, no decimal point will be displayed. Examples of the display format are as follows:

Value in A register	format	Display format
123.567	61 (6.1)	123.6
123.567	62 (6.2)	123.57
123.567	42 (4.2)	*.**
0.9999	20 (2.0)	1
0.9999	31 (3.1)	1.0

print_long

```
void print_float(char format);
```

The value in register A is sent to *stdout* as a signed long integer string. The `format` parameter is used to specify the desired format. If the `format` parameter is zero, the length of the displayed value is variable and the displayed value can range from 1 to 11 characters in length. Examples of the display format are as follows:

```
1
500000
-3598390
```

If the `format` parameter is non-zero, it determines the display format. A value between 0 and 15 specifies the width of the display field for a signed long integer. The number is displayed right justified. If 100 is added to the format value the value is displayed as an unsigned long integer. If the value is larger than the specified width, asterisks will be displayed. If the width is specified as zero, the length will be variable. Examples of the display format are as follows:

Value in register A	format	Display format
-1	10 (signed 10)	-1
-1	110 (unsigned 10)	4294967295
-1	4 (signed 4)	-1
-1	104 (unsigned 4)	****
0	4 (signed 4)	0
0	0 (unformatted)	0
1000	6 (signed 6)	1000

print_fpuString

```
void print_fpuString(char opcode);
```

This routine sends the contents of the FPU string buffer to *stdout*. The opcode can be `READSTR` to read the entire string, or `READSEL` to read the current string selection.

print_CRLF

```
void print_CRLF(void);
```

This routine sends a carriage return and linefeed to *stdout*.

Further Information

The following documents are also available:

uM-FPU V3.1 Datasheet	provides hardware details and specifications
uM-FPU V3.1 Instruction Reference	provides detailed descriptions of each instruction
uM-FPU Application Notes	various application notes and examples

Check the Micromega website at www.micromegacorp.com for up-to-date information.