Micromega Corporation

uM-FPU Application Note 7

Developing a SPI Interface for uM-FPU V2

This application note describes a suggested method of developing support software for connecting a microcontroller to the uM-FPU V2 floating point coprocessor using a SPI interface.

Introduction

Micromega provides support software for many popular microcontrollers, so it's worth checking the Micromega website (http://www.micromegacorp.com) to see if software is already available for your microcontroller, or email info@micromegacorp.com to enquire about any plans for development. If support is not currently available for your microcontroller, you can easily develop your own support software. This application note describes a suggested method for developing a SPI interface. Since implementation details vary with each different microcontroller, pseudo-code is used to describe the actions of the each routine, which can then be translated into assembly code, C or Basic for your microcontroller.

Additional Documents

Before getting started it is recommended that you review the *uM-FPU V2 Datasheet*, and the *uM-FPU V2 Instruction Set* documents. It is also recommended that the serial interface for the uM-FPU debug monitor be connected as described in the *uM-FPU V2 Datasheet*. This provides access to valuable debugging information while testing the support routines.

Support software and documentation provided by Micromega for other microcontrollers can also serve as a good example of the code you will need to develop.

SPI Interface

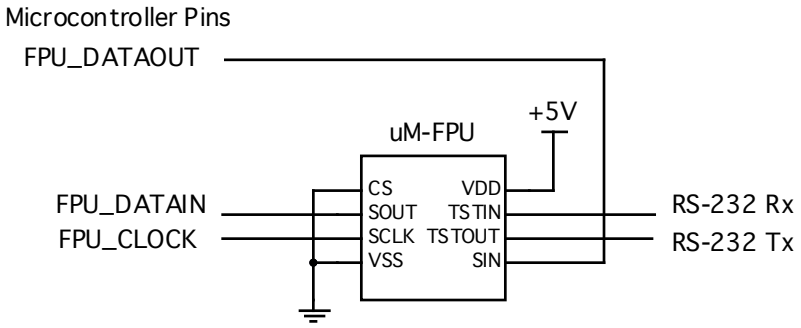
The uM-FPU can be connected using either a 2-wire or 3-wire SPI interface depending on the capabilities of the microcontroller. The 3-wire SPI interface uses separate data input and data output pins on the microcontroller, while the 2-wire SPI interface uses a single bidirectional pin for both data input and data output. If a 2-wire interface is used, the microprocessor pin must be changed from output to input as required. It can be easier to debug a 3-wire interface because the input data and output data are always clearly distinguishable when tracing with an oscilloscope or logic analyzer. If a 2-wire interface is required, a suggested approach is to initially test and debug using a 3-wire interface, then change to a 2-wire interface.

If you plan to implement SPI through software (bit-bang), any general purpose I/O pins can be used. If your microcontroller has built-in SPI hardware support that you intend to use, the I/O pins will generally be predefined.

This application note describes a software (bit-bang) implementation of a 3-wire SPI interface, and refers to the pins as follows:

Microcontroller Pin	uM-FPU Pin	Description
`FPU_CLOCK`	`SCLK`	clock
`FPU_DATAOUT`	`SIN`	output from microcontroller, input to uM-FPU
`FPU_DATAIN`	`SOUT`	input to micro, output from uM-FPU

3-wire SPI Connection



Device Level Support Routines

The interface with the uM-FPU is implemented with five device level support routines as follow:

`fpu_reset`	resets the uM-FPU
`fpu_send`	sends a byte to the uM-FPU
`fpu_read`	reads a byte from the uM-FPU
`fpu_wait`	waits until the uM-FPU buffer is empty
`fpu_readDelay`	implements the required read delay

The steps required to implement these routines are as follows:

1. Implement initial version of `fpu_reset`
2. Implement `fpu_send`
3. Implement `fpu_wait`
4. Implement `fpu_readDelay`
5. Implement `fpu_read`
6. Add synchronization check to `fpu_reset`
7. Create include file with uM-FPU opcode definitions

Step 1 – Implement initial version of fpu_reset

The uM-FPU must be reset at the start of every program to establish synchronization with the microprocessor. This is the first routine that needs to be implemented. The `fpu_reset` routine sends a reset pulse, waits for the reset to complete, then checks for proper synchronization by sending a `SYNC` opcode (`0xF0`) and reading the response byte. Since we have not yet developed the code to send and read data, we will add the synchronization check later (in step 6). The `fpu_reset` routine can also set the direction and initial value of the I/O pins used to interface to the uM-FPU.

Parameters: none
Return: sync character (`0x5C`), if reset is successful
C prototype: `unsigned char fpu_reset(void);`
Pseudo-code:

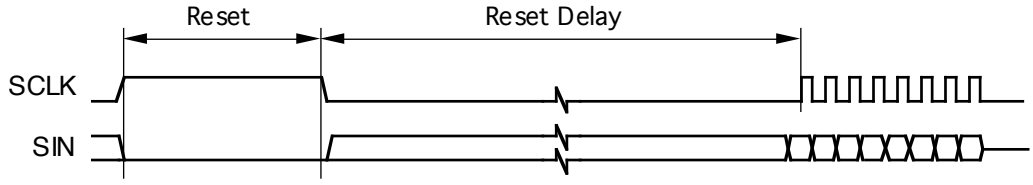
```
set FPU_CLOCK as output           ; set I/O pin direction
set FPU_DATAIN as input
set FPU_DATAOUT as output

FPU_CLOCK = 0                     ; set clock and data Low
FPU_DATAOUT = 0
FPU_CLOCK = 1                     ; pulse clock High for 500 microseconds
delay for 500 microseconds
FPU_CLOCK = 0
delay for 8 milliseconds
```

```
return
```

Using Basic:

The `INPUT` and `OUTPUT` commands are used to set the direction and initial value of the I/O pins. Use the `HIGH` and `LOW` commands to set the value of an I/O pin, and the `Pause` command for implementing a time delay.

Write a test routine to call `fpu_reset` and check the timing on uM-FPU `SCLK` and `SIN` pins. The Reset pulse must be a minimum of 500 microseconds and the Reset Delay must be a minimum of 8 milliseconds.



Debug Monitor:
Whenever a reset occurs , the following message is displayed by the debug monitor:

```
{RESET}
```

Step 2 – Implement fpu_send

The `fpu_send` routine sends an 8-bit byte to the uM-FPU. The data is sent to the uM-FPU in SPI Mode 0 format, summarized as follows:

Clock idle state is Low
Clock is active High
Data is transmitted most significant bit first
Data is latched on leading edge of the Clock
Data changes on trailing edge of the Clock

If you are implementing a software (bit-bang) interface it helps to keep the clock High and Low times relatively equal, and to balance the timing for zero and one bits. This ensures that timing is consistent regardless of the data being sent, and minimizes the additional delay code that might be required at higher microcontroller clock frequencies to ensure correct timing specifications for `SCLK` High, `SCLK` Low, and the minimum data period.

Parameters: 8-bit byte
Return: none
C prototype: `void fpu_reset(unsigned char dataByte);`
Pseudo-code:

```
dataByte                            ; 8-bit byte passed as input parameter
bit                                 ; bit value
cnt                                 ; bit count

cnt = 8                             ; set bit count

loop:                               ; shift out 8 data bits (MSB first)
   set bit = most significant bit of dataByte
   shift dataByte left
   FPU_CLOCK = 0
   if bit = 0, FPU_DATAOUT = 0
   else if bit = 1, FPU_DATAOUT = 1
   FPU_CLOCK = 1
```

```
      cnt = cnt - 1
      if cnt > 0 then goto loop

   FPU_CLOCK = 0                          ; return clock to idle state
   delay for remainder of Minimum Data Period (if required)
   return
```

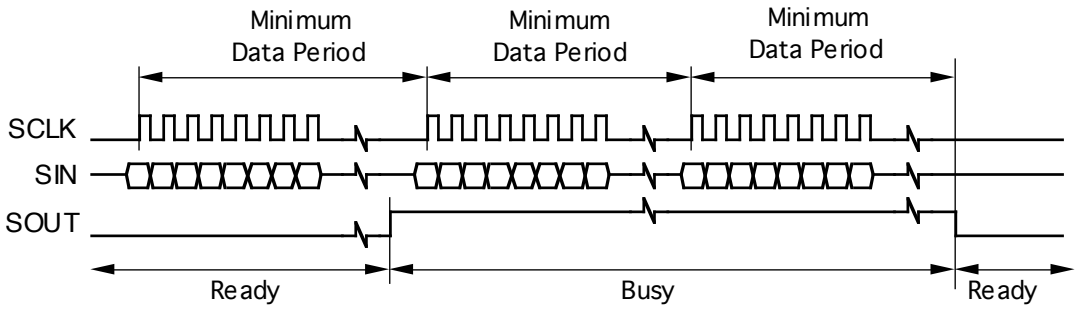
Using Basic:

There is often a `SHIFTOUT` command available for shifting data out to an I/O pin. The uM-FPU has a 32-byte instruction buffer, so if the `SHIFTOUT` command supports sending more than one byte at a time, multiple bytes can be sent with a single `SHIFTOUT` command. Alternatively the pseudo-code can be implemented using the `HIGH` and `LOW` commands.

Write a test routine to send three bytes using the `fpu_send` routine (e.g. `0x00`, `0xFF` and `0xAA`).

```
fpu_reset
fpu_send(0x00)
fpu_send(0xFF)
fpu_send(0xAA)
```

Use an oscilloscope or logic analyzer to confirm that the timing parameters meet the specifications outlined in the *uM-FPU V2 Datasheet*. `SCLK High` and `SCLK Low` should be a minimum of 250 microseconds each, and the minimum data period should be 15 microseconds. A delay may be required at to the end of the `fpu_send` routine to ensure the minimum data period is provided. The minimum data period is the time from the start of one byte to the start of the next byte.



Debug Monitor:

If the instructions are received properly by the uM-FPU, the debug monitor will display the following:

```
00 FF AA
```

Step 3 – Implement fpu_wait

The `fpu_wait` routine is used to ensure that the 32-byte instruction buffer and the debug trace buffer are both empty. The `fpu_wait` routine must always be called before data is read from the uM-FPU. It should also be called at least once for every 32 bytes of output to ensure that the instruction buffer doesn't overflow. If the uM-FPU is busy, the `FPU_DATAIN` pin will be High. The `fpu_wait` routine monitors the `FPU_DATAIN` pin and waits until it is Low.

Parameters: none
Return: none
C prototype: `void fpu_wait(void);`
Pseudo-code:

```
   loop:                                  ; wait until uM-FPU is ready
      if FPU_DATAIN = 1 then goto loop
```

```
return
```

Using Basic:
The input pins can generally be referenced as a bit or byte variable to test the value of `FPU_DATAIN`.

Step 4 – Implement fpu_readDelay

Instructions that read data from the uM-FPU require a minimum 180 microseconds delay after the opcode has been sent before data can be read (or 90 microseconds if debug trace is not enabled). The read delay routine simply delays for 180 microseconds. In some implementations, the `fpu_readDelay` routine may not be required if sufficient delay occurs due to the overhead associated with calling the interface routines. For future compatibility, and when operating at higher speeds, it is a good idea to still implement code using `fpu_readDelay` before reading data.

Parameters: none
Return: none
C prototype: `void fpu_readDelay(void);`
Pseudo-code:

```
delay for 180 microseconds          ; wait for Read Setup delay
return
```

Using Basic:
In many Basic implementations, the `fpu_readDelay` routine is not required because the overhead associated with the `SHIFTOUT` and `SHIFTIN` commands provides sufficient delay. On faster microcontrollers or when using compiled Basic it may be necessary to use `fpu_readDelay` to ensure the minimum read delay period is provided. The `PAUSE` command can be used to provide a delay.

Step 5 – Implement fpu_read

The `fpu_read` routine reads an 8-bit byte from the uM-FPU, and is implemented in a similar manner to the `fpu_send` routine. If you are implementing a software (bit-bang) interface it helps to keep the clock High and clock Low times relatively equal, and to balance the timing for zero bits and one bits. This ensures that timing is consistent regardless of the data being sent, and minimizes the additional delay code that might be required at higher microcontroller clock frequencies to ensure correct timing specifications for `SCLK` High, `SCLK` Low, and the minimum data period.

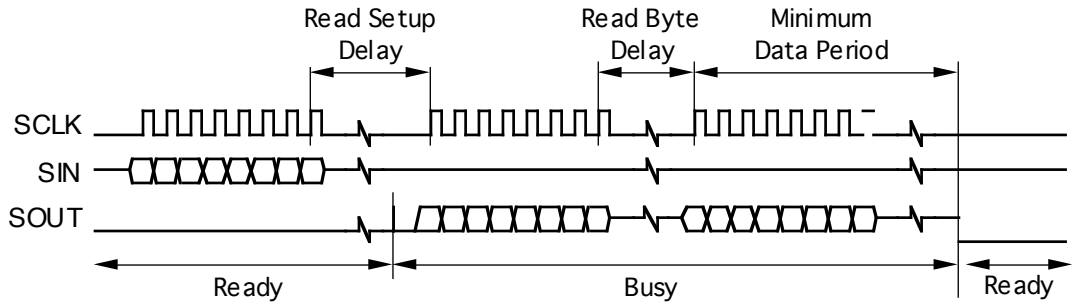
Parameters: none
Return: 8-bit byte
C prototype: `unsigned char fpu_read(void);`
Pseudo-code:

```
dataByte                            ; 8-bit byte
bit                                 ; most significant bit of dataByte
cnt                                 ; bit count

cnt = 8                             ; set bit count

loop:                               ; shift in 8 bits of data (MSB first)
   if FPU_DATAIN = 0, then bit = 0
   else if FPU_DATAIN = 1, then bit = 1
   shift dataByte left
   set least significant bit of dataByte = bit
   FPU_CLOCK = 1
   FPU_CLOCK = 0
```

```
    cnt = cnt – 1
    if cnt > 0 then goto loop

delay for remainder of Minimum Data Period (if required)
return dataByte
```

Using Basic:

There is often a SHIFTIN command available for shifting data in from an I/O pin. If the SHIFTIN command supports reading more than one byte at a time, multiple bytes can be read. Alternatively the pseudo-code can be implemented using the HIGH and LOW commands.



Write a test routine to send the SYNC opcode (0xF0) to the uM-FPU and read the byte that is returned. If SYNC is successful, a 0x5C byte will be returned. The sequence is as follows:

```
fpu_send(0xF0)                  ; send SYNC command
fpu_readDelay                   ; wait for Read Setup Delay
n = fpu_read                    ; get the return value
```

Debug Monitor:

If the instructions are received properly by the uM-FPU, the debug monitor will display the following:

```
F0:5C
```

Step 6 – Add synchronization check to fpu_reset

Add the synchronization code shown above to the end of the fpu_reset routine. The byte that is read after sending the SYNC opcode is returned by fpu_reset. The calling routine can check if the return value is 0x5C to ensure that the reset and synchronization was successful. Note: In all other situations fpu_wait must be called before sending an opcode that returns data, but fpu_wait is specifically not used in the fpu_reset routine, since an 8 millisecond delay immediately precedes it, and the uM-FPU will always be ready if the reset is successful. If the reset is not successful an fpu_wait could wait indefinitely, so by not including it, the fpu_reset routine will always return with a value.

Step 7 – Create include file with uM-FPU opcode definitions

To make it easer to write code for the uM-FPU, an include file should be created that contains definitions for all of the uM-FPU opcodes and the sync character. Various include files are available on the Micromega website and can easily be adapted as required.

Examples using the Device Level Support Routines

Calculate y = 5x + 30

```
fpu_send(SELECTA+Y)          ; select Y register
fpu_send(LOADBYTE)           ; load 5 to register 0 and convert to floating point
fpu_send(5)
fpu_send(FSET)               ; y = 5.0
fpu_send(FMUL+X)             ; y = y * x
fpu_send(LOADBYTE)
fpu_send(30)                 ; load 30 to register 0 and convert to floatint point
fpu_send(FADD)               ; y = y + 30
```

Read byte from uM-FPU register n

```
fpu_wait                     ; wait until uM-FPU is ready
fpu_send(SELECTA+N)          ; select N register (32-bit integer value)
fpu_send(XOP)                ; send READBYTE instruction
fpu_send(READBYTE)
fpu_readDelay                ; wait for read delay
n = fpu_read                 ; read the byte
```

Adding Additional Support Routines

The five device level support routines developed above provide all of the necessary support for using the uM-FPU with your microcontroller. Using these device level routines, additional routines can be developed to provide a higher level of support. Examples might include:

print_float	print formatted floating point number
print_long	print formatted long integer number
print_version	print uM-FPU version number
load_float	load floating point value
load_long	load long integer value
load_floatStr	load floating point string
load_longStr	load long integer string

Further Information

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