Micromega Corporation

Application Note 31

uM-FPU V3 Floating Point Calculations

This application note shows some examples of how to perform floating point calculations using the uM-FPU V3 floating point coprocessor. See *Application Note 32 - uM-FPU V3 Long Integer Calculations* for long integer examples and the *uM-FPU V3 Instruction Reference* for a full description of all instructions.

The *uM-FPU V3 IDE (Integrated Development Environment)* software provides a code generator that automatically produces uM-FPU V3 code from standard mathematical equations. It also generates code for various supported microcontrollers. Using the *uM-FPU V3 IDE* is an excellent way of becoming familiar with uM-FPU V3 instruction set, and is very useful for implementing your own equations and calculations. The *uM-FPU IDE* is available for download on the Micromega website.

Brief Overview of the uM-FPU V3 chip

For a full description of the uM-FPU V3 chip, please refer to the *uM-FPU V3 Datasheet*, *uM-FPU V3 Instruction Reference,* and the reference documentation for each of the supported microcontrollers.

The uM-FPU V3 chip is a separate coprocessor with its own set of registers and instructions designed to provide microcontrollers with 32-bit floating point and long integer capabilities. The microcontroller communicates with the uM-FPU using a SPI or I^2C interface. The microcontroller sends instructions and data to the uM-FPU, and the uM-FPU performs the calculations. The microcontroller is free to do other tasks while the uM-FPU performs calculations. Results can be read back to the microcontroller or stored on the uM-FPU for later use. The uM-FPU V3 chip has 128 registers, numbered 0 through 127, that can hold 32-bit floating point or long integer values. Register 0 is often used as a temporary register and is modified by some of the uM-FPU V3 instructions. Registers 1 through 127 are available for general use.

Arithmetic instructions use the value in register A as an operand and store the result of the operation in register A. Register A can be regarded as an accumulator or working register. The `SELECTA` instruction is used to select any one of the 128 registers as register A. If an instruction requires more than one operand, the additional operand is specified by the instruction. For example,

uM-FPU Instruction	Description
`FSET, 5`	register[A] = register[5]
`FADD, 10`	register[A] = register[A] + register[10]
`FSUB, 64`	register[A] = register[A] – register[64]
`FMUL, 1`	register[A] = register[A] * register[1]
`FDIV, 16`	register[A] = register[A] / register[16]

Each of the basic floating point arithmetic instructions are provided in three different forms as shown in the table below. For example, `FADD,nn` allows any general purpose register to be added to register A. The nn byte following the opcode specifies the register to be added. The `FADD0` instruction only requires the opcode and adds register 0 to register A. The `FADDI` instruction adds a small integer value to register A. The signed byte (-128 to 127) following

the opcode is converted to floating point and added to register A. The `FADD,nn` instruction is most general, but the `FADD0` and `FADDI,bb` instructions are more efficient for many common operations.

Register nn	Register 0	Immediate value	Description
`FSET,nn`	`FSET0`	`FSETI,bb`	Set
`FADD,nn`	`FADD0`	`FADDI,bb`	Add
`FSUB,nn`	`FSUB0`	`FSUBI,bb`	Subtract
`FSUBR,nn`	`FSUBR0`	`FSUBRI,bb`	Subtract Reverse
`FMUL,nn`	`FMUL0`	`FMULI,bb`	Multiply
`FDIV,nn`	`FDIV0`	`FDIVI,bb`	Divide
`FDIVR,nn`	`FDIVR0`	`FDIVRI,bb`	Divide Reverse
`FPOW,nn`	`FPOW0`	`FPOWI,bb`	Power
`FCMP,nn`	`FCMP0`	`FCMPI,bb`	Compare

The uM-FPU V3 instructions to add register 1 to register 2 is as follows:

```
SELECTA,2
FADD,1
```

To create more readable programs, names are generally assigned to the register values. This application note uses equations with the variables *x*, *y*, and *z*, so we define three uM-FPU registers `XVAL`, `YVAL` and `ZVAL` to store these values. The method of defining symbols varies depending on the microcontroller being used. See the sample programs for specific examples. Pseudo-code is used for the following definitions:

```
define XVAL  as 1       ; x value         uM-FPU register 1
define YVAL  as 2       ; y value         uM-FPU register 2
define ZVAL  as 3       ; z value         uM-FPU register 3
define C1 as 4          ; constant c1     uM-FPU register 4
define C2 as 5          ; constant c2     uM-FPU register 5
```

Using names for the registers, the sequence of uM-FPU V3 instructions for $y = x$ is:

```
SELECTA, YVAL
FADD, XVAL
```

The uM-FPU V3 instructions required for various examples are shown on the following pages. The interface routines for sending the uM-FPU V3 code depends on the microcontroller being used. Please see the reference guides for the various supported microcontrollers for more specific information.

Examples

The following examples show how to translate common mathematical equations into the uM-FPU V3 instructions required to perform the calculation. A brief explanation of each example is provided. The *uM-FPU V3 IDE* software can be used to automatically generate code for standard mathematical equations.

x = 0.0

The CLRA instruction can be used to set a register to 0.0.

```
SELECTA, XVAL           ; select XVAL as register A
CLRA                    ; XVAL = 0.0
```

x = -10.0

Immediate instructions can be used for small integer values.

```
SELECTA, XVAL           ; select XVAL as register A
FSETI, -10              ; XVAL = -10.0
```

y = x / 10.0

```
SELECTA, YVAL           ; select YVAL as register A
FSET, XVAL              ; YVAL = XVAL
FDIVI, 10               ; YVAL = YVAL / 10.0
```

x = x + pi

The LOADPI, LOADE instructions provide a convenient way to load the values of pi and e.

```
SELECTA, XVAL           ; select XVAL as register A
LOADPI                  ; register[0] = 3.1415927
FADD0                   ; XVAL = register[0]
```

x = x + acceleration of gravity

Other common constants can be loaded using the LOADCON instruction. This example uses constant 14, which is the value for the acceleration of gravity. See the *uM-FPU V3 Instruction Reference* for a full list of constants.

```
SELECTA, XVAL           ; select XVAL as register A
LOADCON, 14             ; register[0] = 9.80665
FADD0                   ; XVAL = XVAL + register[0]
```

x = x + y + z

Performing an operation using multiple registers is very straightforward.

```
SELECTA, XVAL           ; select XVAL as register A
FADD, YVAL              ; XVAL = XVAL + YVAL
FADD, ZVAL              ; XVAL = XVAL + ZVAL
```

x = y / z

```
SELECTA, XVAL           ; select XVAL as register A
FSET, YVAL              ; XVAL = YVAL
FDIV, ZVAL              ; XVAL = XVAL / ZVAL
```

y = 5x + 30

Constant values that are small integers can be easily handled using the immediate instructions. These instructions load an integer value, converts it to floating point, and stores the result in register 0.

```
SELECTA, YVAL           ; select YVAL as register A
FSETI, 5                ; YVAL = 5.0
FMUL, XVAL              ; YVAL = YVAL * XVAL
FADDI, 30               ; YVAL = YVAL + 30.0
```

y = 512.0

The LOADBYTE, LOADUBYTE, LOADWORD and LOADUWORD instructions can also be used to load integer constants. These instructions load an integer value, converts it to floating point, and stores the result in register 0. The LOADWORD and LOADUWORD are used for 16-bit values and require two 8-bit bytes after the opcode.

```
SELECTA, YVAL           ; select YVAL as register A
LOADWORD, 2, 0          ; register[0] = 512.0 (high byte, low byte)
FSET0                   ; YVAL = register[0]
```

y = 0.15x + 0.5 (using ATOF instruction)

The ATOF instruction is used to convert a zero terminated string to a floating point value. (Note: make sure there is a zero terminator on the string.) Constants can often be loaded as part of the program initialization code. In the example below, the values of C1 and C2 could be set in the initialization section and then only the last four lines of code would be required in the main program.

```
SELECTA, C1             ; select C1 as register A
ATOF, ".15", 0          ; load string, convert to floating point,
                        ;  and store in register 0
FSET0                   ; C1 = register[0]

SELECTA, C2             ; select C2 as register A
ATOF, ".5", 0           ; load string, convert to floating point,
                        ;  and store in register 0
FSET0                   ; C2 = register[0]

SELECTA, YVAL           ; select YVAL as register A
FSET, C1                ; YVAL = C1
FMUL, XVAL              ; YVAL = YVAL * XVAL
FADD, C2                ; YVAL = YVAL + C2
```

y = 0.15x + 0.5 (using FWRITE instruction)

The FWRITE instruction is a very efficient way of loading a floating point value to a uM-FPU register, but it requires that you know the 32-bit representation for the floating point number. The automatic code generation provided by the *uM-FPU V3 IDE* is a convenient way to generate floating point constants.

```
FWRITE, C1, $3E, $19, $99, $9A      ; C1 = 0.15
```

```
FWRITE, C2, $3F, $00, $00, $00      ; C2 = 0.5

SELECTA, YVAL           ; select YVAL as register A
FSET, C1                ; YVAL = C1
FMUL, XVAL              ; YVAL = YVAL * XVAL
FADD, C2                ; YVAL = YVAL + C2
```

$y = x^2$

To calculate the square of a value you can just multiply the number by itself.

```
SELECTA, YVAL           ; select YVAL as register A
FSET, XVAL              ; YVAL = XVAL
FMUL, XVAL              ; YVAL = YVAL * XVAL
```

$y = x^{-3}$

The immediate power instruction can also be used for small integer powers (-128 to +127).

```
SELECTA, YVAL           ; select YVAL as register A
FSET, XVAL              ; YVAL = XVAL
FPOWI, -3               ; YVAL = YVAL to the power -3
```

$y = x^e$

The power instruction is used with any floating point value as a power.

```
SELECTA, YVAL           ; select YVAL as register A
FSET, XVAL              ; YVAL = XVAL
LOADE                   ; register[0] = 2.7182818
FPOW0                   ; YVAL = YVAL to the power register[0]
```

y = sin(x)

The `SIN` instruction is an example of an instruction that operates on register A and doesn't require any additional operands. It calculates the sine of the value in register A and stores the result in register A. If the original value needs to be retained, you should first assign it to another variable as shown below.

```
SELECTA, YVAL           ; select YVAL as register A
FSET, XVAL              ; YVAL = XVAL
SIN                     ; YVAL = sin(YVAL)
```

y = x / y

In the previous examples, we've been able to use the variable on the left side of the equation to hold the partial results as each step of the equation is calculated. In the examples below, this is not possible because the value on the left is used in the equation and can't be modified until after it is used. Fortunately, the uM-FPU provides temporary registers through the use of parentheses instructions. The equation below can be rewritten as *y = (x / y),* and the calculation inside the parentheses uses a temporary register. Here's how it works. When a `LEFT` parenthesis instruction is sent, the current selection for register A is saved, and a temporary register is set to register A. Operations can now be performed as normal, with the temporary register selected as register A. When `RIGHT` parenthesis instruction is sent, the current value of register A is copied to register 0, and the previous selection for

register A is restored. Although it sounds complicated, the sequence of instructions is quite straightforward, a `LEFT` and `RIGHT` parenthesis simply enclose the temporary value calculations.

```
SELECTA, YVAL           ; select YVAL as register A
LEFT                    ; select temp1 as register A
FSET, XVAL              ; temp1 = XVAL
FDIV, YVAL              ; temp1 = temp1 / YVAL
RIGHT                   ; register[0] = temp1, and
                        ;  restore YVAL as register A
FSET0                   ; Y = register[0]
```

z = (x + y) / 10

The uM-FPU executes instructions in the order in which they occur. In this example, parentheses are not necessary. The addition is done first, followed by the divide.

```
SELECTA, Z              ; select ZVAL as register A
FSET, XVAL              ; ZVAL = XVAL
FADD, YVAL              ; ZVAL = ZVAL + YVAL
FDIVI, 10               ; ZVAL = ZVAL / 10.0
```

y = (x + 1) / (x – 1)

In this example, the first set of parentheses are not necessary, since *y* can be used to hold the result of the first part of the equation *(x + 1)*. The second part of the equation requires parentheses to calculate the temporary value *(x – 1)*.

```
SELECTA, YVAL           ; select YVAL as register A
FSET, XVAL              ; YVAL = XVAL
FADDI, 1                ; YVAL = YVAL + 1.0
LEFT                    ; select temp1 register as register A
FSET, XVAL              ; temp1 = XVAL
FSUBI, 1                ; temp1 = temp1 - 1.0
RIGHT                   ; register[0] = temp1, and
                        ;  restore YVAL as register A
FDIV0                   ; YVAL = YVAL / register[0]
```

z = x / (y + z)

In this example, *z* is used on the right of the equation so a temporary value is required. The equation can be rewritten as *z = (x / (y + z))*. The uM-FPU V3 supports up to eight levels of nested parentheses.

```
SELECTA, ZVAL           ; select ZVAL as register A
LEFT                    ; select temp1 register as register A
FSET, XVAL              ; temp1 = XVAL
LEFT                    ; select temp2 register as register A
FSET, YVAL              ; temp2 = YVAL
FADD, ZVAL              ; temp2 = temp2 + ZVAL
RIGHT                   ; register[0] = temp1, and
                        ;  restore temp1 as register A
FDIV0                   ; temp1 = temp1 / temp2
RIGHT                   ; register[0] = temp1, and
                        ;  restore ZVAL as register A
FSET0                   ; ZVAL = temp1
```

$z = sqrt(x^3 + y^4)$

Calculating x^3 and y^4 requires temporary values, so the equation can be rewritten as $z = sqrt((x^3) + (y^4))$.

```
SELECTA, ZVAL           ; select ZVAL as register A
LEFT                    ; select temp1 register as register A
FSET, XVAL              ; temp1 = XVAL
FPOWI, 3                ; temp1 = temp1 to the power 3
RIGHT                   ; register[0] = temp1, and
                        ;  restore ZVAL as register A
FSET0                   ; ZVAL = temp1
LEFT                    ; select temp register as register A
FSET, YVAL              ; temp1 = YVAL
FPOWI, 4                ; temp1 = temp1 to the power 4
RIGHT                   ; register[0] = temp1, and
                        ;  restore ZVAL as register A
FADD0                   ; ZVAL = ZVAL + temp1
SQRT                    ; ZVAL = sqrt(ZVAL)
```

Read 100 unsigned 8-bit samples from a sensor, and calculate the average.

The method of implementing the loop and reading the sensor will vary depending on the microcontroller sensor used. See the sample programs for specific examples. Pseudo-code is shown below.

In this example, `X` is used to accumulate the sum of 100 sensor readings. The sum is then divided by 100 to calculate the average value. Note: Since each reading sends 3 bytes to the uM-FPU and there are 100 passes through the loop, it is important to check the status of the uM-FPU inside the loop to ensure that the 256 byte uM-FPU V3 instruction buffer is not exceeded.

```
SELECTA, XVAL           ; select XVAL as register A
CLRA                    ; XVAL = 0.0

loop for i = 1 to 100
{
   read sensor value and store in svalue
   wait for uM-FPU V3 to be ready
   LOADUBYTE, svalue    ; load unsigned 8-bit value to register 0
                        ;  and convert to floating point
   FADD0                ; XVAL = XVAL + svalue (as floating point)
}
FDIVI, 100              ; XVAL = XVAL / 100.0
```

Further Information

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