Micromega Corporation

Application Note 100

Converting uM-FPU V3.1 code to uM-FPU64

Introduction

This application note provides information on converting uM-FPU V3.1 code to uM-FPU64 code. The uM-FPU64 was designed to be software compatible with the uM-FPU V3.1, but there are a few uM-FPU V3.1 instructions that are not directly supported by uM-FPU64. They are as follows:

```
COPYI
EECALL, EELOAD, EELOADA, EEREAD, EESAVE, EESAVEA, EEWRITE
LOADCON
RDBLK, WRBLK
SETOUT
```

If none of these instructions are used in the uM-FPU V3.1 code, the code will run on uM-FPU64 with no modifications.

This application note provides an overview of different features provided by uM-FPU64, instruction changes, new and modified instructions, and converting from 32-bit calculations to 64-bit calculations. For a full description of the uM-FPU64 chip, please refer to the *uM-FPU64 Datasheet*, *uM-FPU64 Instruction Set,* and additional documentation on the Micromega website.

uM-FPU64 Features that are Different from uM-FPU V3.1

64-bit support

In addition to the 128 32-bit registers (numbered 0 to 127) provided by uM-FPU V3.1, the uM-FPU64 also has 128 64-bit registers (numbered 128 to 255). All floating point and integer math operations support both 32-bit and 64-bit operations (excluding `MOP` and `FFT` which are 32-bit only) .

3.3V operating voltage

The uM-FPU64 has an operating voltage of 3.3V, but the SPI and I²C interfaces are 5V tolerant, and selected digital I/O pins are also 5V tolerant.

No EEPROM

There is no EEPROM on the uM-FPU64 chip, but local RAM or local peripheral devices can be used for data storage on the uM-FPU64.

Digital I/O and Support for Local Peripheral Devices

Extensive digital input and output support is provided, and built-in support for local peripherals including: asynchronous serial interface (with optional hardware flow control), local SPI bus, local I²C bus, local 1-Wire bus, LCD display, VDRIVE2 (USB storage). The local peripheral devices can be assigned to any of the digital pins.

Expanded ADC Support

The ADC has multiple channels and DMA support. Using DMA support, data can be sampled at faster data rates.

Local RAM

Provides general data storage and can be used for handling data from the local peripherals.

Background Event Processing

User-defined functions can be defined to handle background events (e.g. serial input, GPS input, analog-to-digital conversion completion, real-time events, etc.) Foreground/background processing is supported.

Field Upgradable Firmware

New firmware can be downloaded from the internet and loaded to the uM-FPU64 using the uM-FPU64 IDE software.

uM-FPU V3.1 Instruction Changes

The following uM-FPU V3.1 instructions have been removed from the uM-FPU64 instruction set or have been replaced by new instructions:

```
COPYI
EECALL, EELOAD, EELOADA, EEREAD, EESAVE, EESAVEA, EEWRITE
LOADCON
RDBLK, WRBLK
SETOUT
```

COPYI

The `COPYI` instruction (uM-FPU V3.1) has been replaced by the `LCOPYI` and `FCOPYI` instructions (uM-FPU64). `COPYI` uses an unsigned byte, while `LCOPYI` and `FCOPYI` use a signed byte.

uM-FPU V3.1

`COPYI, unsignedByte, register`

Converts an unsigned byte to a 32-bit integer, and stores it to the specified *register*.

uM-FPU64

`LCOPYI, signedByte, register`

Converts a signed byte to a 32-bit or 64-bit integer and stores the value in the specified *register*.

`FCOPYI, signedByte, register`

Converts a signed byte to a 32-bit or 64-bit floating point value and stores it in the specified *register*.

Suggested changes:

If the `COPYI` instruction uses an unsigned byte with a value of 0 to 127, then:

```
COPYI, unsignedByte
```

is replaced by:

```
LCOPYI, signedByte
```

If the `COPYI` instruction uses an unsigned byte with a value of 128 to 255 then:

```
COPYI, unsignedByte
```

is replaced by:

```
LONGUBYTE, unsignedByte
COPY0, register
```

EECALL, EELOAD, EELOADA, EEREAD, EESAVE, EESAVEA, EEWRITE

There is no EEPROM on the uM-FPU64, so all EEPROM instructions have been removed from the uM-FPU64 instruction set.

Suggested changes:

The uM-FPU64 `DEVIO` instruction provides local device support, including local SPI and I²C bus support. This can be used to interface an external SPI or I²C EEPROM chip, or other device, directly to the uM-FPU64 chip for reading and writing data.

LOADCON

The uM-FPU V3.1 `LOADCON` instruction has been removed from the uM-FPU64 instruction set. `LOADCON` loads a floating point constant to register 0. It can be replaced by the `FWRITE0` instruction, using the floating point value for the constant. The constants are listed in the *uM-FPU V3.1 Instruction Set* document.

Suggested changes:

For example, to load the constant for the acceleration of gravity:

```
LOADCON, 14
```

is replaced by:

```
FWRITE0, 9.80665
```

RDBLK, WRBLK

The uM-FPU V3.1 `RDBLK` instructions is used to read multiple 32-bit values from sequential registers (using register X). The uM-FPU V3.1 `WRBLK` instructions is used to write multiple 32-bit values to sequential registers (using register X). These instructions have been replaced on uM-FPU64 by the `RDIND` and `WRIND` instructions. These instructions are more comprehensive, and use indirect pointers to provide support for reading and writing multiple 8-bit, 16-bit, 32-bit and 64-bit values. The values are automatically converted as required, between integer and floating point, and between data sizes.

Suggested changes:

The uM-FPU V3.1 instruction sequence to write twenty 32-bit floating point values to registers 10 through 29:

```
SELECTX, 10
WRBLK, 20
{send twenty 32-bit floating point values}
```

is replaced by:

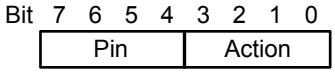
```
SELECTA, 1
SETIND, REG_FLOAT, 10
WRIND, FLOAT32, 1, 20
{send twenty 32-bit floating point values}
```

The `SETIND` instruction loads register 1 with a pointer to register 10. The `WRIND` instruction uses the indirect pointer in register 1 to determine where to store the twenty 32-bit floating point values. If the destination was a 64-bit register, an automatic 32-bit to 64-bit conversion is done. See the *uM-FPU64 Instruction Set* for more details.

SETOUT

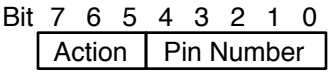
The uM-FPU V3.1 `SETOUT` instruction is used to output digital values to pins `OUT1` and `OUT2`. The uM-FPU64 replaces this instruction with the `DIGIO` instruction, which supports digital input and output to pins `D0` to `D22`.

The action byte for the uM-FPU V3.1 `SETOUT` instruction is as follows:



Bits 7:4 Output pin (upper nibble)
0 - OUT 0
1 - OUT 1
Bits 3:0 Action (lower nibble)
0 - set output low
1 - set output high
2 - toggle the output to opposite level
3 - set output to high impedance

The action byte for the uM-FPU64 `DIGIO` instruction is as follows:



Bits 7:5 **Action**

IDE Symbol	*IDE Value*	*Description*
`LOW`	0x00	Set pin as output. Set pin to low.
`HIGH`	0x20	Set pin as output. Set set to high.
`TOGGLE`	0x40	Set pin as output. Toggle pin level.
`INPUT`	0x60	Set pin as input. Read pin value and set Z
`WRITE_BITS`	0x80	Write serial value to pin.
`READ_BITS`	0xA0	Read serial value from pin.
`WRITE_BITP`	0xC0	Write parallel value to pins.
`READ_BITP`	0xE0	Read parallel value from pins.

Suggested changes:
The following uM-FPU V3.1 instructions set `OUT0` low, then high.

```
SETOUT, $00
SETOUT, $01
```

is replaced by the following uM-FPU64 instructions to set `D0` low, then high:

```
DIGIO, LOW+D0
DIGIO, HIGH+D0
```

New Instructions in uM-FPU64

The following instructions have been added to the uM-FPU64 instruction set:

SETIND, ADDIND, COPYIND, RDIND, WRIND
Expanded indirect pointer support.

DELAY
Delay in milliseconds.

DEVIO
Local peripheral device support.

DIGIO
Digital input/output on pins `D0` to `D22`.

DREAD, DWRITE
64-bit read and write.

EVENT
Background event processing.

LBIT
Clear, set, toggle, and test bits.

LCOPYI, FCOPYI
Copy 8-bit unsigned value to a register and convert to integer or floating point value.

LSHIFTI, LORI, LANDI
Shift, bitwise-AND, or bitwise-OR operation using immediate value.

RTC
Real-time clock support.

SETARGS
Enables parameter passing mechanism for `FCALL`.

SETREAD
Issued prior to all read operations to set the read mode and ensure that the foreground process is active.

Note: If background processing is enabled (see `EVENT` instruction), then `SETREAD` must precede all read operations. If background processing is not used, `SETREAD` is optional, but it's a good idea to use this instruction for all new code, to ensure it's compatible if background processing is used in the future.

uM-FPU V3.1 instructions Modified in uM-FPU64
The following instructions have been been modified in uM-FPU64:

All Math Instructions (except MOP and FFT)
Support both 32-bit and 64-bit operations.

ADCLOAD, ADCLONG, ADCMODE, ADCSCALE
Supports multiple ADC channels on pins `AN0` to `AN8`. `ADCMODE` provides additional modes and configurations. `ADCSCALE` supports both scale and offset.

SERIN, SEROUT
Supports serial input/output to both `SERIN/SEROUT`, and the `DEVIO,ASYNC` device,

Converting from 32-bit Calculations to 64-bit Calculations
The uM-FPU64 chip was designed to use the same instruction set as uM-FPU V3.1, so it's easy to move existing code from uM-FPU V3.1 to uM-FPU64. Likewise, since the same instructions are used, it's easy to convert 32-bit calculations to 64-bit calculations - it simple requires changing the register definitions.

Registers
The uM-FPU V3.1 chip has 128 32-bit registers numbered 0 to 127.

Registers 0 to 127	32-bit registers

The uM-FPU64 chip has the same 128 32-bit registers numbered 0 to 127, but also has 128 64-bit registers numbered 128 to 255.

Registers 0 to 127	32-bit registers
Registers 128 to 255	64-bit registers

Register A

Register A determines if the operation is 32-bit or 64-bit. If register A selects a 32-bit register, the calculations are performed using 32-bit operations. If register A selects a 64-bit register, the calculations are performed using 64-bit operations.

Register 0 and 128

On the uM-FPU V3.1 chip, many instructions implicitly load 32-bit values to register 0 (e.g. the `LOADBYTE`, `LOADWORD`, etc.). On the uM-FPU64 chip, these instructions can load a 32-bit value to register 0, or a 64-bit value to register 128. If register A selects a 32-bit register, a 32-bit value is loaded to register 0. If register A selects a 64-bit register, a 64-bit value is loaded to register 128.

Converting from 32-bit to 64-bit

To convert 32-bit calculations to 64-bit calculations, you simply reassign the registers from the 32-bit register bank (registers 0 to 127) to the 64-bit register bank (registers 128 to 255).

The following examples assume that registers 1 and 2 contain a 32-bit floating point value, and registers 129 and 130 contain a 64-bit floating point value.

The following example adds two 32-bit numbers.

```
SELECTA, 1          ; select register 1 as reg[A]
FADD, 2             ; reg[A] = reg[A] + reg[2] (32-bit floating point operation)
```

By simply changing the register definitions, the following example adds two 64-bit numbers.

```
SELECTA, 129        ; select register 129 as reg[A]
FADD, 130           ; reg[A] = reg[A] + reg[130] (64-bit floating point operation)
```

Conversions between 32-bit and 64-bit values are handled automatically. If register A is 32-bit, but one of the operands is 64-bit, the operand will be converted to 32-bit before the calculation. If register A is 64-bit, but one of the operands is 32-bit, the operand will be converted to 64-bit before the calculation. The conversion is done inside the instruction, the original value in the operand register is unchanged.

The following example adds a 64-bit number to a 32-bit number.

```
SELECTA, 1          ; select register 1 as reg[A]
FADD, 130           ; reg[A] = reg[A] + reg[130]
                    ; the value from reg[130] is converted to 32-bit floating point
                    ; before being added to reg[A]
```

The following example adds a 32-bit number to a 64-bit number.

```
SELECTA, 129        ; select register 129 as reg[A]
FADD, 2             ; reg[A] = reg[A] + reg[2]
```

```
; the value from reg[2] is converted to 64-bit floating point
; before being added to reg[A])
```

Further Information

See the Micromega website (http://www.micromegacorp.com) for additional information regarding the uM-FPU64 floating point coprocessor, including:

uM-FPU64 Datasheet
uM-FPU64 Instruction Set