Micromega Corporation

uM-FPU64

64-bit Floating Point Coprocessor

Datasheet

Release 411

Introduction

The uM-FPU64 floating point coprocessor provides extensive support for 32-bit IEEE 754 compatible floating point and integer operations, 64-bit IEEE 754 compatible floating point and integer operations, and local peripheral device support. The uM-FPU64 chip easily interfaces to virtually any microcontroller using an SPI™ or I^{2}C™ interface. It is upwardly code compatible with the uM-FPU V3.1 floating point processor, allowing for easy migration of existing code.

Features

- 3.3V operating voltage
- 40 MHz instruction cycle
- 5V tolerant SPI and I^{2}C interface
- 32-bit IEEE 754 compatible floating point and 32-bit integer operations
- 64-bit IEEE 754 compatible floating point and 64-bit integer operations
- Local Peripheral Device Support
  - Asynchronous serial port (with optional hardware flow control)
  - SPI bus
  - I^{2}C bus
  - 1-Wire bus
  - Counters with switch debounce
  - Servo controller
  - LCD display
  - VDRIVE2 (USB storage)
  - RAM storage
  - FIFO buffers
- Digital Input/Output
  - 9 pins on 28-pin device
  - 23 pins on 44-pin device
  - 5V tolerant input/output on selected pins
- Analog-to-Digital input
  - 6 channels on 28-pin device
  - 9 channels on 44-pin device
- 6144 bytes of Flash memory for user-defined functions
- 2304 bytes of user RAM
- GPS serial input with NMEA sentence parsing
- FFT operations
- Matrix operations
- Background event processing
- Real-time clock
- Field upgradeable firmware

Block Diagram

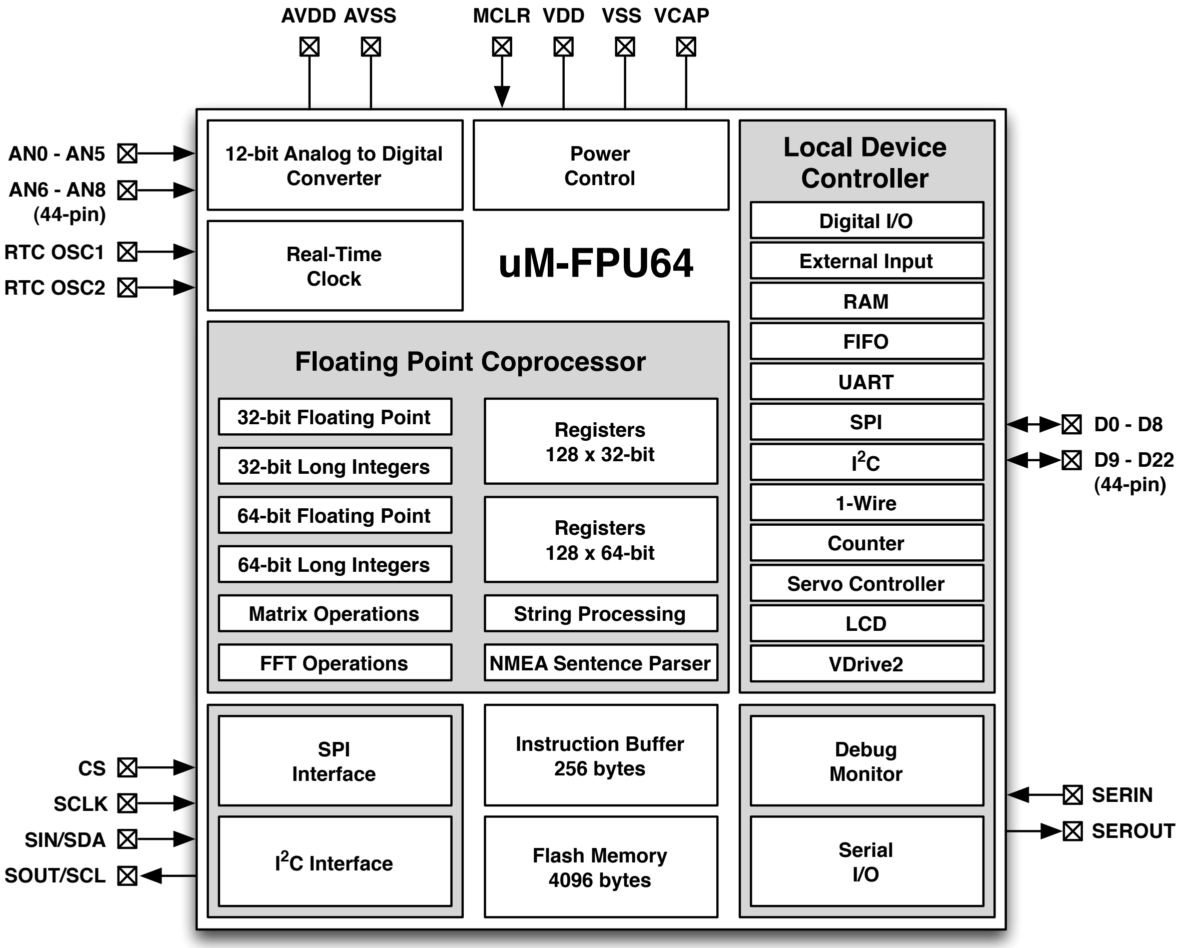


Table of Contents

Brief Overview

64-bit and 32-bit Floating Point

A comprehensive set of 64-bit and 32-bit floating point operations are provided. See the uM-FPU64 datasheet for details.

64-bit and 32-bit Integer

A comprehensive set of 64-bit and 32-bit integer operations are provided. See the uM-FPU64 datasheet for details.

Local Device Support

Local peripheral device support includes: RAM, 1-Wire, I_2C, SPI, UART, counter, servo controller, LCD, and VDrive2 devices. The uM-FPU64 can act as a complete subsystem controller for GPS, sensor networks, robotic subsystems, IMUs, and other applications. Local devices are assigned to digital I/O pins at run-time, and controlled with the DEVIO instruction.

User-defined Functions

User-defined functions can be stored in Flash memory. Flash functions are programmed through the SERIN/ SEROUT pins using the *uM-FPU64 IDE*. A high level language is supported, including control statements and conditional execution.

Matrix Operations

A matrix can be defined as any set of sequential registers. The MOP instruction provides scalar operations, element-wise operations, matrix multiply, inverse, determinant, count, sum, average, min, max, copy and set operations.

FFT Instruction

Provides support for Fast Fourier Transforms. Used as a single instruction for data sets that fit in the available registers, or as a multi-pass instruction for working with larger data sets.

Serial Input / Output

When not required for debugging, the SERIN and SEROUT pins can be used for serial I/O. A second asynchronous serial port, with hardware flow control, is available as a local device using the DEVIO instruction.

NMEA Sentence Parsing

The serial input can be set to scan for valid NMEA sentences with optional checksum. Multiple sentences can be buffered for further processing.

String Handling

String instructions are provided to insert and append substrings, search for fields and substrings, convert from floating point or long integer to a substring, or convert from a substring to floating point or long integer. For example, the string instructions could be used to parse a GPS NMEA sentence, or format multiple numbers in an output string.

Table Lookup Instructions

Instructions are provided to load 32-bit values from a table or find the index of a floating point or long integer table entry that matches a specified condition.

MAC Instructions

Instructions are provided to support multiply and accumulate and multiply and subtract operations.

A/D Conversion

Multiple 12-bit A/D channels are provided (six on 28-pin device, nine on 44-pin device). The A/D conversion can be triggered manually, through an external input, or from a built-in timer. The A/D values can be read as raw values or automatically scaled to a floating point value. Data rates of up to 10,000 samples per second are supported.

Real-Time Clock

A built-in real-time clock is provided, for scheduling events or creating date/time stamps.

Foreground/Background Processing

Event driven foreground/background processing can be used to provide independent monitoring of local peripherals. The microcontroller communicates with the foreground, while background processes can be used to monitor local device activity.

Timers

Timers can be used to trigger the A/D conversion, or to track elapsed time. A microsecond and second timer are provided.

External Input

An external input can be used to trigger an A/D conversion, trigger an eventor to count external events.

Low Power Modes

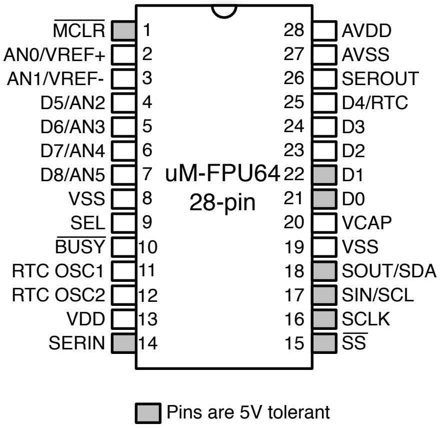
When the uM-FPU64 chip is not busy it automatically enters a power saving mode. It can also be configured to enter a sleep mode which turns the device off while preserving register contents. In sleep mode the uM-FPU64 chip consumes negligible power.

Firmware Upgrades

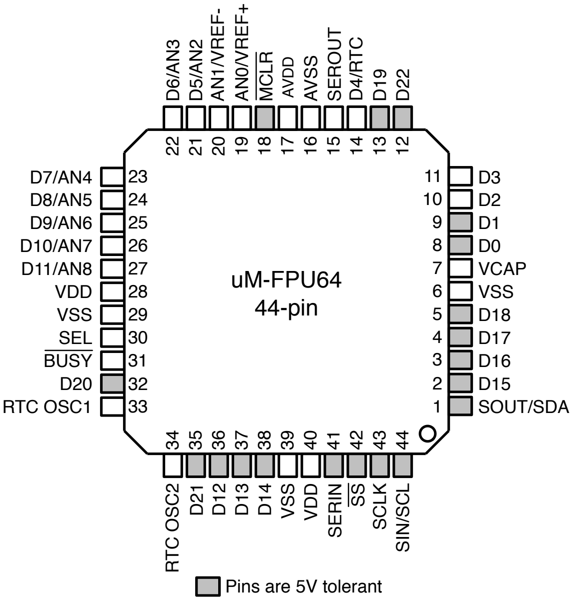
When updates become available, the uM-FPU64 firmware can be upgraded in the field using the *uM-FPU64 IDE*.

uM-FPU64 Pin Diagrams

PDIP-28, SOIC-28



TQFP-44

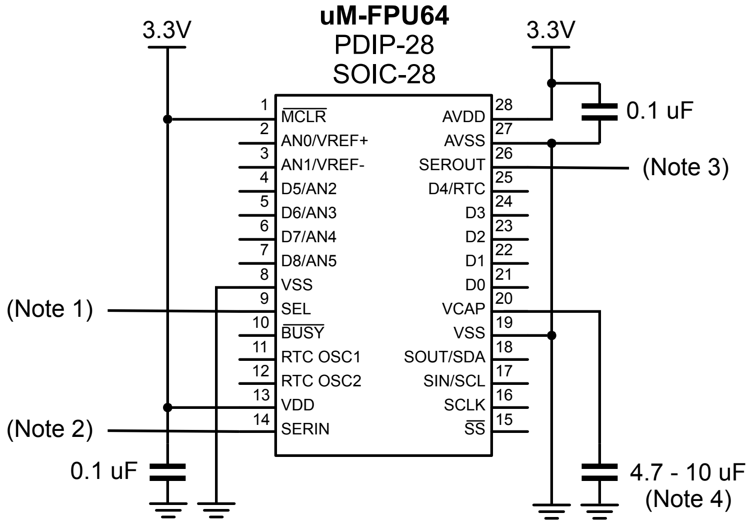


Pin Descriptions

Name	Type	Description
Power/Ground Pins		
VDD	Power	Digital Supply Voltage
VSS	Power	Digital Ground
AVDD	Power	Analog Supply Voltage
AVSS	Power	Analog Ground
VCAP	Power	Filter Capacitor (6.8uF to 10uF)
System Pins		
/MCLR	Input	Master Clear (Reset)
RTC OSCI	Input	Real-time Clock 32.768 Crystal Oscillator
RTC OSCO	Output	Real-time Clock 32.768 Crystal Oscillator
SEL	Input	Interface Select
/BUSY	Output	Ready/Busy Status
SPI Interface Pins		
SCLK	Input	SPI Clock
SIN	Input	SPI Input
SOUT	Output	SPI Output, Busy/Ready Status
/SS	Input	SPI Slave Select
I^{2}C Interface Pins		
SCL	Input	I^{2}C Clock
SDA	Input/Output	I^{2}C Data
Serial Input/Output, Debug Monitor Pins		
SERIN	Input	Serial Input, Debug Monitor - Rx
SEROUT	Output	Serial Output, Debug Monitor - Tx
Digital Input/Output Pins		
D0-D8	Input/Output	Digital Input/Output
D9-D22	Input/Output	Digital Input/Output (44-pin)
RTC	Output	Real-time Clock Output
Analog Input Pins		
AN0-AN5	Input	Analog Input Channels
AN6-AN8	Input	Analog Input Channels (44-pin)
VREF+	Input	Analog Voltage Reference (high)
VREF-	Input	Analog Voltage Reference (low)

Minimum Recommended Connections

PDIP-28, SOIC-28



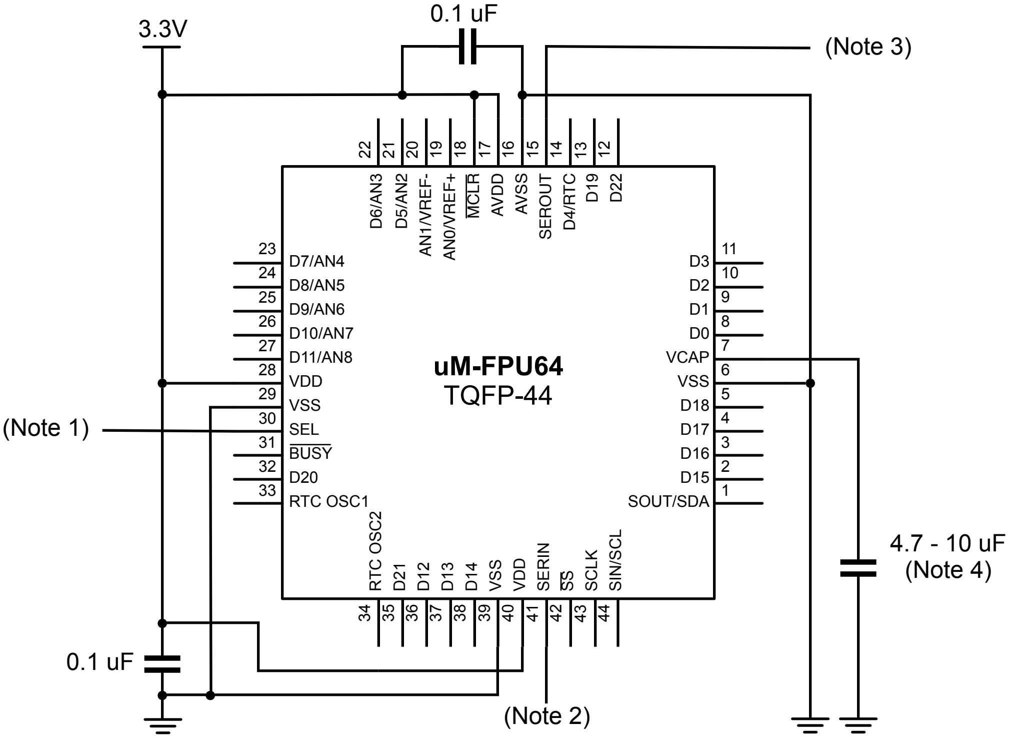
Note 1: The SEL pin is used to select the type of interface. It is connected to 3.3V for I2C, or GND for SPI. The type of interface can also be set using parameter byte 0.

Note 2: To use the debug monitor, the SERIN pin must be connected to the PC serial output through a USB-to-3.3V Serial Adapter. If the debug monitor is not used, the SERIN pin must be connected to GND.

Note 3: To use the debug monitor, the SEROUT pin is connected to the PC serial output through a USB-to-3.3V Serial Adapter. If the debug monitor is not used, no connection is required to the SEROUT pin.

Note 4: A low-ESR (less than 5 ohms) tantalum or ceramic capacitor is required to provide regulated power.

TQFP-44



Note 1: The SEL pin is used to select the type of interface. It is connected to 3.3V for I2C, or GND for SPI. The type of interface can also be set using parameter byte 0.

Note 2: To use the debug monitor, the SERIN pin must be connected to the PC serial output through a USB-to-3.3V Serial Adapter. If the debug monitor is not used, the SERIN pin must be connected to GND.

Note 3: To use the debug monitor, the SEROUT pin is connected to the PC serial output through a USB-to-3.3V Serial Adapter. If the debug monitor is not used, no connection is required to the SEROUT pin.

Note 4: A low-ESR (less than 5 ohms) tantalum or ceramic capacitor is required to provide regulated power.

Connecting to the uM-FPU64 chip

The uM-FPU64 chip can be interfaced using one of several different types of SPI interface, or an I^{2}C interface. The different types are as follows:

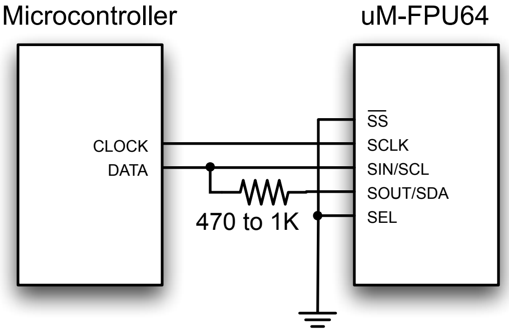
- 2-wire SPI interface, single device
- 3-wire SPI interface, single device
- SPI bus interface, multiple devices
- I^{2}C interface, multiple devices

By default, the SEL pin is used to select between SPI or I^{2}C interfaces. Alternatively, the interface type can also be set using a parameter byte stored in Flash (see the section called *Mode – Set Parameter Bytes*).

2-wire SPI interface

When the uM-FPU64 chip is connected directly to the microcontroller as a single device, no chip select is required, and either a 2-wire or 3-wire SPI interface can be used depending on the capabilities of the microcontroller. The 2-wire SPI connection uses a single bidirectional pin for both data input and data output. When a 2-wire SPI interface is used, the SOUT and SIN pins should not be connected directly together, ***they must be connected through a 470Ω to 1K resistor.*** The microcontroller data pin is connected to the SIN pin. The SEL pin is tied low to select SPI mode at Reset. The /SS pin must remain low during operation. The connection diagrams are shown below.

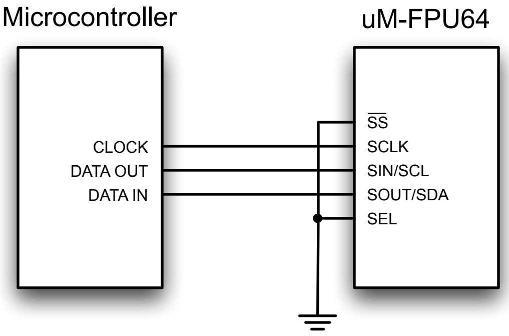
2-wire SPI Connection



3-wire SPI interface

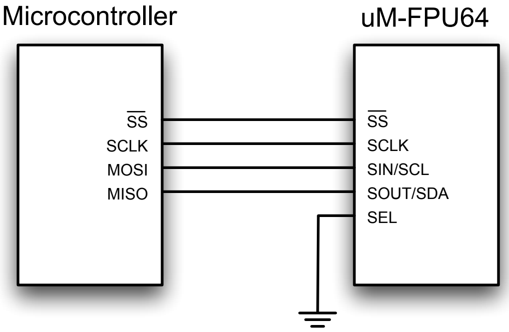
The 3-wire SPI connection uses separate data input and data output pins on the microcontroller. The SEL pin is tied low to select SPI mode at Reset.

3-wire SPI Connection



SPI Bus Interface

The SPI bus interface allows multiple SPI devices to be controlled by the microcontroller. The SEL pin is tied low to select SPI mode at Reset. The /SS pin is used as an active low SPI slave device select. The SOUT pin is a tri-state output and is high impedance when the FPU chip is not selected. The connection diagram is shown below:

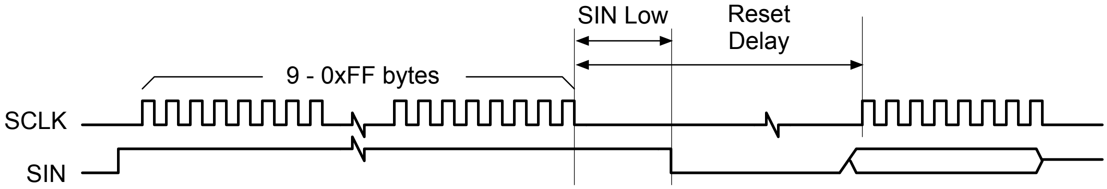


The clock signal is idle low and data is read on the rising edge of the clock (often referred to as SPI Mode 0).

SPI Reset Operation

The FPU should be reset at the beginning of every program to ensure that the microcontroller and the FPU are synchronized. The uM-FPU will prepare for a reset after nine consecutive 0xFF bytes are read, but it is recommended that ten 0xFF bytes be sent by the microcontroller to ensure that at least nine 0xFF bytes are recognized even if the microcontroller and FPU are out of sync. The reset does not occur until the SIN goes Low. If SIN remains High after sending the ten 0xFF bytes, a 0x00 byte must be sent (or SIN must be set Low) to trigger the reset. Note: If SIN does not go Low within 100 milliseconds of receiving nine 0xFF bytes, a reset will be triggered by default. A delay of 10 milliseconds is recommended after the reset is triggered to ensure that the reset sequence is complete and the FPU is ready to receive commands. All FPU registers are reset to the special value NaN (Not a Number), which is equal to hexadecimal 0x7FFFFFFF (32-bit) or 0x7FFFFFFFFFFFFFFF (64-bit).

Reset Timing Diagram



Item	Min	Typical	Max	Unit
Reset - 0xFF bytes	9	10		bytes
Reset - SIN Low			100	msec
Reset Delay	10			msec

SPI Reading and Writing Data

The uM-FPU64 is configured as a Serial Peripheral Interconnect (SPI) slave device. Data is transmitted and received with the most significant bit (MSB) first using SPI mode 0, summarized as follows:

SCLK is active High (idle state is Low)
Data latched on leading edge of SCLK
Data changes on trailing edge of SCLK
Data is transmitted most significant bit first

The maximum SCLK frequency is 15 MHz, but there must be minimum data period between bytes. The minimum data period is measured from the rising edge of the first bit of one date byte to the rising edge of the first bit of the next data byte. The minimum data period must elapse before the Busy/Ready status is checked.

Read Delay

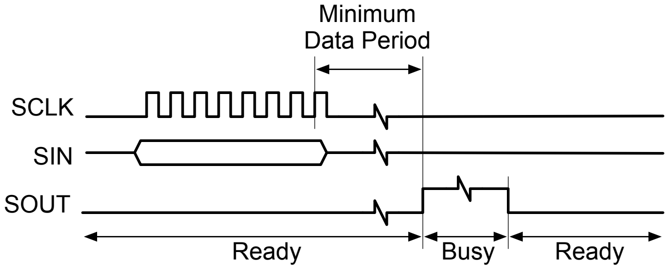
There is a minimum delay (Read Setup Delay) required from the end of a read instruction opcode until the first data byte is ready to be read. With many microcontrollers the call overhead for the interface routines is long enough that no additional delay is required. On faster microcontrollers a suitable delay must be inserted after a read instruction to ensure that data is valid before the first byte is read.

SPI Busy/Ready Status

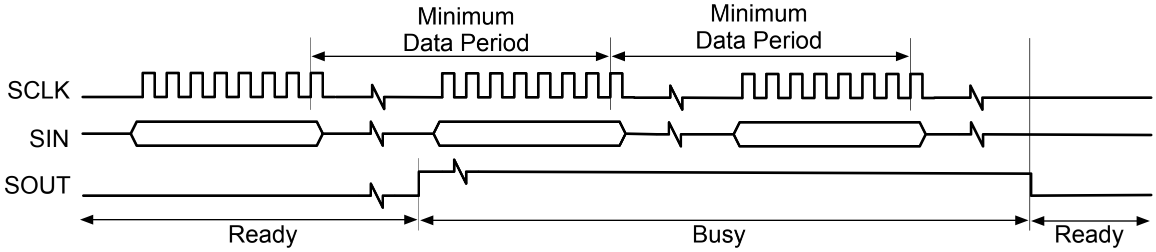
The busy/ready status must always be checked to confirm the Ready status prior to any read operation.
The Busy status is asserted as soon as an instruction byte is received. The Ready status is asserted when both the instruction buffer and trace buffer are empty. If the FPU is Ready the SOUT pin is held Low. If the FPU is Busy, either executing instructions, or because the debug monitor is active, the SOUT pin is held High. The minimum data period must have elapsed since the last byte was transmitted before the SOUT status is checked. If more than 256 bytes of data are sent between read operations, the Ready status must also be checked at least once every 256 bytes to ensure that the instruction buffer does not overflow. The /BUSY pin can also be used to check the Busy/Ready Status.

SPI Instruction Timing Diagrams

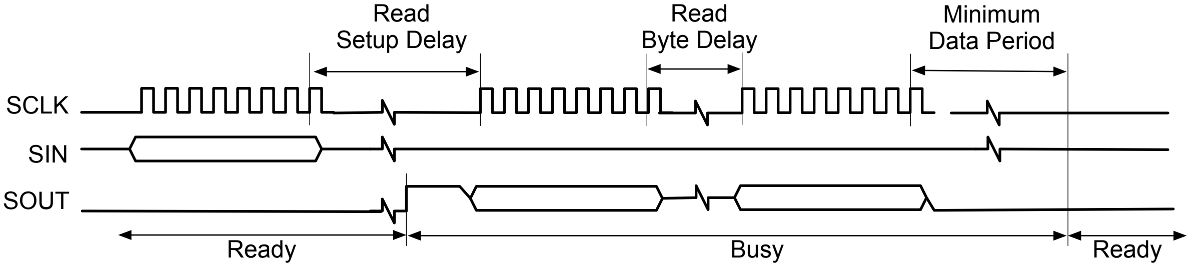
Single Byte Opcode



Multiple Byte Opcode



Opcode followed by return value

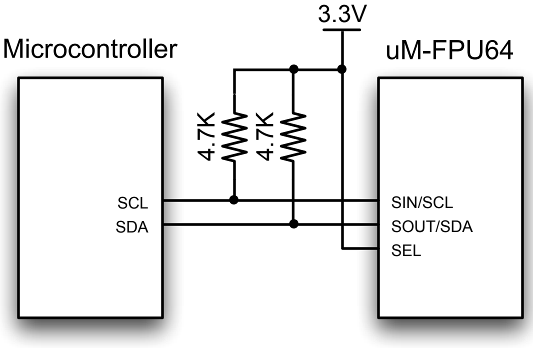


Item	Min	Max	Unit
SCLK Output Low	30		nsec
SCLK Output High	30		nsec
SCLK Frequency - single byte		15	MHz
SCLK Frequency - continuous		5	MHz
Minimum Data Period	1.6		usec
Read Setup Delay	15		usec
Read Byte Delay	1		usec
Falling Edge of /SS to Rising Edge of SCLK	120		nsec
Falling Edge of /SS to Busy/Ready Check	1		usec
Rising Edge of /SS to Bus Released		500	nsec

I²C interface

If the SEL pin is a logic high at reset (e.g. tied to VDD), the uM-FPU64 will be configured as an I²C slave device. Using an I²C interface allows the FPU to share the I²C bus with other peripheral chips. The connection diagram is shown below.

I²C Connection



I²C Slave Address

The slave address is 7 bits long, followed by an 8th bit which specifies whether the master wishes to write to the slave (0), or read from the slave(1). The default slave address for the FPU is 1100100x (binary).

- expressed as a 7-bit value, the default slave address is 100 (decimal), or 0x64 (hex).
- expressed as a left justified 8-bit value the default slave address is 200 (decimal) or 0xC8 (hex).

The slave address can be changed using the built-in serial debug monitor and stored in nonvolatile flash memory.

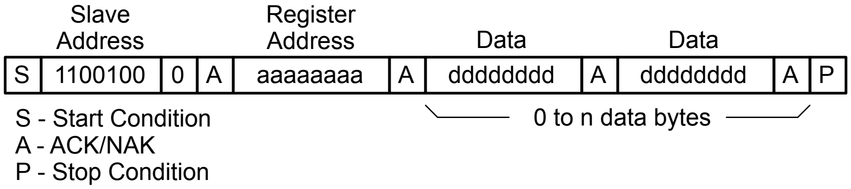
I²C Bus Speed

The uM-FPU64 can handle I²C data speeds up to 400 kHz.

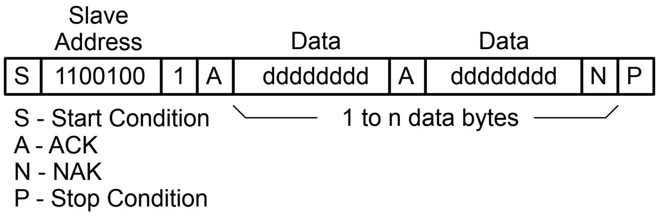
I²C Data Transfers

The following diagrams show the write and read data transfers. A write transfer consists of a slave address, followed by a register address, followed by 0 to n data bytes. A read transfer is normally preceded by a write transfer to select the register to read from.

I²C Write Data Transfer



I²C Read Data Transfer



I²C Registers

I²C Register Address	Write	Read
0	Data	Data / Status
1	Reset	Buffer Space

Item	Min	Max	Unit
I²C transfer speed		400	kHz
Read Delay – normal operation	TBD	TBD	usec
Read Delay – debug enabled	TBD	TBD	usec

I²C Reset Operation

The uM-FPU64 should be reset at the beginning of every program to ensure that the microcontroller and the FPU are synchronized. The FPU is reset by writing a zero byte to I²C register address 1. A delay of 10 milliseconds is recommended after the reset operation to ensure that the Reset is complete and the FPU is ready to receive commands. All FPU registers are reset to the special value NaN (Not a Number), which is equal to hexadecimal value 0x7FC00000.

I²C Reading and Writing Data

uM-FPU64 instructions and data are written to I²C register 0. Reading I²C register 0 will return the next data byte, if data is waiting to be transferred. If no data is waiting to be transferred the Busy/Ready status is returned. A read operation is normally preceded by a write operation to select the I²C register to read from.

I²C Busy/Ready Status

The Busy/Ready status must always be checked to confirm that the FPU is Ready prior to any read operation. The Busy status is asserted as soon as an instruction byte is received. The Ready status is asserted when both the instruction buffer and trace buffer are empty. If the FPU is Ready, a zero byte is returned. If the FPU is Busy, either executing instructions, or because the debug monitor is active, a 0x80 byte is returned. If more than 256 bytes of data are sent between read operations, the Ready status must also be checked at least once every 256 bytes to ensure that the instruction buffer does not overflow.

I²C Buffer Space

Reading I²C register 1 will return the number of bytes of free space in the instruction buffer. This can be used by more advanced interface routines to ensure that the instruction buffer remains fully utilized. It is only used to determine if there is space to write data to the FPU. The Busy/Ready status must still be used to confirm the Ready status prior to any read operation.

Read Delay

There is a minimum delay (Read Setup Delay) required from the end of a read instruction opcode until the first data byte is ready to be read. With many microcontrollers the call overhead for the interface routines is long enough that no additional delay is required. On faster microcontrollers a suitable delay must be inserted after a read instruction to ensure that data is valid before the first byte is read.

Using /BUSY as a Busy/Ready Status

The Busy/Ready status of the uM-FPU64 is always output on the `/BUSY` pin. This can be used to create an activity indicator by connecting it to a LED with a pull-up resistor, or in certain cases can be used by the microcontroller to monitor the Busy/Ready status of the FPU. By default, the FPU outputs the Busy/Ready status on the `SOUT` pin, when the `SOUT` pin is not being used for data input. This is the normal method for checking the Busy/Ready status when using an SPI interface, but some microcontrollers that use hardware SPI support are not able to access this pin directly. In those cases, the `/BUSY` pin can be used instead. The Busy/Ready status can be removed from the `SOUT` pin by programming bit 6 of mode parameter byte 0. See the section entitled *Mode - set mode parameters*.

Note: The polarity of the Busy/Ready status is opposite for the `SOUT` pin and the `/BUSY` pin.

- The `SOUT` pin is High when Busy
- The `/BUSY` pin is Low when Busy

This allows for a simple connection from the `/BUSY` pin to an LED with a pull-up resistor. The LED will turn on when the FPU is Busy.

Reset using the /MCLR pin

The /MCLR pin can optionally be used as a hardware reset for the uM-FPU64. This is not required since a reset can be initiated through software for both SPI and I^{2}C interfaces. If the /MCLR pin is used as a hardware reset, it should be connected through a 10K resistor to VDD (+3.3V). The /MCLR pin must be brought low for a minimum of two microseconds to initiate a hardware reset.

Idle and Sleep Power Saving Modes

The uM-FPU64 can be enabled for two power saving modes. Idle and Sleep modes reduce power consumption when debug mode is not enabled, and the FPU is not executing instructions. The power saving modes are enabled by setting bits 3 and 4 of parameter byte 0. See the *Debug Monitor*, *Mode – Set Parameter Bytes* command description later in this document for a description of the parameter bytes. The *uM-FPU64 IDE* can be used to set the parameter bytes using the *Functions>Set Parameters…* menu item.

Idle mode is normally enabled, since no special action is required to take advantage of the power savings.

The FPU will enter Idle mode when:

- the Idle mode parameter bit is set (Parameter byte 0, bit 4)
- Debug mode is disabled
- the device is idle

The FPU will exit Idle mode when:

- any event occurs that requires the FPU to resume execution

In Sleep mode supply current is significantly reduced (measured in microamps), but special action must be taken to wake up the FPU.

The FPU will enter Sleep mode when:

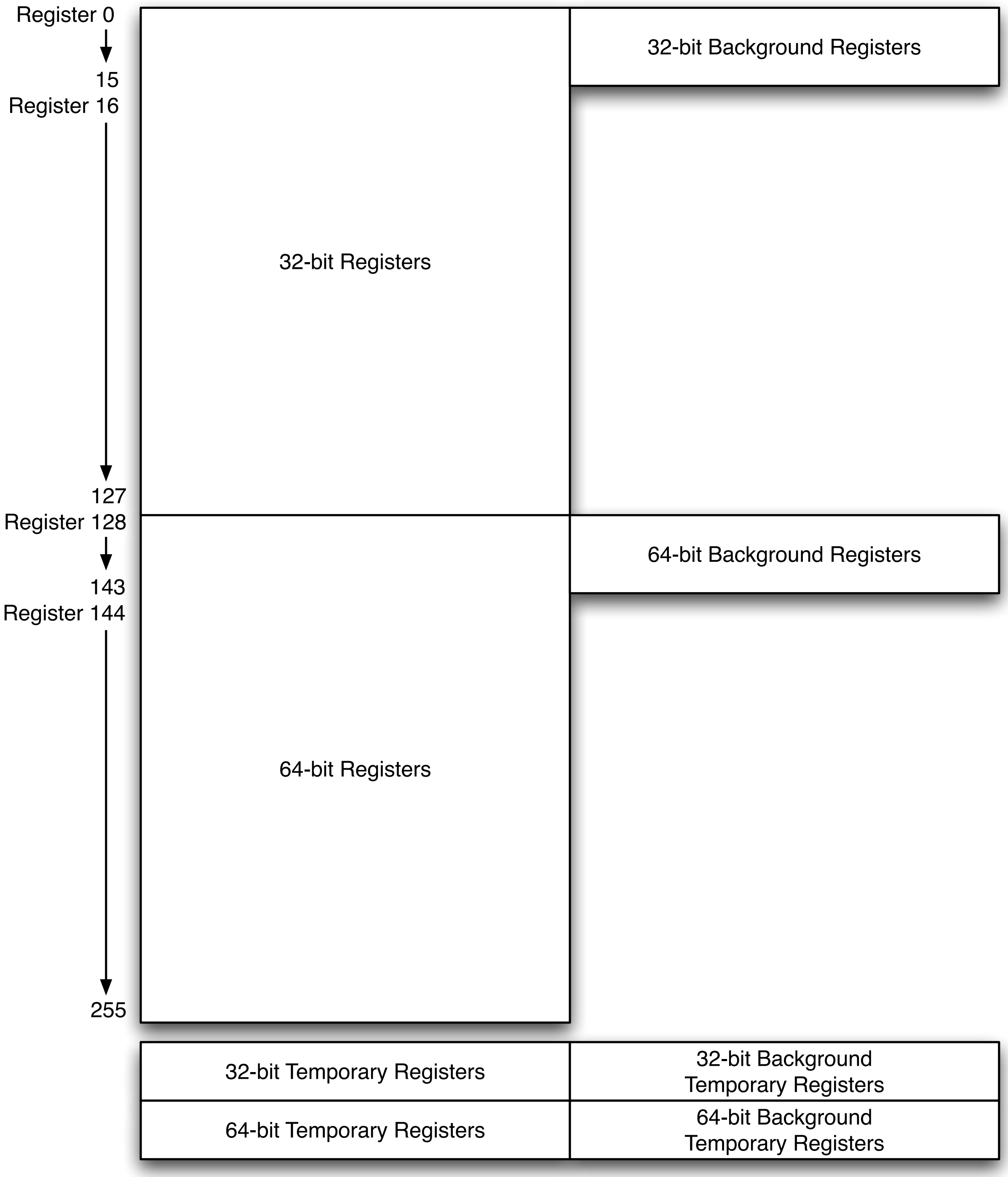
- the Sleep mode parameter bit is set (Parameter byte 0, bit 3)
- Debug mode is disabled
- the `/SS` pin is high
- the device is idle

The FPU will exit Sleep mode when:

- `/SS` goes low.
- a minimum delay of 500 usec is recommended from `/SS` low to the first data or clock bit is sent.

uM-FPU64 Register Map

The uM-FPU64 has 256 general purpose registers that can be used for storing floating point or integer values. Registers 0 to 127 are 32-bit registers, and registers 128 to 255 are 64-bit registers. Registers 0 to 15, registers 128 to 143, and the temporary registers, have separate registers for background processes. These registers are local to the background processes. Registers 16 to 127 and registers 144 to 255 are global and can be accessed by both foreground and background processes. The temporary registers are used by the `LEFT` and `RIGHT` instructions.


Register 0
15
Register 16
127
Register 128
143
Register 144
255
32-bit Registers
32-bit Background Registers
64-bit Registers
64-bit Background Registers
32-bit Temporary Registers
32-bit Background Temporary Registers
64-bit Temporary Registers
64-bit Background Temporary Registers

Digital Input/Output Pins

The 28-pin package has 9 digital I/O pins (`D0-D8`), and the 44-pin device has 23 digital I/O pins (`D0-D22`). Digital I/O pins are controlled by the `DIGIO` and `DEVIO` instructions (see the *uM-FPU64 Instruction Set* document for more details regarding these instructions). Parameter byte 6 is used to select one of the digital I/O pins as the external input. The external input is used with the `EXTLONG`, `EXTSET`, and `EXTWAIT` instructions, and as the external trigger for analog input.

Some of the digital I/O pins are 5V tolerant (`D0`, `D1`, `D12-D22`). For 5V input, no additional hardware setup is required. For 5V output, parameter bytes 3, 4, and 5 must be used to enable the open drain output for that pin, and a pull-up resistor must be added from the pin to the 5V supply.

Analog Input Pins

The 28-pin package has 6 analog input pins (`AN0-AN5`), and the 44-pin device has 9 analog input pins (`AN0-AN8`). Analog input is handled by the `ADCLOAD`, `ADCLONG`, `ADCMODE`, `ADCSCALE`, `ADCWAIT` and `ADCTRIG` instructions (see the *uM-FPU64 Instruction Set* for more details regarding these instructions). Analog pins `AN0` and `AN1` can be configured for use as `VREF+` and `VREF-`, instead of analog inputs. The analog pins `AN2-AN8` are shared by `D5-D11` and can be used for either analog input or digital I/O.

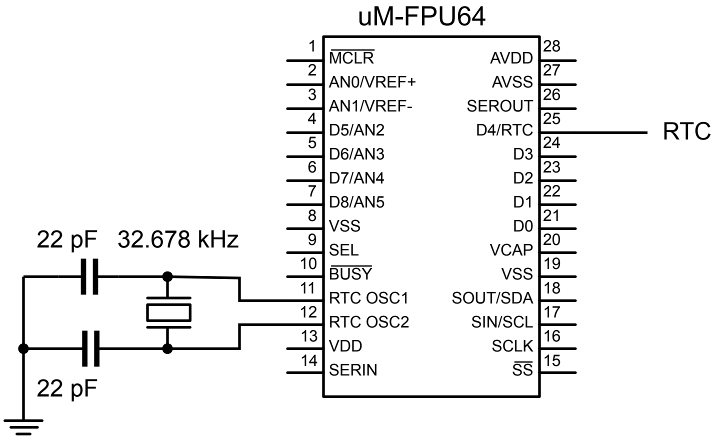
Local Peripheral Device Support

The uM-FPU64 provides support for local peripheral devices, including: RAM, FIFO buffers, 1-wire bus, I^{2}C bus, SPI bus, asynchronous serial connection (with hardware flow control), counters (with debounce and auto repeat), servo controllers, LCD display and VDrive2 USB storage. The interface to local peripheral devices is controlled by the `DEVIO` instruction (see the *uM-FPU64 Instruction Set* document for more details regarding this instruction). Local peripheral devices are assigned to digital I/O pins at run-time. `D0-D18` can be used by all devices, and `D19-D22` can be used by most devices. Devices that use multiple pins are assigned sequential pins.

Real-time Clock

The real-time clock can be used for keeping track of the date/time and for setting alarm events. The alarm events can be used to trigger processing events, or to output a signal on the `D4/RTC` pin. If the RTC output is enabled, it overrides any `DIGIO` or `DEVIO` settings for the `D4/RTC` pin. A 32.768 kHz (12.5 pF) crystal must be connected to the `RTC OSC1` and `RTC OSC2` pins to enable the real-time clock. The `RTC` instruction is used to control the real-time clock (see the *uM-FPU64 Instruction Set* for more details regarding this instruction). The RTC continues to run in Sleep mode.

Real-time Clock Connection

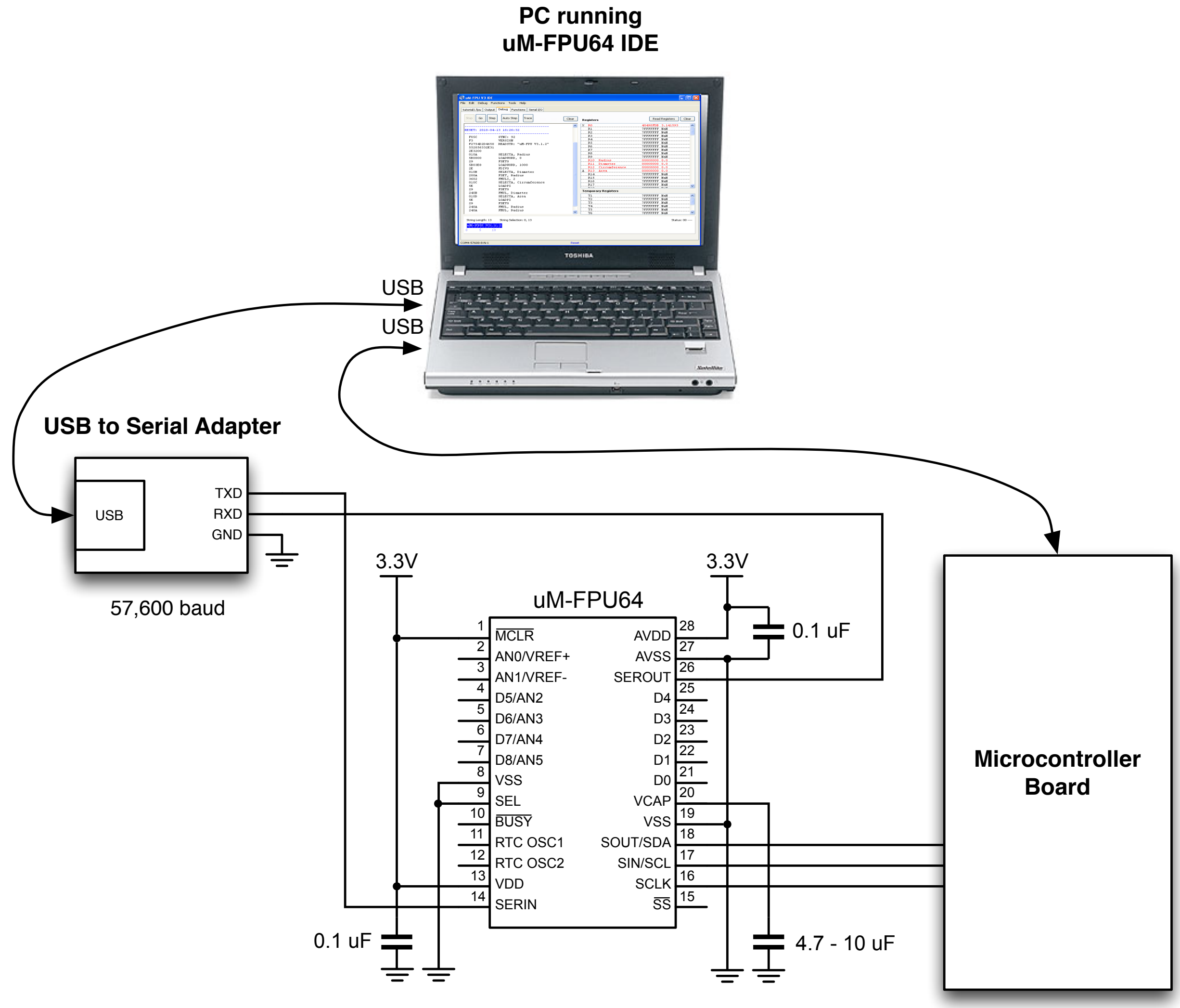


These connections are required in addition to the minimum recommended connections.

Using the SERIN and SEROUT Pins

The SERIN and SEROUT pins provide a serial interface for the built-in Debug Monitor, and can also be used for general purpose serial I/O when the Debug Monitor is not being used. The Debug Monitor communicates at 57,600 baud, using 8 data bits, no parity, one stop bit, and no flow control. The Debug Monitor is enabled if the SERIN pin is high when the FPU is Reset. Note: The idle state of an RS-232 connection will assert a high level on the SERIN pin, so provided the FPU is connected to an active idle RS-232 port when the FPU is reset, the Debug Monitor will be enabled. The SEROUT,0 instruction can also be used to enable/disable the Debug Monitor.

When the Debug Monitor is not being used, the serial I/O pins can be used for other purposes. The SEROUT,0 instruction is used to set the baud rate for the SERIN and SEROUT pins from 300 to 115,200 baud, using 8 data bits, no parity, one stop bit, and no flow control. The SERIN instruction supports reading serial data from the SERIN pin, and the SEROUT instruction supports sending serial data to the SEROUT pin. The uM-FPU64 chip includes support for NMEA sentence parsing, making it easy to connect to a GPS or other NMEA compliant device. The serial output can also be used to drive a serial LCD display or other serial device.



Debug Monitor

The built-in Debug Monitor provides support for displaying the contents of FPU registers, tracing the execution of FPU instructions, setting breakpoints for debugging, and programming Flash memory. A terminal emulator can be used to issue the serial commands manually, or the *uM-FPU64 IDE (Integrated Development Environment)* software can be used to communicate with the debug monitor through a fully featured graphical user interface. The *uM-FPU64 IDE* provides support for compiling, assembling, programming, and debugging. The *uM-FPU64 IDE* software and documentation can be downloaded from the Micromega website.

The debug monitor serial commands are described below. If the *uM-FPU64 IDE* is being used, these commands are handled automatically by the IDE.

Debug Monitor Serial Commands

Whenever the uM-FPU64 chip is reset and debug mode is enabled, the following message is sent to the serial output (`SEROUT` pin):

```
{RESET}
```

Commands are specified by typing an uppercase or lowercase character followed by a return key. The command is not processed (or echoed) until the return key is pressed. Once the return key is pressed, the command prompt and command are displayed, and the command is executed. If the command is not recognized, a question mark is displayed.

`B`	Break	Stop execution after next instruction
`D`	Debug Read	Reserved for use by IDE.
`F`	Read Flash	Read Flash memory (proprietaryASCII format). Used by IDE.
`G`	Go	Continue execution.
return	Single Step	Continue execution for one instruction.
`H`	Go/Step	Continue execution, or single step for one instruction.
`R`	Register	Display registers.
`S`	String	Display string, length and selection point.
`T`	Trace	Toggle trace mode on/off.
`V`	Version	Display version information.
`W`	Debug Write	(Reserved for use by IDE)
`X`	Change	Display all register that have changed.

Special commands are prefixed with a dollar sign. These commands are used to program the user functions and to check the contents of the FPU. They are not generally used when debugging a running application. The $M (Mode) and $P (Program) commands will reset the FPU on completion. The commands are listed below:

`$B`	Boot	Enter bootstrap loader. Used by IDE.
`$F`	Read Flash	Read Flash Memory (proprietary binary format). Used by IDE.
`$M`	Mode	Set mode parameters.
`$P`	Program	Program Flash memory.
`$S`	Checksum	Display checksum value.
`$X`	Reset	Reset the FPU.

B Break – Stop execution after next instruction.

This command is used to set hardware breakpoints, or to interrupt operation of the FPU.

This command causes an immediate break.

```
>B
```

This command sets the hardware breakpoints.

```
>B:mode,bp0Func,bp0Start,bp0End,bp1Func,bp1Addr,bp2Func,bp2Addr
```

where:

mode	Specifies the hardware breakpoint modes.
bit 0:	breakpoint 0 enabled
bit 1:	breakpoint 1 enabled
bit 2:	breakpoint 2 enabled
bit 4:	breakpoint 0 occurs always
bit 5:	breakpoint 0 occurs if address is not in range
bit 6:	breakpoint 0 occurs if call level is not equal to last break level
bit 7:	breakpoint 0 occurs if call level is less than last break level
bp0Func,bp0Start,bp0End	Breakpoint 0: function number, start address, end address
bp1Func,bp1Addr	Breakpoint 1: function number, address
bp2Func,bp2Addr	Breakpoint 2: function number, address

Hardware breakpoints can only be used with user-defined functions stored in Flash. A break will occur when any of the hardware breakpoint conditions is met. Breakpoint 0 has several different modes of operation, and is used by the *uM-FPU64 IDE* for source level single step execution. Breakpoints 1 and 2 will occur if they are enabled and the instruction at the address specified is about to be executed.

If an immediate break is specified, the break will not occur until after the next instruction is executed by the FPU. The debug monitor displays the hex value of the last instruction executed and any additional data. Entering another Break command, or simply pressing the return key, will single step to the next instruction. Entering the Go command will continue execution. Note: the *uM-FPU64 IDE* includes a disassembler that translates the trace bytes into a readable instruction sequence.

```
{BREAK}
>B
 0103              (i.e. SELECTA,3)
{BREAK}
>
 2001              (i.e. FSET,1)
{BREAK}
>
 3702              (i.e. FDIVI,2)
{BREAK}
>
 2403              (i.e. FMUL,3)
{BREAK}
>
```

F Flash – Read Flash memory (ASCII).

This command is used to read the contents of user accessible Flash memory in a proprietary format. The *uM-FPU64 IDE* uses this command to display the contents of user accessible Flash memory.

G Go – Continue execution.

This command is used to continue normal execution after a B (Break) command.

```
>G
```

H Go/Step – Continue execution, or single step.

This command is used to continue execution after a B (Break) command. If the debugger was in single step mode when the last breakpoint occurred, another single step is performed. If the debugger was in run mode when the last breakpoint occurred, normal execution is continued.

```
>H
```

### R	Registers – Display registers.

The Register command displays a header line showing the currently selected register A, register X, the internal status value, and if selected, matrix A, B and C. The current contents of all FPU registers are then displayed.

```
>R
{A=R0, X=R0, S=00, BA=B0, BX=B0, BS=00
R0-127:7FFFFFFF T1-8:7FFFFFFF B0-15:7FFFFFFF T17-24:7FFFFFFF
R128-255:7FFFFFFFFFFFFFFF T9-16:7FFFFFFFFFFFFFFF
B128-143:7FFFFFFFFFFFFFFF T25-32:7FFFFFFFFFFFFFFF}
```

### S	String – Display string, length and selection point.

The String command displays the current string buffer and selection point. The string length, selection start point and selection length are displayed, followed by the string. The following example shows an empty string.

```
>S
0,0,0                        foreground string

0,0,0                        background string
```

The following example shows the string buffer after the `VERSION` instruction has been executed.

```
>S
15,0,15                      foreground string
uM-FPU64 r401b3
0,0,0                        background string
```

### T	Trace – toggle trace mode on/off

The Trace command toggles the trace mode. The current state of the trace mode is displayed. Tracing can also be turned on and off by the user program with the `TRACEON` and `TRACEOFF` instructions.

When debug mode is enabled and the trace mode is on, each instruction that is executed by the FPU is displayed. Parameter byte 7 controls how tracing occurs. At reset, tracing is disabled unless *Trace on Reset - Foreground* is enabled. If *Trace Inside Functions* is enabled, all instructions will be traced. If *Trace Inside Functions* is not enabled, then only the function call will be traced, not the instructions inside the function. The trace status is maintained for each level of nested function calls. If tracing is enabled when a function is called, then disabled inside that function, tracing will be automatically enabled upon returning from the function. The *uM-FPU64 IDE* intends the trace display to shown the level of the function calls.

```
>T                           turn trace mode on
{TON,0,0,0000}

 7E01
{TON,1}                      if  Trace Inside Functions is enabled, the code inside the function is traced
 010A 7E02
{TON,2}
 5E 29 3702 80
{TON,1}
 80
{TON,0}
 1F00 FD F2"1.5707963"

>T                           turn trace mode off
{TOFF}
```

The *uM-FPU64 IDE* displays this information in the trace window as follows:

```
                 Trace On
   7E01          FCALL, sample
   010A          | SELECTA, 10
   7E02          | FCALL, getPi2
   5E            | | LOADPI
   29            | | FSET0
   3702          | | FDIVI, 2
   80            | | RET
   80            | RET
   1F00          FTOA, 0
   FD            SETREAD
   F2312E353730  READSTR: "1.5707963"
   3739363300
                 Trace Off
```

V Version – display version information

The Version command displays the version string for the uM-FPU64 chip, the currently selected interface, and the current clock speed. If the selected interface is I^{2}C the device address is also shown.

```
>V
uM-FPU64 r401, SPI 39.92 MHz

>V
uM-FPU64 r401, I2C C8 39.92 MHz
```

X Change – display changed registers

The Change command displays a header line showing the currently selected register A, register X, the internal status value for both the foreground and background processes. If selected, the values for matrix A, B and C are also shown. The current contents of all FPU registers that have changed since the last Change command (or Reset) are then displayed.

```
>X
{A=R0, X=R0, S=00, BA=B0, BX=B0, BS=00
R0:00004013}

>X
{A=R0, X=R0, S=00, BA=B0, BX=B0, BS=00}
```

$B Boot – Enter bootstrap loader.

This command is used to enter the bootstrap loader. The *uM-FPU64 IDE* uses this command to update firmware.

$F Flash – Read Flash memory (binary).

This command is used to read the contents of user accessible Flash memory in a proprietary format. The *uM-FPU64 IDE* uses this command to display the contents of user accessible Flash memory.

$S Checksum – display checksum value

This command displays a checksum for the uM-FPU64 program code and user-defined functions stored in Flash. This can be used to check that the chip is valid, or that a particular set of user-defined functions is installed.

```
>$S:00670EBD
```

$M Mode – Set Parameter Bytes

Thiscommand is used to set the parameter bytes that are stored in Flash memory. The parameter bytes are read at reset to determine various modes of operation. The mode command displays the current parameter values and the user is prompted to enter new values. The values are entered as hexadecimal values. The new values are programmed into Flash memory and the uM-FPU64 chip is Reset.

```
>$M
 10C800000000080F
:10CA00000000080F
```

Two hexadecimal digits represent each parameter byte. The mode parameter bytes are interpreted as follows:

Parameter Byte 0 - Mode

Bit	7	6	5	4	3	2	1	0
	B	D	-	I	S	P	Mode	

Bit 7 Break on Reset (if debug mode is enabled)
Bit 6 Disable Ready/Busy status on `SOUT` pin
Bit 5 Trace on Reset (if debug mode is enabled)
Bit 4 Idle Mode power saving enabled
Bit 3 Sleep Mode power saving enabled
Bit 2 PIC mode enabled (see PICMODE instruction)
Bits 1:0 Mode
00 – `SEL` pin determines interface mode (default)
if `SEL` pin = Low, SPI mode selected
if `SEL` pin = High, I²C mode selected
01 – I²C mode selected
1x – SPI mode selected

Parameter Byte 1 - I²C Address

Bit	7	6	5	4	3	2	1	0
	I2C Address							-

Bits 7:1 I²C Address
Bit 0 Unused

The 7-bit I²C device address is entered as a left justified 8-bit value. The last bit is ignored. If zero, the default address of (0xC8) is used.

Parameter Byte 2 - Auto-Start Function

Bit	7	6	5	4	3	2	1	0
	D	F	Function					

Bit 7 Debug mode
0 - use SERIN to select debug mode
SERIN = Low, Disable debug mode
SERIN = High, Enable debug mode
1 - Disable debug mode
Bit 6 Auto-start function call
0 - No function called

1 - Call the function specified by bits 5:0
Bit 5:0 Function number

Parameter byte 2 now specifies a user-defined function that can optionally be called when the chip is Reset. Mode parameter byte 2 is only checked at Reset if the `SEL` pin is Low. If both the `SEL` pin and SERIN pin are High at Reset, Debug Mode will always be entered. To use auto-start with the I^{2}C interface, the `SEL` pin must be Low at Reset, and the I^{2}C mode must be selected using mode 01 in mode parameter byte 0.

Parameter Byte 3 - Open Drain Control

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	-	-	D1	D0

Bits 7:2 Unused
Bit 1 Enable open drain for D1
Bit 0 Enable open drain for D0

Parameter Byte 4 - Open Drain Control

Bit	7	6	5	4	3	2	1	0
	D15	D14	D13	D12	-	-	-	-

Bit 7 Enable open drain for D15 pin (44-pin device)
Bit 6 Enable open drain for D14 pin (44-pin device)
Bit 5 Enable open drain for D13 pin (44-pin device)
Bit 4 Enable open drain for D12 pin (44-pin device)
Bits 3:0 Unused

Parameter Byte 5 - Open Drain Control

Bit	7	6	5	4	3	2	1	0
	S	D22	D21	D20	D19	D18	D17	D16

Bit 7 Enable open drain for SOUT pin
Bit 6 Enable open drain for D22 pin (44-pin device)
Bit 5 Enable open drain for D21 pin (44-pin device)
Bit 4 Enable open drain for D20 pin (44-pin device)
Bit 3 Enable open drain for D19 pin (44-pin device)
Bit 2 Enable open drain for D18 pin (44-pin device)
Bit 1 Enable open drain for D17 pin (44-pin device)
Bit 0 Enable open drain for D16 pin (44-pin device)

Parameter Byte 6 - External Input Pin

Bit	7	6	5	4	3	2	1	0
	-	-	F	External Input				

Bits 7:6 Unused
Bit 5 Active Edge
0 - Rising Edge
1 - Falling Edge
Bit 4:0 Digital input pin

0 - 8	28-pin device
0 - 22	44-pin device

Parameter Byte 7 - Debug Mode

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	BI	BR	FI	FR

Bits 7:4	Unused
Bit 3	Trace Inside Functions - Background process
Bit 2	Trace on Reset - Background process
Bit 1	Trace Inside Functions - Foreground process
Bit 0	Trace on Reset - Foreground process

These settings are only active if debug mode is enabled.

$P Program – program user function memory

The Program command is used to program the user function memory. Once in program mode, the FPU looks for valid Intel Hex format records. The records must have an address between 0x0000 and 0x03C0, start on a 64-byte boundary, and have a length of 1 to 64 bytes. The data is not echoed, but an acknowledge character is sent after each record. The acknowledge characters are as follows:

`+`	The record was programmed successfully.
`F`	A format error occurred.
`A`	An address error occurred.
`C`	A checksum error occurred.
`P`	A programming error occurred.

The *uM-FPU IDE* program (or another PC based application program) would normally be used to send the required data for the program command. (See documentation for the *uM-FPU IDE* software.) To exit the program mode, an escape character is sent. The program command will reset the FPU on exit.

```
>$P
{*** PROGRAM MODE ***}
+++

{RESET}
```

$X Reset – Reset the FPU

The uM-FPU64 chip is reset.

```
>$X

{RESET}
```

Debug Instructions

There are several instructions that are designed to work in conjunction with the debug monitor. If the debug monitor is not enabled, these commands are `NOPs`. The instructions are as follows:

BREAK

When the `BREAK` instruction is encountered, execution stops, and the debug monitor is entered. Execution will only resume when a Go command is issued entered with the debug monitor.

TRACEOFF

Turns the debug trace mode off.

TRACEON

Turns the debug trace mode on. All instructions will be traced on the debug terminal until the trace mode is turned off by a `TRACEOFF` instruction or is turned off using the debug monitor.

TRACESTR

Displays a trace string to the debug monitor output. This can be useful for keeping track of a debug session. Trace strings are always output; they are not affected by the trace mode.

TRACEREG

Displays a trace string with the value of the register to the debug monitor output. Trace registers are always output; they are not affected by the trace mode.

Error Status

Bit	15	14	13	12	11	10	9	8
	Error Flag	-	-	-	-	-	-	Alloc Error

Bit	7	6	5	4	3	2	1	0
	Device Error	Paren Level	Function Level	Address	XOP Call	Function Call	Under run	Over run

Bit 15 Error Flag
At least one error bit was been set.

Bit 8 Allocation Error
A memory allocation failed because the number of bytes requested were not available in the dynamic allocation area.

Bit 7 Device Error
A `DEVIO,`*`device`*`,LOAD_DEVICE,…` instruction failed because the loadable device was not programmed into Flash memory.

Bit 6 Parenthesis Level Error
The maximum temporary register level (8) was exceeded by a `LEFT` instruction. The temporary register level is reset to zero. `RIGHT` instructions will return NaN.

Bit 5 Function Level Error
The maximum function call level (16) was exceeded. The `FCALL` is aborted and NaN is returned in register 0 (32-bit) or register 128 (64-bit).

Bit 4 Address Error
An address error occurred within an `XOP` instruction.

Bit 3 XOP Call Error
An `XOP` instruction was executed that specified an extended instruction that was not programmed into Flash memory.

Bit 2 Function Call Error
An `FCALL` instruction was executed that specified a function that was not programmed

into Flash memory.

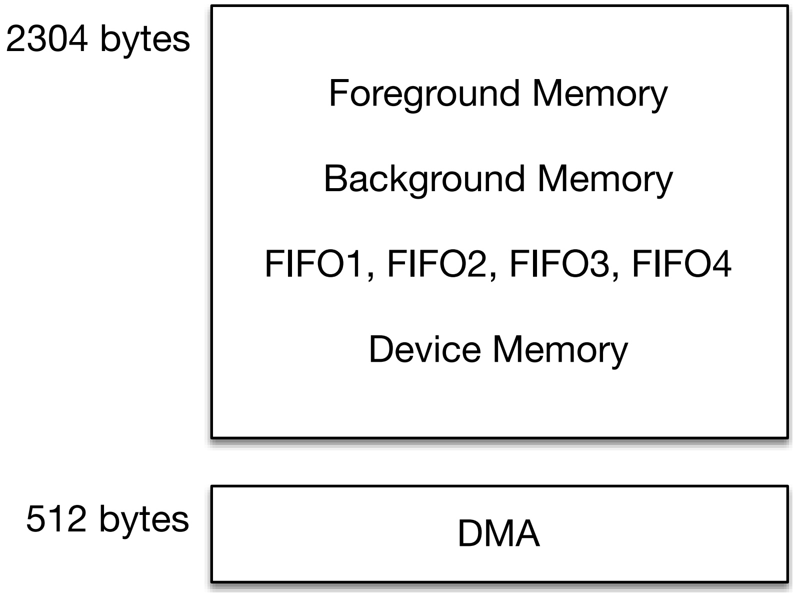
Bit 1 **Underrun Error**

An instruction required additional bytes that were not received before a 5 msec timeout. A data byte of zero was returned.

Bit 0 **Overrun Error**

The maximum size of the instruction buffer (256 bytes) was exceeded. One or more data bytes were lost.

RAM and DMA Memory



The total amount of available RAM is 2304 bytes, which is split into a foreground memory, background memory, FIFO1, FIFO2, FIFO3, and FIFO4, dynamic allocation pool. At Reset, the default allocation of RAM is as follows:

Foreground	2304 bytes
Background	0 bytes
FIFO1	0 bytes
FIFO2	0 bytes
FIFO3	0 bytes
FIFO4	0 bytes
Dynamic Allocation	0 bytes

The memory allocation can be changed using the `DEVIO,MEM,ALLOCATE` instruction. Foreground memory is only accessible from foreground tasks, and Background memory is only available from background tasks. FIFO memory can be allocated by the following instructions:

```
DEVIO,MEM,ALLOCATE
DEVIO,FIFOn,ALLOC_MEM
DEVIO,FIFOn,ALLOC_MEMR
```

The dynamic allocation area is used by loadable devices. The amount of memory required by a loadable device is specified in the device *xopdev* file and is allocated when the device is loaded. Devices are loaded using the `DEVIO,`*`device`*`,LOAD_DEVICE,`*`xopdev`* instruction.

DMA memory is used directly by the analog-to-digital converter. The amount of memory used is determined by the `ADCMODE` instruction.

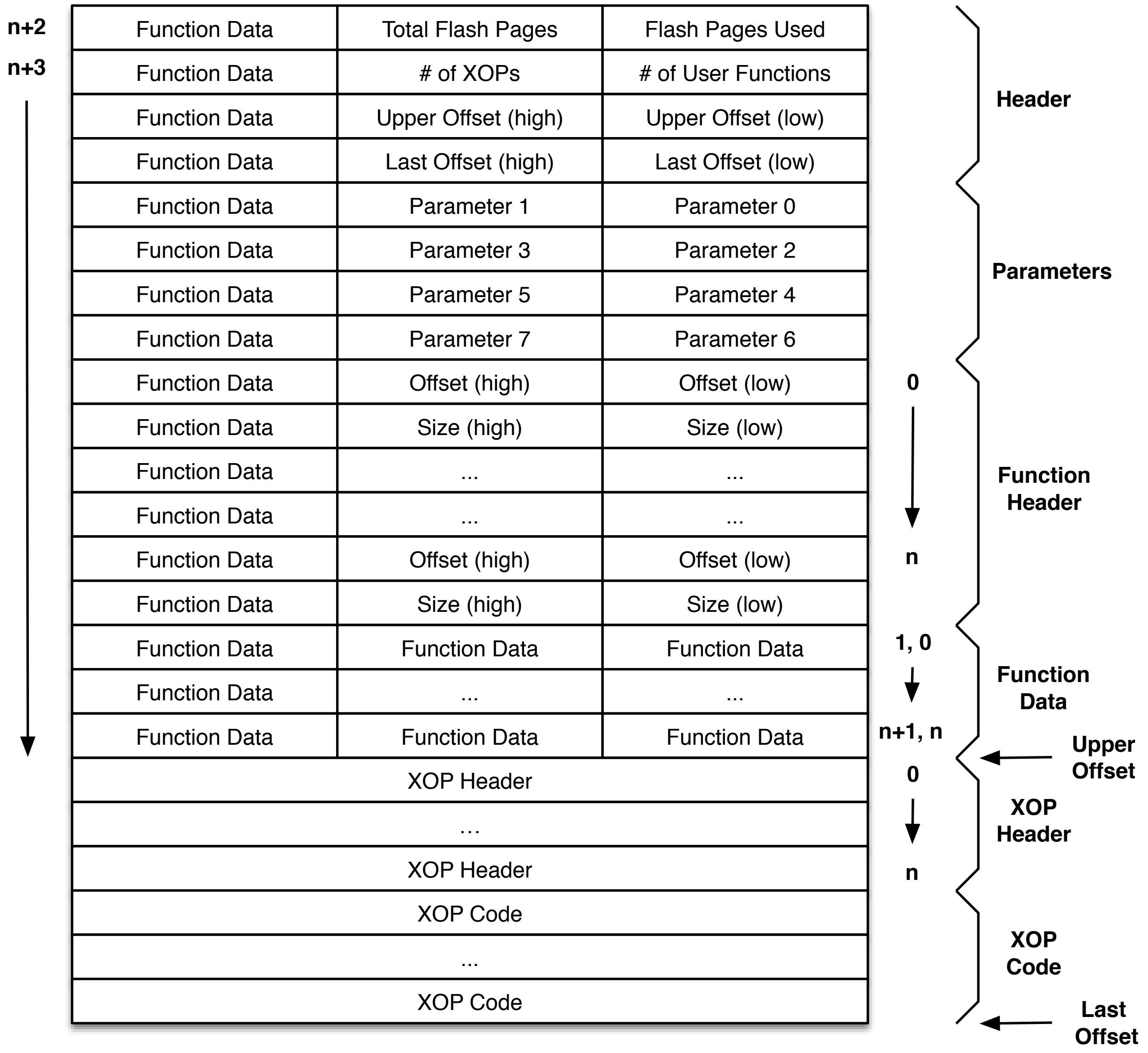
DMA memory can also be accessed using the `SETIND`, `ADDIND`, `COPYIND`, `LOADIND`, `RDIND`, `SAVEIND`, and `WRIND` instructions.

Flash Memory

There are 6144 bytes of Flash memory reserved on the uM-FPU64 for storing user-defined functions and the mode parameters. Up to 64 user-defined functions can be stored in Flash memory. User-defined functions have the advantage of conserving space on the microcontroller and greatly reducing the communications overhead between the microcontroller and the FPU. In addition, certain instructions (e.g. `BRA`, `JMP`, `TABLE`, `POLY`) are only valid in user-defined functions. The `FCALL` instruction is used to call the user-defined functions stored in Flash memory. The Busy condition remains set while all of the instructions in the called function execute.

The *uM-FPU64 IDE* provides support for defining and programming user-defined functions. Refer to *uM-FPU64 IDE* documentation for details.

Flash Memory Layout

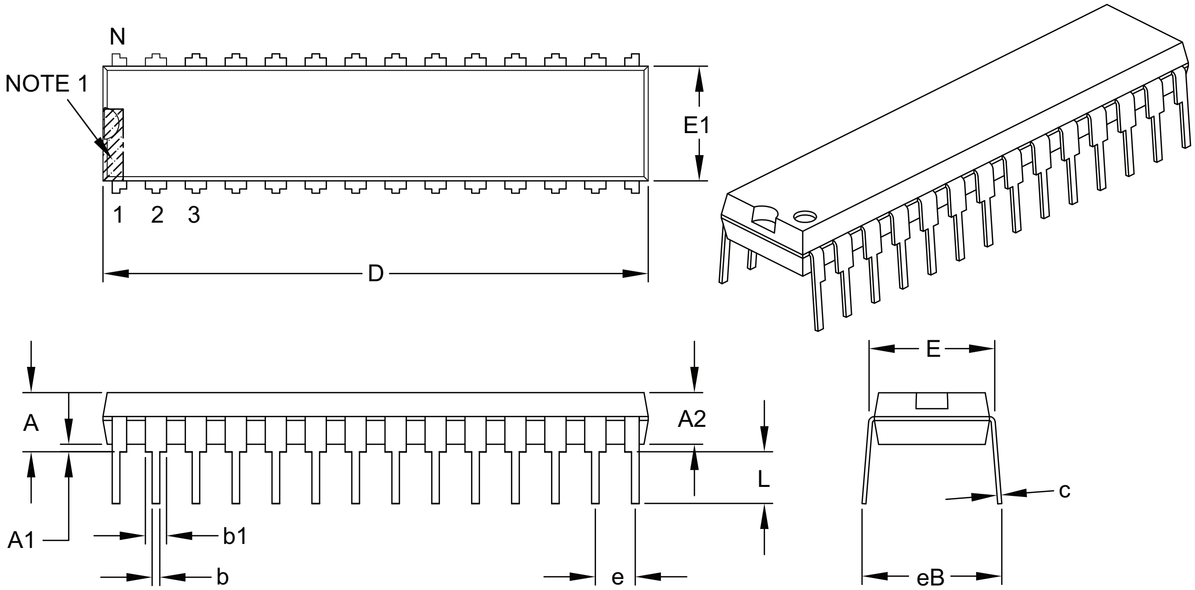


Function Data	Total Flash Pages	Flash Pages Used
Function Data	# of XOPs	# of User Functions
Function Data	Upper Offset (high)	Upper Offset (low)
Function Data	Last Offset (high)	Last Offset (low)
Function Data	Parameter 1	Parameter 0
Function Data	Parameter 3	Parameter 2
Function Data	Parameter 5	Parameter 4
Function Data	Parameter 7	Parameter 6
Function Data	Offset (high)	Offset (low)
Function Data	Size (high)	Size (low)
Function Data	...	...
Function Data	...	...
Function Data	Offset (high)	Offset (low)
Function Data	Size (high)	Size (low)
Function Data	Function Data	Function Data
Function Data	...	...
Function Data	Function Data	Function Data
XOP Header		
…		
XOP Header		
XOP Code		
...		
XOP Code		

Firmware Updates

A bootstrap loader allows for firmware upgrades in the field. The *uM-FPU64 IDE* can used to upgrade the uM-FPU64 firmware using the *Tools>Upgrade Firmware…* menu command.

PDIP-28 Through-Hole Package



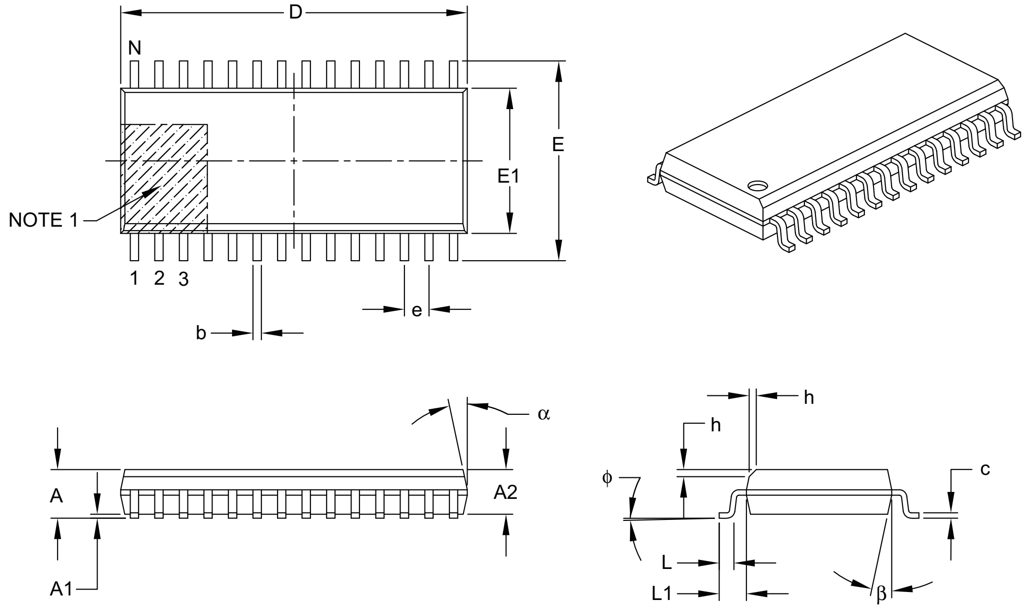
Units		INCHES		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N	28		
Pitch	e	.100 BSC		
Top to Seating Plane	A	–	–	.200
Molded Package Thickness	A2	.120	.135	.150
Base to Seating Plane	A1	.015	–	–
Shoulder to Shoulder Width	E	.290	.310	.335
Molded Package Width	E1	.240	.285	.295
Overall Length	D	1.345	1.365	1.400
Tip to Seating Plane	L	.110	.130	.150
Lead Thickness	c	.008	.010	.015
Upper Lead Width	b1	.040	.050	.070
Lower Lead Width	b	.014	.018	.022
Overall Row Spacing §	eB	–	–	.430

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

SOIC-28 Surface Mount Package



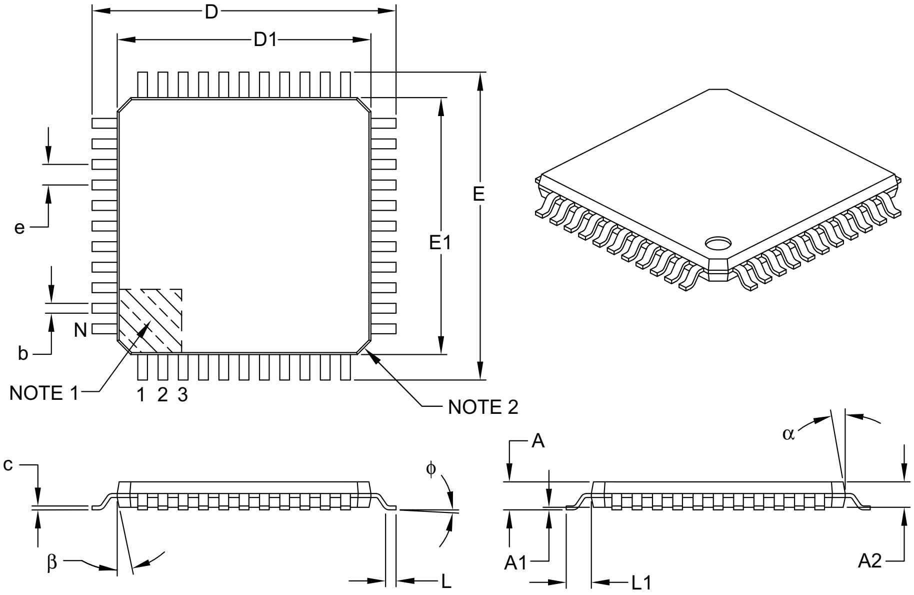
Units		MILLMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N	28		
Pitch	e	1.27 BSC		
Overall Height	A	–	–	2.65
Molded Package Thickness	A2	2.05	–	–
Standoff §	A1	0.10	–	0.30
Overall Width	E	10.30 BSC		
Molded Package Width	E1	7.50 BSC		
Overall Length	D	17.90 BSC		
Chamfer (optional)	h	0.25	–	0.75
Foot Length	L	0.40	–	1.27
Footprint	L1	1.40 REF		
Foot Angle Top	φ	0°	–	8°
Lead Thickness	c	0.18	–	0.33
Lead Width	b	0.31	–	0.51
Mold Draft Angle Top	α	5°	–	15°
Mold Draft Angle Bottom	β	5°	–	15°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.
REF: Reference Dimension, usually without tolerance, for information purposes only.

TQFP-44 Surface Mount Package



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Leads	N	44		
Lead Pitch	e	0.80 BSC		
Overall Height	A	–	–	1.20
Molded Package Thickness	A2	0.95	1.00	1.05
Standoff	A1	0.05	–	0.15
Foot Length	L	0.45	0.60	0.75
Footprint	L1	1.00 REF		
Foot Angle	φ	0°	3.5°	7°
Overall Width	E	12.00 BSC		
Overall Length	D	12.00 BSC		
Molded Package Width	E1	10.00 BSC		
Molded Package Length	D1	10.00 BSC		
Lead Thickness	c	0.09	–	0.20
Lead Width	b	0.30	0.37	0.45
Mold Draft Angle Top	α	11°	12°	13°
Mold Draft Angle Bottom	β	11°	12°	13°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Chamfers at corners are optional; size may vary.
3. Dimensions D1 and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.25 mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.
REF: Reference Dimension, usually without tolerance, for information purposes only.

Absolute Maximum Ratings

Parameter	Minimum	Typical	Maximum	Units
Storage Temperature	-65	-	+150	° Celsius
Ambient Temperature with Power Applied	-40	-	+85	° Celsius
Supply Voltage on VDD relative to VSS	-0.3	-	+4.0	V
Voltage on any pin that is not 5V tolerant with respect to VSS	-0.3	-	VDD+0.3	V
Voltage on any pin that is 5V tolerant with respect to VSS	-0.3	-	5.6	V
Maximum Current out of VSS pin			300	mA
Maximum Current into VDD pin			250	mA
Maximum Current sourced by any I/O pin			4	mA
Maximum Current sunk by any I/O pin			4	mA
Maximum Current sourced by all I/O pins			200	mA
Maximum Current sunk by all I/O pins			200	mA

DC Characteristics

Parameter	Minimum	Typical	Maximum	Units
I/O Pin Input Low Voltage	VSS	-	0.2 VDD	V
I/O Pin Input High Voltage on any pin that is not 5V tolerant	0.7 VDD	-	VDD	V
I/O Pin Input High Voltage on any pin that is 5V tolerant	0.7 VDD		5.5	V
AVDD	greater of VDD - 0.3 or 3.0	-	lesser of VDD + 0.3 or 3.6	V
AVSS	VSS - 0.3		VSS + 0.3	V
-VREF	AVSS	-	AVDD-2.7	V
+VREF	AVSS+2.7	-	AVDD	V
voltage between +VREF and -VREF	2.7	-	3.6	V
Operating MIPS			40	MIPS
Supply Current - Full speed		60	90	mA
Supply Current - Idle mode		28		mA
Supply Current - Sleep mode		< 1		mA

Further Information

Check the Micromega website at www.micromegacorp.com

Appendix A
uM-FPU64 Instruction Summary

Instruction	Opcode	Arguments	Returns	Description
NOP	00			No Operation
SELECTA	01	*register*		Select register A, A = *register*
SELECTX	02	*register*		Select register X, X = *register*
CLR	03	*register*		reg[*register*] = 0
CLRA	04			reg[A] = 0
CLRX	05			reg[X] = 0, X = X + 1
CLR0	06			reg[0 I 128] = 0
COPY	07	*register1, register2*		reg[*register2*] = reg[*register1*]
COPYA	08	*register*		reg[*register*] = reg[A]
COPYX	09	*register*		reg[*register*] = reg[X], X = X + 1
LOAD	0A	*register*		reg[0 I 128] = reg[*register*]
LOADA	0B			reg[0 I 128] = reg[A]
LOADX	0C			reg[0 I 128] = reg[X], X = X + 1
ALOADX	0D			reg[A] = reg[X], X = X + 1
XSAVE	0E	*register*		reg[X] = reg[*register*], X = X + 1
XSAVEA	0F			reg[X] = reg[A], X = X + 1
COPY0	10	*register*		reg[*register*] = reg[0 I 128]
LCOPYI	11	*signedByte, register*		reg[*register*] = long(*signedByte*)
SWAP	12	*register1, register2*		Swap reg[*register1*] and reg[*register2*]
SWAPA	13	*register*		Swap reg[*register*] and reg[A]
LEFT	14			Left parenthesis
RIGHT	15			Right parenthesis
FWRITE	16	*register, float32Value*		Write 32-bit floating point to reg[*register*]
FWRITEA	17	*float32Value*		Write 32-bit floating point to reg[A]
FWRITEX	18	*float32Value*		Write 32-bit floating point to reg[X]
FWRITE0	19	*float32Value*		Write 32-bit floating point to reg[0 I 128]
FREAD	1A	*register*	*float32Value*	Read 32-bit floating point from reg[*register*]
FREADA	1B		*float32Value*	Read 32-bit floating point from reg[A]
FREADX	1C		*float32Value*	Read 32-bit floating point from reg[X]
FREAD0	1D		*float32Value*	Read 32-bit floating point from reg[0 I 128]
ATOF	1E	*string*		Convert ASCII to floating point
FTOA	1F	*format*		Convert floating point to ASCII
FSET	20	*register*		reg[A] = reg[*register*]
FADD	21	*register*		reg[A] = reg[A] + reg[*register*]
FSUB	22	*register*		reg[A] = reg[A] - reg[*register*]
FSUBR	23	*register*		reg[A] = reg[*register*] - reg[A]
FMUL	24	*register*		reg[A] = reg[A] * reg[*register*]
FDIV	25	*register*		reg[A] = reg[A] / reg[*register*]
FDIVR	26	*register*		reg[A] = reg[*register*] / reg[A]
FPOW	27	*register*		reg[A] = reg[A] ** reg[*register*]

FCMP	28	*register*		Compare reg[A] and reg[*register*], and set status
FSET0	29			reg[A] = reg[0 l 128]
FADD0	2A			reg[A] = reg[A] + reg[0 l 128]
FSUB0	2B			reg[A] = reg[A] - reg[0 l 128]
FSUBR0	2C			reg[A] = reg[0] - reg[A]
FMUL0	2D			reg[A] = reg[A] * reg[0 l 128]
FDIV0	2E			reg[A] = reg[A] / reg[0 l 128]
FDIVR0	2F			reg[A] = reg[0 l 128] / reg[A]
FPOW0	30			reg[A] = reg[A] ** reg[0 l 128]
FCMP0	31			Compare reg[A] and reg[0 l 128]
FSETI	32	*signedByte*		reg[A] = float(*signedByte*)
FADDI	33	*signedByte*		reg[A] = reg[A] - float(*signedByte*)
FSUBI	34	*signedByte*		reg[A] = reg[A] - float(*signedByte*)
FSUBRI	35	*signedByte*		reg[A] = float(*signedByte*) - reg[A]
FMULI	36	*signedByte*		reg[A] = reg[A] * float(*signedByte*)
FDIVI	37	*signedByte*		reg[A] = reg[A] / float(*signedByte*)
FDIVRI	38	*signedByte*		reg[A] = float(*signedByte*) / reg[A]
FPOWI	39	*signedByte*		reg[A] = reg[A] ** *signedByte*
FCMPI	3A	*signedByte*		Compare reg[A] and float(*signedByte*), and set floating point status
FSTATUS	3B	*register*		Set floating point status for *register*
FSTATUSA	3C			Set floating point status for reg[A]
FCMP2	3D	*register1*, *register2*		Compare reg[*register1*] and reg[*register2*], and set floating point status
FNEG	3E			reg[A] = -reg[A]
FABS	3F			reg[A] = l reg[A] l
FINV	40			reg[A] = 1 / reg[A]
SQRT	41			reg[A] = sqrt(reg[A])
ROOT	42	*register*		reg[A] = root(reg[A], reg[*register*])
LOG	43			reg[A] = log(reg[A])
LOG10	44			reg[A] = log10(reg[A])
EXP	45			reg[A] = exp(reg[A])
EXP10	46			reg[A] = exp10(reg[A])
SIN	47			reg[A] = sin(reg[A])
COS	48			reg[A] = cos(reg[A])
TAN	49			reg[A] = tan(reg[A])
ASIN	4A			reg[A] = asin(reg[A])
ACOS	4B			reg[A] = acos(reg[A])
ATAN	4C			reg[A] = atan(reg[A])
ATAN2	4D	*register*		reg[A] = atan2(reg[A], reg[*register*])
DEGREES	4E			reg[A] = degrees(reg[A])
RADIANS	4F			reg[A] = radians(reg[A])
FMOD	50	*register*		reg[A] = reg[A] MOD reg[*register*]
FLOOR	51			reg[A] = floor(reg[A])
CEIL	52			reg[A] = ceil(reg[A])
ROUND	53			reg[A] = round(reg[A])
FMIN	54	*register*		reg[A] = min(reg[A], reg[*register*])
FMAX	55	*register*		reg[A] = max(reg[A], reg[*register*])

FCNV	56	*conversion*		reg[A] = conversion(reg[A])
FMAC	57	*register1, register2*		reg[A] = reg[A] + (reg[*register1*] * reg[*register2*])
FMSC	58	*register1, register2*		reg[A] = reg[A] - (reg[*register1*] * reg[*register2*])
LOADBYTE	59	*signedByte*		reg[0 I 128] = float(*signedByte*)
LOADUBYTE	5A	*unsignedByte*		reg[0 I 128] = float(*unsignedByte*)
LOADWORD	5B	*signedWord*		reg[0 I 128] = float(*signedWord*)
LOADUWORD	5C	*unsignedWord*		reg[0 I 128] = float(un*signedWord*)
LOADE	5D			reg[0 I 128] = 2.7182818
LOADPI	5E			reg[0 I 128] = 3.1415927
FCOPYI	5F	*signedByte, register*		reg[*register*] = float(*signedByte*)
FLOAT	60			reg[A] = float(reg[A])
FIX	61			reg[A] = fix(reg[A])
FIXR	62			reg[A] = fix(round(reg[A]))
FRAC	63			reg[A] = fraction(reg[A])
FSPLIT	64			reg[A] = integer(reg[A]), reg[0 I 128] = fraction(reg[A])
SELECTMA	65	*register, rows,columns*		Select matrix A, starting at *register*. size = *rows* x *columns*
SELECTMB	66	*register, rows,columns*		Select matrix B, starting at *register*. size = *rows* x *columns*
SELECTMC	67	*register, rows,columns*		Select matrix C, starting at *register*. size = *rows* x *columns*
LOADMA	68	*row,column*		reg[0] = Matrix A[*row*, *column*]
LOADMB	69	*row,column*		reg[0] = Matrix B[*row*, *column*]
LOADMC	6A	*row,column*		reg[0] = Matrix C[*row*, *column*]
SAVEMA	6B	*row,column*		Matrix A[*row*, *column*] = reg[0]
SAVEMB	6C	*row,column*		Matrix B[*row*, *column*] = reg[0]
SAVEMC	6D	*row,column*		Matrix C[*row*, *column*] = reg[0]
MOP	6E	*action*		Matrix/Vector operation
FFT	6F	*action*		Fast Fourier Transform
WRIND	70	*dataType,pointer ,count,value1… valueN*		Write multiple data values to indirect pointer
RDIND	71	*dataType,pointer ,count*	*value1…valueN*	Read multiple data values from indirect pointer
DWRITE	72	*register, value64*		Write 64-bit value
DREAD	73	*register*	*value64*	Read 64-bit value
LBIT	74	*action, register*		Bit Clear, Set, Toggle, Test
SETIND	77	*type,{register| address}*		Set indirect pointer
ADDIND	78	*register, unsignedByte*		Add to indirect pointer
COPYIND	79	*register1, register2, register3*		Copy using indirect pointers
LOADIND	7A	*register*		Load reg[0 I 128] using indirect pointer
SAVEIND	7B	*register*		Save reg[A] using indirect pointer

INDA	7C	*register*		Select register A using reg[*register*] value
INDX	7D	*register*		Select register X using reg[*register*] value
FCALL	7E	*function*		Call user-defined function in Flash
EVENT	7F	*action {,function}*		Background Events
RET	80			Return from user-defined function
BRA	81	*relativeOffset*		Unconditional branch
BRA,cc	82	*conditionCode, relativeOffset*		Conditional branch
JMP	83	*absoluteOffset*		Unconditional jump
JMP,cc	84	*conditionCode, absoluteOffset*		Conditional jump
TABLE	85	*tableSize, tableItem1... tableItemN*		Table lookup
FTABLE	86	*conditionCode, tableSize, tableItem1... tableItemN*		Floating point reverse table lookup
LTABLE	87	*conditionCode, tableSize, tableItem1... tableItemN*		Long integer reverse table lookup
POLY	88	count, float32Value1... float32ValueN		reg[A] = nth order polynomial
GOTO	89	*register*		Computed GOTO
RET,cc	8A	*conditionCode*		Conditional return from user-defined function
LWRITE	90	*register, int32Value*		Write 32-bit long integer to reg[*register*]
LWRITEA	91	*int32Value*		Write 32-bit long integer to reg[A]
LWRITEX	92	*int32Value*		Write 32-bit long integer to reg[X], X = X + 1
LWRITE0	93	*int32Value*		Write 32-bit long integer to reg[0 | 128]
LREAD	94	*register*	*int32Value*	Read 32-bit long integer from reg[*register*]
LREADA	95		*int32Value*	Read 32-bit long value from reg[A]
LREADX	96		*int32Value*	Read 32-bit long integer from reg[X], X = X + 1
LREAD0	97		*int32Value*	Read 32-bit long integer from reg[0 | 128]
LREADBYTE	98		*byteValue*	Read lower 8 bits of reg[A]
LREADWORD	99		*wordValue*	Read lower 16 bits reg[A]
ATOL	9A	*string*		Convert ASCII to long integer
LTOA	9B	*format*		Convert long integer to ASCII
LSET	9C	*register*		reg[A] = reg[*register*]
LADD	9D	*register*		reg[A] = reg[A] + reg[*register*]
LSUB	9E	*register*		reg[A] = reg[A] - reg[*register*]
LMUL	9F	*register*		reg[A] = reg[A] * reg[*register*]
LDIV	A0	*register*		reg[A] = reg[A] / reg[*register*] reg[0 | 128] = remainder

LCMP	A1	*register*		Signed compare reg[A] and reg[*register*], and set status
LUDIV	A2	*register*		reg[A] = reg[A] / reg[*register*] reg[0 l 128] = remainder
LUCMP	A3	*register*		Unsigned compare reg[A] and reg[*register*], and set long integer status
LTST	A4	*register*		Test reg[A] AND reg[*register*], and set long integer status
LSET0	A5			reg[A] = reg[0]
LADD0	A6			reg[A] = reg[A] + reg[0 l 128]
LSUB0	A7			reg[A] = reg[A] - reg[0 l 128]
LMUL0	A8			reg[A] = reg[A] * reg[0 l 128]

LDIV0	A9			reg[A] = reg[A] / reg[0 l 128]
				reg[0] = remainder

LCMP0	AA			Signed compare reg[A] and reg[0 l 128], and set long integer status

LUDIV0	AB			reg[A] = reg[A] / reg[0 l 128]
				reg[0] = remainder

LUCMP0	AC			Unsigned compare reg[A] and reg[0 l 128], and set long integer status
LTST0	AD			Test reg[A] AND reg[0 l 128], and set long integer status
LSETI	AE	*signedByte*		reg[A] = long(*signedByte*)
LADDI	AF	*signedByte*		reg[A] = reg[A] + long(*signedByte*)
LSUBI	B0	*signedByte*		reg[A] = reg[A] - long(*signedByte*)
LMULI	B1	*signedByte*		reg[A] = reg[A] * long(*signedByte*)
LDIVI	B2	*signedByte*		reg[A] = reg[A] / long(*signedByte*) reg[0 l 128] = remainder
LCMPI	B3	*signedByte*		Signed compare reg[A] - long(*signedByte*), and set long integer status
LUDIVI	B4	*unsignedByte*		reg[A] = reg[A] / long(*unsignedByte*) reg[0 l 128] = remainder
LUCMPI	B5	*unsignedByte*		Unsigned integer compare reg[A] and long(*unsignedByte*), and set status
LTSTI	B6	*unsignedByte*		Test reg[A] AND long(*unsignedByte*), and set long integer status
LSTATUS	B7	*register*		Set long integer status for reg[*register*]
LSTATUSA	B8			Set long integer status for reg[A]
LCMP2	B9	*register1, register2*		Signed integer compare reg[*register1*], reg[*register2*], and set status
LUCMP2	BA	*register1, register2*		Unsigned integer compare reg[*register1*], reg[*register2*], and set status
LNEG	BB			reg[A] = -reg[A]
LABS	BC			reg[A] = absolute value (reg[A])
LINC	BD	*register*		reg[*register*] = reg[*register*] + 1
LDEC	BE	*register*		reg[*register*] = reg[*register*] - 1
LNOT	BF			reg[A] = NOT reg[A]
LAND	C0	*register*		reg[A] = reg[A] AND reg[*register*]
LOR	C1	*register*		reg[A] = reg[A] OR reg[*register*]

LXOR	C2	*register*		reg[A] = reg[A] XOR reg[*register*]
LSHIFT	C3	*register*		reg[A] = reg[A] shift reg[*register*]
LMIN	C4	*register*		reg[A] = min(reg[A], reg[*register*])
LMAX	C5	*register*		reg[A] = max(reg[A], reg[*register*])
LONGBYTE	C6	*signedByte*		reg[0 I 128] = long(*signedByte*)
LONGUBYTE	C7	*unsignedByte*		reg[0 I 128] = long(*unsignedByte*)
LONGWORD	C8	*signedWord*		reg[0 I 128] = long(*signedWord*)
LONGUWORD	C9	*unsignedWord*		reg[0 I 128] = long(*unsignedWord*)
LSHIFTI	CA	*unsignedByte*		reg[A] = reg[A] shift *unsignedByte*
LANDI	CB	*unsignedByte*		reg[A] = reg[A] AND *unsignedByte*
LORI	CC	*unsignedByte*		reg[A] = reg[A] OR *unsignedByte*
SETSTATUS	CD	*status*		Set status byte
SEROUT	CE	*action,* *{baud}|{string}*		Serial output
SERIN	CF	*action*		Serial input
DIGIO	D0	*action,{mode}*		Digital I/O
ADCMODE	D1	*mode*		Set A/D trigger mode
ADCTRIG	D2			A/D manual trigger
ADCSCALE	D3	*channel*		ADCscale[ch] = reg[0]
ADCLONG	D4	*channel*		reg[0] = ADCvalue[*channel*]
ADCLOAD	D5	*channel*		reg[0] = float(ADCvalue[*channel*]) * ADCscale[*channel*]
ADCWAIT	D6			wait for next A/D sample
TIMESET	D7			time = reg[0]
TIMELONG	D8			reg[0] = time (long integer)
TICKLONG	D9			reg[0] = ticks (long integer)
DEVIO	DA	*device,action* *{,…}*		Device I/O
DELAY	DB	*period*		Delay (in milliseconds)
RTC	DC	*action*		Real-time Clock
SETARGS	DD			Enable FCALL argument loading
	DE			
	DF			
EXTSET	E0			external input count = reg[0]
EXTLONG	E1			reg[0] = external input counter
EXTWAIT	E2			wait for next external input
STRSET	E3	*string*		Copy string to string buffer
STRSEL	E4	*start,length*		Set selection point
STRINS	E5	*string*		Insert string at selection point
STRCMP	E6	*string*		Compare string with string selection
STRFIND	E7	*string*		Find string
STRFCHR	E8	*string*		Set field separators
STRFIELD	E9	*field*		Find field
STRTOF	EA			Convert string selection to floating point
STRTOL	EB			Convert string selection to long integer
READSEL	EC		*string*	Read string selection
STRBYTE	ED			Insert byte at selection point
STRINC	EE			Increment string selection point
STRDEC	EF			Decrement string selection point
SYNC	F0		*5C*	Get synchronization byte
READSTATUS	F1		*status*	Read status byte

READSTR	F2		*string*	Read string from string buffer
VERSION	F3			Copy version string to string buffer
IEEEMODE	F4			Set IEEE mode (default)
PICMODE	F5			Set PIC mode
CHECKSUM	F6			Calculate checksum for FPU code
BREAK	F7			Debug breakpoint
TRACEOFF	F8			Turn debug trace off
TRACEON	F9			Turn debug trace on
TRACESTR	FA	*string*		Send string to debug trace buffer
TRACEREG	FB	*register*		Send register value to trace buffer
READVAR	FC	*item*		Read internal register value
SETREAD	FD			Set read mode
RESET	FF			Reset (9 consecutive FF bytes cause a reset, otherwise it is a NOP)

Appendix B
uM-FPU64 Instruction Timing

				Execution Time (usec)		
Instruction	**Opcode**	**Arguments**	**Returns**	**32-bit**	**64-bit**	**Notes**
NOP	00			4.0	4.0	
SELECTA	01	*register*		5.3	5.3	
SELECTX	02	*register*		5.4	5.4	
CLR	03	*register*		7.0	7.1	
CLRA	04			5.5	5.6	
CLRX	05			6.7	6.8	
CLR0	06			5.7	5.9	
COPY	07	*register1, register2*		8.7	8.9	
COPYA	08	*register*		7.2	7.5	
COPYX	09	*register*		8.1	8.3	
LOAD	0A	*register*		6.5	6.7	
LOADA	0B			5.3	5.5	
LOADX	0C			6.3	6.4	
ALOADX	0D			6.5	6.6	
XSAVE	0E	*register*		8.7	8.9	
XSAVEA	0F			6.9	7.1	
COPY0	10	*register*		7.2	7.4	
LCOPYI	11	*signedByte, register*		7.9	8.0	
SWAP	12	*register1, register2*		8.6	8.8	
SWAPA	13	*register*		7.1	7.4	
LEFT	14			5.3	5.4	
RIGHT	15			5.5	5.6	
FWRITE	16	*register, float32Value*		10.2	11.3	
FWRITEA	17	*float32Value*		8.9	10.0	
FWRITEX	18	*float32Value*		10.0	11.0	
FWRITE0	19	*float32Value*		9.2	10.3	
FREAD	1A	*register*	*float32Value*			
FREADA	1B		*float32Value*			
FREADX	1C		*float32Value*			
FREAD0	1D		*float32Value*			
ATOF	1E	*string*		15 - 51	22 - 64	see note 8
FTOA	1F	*format*		24 - 138	33 - 472	see note 9
FSET	20	*register*		6.5	6.7	see note 3
FADD	21	*register*		9.4 - 13.2	11.0 - 15.8	see note 2, 3
FSUB	22	*register*		10.3 - 13.5	12.3 - 16.2	see note 2, 3
FSUBR	23	*register*		10.4 - 13.6	12.5 - 16.4	see note 2, 3
FMUL	24	*register*		9.2 - 9.4	14.1 - 14.3	see note 2, 3
FDIV	25	*register*		15.2 - 16.3	35.9 - 38.1	see note 2, 3
FDIVR	26	*register*		15.3 - 16.4	36.2 - 38.4	see note 2, 3
FPOW	27	*register*		186 - 203	474 - 500	see note 2, 3
FCMP	28	*register*		7.7-7.8	8.0 - 8.1	see note 2, 3

FSET0	29			5.6	5.9	see note 2
FADD0	2A			8.5 - 12.4	10.2 - 15.0	see note 2
FSUB0	2B			9.4 - 12.6	11.4 - 15.4	see note 2
FSUBR0	2C			9.5 - 12.7	11.7 - 15.7	see note 2
FMUL0	2D			8.4 - 8.6	13.3 - 13.5	see note 2
FDIV0	2E			14.3 - 15.5	35.0 - 37.3	see note 2
FDIVR0	2F			14.5 - 15.6	35.3 - 37.6	see note 2
FPOW0	30			186 - 202	473 - 498	see note 2
FCMP0	31			6.9 - 7.0	7.2 - 7.3	see note 2
FSETI	32	*signedByte*		10.5	17.7	
FADDI	33	*signedByte*		12.9 - 16.2	21.4 - 25.5	see note 4
FSUBI	34	*signedByte*		13.8 - 16.5	22.7 - 25.8	see note 4
FSUBRI	35	*signedByte*		13.9 - 16.5	22.9 - 26.0	see note 4
FMULI	36	*signedByte*		12.8 - 12.9	24.6 - 24.7	see note 4
FDIVI	37	*signedByte*		18.8 - 19.7	46.5 - 48.5	see note 4
FDIVRI	38	*signedByte*		18.9 - 20.1	46.8 - 48.8	see note 4
FPOWI	39	*signedByte*		21.8 - 22.1	47.6 - 48.2	see note 5
FCMPI	3A	*signedByte*		11.3	18.5	see note 4
FSTATUS	3B	*register*		6.2	6.4	
FSTATUSA	3C			4.5	4.6	
FCMP2	3D	*register1, register2*		9.3	9.7	
FNEG	3E			4.9	5.0	
FABS	3F			4.9	5.0	
FINV	40			14.3 - 15.1	34.3 - 35.5	see note 2
SQRT	41			16.9 - 17.3		see note 2
ROOT	42	*register*		205	522	see note 2, 10
LOG	43			79 - 81	236 - 238	see note 2
LOG10	44			82 - 84	243 - 246	see note 2
EXP	45			72 - 80	182 - 191	see note 11
EXP10	46			31 - 198	405 - 488	see note 11
SIN	47			66 - 73	198 - 211	see note 2
COS	48			78 - 80	208	see note 2
TAN	49			75 - 76	216 - 221	see note 2
ASIN	4A			53 - 74	191 - 238	see note 12
ACOS	4B			56 - 70	196 - 233	see note 12
ATAN	4C			45 - 74	161 - 222	see note 12
ATAN2	4D	*register*		58 - 86	196 - 258	see note 12
DEGREES	4E			7.7	12.5 - 12.8	see note 2
RADIANS	4F			7.7	12.6 - 12.7	see note 2
FMOD	50	*register*		8.5 - 15.3	10.2 - 19.2	see note 2
FLOOR	51			6.1 - 7.5	6.9 - 9.1	see note 2
CEIL	52			7.5 - 8.3	9.1 - 10.4	see note 2
ROUND	53			12.1 - 19.0	14.6 - 34.0	see note 2
FMIN	54	*register*		7.7 - 8.8	8.2 - 9.4	see note 2
FMAX	55	*register*		7.7 - 8.8	8.2 - 9.4	see note 2
FCNV	56	*conversion*		9.2 - 22.0	14.4 - 45.0	see note 13
FMAC	57	*register1, register2*		15.7 - 16.0	22.9 - 23.3	see note 2

FMSC	58	*register1, register2*		15.9 - 16.2	23.2 - 24.0	see note 2
LOADBYTE	59	*signedByte*		10.4	17.4	
LOADUBYTE	5A	*unsignedByte*		10.3	17.3	
LOADWORD	5B	*signedWord*		10.4	17.1	
LOADUWORD	5C	*unsignedWord*		10.3	17.0	
LOADE	5D			5.1	5.2	
LOADPI	5E			5.1	5.2	
FCOPYI	5F	*signedByte, register*		11.1	18.0	
FLOAT	60			8.8	15.6	
FIX	61			8.4	15.4	see note 2
FIXR	62			13.3	32.6	see note 2
FRAC	63			5.7	6.0	
FSPLIT	64			7.5	7.7	
SELECTMA	65	*register, rows,columns*				
SELECTMB	66	*register, rows,columns*				
SELECTMC	67	*register, rows,columns*				
LOADMA	68	*row,column*				
LOADMB	69	*row,column*				
LOADMC	6A	*row,column*				
SAVEMA	6B	*row,column*				
SAVEMB	6C	*row,column*				
SAVEMC	6D	*row,column*				
MOP	6E	*action*				
FFT	6F	*action*				
WRIND	70	*dataType,pointer ,count,value1… valueN*				
RDIND	71	*dataType,pointer ,count*	*value1…valueN*			
DWRITE	72	*register, value64*				
DREAD	73	*register*	*value64*			
LBIT	74	*action, register*				
SETIND	77	*type,{register| address}*				
ADDIND	78	*register, unsignedByte*				
COPYIND	79	*register1, register2, register3*				
LOADIND	7A	*register*				
SAVEIND	7B	*register*				
INDA	7C	*register*				
INDX	7D	*register*				
FCALL	7E	*function*				

EVENT	7F	*action {,function}*				
RET	80					
BRA	81	*relativeOffset*				
BRA,cc	82	*conditionCode, relativeOffset*				
JMP	83	*absoluteOffset*				
JMP,cc	84	*conditionCode, absoluteOffset*				
TABLE	85	*tableSize, tableItem1... tableItemN*				
FTABLE	86	*conditionCode, tableSize, tableItem1... tableItemN*				
LTABLE	87	*conditionCode, tableSize, tableItem1... tableItemN*				
POLY	88	count, float32Value1... float32ValueN				
GOTO	89	*register*				
RET,cc	8A	*conditionCode*				
LWRITE	90	*register, int32Value*				
LWRITEA	91	*int32Value*				
LWRITEX	92	*int32Value*				
LWRITE0	93	*int32Value*				
LREAD	94	*register*	*int32Value*			
LREADA	95		*int32Value*			
LREADX	96		*int32Value*			
LREAD0	97		*int32Value*			
LREADBYTE	98		*byteValue*			
LREADWORD	99		*wordValue*			
ATOL	9A	*string*				
LTOA	9B	*format*				
LSET	9C	*register*				
LADD	9D	*register*		7.0	7.4	see note 6
LSUB	9E	*register*		7.0	7.4	see note 6, 7
LMUL	9F	*register*		7.1	9.5	see note 6, 7
LDIV	A0	*register*		20.1 - 20.3	385 - 389	see note 6, 7
LCMP	A1	*register*		5.9 - 6.0	6.3 - 6.5	see note 6, 7
LUDIV	A2	*register*		8.4 - 19.9	385 - 389	see note 6, 7
LUCMP	A3	*register*		6.0	6.3 - 6.5	see note 6, 7
LTST	A4	*register*		6.1	6.3	see note 6, 7
LSET0	A5			6.1	6.3	see note 6
LADD0	A6			6.1	6.5	see note 6
LSUB0	A7			6.0	6.5	see note 6

LMUL0	A8			6.3	8.6	see note 6
LDIV0	A9			7.6 - 18.8	7.9 - 19.4	see note 6
LCMP0	AA			5.1	5.6	see note 6
LUDIV0	AB			7.4 - 18.5	7.6 - 19.1	see note 6
LUCMP0	AC			5.0	5.5	see note 6
LTST0	AD			5.1	5.4	see note 6
LSETI	AE	*signedByte*		6.5	6.8	
LADDI	AF	*signedByte*		6.5	6.9	
LSUBI	B0	*signedByte*		6.5	6.9	
LMULI	B1	*signedByte*		6.6	9.0	
LDIVI	B2	*signedByte*		19.8	384.1	
LCMPI	B3	*signedByte*		5.5	6.0	
LUDIVI	B4	*unsignedByte*		19.3	384.1	
LUCMPI	B5	*unsignedByte*		5.5	5.9	
LTSTI	B6	*unsignedByte*		5.5	5.8	
LSTATUS	B7	*register*		5.8	6.0	
LSTATUSA	B8			4.2	4.3	
LCMP2	B9	*register1, register2*		7.4	7.8	
LUCMP2	BA	*register1, register2*		7.4	7.8	
LNEG	BB			5.3	5.4	
LABS	BC			5.5	5.6	
LINC	BD	*register*		6.9	7.1	
LDEC	BE	*register*		6.9	7.1	
LNOT	BF			5.2	5.3	
LAND	C0	*register*		7.0	7.4	
LOR	C1	*register*		7.0	7.4	
LXOR	C2	*register*		6.9	7.3	
LSHIFT	C3	*register*		7.4 - 11.3	7.6 - 12.2	
LMIN	C4	*register*		7.0	7.5	
LMAX	C5	*register*		7.0	7.5	
LONGBYTE	C6	*signedByte*		5.6	5.7	
LONGUBYTE	C7	*unsignedByte*		5.6	5.7	
LONGWORD	C8	*signedWord*		6.5	6.6	
LONGUWORD	C9	*unsignedWord*		6.5	6.6	
LSHIFTI	CA	*unsignedByte*		6.9 - 10.9	7.2 - 11.7	
LANDI	CB	*unsignedByte*		6.5	6.8	
LORI	CC	*unsignedByte*		6.5	6.8	
SETSTATUS	CD	*status*				
SEROUT	CE	*action, {baud}|{string}*				
SERIN	CF	*action*				
DIGIO	D0	*action,{mode}*				
ADCMODE	D1	*mode*				
ADCTRIG	D2					
ADCSCALE	D3	*channel*				
ADCLONG	D4	*channel*				
ADCLOAD	D5	*channel*				
ADCWAIT	D6					

TIMESET	D7					
TIMELONG	D8					
TICKLONG	D9					
DEVIO	DA	*device,action {,…}*				
DELAY	DB	*period*				
RTC	DC	*action*				
SETARGS	DD					
EXTSET	E0					
EXTLONG	E1					
EXTWAIT	E2					
STRSET	E3	*string*				
STRSEL	E4	*start,length*				
STRINS	E5	*string*				
STRCMP	E6	*string*				
STRFIND	E7	*string*				
STRFCHR	E8	*string*				
STRFIELD	E9	*field*				
STRTOF	EA					
STRTOL	EB					
READSEL	EC		*string*			
STRBYTE	ED					
STRINC	EE					
STRDEC	EF					
SYNC	F0		*5C*			
READSTATUS	F1		*status*			
READSTR	F2		*string*			
VERSION	F3					
IEEEMODE	F4					
PICMODE	F5					
CHECKSUM	F6					
BREAK	F7					
TRACEOFF	F8					
TRACEON	F9					
TRACESTR	FA	*string*				
TRACEREG	FB	*register*				
READVAR	FC	*item*				
SETREAD	FD					
RESET	FF					

Notes:

1. All read instructions must be preceded by the minimum Read Setup Delay.
2. Floating point values 0.001 and 1000.0 used for timing.
3. If a 64-bit to 32-bit conversion is required, an additional 5.1 - 5.3 usec is required.
   If a 32-bit to 64-bit conversion is required, an additional 1.1 usec is required.
4. Floating point values 0.001 and 100.0 used for timing.
5. Floating point values 0.001 and 10.0 used for timing.
6. Integer values 100 and 1000000 (32-bit) or 1000000000000 (64-bit) used for timing.
7. If a 64-bit to 32-bit conversion is required, an additional 0.1 usec is required.
   If a 32-bit to 64-bit conversion is required, an additional 0.1 usec is required.
8. Strings “1”, “1234567890”, and “-1234.5678e20” used for testing

9. Values “1”, “1234567890”, and “-1234.5678e20” used for testing
10. The root of 4.0 is used for timing
11. Floating point values 30.0 and 0.001 used for timing.
12. Floating point values 0.25 and 0.75 used for timing.
13. Conversions 0, 1, 2, and 3 used for timing.