Micromega Corporation

Release Notes

uM-FPU64 Firmware Release 411

The version numbers of the uM-FPU64 firmware and uM-FPU64 IDE software have previously been maintained separately, but starting with uM-FPU64 IDE release 411, the version numbers of the firmware and IDE software will be synchronized .

Changes in Release 411

- fixed table size for `TABLE`, `FTABLE`, and `LTABLE` instructions to allow up to 256 table items.
- added additional support for loadable device driver
- added `DEVIO` support for SD cards with support for FAT16/FAT32
- added `WRITE_FLOAT`, `WRITE_LONG`, `WRITE_COMMA`, `WRITE_CRLF` routines to `DEVIO,ASYNC` device
- changed `READSTR` and `READSEL` to terminate transfer if zero byte in string buffer
- fixed bug in `MOP,LU_DECOMP`
- **Ram allocation**
  - at Reset all 2304 bytes are assigned to the foreground
  - a minimum of 256 bytes is always allocated to the foreground
  - an alternate allocation method has been added to `DEVIO(MEM,ALLOCATE,…)`, if the Foreground bits of memSize are zero, the 16-bit word that follows specifies the number of bytes to allocate to the foreground
  - any unused bytes from `DEVIO(MEM,ALLOCATE,…)` are now assigned to a dynamic allocation pool
  - FIFO memory can be assigned dynamically using the `DEVIO(FIFOn, ALLOC_MEM,size)` or `DEVIO(FIFOn,ALLOC_MEMR,regSize)` instructions
- removed VDRIVE2 device
- added support for loadable devices
- SD card device added to `DEVIO` with support for FAT16 and FAT32.

  Note: The loadable SD device is still in beta test. If you would like to be part of the beta testing, please send a request by email to support@micromegacorp.com)
- added *`ALLOC_ERROR`* and *`DEVICE_ERROR`* bits to Error Status
- `WRIND`: if the count byte is zero, the count is now loaded from the lower 16-bits of register 0

- `RDIND`: if the count byte is zero, the count is now loaded from the lower 16-bits of register 0
- `WRIND`: optimized 32-bit long to 32-bit long, and 32-bit float to 32-bit float operations so that no delay is required between transfers at the maximum data rate.
- `RDIND`: optimized 32-bit long to 32-bit long, and 32-bit float to 32-bit float operations so that no delay is required between transfers at the maximum data rate.
- `TRACESTR` : added `$Rxx` as a special string to display `READVAR` values. Where `xx` is the decimal value of the `READVAR` value to display. e.g. `$R14` displays length of string buffer.
- if debug mode is enabled and an error occurs, a trace message is displayed and a break occurs. The trace message is `{ERROR:xxxx}`, where `xxxx` is the hexadecimal value of the Error Status. (See description of Error Status below).

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 dynamic memory allocation failed because the number of bytes requested were not available.

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.

DEVIO *Device Input/Output*

Syntax: **`DEVIO,`*`device,action{,…}`***

...

`DEVIO, `*`device`*`, LOAD_DEVICE, `*`xopdev`*

...

Load Device (0x5x)

`DEVIO, `*`device`*`, LOAD_DEVICE, `*`xopdev`*

Attach the loadable *device* to the device code loaded in the XOP area of Flash memory as specified by *xopdev*. Ram is allocated from the dynamic allocation for use by the *device*. This call is required before using a loadable device.

DEVIO, SDFAT *SD card with FAT16 and FAT32 support*

Syntax: **`DEVIO,SDFAT,`*`action{,…}`***

Description: This instruction provides support for SD cards and supports the FAT16 and FAT32 file systems. The SD card can be used in raw mode for general non-volatile storage of up to 32GB of data, or in file mode with FAT16 and FAT32 support. Files stored on SD cards can be read by any device that supports the FAT16 and FAT32 file system..

```
DEVIO, SDFAT, DISABLE
DEVIO, SDFAT, ENABLE, pin, config
DEVIO, SDFAT, STATUS
DEVIO, SDFAT, GET_VALUE, item
DEVIO, SDFAT, READ_BLOCK, regBlock, regPtr
DEVIO, SDFAT, WRITE_BLOCK, regBlock, regPtr
DEVIO, SDFAT, FIND, filename
DEVIO, SDFAT, NEXT
DEVIO, SDFAT, OPEN, type, filename
DEVIO, SDFAT, CLOSE
DEVIO, SDFAT, UPDATE
DEVIO, SDFAT, GET_POSITION
DEVIO, SDFAT, SET_POSITION, regAddress
```

Opcode: ***DA***

Byte 2: ***SDFAT (0xA0)***

Byte 3: ***action***

An unsigned byte specifying the device action. Actions that are specific to the `SDFAT` device are shown below. For actions that are common to all devices, see the *DEVIO* description.

Disable (0x00)

`DEVIO, SDFAT, DISABLE`

Disable the `SDFAT` device and release the digital pins.

Enable (0x01)

`DEVIO, SDFAT, ENABLE, `*`pin`*`, `*`config`*

Selects the pins to use for the `SDFAT`, configures and initializes the display.

Byte 4: ***pin***

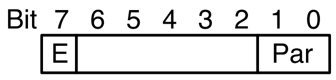
Specifies the pins to use for the SD card select.

D0 to D8	28-pin uM-FPU64 chip
D0 to D22	44-pin uM-FPU64 chip

Pin Assignments

pin	SD card select

Byte 5: ***config***



Bit 7 **Write Block Enable**

IDE Symbol	*IDE Value*	*Description*
–	0x00	`WRITE_BLOCK` action disabled
WRBLK_ENABLE	0x80	`WRITE_BLOCK` action disabled

Bits 1:0 **Partition**

IDE Symbol	*IDE Value*	*Description*
-	0x00	Use partition 1 on SD card (default)
-	0x01	Use partition 2 on SD card (default)
-	0x02	Use partition 3 on SD card (default)
-	0x03	Use partition 4 on SD card (default)

Status (0x02)

`DEVIO, SDFAT, STATUS`

Returns the current status of the SDFAT device in register 0.

0	OK
-1	End of File
< -2	Error value

Get Value (0x03)

`DEVIO, SDFAT, GET_VALUE,` *`item`*

Returns the value specified by *item*.

0	Current status
1	Volume Size

Read Block (0x04)

`DEVIO, SDFAT, READ_BLOCK,` *`regBlock, regPtr`*

Write Block (0x05)

`DEVIO, SDFAT, WRITE_BLOCK,` *`regBlock, regPtr`*

Find File (0x06)

`DEVIO, SDFAT, FIND,` *`filename`*

Next File (0x07)

`DEVIO, SDFAT, NEXT`

Open File (0x08)

`DEVIO, SDFAT, OPEN,` *`type`*`,` *`filename`*

Close FIle (0x09)
```
   DEVIO, SDFAT, CLOSE
```

Update FIle (0x0A)
```
   DEVIO, SDFAT, UPDATE
```

Get Position (0x0B)
```
   DEVIO, SDFAT, GET_POSITION
```

Set Position (0x0C)
```
   DEVIO, SDFAT, SET_POSITION, regAddress
```

DEVIO, FIFO — *FIFO Buffer Interface*

Syntax:

```
DEVIO,FIFO1,action{,…}
DEVIO,FIFO2,action{,…}
DEVIO,FIFO3,action{,…}
DEVIO,FIFO4,action{,…}
```

Description: These instructions provide support for First In First Out (FIFO) buffers. They can be used to buffer data, or to transfer data from one process to another. Memory must be allocated to the FIFOs from the dynamic allocation area using one of the following instructions:

```
   DEVIO, MEM, ALLOCATE, memSize, fifoSize
   DEVIO, FIFOn, ALLOC_MEM, size
   DEVIO, FIFOn, ALLOC_MEMR, regSize
```

```
DEVIO, FIFOn, DISABLE
DEVIO, FIFOn, ENABLE, pin, config
DEVIO, FIFOn, CLEAR
DEVIO, FIFOn, USED
DEVIO, FIFOn, FREE
DEVIO, FIFOn, STATUS
DEVIO, FIFOn, CLEAR_OVERFLOW
DEVIO, FIFOn, ALLOC_MEM, size
DEVIO, FIFOn, ALLOC_MEMR, regSize
…
```

Dynamic Memory Allocation (0x08)
```
   DEVIO, FIFOn, ALLOC_MEM, size
```
The number of memory bytes specified by `size` are allocated from the dynamic allocation for use by `FIFOn`. If not enough bytes are available in the dynamic allocation, the size of `FIFOn` is set to zero.

Byte 4: ***size***

Unsigned 16-bit word specifying the number of consecutive memory bytes to allocate from the dynamic allocation to `FIFOn`.

Dynamic Memory Allocation (0x09)
```
   DEVIO, FIFOn, ALLOC_MEMR, regSize
```
The number of memory bytes specified by the value of `regSize` are allocated from the dynamic allocation. The memory address of the first byte is returned in register 0. If there are not enough bytes available in the dynamic allocation, the size of `FIFOn` is set to zero.

Byte 4: ***regSize***

An 8-bit register value. The lower 16-bits of the register specify the number of consecutive memory bytes to allocate from the dynamic allocation to `FIFOn`.

DEVIO, MEM   Memory Interface

Syntax: **`DEVIO,MEM,`*`action{,…}`***

Description: This instruction stores data to the general memory area in RAM. 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. 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 allocation can be changed with the *`DEVIO,MEM,ALLOCATE`* instruction. All memory not allocated to the Foreground, Background, FIFO1, FIFO2, FIFO3, FIFO4 is available for dynamic allocation to FIFOs or loadable devices using the *`DEVIO,FIFOn,ALLOC_MEM,size`*, *`DEVIO,FIFOn,ALLOC_MEMR,regSize`*, or *`DEVIO,device,LOAD_DEVICE,xopdev`* instruction.

```
DEVIO, MEM, DISABLE
DEVIO, MEM, ENABLE, pin, config
DEVIO, MEM, ALLOCATE, memSize, fifoSize
DEVIO, MEM, ALLOCATE, memSize, fgSize
…
```

Allocate Memory Buffers (0x02)

`DEVIO, MEM, ALLOCATE, `*`memSize, fifoSize`*

The 2304 byes of available memory are allocated to the foreground, background, FIFO1, FIFO2, FIFO3, and FIFO4. The 4-bit value for each memory type specifies the amount of memory to allocate. At least 256 bytes are always allocated to the foreground. If the sum of all allocations is greater than the maximum 2304 bytes of available RAM, the foreground allocation is as specified, but no memory bytes are allocated to the background, FIFO1, FIFO2, FIFO3, or FIFO4. All remaining bytes are used for dynamic allocation.

Byte 4: ***memSize***

Unsigned byte specifying the number of bytes to allocate to the foreground and background memory buffers.

Bit	7 4	3 0
	Foreground	Background

Bits 7:4	**Foreground Memory Allocation**
Bits 3:0	**Background Memory Allocation**

Byte 5-6: ***fifoSize*** *(unsigned word)*

Unsigned 16-bit word specifying the number of bytes to allocate to the FIFO buffers.

Bit 15 12	11 8	7 4	3 0
FIFO1	FIFO2	FIFO3	FIFO4

Bits 15:12	**FIFO1 Memory Allocation**
Bits 11:8	**FIFO2 Memory Allocation**
Bits 7:4	**FIFO3 Memory Allocation**
Bits 3:0	**FIFO4 Memory Allocation**

Memory Allocation Codes

Value	*Description*
0x0	No memory
0x1	2 bytes
0x2	4 bytes
0x3	8 bytes
0x4	16 bytes
0x5	32 bytes
0x6	64 bytes
0x7	128 bytes
0x8	256 bytes
0x9	512 bytes
0xA	1024 bytes
0xB	2048 bytes
0xC	4096 bytes
0xD	8192 bytes
0xE	16384 bytes
0xF	default (FG: 1024 bytes, BG: 1024 bytes, FIFO1-4: 64 bytes)

Allocate Memory Buffers (0x02)

`DEVIO, MEM, ALLOCATE,` ***`memSize, fgSize`***

If the foreground memory allocation bits (bits 7:4) of ***`memSize`*** are zero, then ***`fgSize`*** specifies the number of memory byes allocated to the foreground, and no memory is allocated to FIFO1, FIFO2, FIFO3, and FIFO4. At least 256 bytes are always allocated to the foreground. The background memory allocation bits (bits 3:0) of ***`memSize`*** specify the number of memory byes allocated to the background. Any remaining bytes are available for dynamic allocation memory for FIFOs or loadable devices.

Byte 4: ***`0`***

Byte 5-6: ***`fgSize`*** *`(unsigned word)`*

Unsigned 16-bit word specifying the number of memory bytes to allocate to the foreground.

RDIND *Read data using indirect pointer*

Syntax: **`RDIND,`*`dataType,pointer,count`***

Description: Read *count* data values of the specified *dataType* from the *pointer* location. If *count* = 0, then the count is loaded from the lower 16 bits of register 0. The pointer can be a register pointer or a memory pointer. If *dataType* is different then the data type of the *pointer* data conversion is automatically performed. The data items must be read immediately following this instruction. See the `SETIND` instruction for a description of pointers. The `RDIND` instruction has been optimized for 32-bit transfers of the same data type (e.g. long-to-long or float-to-float). These transfers can be done at the maximum transfer rate without filling the instruction buffer. Transfers that require data conversions may require an additional delay between data transfers to avoid exceeding the 256 byte FPU instruction buffer.

Opcode: **`71`**

Byte 2: ***`dataType`***

Bit 7 6 5 4	3 2 1 0
-	Data Type

Bits 3:0 **Data Type**

IDE Symbol	*IDE Value*	*Description*
INT8	0x08	8-bit signed integer data
UINT8	0x09	8-bit unsigned integer data
INT16	0x0A	16-bit signed integer data
UINT16	0x0B	16-bit unsigned integer data
LONG32	0x0C	32-bit long integer data
FLOAT32	0x0D	32-bit floating point data
LONG64	0x0E	64-bit long integer data
FLOAT64	0x0F	64-bit float point data

Byte 3: ***`pointer`***
The register number of a register that contains a pointer (0 to 255).

Byte 4: ***`count`***
An 8-bit value that specifies the number of data items to read from the pointer location (0 to 255). If *count* = 0, the lower 16 bits of register 0 specify the number of data items to read from the pointer location.

Special Cases: • if *dataType* is 32-bit floating point, and PICMODE is enabled, the values are converted from IEEE-754 format before being sent

See Also: `SETIND, ADDIND, WRIND, COPYIND, LOADIND, SAVEIND`
`SETREAD, FREAD, FREAD0, FREADA, FREADX, LREAD, LREAD0, LREADA, LREADX, LREADBYTE, LREADWORD, DREAD`

TRACESTR Display debug trace message

Syntax: **`TRACESTR,`*`string`***

Description: Used with the built-in debugger. If the debugger is not enabled, this instruction is ignored. If the debugger is enabled, the *string* will be displayed on the debug terminal. If the string is of the form `$Rxx`, a `READVAR` value is output as a hexadecimal string, where `xx` is the decimal value of the `READVAR` value.

Opcode: **`FA`**

Bytes 2-n: ***`string`***
A zero-terminated string.

Examples:	`TRACESTR, "test"`	sends {"test"} to debug terminal
	`TRACESTR, "$R14"`	sends {"$R14":0000} to debug terminal (READVAR, 14 value)

See Also: `TRACEOFF, TRACEON, TRACEREG, BREAK`

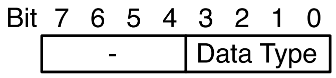
WRIND *Write data using indirect pointer*

Syntax: **`WRIND,`*`dataType,pointer,count,value1...valueN`***

Description: Write *count* data values of the specified *dataType* to the *pointer* location. If *count* = 0, then the count is loaded from the lower 16 bits of register 0. The pointer can be a register pointer or a memory pointer. If *dataType* is different then the data type of the *pointer* data conversion is automatically performed. See the `SETIND` instruction for a description of pointers.
The `WRIND` instruction has been optimized for 32-bit transfers of the same data type (e.g. long-to -long or float-to-float). These transfers can be done at the maximum transfer rate without filling the instruction buffer. Transfers that require data conversions may require an additional delay between data transfers to avoid exceeding the 256 byte FPU instruction buffer.

Opcode: **`70`**

Byte 2: ***`dataType`***



Bits 3:0 **Data Type**

IDE Symbol	*IDE Value*	*Description*
INT8	0x08	8-bit signed integer data
UINT8	0x09	8-bit unsigned integer data
INT16	0x0A	16-bit signed integer data
UINT16	0x0B	16-bit unsigned integer data
LONG32	0x0C	32-bit long integer data
FLOAT32	0x0D	32-bit floating point data
LONG64	0x0E	64-bit long integer data
FLOAT64	0x0F	64-bit float point data

Byte 3: ***`pointer`***
The register number of a register that contains a pointer (0 to 255).

Byte 4: ***`count`***
An 8-bit value that specifies the number of data items to read from the pointer location (0 to 255). If *count* = 0, the lower 16 bits of register 0 specify the number of data items to read from the pointer location.

Bytes 5-n: ***`value1...valueN`***
Data values of the type specified by *dataType*.

Special Cases: • if *dataType* is 32-bit floating point, and PICMODE is enabled, the values are converted to IEEE-754 format before being stored

See Also: `SETIND, ADDIND, RDIND, COPYIND, LOADIND, SAVEIND`

FWRITE, FWRITE0, FWRITEA, FWRITEX, LWRITE, LWRITE0, LWRITEA, LWRITEX, DWRITE