mComm Users Guide

Contents

Quick Start Guides

One-way Data Output

The one-way communication type is implemented only for UART transmission. The data it outputs depends on the values that are defined in the configuration file. The packet is sent at the end of each mTouch decode routine before returning to the application.

Open **MCOMM_CONFIG.H**.
Make sure:

- MCOMM_ENABLED is defined
- MCOMM_TYPE is defined as MCOMM_UART_ONE_WAY

Configuring a Hardware UART Module

```
#define MCOMM_UART_1WAY_MODULE    MCOMM_UART_HARDWARE_MODULE
```

Hardware module will be used.
The valid values for MCOMM_UART_BAUDRATE are greater for this configuration. The top supported speed is 115200 in most cases.

Configuring a Bit-Banged Software-UART Implementation

```
#define MCOMM_UART_1WAY_MODULE    MCOMM_UART_SOFTWARE_IMPLEMENTATION
#define MCOMM_UART_SOFT_TXPORT    PORTA   // TX Port Register
#define MCOMM_UART_SOFT_TXTRIS    TRISA   // TX Tris Register
#define MCOMM_UART_SOFT_TXPIN     5       // TX Pin
```

Bit-banged, software-UART will be implemented.
The 'Soft TX' definitions, above, determine which pin is used for TX.
The valid values for MCOMM_UART_BAUDRATE are fewer for this configuration. The most commonly supported speeds are 9600, 19200, and 38400.

Other Configuration Options

```
#define MCOMM_UART_1WAY_OUTPUT          MCOMM_UART_1WAY_DECIMAL
#define MCOMM_UART_1WAY_DELIMITER       ';'

#define MCOMM_UART_1WAY_OUT_STATE       // Output state mask
#define MCOMM_UART_1WAY_OUT_TOGGLE      // Output toggle mask
#define MCOMM_UART_1WAY_OUT_READING     // Output readings
#define MCOMM_UART_1WAY_OUT_BASELINE    // Output baselines
#define MCOMM_UART_1WAY_OUT_MATRIX      // Output matrix value
#define MCOMM_UART_1WAY_OUT_SLIDER      // Output slider value
```

The 1-way output definition decides whether the output will be in hex or decimal.
The delimiter is placed between each output value.

If the “1-way Out” definitions are defined, that data will be output by the communications.

Two-way Data Communication

The two-way communications for mComm are highly flexible. Most features may be optionally disabled to reduce the resource requirements of the program.

There are three supported communication types for the two-way communications:

- Asynchronous, hardware UART
- I^2C
- SPI

If I^2C or SPI are used, the communications will be synchronous. If enabled, the stream functions as a prebuilt output packet of RAM arrays.

If UART is used, the communications will be asynchronous. If enabled, the stream outputs at the end of the mTouch decode function or if the master issues a ‘read stream’ command.

RAM Access

If two-way communications are enabled, RAM access is implemented. Unlike the other memory access methods, this one is not optional.

- 0x00 :: Write to RAM
- 0x01 :: Read from RAM

Non-Volatile Memory Access

Access to reading/writing EEPROM is toggled by (un)commenting `MCOMM_ENABLE_NVM_ACCESS`.

> NOTE: This value turns on/off the ability of the master to access this region of memory. It does not affect whether or not mTouch configuration values are stored in EEPROM. To turn on that feature of the mTouch Framework, see the top part of **MTOUCH_CONFIG.H**

If `MCOMM_ENABLE_NVM_ACCESS` is defined, mComm will implement the opcodes:

- 0x02 :: Write to EEPROM
- 0x03 :: Read from EEPROM

If it is not defined, commands received with those opcodes will be ignored.

Stream

Implementation of the stream feature can be toggled using MCOMM_ENABLE_STREAM

If `MCOMM_ENABLE_STREAM` is defined, mComm will implement the opcodes:

- 0x04 :: Write to Stream
- 0x05 :: Read from Stream

If it is not defined, commands received with those opcodes will be ignored.

If MCOMM_STREAM_STORED_IN_RAM is defined, the stream will be stored in RAM. This allows the master to edit the vectors stored in the stream at run-time. If this is not defined, the stream will be stored in constant memory to reduce the RAM requirements.

If MCOMM_STREAM_EN_ON_POR is defined and the UART is being used to asynchronously send new data, the stream will begin outputting immediately on power-up. Otherwise, the stream will only be enabled once bit 0 of the mComm_streamConfig variable is set. (The address of mComm_streamConfig in RAM will be stored in the configuration address block. This is accessed by reading from RAM location 0x0001.)

The stream is implemented as an array of vectors. Each vector is three bytes: a pointer and an 8-bit length. MCOMM_STREAM_SIZE determines how many vectors the stream will store. NOTE: The actual size of the stream array will be MCOMM_STREAM_SIZE+1. A 0-length vector is always placed at the end of the array.

To define the default values of the stream, start with MCOMM_STREAM_VALUE0 and increment the index with each vector entry. If the size of the stream array is larger than the number of defined initial stream values, the non-defined values will be filled with 0's. The stream will stop after reaching the first 0-length vector.

By default, the stream is configured to output the sensor state mask, reading, and baseline values. Use these as examples for customizing the stream to your needs.

Custom, User-Implemented Commands

There is built-in support for custom opcodes. MCOMM_ENABLE_CUSTOM_OPCODE must be defined.

If `MCOMM_ENABLE_CUSTOM_OPCODE` is defined, mComm will implement the opcodes:

- 0x06 :: Custom Write Operation
- 0x07 :: Custom Read Operation

If it is not defined, commands received with those opcodes will be ignored.

If UART is being implemented, read and write commands are processed at the end of the mTouch decode. While processing the input commands, mComm will also ask the application if it has any data to send.

> **Only if UART is being implemented:**
> MCOMM_CUSTOM_CALLBACK must be defined to the name of the callback function which returns a 1 if data needs to be sent, 0 otherwise.
> An example is available in mComm_config.h which checks to see if any sensors have changed state and if the custom command is currently enabled.

If I2C or SPI is being implemented, read and write commands are executed immediately.

For all communication types (UART, I2C, SPI):
MCOMM_CUSTOM_PROCESS must be defined to the name of the function that will process the input buffer and prepare for either the read or write iterator function to be called for the first time.

MCOMM_CUSTOM_READ_ITERATOR must be defined to the name of the function that will be called when the opcode is '0x07'. This iterator will be continuously called until the output vector's hasNext flag is cleared. The first time this function is called will be right after the process function has been called.

MCOMM_CUSTOM_WRITE_ITERATOR must be defined to the name of the function that will be called when the opcodes is '0x08'. This iterator will be continuously called until the output vector's hasNext flag is cleared. The first time this function is called will be right after the process function has been called.

Configuring the Literal and Address Configuration Blocks

The first thing the master should do on power-up is read the PIC's configuration. The configuration comes in two packets of data: literals and addresses.

The literal data contains values that change only at compile-time. It stores information about the number of implemented sensors, the size of the EEPROM, the application's version numbers, and other configuration details.

> The values stored in this packet can be edited at the bottom of the configuration file.

The address data contains the addresses of RAM variables and arrays. Since compilers will relocate variables during development, this allows the master to always find the correct location to access.

> The values stored in this packet can be edited at the bottom of the configuration file.

NOTE: If you change the configuration blocks, adjust the associated size definitions. (MCOMM_CONFIG_LIT_SIZE and MCOMM_CONFIG_ADDR_SIZE)

mComm Two-Way Protocol

The packet structures for the communication types are different, but the payloads for both master and slave do not change.

UART

Two-way UART communications start with a BREAK character and a byteCount. After that, the payload is consistent with the other protocols.

Master → mComm:

BREAK	byteCount	opCode	addr:l	addr:h	length	optional, variable payload	checksum

Where:

- byteCount :: Total length of the packet in bytes, including the checksum
- length
  - If 'read' :: Number of requested bytes to read
  - If 'write' :: Number of bytes in the payload, excluding the checksum
- checksum :: XOR of all bytes, excluding byteCount

mComm → Master:

BREAK	byteCount	opCode	optional, variable payload	checksum

Where:

- byteCount :: Total length of the packet in bytes, including the checksum
- opCode :: Either:
  - Repeat of opcode used for master's request
  - 0xAA for 'ACK'
  - 0xA5 for 'NACK'
- checksum :: XOR of all bytes, excluding the byteCount

I²C

Two-way I2C communications begin with a START bit and a write address, and end with a STOP bit. Other than these changes, the payload for both master and slave remain the same as with the other communication types.

Write:

Where:

- addr:W :: I2C write address of the PIC
- addr:l :: low-byte of the address to access
- addr:h :: high-byte of the address to access
- length :: Number of bytes in the payload, excluding the checksum
- checksum :: XOR of all bytes, excluding I2C write address

Read:

Where:

- addr:W :: I2C write address of the PIC
- addr:l :: low-byte of the address to access
- addr:h :: high-byte of the address to access
- chksum :: XOR of all previous bytes, excluding the write address
- addr:R :: I2C read address of the PIC

- variable payload :: Sent by PIC, number of bytes dependant on ‘length’ value
- chksum :: Sent by PIC, XOR of all bytes in the variable payload.

SPI

Two-way SPI communications begin the address of the write address of the PIC and the byteCount for the packet length. After this, the payload is identical to the other communication methods with the exception of the 'ack' value being sent after a valid address has been read and once the write is complete.

Write:

Master	addr:W	byteCount	opCode	addr:l	addr:h	length	optional, variable payload			checksum	0x00
mComm	0x00	ack	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	ack

Where:

- addr:W :: SPI write address of the PIC
- byteCount :: length of the packet in bytes, including checksum
- addr:l :: low-byte of the address to access
- addr:h :: high-byte of the address to access
- length :: Number of bytes in the payload, excluding the checksum
- checksum :: XOR of all bytes, excluding the write address and byteCount
- ack
  - Sent after valid address
  - Sent once a write is complete.
    - NOTE: mComm will send 0x00 until the write has finished.

Read:

Master	addr:W	byteCount	opCode	addr:l	addr:h	length	checksum	0x00	0x00	0x00	0x00
mComm	0x00	ack	0x00	0x00	0x00	0x00	0x00	variable payload			chksum

Where:

- addr:W :: SPI write address of the PIC
- byteCount :: length of the packet in bytes, including checksum
- addr:l :: low-byte of the address to access
- addr:h :: high-byte of the address to access
- checksum :: XOR of all previous bytes, excluding the write address and byteCount
- addr:R :: I2C read address of the PIC

- variable payload :: Sent by PIC, number of bytes dependant on 'length' value
- chksum :: Sent by PIC, XOR of all bytes in the variable payload.

mComm Master Payloads

These payloads are common across all communication types. The payload is stored in the input buffer and is provided to the 'process' and 'iterator' functions for execution of the command.

The 'process' and 'iterator' functions are only called if the checksum has been verified as valid. The checksum is provided in the input buffer, but does not need to be checked.

Reading

RAM

0x01	addr:l	addr:h	length	checksum

Where:

- addr:l :: RAM address, low byte
- addr:h :: RAM address, high byte
- length :: Number of requested bytes
- checksum :: XOR of all bytes in the payload

mComm response:

0x01	'Length' RAM bytes starting at Addr	checksum

EXCEPTION: Reading from RAM address 0x0000 or 0x0001 will result in the output of the mComm configuration arrays. Address 0x0000 will output the literal configuration array. Address 0x0001 will output the address configuration array. The length value is ignored for these two conditions. The full array is always output.

NVM

0x03	addr:l	addr:h	length	checksum

Where:

- addr:l :: EEPROM address, low byte
- addr:h :: EEPROM address, high byte
- length :: Number of requested bytes
- checksum :: XOR of all bytes in the payload

mComm response:

0x03	'Length' EEPROM bytes starting at Addr	checksum

Stream

0x05	addr:l	addr:h	length	checksum

Where:

- addr:l :: Stream vector index to start reading from
- addr:h :: Not applicable to this opcode. Ignored.

- length :: Not applicable to this opcode. Ignored.
- checksum :: XOR of all bytes in the payload

mComm response:

0x05	All stream vectors w/ index 'addr:l' and higher	checksum

Custom

0x07	User-Defined Structure	checksum

Where:

- 0x07 :: Fixed ‘user opcode’ value
- checksum :: XOR of all bytes in the payload

mComm response:

User-Defined Output Structure	checksum

Writing

The response of the mComm module to a write will vary with the communication type.

When UART is enabled, write commands are not executed until the mComm_Service() function has been called from either the mTouch decode routine or elsewhere in the main loop. When the write completes, an 'acknowledge' packet is sent. (BREAK 0x02 0xAA 0xAA – where 0x02 is the byteLength, 0xAA is the opcode, and the final 0xAA is the checksum: the XOR of 0xAA with 0).

When I2C or SPI is enabled, the write command is executed immediately on receiving the valid checksum and before returning from the receive ISR.

RAM

0x00	addr:l	addr:h	length	variable-length data	checksum

Where:

- addr:l :: The RAM address at which to start writing, low byte
- addr:h :: The RAM address at which to start writing, high byte
- length :: Number of data bytes to write
- checksum :: XOR of all bytes in the payload (opcode, addr, length, and data)

NVM

0x02	addr:l	addr:h	length	variable-length data	checksum

Where:

- addr:l :: The EEPROM address at which to start writing, low byte
- addr:h :: The EEPROM address at which to start writing, high byte
- length :: Number of data bytes to write
- checksum :: XOR of all bytes in the payload (opcode, addr, length, and data)

EXCEPTION: Writing to EEPROM location 0x0000 will reset the mTouch EEPROM values to compile-time settings. All applications should avoid placing data at address 0x0000 due to errors that can occur with EEPROM writes during brown-outs. The mTouch EEPROM initialization byte is stored at address 0x0001, so the recommended starting address for all EEPROM applications, including mTouch, is 0x0002.

Stream

If the stream is stored in RAM (configuration option, located in mComm_config.h), the stream's vectors may be changed by the master. Writing to the stream (changing the vector address/length) should be handled with a RAM write instruction. The address of the stream array is provided in the address configuration output.

Each vector in the stream consists of 3 bytes. The first two bytes are the RAM address to access. The third byte is the number of bytes to read from that location.

The final stream vector must have a length of 0. To help with this, the length of the stream provided in the configuration file is automatically increased by 1 and a 0-length vector is added to the final position.

You have the ability to write to this location, but the length must be 0 or the stream will continue outputting until it stumbles upon a 0 somewhere in RAM.

Custom

The default operation for a ‘custom write’ is to perform a software reset on the microcontroller. The implementation for this can be edited in mComm_custom.c.

Examples

Reading the Literal Configuration Block

UART:	BREAK	0x05	0x01	0x00	0x00	0x00	0x01				
I2C:	START	0xA0	0x01	0x00	0x00	0x00	0x01	START	0xA1	<PIC>	STOP
SPI:	0xA0	0x05	0x01	0x00	0x00	0x00	0x01				