mComm™ Programmer’s Guide

Table of Contents

a) INTRODUCTION TO mComm:

Because of their highly-configurable property, mTouch applications rely on two-way communication to fine-tune their acquisition and filtering functions, for optimal operation.   Microchip has designed the mComm communication protocol, to address mTouch debug and optimization growing needs.

The primary purpose of the mComm protocol is to provide a Master application with the capability to directly access all types of memory in the mTouch device. Therefore, by just specifying a memory type, and the access type (read or write), the user can theoretically get or set any data in the mTouch Slave device. In reality, and for security reasons, it is up to the slave (the PIC microcontroller) to decide what can be accessed. Anything else will remain sealed inside the MCU.

This protocol is designed to support 3 types of memory access: RAM, EEPROM and FLASH.

The mComm protocol can use several hardware communication layers. It uses UART (Hardware and software), SPI and I2C peripherals to carry the data between the Master and the mTouch device.

Finally, the mComm protocol provide the application designer with great flexibility to design their own custom commands to address any mTouch need, by providing services to transport any set of data. The PIC designer specifies a buffer with the data to transmit, and the mTouch service code handles all the formatting and the transmission to the Master. It also collects any response and makes it available in a receive buffer to the user’s application.

The following figure shows how a mTouch system with the 2-way communication should be assembled:

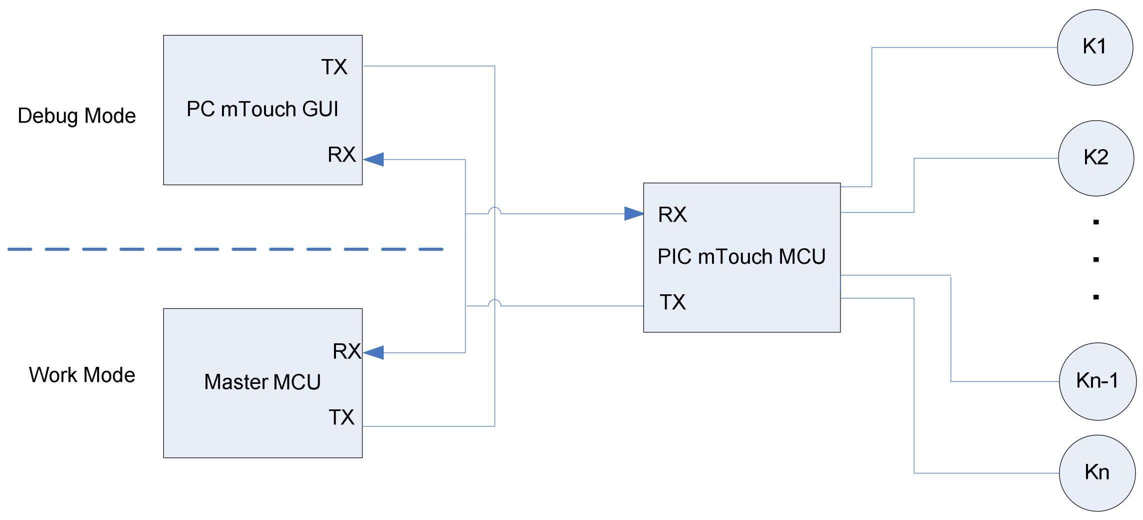


Figure 1: System Block Diagram

b) mTouch GUI PACKET STRUCTURE:

Two-way UART communications start with a BREAK character and a Byte Count. The frames or packets have the following structure, depending on who is sending and who is receiving:

Master → PIC:

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 in packet, starting from the opCode

PIC → 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, starting from the opCode

1.1 The Break Character:

Because of its streaming capability, the mTouch GUI uses binary data transmission (unlike any form for character coding). In order to synchronize the receiver to the beginning of a packet, the system uses a "Break" mechanism, which consists of sending 13 bits with 0 value, then a stop bit. On the receiver side, the UART samples a normal character (8 bits), then expects a stop condition which doesn't occur at that time, consequently generating a "Framing" error. When both conditions exist (binary 0 value and a Framing error), the receiver assumes a Break character was received and will expect a new data packet.

1.2 The Byte Count:

Immediately following the Break character, the receiver will expect a byte count for the remainder of the packet (not including the byte count itself, but including the checksum character).

1.3 The Operation Code (OpCode):

Next, the receiver will expect a code for the operation to be performed or any data response it might be expecting.

1.4 The ADDRESS FIELDS (addr:L and Addr:H)

The addr:L character holds the low byte of an address and the addr:H holds the high byte. This address points to the location that PC/Master want to access in the PIC. **This field applies to UART-based communication.**

1.5 The Length:

This field contains the size of the payload, not including the checksum. **Used in UART-Based communication**

1.6 The Packet Payload:

This entire area of the mComm packet contains any data relevant to a given OpCode and length. It could contain user's custom data as well. Depending on the command, a related number of bytes will be expected for decoding purposes.

In some cases there could even be no data in the payload field.

1.6 The Checksum:

Because of the binary data format used for the exchange, a form of packet-integrity check needs to be implemented. The mComm protocol uses a bitwise XOR on each single byte of the data packet, starting from the OpCode. The byte count is intentionally ignored in this process

c) COMMUNICATION SPEED:

The mTouch communication protocol performs all data exchange at **38400 baud**.

d) COMMAND/RESPONSE DESCRIPTION

The two-way communication for mTouch devices is highly flexible. It provides the ability to read/write data from/to RAM or Non-Volatile Memory, and continuously stream out any data from RAM or NVM. Each kind of operation is assigned a unique opCode.

The master uses these basic read and write commands to access mTouch configuration data, sensor information, and configure the stream output. The details of the mComm protocol can be found in mComm Users Guide in Framework documentation.

e) mComm OPERATIONS FOR mTouch FRAMEWORK

In order to have a good understanding of what mComm has to offer, a good approach would be to go through the mandatory steps of a communication session between the mTouch device (Slave of the communication), and a Master Device (as a PC-GUI for example).

A critical and mandatory step, prior to any data exchange between the Master and mTouch Slave device, is to get the system information which includes critical parameters like the number of sensors of the application, the Data Block location in the EEPROM, various pointers to some critical variables, etc.

This Phase is called: INITIAL SESSION SETUP

Then we will look into how we can retrieve and update parameters in the mTouch Device: This section will be called: ACCESSING THE PARAMETERS

Finally we will explore another important feature of the mComm protocol, that allows to configure, enable and disable data streaming, in the STREAM OPERATIONS section.

5.1 INITIAL SESSION SETUP

The Master device must retrieve critical information, without which the communication session would simply be impossible. The variables needed for the mComm session are:

- The number of sensors: Needed to just build threshold tables, calculate offsets, …
- Start of EEPROM data: Because the mTouch application can have its data block located anywhere in the EEPROM area, the Master has to retrieve this information, prior to any parameter access.
- The mComm configuration Byte will let the Master know if the mTouch device has any EEPROM or not and whether the application permits the access to it or not.
- Input buffer length: Depending on the RAM size on the mTouch device, the Slave limits the size of its mComm receive buffer. Any attempt to send too long packets will result in a communication error.

So the first step is to get the system configuration which contains the locations of mTouch configuration data and variables. The system configuration data is organized in two blocks in RAM memory. One block holds direct information for certain mTouch variables (like the actual number of sensors, EEPROM size, etc.). This block is referred to as “Literal”. The second block is meant to hold pointers to variables, whose location can only be known when the PIC application is generated (compile time). This area is referred to as “Addresses” or “Pointers”. The two tables below contain the mappings and description for these two blocks:

RAM Configuration Block Mapping : Literals		
0x00	mComm Configuration Byte	See mComm Configuration Byte details
0x01	mComm Input Buffer Length	Length of the input buffer in bytes
0x02	mComm Stream Vector Length	Number of vectors in the stream array
0x03	mTouch Number of Sensors	Number of sensors
0x04	mTouch Configuration Byte	See mTouch Configuration Byte details
0x05	Microcontroller's EEPROM Size	Stored as a power of 2, the size of the EEPROM space. Ex: 256 bytes = 8
0x06	mTouch Framework Version	Stores the framework version number
0x07	Application Software Version	Stores the application's software version number - customer use only
0x08	Application Hardware Version	Stores the application's hardware version number - customer use only
0x09	Board ID	Stores the board ID for this application

RAM Configuration Block Mapping : Addresses			
0x00	Addr:L	mComm_streamConfig (RAM)	Address of the configuration byte for the stream in RAM. See mComm Stream Configuration byte details
0x01	Addr:H		
0x02	Addr:L	mComm_stream	Address of the stream array in RAM. Useful if trying to read the address and length values of the vectors
0x03	Addr:H		
0x04	Addr:L	mTouch EEPROM Start Address	The starting address of the mTouch storage block will be located.
0x05	Addr:H		
0x06	Addr:L	mTouch State Mask Variable	The latest sensor state, one bit per enabled sensor. Variable length is the smallest number of bytes to hold all sensors' bits.
0x07	Addr:H		
0x08	Addr:L	mTouch Raw Data Array	Address of the sensor raw data array in RAM.
0x09	Addr:H		
0x0A	Addr:L	mTouch Baseline Array	Address of the sensor baseline data array in RAM.
0x0B	Addr:H		
0x0C	Addr:L	mTouch Slider Array	Address of the slider output array, if enabled. 0 if not.
0x0D	Addr:H		
0x0E	Addr:L	mTouch Toggle Variable	Structured in the same way as the state mask, but only toggles the state of a sensor onPress, not onRelease.
0x0F	Addr:H		
0x10	Addr:L	mComm GUI Custom Variable	Used in mComm_custom.c to show how a custom variable can be accessed by the mComm module
0x11	Addr:H		

The way to read these two blocks is using the “Read From RAM” command. Reading from RAM address 0x0000 will result in the output of the “literals” block, which has 10 bytes of data, and reading from RAM address 0x0001 will result in the output of the “Pointer” block, which has 18 bytes of data.

In both cases, even though the length information has to be provided, it is in reality ignored by the mTouch Slave device, as the full arrays are always sent to the Master device.

Example: read Literal block

Type of command: Reading from RAM

Request from Master:

BREAK	byteCount	opCode	addr:l	addr:h	length	checksum
BREAK	0x05	0x00	0x00	0x00	0x0a	0x04

mComm response:

BREAK	byteCount	opCode	variable payload	checksum
BREAK	0x0c	0x00	10 bytes data in Literal block	checksum

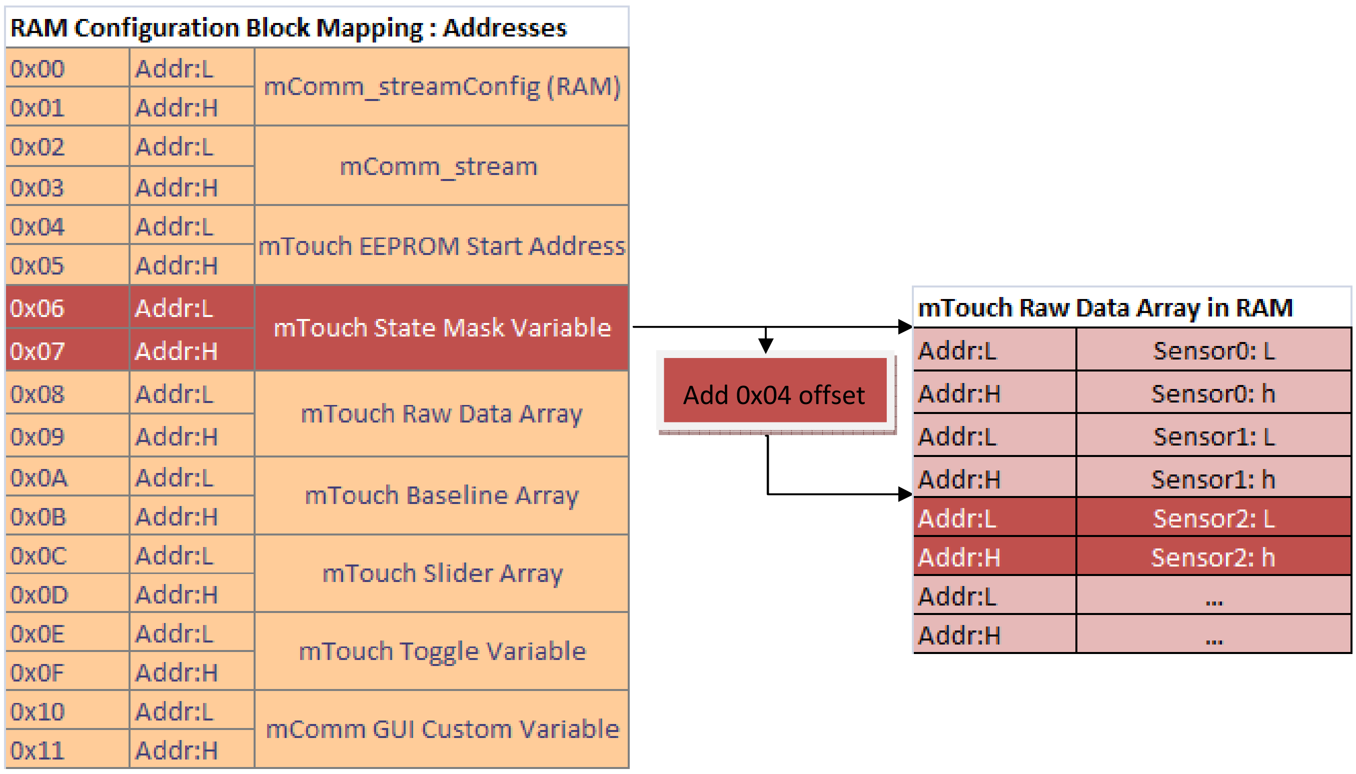
5.2 ACCESSING THE PARAMETERS

Once the Master has setup the communication session by retrieving the Configuration Blocks in RAM described in the previous section, it is able to establish a valid communication session, and can start reading and updating the parameters.

Because the mComm protocol is all based on memory access, we will take just a few examples to illustrate how parameters can be accessed, depending on the type of memory they are stored in.

5.2.1 Reading raw data for sensor2

Assuming the master has already acquired the address of "mTouch Raw Data Array", and it located at 0x0100. Since we are reading the data for sensor 2, we need to offset to location 0x100+2*2 (The raw data is stored as an unsigned integer with a length of 2 bytes).



RAM Configuration Block Mapping : Addresses		
0x00	Addr:L	mComm_streamConfig (RAM)
0x01	Addr:H	
0x02	Addr:L	mComm_stream
0x03	Addr:H	
0x04	Addr:L	mTouch EEPROM Start Address
0x05	Addr:H	
0x06	Addr:L	mTouch State Mask Variable
0x07	Addr:H	
0x08	Addr:L	mTouch Raw Data Array
0x09	Addr:H	
0x0A	Addr:L	mTouch Baseline Array
0x0B	Addr:H	
0x0C	Addr:L	mTouch Slider Array
0x0D	Addr:H	
0x0E	Addr:L	mTouch Toggle Variable
0x0F	Addr:H	
0x10	Addr:L	mComm GUI Custom Variable
0x11	Addr:H	

mTouch Raw Data Array in RAM	
Addr:L	Sensor0: L
Addr:H	Sensor0: h
Addr:L	Sensor1: L
Addr:H	Sensor1: h
Addr:L	Sensor2: L
Addr:H	Sensor2: h
Addr:L	...
Addr:H	...

The exchange allowing to read this data, will look like:

Type of command: Read from RAM

Request from Master:

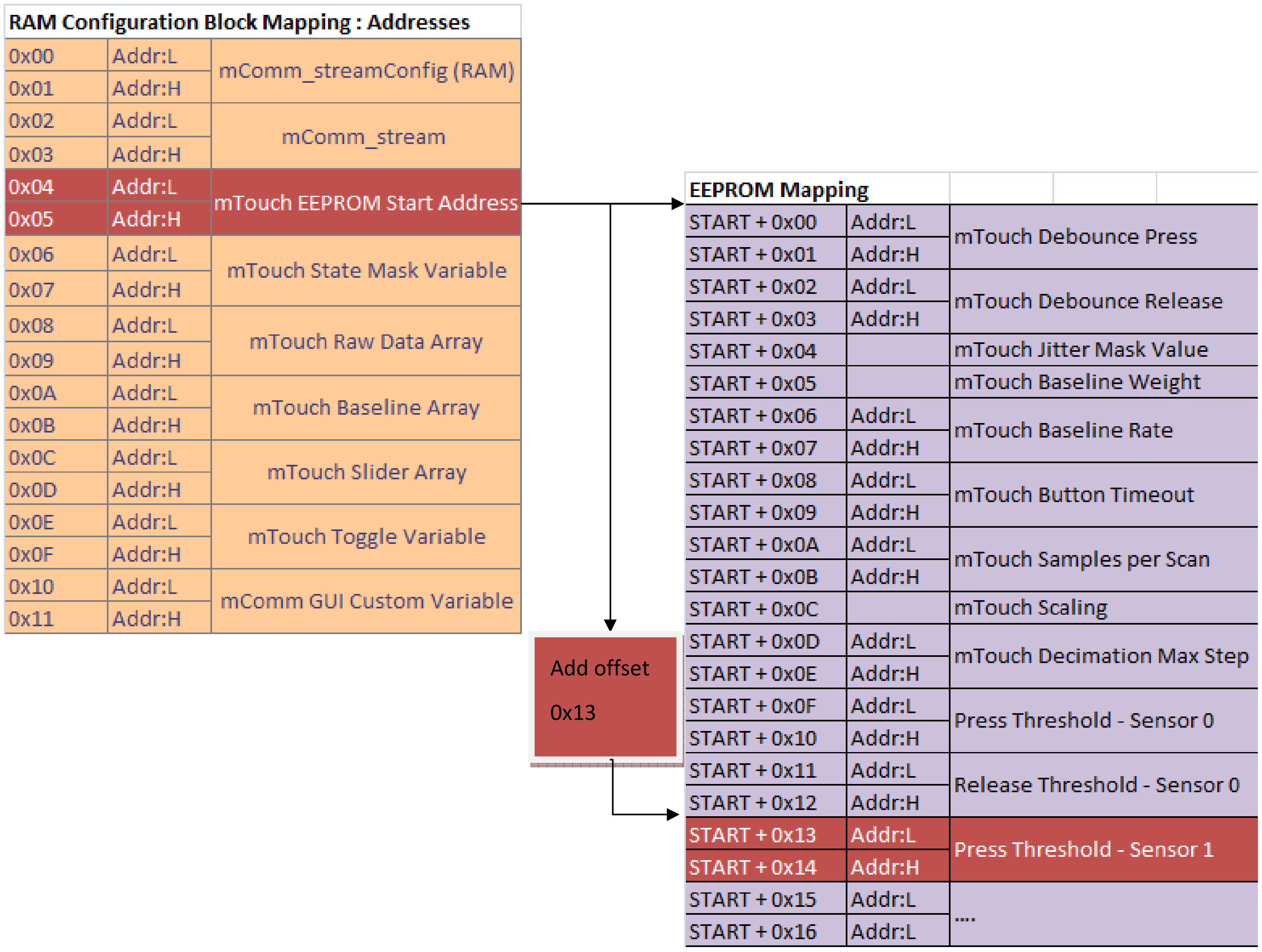
BREAK	byteCount	opCode	addr:l	addr:h	length	checksum
BREAK	0x05	0x01	0x04	0x01	0x02	0x0d

mComm response:

BREAK	byteCount	opCode	variable payload		checksum
BREAK	0x04	0x01	Raw:l	Raw:h	checksum

5.2.2 Reading the press threshold for sensor 1

Assuming the master has already retrieved the "mTouch EEPROM Start Address" in the "Pointers" block, and the start address is 0x02, then the press threshold for sensor 1 is located at 0x13, length is 2.



RAM Configuration Block Mapping : Addresses

0x00	Addr:L	mComm_streamConfig (RAM)
0x01	Addr:H	
0x02	Addr:L	mComm_stream
0x03	Addr:H	
0x04	Addr:L	mTouch EEPROM Start Address
0x05	Addr:H	
0x06	Addr:L	mTouch State Mask Variable
0x07	Addr:H	
0x08	Addr:L	mTouch Raw Data Array
0x09	Addr:H	
0x0A	Addr:L	mTouch Baseline Array
0x0B	Addr:H	
0x0C	Addr:L	mTouch Slider Array
0x0D	Addr:H	
0x0E	Addr:L	mTouch Toggle Variable
0x0F	Addr:H	
0x10	Addr:L	mComm GUI Custom Variable
0x11	Addr:H	

EEPROM Mapping

START + 0x00	Addr:L	mTouch Debounce Press
START + 0x01	Addr:H	
START + 0x02	Addr:L	mTouch Debounce Release
START + 0x03	Addr:H	
START + 0x04		mTouch Jitter Mask Value
START + 0x05		mTouch Baseline Weight
START + 0x06	Addr:L	mTouch Baseline Rate
START + 0x07	Addr:H	
START + 0x08	Addr:L	mTouch Button Timeout
START + 0x09	Addr:H	
START + 0x0A	Addr:L	mTouch Samples per Scan
START + 0x0B	Addr:H	
START + 0x0C		mTouch Scaling
START + 0x0D	Addr:L	mTouch Decimation Max Step
START + 0x0E	Addr:H	
START + 0x0F	Addr:L	Press Threshold - Sensor 0
START + 0x10	Addr:H	
START + 0x11	Addr:L	Release Threshold - Sensor 0
START + 0x12	Addr:H	
START + 0x13	Addr:L	Press Threshold - Sensor 1
START + 0x14	Addr:H	
START + 0x15	Addr:L	....
START + 0x16	Addr:L	

Here again the corresponding exchange will look like the following:

Reading from NVM

Request from Master:

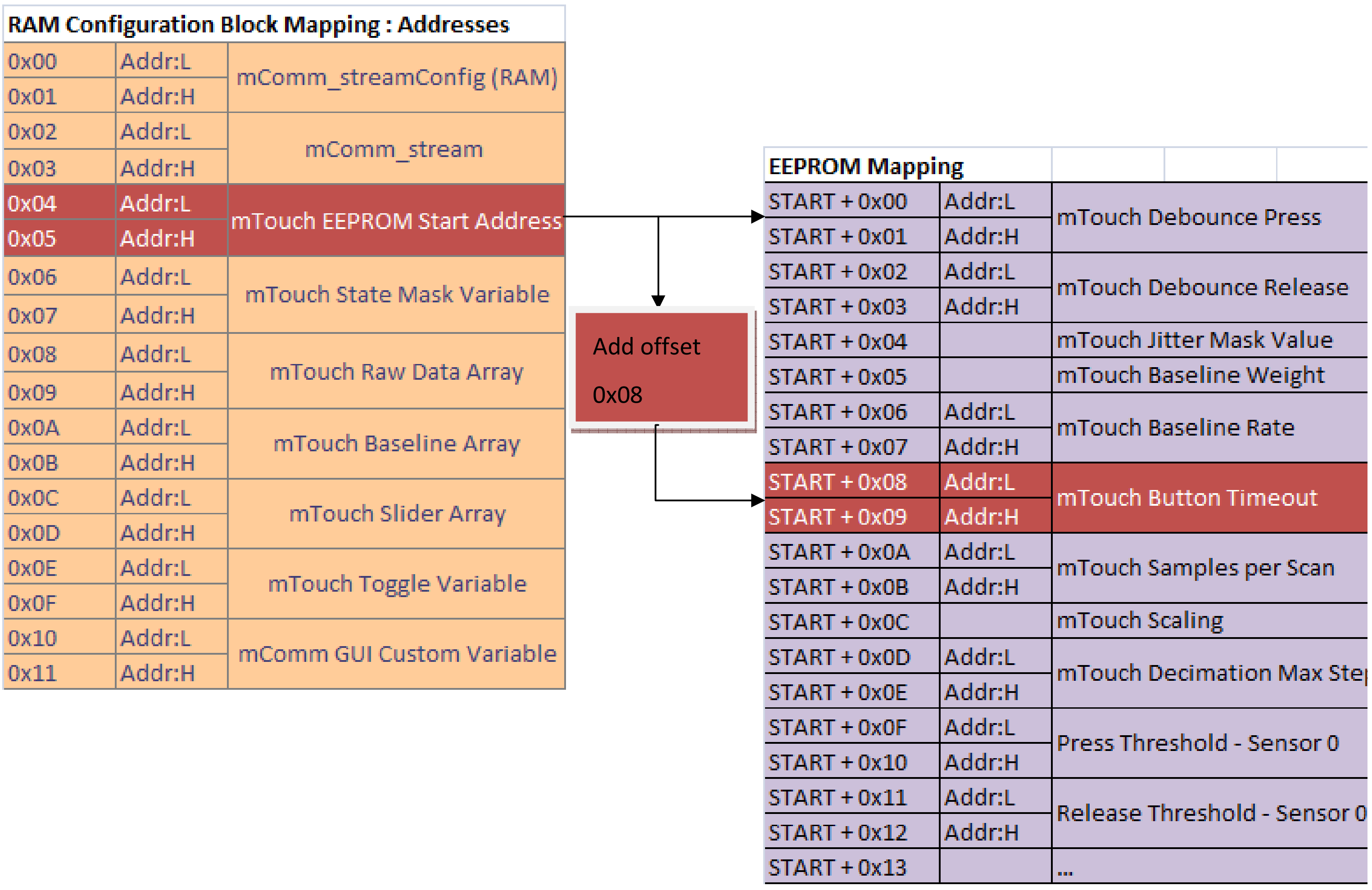
BREAK	byteCount	opCode	addr:l	addr:h	length	checksum
BREAK	0x05	0x03	0x11	0x00	0x02	0x15

mComm response:

BREAK	byteCount	opCode	variable payload		checksum
BREAK	0x04	0x01	Press Threshold - Sensor 0:L	Press Threshold - Sensor 0:H	checksum

5.2.3 Writing the mTouch Button Timeout parameter

We assume here that the master has already retrieved the “mTouch EEPROM Start Address” in the “Pointers” block, and the start address is 0x02, then the mTouch Button Timeout variable’s address will be 0x0A with a length of 2.



RAM Configuration Block Mapping : Addresses		
0x00	Addr:L	mComm_streamConfig (RAM)
0x01	Addr:H	
0x02	Addr:L	mComm_stream
0x03	Addr:H	
0x04	Addr:L	mTouch EEPROM Start Address
0x05	Addr:H	
0x06	Addr:L	mTouch State Mask Variable
0x07	Addr:H	
0x08	Addr:L	mTouch Raw Data Array
0x09	Addr:H	
0x0A	Addr:L	mTouch Baseline Array
0x0B	Addr:H	
0x0C	Addr:L	mTouch Slider Array
0x0D	Addr:H	
0x0E	Addr:L	mTouch Toggle Variable
0x0F	Addr:H	
0x10	Addr:L	mComm GUI Custom Variable
0x11	Addr:H	

EEPROM Mapping		
START + 0x00	Addr:L	mTouch Debounce Press
START + 0x01	Addr:H	
START + 0x02	Addr:L	mTouch Debounce Release
START + 0x03	Addr:H	
START + 0x04		mTouch Jitter Mask Value
START + 0x05		mTouch Baseline Weight
START + 0x06	Addr:L	mTouch Baseline Rate
START + 0x07	Addr:H	
START + 0x08	Addr:L	mTouch Button Timeout
START + 0x09	Addr:H	
START + 0x0A	Addr:L	mTouch Samples per Scan
START + 0x0B	Addr:H	
START + 0x0C		mTouch Scaling
START + 0x0D	Addr:L	mTouch Decimation Max Ste
START + 0x0E	Addr:H	
START + 0x0F	Addr:L	Press Threshold - Sensor 0
START + 0x10	Addr:H	
START + 0x11	Addr:L	Release Threshold - Sensor 0
START + 0x12	Addr:H	
START + 0x13		...

The corresponding write command structure is as follows:

Writing to NVM

Request from Master:

BREAK	byteCount	opCode	addr:l	addr:h	length	variable payload		checksum
BREAK	0x08	0x02	0x0a	0x00	2	Timeout: L	Timeout: H	checksum

mTouch device response:

If the write is successful, the PIC will respond with an ACK packet, otherwise it is either a NACK or no response.

ACK packet:

BREAK	byteCount	opCode	variable payload	checksum
BREAK	0x03	0x02	0xAA	checksum

NACK packet:

BREAK	byteCount	opCode	variable payload	checksum
BREAK	0x03	0x02	0xA5	checksum

5.3 STREAM OPERATIONS

mTouch devices, have the great capability of automatically streaming any data out to a Master Device, on a periodical basis. The mTouch programmer can decide what data is to be output and what structure will the stream have. This is extremely helpful for debugging purposes. There are two ways to configure the stream output list. One way is to configure it in the framework and save the configuration in the Flash memory space. With this approach, the master cannot modify the stream descriptor. The other approach is to store the stream descriptor in RAM, allowing the Master Device to alter the default descriptor of the stream based on the user's custom needs, using the write to RAM command to configure the list. The stream output list contains 3-byte Vectors, where the 2 first bytes would contain a pointer to any variable in the mTouch Device, while the 3rd byte holds the length of the data to be output.

Stream operation involves 3 distinct types of action:

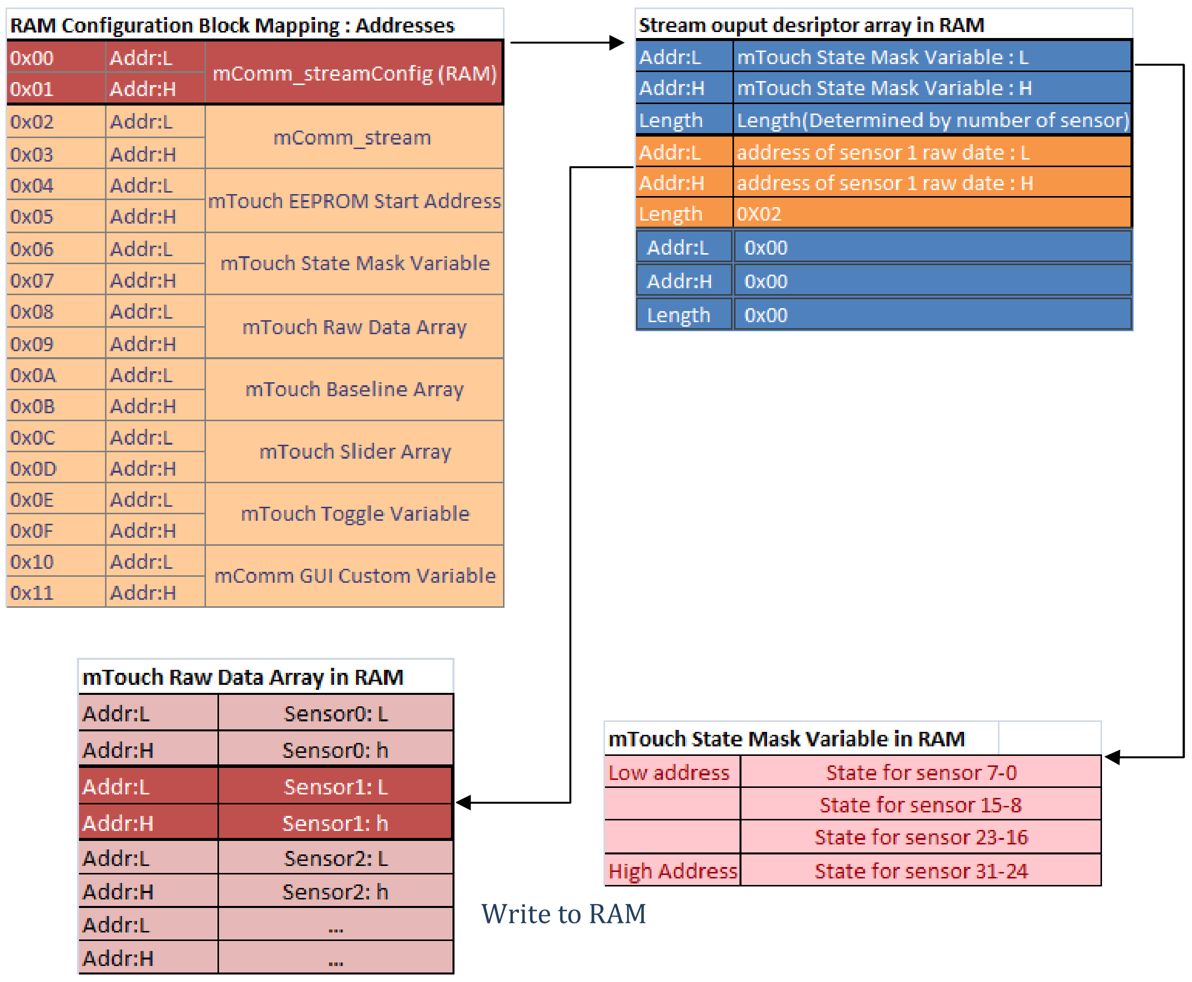
a) Configuration: where the descriptor is set and written to the mTouch device (for example the user selects the number of sensors the Slave needs to stream for)

b) Enable: Switching the stream on

c) Disable: Switching the stream Off

5.3.1 Cconfiguring the State Mask and raw data for sensor1 as stream output

We assume the master has already retrieved the “mComm_stream”, “mTouch State Mask Variable” and “mTouch Raw Data Array” in the “Addresses” block, and the “mTouch Number of Sensors” in the “Literals” block. So the RAM address that should write into is “mComm_stream”, which is the address of the stream list, then the length is 6, because 2 variables need to be output. And the payload are the “mTouch State Mask Variable” followed by its data length, and the address of raw data of sensor 1, which equals to “mTouch Raw Data Array + 4”, and its length. The length for mTouch State Mask Variable is determined by how many sensor it has, every 8 sensor uses one byte, and the length for raw data is two bytes.



RAM Configuration Block Mapping : Addresses

0x00	Addr:L	mComm_streamConfig (RAM)
0x01	Addr:H	
0x02	Addr:L	mComm_stream
0x03	Addr:H	
0x04	Addr:L	mTouch EEPROM Start Address
0x05	Addr:H	
0x06	Addr:L	mTouch State Mask Variable
0x07	Addr:H	
0x08	Addr:L	mTouch Raw Data Array
0x09	Addr:H	
0x0A	Addr:L	mTouch Baseline Array
0x0B	Addr:H	
0x0C	Addr:L	mTouch Slider Array
0x0D	Addr:H	
0x0E	Addr:L	mTouch Toggle Variable
0x0F	Addr:H	
0x10	Addr:L	mComm GUI Custom Variable
0x11	Addr:H	

Stream ouput desriptor array in RAM

Addr:L	mTouch State Mask Variable : L
Addr:H	mTouch State Mask Variable : H
Length	Length(Determined by number of sensor)
Addr:L	address of sensor 1 raw date : L
Addr:H	address of sensor 1 raw date : H
Length	0X02
Addr:L	0x00
Addr:H	0x00
Length	0x00

mTouch Raw Data Array in RAM

Addr:L	Sensor0: L
Addr:H	Sensor0: h
Addr:L	Sensor1: L
Addr:H	Sensor1: h
Addr:L	Sensor2: L
Addr:H	Sensor2: h
Addr:L	...
Addr:H	...

mTouch State Mask Variable in RAM

Low address	State for sensor 7-0
	State for sensor 15-8
	State for sensor 23-16
High Address	State for sensor 31-24

Write to RAM

Request from Master:

BREAK	byteCount	opCode	addr:l	addr:h	length
BREAK	0x09	0x00	mComm_stream:L	mComm_stream:H	6

variable payload						checksum
mTouch State Mask Variable : L	mTouch State Mask Variable :H	Length(Determined by number of sensor)	address of sensor 1 raw date : L	address of sensor 1 raw date : H	0x02	checksum

Note: The whole length of this packet is still constrained by the input buffer of the framework.

mComm response:

If write successfully, the PIC will respond with an ACK packet, otherwise it is either a NACK or no response.

ACK packet:

BREAK	byteCount	opCode	variable payload	checksum
BREAK	0x03	0x02	0xAA	checksum

NACK packet:

BREAK	byteCount	opCode	variable payload	checksum
BREAK	0x03	0x02	0xA5	checksum

After configuring the stream list either in the code or using the write to RAM packet, the master needs to turn on the stream. The output enable bit is stored in the RAM, so the master will use the write to RAM packet.

5.3.2 Enabling the Stream output

Assume the master already read the “mComm_streamConfig” in the “Addresses” block, since the enable switch is in bit 0 of a mComm_streamConfig byte, the master only needs to pay attention to that one bit.

Write to RAM

Request from Master:

BREAK	byteCount	opCode	addr:l	addr:h	length	variable payload	checksum
BREAK	0x06	0x00	mComm_streamConfig:L	mComm_streamConfig:H	1	0x01	checksum

mComm response:

If write successfully, the PIC will respond with an ACK packet, otherwise it is either a NACK or no response.

ACK packet:

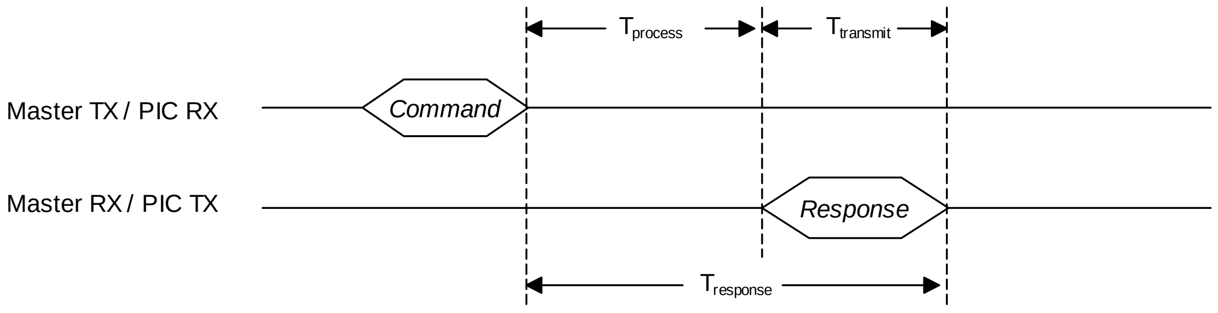
BREAK	byteCount	opCode	variable payload	checksum
BREAK	0x03	0x02	0xAA	checksum

NACK packet:

BREAK	byteCount	opCode	variable payload	checksum
BREAK	0x03	0x02	0xA5	checksum

3.2 APPENDIX:

Communication response time:



$T_{transmit}$: The time to transmit a full packet is defined as the transmit time, it is related to the length of the packet and the baud rate of the UART. If using baud rate 38400 bps, then

$$T_{transmit} = \frac{\text{Bits(Break Ch)} + \text{Bits(ByteCount)} + \text{Bits(OpCode)} + \text{Bits(Payload)} + \text{Bits(CheckSum)}}{\text{Baud rate}}$$

$$= \frac{13 + 10 + 10 + 10 * \text{bytes(Payload)} + 10}{38400}$$

$$= \frac{43 + 10 * \text{bytes(Payload)}}{38400}$$

$$T_{process} = T_{wait} + T_{command\ process\ and\ decode} + T_{EEPROM}$$

T_{wait} : This is the time between PIC receives command in the UART hardware buffer and PIC starts to decode the command. The CVD Framework scan is in the interrupt service routine, so if there is a command received by PIC during the scan, the received data will be stored in the UART hardware and won't be process until the PIC finishes this scan. Therefore, the worst case is that PIC receives a command when it just starts a scan routine, then the worst case T_{wait} equals to the time to finish one sample CVD scan.

T_{EEPROM}: This is the time for EEPROM read or write operation. If there is no need to access EEPROM like ACK command, then T_{EEPROM}=0. If the command involves accessing the EEPROM, the T_{EEPROM} =

No. of bytes * EEPROM cycle time. According to the datasheet for enhanced-mid range core, the typical and maximum EEPROM cycle time is **4m**s and **5ms**, respectively.

$T_{\text{command process and decode}}$: This is the code execution time to process the packet and perform framework decoding before sending out the response. This time depend on the system Fosc, how many sensors is enabled, if matrix or proximity decoding is enable, and how many bytes in the input buffer.