The purpose of this document is to explain & illustrate MMA9555 mailbox operation. To get an overview of the device and its embedded firmware features, the reader shall have gone already through MMA9555L.pdf complete datasheet that can be found at NXP MMA9555L Intelligent Motion-Sensing Pedometer - Data sheet

First of all, it is important to understand the chapter 4 (communication interface) of MMA9555 Data Sheet document. Maybe the wording "Mailbox" is confusing, it is simply used instead of registers. So the communication between the host MCU and MMA9555 is performed by writing and reading in those 32 mailboxes (1 byte registers with consecutive addresses from 0x00 to 0x1F). Going forward, we'll name them MBx0 to MBx1F.

There is no specific configuration needed to set-up this communication interface, besides the I2C vs SPI hardware selection (governed by the RGPIO3 pin level during reset). The slave interface will operate upfront in the "command/response" normal mode.

Interface normal mode description ("command/response" scheme)

- Host MCU sends a **command** (e.g. application configuration /data request) to MMA9555 by writing in a few consecutive registers, always starting at address 0x00 but total count will depend on the command payload size. There's a typo in Table 13 (extracted from datasheet) as first column header is not an offset but indeed the mailbox/register absolute address. Others columns headers are obviously the bit numbers for the byte to be written in the registers.

Table 13. Mailbox commands formats

Offset	7	6	5	4	3	2	1	0
0x00	Application ID (APP_ID)							
0x01	0	Command			Byte offset (upper 4 bits)			
0x02	Byte offset (lower 8 bits)							
0x03	Requested number of bytes to read/write							
0x04	Write data 0							
0x05	Write data 1							
0x06	Write data 2							
....	Write data n							

There are 4 types of commands (cf bits 6-4 in MBx01)

- 0 is used only for Application ID 0x00 (Version Application)
- 1 is used to read de configuration registers or settings of an Application
- 2 is used to change/write the configuration registers or settings of an Application
- 3 is used to read the status registers or results (output data) of an Application

Commands 0, 1 and 3 are sent to MMA9555 to query information, they simply necessitates to write in MBx0 to MBx3. Command 2 requires at least MBx0 to MBx4 as it contains also a "payload" (of 1 byte min) to write in the Application configuration registers.

The "Write data" payload bytes and the offset where to put them (MBx02) are directly linked to the Application ID involved in the command. As an example, let's use Application ID 0x12 (Sleep/Wake) to get MMA9555 device out of sleep mode (which is its state out of reset). For that, we need to clear bit0 of **cfg** register (configuration register with offset 0x06 for this application, see §13.2.5 in Datasheet).

Register	Address	Byte to write	Meaning
MBx0	0x00	0x12	Command is meant for Application ID 0x12 (Sleep/Wake)
MBx1	0x01	0x20	Write in the Application configuration registers
MBx2	0x02	0x06	Payload will start at register with offset 0x06 (which is **cfg** register)
MBx3	0x03	0x01	Payload size is a single byte
MBx4	0x04	0x00	Data to write (clear SNCEN bit-field to disable sleep mode)

We'll represent this write transaction with following notation: W00 12 20 6 1 0 (write "12 20 6 1 0" bytes into MMA9555 slave interface, starting at mailbox 0x00). Use of Hexadecimal notation is implicit.

Basically the host performs a "multiple byte write" of 5 bytes in MBx0 to MBx4.

The trick is maybe not to be confused between the MMA9555 mailbox addresses and the Application registers offset. Note that wording offset for the application registers may be overused as there's no real "base address" to declare for them. The App_ID number is enough for MMA9555 to sort things out.

1. MMA9555 provides the command **answer** to the host by updating the content of the first registers.

Table 16. Mailbox response formats

Mailbox	7	6	5	4	3	2	1	0
0x00	Application ID (APP_ID)							
0x01	COCO	Error code						
0x02	Actual number of bytes read/written							
0x03	Requested number of bytes to read/write							
0x04	Read data 0							
0x05	Read data 1							
0x06	Read data 2							
....	Read data n							

MBx0 keeps the Application ID number that has been involved

MBx1 is quite important for at least 2 reasons:

- bit7 (COCO flag) is set to 1 when command processing is complete and answer available in the "Read data" MBx. Hence it can be polled by the host MCU to know when complete response to the command can be retrieved
- bits 6-0 shall be all 0 (once COCO is set) to reflect a successful command execution otherwise they contain an Error code (cf Datasheet table 18)

MBx2 and MBx3 contain respectively the number of bytes actually processed vs requested number (i.e. MBx3 in the command). Both numbers should match when command is consistent.

Next MBx contain the requested information: settings or status data bytes whose count is indicated in MBx2. Only commands 0, 1 and 3 provide information to be read in those MBx.

<u>Note:</u> for the sake of simplicity, we will not deal with the FIFO application which can handle up to 255 bytes (using streaming mode).

As an example, the complete response to previous command W00 12 20 6 1 0 (get MMA9555 out of Sleep mode) is given in the following table.

Register	Address	Byte Read	Meaning
MBx0	0x00	0x12	Answer pertains to Application ID 0x12 (Sleep/Wake)
MBx1	0x01	0x80	COCO flag is set, command has been executed with no Error
MBx2	0x02	0x01	1 byte has been processed (command MBx4 written in cfg register)
MBx3	0x03	0x01	1 byte write was requested (cf command MBx3)

We'll represent this read transaction with following notation: R00 12 80 1 1 (reading 4 bytes on MMA9555 slave interface, starting at mailbox 0x00, yields "12 80 1 1"). Use of Hexadecimal notation is implicit.

Basically the host performs a "multiple byte read" of 4 bytes from MBx0 to MBx4.

We can do a "sanity check" by actually reading the Sleep/Wake App_ID configuration register **cfg**.

Register	Address	Byte to write	Meaning
MBx0	0x00	0x12	Command is meant for Application ID 0x12 (Sleep/Wake)
MBx1	0x01	0x10	Query Application configuration registers content
MBx2	0x02	0x06	First requested register offset is 0x06 (corresponds to **cfg** register)
MBx3	0x03	0x01	Number of configuration registers to be read (only one)

W00 12 10 6 1 query the content of Sleep/Wake Application configuration register **cfg**

Register	Address	Byte Read	Meaning
MBx0	0x00	0x12	Answer pertains to Application ID 0x12 (Sleep/Wake)
MBx1	0x01	0x80	COCO flag is set, command has been executed with no Error
MBx2	0x02	0x01	1 byte has been processed (content of **cfg** register provided)
MBx3	0x03	0x01	1 byte write was requested (cf command MBx3)
MBx4	0x04	0x00	Requested configuration register **cfg** is 0x00

R00 12 80 1 1 0 retrieve the response to previous command (make sure COCO flag is set)

Command/response synchronization consideration:

In order for the information contained in the response to be "up to date", the COCO flag shall read as 1.
We'll present 3 different methods to achieve this.

1. Use a Fix delay between Write transaction (send command) and Read transaction (get response).

   The latency between command and response depends on the tasks currently running is MMA9555 MCU (and their priority) when the command was sent. This latency amounts typically to ms but is not constant. As MMA9555 device Scheduler is running at 30.5Hz (sampling frequency for the pedometer application), a safe rule for the host MCU is to wait for 1 period of the scheduler (i.e. about 33ms) after sending a command before reading the answer.

   - Send the Command
   - Wait 33ms
   - Read the complete Answer (& confirm that COCO is set)

2. Poll the COCO flag till it is set.
    - Send the command
    - Loop: Read MBx1 register till its bit7 is 1
    - Read the complete Answer

    This method allows to stick to actual/variable latency between command & response but loads the host MCU and communication interface.

3. Configure INT_O pin as a “response ready” hardware flag in order to trigger an interrupt at the host MCU.

    This is the most elegant solution. Response latency, MCU and communication traffic load are all minimized. The polarity of the INT_O signal is also programmable.

    INT_O configuration is performed using Application ID 0x18 (MBOX Configuration application) with following command:

    W00 18 20 0 1 80   set INT_O_EN bit-field to 1 in MBOX configuration register 0x00

    R00 18 80 1 1   verify that command has been successfully processed

    Then the subsequent command-response read cycles can be handled by the host through an Interrupt associated to INT_O signal. Note that the INT_O line will be cleared as soon as the read transaction is initiated on the slave interface.

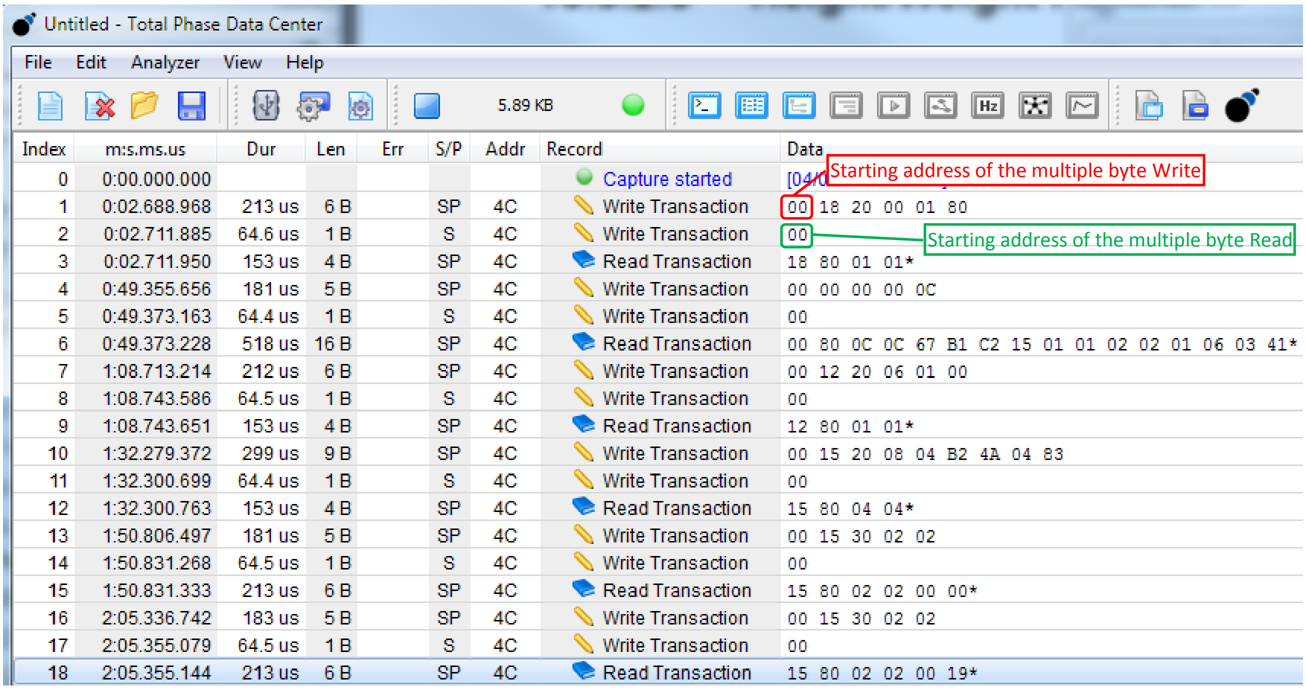
    - Send the Command
    - When INT_O is asserted, Read the complete Answer

A few words about the legacy protocol

There is a way to avoid the 2 transactions (Write-Read scheme) needed with the Command/Response default mode of the communication interface. This alternative is called the Legacy mode but applies only to MBx14 to MBx1F. Basically, Application ID 0x04 is used to assign specific status/output registers of given Applications as the “permanent” content of those mailboxes. Then those “Quick-Read” mailboxes are updated automatically every scheduler frame (30.5Hz recurrence), without the need for the host MCU to send any command to query the data.

Communication examples for MMA9555 with Pedometer application

I2C log (captured with Total Phase data Center protocol analysis SW)



Untitled - Total Phase Data Center

File Edit Analyzer View Help

5.89 KB

Index	m:s.ms.us	Dur	Len	Err	S/P	Addr	Record	Data
0	0:00.000.000						Capture started	[04/0
1	0:02.688.968	213 us	6 B		SP	4C	Write Transaction	00 18 20 00 01 80
2	0:02.711.885	64.6 us	1 B		S	4C	Write Transaction	00
3	0:02.711.950	153 us	4 B		SP	4C	Read Transaction	18 80 01 01*
4	0:49.355.656	181 us	5 B		SP	4C	Write Transaction	00 00 00 00 0C
5	0:49.373.163	64.4 us	1 B		S	4C	Write Transaction	00
6	0:49.373.228	518 us	16 B		SP	4C	Read Transaction	00 80 0C 0C 67 B1 C2 15 01 01 02 02 01 06 03 41*
7	1:08.713.214	212 us	6 B		SP	4C	Write Transaction	00 12 20 06 01 00
8	1:08.743.586	64.5 us	1 B		S	4C	Write Transaction	00
9	1:08.743.651	153 us	4 B		SP	4C	Read Transaction	12 80 01 01*
10	1:32.279.372	299 us	9 B		SP	4C	Write Transaction	00 15 20 08 04 B2 4A 04 83
11	1:32.300.699	64.4 us	1 B		S	4C	Write Transaction	00
12	1:32.300.763	153 us	4 B		SP	4C	Read Transaction	15 80 04 04*
13	1:50.806.497	181 us	5 B		SP	4C	Write Transaction	00 15 30 02 02
14	1:50.831.268	64.5 us	1 B		S	4C	Write Transaction	00
15	1:50.831.333	213 us	6 B		SP	4C	Read Transaction	15 80 02 02 00 00*
16	2:05.336.742	183 us	5 B		SP	4C	Write Transaction	00 15 30 02 02
17	2:05.355.079	64.5 us	1 B		S	4C	Write Transaction	00
18	2:05.355.144	213 us	6 B		SP	4C	Read Transaction	15 80 02 02 00 19*

Description of the I2C log using simplified notation

Log Index	Transaction type	content	comment
1	Command	W00 18 20 0 1 80	Write byte 0x80 in configuration register 0x00 of the MBOX configuration Application (App_ID 0x18). This will program INT_O pin as a “response ready” signal
2+3	Response	R00 0 80 1 1	Command completed with no error
4	Command	W00 0 0 0 C	Query the 12bytes device information packet with the Version Application (ID 0x00)
5+6	Response	R00 0 80 C C **67 B1 C2 15 1 1 2 2 1 6 3 41**	Command successfully completed, with the **12 bytes answer** collected in **MBx4 to MBxF**
7	Command	W00 12 20 6 1 0	Write 0x00 to **cfg** (configuration register 0x06) of Sleep/Wake application (App_ID 0x12)
8+9	Response	R00 12 80 1 1	Command completed with no error
10	Command	W00 15 20 8 4 **B2 4A 4 83**	Write **4 bytes** in configuration registers (0x8 to 0xB) of Pedometer application (App_ID 0x15). Settings changes from default/reset values are: height=178cm, weight=74kg, gender=male
11+12	Response	R00 15 80 4 4	Command completed with no error
13	Command	W00 15 30 2 2	Query the pedometer step count (2 bytes register)
14+15	Response	R00 15 80 2 2 **0 0**	Command successfully completed, with the **2 bytes answer** (step count = 0) collected in **MBx4 to MBx5**
16	Command	W00 15 30 2 2	Query the step count again (**after a short walk**)
17+18	Response	R00 15 80 2 2 **0 19**	Command successfully completed, with the **2 bytes answer** (step count = 25) collected in **MBx4 to MBx5**

Below screenshot is a Digital Scope plot of a command/response transaction with INT_O configured as a "response ready" flag (default polarity: active high). Trace 1 (yellow) is INT_O pin, trace 2 (magenta) is I2C serial clock line. For the record, response reading is not triggered by an interrupt based on INT_O level.