MMA955xL I2C / SPI Communication

Jacques Trichet
Sensor Applications Engineer

The MMA955xL intelligent motion sensor family has two independent serial communications ports. The slave interface and the master interface. The slave interface connects to the system host processor, the MMA955xL behaves as a slave device on the system bus. The master interface connects to other downstream sensors, and the MMA955xL acts as a bus master on this I2C bus.

The MMA955x is controlled through the slave interface which can be configured as either a slave I2C interface or a slave SPI interface.
I2C or SPI mode is configured at reset when the level of pin 8 “IO3/SDA1/SSB” is sampled. If pin 8 is high, then I2C mode is selected. If pin 8 is low, then SPI mode is selected.
The master interface is an I2C interface. Because of pin sharing with the slave SPI interface, the master I2C is only available when the slave I2C interface is configured. When the slave SPI interface is configured, the master I2C bus is not available.

The MMA955xL has a set of 32 registers or mailboxes where commands and responses are exchanged between the system host processor and the MMA955xL device. Writes and reads via the slave serial interface mailboxes configure and control the MMA955xL, and are also used to read status and data back from the MMA955xL.

This document describes the physical interfaces, and provide examples of communications and control of the MMA955xL device.

The MMA955xL “Hardware Reference Manual” as well as the “Software Reference Manual” can be consulted for further details.

There are 4 pins on the MMA955xL device that are used in the serial interfaces.

Pin #	Pin Name	I2C Function	SPI Function	Notes
4	SCL0 / RGPIO0 / SCLK	Slave I2C CLOCK	Slave SPI CLOCK	
6	SDA0 / RGPIO1 / SDI	Slave I2C DATA	Slave SPI MOSI	Master Out, Slave In
7	RGPIO2 / SCL1 / SDO	Master I2C CLOCK	Slave SPI MISO	Master In, Slave Out
8	RGPIO3 / SDA1 / SSB	Master I2C DATA	Slave SPI SLAVE SELECT	Sampled at reset 1 = selects Slave I2C 0 = selects Slave SPI

Table 1 Communication and Interface Pins

Two Command Interpreters

There are two command interpreters (CI) in the MMA955xL device family. They are slightly different. One is in the hardware ROM code, and the other is in the Flash Firmware. The MMA9559 device has the same ROM based CI, but the Firmware is user supplied, therefore the user application defines the commands and responses for the flash firmware. Depending on boot options and programming state, the MMA955xL will be running from ROM or running from FLASH. The ROM CI has a very basic limited number of commands that are explained in the Hardware Reference Manual. The FLASH based CI in the MMA9550L/51L/53L/55L devices has many more system and application level commands and is described in the associated Software Reference Manual. The MMA9559L device is fully user programmed, and will have a user specific command interface.

How to tell the difference

The two CIs respond in slightly different ways. The FLASH command interpreter has two bytes prefixed to the response data. These two bytes in mailboxes 3 and 4 show the actual number of data bytes that were returned, and the requested number of data bytes. The ROM CI does not include the data count information.

One quick way is to request the device ID information. If the FLASH command interpreter were running there would be two data bytes of 0x0C and 0x0C in mailboxes 3 and 4 because the ID command response, the MMA955xL returns 12 bytes of data.

If you don't get the "0C 0C" values in mailboxes 3 and 4 when reading version information, then you are most likely running in ROM code. Note that there is a slim chance that the device ID could have a 0x0C, 0x0C as the first bytes, so be sure to check the hardware and firmware versions also.

The ID command is described in detail further into this document.

Switching between Rom and FLASH

The host can control how the MMA955xL boots on its next boot cycle. The host can force the MMA955xL to boot to ROM or to FLASH code by sending either of these commands.
The format of these commands will be described later in the document.

Reboot to ROM: `Write@00 : 17 20 01 01 80 // Reboot to ROM`

Mail Box	Data	Description
MB0	0x17	App_ID = 0x17 (Reset Suspend Clear Application in NXP Firmware);
MB1	0x20	
MB2	0x01	
MB3	0x01	
MB4	0x80	

Reboot to FLASH: `Write@00 : 29 00 // Reboot to flash`

Mail Box	Data	Description
MB0	0x29	ROM Command Interpreter RESET command;
MB1	0x00	

I2C Interface

Here are the important parts of the I2C interfaces.

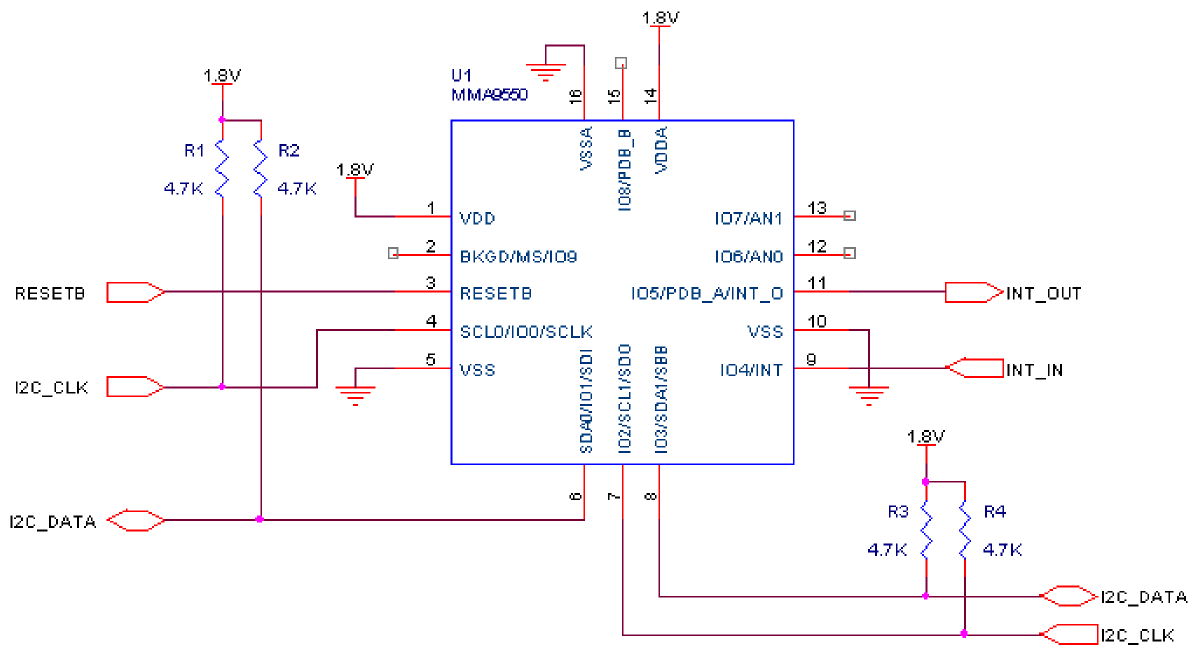


Figure 1 I2C Interface Schematic

Pin8 (RGPIO3/SDA1/SBB) must be pulled **high** at reset to select I2C mode Slave Interface. If the I2C Master Interface is implemented, the I2C pull-up resistor R3 on the master data line is sufficient to configure the I2C Slave Interface at reset.

Note that Pin8 (RGPIO3/SDA1/SBB) may be configured as a GPIO pin in a system. It is important to make sure that Pin8 is stable high during reset to properly configure the slave interface to operate into the I2C mode.

The MMA955xL device's I2C address is 0x4C. In practice, this 7-bit value is shifted left by one bit which gives an I2C write address of 0x98, and an I2C read address of 0x99. Bit 0 is the Read/Write selection bit.

I2C - ID command:

Below is an example of a simple multi byte write where a 0x00 is written to mailbox0 and mailbox1. This happens to be the command to request the device ID, software, and firmware version information.
The host knows that it will transfer a total of two bytes to complete this command.

The "multi-byte" write I2C transaction proceeds like this:

Host sends a START condition (S),
Host sends I2C Device Write Address <0x98>,
  Host waits for Acknowledge from Slave (A)
Host sends Sub address <0x00>,
  Host waits for Acknowledge from Slave (A)
Host sends Data <0x00>,
  Host waits for Acknowledge from Slave (A)
Host sends Data <0x00>,
  Host waits for Acknowledge from Slave (A)
Host now has sent all the data (two byes), now the Host sends a STOP condition (P).

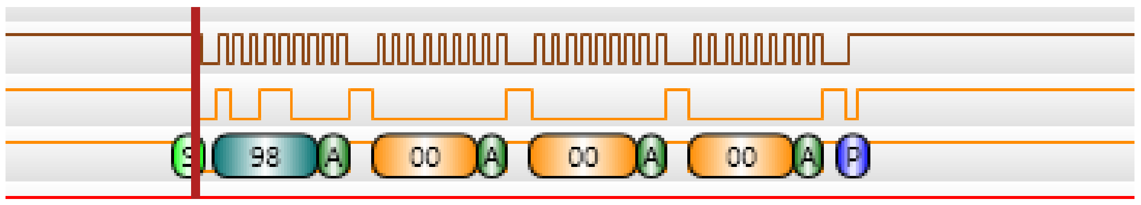


Figure 2 Physical Bus - I2C Multi-Byte Write

<u>I2C - ID ROM response:</u>

Below is an example of the ROM response to the above request. This is a multi-byte I2C read where the host reads back 17 bytes starting from mailbox 0. Note that 14 bytes would suffice.
This is the actual data read back showing the device ID, and version numbers for hardware, rom code, and firmware.

The read I2C transaction proceeds like this:

- Host sends a START condition,
- Host sends I2C Device Write Address (Bit 0 is clear),
    - Slave sends ACK, Host waits for Acknowledge from Slave
- Host sends Sub address,
    - Slave sends ACK, Host waits for Acknowledge from Slave
- Host sends a START condition (sometimes called RE-START)
- Host sends I2C Device READ Address (Bit 0 is set),
    - Host waits for Acknowledge from Slave
- Slave Sends data to Host
    - Host asserts Acknowledge so that Slave knows to continue, or
    - When the host is done reading, it doesn't send the ACK signal.
- Host sends a STOP condition.

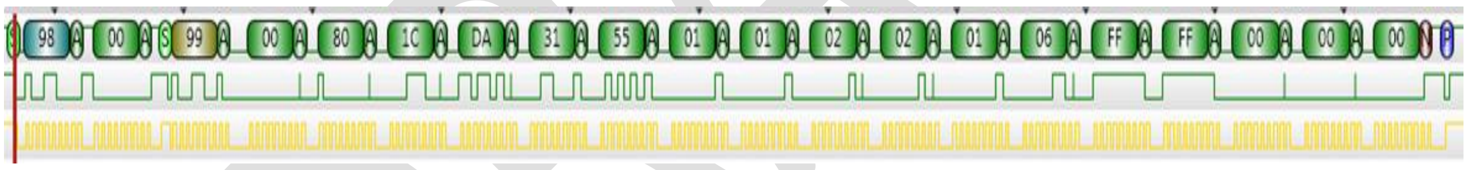


Figure 3 Physical Bus - I2C Multi Byte Read from ROM command interpreter

The bytes sequence on the SDA line is: 98 00 99 00 80 1C DA 31 55 01 01 02 02 01 06 FF FF 00 00 00.
Highlighted section is the actual identification data read back from the device through ROM Command Interpreter.

START
0x98 = I2C device address, Write to I2C device address 0x4C
0x00 = Sub Address, start reading from mailbox 0
RE-START
0x99 = I2C device address, Read from I2C device address 0x4C
0x00 = Sub Address, start reading from mailbox 0
0x80 = status, done w/ no error (COCO flag set to 1)

(*)NOTE: This example is running the ROM command interpreter.
If the FLASH command interpreter were running there would be two more data bytes inserted at this location, 0x0C and 0x0C. These two bytes show the requested number of bytes and the actual number of bytes of the I2C read transaction. The host could technically ask for more data than the MMa955xL is ready to provide, the actual byte count should be used in reading back data.

0x1CDA3155 = Device ID, a unique value for each device
0x0101 = 01.01 ROM version

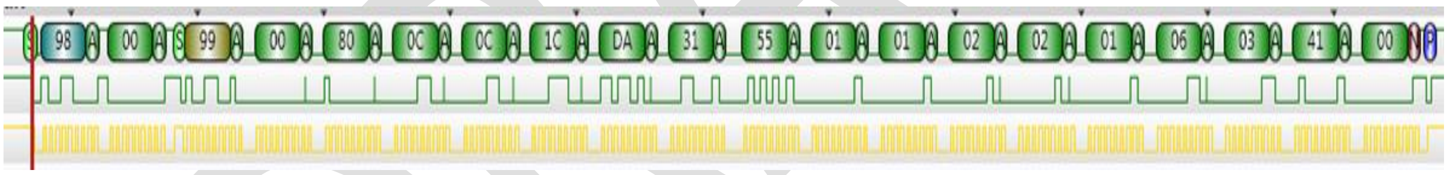
0x0202 = 02.02 Flash version
0x0106 = 01.06 HW version
0xFFFF = reserved 2 bytes (always read as 0xFFFF)
0x000000 = those last 3 bytes are not relevant and are returned simply because the host read back 17 bytes total. The former content of those mailboxes was actually not altered by the command.

I2C - ID FLASH response:

Below is an example of the FLASH response to the above request. This is a multi-byte I2C read where the host reads back 17 bytes starting from mailbox 0. Note that 16 bytes would suffice. This is the actual data read back showing the device ID, and version numbers for hardware, ROM code, and firmware.

The read I2C transaction proceeds like this:
- Host sends a START condition,
- Host sends I2C Device Write Address (Bit 0 is clear),
  - Slave sends ACK, Host waits for Acknowledge from Slave
- Host sends Sub address,
  - Slave sends ACK, Host waits for Acknowledge from Slave
- Host sends a START condition (sometimes called RE-START)
- Host sends I2C Device READ Address (Bit 0 is set),
  - Host waits for Acknowledge from Slave
- Slave Sends data to Host
  - Host asserts Acknowledge so that Slave knows to continue, or when the host is done reading, it doesn't send the ACK signal.
- Host sends a STOP condition.

Flash



The bytes sequence on the SDA line is: 98 00 99 00 80 0C 0C 1C DA 31 55 01 01 02 02 01 06 03 41 00. Highlighted section is the actual identification data read back from the device through the Version Application of NXP Flash Firmware.

START
0x98 = I2C device address, Write to I2C device address 0x4C
0x00 = Sub Address, start reading from mailbox 0
RE-START
0x99 = I2C device address, Read from I2C device address 0x4C
0x00 = Sub Address, start reading from mailbox 0
0x80 = status, done w/ no error
0x0C = Requested number of Bytes
0x0C = Actual number of bytes

(*)NOTE: This example is running the FLASH command interpreter.
If the ROM command interpreter were running these two count bytes would not be present.

0x1CDA3155 = Device ID, a unique value for each device
0x0101 = 01.01 ROM version
0x0202 = 02.02 Flash version
0x0106 = 01.06 HW version

0x0341 = Build date code (this is the only an additional information compared to the ROM CI response)
0x00 = this last byte is not relevant and is returned simply because the host reads back 17 bytes total. The former content of this mailbox is actually not altered by the command.

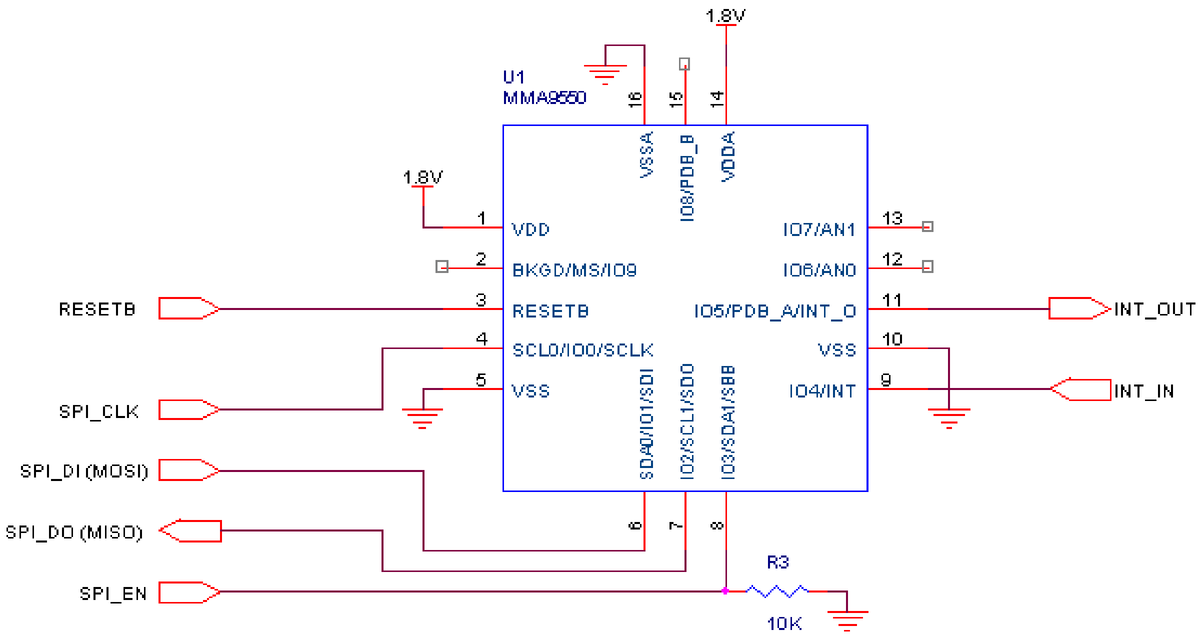
Generic I2C Information

The MMA955x I2C communication protocol follows the legacy Philips Semiconductors (now NXP Semiconductors) standard. In this interface two bus lines are defined: a data line (SDA) and a clock line (SCL). Both SDA and SCL are bidirectional lines, driven by open-collector buffers at every bus connection point. Open collector buffers need a pull-up resistor to the positive supply voltage. The recommended value is between 2.2kΩ - 4.7kΩ but this depends on bus loading and clock rates of the specific application. Many ICs can share the I2C bus, the only limitation is the bus capacitance which forms an RC circuit and limits clock and data signal rise times.

The following are five simple rules of the IIC bus to be aware of:
1. The SDA (data) and SCL (clock) cannot actively be driven high by any I2C device. I2C devices must use open-drain drivers.
2. The information on the data line is only read on the high phase of the clock.
3. Changing the level of the data is only allowed in the low phase of the clock except during start or stop conditions and this is how these events are signified.
4. When the bus is not busy SDA and SCL lines are pulled back to logic “1” by the external pull up resistors.
5. A slave device may employ clock-stretching to signal the host to wait before sending more data. The slave does this by holding the clock signal low. The host must check that the clock line did in fact go high before attempting to drive the next data and clock cycle.

SPI Interface

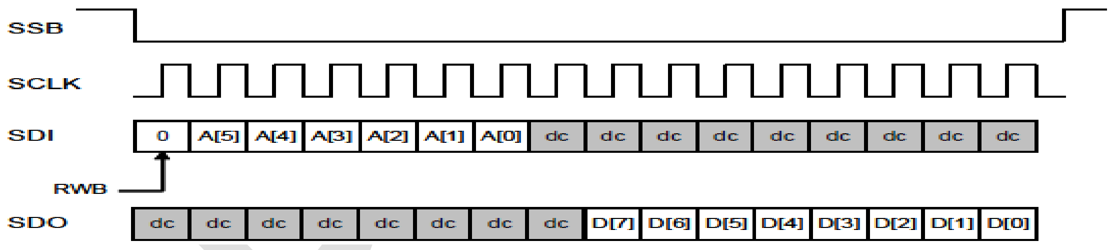
Here are the important parts if the SPI interface.



Pin8 (IO3/SDA1/SBB) must be pulled **low** at reset to select SPI mode Slave Interface. Because of pin sharing, the Master I2C bus is unavailable when the Slave SPI Interface is configured.

Note that Pin8 (IO3/SDA1/SBB) may be configured as a GPIO pin in the system. It is important to make sure that Pin8 is stable low during reset to properly configure the slave interface to operate in SPI mode.

The first byte transmitted in an SPI transaction consists of a W/R bit (1 for write, 0 for read), 6 address bits, and a don't care bit. Subsequent bytes are all data bytes.



Below is an example of a simple multi byte write where a 0x00 is written to mailbox0 and mailbox1. This happens to be the command to request the device ID, software, and firmware version information.

The write SPI transaction proceeds like this:

SSB=0, Write bit = 1, Address[5:0] = 0, Data, Data,…, SSB=1

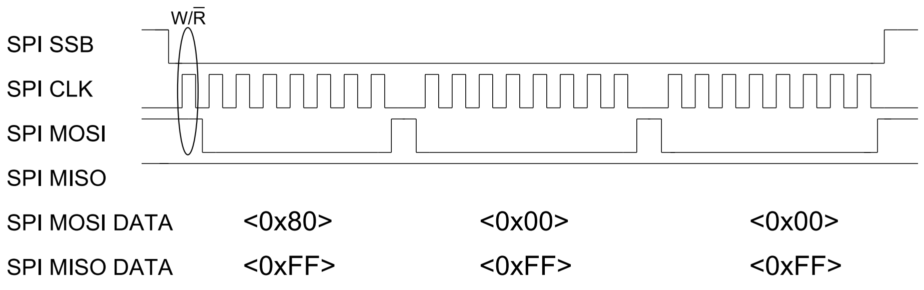


Figure 4 Physical Bus - SPI Multi Byte Write

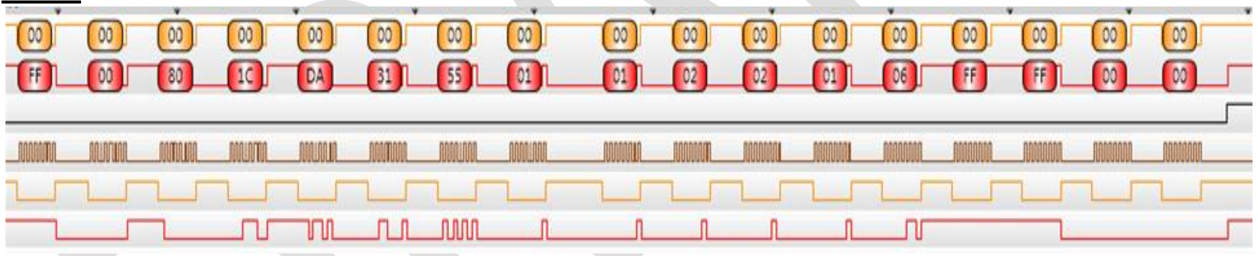
The transaction bytes sequence is:
80 00 00

0x80 = Write transaction, starting at MailBox0 (Write Bit is Set, Address = 0)
0x00 = First Data (this value will be loaded into MB0)
0x00 = Second Data (this value will be loaded into MB1)

Below is an example of the ROM response to the above request. This is a multi-byte SPI read where the host reads back 16 bytes starting from mailbox 0.
This is the actual data read back showing the device ID, and version numbers for hardware, rom code, and firmware.

Clock happens to be running at about 2.4MHz

<u>ROM</u>



```
SPI READ TRANSACTION :: Starting MB = 00, Number of bytes read = 16
  Read   : 00 80 1C DA 31 55 01 01 02 02 01 06 FF FF 00 00
```

Figure 5 Physical Bus - SPI Multi Byte Read from ROM command interpreter

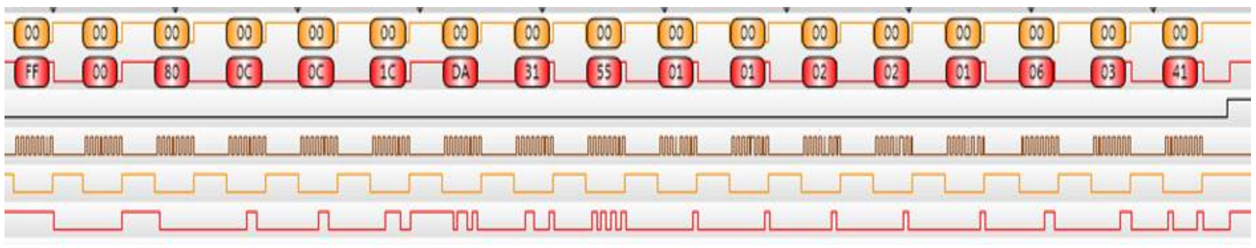
The transaction bytes sequence is:
00 00 80 1C DA 31 55 01 01 02 00 01 01

0x00 = Read starting at mailbox address 0	(on MOSI line) (W/R Bit is Clear, Addr = 0)
0x00 = repeat of command	(on MISO line)
0x80 = status, done w/ no error	

0x1CDA3155 = Part Identifier pseudo unique value
0x0101 = 01.01 ROM version
0x0200 = 02.00 Flash version
0x0101 = 01.01 HW version

As expected, answer is the same as for the I2C transaction.

<u>FLASH</u>



```
SPI READ TRANSACTION :: Starting MB = 00, Number of bytes read = 16
  Read  : 00 80 0C 0C 1C DA 31 55 01 01 02 02 01 06 03 41
```

Generic SPI Information

In the MMA955x, slave SPI communication is implemented using these 4 signal lines:

SSB (slave select active low)
SPSCK (SPI serial clock)
MOSI (master-output-slave-input, data from host to MMA955xL)
MISO (master-input-slave-output, data from MMA955xL to host)

The system microprocessor is the bus master and it initiates all SPI data transfers. The master shifts data out (on the MOSI pin) to the slave while simultaneously shifting data in on the MISO pin from the slave (SDO).
The SPSCK signal is a clock output from the master and an input to the slave.
A slave device is selected or enabled, by a low level on the slave select input, SSB pin.

In an SPI bus transaction, the first bit that is sent is the W/R bit (Write/Read) a 1 indicates a write transaction and a 0 indicates a read transaction. The next bits are the address of the targeted register.

Note the SPI W/R bit polarity is opposite from the IIC R/W bit (In I2C, a 1 indicates read and a 0 indicates write.)

The MMA955xL device can respond to SPI clock rates up to 4MHz. This is set by the SPI clock rate chosen in the MCU.

Using Mailboxes – please see the Hardware Reference Manual and Software Reference Manual for more detail and specifics on ROM and firmware applications and their use.

Every write or read operation must be done through a specific structured command.
The communication interface allows the host to configure all gesture algorithms and the operation modes of the device.

Mailbox Command Format

The command format used to write/read a gesture or application is shown below:

MB0	MB1								MB2	MB3	MB4-23
7:0	7	6	5	4	3	2	1	0	7:0	7:0	7:0
APP_ID	0	CMD			OFFSET_H				OFFSET_L	COUNT	WDATA

Table 2 Command Message Format (Write Command Structure)

MB0 (APP_ID): Identifies the application that the Host wants to write or read.

MB1 [6:4](CMD): Command Type to execute. Four different command types are available for use. These commands are:

- ***FCI_VER – 0x00***: Used to read version information of the device such as ROM, firmware and hardware versions.
- ***FCI_CONFIG_R – 0x01*** : Used to read configuration data from a specific application within the
  device.
- ***FCI_CONFIG_W – 0x02***: Used to write configuration data from a specific application within the
  device.
- ***FCI_DATA_R – 0x03:*** Command used to read data from a specific application within the device.

MB1 [3:0](OFFSET_H): High offset nibble. It is used to point to specific data from the requested application.

MB2 (OFFSET_L): Low offset byte. It is used to point to specific data from the requested application.

MB3 (COUNT): Number of bytes the host intends to transfer.

MB4-MB23 (WDATA): Data to transfer. The length depends on the payload. These fields are not used when sending a command to read data.

Mailbox Response Format

Every time a command is sent, the response data will be presented to the host following the format shown in Table 4.

MB0	MB1								MB2	MB3	MB4-23
7:0	7	6	5	4	3	2	1	0	7:0	7:0	7:0
APP_ID	CC	STATUS							BYTES_XFER	COUNT	RDATA

Table 3 Command Response Format

MB0 (APP_ID): The Application identifier from where Host request data.

MB1 [7](CC): The Command Complete Bit is set by the MMA955x to 1 when the processing of the previous command has completed. This bit signals a complete command has been processed. This bit can be polled frequently by the host to see if the command has been processed.

MB1 [6:0](STATUS): Status result of the transaction. If no error is generated during command processing, these bits will all be 0.

STATUS of 0 = no error. STATUS of = NON-Zero = error. Actual Error values are ROM and Firmware specific.

MB2 (BYTES_XFER): This is the actual number of bytes transferred from the MMA955xL to the host. This field is usually the same as MB3 (COUNT). There may be a difference between them if the host tries to request more bytes than the size of the data structure requested.

MB3 (COUNT): Actual Number of bytes requested by the host to write/read. This number comes from the command request.

MB4-23 (RDATA): Data bytes when a read command is processed. This field is only used when data is read back. The number of read back bytes depends on the number of bytes to be read.

Example on reading device ID and version information

ID COMMAND

Mail Box	Data	Command Description
MB0	0x00	App_ID = 0x00; ID
MB1	0x00	Command 0x00 = Version Information

ID RESPONSE from ROM CI

Mail Box	Data	Description
MB0	0x00	App_ID = 0x00; ID
MB1	0x80	Command Complete w/ no error codes.
MB2	0x1C	Device ID = 0x1CDA3155
MB3	0xDA	
MB4	0x31	
MB5	0x55	
MB6	0x01	= 0x0101
MB7	0x01	
MB8	0x02	= 0x0202
MB9	0x02	
MB10	0x01	= 0x0106
MB11	0x06	
MB12	0xFF	
MB13	0xFF	

ID RESPONSE from FLASH CI

Mail Box	Data	Description
MB0	0x00	App_ID = 0x00; ID
MB1	0x80	Command Complete w/ no error codes.
MB2	0x0C	12 Bytes Actually Transferred
MB3	0x0C	12 Bytes Requested to transfer
MB4	0x1C	Device ID = 0x1CDA3155
MB5	0xDA	
MB6	0x31	
MB7	0x55	
MB8	0x01	= 0x0101
MB9	0x01	
MB10	0x02	= 0x0202
MB11	0x02	
MB12	0x01	= 0x0106
MB13	0x06	
MB14	0x03	= 0x0341
MB15	0x41	

Example on setting up to read XYZ data – Normal Mode

To read some accelerometer data from the MMA955x part, first it must be commanded to wake up, then it must be commanded to return the accelerometer data.

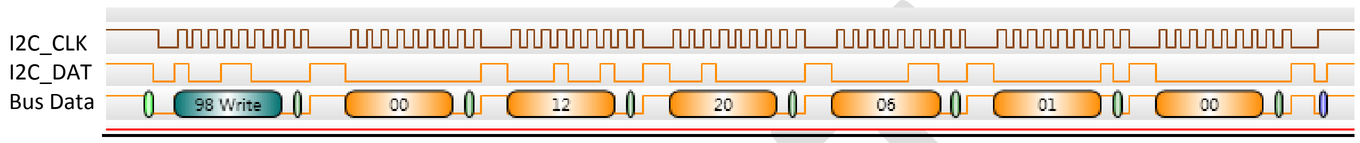
Wake up the part

```
Write : 12 20 06 01 00 // Disable sleep mode
Read  : 12 80 01 01    // Confirms the command was executed successfully
```

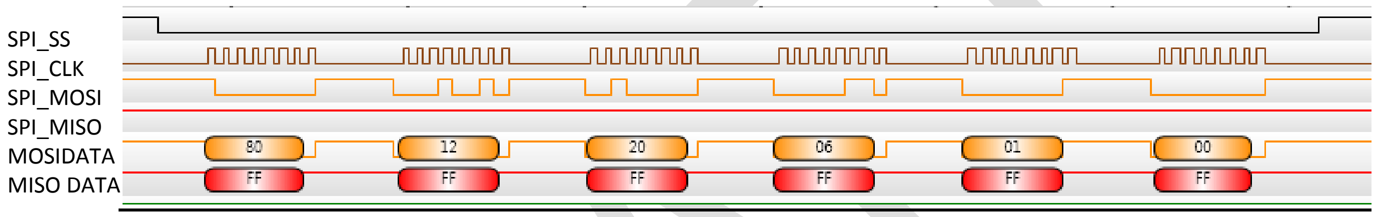
Wake Up COMMAND

Mail Box	Data	Command Description
MB0	0x12	App_ID = 0x12; Power Controller Modes
MB1	0x20	Command 0x2 = write configuration; Offset = 0
MB2	0x06	Offset = 0x06
MB3	0x01	Count of Data to write
MB4	0x00	Actual Data Value

I2C Interface - Wake Up Command



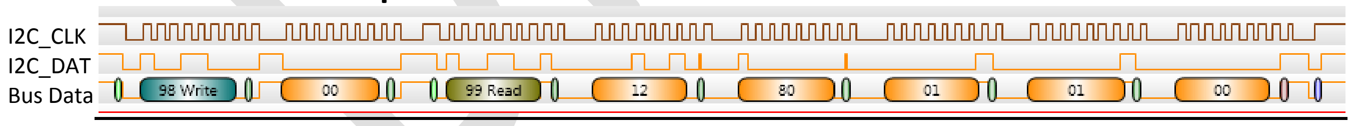
SPI Interface - Wake Up Command



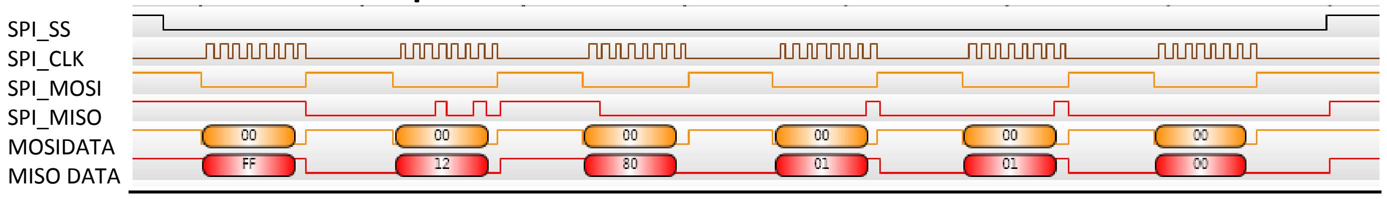
RESPONSE

Mail Box	Data	Description
MB0	0x12	App_ID = 0x12; Power Controller Modes
MB1	0x80	Command Complete w/ no error codes.
MB2	0x01	Bytes Actually Transferred
MB3	0x01	Bytes Requested to transfer
MB4	0x00	Actual Data Value;

I2C Waveform Response



SPI Waveform Response



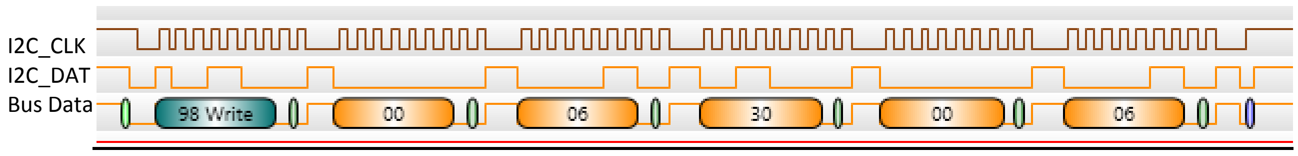
Read Normalized (Stage 0) AFE data

```
Write : 06 30 00 06 // Read normalized AFE data
Read  : 06 80 06 06 00 C8 00 13 10 01
```

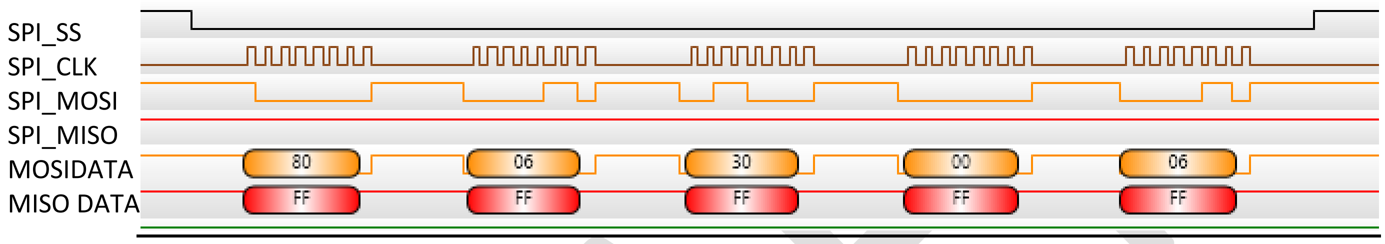
COMMAND

Mail Box	Data	Description
MB0	0x06	App_ID = 0x06; XYZ DATA
MB1	0x30	Command 0x3 = read data; Offset = 0
MB2	0x00	Offset
MB3	0x06	Bytes Requested to transfer

I2C Interface - Read Normalized XYZ Command



SPI Interface - Read Normalized XYZ Command



RESPONSE

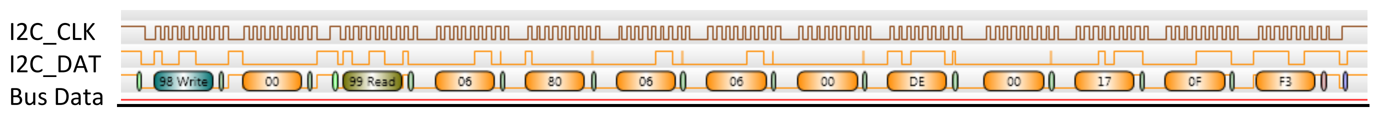
Mail Box	Data	Description
MB0	0x06	App_ID = 0x06; XYZ DATA
MB1	0x80	Command Complete w/ no error codes.
MB2	0x06	Bytes Actually Transferred
MB3	0x06	Bytes Requested to transfer
MB4	0x00	Data X – MSB
MB5	0xC8	Data X-LSB
MB6	0x00	Data Y – MSB
MB7	0x13	Data Y-LSB
MB8	0x10	Data Z – MSB
MB9	0x01	Data Z- LSB

In this example the Accelerometer XYZ measurements normalized to 0x1000 = 1g are presented to as the data payload in mailboxes MB4-MB9.
Each axis returned as a 16-bit number.

X = 0x00C8
Y = 0x0013
Z = 0x1001

The MMA955x evaluation/test board was placed horizontally on a desk surface.
If the desk were perfectly level, the X and Y values would be 0x0000, and the Z value would be 0x1000.

I2C Interface - Read Normalized XYZ Response



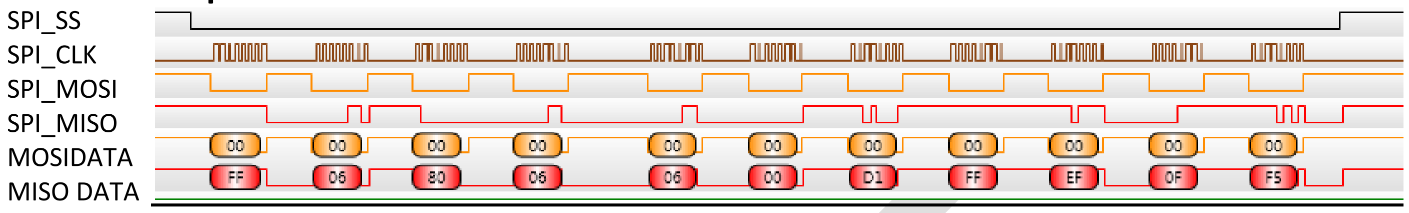
X = 0x00DE

Y = 0x0017

Z = 0x0FF3

SPI Interface - Read Normalized XYZ

Response



X = 0x00D1

Y = 0xFFEF

Z = 0x0FF5