10555 Scripps Trl, San Diego, CA 92131
858-877-3485 858-408-9550
http://www.pulserain.com

Doc# TRM-0922-01007, Rev 1.0.0

PulseRain M10 – I2C

Technical Reference Manual

Sep, 2017

This page is intentionally left blank.

Table of Contents

References

1. Specification – I2C Bus, https://www.i2c-bus.org/specification/
2. The schematic of PulseRain M10 board, Doc# SH-0922-0039, Rev 1.0, 02/2017
3. Aardvark I2C/SPI Host Adapter, https://www.totalphase.com/products/aardvark-i2cspi/
4. TXS0108E 8-Bit Bi-directional, Level-Shifting, Voltage Translator for Open-Drain and Push-Pull Applications

1 Introduction

1.1 The Whole Picture

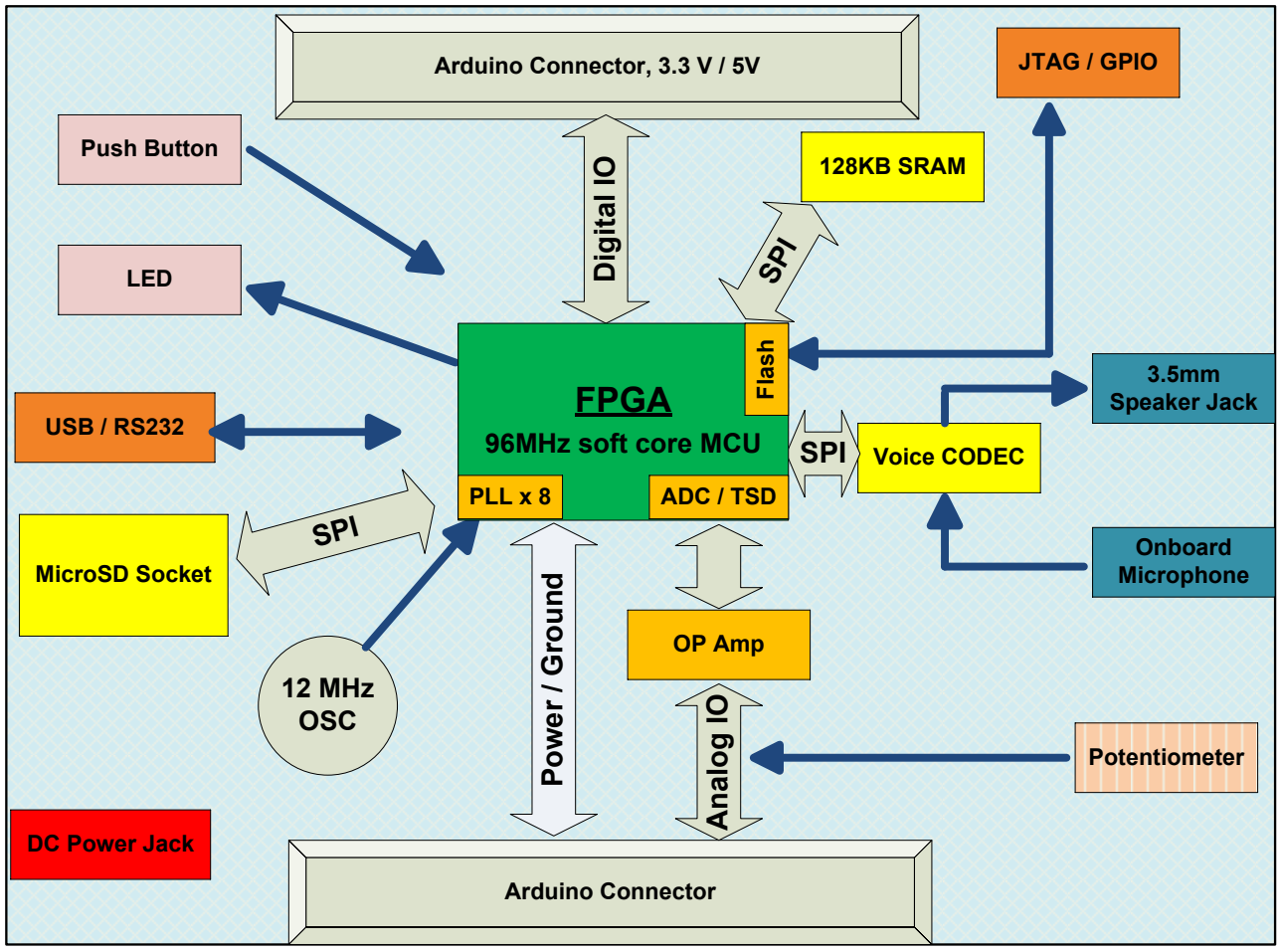

Figure 1-1 The Close View of M10

As shown Figure 1-1, the M10 board takes a distinctive technical approach by embedding an open source soft MCU core (96MHz) into an Intel MAX10 FPGA, while offering an Arduino compatible software interface and form factors. Among all the peripherals supported by the FPGA, there is an I2C Bus controller that can work in both master and slave mode, for which this document serves as a technical reference manual.

1.2 The Physical Interface

The specification of I2C bus can be found in Ref [1]. It has only two signals: SDA and SCL, for which the SDA is for the data and SCL is for the clock. Both the SDA and SCL are bidirectional so that the controller can act as either master or slave, and the data can flow in both directions.

The I2C slave device needs an address to identify itself. And the address can be either 7 bits or 10 bits, both of which are supported by the I2C controller and M10I2C library in master mode. However only the 7-bit address format is fully verified by the test equipment in reference (Ref [3]).

The SDA and SCL both need to work in the open-drain mode. Fortunately, the voltage translator (Ref [4]) used on the M10 board has built-in pull-up resistors, so that users don't have to place external pull-up in this regard. And 3.3V and 5V IOs are both supported thanks to the voltage translator.

1.3 The Golden Reference

To make sure the I2C controller functions properly, the test equipment in Ref [3] was used as a golden reference for testing purpose. And only 7-bit address mode has been fully verified by the golden reference at this point.

2 Hardware

2.1 Architecture

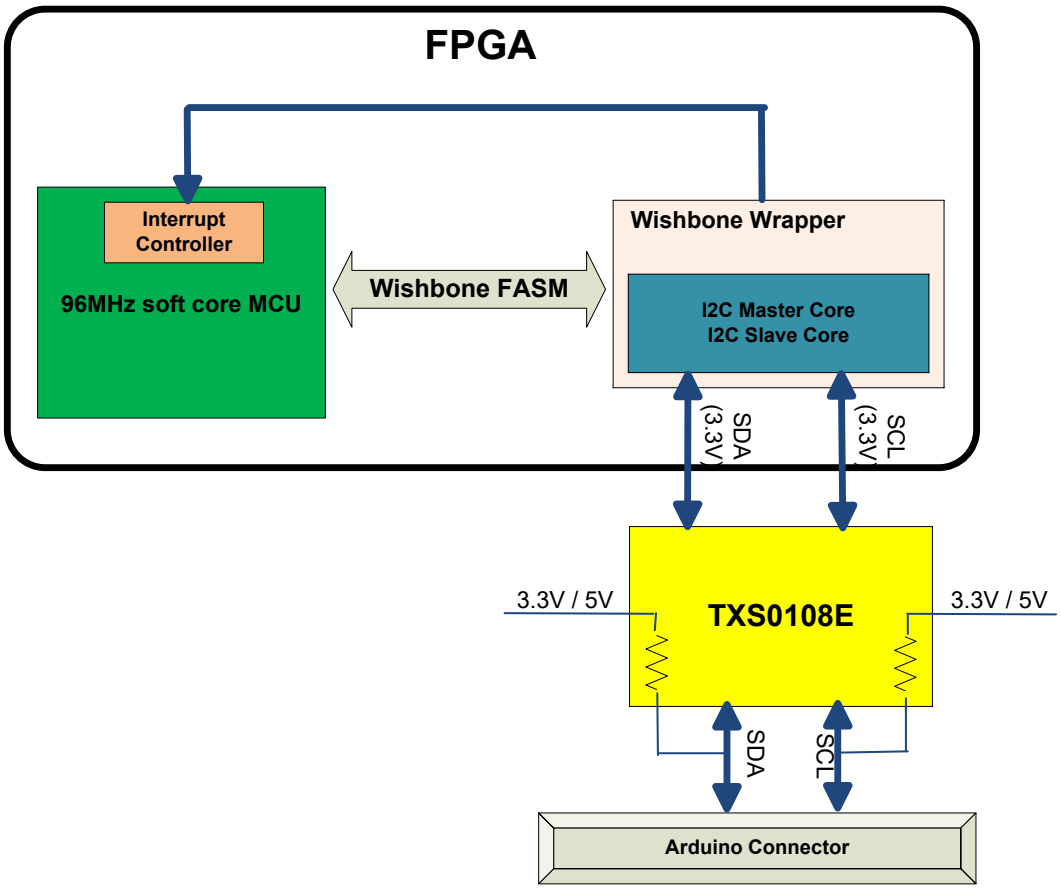

Figure 2-1 The I2C Controller

As illustrated in Figure 2-1, the I2C controller is mainly composed of two parts: the I2C controller core (Master / Slave) and the Wishbone wrapper. The Wishbone wrapper is used to communicate with the MCU core for register read/write. And the SDA / SCL from the I2C controller core will bridge through a voltage translator (TXS0108E, Ref [4]) to the Arduino Connector, so that both 3.3V and 5V IO can be supported. Internal pull-ups are also available in the voltage translator for open drain applications.

Interrupt is supported by the I2C controller for slave mode.

2.2 Arduino Compatible Pin Map

The M10 board has a connector pin map that is compatible with Arduino UNO Rev 3. As mentioned in Section 2.1, the SDA and SCL signals are connected to the Arduino Connector through a voltage translator (TXS0108E, Ref [4]), and the pin map for SDA/SCL is shown in Figure 2-2 (SDA/SCL marked by the red box at the upper right corner.)

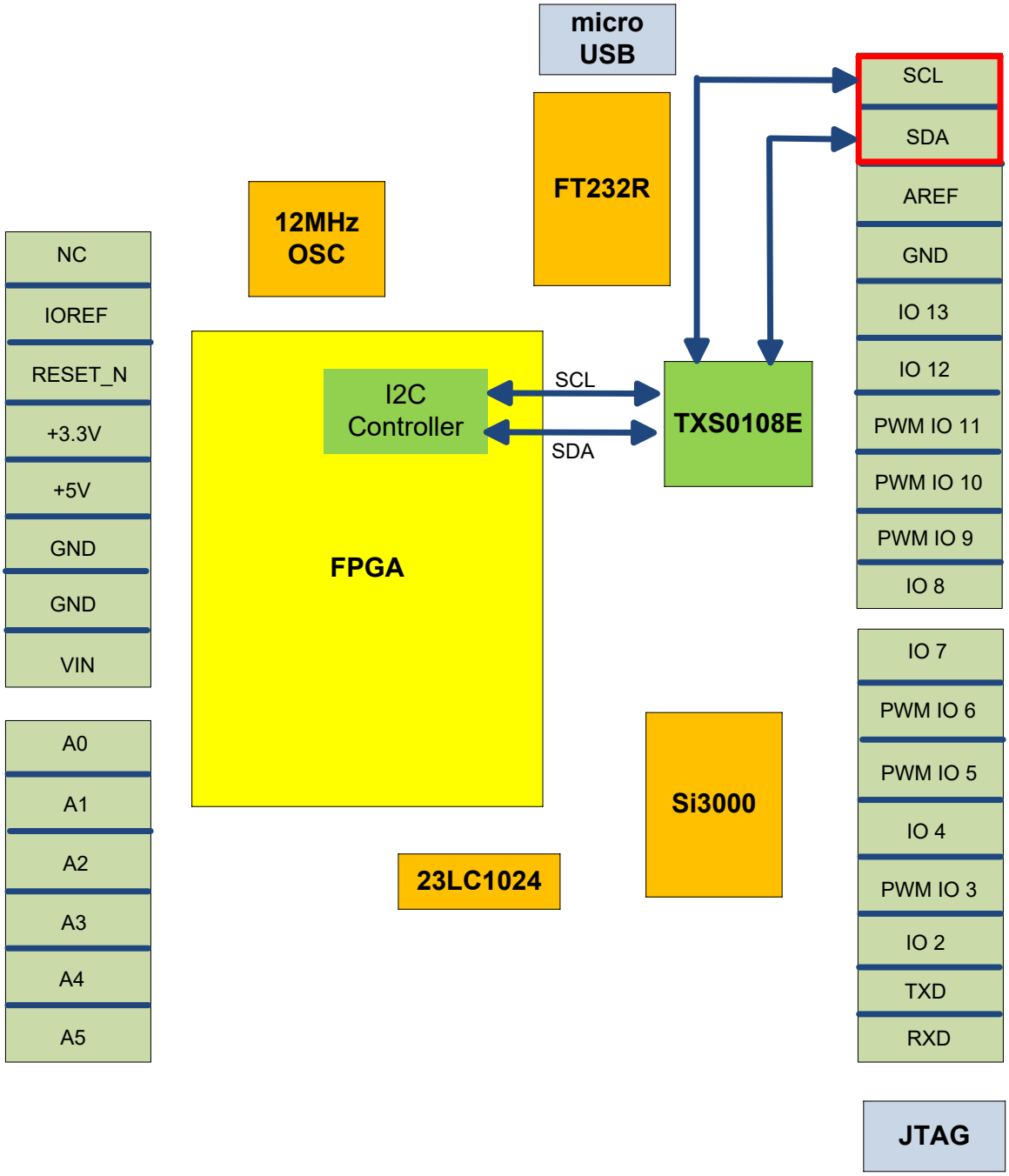

Figure 2-2 The Connector Pin Map for M10 Board

2.3 Bus Clock Rate

For master mode, the SCL clock rate is determined by the following parameters:

- ACTUAL_CLK_RATE in common_pkg.svh (default to 96 MHz)
- I2C_STANDARD_DIV_FACTOR in I2C.svh (default to ACTUAL_CLK_RATE / 100000)

And the SCL master clock rate is derived as ACTUAL_CLK_RATE / I2C_STANDARD_DIV_FACTOR, so it will be ***100 kHz*** with the above default setting.

For slave mode, the SCL is sampled by the main clock (ACTUAL_CLK_RATE, 96MHz by default). As the I2C clock usually works in the 100 kHz - 400 kHz range, 96MHz sample rate would suffice.

2.4 Port List

The port list of the master controller core is defined in Table 2-1.

Group Name	Signal Name	In/Out	Bit Width	Description
Clock / Reset	clk	Input	1	Clock input, 96MHz
	reset	Input	1	Asynchronous reset, active low
	sync_reset	Input	1	synchronous reset, active high
host interface	start	Input	1	Setting this signal to high will start a new bus transaction
	stop	Input	1	The stop can be set to high only when an ACK has been received, and action will be taken according to the following: stop = 0: No action taken stop = 1, restart = 0: a stop condition will be sent to the bus stop = 1, restart = 1: a repeated start condition will be sent
	restart	Input	1	
	read1_write0	Input	1	set this signal to high for master bus read, and set it to low for write
	addr_or_data_to_write	Input	8	The address or data to be sent to the bus, one byte at a time.
	addr_or_data_load	Input	1	Set this signal to high to latch "addr_or_data_to_write"
	data_request	Input	1	This signal will be set to high after an ACK is received.
	data_ready	Output	1	This signal will be set to high after a complete byte has been received from the bus.
	data_read_reg	Output	8	The byte data received from the bus
	no_ack_flag	Output	1	This signal will be set to high if the expected ACK is not received.
	idle_flag	Output	1	This bit will be set to high when the I2C is in idle state.
I2C Interface	sda_in	Input	1	The I2C is a bidirectional open-drain bus. For this module, the SDA/SCL are split into single directions for Input/Output. And the SDA and SCL signals are supposed to be converted to open-drain at a level above this module.
	scl_in	Input	1	
	sda_out	Output	1	
	scl_out	Output	1	

Table 2-1 Port List of the Master Controller Core

The port list of the slave controller core is defined in Table 2-2.

Group Name	Signal Name	In/Out	Bit Width	Description
Clock / Reset	clk	Input	1	Clock input, 96MHz
	reset	Input	1	Asynchronous reset, active low
	sync_reset	Input	1	synchronous reset, active high
host interface	start	Input	1	Setting this signal to high will start a new bus transaction
	stop	Input	1	Set this signal to high to send a stop condition to the bus. And the stop condition can only be sent after an ACK is received.
	outbound_data	Input	8	The byte data to be sent to the bus
	addr_or_data_load	Input	1	Set this signal to high to latch "outbound_data"
	data_request	Input	1	This signal will be set to high after an ACK is received.
	data_ready	Output	1	This signal will be set to high after a complete byte has been received from the bus.
	i2c_addr_match	Output	1	This signal will be set to high if the current I2C device address on the bus matches the slave's own device address.
	incoming_data_reg	Output	8	The byte data received from the bus
	no_ack_flag	Output	1	This signal will be set to high if the expected ACK is not received.
I2C Interface	sda_in	Input	1	The I2C is a bidirectional open-drain bus. For this module, the SDA/SCL are split into single directions for Input/Output. And the SDA and SCL signals are supposed to be converted to open-drain at a level above this module.
	scl_in	Input	1	
	sda_out	Output	1	
	scl_out	Output	1	

Table 2-2 Port List of the Slave Controller Core

2.5 Pin Assignment

The pin assignment for the microSD Controller is as following for the onboard FPGA (10M08SAE144C8G):

Signal Name	FPGA Pin Assignment (10M08SAE144C8G)	Description
I2C_SDA	101	I2C data
I2C_SCL	102	I2C clock

Table 2-3 FPGA Pin Assignment

And please note that both the I2C_SDA and I2C_SCL pins are working in open-drain mode. And on M10 board those pins rely on the pull-up from voltage translator (TXS0108E, Ref [4]) to work properly. If this FPGA design is ported to a design out of the context of M10 board, the user could choose to enable the "Weak Pull-up Resistor" option on those two pins to forgo the necessity of external pull-ups.

2.6 Repository

The RTL code for I2C controller and its Wishbone wrapper is part of PulseRain Technology's RTL library, which can be found on GitHub:

https://github.com/PulseRain/PulseRain_rtl_lib

3 Software

3.1 Register Definition

The Wishbone wrapper shown in Figure 2-1 contains all the registers to control the I2C controller. In a nutshell, the registers are defined as following:

- ADDR_DATA (8 bit)
  Bits being written to this register will be sent to the bus, and byte data received from the bus will also be placed in this register. (As the bus transaction goes, there is no explicit difference between device address and data.)

- CSR (Control and Status Register)
  The bits for CSR are defined in Table 3-1.

Bits	R/W	Default	Name	Description
0	RW	0	i2c_slave_addr_match(R) sync_reset (W)	Writing 1 to this bit will reset the I2C controller. This bit is only valid in slave mode when being read, and it will be set to high if the current I2C device address on the bus matches the slave's own device address.
1	RW	0	slave_data_request (R) start1_stop0 (W)	Writing 1 to this bit will start a new transaction on the bus. This bit is only valid in slave mode when being read, and it will be set to high after a byte data is written to the bus with ACK being received
2	RW	0	slave_data_ready (R) r1_w0 (W)	Writing to this bit will set the data transfer direction in master mode (1 for master read and 0 for master write). This bit is only valid in slave mode when being read. and it will be set to high after a complete byte data is received from the bus
3	RW	0	slave_no_ack_flag (R) master1_slave0 (W)	Writing to this bit will set the operation mode for the controller (1 for master mode and 0 for slave mode) This bit is only valid in slave mode when being read, and it will be set to high if the expected ACK is not received.
4	RW	0	master_data_request (R) restart_condition (W)	Writing 1 to this bit will let the I2C controller (master mode only) issue a restart condition instead of a stop condition at the end of the current transaction. This bit is only valid in master mode when being read, and it will be set to high after a byte data is written to the bus with ACK being received
5	RO	0	master_data_ready	This bit is only valid in master mode, and it will be set to high after a complete byte data is received from the bus
6	RO	0	master_no_ack_flag	This bit is only valid in master mode, and it will be set to high if the expected ACK is not received.
7	RW	0	master_idle_flag (R) irq_enable (W)	Write 1 to this bit will enable interrupt for slave mode. This flag is only valid in master mode when being read, and it will be set to high when the I2C controller is in idle state.

Table 3-1 Bit Map for CSR (Control and Status Register)

3.2 Address Map

The registers defined in Section 3.1 are mapped into MCU's address space, as shown in Table 3-2.

Address	Register Name
0xD1	I2C_CSR
0xD2	I2C_ADDR_DATA

Table 3-2 Address Definition for I2C Controller

3.3 Work Flow

3.3.1 Master Write

To write data to the bus in master mode, do the following:

1. Write bit 0 in CSR to reset the I2C controller.
2. Set CSR [3:2] as 2'b10 to put the I2C controller into master write mode.
3. Write the 7-bit device address into the higher 7 bits of ADDR_DATA register, and leave the LSB to be 0. (10-bit device address will be discussed in later sections.)
4. OR CSR [1] to start the master write transaction.
5. Keep reading the CSR until master_data_request or master_idle_flag is set to high.
6. Give out error message if master_data_request is not set.
7. Write the next byte of data to the ADDR_DATA register, and repeat step 5 again until all data bytes are sent.
8. set "start1_stop0" in CSR to zero to send the stop condition.

3.3.2 Master Read

The master read is a little bit more complicated than master write. For master read, two addresses are needed: one is used to identify the slave device while the other is used to identify the register inside that slave device. The master read in fact is composed of two sub-steps. The first step is actually a master write, in which the register address (not the device address) is being sent to the slave device. And the second sub-step is a read for the data sent by the slave device. In order to make sure there is no interruption between those two sub-steps, a restart condition (instead of a stop condition) is being sent to the bus at the end of the first sub-step. In this way, the bus never gets released when the operation is switching from write to read.

And do the following to read data from the slave device:

1. Do a master write to send the register address to the slave device.
2. At the end of master write, instead of sending a stop condition, sending a restart condition (set restart_condition in CSR to 1, set start1_stop0 to 0, set r1_w0 to 1)

3. Set CSR [3:2] as 2'b11 to put the I2C controller into master read mode
4. Write the 7-bit device address into the higher 7 bits of ADDR_DATA register, and leave the LSB to be 0. (10-bit device address will be discussed in later sections.)
5. set "start1_stop0" to 1 to start the read
6. Keep reading the CSR until master_data_ready or master_idle_flag is set to high.
7. Give out error message if master_data_ready is not set
8. Save ADDR_DATA to get the data received from the bus
9. Repeat step 6 until all the data are received.
10. set "start1_stop0" in CSR to zero to send the stop condition.

3.3.3 10-Bit I2C Address

The I2C specification has also defined a format for 10-bit device address (Ref [1]), in which the device address is split into two parts. The first part is a 7-bit word formatted as {5'b11110, address [9 : 8]}, while the second part is a byte of address [7 : 0]. As far as bus transaction is concerned, the second part can be treated as data for those procedures mentioned in Section 3.3.1 and 3.3.2.

3.3.4 Slave Mode

Since the slave device has to operate asynchronously to the master devices, it usually relies on interrupt to capture and respond the requests, which means the user has to provide his own call-back function for the slave mode. Examples of such call-back functions can be found in Section 3.4.2.

3.4 Arduino Library

3.4.1 APIs

To turn the work flow in Section 3.3 into APIs, PulseRain Technology has come up with the M10I2C library, which covers the following:

- *uint8_t masterWrite (uint16_t addr, uint8_t buf[], uint8_t buf_size)*

  Call this function for I2C master write.

  Parameters:

  addr: the device address for the targeted slave device
  buf: pointer to the data buffer to be sent to the slave
  buf_size: the buffer size in bytes

  Return Value:

  1: operation failed
  0: operation succeeded

- *uint8_t masterRead (uint16_t i2c_addr, uint8_t reg_addr, uint8_t buf[], uint8_t buf_size)*

  Call this function for I2C master read.

  Parameters:

  i2c_addr: the device address for the targeted slave device
  reg_addr: the address of the register inside the slave device
  buf: the data buffer to be filled
  buf_size: the size of the data buffer

  Return Value:

  1: operation failed
  0: operation succeeded

- *void slave (uint8_t addr, void (*call_back_func)(void))*

  Call this function to provide a callback function in slave mode.

  Parameters:

  addr: the device address for the slave device
  call_back_func: pointer to the call-back function

3.4.2 Examples

To further facilitate the software development, the following examples can be referenced:

- *I2C_Master*
  This example will try to write data to slave address 0x35, and read data from it with register address 0. This example has to work with an I2C host adapter for data interaction, such as the Aardvark adapter from Total Phase (Ref [3]).

- *I2C_Slave*
  This example will demonstrate how to prepare a call-back function in slave mode. And in the call-back function, data are sent and received from the master, and verified through serial console and a host adapter (such as the one in Ref [3]). The default slave address is set to be 0x29.