1. Specification

Unless stated otherwise, the given values are over lifetime, operating temperature and voltage ranges. Minimum/maximum values are ±3σ.

Parameter Specification

Parameter	Symbol	Condition	Min	Typ	Max	Units
Acceleration Range	g_{FS2g}			±2		g
	g_{FS4g}			±4		g
	g_{FS8g}			±8		g
	g_{FS16g}			±16		g
Supply Voltage Internal Domains	V_{DD}		1.72	1.8	3.6	V
Supply Voltage I/O Domain	V_{DDIO}		1.2	1.8	3.6	V
Voltage Input Low Level	V_{IL}	SPI & I²C			$0.3V_{DDIO}$	
Voltage Input High Level	V_{IH}	SPI & I²C	$0.7V_{DDIO}$			
Voltage Output Low Level	V_{OL}	V_{DDIO}=1.8V, I_{OL}=3mA, SPI			$0.2V_{DDIO}$	
		V_{DDIO}=1.2V, I_{OL}=3mA, SPI			$0.23V_{DDIO}$	
Voltage Output High Level	V_{OH}	V_{DDIO}=1.8V, I_{OH}=3mA, SPI	$0.8V_{DDIO}$			
		V_{DDIO}=1.2V, I_{OH}=3mA, SPI	$0.62V_{DDIO}$			
Total Supply Current in Normal mode	I_{DD}	Nominal VDD and VDDIO, 25°C, OSR=3		14.5		µA
		OSR=0		3.5		µA
Total Supply Current in Sleep Mode	I_{DDsum}	Nominal VDD and VDDIO, 25°C		160		nA
Total Supply Current in Low-power Mode	I_{DDlp1}	Nominal VDD and VDDIO, 25°C 25 Hz ODR OSR=0		850		nA
Wake-Up Time	t_{w_up}	From sleep to normal mode			2/ODR	
Power-Up Time	t_{s_up}	Starting the device to sleep mode			1	ms
Operating Temperature	T_A		-40		+85	°C

4.4. **Sensor Data**

Acceleration Data

The width of acceleration data is 12 bits given in two´s complement representation in the registers 0x04 to 0x09 (ACC_X_LSB, ACC_X_MSB, ACC_Y_LSB, ACC_Y_MSB, ACC_Z_LSB, ACC_Z_MSB). The 12 bits for each axis are split into an MSB upper part (4 bit) and an LSB lower part (8 bit).
In order to ensure the integrity of the acceleration data read, the content of all data registers must be read in a single burst read, since these registers are write-protected during a read access. As soon as the burst read is finished the register content will be updated if new data are available.

Filter Configuration

Two major filter paths are implemented, see block diagram. Filter output can either be fed into the data registers, into the FIFO, or used to process interrupts in the interrupt engine. This is selectable by customer.

Filter1 (acc_filt1) has a data rate between 800Hz and 12.5Hz, controlled by ACC_CONFIG0.acc_odr. Its bandwidth can be configured additionally by ACC_CONFIG0.filt1_bw:

- ACC_CONFIG0.filt1_bw = 0x0 →0.48 x ODR
- ACC_CONFIG0.filt1_bw = 0x1 →0.24 x ODR

Please note that the filt1_bw = 0x1 setting is invalid when the sensor is in a) low power mode or b) auto low power feature enabled during normal mode.

ACC_CONFIG0.acc_odr <3:0>	Output Data Rate [Hz]
0xB .. 0xF	800
0xA	400
0x9	200
0x8	100
0x7	50
0x6	25
0x0 .. 0x5	12.5

Filter2 (acc_filt2) has a fixed data rate of 100 Hz.
In addition, these 100 Hz data of filter2 can be used by a low pass filter (*acc_filt_lp*) with a bandwidth of 1 Hz. The output data rate will stay at 100 Hz. This data can be used as input for the data registers and also in the interrupt engine. Access via FIFO is not possible.

ACC_CONFIG2.data_src_reg<1:0>	Filter output going into data registers (not FIFO!)
0x0,0x3	acc_filt1(selectable ODR)
0x01	acc_filt2 (48Hz BW, 100Hz ODR)
0x02	acc_filt_lp (1 Hz BW, 100 Hz ODR)

FIFO_CONFIG0.fifo_data_src	Filter output going into FIFO
0x0	acc_filt1(selectable ODR)
0x1	acc_filt2 (100Hz ODR)

In low-power mode, only data at 25Hz ODR is available. Depending on the setting of ACC_CONFIG0.osr_lp, noise and current consumption is controllable. *But other than this, please note that no other configuration is possible during low power mode.*

5. Register description

Addr (hex)	RegName	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Rst val	Access mode
0x00	CHIPID	chipid<7:0>								0x90	R
0x01	reserved	reserved								0x88	R
0x02	ERR_REG							cmd_err		0x00	R
0x03	STATUS	drdy			cmd_rdy		power_mode<1:0>		int_active	0x00	R
0x04	ACC_X_LSB	acc_x<7:0>								0x00	R
0x05	ACC_X_MSB	0	0	0	0	acc_x<11:8>				0x00	R
0x06	ACC_Y_LSB	acc_y<7:0>								0x00	R
0x07	ACC_Y_MSB	0	0	0	0	acc_y<11:8>				0x00	R
0x08	ACC_Z_LSB	acc_z<7:0>								0x00	R
0x09	ACC_Z_MSB	0	0	0	0	acc_z<11:8>				0x00	R
0x0A	SENSOR_TIME0	sensor_time<7:0>								0x00	R
0x0B	SENSOR_TIME1	sensor_time<15:8>								0x00	R
0x0C	SENSOR_TIME2	sensor_time<23:16>								0x00	R
0x0D	EVENT								por_detected	0x00	R
0x0E	INT_STAT0	drdy_int	fwm_int	ffull_int	ieng_overrun	gen2_int	gen1_int	orientch_int	wkup_int	0x00	R
0x0F	INT_STAT1				ieng_overrun	d_tap_int	s_tap_int	step_int<1:0>		0x00	R
0x10	INT_STAT2				ieng_overrun		actch_z_int	actch_y_int	actch_x_int	0x00	R
0x11	TEMP_DATA	temp_data<7:0>								0x00	R
0x12	FIFO_LENGTH0	fifo_bytes_cnt<7:0>								0x00	R
0x13	FIFO_LENGTH1						fifo_bytes_cnt<10:8>			0x00	R
0x14	FIFO_DATA	fifo_data<7:0>								0x00	R
0x15	STEP_CNT0	step_cnt<7:0>								0x00	R
0x16	STEP_CNT1	step_cnt<15:8>								0x00	R
0x17	STEP_CNT2	step_cnt<23:16>								0x00	R
0x18	STEP_STAT							step_stat<1:0>		0x00	R
0x19	ACC_CONFIG0	filt1_bw	osr_lp<1:0>					power_mode<1:0>		0x00	RW
0x1A	ACC_CONFIG1	acc_range<1:0>		osr<1:0>		acc_odr<3:0>				0x49	RW
0x1B	ACC_CONFIG2					data_src_reg				0x00	RW
0x1F	INT_CONFIG0	drdy_int	fwm_int	ffull_int		gen2_int	gen1_int	orientch_int		0x00	RW
0x20	INT_CONFIG1	latch_int			actch_int	d_tap_int	s_tap_int		step_int	0x00	RW

Register (0x04) ACC_X_LSB

DESCRIPTION: Register for accelerometer data. The ACC_X_LSB-ACC_Z_MSB registers contain the latest data for x, y and z axis of accelerometer. A read operation on the register ACC_X_LSB-ACC_Z_MSB resets the INT_STAT0.drdy_int (data ready bit).
RESET: 0x00
DEFINITION (Go to register map):

Name	**Register (0x04) ACC_X_LSB**			
Bit	7	6	5	4
Read/Write	R	R	R	R
Reset Value	0	0	0	0
Content	acc_x_7_0			
Bit	3	2	1	0
Read/Write	R	R	R	R
Reset Value	0	0	0	0
Content	acc_x_7_0			

acc_x_7_0: lsb of accelerometer x-axis data acceleration is obtained by the following operations:
acc_x/y/z = acc_x/y/z_7_0 + 256*acc_x/y/z_11_8
if(acc_x/y/z> 2047) acc_x/y/z=acc_x/y/z-4096

Register (0x05) ACC_X_MSB

DESCRIPTION: Register for accelerometer data. The ACC_X_LSB-ACC_Z_MSB registers contain the latest data for x, y and z axis of accelerometer. A read operation on the register ACC_X_LSB-ACC_Z_MSB resets the INT_STAT0.drdy_int (data ready bit).
RESET: 0x00
DEFINITION (Go to register map):

Name	**Register (0x05) ACC_X_MSB**			
Bit	7	6	5	4
Read/Write	n/a	n/a	n/a	n/a
Reset Value	0	0	0	0
Content	reserved			
Bit	3	2	1	0
Read/Write	R	R	R	R
Reset Value	0	0	0	0
Content	acc_x_11_8			

acc_x_11_8: msb of accelerometer x-axis data acceleration is obtained by the following operations:
acc_x/y/z = acc_x/y/z_7_0 + 256*acc_x/y/z_11_8
if(acc_x/y/z> 2047) acc_x/y/z=acc_x/y/z-4096

ACC_Y/Z are omitted from print because they are identical to ACC_X other than register address (see previous page).

Register (0x1A) ACC_CONFIG1

DESCRIPTION: the registers contain the accelerometer configuration
RESET: 0x49
DEFINITION (Go to register map):

Name	**Register (0x1A) ACC_CONFIG1**			
Bit	7	6	5	4
Read/Write	RW	RW	RW	RW
Reset Value	0	1	0	0
Content	acc_range		osr	
Bit	3	2	1	0
Read/Write	RW	RW	RW	RW
Reset Value	1	0	0	1
Content	acc_odr			

acc_odr: output data rate of accelerometer for acc_filt1

acc_odr	**ODR [Hz]**
0x00	reserved
0x01	reserved
0x02	reserved
0x03	reserved
0x04	reserved
0x05	12.5
0x06	25
0x07	50
0x08	100
0x09	200
0x0a	400
0x0b	800
0x0c	reserved
0x0d	reserved
0x0e	reserved
0x0f	reserved

osr: oversampling rate 0/1/2/3 for normal mode
osr=0: lowest power, lowest oversampling rate, lowest accuracy
osr=3: highest accuracy, highest oversampling rate, highest power
settings 0, 1, 2 and 3 allow linearly trading power versus accuracy(noise)

acc_range: accelerometer measurement range

acc_range		
0x00	2g	+/-2g measurement range
0x01	4g	+/-4g measurement range
0x02	8g	+/-8g measurement range
0x03	16g	+/-16g measurement range

6. Digital Interfaces

6.1. **Interface**

By default, the BMA400 operates in I2C mode. The BMA400 interface can also be configured to operate in a SPI 4-wire configuration. It can also be re-configured by software to work in 3-wire mode instead of 4-wire mode.
All three possible digital interfaces share partly the same pins. The mapping for the primary interface of the BMA400 is given in the following table:

Pin#	Name	I/O Type	Description	Connect to (Primary IF)		
				in SPI4W	in SPI3W	in I2C
1	SDO	Digital I/O	Serial data output in SPI Address select in I²C mode see chapter 7.2	SDO	DNC (float)	GND for default I2C addr.
2	SDX	Digital I/O	SDA serial data I/O in I²C SDI serial data input in SPI 4W SDA serial data I/O in SPI 3W	SDI	SDA	SDA
5	INT1	Digital I/O	Interrupt output 1 (default)	INT1	INT1	INT1
6	INT2	Digital I/O	Interrupt output 2 (default)	INT2	INT2	INT2
10	CSB	Digital in	Chip select for SPI mode	CSB	CSB	V_{DDIO}
12	SCX	Digital in	SCK for SPI serial clock SCL for I²C serial clock	SCK	SCK	SCL

* If INT1 and/or INT2 are not used, please do not connect them (DNC).

The following table shows the electrical specifications of the interface pins:

Parameter	Symbol	Condition	Min	Typ	Max	Units
Pull-up Resistance, CSB pin	R_{up}	Internal Pull-up Resistance to VDDIO		120		kΩ
I²C Bus Load Capacitance (max. drive capability)	C_{I2C_Load}				400	pF

6.4. **Primary I2C Interface**

The I²C bus uses SCL (= SCx pin, serial clock) and SDA (= SDx pin, serial data input and output) signal lines. Both lines Must be connected to V_{DDIO} externally via pull-up resistors so that they are pulled high when the bus is free.
The default I²C address of the device is b001010X . 'X' is defined by the SDO pin: if SDO pulled to ´GND´ X equals 0, is SDO is pulled to VDDIO, X equals 1. In I2C, the SDO level must be defined, it cannot be left floating.
The I²C interface of the BMA400 is compatible with the I²C Specification UM10204 Rev. 06 (April 2014), available at http://www.nxp.com. The BMA400 supports **I²C standard mode and fast mode**, only 7-bit address mode is supported. For V_{DDIO} = 1.2V to 1.62 V the guaranteed voltage output levels are slightly relaxed as described in Table 1 of the electrical specification section.
BMA400 also supports an **extended I²C mode** that allows using clock frequencies up to 3.4 MHz. In this mode all timings of the fast mode apply and it additionally supports clock frequencies up to 3.4MHz.

The timing specification for I²C of the BMA400 is given in the following table:

Parameter	Symbol	Condition	Min	Max	Units
Clock Frequency	f_{SCL}			3400	kHz
SCL Low Period	t_{LOW}		1.3		µs
SCL High Period	t_{HIGH}		0.6		
SDA Setup Time	t_{SUDAT}		0.1		
SDA Hold Time	t_{HDDAT}		0.0		
Setup Time for a repeated Start Condition	t_{SUSTA}		0.6		
Hold Time for a Start Condition	t_{HDSTA}		0.6		
Setup Time for a Stop Condition	t_{SUSTO}		0.6		
Time before a new Transmission can start	t_{BUF}	normal mode	1.3		

I²C write access:
I²C write access can be used to write a data byte in one sequence.

The sequence begins with start condition generated by the master, followed by 7 bits slave address and a write bit (RW = 0). The slave sends an acknowledge bit (ACKS = 0) and releases the bus. Then the master sends the one byte register address. The slave again acknowledges the transmission and waits for the 8 bits of data which shall be written to the specified register address. After the slave acknowledges the data byte, the master generates a stop signal and terminates the writing protocol.

Example of an I²C write access:

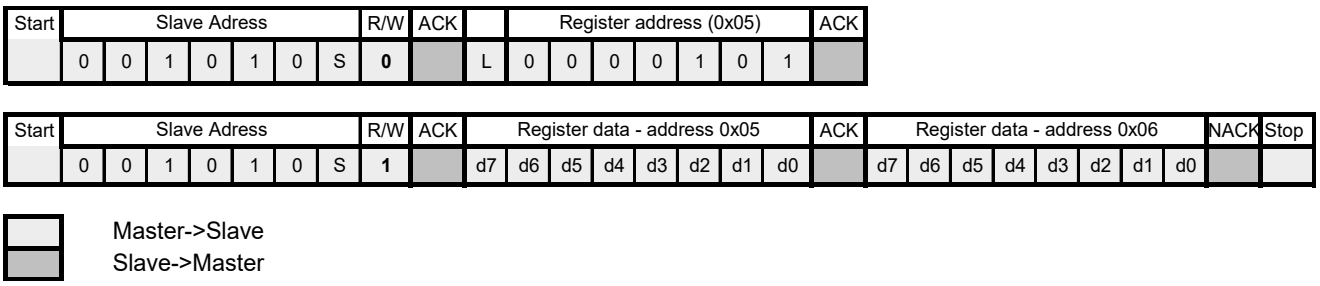


Multi-byte writes are supported without restriction on normal registers.



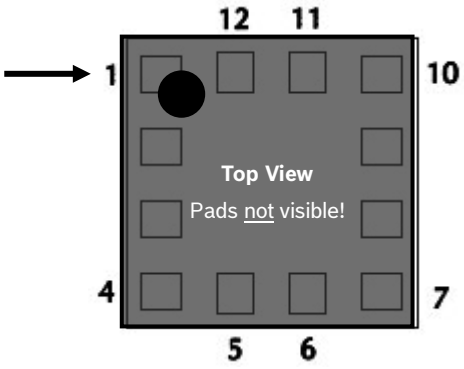
I²C read access:

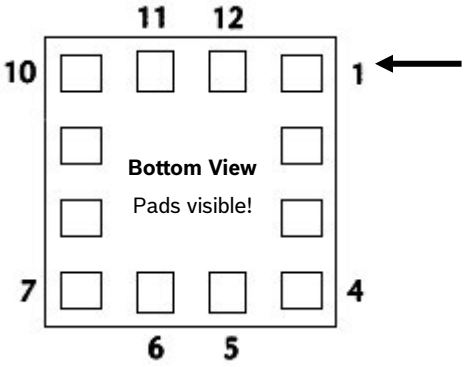
I²C read access also can be used to read one or multiple data bytes in one sequence.
A read sequence consists of a one-byte I²C write phase followed by the I²C read phase. The two parts of the transmission must be separated by a repeated start condition (S). The I²C write phase addresses the slave and sends the register address to be read. After slave acknowledges the transmission, the master generates again a start condition and sends the slave address together with a read bit (RW = 1). Then the master releases the bus and waits for the data bytes to be read out from slave. After each data byte the master has to generate an acknowledge bit (ACKS = 0) to enable further data transfer. A NACKM (ACKS = 1) from the master stops the data being transferred from the slave. The slave releases the bus so that the master can generate a STOP condition and terminate the transmission.
The register address is automatically incremented and, therefore, more than one byte can be sequentially read out. Once a new data read transmission starts, the start address will be set to the register address specified since the latest I²C write command. By default the start address is set at 0x00. In this way repetitive multi-bytes reads from the same starting address are possible.



7. Pin-out and Connection Diagrams

7.1. **Pin-out**

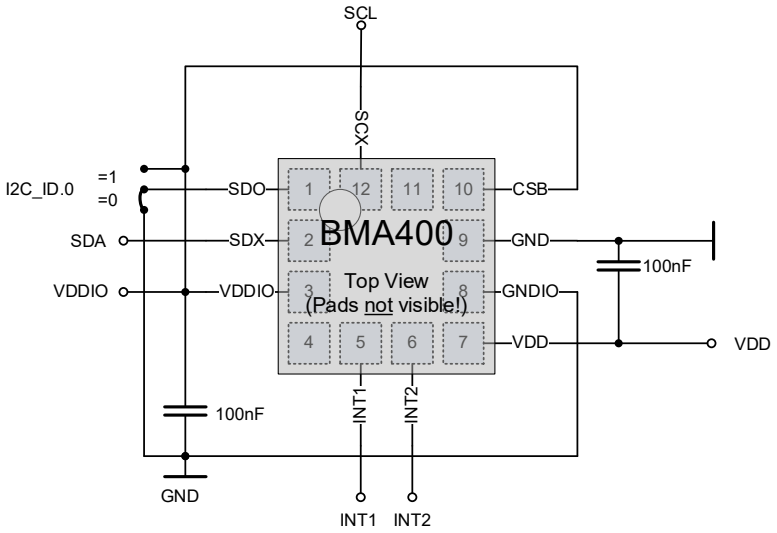




Pin description

Pin#	Name	I/O Type	Description	Connect to		
				in SPI 4W	In SPI 3W	in I²C
1	SDO	Digital I/O	Serial data output in SPI Address select in I²C mode see chapter 7.2	SDO	DNC (float)	GND for default I2C addr.
2	SDX	Digital I/O	SDA serial data I/O in I²C SDI serial data input in SPI 4W SDA serial data I/O in SPI 3W	SDI	SDA	SDA
3	VDDIO	Supply	Digital I/O supply voltage (1.2V … 3.6V)	V_{DDIO}	V_{DDIO}	V_{DDIO}
4	NC					
5	INT1	Digital I/O	Interrupt output 1 (default)	INT1	INT1	INT1
6	INT2	Digital I/O	Interrupt output 2 (default)	INT2	INT2	INT2
7	VDD	Supply	Power supply for analog & digital domain (1.62V … 3.6V)	V_{DD}	V_{DD}	V_{DD}
8	GNDIO	Ground	Ground for I/O	GND	GND	GND
9	GND	Ground	Ground for digital & analog	GND	GND	GND
10	CSB	Digital in	Chip select for SPI mode	CSB	CSB	V_{DDIO}
11	NC					
12	SCX	Digital in	SCK for SPI serial clock SCL for I²C serial clock	SCK	SCK	SCL

I2C



It is recommended to use 100nF decoupling capacitors at pin 3 (VDDIO) and pin 7 (VDD).

Maximum Ratings

Parameter	Symbol		Values
Operating Temperature	T_{op}	min. max.	-40 °C 125 °C
Storage Temperature	T_{stg}	min. max.	-40 °C 125 °C
Junction Temperature [2)]	T_j	max.	125 °C
Forward Current [2)]	I_F	max.	1000 mA
Forward Current pulsed t ≤ 10 ms; D ≤ 0.1 ; T_C = 25 °C	$I_{F\ pulse}$	max.	1500 mA
ESD withstand voltage acc. ANSI/ESDA/JEDEC JS-001 (HBM, Class 0)	V_{ESD}		ESD sensitive device
Reverse voltage [3)]	V_R		Not designed for reverse operation

Characteristics

I_F = 700 mA; T_C = 25 °C; all chips operated in series

Parameter	Symbol		Values
Luminous Flux [4)]	Φ_V	typ.	470 lm
Dominant Wavelength [5)] I_F = 700 mA	λ_{dom}	min. typ. max.	587.5 nm 590.0 nm 597.5 nm
Viewing angle at 50% I_V	2φ	typ.	100 °
Forward Voltage [6)4)] I_F = 700 mA	V_F	min. typ. max.	8.96 V 9.96 V 11.60 V
Reverse current [3)]	I_R		Not designed for reverse operation
Electrical thermal resistance junction/case	$R_{thJC\ elec.}$	typ.	2.8 K / W

Sections taken from:

Arduino API - A reference to the Arduino Programming Language.

Author: Karl Söderby **Last revision: 02/18/2024**

Digital I/O

Method & Parameters	Description	Returns
`int digitalRead(int pin)`	Reads the state of a digital pin.	`int`
`void digitalWrite(int pin, int state)`	Writes a state to a digital pin.	Nothing
`void pinMode(int pin, int mode)` *	Define the mode of a pin.	Nothing

*Available modes are:

- `INPUT` (0)
- `OUTPUT` (1)
- `INPUT_PULLUP` (2)
- `INPUT_PULLDOWN` (3)
- `OUTPUT_OPENDRAIN` (4)

Analog I/O

Method & Parameters	Description	Returns
`int analogRead(int pin)`	Reads the value of an analog pin in a 10-bit resolution (0-1023).*	`int`
`void analogReadResolution(int resolution)`	Sets ADC read resolution in bits.	Nothing
`void analogReference(int reference)`	Changes the voltage reference for a board.**	Nothing
`void analogWrite(int pin, int value)`	Writes a value to a PWM supported pin in a 8-bit resolution (0-255).**	Nothing
`void analogWriteResolution(int resolution)`	Sets write resolution for a board.	Nothing

- *The value range changes based on the resolution. 0-1023 is 10-bit resolution, 0-4096 is 12-bit and so on.
- **Each board/architecture has a set of different reference voltages available.
- ***The value range changes based on the resolution. 0-255 is default (8-bit).

Time

Method & Parameters	Description	Returns
`void delay(long milliseconds)`	Freezes program execution for specified number of **milliseconds**.	Nothing
`void delayMicroseconds(int microseconds)`	Freezes program execution for specified number of **microseconds**.	Nothing
`long millis()`	Returns **milliseconds** passed since program start.	`long`
`long micros()`	Returns **microseconds** passed since program start.	`long`

Enums

Enum Type	Enumeration	Description
`PinStatus`	`HIGH` / `LOW`	Logical HIGH and LOW values (`1` and `0`).
`PinMode`	`INPUT` / `OUTPUT` / `INPUT_PULLUP` / `INPUT_PULLDOWN` / `OUTPUT_OPENDRAIN`	Constants for specifying pin modes (`0` , `1` , `2` , `3` , `4`).
	`LED_BUILTIN`	Constant representing the built-in LED pin.*
	`true / false`	Boolean constants for true and false (`1` and `0`).

Conversion

Method & Parameter	Description
`(unsigned int)`	Type casting to unsigned int.
`(unsigned long)`	Type casting to unsigned long.
`byte()`	Type casting to byte.
`char()`	Type casting to char.
`float()`	Type casting to float.
`int()`	Type casting to int.
`long()`	Type casting to long.
`word()`	Type casting to word.

I2C

#include <Wire.h>

Wire.beginTransmission(addr); // Begin a transmission to the I2C servant device with the given address.
// Subsequently, queue bytes for transmission with the write() function
// and transmit them by calling endTransmission().

Wire.write(val); // Writes data from a servant device in response to a request from a
// master, or queues bytes for transmission from a master to servant
// device (in-between calls to beginTransmission() and
// endTransmission()).

Wire.endTransmission(); // Ends a transmission to a servant device that was begun by
// beginTransmission() and transmits the bytes that were queued by
// write().

Wire.requestFrom(addr, num) //Send read request for 'num' bytes to device with I2C address 'addr'

Wire.available() //Is data we've asked to read available on the I2C bus? Returns
//how many bytes are available.

Wire.read() // Reads a byte that was transmitted from a servant device to a master
// after a call to requestFrom() or was transmitted from a master to a
// servant. **This is a blocking transaction**. If a NACK is received, function returns 0.

Note: *Only the pins **A4 and A5** can be used as I2C pins. It is set automatically by the Wire library. **A4 is SDA and A5 is SCL**.*

I2C example

```
#include <Wire.h>
void setup()
{
  Wire.begin();         // join i2c bus (address optional for master)
  Serial.begin(9600);   // start serial for output
}

void loop()
{
  Wire.requestFrom(2, 6);     // request 6 bytes from servant device #2
  while(Wire.available())     // servant may send less than requested
  {
    char c = Wire.read();     // receive a byte as character
    Serial.print(c);          // print the character
  }

  delay(500);
}
```

SPI

Default SPI Pins on Arduino UNO: MOSI: GPIO 11; MISO: GPIO 12; CLK: GPIO 13; SS: GPIO 10

SPI.begin() : Initializes the SPI pins to SS = 1, SCLK = 0 , MOSI = 0;

SPISettings my_spi_setting(speed, data order, mode):

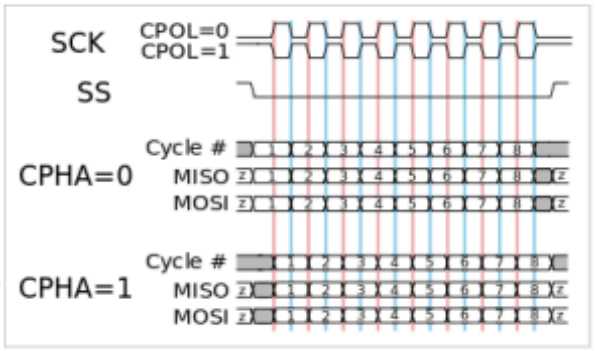
my_spi_setting is global that contains the following after execution

speed: integer expressed in Hz

data order: MSBFIRST or LSBFIRST

mode: SPI_MODE0, SPI_MODE1, SPI_MODE2, and SPI_MODE3

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



SPI.beginTransaction(SPI_settings):
Initializes the SPI bus with the settings in SPI_settings

SPI.endTransaction(): Ends a SPI transaction

receivedVal = **SPI.transfer**(val): Sends an 8-bit value on the SPI bus. At the same time it reads the value from the servant and returns the value.

SPI sample code:

```
#include <SPI.h>

	// Example with incompatible SPI devices (i.e they need different SPI_MODE
	const int servantAPin = 20;
	const int servantBPin = 21;

	// set up the speed, data order and data mode
	SPISettings settingsA(2000000, MSBFIRST, SPI_MODE1);
	SPISettings settingsB(16000000, LSBFIRST, SPI_MODE3);

	void setup() {
	  // set the Servant Select Pins as outputs and drive them high.
	  pinMode (servantAPin, OUTPUT); digitalWrite (servantAPin, HIGH);
	  pinMode (servantBPin, OUTPUT); digitalWrite (servantBPin, HIGH);
	  SPI.begin();
	}

	uint8_t stat, val1, val2, result;

	void loop() {
	  // read three bytes from device A
	  SPI.beginTransaction(settingsA);   digitalWrite (servantAPin, LOW);
	  // reading only, so data sent does not matter
	  stat = SPI.transfer(0);   val1 = SPI.transfer(0);   val2 = SPI.transfer(0);
	  digitalWrite (servantAPin, HIGH);
	  SPI.endTransaction();
	  // if stat is 1 or 2, send val1 or val2 else zero
	  if (stat == 1) {
	   result = val1;
	  } else if (stat == 2) {
	   result = val2;
	  } else {
	   result = 0;
	  }
	  // send result to device B
	  SPI.beginTransaction(settingsB);
	  digitalWrite (servantBPin, LOW);
	  SPI.transfer(result);
	  digitalWrite (servantBPin, HIGH);
	  SPI.endTransaction();
	}
```

<math.h>: Mathematics

Macros

#define **M_PI** 3.14159265358979323846 /* pi */

Functions

float **cosf** (float __x)

float **sinf** (float __x)

float **tanf** (float __x)

float **sqrtf** (float __x)

float **acosf** (float __x)

float **asinf** (float __x)

float **atanf** (float __x)

Detailed Description

```
#include <math.h>
```

This header file declares basic mathematics constants and functions.

Notes:

- In order to access the functions declared herein, it is usually also required to additionally link against the library `libm.a`. See also the related **FAQ entry**.
- Math functions do not raise exceptions and do not change the `errno` variable. Therefore the majority of them are declared with const attribute, for better optimization by GCC.

Macro Definition Documentation

◆ M_PI

#define M_PI 3.14159265358979323846 /* pi */

The constant *pi*.

Function Documentation

float acosf (float __x)

The **acosf()** function computes the principal value of the arc cosine of *__x*. The returned value is in the range [0, pi] radians. A domain error occurs for arguments not in the range [-1, +1].

float asinf (float __x)

The **asinf()** function computes the principal value of the arc sine of *__x*. The returned value is in the range [-pi/2, pi/2] radians. A domain error occurs for arguments not in the range [-1, +1].

float atanf (float __x)

The **atanf()** function computes the principal value of the arc tangent of *__x*. The returned value is in the range [-pi/2, pi/2] radians.

float cosf (float __x)

The **cosf()** function returns the cosine of *__x*, measured in radians.

float sinf (float __x)

The **sinf()** function returns the sine of *__x*, measured in radians.

float sqrtf (float __x)

The **sqrtf()** function returns the non-negative square root of *__x*.

float tanf (float __x)

The **tanf()** function returns the tangent of *__x*, measured in radians.