

I2C Library

0.1.0

Project Overview

Contents

1	Changelog for the I2C Library	2
2	Background	3
3	Class Index	4
3.1	Class List	4
4	File Index	5
4.1	File List	5
5	Class Documentation	6
5.1	_I2C_Device Struct Reference	6
5.1.1	Detailed Description	6
5.1.2	Member Data Documentation	6
5.2	_I2C_Port Struct Reference	8
5.2.1	Detailed Description	8
5.2.2	Member Data Documentation	8
6	File Documentation	9
6.1	Changelog.md File Reference	9
6.2	i2c.c File Reference	9
6.3	i2c.h File Reference	9
6.3.1	Detailed Description	10
6.3.2	Typedef Documentation	10
6.3.3	Enumeration Type Documentation	11
6.3.4	Function Documentation	13
6.4	i2c.o.d File Reference	16
6.5	i2c.o.d File Reference	16
6.6	i2c.o.d File Reference	16
6.7	i2c.o.d File Reference	16
6.8	README.md File Reference	16

1 Changelog for the I2C Library

Release v0.1.0 - [2018-11-07]

1. Initial release of library. Validated to work on:
 - PIC32MX370F512L
 - PIC32MX795F512H
 2. Developed using Microchip's legacy library on MPLAB 8 and then ported over to MPLAB X.
-

2 Background

The 'I2C Library' provides a simple I2C implementation for the following platform(s):

- PIC32MX

Dependencies

This project is dependent upon the following projects:

1. [doxygen](#)

- Used to generate <docs> folder output.
- Cloned instance should be named "doxygen" and live as a sibling to this repository.

2. [lib-timing](#)

- Directory cloned into must be <lib-timing>.
- Directory must be a sibling to the clone of this repository.

Detailed Overview

For complete details on how to use, modify, and expand this utility, please see the provided [Doxygen Summary](#)

Development History

For complete details on the what was changed for the latest release, please see the [Changelog.md](#)

Licensing

All code is provided 'as is'. You are free to modify, distribute, etc. the code within the bounds of the [Mozilla Public License \(v2.0\)](#).

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_I2C_Device	
Tracks the device-specific I2C information	6
_I2C_Port	
Tracks I2C port settings (e.g. which module and clock speed for communication)	8

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

i2c.c	
Implements functions used to interact with the I2C bus and peripheral devices	9
i2c.h	
Defines public constants, macros, and constant functions used to interact with an external I2C bus	9
lib-i2c--pic32mx370f512h.X/build/default/debug/_ext/1360937237/i2c.o.d	16
lib-i2c--pic32mx370f512h.X/build/default/production/_ext/1360937237/i2c.o.d	16
lib-i2c--pic32mx795f512l.X/build/default/debug/_ext/1360937237/i2c.o.d	16
lib-i2c--pic32mx795f512l.X/build/default/production/_ext/1360937237/i2c.o.d	16

5 Class Documentation

5.1 `_I2C_Device` Struct Reference

Tracks the device-specific I2C information.

```
#include <i2c.h>
```

Collaboration diagram for `_I2C_Device`:

Public Attributes

- [I2C_Port](#) `port`
Structure tracking I2C port specific details.
- [I2C_MODE](#) `mode`
Mode for I2C device (master, slave, etc.).
- `uint16_t` [addr](#)
Identifier for device (e.g. 0x50).
- [I2C_ADDR_LEN](#) `addrLength`
Length of I2C address format (e.g. 7-bits).
- `uint32_t` [delayAfterSend_Ms](#)
Amount of time (in milliseconds) to optionally delay processing after successful sending a payload over the bus.
- `uint32_t` [delayAfterReceive_Ms](#)
Amount of time (in milliseconds) to optionally delay processing after successful reading a payload from the bus.
- `uint32_t` [delayBetweenTxRx_Ms](#)
Amount of time (in milliseconds) to optionally delay processing within a TxRx request after sending a payload and before reading from the bus.

5.1.1 Detailed Description

Tracks the device-specific I2C information.

Note

Also encapsulates the '`I2C_Port{}`' information for easier tracking.

5.1.2 Member Data Documentation

5.1.2.1 `addr`

```
uint16_t _I2C_Device::addr
```

Identifier for device (e.g. 0x50).

5.1.2.2 **addrLength**

`I2C_ADDR_LEN` `_I2C_Device::addrLength`

Length of I2C address format (e.g. 7-bits).

5.1.2.3 **delayAfterReceive_Ms**

`uint32_t` `_I2C_Device::delayAfterReceive_Ms`

Amount of time (in milliseconds) to optionally delay processing after successful reading a payload from the bus.

5.1.2.4 **delayAfterSend_Ms**

`uint32_t` `_I2C_Device::delayAfterSend_Ms`

Amount of time (in milliseconds) to optionally delay processing after successful sending a payload over the bus.

5.1.2.5 **delayBetweenTxRx_Ms**

`uint32_t` `_I2C_Device::delayBetweenTxRx_Ms`

Amount of time (in milliseconds) to optionally delay processing within a TxRx request after sending a payload and before reading from the bus.

5.1.2.6 **mode**

`I2C_MODE` `_I2C_Device::mode`

Mode for I2C device (master, slave, etc.).

5.1.2.7 **port**

`I2C_Port` `_I2C_Device::port`

Structure tracking I2C port specific details.

The documentation for this struct was generated from the following file:

- [i2c.h](#)
-

5.2 `_I2C_Port` Struct Reference

Tracks I2C port settings (e.g. which module and clock speed for communication).

```
#include <i2c.h>
```

Collaboration diagram for `_I2C_Port`:

Public Attributes

- `I2C_CONFIGURATION` [config](#)
Configuration flags for port (e.g. stop in idle).
- `I2C_MODULE` [module](#)
I2C module as defined by core MCP library (I2C1, I2C2, ...).
- `uint32_t` [clkFreq](#)
Clock frequency to use when communicating with the bus (as a master).
- `I2C_ACK_MODE` [ackMode](#)
Mode to use when acknowledging received data (high vs low ack).

5.2.1 Detailed Description

Tracks I2C port settings (e.g. which module and clock speed for communication).

5.2.2 Member Data Documentation

5.2.2.1 `ackMode`

```
I2C\_ACK\_MODE _I2C_Port::ackMode
```

Mode to use when acknowledging received data (high vs low ack).

5.2.2.2 `clkFreq`

```
uint32_t _I2C_Port::clkFreq
```

Clock frequency to use when communicating with the bus (as a master).

5.2.2.3 `config`

```
I2C\_CONFIGURATION _I2C_Port::config
```

Configuration flags for port (e.g. stop in idle).

5.2.2.4 `module`

```
I2C\_MODULE _I2C_Port::module
```

I2C module as defined by core MCP library (I2C1, I2C2, ...).

The documentation for this struct was generated from the following file:

- [i2c.h](#)
-

6 File Documentation

6.1 Changelog.md File Reference

6.2 i2c.c File Reference

Implements functions used to interact with the I2C bus and peripheral devices.

```
#include <plib.h>
#include "i2c.h"
Include dependency graph for i2c.c:
```

6.3 i2c.h File Reference

Defines public constants, macros, and constant functions used to interact with an external I2C bus.

```
#include <plib.h>
#include <stdint.h>
Include dependency graph for i2c.h: This graph shows which files directly or indirectly include this file:
```

Classes

- struct [_I2C_Port](#)
Tracks I2C port settings (e.g. which module and clock speed for communication).
- struct [_I2C_Device](#)
Tracks the device-specific I2C information.

Typedefs

- typedef enum [_I2C_RC](#) [I2C_RC](#)
Enum of return code values.
 - typedef enum [_I2C_CLOCK_RATE](#) [I2C_CLOCK_RATE](#)
Enum of available clock rates supported by Wii devices.
 - typedef enum [_I2C_MODE](#) [I2C_MODE](#)
Enum of I2C bus modes.
 - typedef enum [_I2C_ACK_MODE](#) [I2C_ACK_MODE](#)
Acknowledgement methods used when receiving data from devices over I2C.
 - typedef enum [_I2C_ADDR_LEN](#) [I2C_ADDR_LEN](#)
Number of bits (length) of address format for target device.
 - typedef struct [_I2C_Port](#) [I2C_Port](#)
Tracks I2C port settings (e.g. which module and clock speed for communication).
 - typedef struct [_I2C_Device](#) [I2C_Device](#)
Tracks the device-specific I2C information.
-

Enumerations

- enum `_I2C_RC` {
`I2C_RC_SUCCESS` = 0, `I2C_RC_START_FAILED` = 1, `I2C_RC_RESTART_FAILED` = 2, `I2C_RC_SEND_BYTE_BUFFER_FAILED` = 3,
`I2C_RC_NO_ACK` = 4, `I2C_RC_RECEIVE_OVERFLOW` = 5 }
Enum of return code values.
- enum `_I2C_CLOCK_RATE` { `I2C_CLOCK_RATE_STANDARD` = 100000, `I2C_CLOCK_RATE_FAST` = 400000 }
Enum of available clock rates supported by Wii devices.
- enum `_I2C_MODE` { `I2C_MODE_MASTER` = 1, `I2C_MODE_SLAVE` = 2 }
Enum of I2C bus modes.
- enum `_I2C_ACK_MODE` { `I2C_ACK_MODE_NACK` = 0, `I2C_ACK_MODE_ACK` = 1 }
Acknowledgement methods used when receiving data from devices over I2C.
- enum `_I2C_ADDR_LEN` { `I2C_ADDR_LEN_7_BITS` = 7, `I2C_ADDR_LEN_10_BITS` = 10 }
Number of bits (length) of address format for target device.

Functions

- `I2C_RC I2C_InitPort` (`I2C_Port` *port, uint32_t pbClk, BOOL force)
Initializes the target I2C port.
- `I2C_RC I2C_Transmit` (`I2C_Device` *device, uint8_t *data, uint32_t len, BOOL ackRequired)
Transmit the identifier + requested data out over I2C.
- `I2C_RC I2C_Receive` (`I2C_Device` *device, uint8_t *data, uint32_t len, BOOL ackMessages)
Read data from the.
- `I2C_RC I2C_TxRx` (`I2C_Device` *device, uint8_t *dataTx, uint32_t lenTx, uint8_t *dataRx, uint32_t lenRx, BOOL ack, BOOL useRepeatedStart)
Handle a combined write + read process over I2C to a target device.

6.3.1 Detailed Description

Defines public constants, macros, and constant functions used to interact with an external I2C bus.

Note

For references to functions, macros, and constants provided by Microchip, please see the library file:

- [mc32-install]-libs\include\peripheral\i2c.h

6.3.2 Typedef Documentation

6.3.2.1 I2C_ACK_MODE

```
typedef enum _I2C_ACK_MODE I2C_ACK_MODE
```

Acknowledgement methods used when receiving data from devices over I2C.

6.3.2.2 I2C_ADDR_LEN

```
typedef enum _I2C_ADDR_LEN I2C_ADDR_LEN
```

Number of bits (length) of address format for target device.

6.3.2.3 I2C_CLOCK_RATE

```
typedef enum _I2C_CLOCK_RATE I2C_CLOCK_RATE
```

Enum of available clock rates supported by Wii devices.

Note

This is not a list of all I2C supported rates. Rather, this is a list of known clock speeds used by this library.

6.3.2.4 I2C_Device

```
typedef struct _I2C_Device I2C_Device
```

Tracks the device-specific I2C information.

Note

Also encapsulates the 'I2C_Port{}' information for easier tracking.

6.3.2.5 I2C_MODE

```
typedef enum _I2C_MODE I2C_MODE
```

Enum of I2C bus modes.

6.3.2.6 I2C_Port

```
typedef struct _I2C_Port I2C_Port
```

Tracks I2C port settings (e.g. which module and clock speed for communication).

6.3.2.7 I2C_RC

```
typedef enum _I2C_RC I2C_RC
```

Enum of return code values.

6.3 Enumeration Type Documentation

6.3.3.1 _I2C_ACK_MODE

```
enum _I2C_ACK_MODE
```

Acknowledgement methods used when receiving data from devices over I2C.

Enumerator

I2C_ACK_MODE_NACK	Acknowledge data received with a low-bit [0].
I2C_ACK_MODE_ACK	Acknowledge data received with a high-bit [1].

6.3.3.2 _I2C_ADDR_LEN

enum [_I2C_ADDR_LEN](#)

Number of bits (length) of address format for target device.

Enumerator

I2C_ADDR_LEN_7_BITS	Target device address is 7-bits.
I2C_ADDR_LEN_10_BITS	Target device address is 10-bits.

6.3.3.3 _I2C_CLOCK_RATE

enum [_I2C_CLOCK_RATE](#)

Enum of available clock rates supported by Wii devices.

Note

This is not a list of all I2C supported rates. Rather, this is a list of known clock speeds used by this library.

Enumerator

I2C_CLOCK_RATE_STANDARD	Standard clock rate used for (low speed) I2C communication over a I2C bus.
I2C_CLOCK_RATE_FAST	Fastest [supported] clock rate used for I2C communication over a I2C bus.

6.3.3.4 _I2C_MODE

enum [_I2C_MODE](#)

Enum of I2C bus modes.

Enumerator

I2C_MODE_MASTER	I2C device is a master controlling the communication.
I2C_MODE_SLAVE	I2C device is a slave controlling the communication.

6.3.3.5 I2C_RC

```
enum I2C_RC
```

Enum of return code values.

Enumerator

I2C_RC_SUCCESS	Successfully completed task(s).
I2C_RC_START_FAILED	Failed to set start condition (as I2C master).
I2C_RC_RESTART_FAILED	Failed to send restart message (as I2C master).
I2C_RC_SEND_BYTE_BUFFER_FAILED	Failed to buffer a byte for transmission over I2C bus.
I2C_RC_NO_ACK	Data transmitted over bus but no 1.
I2C_RC_RECEIVE_OVERFLOW	Unable to start receiving data due to buffer overflow.

6.3.4 Function Documentation

6.3.4.1 I2C_InitPort()

```
I2C_RC I2C_InitPort (
    I2C_Port * port,
    uint32_t pbClk,
    BOOL force )
```

Initializes the target I2C port.

Parameters

in	<i>*port</i>	Instance of 'I2C_Port{}' struct. Values used to define target to initialize and how it should be initialized.
in	<i>pbClk</i>	Current peripheral bus clock for device (referenced during I2C initialization).
in	<i>force</i>	Boolean flag indicating if port should be reinitialized if already initialized.

Returns

Return code corresponding to an entry in the 'I2C_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

6.3.4.2 I2C_Receive()

```
I2C_RC I2C_Receive (
    I2C_Device * device,
    uint8_t * data,
    uint32_t len,
    BOOL ackMessages )
```

Read data from the.

Handles bus arbitration (start and stop). Ensures data is ready to be read and supervise the process of invoking a byte-by-byte read of all necessary data.

Warning

Presumes caller has appropriately allocated space for copying read data into.

Note

The address for the I2C is defined in the 'device' parameter provided to the function and is transmitted prior to sending the provided 'len' of data (total data transmitted == 'len + 1').

Parameters

in	<i>*device</i>	Instance of 'I2C_Device{}' struct.
in	<i>*data</i>	Buffer of data to populate.
in	<i>len</i>	Amount of data to read (in bytes).
in	<i>ackRequired</i>	Flag indicating if an acknowledgement of data read should occur (the ack mode is defined in the 'device' instance).

Returns

Return code corresponding to an entry in the 'I2C_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function:

6.3.4.3 I2C_Transmit()

```
I2C_RC I2C_Transmit (
    I2C_Device * device,
    uint8_t * data,
    uint32_t len,
    BOOL ackRequired )
```

Transmit the identifier + requested data out over I2C.

Handles bus arbitration (start and stop) and prepends the address indicated within the provided 'device' structure's address field.

Note

The address for the I2C is defined in the 'device' parameter provided to the function and is transmitted prior to sending the provided 'len' of data (total data transmitted == 'len + 1').

Parameters

in	<i>*device</i>	Instance of 'I2C_Device{}' struct.
in	<i>*data</i>	Buffer of data to send out over bus.
in	<i>len</i>	Amount of data to transfer [excluding address of target].
in	<i>ackRequired</i>	Flag indicating if an acknowledgement of message transmission is required.

Returns

Return code corresponding to an entry in the 'I2C_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function:

6.3.4.4 I2C_TxRx()

```
I2C_RC I2C_TxRx (
    I2C_Device * device,
    uint8_t * dataTx,
    uint32_t lenTx,
    uint8_t * dataRx,
    uint32_t lenRx,
    BOOL ack,
    BOOL useRepeatedStart )
```

Handle a combined write + read process over I2C to a target device.

Handles bus arbitration (start and stop) and prepends the address indicated within the provided 'device' structure's address field. Ensures data is ready to be read and supervise the process of invoking a byte-by-byte read of all necessary data.

Warning

Presumes caller has appropriately allocated space for copying read data into.

Note

The address for the I2C is defined in the 'device' parameter provided to the function and is transmitted prior to sending the provided 'len' of data (total data transmitted == 'len + 1').

Parameters

in	<i>*device</i>	Instance of 'I2C_Device{ }' struct.
in	<i>*dataTx</i>	Buffer of data to send out over bus.
in	<i>lenTx</i>	Amount of data to transfer [excluding address of target].
in	<i>*dataRx</i>	Buffer of data to populate.
in	<i>lenRx</i>	Amount of data to read (in bytes).
in	<i>ackRequired</i>	Flag indicating if an acknowledgement of message should occur during tx and be expected during rx.
in	<i>useRepeatedStart</i>	Boolean flag controlling if send repeated start rather than releasing and re-acquiring the bus between tx and rx. TRUE == send repeatedstart condition; FALSE == send STOP and then a fresh START condition.

Returns

Return code corresponding to an entry in the 'I2C_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function:

6.4 i2c.o.d File Reference

6.5 i2c.o.d File Reference

6.6 i2c.o.d File Reference

6.7 i2c.o.d File Reference

6.8 README.md File Reference
