My Project

Generated by Doxygen 1.8.12

Contents

Chapter 1

RTC Library

A class for playing with a RTC module for Arduino, based on the chip MCP79410.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

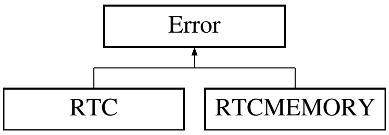
Class Documentation

5.1 Error Class Reference

Class for basic error management.

`#include <Error.h>`

Inheritance diagram for Error:



Public Member Functions

- Error (void)

    Basic constructor.
- Error (uint16_t e)

    Basic constructor.
- uint16_t getError (void)

    Returns the code of the last error occurred or `ERROR_NONE` for no errors.
- uint16_t setError (uint16_t e)

    Sets the code for the error just occurred (or `ERROR_NONE` for no errors for resetting error status).
- bool hasErrorOccurred (void)

    Returns `True` if an error has occurred, `False` otherwise.
- void clearError (void)

    reset the error to `ERROR_NONE`

5.1.1 Detailed Description

Class for basic error management.

This class implements very basic error management. All other component classes should inherit from this one.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Error::Error (uint16_t *e*) [inline]

Basic constructor.

Parameters

e	Code of the error occurred. See the macro definitions for error codes.

Remarks

Please use the macros for error codes in order to be compatible with future versions of this software.

5.1.3 Member Function Documentation

5.1.3.1 uint8_t Error::setError (uint16_t *e*) [inline]

Sets the code for the error just occurred (or `ERROR_NONE` for no errors for resetting error status).

Parameters

e	Code of error that has occurred.

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

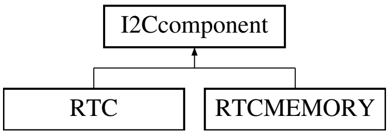
- Error.h

5.2 I2Ccomponent Class Reference

Basic class for dealing with components which use the I2C bus.

`#include <I2Ccomponent.h>`

Inheritance diagram for I2Ccomponent:



Public Member Functions

- I2Ccomponent (const uint8_t a)

  Constructor.
- uint8_t getAddress (void)

  Returns the address of this component on the I2C bus.

Protected Member Functions

- uint8_t readByte (const uint8_t adr)

  Reads a byte from the component using the I2C bus.
- void writeByte (const uint8_t adr, const uint8_t data)

  Writes (sends) a byte to the component via the I2C bus.

5.2.1 Detailed Description

Basic class for dealing with components which use the I2C bus.

Author

Enrico Formenti

5.2.2 Constructor & Destructor Documentation

5.2.2.1 I2Ccomponent::I2Ccomponent (const uint8_t *a*) `[inline]`

Constructor.

Parameters

a	Address of this component on the I2C bus.

Remarks

No check is made if there are conflicting addresses on the bus.

5.2.3 Member Function Documentation

5.2.3.1 uint8_t I2Ccomponent::readByte (const uint8_t *adr*) `[protected]`

Reads a byte from the component using the I2C bus.

Parameters

adr	Addresss to be read (ranging from 0x0 to 0xFF)

Remarks

No check is made here to establish if `adr` is a valid address for this component.

5.2.3.2 void I2Ccomponent::writeByte (const uint8_t *adr*, const uint8_t *data*) `[protected]`

Writes (sends) a byte to the component via the I2C bus.

Parameters

adr	Addresss to be written to (ranging from 0x0 to 0xFF)
data	Data to be written.

Remarks

No check is made here to establish if `adr` is a valid address for this component.

The documentation for this class was generated from the following files:

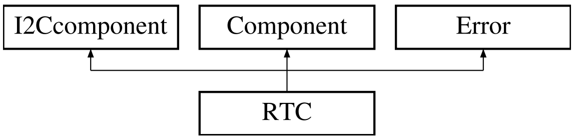
- I2Ccomponent.h
- I2Ccomponent.cpp

5.3 RTC Class Reference

A class for real time clock module based on MCP79410 chip.

`#include <RTC.h>`

Inheritance diagram for RTC:



Public Member Functions

- RTC (void)

  Default Constructor.
- RTC (boolean allowOverflow)

  extendend constructor for specifically allow or disallow the EEprom page overflow
- void setDate (const uint8_t target, const char ∗format,...)

  Sets the date for the main clock or for one of the alarms.
- void getDate (const uint8_t target, const char ∗format,...)

  Read the date from the main clock or from one of the alarms.
- void setTime (const uint8_t target, const char ∗format,...)

  Sets the time for the onboard clock or for one of the alarms.
- void getTime (const uint8_t target, const char ∗format,...)

  Reads the time from the main clock or from one of the alarms.
- boolean isAlarmTriggered (const uint8_t target)

  Returns true if the alarm for the given target has been triggered.
- void alarmFlagReset (const uint8_t target)

  Resets the alarm flag for the `target` alarm.
- void setAlarmMatch (const uint8_t target, const char ∗format,...)

  Sets the criteria used by the module for trigering the alarm `target`.
- const char getAlarmMatch (const uint8_t target)

  Gets the criteria used by the module for triggering the alarm `target`.
- boolean get1224Mode (const uint8_t target)

  Returns the display mode for the target clock or alarm.
- void set1224Mode (const uint8_t target, const boolean mode)

  Sets the clock display mode for the given target clock or alarm.

- boolean isLeapYear (void)

  Returns if it is a leap year or not.
- void setAlarmLevel (const uint8_t target, const uint8_t lvl)

  Sets the TTL level for MFP pin when the `target` alarm is triggered.
- uint8_t getAlarmLevel (const uint8_t target)

  Returns the TTL level of the MFP pin when the `target` alarm is triggered.
- void batterySupply (const boolean enable)

  Enables/Disables the external battery supply when main power fails.
- void configureAlarmMode (const char format)

  configures what alarm (ALM0, ALM1, none, both) is active
- boolean isAlarmActive (const uint8_t target)

  returns whether the selected alarm is active.
- const char getAlarmMode (void)

  gets which alarm are active
- void getConfBits (void)

  prints values of some meaningful registers
- void printConfBit (const uint8_t reg)

  prints on serial port the conf bit relative to the register
- uint8_t getTrimmingValue (void)

  returns the contents of the oscillator trimming register
- void setTrimming (uint8_t trimval)

  sets the trimming register value, with bit 7 as the sign
- void setSquareWaveOutput (uint8_t freqval)

  configure the multifunction pin to output a certain frequency
- void clearSquareWaveOutput (void)

  disables the square wave line output
- const char getStatusRegister (void)

  gets the status of mem protection
- void writeArrayToEEprom (const uint8_t addr, uint8_t ∗data, uint8_t length)

  Facade for inserting an array of data to the EEPROM, using writeBytesToMemory.
- void writeByteToEEprom (const uint8_t addr, uint8_t data)

  write a single byte onto the RTC eeprom. It internally calls writeArrayToEEprom
- void readArrayFromEEprom (const uint8_t addr, uint8_t ∗data, uint8_t length)

  reads a sequence of maximum 8 bytes from the RTC eeprom using readBytesFromMemory
- uint8_t readByteFromEEprom (const uint8_t addr)

  reads a single byte from the RTC eeprom using readSingleByteFromMemory
- void writeArrayToSRAM (const uint8_t addr, uint8_t ∗data, uint8_t length)

  Facade for inserting an array of data to the SRAM.
- void writeByteToSRAM (const uint8_t addr, uint8_t data)

  writes a single byte to SRAM memory
- void readArrayFromSRAM (const uint8_t addr, uint8_t ∗data, uint8_t length)

  facade for reading a set of bytes from the SRAM memory
- uint8_t readByteFromSRAM (const uint8_t addr)

  reads a single byte from the SRAM memory

Static Public Attributes

- static const uint8_t RTC_MAIN = 0x0

  Address of the main clock.
- static const uint8_t RTC_ALM0 =0x0A

  Address of alarm 0.
- static const uint8_t RTC_ALM1 =0x11

  Address of alarm 1.

Additional Inherited Members

5.3.1 Detailed Description

A class for real time clock module based on MCP79410 chip.

This class provides an interface for the component, defining a driver for almost all of the functionalities described in the datasheet. The Error class is used to set errors on the error buffer, and the I2Ccomponent class contains the basic I2C communication specifications.

A specific class has been created to manage the memory (EEPROM and SRAM) communications, and is incapsulated as a strategy class.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 RTC::RTC (boolean *allowOverflow*) `[inline]`

extendend constructor for specifically allow or disallow the EEprom page overflow

Parameters

allowOverflow	boolean `True` when the page overflow is allowed

5.3.3 Member Function Documentation

5.3.3.1 void RTC::alarmFlagReset (const uint8_t *target*)

Resets the alarm flag for the `target` alarm.

Parameters

target	can take one of the two vales: `RTC_ALM0`, `RTC_ALM1` for alarm 0 and alarm 1, respectively.

Remarks

> The flag 'alarm triggered' will stay on untill reset from software. Use the function `alarmFlagReset` to reset it and start waiting for another alarm event.

See also

> isAlarmTriggered, setAlarmMatch

5.3.3.2 void RTC::batterySupply (const boolean *enable*)

Enables/Disables the external battery supply when main power fails.

Parameters

enable	setting this variable to true enables the external battery supply. Set it to false for disabling it.

5.3.3.3 void RTC::configureAlarmMode (const char *format*)

configures what alarm (ALM0, ALM1, none, both) is active

Parameters

format	character indicating the match criteria. Here are the admissible values: • `0` : sets only `RTC_ALM0` as active • `1` : sets only `RTC_ALM1` as active • `b` : sets both `RTC_ALM0` and `RTC_ALM1` as active • `n` : disables all alarms

See also

getAlarmMode, isAlarmActive

5.3.3.4 boolean RTC::get1224Mode (const uint8_t *target*) [inline]

Returns the display mode for the target clock or alarm.

Parameters

target	can take one of the three vales: `RTC_MAIN`, `RTC_ALM0`, `RTC_ALM1` for main clock, alarm 0 and alarm 1, respectively.

Returns

`True` if the target clock/alarm is in 12 hours display mode, `False` for 24 hours mode.

See also

set1224Mode

5.3.3.5 uint8_t RTC::getAlarmLevel (const uint8_t *target*) [inline]

Returns the TTL level of the MFP pin when the `target` alarm is triggered.

Parameters

target	can take one of the two vales: `RTC_ALM0`, `RTC_ALM1` for alarm 0 and alarm 1, respectively.

See also

setAlarmLevel

5.3.3.6 void RTC::getAlarmMatch (const uint8_t *target*)

Gets the criteria used by the module for triggering the alarm `target`.

Parameters

target	can take one of the two vales: `RTC_ALM0`, `RTC_ALM1` for alarm 0 and alarm 1, respectively.

Returns

a character indicating the match criteria. Here are the possible values:

- `s` : seconds
- `m` : minutes
- `h` : hours
- `d` : day (alarm triggered at 12:00:00 AM)
- `x` : date
- `a` : matches seconds, minutes, hours, day, date, month.

See also

setAlarmMatrch, isAlarmTriggered, alarmFlagReset

5.3.3.7 char RTC::getAlarmMode (void)

gets which alarm are active

Returns

a character indicating which alarm mode is active; here are the possible values:

- `0` : only `RTC_ALM0` is active
- `1` : only `RTC_ALM1` is active
- `n` : no alarms are active
- `b` : both `RTC_ALM0` and `RTC_ALM1` are active

5.3.3.8 void RTC::getConfBits (void)

prints values of some meaningful registers

Warning

> since this reads some meaningful registers "unprotected", while not stopping the clock, it may be best not to use it or to stop and start the clock

5.3.3.9 void RTC::getDate (const uint8_t *target*, const char ∗ *format*, ...)

Read the date from the main clock or from one of the alarms.

Parameters

target	can take one of the three vales: RTC_MAIN, RTC_ALM0, RTC_ALM1 for main clock, alarm 0 and alarm 1, respectively.
format	chain of characters indicating the variables to set, similarly to classical printf function of C language. Here are the possibles characters: • `d` : number indicating the day of the week, 1 = monday, 2 = tuedsay, etc. • `D` : same as `d` • `n` : number of the day (ranging from 1 to 31) • `N` : same as `n` • `m` : number indicating the month, 1 = january, 2 = febrary, etc. • `M` : same as `m` • `y` : year (ranging from 0 to 99) • `Y` : same as `y`

Remarks

> Parameters are processed according to the order of appearence in `format`.

See also

> setDate, setTime, getTime

5.3.3.10 char RTC::getStatusRegister (void) `[inline]`

gets the status of mem protection

Returns

> a char indicating what part of memory is protected:
>
> - `0` means none
> - `q` means the upper quarter is protected
> - `h` means the upper half is protected
> - `a` means all of the eeprom is protected

5.3.3.11 void RTC::getTime (const uint8_t *target,* const char ∗ *format,* *...*)

Reads the time from the main clock or from one of the alarms.

Parameters

target	can take one of the three vales: RTC_MAIN, RTC_ALM0, RTC_ALM1 for main clock, alarm 0 and alarm 1, respectively.
format	chain of characters indicating the variables to set, similarly to classical printf function of C language. Here are the possibles characters: • `s` : seconds (ranging from 0 to 59) • `S` : same as `s` • `m` : minutes (ranging from 0 to 59) • `M` : same as `m` • `h` : hour (ranging from 0 to 24 or from 0 to 12 according to the 12/24 display format) • `H` : same as `h`

Remarks

Parameters are processed according to the order of appearence in `format`.

See also

set1224Mode, getTime, setTime, getDate, setDate

5.3.3.12 uint8_t RTC::getTrimmingValue (void) `[inline]`

returns the contents of the oscillator trimming register

Returns

the value of the trimming register

See also

setTrimmingValueUnsigned, setTrimmingValueSigned

5.3.3.13 boolean RTC::isAlarmActive (const uint8_t *target*)

returns whether the selected alarm is active.

Parameters

target	can take one of the two values: `RTC_ALM0`, `RTC_ALM1` for alarm 0 and alarm 1, respectively.

See also

getAlarmMode, configureAlarmMode

5.3.3.14 boolean RTC::isAlarmTriggered (const uint8_t *target*) [inline]

Returns true if the alarm for the given target has been triggered.

Parameters

target	can take one of the two vales: `RTC_ALM0`, `RTC_ALM1` for alarm 0 and alarm 1, respectively.

Remarks

The flag 'alarm triggered' will stay on untill reset from software. Use the function `alarmFlagReset` to reset it and start waiting for another alarm event.

See also

alarmFlagReset, setTime, setDate, setAlarmMatch

5.3.3.15 boolean RTC::isLeapYear (void)

Returns if it is a leap year or not.

Returns

`True` if the year set in main clock is a leap year, `False` otherwise.

5.3.3.16 void RTC::printConfBit (const uint8_t *reg*)

prints on serial port the conf bit relative to the register

Parameters

reg	is the registry value

Warning

since this can read some meaningful registers "unprotected", while not stopping the clock, it may be best not to use it or to stop and start the clock

5.3.3.17 void RTC::readArrayFromEEprom (const uint8_t *addr,* uint8_t ∗ *data,* uint8_t *length*) [inline]

reads a sequence of maximum 8 bytes from the RTC eeprom using readBytesFromMemory

Parameters

addr	is the memory start address. It should be between 0x00 and 0x7F, otherwise the counter will overflow
data	is an array in which the data will be stored
length	is the length of the data to read

See also

readBytesFromMemory

5.3.3.18 void RTC::readArrayFromSRAM (const uint8_t *addr,* uint8_t ∗ *data,* uint8_t *length*) [inline]

facade for reading a set of bytes from the SRAM memory

Parameters

addr	the memory starting address
data	the array on which the data will be written
length	the number of bytes to read

See also

readBytesFromMemory

5.3.3.19 uint8_t RTC::readByteFromEEprom (const uint8_t *addr*) [inline]

reads a single byte from the RTC eeprom using readSingleByteFromMemory

Parameters

addr	is the memory start address. It should be between 0x00 and 0x7F, otherwise the counter will overflow

Returns

the value stored in the memory

See also

readSingleByteFromMemory, readBytesFromMemory

5.3.3.20 uint8_t RTC::readByteFromSRAM (const uint8_t *addr*) [inline]

reads a single byte from the SRAM memory

Parameters

addr	the address from which to read

Returns

the value stored in the SRAM

See also

readBytesFromMemory

5.3.3.21 void RTC::set1224Mode (const uint8_t *target,* const boolean *mode*)

Sets the clock display mode for the given target clock or alarm.

Parameters

target	can take one of the three vales: `RTC_MAIN`, `RTC_ALM0`, `RTC_ALM1` for main clock, alarm 0 and alarm 1, respectively.

Warning

only `RTC_MAIN` register is writeable, while the other are a copy of it and readonly.

Parameters

mode	true for 12 hours mode, false for 24 hours display mode.

See also

get1224Mode

5.3.3.22 void RTC::setAlarmLevel (const uint8_t *target,* const uint8_t *lvl*)

Sets the TTL level for MFP pin when the `target` alarm is triggered.

Parameters

target	can take one of the two vales: `RTC_ALM0`, `RTC_ALM1` for alarm 0 and alarm 1, respectively.
lvl	can take two values: `HIGH` or `LOW`.

See also

getAlarmLevel

5.3.3.23 void RTC::setAlarmMatch (const uint8_t *target,* const char ∗ *format,* ...)

Sets the criteria used by the module for trigering the alarm `target`.

Parameters

target	can take one of the two vales: `RTC_ALM0`, `RTC_ALM1` for alarm 0 and alarm 1, respectively.
format	character indicating the match criteria. Here are the admissible values: • `s` : seconds • `S` : same as `s` • `m` : minutes • `M` : same as `m` • `h` : hours • `H` : same as `h` • `d` : day (alarm triggered at 12:00:00 AM) • `D` : same as `d` • `x` : date • `X` : same as x • `a` : matches seconds, minutes, hours, day, date, month. • `A` : same as `a`

Remarks

Parameters are processed according to the order of appearence in `format`.

See also

isAlarmTriggered, alarmFlagReset

5.3.3.24 void RTC::setDate (const uint8_t *target,* const char ∗ *format,* ...)

Sets the date for the main clock or for one of the alarms.

Parameters

target	can take one of the three vales: RTC_MAIN, RTC_ALM0, RTC_ALM1 for main clock, alarm 0 and alarm 1, respectively.

Parameters

format	chain of characters indicating the variables to set, similarly to classical printf function of C language. Here are the possibles characters: • `d` : number indicating the day of the week, 1 = monday, 2 = tuedsay, etc. • `D` : same as `d` • `n` : number of the day (ranging from 1 to 31) • `N` : same as `n` • `m` : number indicating the month, 1 = january, 2 = febrary, etc. • `M` : same as `m` • `y` : year (ranging from 0 to 99) • `Y` : same as `y`

Remarks

Parameters are processed according to the order of appearence in `format`.

See also

getDate, setTime, getTime

5.3.3.25 void RTC::setSquareWaveOutput (uint8_t *freqval*)

configure the multifunction pin to output a certain frequency

Parameters

freqval	can assume four values • `0` indicates a 32.768 kHz freq • `1` indicates a 8.192 kHz freq • `2` indicates a 4.096 kHz freq • `3` indicates a 1 Hz freq

5.3.3.26 void RTC::setTime (const uint8_t *target,* const char $*$ *format,* ...)

Sets the time for the onboard clock or for one of the alarms.

Parameters

target	can take one of the three vales: RTC_MAIN, RTC_ALM0, RTC_ALM1 for main clock, alarm 0 and alarm 1, respectively.
format	chain of characters indicating the variables to set, similarly to classical printf function of C language. Here are the possibles characters: • `s` : seconds (ranging from 0 to 59) • `S` : same as `s` • `m` : minutes (ranging from 0 to 59) • `M` : same as `m` • `h` : hour (ranging from 0 to 24 or from 0 to 12 according to the 12/24 display format) • `H` : same as `h`

Remarks

Parameters are processed according to the order of appearence in `format`.

See also

set1224Mode, getTime, setDate, getDate

5.3.3.27 void RTC::setTrimming (uint8_t *trimval*)

sets the trimming register value, with bit 7 as the sign

Parameters

trimval	represents the value to put in the register

See also

getTrimmingValue

5.3.3.28 void RTC::writeArrayToEEprom (const uint8_t *addr,* uint8_t ∗ *data,* uint8_t *length*) `[inline]`

Facade for inserting an array of data to the EEPROM, using writeBytesToMemory.

Parameters

addr	is the memory address. It should be between 0x00 and 0x7F, otherwise the counter will overflow
data	is an array of bytes to write onto the memory
length	is the length of the data to write, in general different from the array length

See also

writeBytesToMemory

5.3.3.29 void RTC::writeArrayToSRAM (const uint8_t *addr*, uint8_t ∗ *data*, uint8_t *length*) [inline]

Facade for inserting an array of data to the SRAM.

Parameters

addr	is the memory address. It must be between 0x20 and 0x5F. The counter won't overflow
data	is an array of data to write into the SRAM
length	is the number of bytes to write

See also

writeBytesToMemory

5.3.3.30 void RTC::writeByteToEEprom (const uint8_t *addr*, uint8_t *data*) [inline]

write a single byte onto the RTC eeprom. It internally calls writeArrayToEEprom

Parameters

addr	is the memory address. It should be between 0x00 and 0x7F, otherwise the counter will overflow
data	is the data to write

See also

writeArrayToEEprom, writeBytesToMemory

5.3.3.31 void RTC::writeByteToSRAM (const uint8_t *addr*, uint8_t *data*) [inline]

writes a single byte to SRAM memory

Parameters

addr	is the memory address
data	is the byte to write

See also

writeByteToSRAM, writeBytesToMemory

5.3.4 Member Data Documentation

5.3.4.1 static const uint8_t RTC::RTC_ALM0 =0x0A [static]

Address of alarm 0.

Warning

Use the variable in your programs. Direct use of the value `0x0A` might result incompatible with future versions.

5.3.4.2 static const uint8_t RTC::RTC_ALM1 =0x11 [static]

Address of alarm 1.

Warning

Use the variable in your programs. Direct use of the value `0x11` might result incompatible with future versions.

5.3.4.3 static const uint8_t RTC::RTC_MAIN = 0x0 [static]

Address of the main clock.

Warning

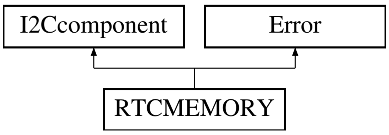
Use the variable in your programs. Direct use of the value `0x0` might result incompatible with future versions.

The documentation for this class was generated from the following files:

- RTC.h
- RTC.cpp

5.4 RTCMEMORY Class Reference

Inheritance diagram for RTCMEMORY:



Public Member Functions

- const char getStatus (void)

  gets the status of eeprom protection
- void writeEEpromBytes (const uint8_t addr, uint8_t ∗data, uint8_t length)

  write a sequence of maximum 8 bytes onto the RTC eeprom using Wire
- void writeEEpromBytesNoOF (const uint8_t addr, uint8_t ∗data, uint8_t length)

  write a sequence of maximum 8 bytes onto the RTC eeprom using Wire. The method only writes if the length is less than or equal to the number of bytes from the start address and the end of page.
- void readEEpromBytes (const uint8_t addr, uint8_t ∗data, uint8_t length)

  reads a sequence of maximum 8 bytes from the RTC eeprom using Wire
- void writeSRAMBytes (const uint8_t addr, uint8_t ∗data, uint8_t length)

  writes on the SRAM
- void readSRAMBytes (const uint8_t addr, uint8_t ∗data, uint8_t length)

  reads bytes from SRAM

Additional Inherited Members

5.4.1 Member Function Documentation

5.4.1.1 char RTCMEMORY::getStatus (void)

gets the status of eeprom protection

Returns

a char indicating what part of memory is protected:

- `0` means none
- `q` means the upper quarter is protected
- `h` means the upper half is protected
- `a` means all of the eeprom is protected

5.4.1.2 void RTCMEMORY::readEEpromBytes (const uint8_t *addr,* uint8_t ∗ *data,* uint8_t *length*)

reads a sequence of maximum 8 bytes from the RTC eeprom using Wire

Parameters

addr	is the memory start address. It should be between 0x00 and 0x7F, otherwise the counter will overflow
data	is an array in which the data will be stored
length	is the length of the data to read

5.4.1.3 RTCMEMORY::readSRAMBytes (const uint8_t *addr,* uint8_t ∗ *data,* uint8_t *length*)

reads bytes from SRAM

Parameters

addr	is the starting address, which must be included between `RTC_SRAM_START` and `RTC_SRAM_END`
data	is the array on which the data is stored
length	is the number of bytes to read

5.4.1.4 void RTCMEMORY::writeEEpromBytes (const uint8_t *addr,* uint8_t ∗ *data,* uint8_t *length*)

write a sequence of maximum 8 bytes onto the RTC eeprom using Wire

Parameters

addr	is the memory address. It should be between 0x00 and 0x7F, otherwise the counter will overflow
data	is an array of bytes to write onto the memory
length	is the length of the data to write, in general different from the array length

Warning

> the memory is paged by 8 bytes. Every write operation exceeding the page length will result in overflow, overwriting effectively the previous bytes of the page. No overflow checks are performed.

5.4.1.5 RTCMEMORY::writeEEpromBytesNoOF (const uint8_t *addr,* uint8_t ∗ *data,* uint8_t *length*)

write a sequence of maximum 8 bytes onto the RTC eeprom using Wire. The method only writes if the length is less than or equal to the number of bytes from the start address and the end of page.

Parameters

addr	is the memory address. It should be between 0x00 and 0x7F, otherwise the counter will overflow
data	is an array of bytes to write onto the memory
length	is the length of the data to write, in general different from the array length

Remarks

> the memory is paged by 8 bytes.

5.4.1.6 void RTCMEMORY::writeSRAMBytes (const uint8_t *addr,* uint8_t ∗ *data,* uint8_t *length*)

writes on the SRAM

Parameters

addr	the starting address, which must be between `RTC_SRAM_START` and `RTC_SRAM_END`
data	the array to write
length	the number of bytes to write

The documentation for this class was generated from the following files:

- RTCMEMORY.h
- RTCMEMORY.cpp

Chapter 6

File Documentation

6.1 Error.h File Reference

Class definition for error management.

```
#include "WProgram.h"
```

Classes

- class Error

  Class for basic error management.

Macros

Error codes

Macro definitions for the error codes. All possible errors are collected here for compactness and reuse sake.

Remarks

The codes are defined as 4 bytes sequences and are randomly generated.

- #define **ERROR_NONE** 0x0
- #define **ERROR_INVALID_CRC** 0xb9e5
- #define **ERROR_NO_MORE_ADDRESSES** 0x52aa
- #define **ERROR_TIME_OUT** 0xab70
- #define **ERROR_READ_FAILURE** 0xca07
- #define **ERROR_WRITE_FAILURE** 0xc807
- #define **ERROR_OUT_OF_RANGE** 0x5437
- #define **ERROR_INVALID_DATA** 0xae92d210
- #define **ERROR_INVALID_FORMAT** 0xa67ea6fe

6.1.1 Detailed Description

Class definition for error management.

This class implements very basic error management common to all components.

Author

Enrico Formenti
Daniele Ratti

Version

0.5

Date

2012-2013; 2016

Warning

This software is provided "as is". The author is not responsible for any damage of any kind caused by this software. Use it at your own risk.

Copyright

BSD license. See license.txt for more details. All text above must be included in any redistribution.

6.2 I2Ccomponent.cpp File Reference

Implementation of the I2Ccomponent class.

```
#include "I2Ccomponent.h"
```

6.2.1 Detailed Description

Implementation of the I2Ccomponent class.

Author

Enrico Formenti

Version

0.1

Date

2012-2013

Warning

This software is provided "as is". The author is not responsible for any damage of any kind caused by this software. Use it at your own risk.

Copyright

BSD license. See license.txt for more details. All text above must be included in any redistribution.

6.3 I2Ccomponent.h File Reference

Definition of a class for dealing with basic I2C communication.

```
#include "WProgram.h"
#include "Wire.h"
```

Classes

- class I2Ccomponent

  Basic class for dealing with components which use the I2C bus.

6.3.1 Detailed Description

Definition of a class for dealing with basic I2C communication.

Header file containing the definition of the I2CComponent class.

Author

Enrico Formenti

Version

0.1

Date

2012-2013

Warning

This software is provided "as is". The author is not responsible for any damage of any kind caused by this software. Use it at your own risk.

Copyright

6.4 RTC.cpp File Reference

Implementation of the RTC class.

```
#include "RTC.h"
```

6.4.1 Detailed Description

Implementation of the RTC class.

Author

Enrico Formenti
Daniele Ratti

Version

1.5

Date

2012-2013, 2016

Warning

This software is provided "as is". The author is not responsible for any damage of any kind caused by this software. Use it at your own risk.

Copyright

BSD license. See license.txt for more details. All text above must be included in any redistribution.

6.5 RTC.h File Reference

Definition of the RTC class.

```
#include "WProgram.h"
#include "RTCMEMORY.h"
#include "I2Ccomponent.h"
#include "Component.h"
#include "Error.h"
```

Classes

- class RTC

  A class for real time clock module based on MCP79410 chip.

Macros

- #define **RTC_SECOND** 0x0
- #define **RTC_MINUTE** 0x01
- #define **RTC_HOUR** 0x02
- #define **RTC_DAY** 0x03
- #define **RTC_DATE** 0x04
- #define **RTC_MONTH** 0x05
- #define **RTC_YEAR** 0x06
- #define **RTC_OSCTRIM** 0x08
- #define **RTC_ALM_I_FLAG** 0x08
- #define **RTC_1224_FLAG** 0x40
- #define **RTC_ALM_LVL_FLAG** 0x80
- #define **RTC_ALM_CFG** 0x03
- #define **RTC_CONFIGURATION_BYTE** 0x07
- #define **RTC_ALM0_CONFIGURATION_BYTE** 0x0D
- #define **RTC_ALM1_CONFIGURATION_BYTE** 0x14

6.5.1 Detailed Description

Definition of the RTC class.

Header file containing the definition of the RTC class.

Author

Enrico Formenti
Daniele Ratti

Version

1.5

Date

2012-2016

Warning

This software is provided "as is". The author is not responsible for any damage of any kind caused by this software. Use it at your own risk.

Copyright

BSD license. See license.txt for more details. All text above must be included in any redistribution.

6.6 RTCMEMORY.cpp File Reference

Implementation of the RTCMEMORY class.

```
#include "RTCMEMORY.h"
```

6.6.1 Detailed Description

Implementation of the RTCMEMORY class.

Author

Daniele Ratti

Version

1.0

Date

2015-2016

Warning

This software is provided "as is". The author is not responsible for any damage of any kind caused by this software. Use it at your own risk.

Copyright

BSD license. See license.txt for more details. All text above must be included in any redistribution.

6.7 RTCMEMORY.h File Reference

Definition of a class for dealing with MCP7921X EEPROM and SRAM.

```
#include "WProgram.h"
#include "Wire.h"
#include "I2Ccomponent.h"
#include "Error.h"
```

Classes

- class RTCMEMORY

Macros

- #define **RTC_STATUS** 0xFF
- #define **ADDRESS_EE** 0x57
- #define **ADDRESS_SR** 0x6F
- #define **BUFFER_EE** 8

6.7.1 Detailed Description

Definition of a class for dealing with MCP7921X EEPROM and SRAM.

Header file containing the definition of the RTCMEMORY class.

Author

Daniele Ratti

Version

1.0

Date

2015-2016

Warning

This software is provided "as is". The author is not responsible for any damage of any kind caused by this software. Use it at your own risk.

Copyright

Index