I2cDiscreteIoExpander

v2.0.0

Generated by Doxygen 1.8.2

Wed Jan 2 2013 13:39:59

Contents

1 Arduino library for TI PCF8575C 16-bit I2C I/O expander.

The PCF8575C provides general-purpose remote I/O expansion for most microcontroller families via the I2C interface serial clock (SCL) and serial data (SDA).

The device features a 16-bit quasi-bidirectional input/output (I/O) port (P07..P00, P17..P10), including latched outputs with high-current drive capability for directly driving LEDs. Each quasi-bidirectional I/O can be used as an input or output without the use of a data-direction control signal. At power on, the I/Os are in 3-state mode. The strong pullup to VCC allows fast-rising edges into heavily loaded outputs. This device turns on when an output is written high and is switched off by the negative edge of SCL. The I/Os should be high before being used as inputs. After power on, as all the I/Os are set to 3-state, all of them can be used as inputs. Any change in setting of the I/Os as either inputs or outputs can be done with the write mode. If a high is applied externally to an I/O that has been written earlier to low, a large current (IOL) flows to GND.

The fixed I2C address of the PCF8575C is the same as the PCF8575, PCF8574, PCA9535, and PCA9555, allowing up to eight of these devices, in any combination, to share the same I2C bus or SMBus.

Author

Doc Walker

Version

`2.0.0`

Date

1 Jan 2013

Copyright

GNU General Public License v3

Arduino IDE:

Requires Arduino 1.0 or later

Source Code Repository:

`https://github.com/4-20ma/I2cDiscreteIoExpander`

Programming Style Guidelines:

`http://geosoft.no/development/cppstyle.html`

Schematic:

```
                                        Arduino
                                       .--------------------.
                                       |Duemilanove         |
                                       |                    |
         TI PCF8575C        .---------o|5V                  |
        .-------_-------.   |          |                    |
      -o|1 /INT   VCC 24|o--'   o-----o|GND                 |
        |               |       |      |                    |
.------o|2  A1    SDA 23|o------)-----o|A4 SDA              |
|       |               |       |      |                    |
o------o|3  A2    SCL 22|o------)-----o|A5 SCL              |
|       |               |       |      '--------------------'
|     -o|4  P00    A0 21|o------o
|       |               |       |
|     -o|5  P01   P17 20|o-     |
|       |               |       |
|     -o|6  P02   P16 19|o-     |
|       |               |       |
|     -o|7  P03   P15 18|o-     |
|       |               |       |
|     -o|8  P04   P14 17|o-     |
|       |               |       |
|     -o|9  P05   P13 16|o-     |
|       |               |       |
|     -o|10 P06   P12 15|o-     |
|       |               |       |
|     -o|11 P07   P11 14|o-     |
|       |               |       |
o------o|12 GND   P10 13|o-     |
|       '---------------'       |
|                               |
'-------------------------------o
                                |
                                |
                               ===
                               GND
```

2 Optional Functions List (Troubleshooting)

Member I2cDiscreteIoExpander::getAddress ()

This function is for testing and troubleshooting.

3 Required Functions List

Member I2cDiscreteIoExpander::digitalRead ()

Call this from within `loop()` in order to read from device.

Member I2cDiscreteIoExpander::digitalWrite (uint16_t)

Call this from within `loop()` in order to write to device.

Member I2cDiscreteIoExpander::getPorts ()

Call this from within `loop()` to retrieve ports.

Member I2cDiscreteIoExpander::I2cDiscreteIoExpander (uint8_t)

Call this to construct I2cDiscreteIoExpander object.

4 Class Index

4.1 Class List

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

5 Class Documentation

5.1 I2cDiscreteIoExpander Class Reference

Public Member Functions

- I2cDiscreteIoExpander (uint8_t)

  Constructor.
- uint8_t digitalRead ()

  Retrieve discrete values from device.
- uint8_t digitalWrite (uint16_t)

  Write discrete values to device.
- uint8_t getAddress ()

  Retrieve device address.
- uint16_t getPorts ()

  Retrieve ports 1 (P17..P10), 0 (P07..P00).
- void enableBitwiseInversion ()

  Enable bitwise inversion.
- void disableBitwiseInversion ()

  Disable bitwise inversion.
- bool isInverted ()

  Indicate whether bitwise inversion is enabled.

Private Attributes

- uint8_t address_

  Device address as defined by pins A2, A1, A0.
- uint16_t ports_

  Storage object for I2cDiscreteIoExpander ports 1 (P17..P10), 0 (P07..P00).
- bool shouldInvert_

  Flag indicating whether bits are to be inverted before read/write (false=don't invert, true=invert).

Static Private Attributes

- static const uint8_t BASE_ADDRESS_ = 0x20

  Factory pre-set slave address.

Related Functions

(Note that these are not member functions.)

- static const uint8_t TWI_SUCCESS = 0

  I2C/TWI success (transaction was successful).
- static const uint8_t TWI_DEVICE_NACK = 2

  I2C/TWI device not present (address sent, NACK received).
- static const uint8_t TWI_DATA_NACK = 3

  I2C/TWI data not received (data sent, NACK received).
- static const uint8_t TWI_ERROR = 4

  I2C/TWI other error.

5.1.1 Detailed Description

Examples:

examples/BareMinimum/BareMinimum.ino, and examples/MultipleDevices/MultipleDevices.ino.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 I2cDiscreteIoExpander::I2cDiscreteIoExpander (uint8_t *address*)

Constructor.

Assigns device address, resets storage object, enables bitwise inversion.

Required Function Call this to construct I2cDiscreteIoExpander object.

Usage:

```
...
I2cDiscreteIoExpander exampleA(1);              // device with address 1
I2cDiscreteIoExpander exampleB[2] = { 2, 3 };   // devices
      with addresses 2, 3
I2cDiscreteIoExpander exampleC[2] = {
     I2cDiscreteIoExpander(4), I2cDiscreteIoExpander(5) }; //
      alternate constructor syntax; devices with addresses 4, 5
I2cDiscreteIoExpander exampleD[8] = { 0, 1, 2, 3, 4, 5, 6,
      7 }; // addresses 0..7
...
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

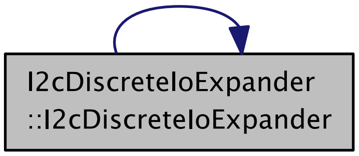
Assigns device address, resets storage object, enables bitwise inversion.

Usage:

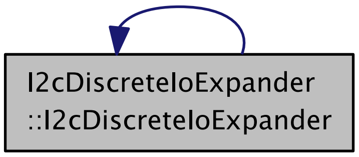
```
    ...
    I2cDiscreteIoExpander device;         // implies device address 0
    ...
```

```
{
  address_ = address & 0b111;
  ports_ = 0;
  shouldInvert_ = true;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3 Member Function Documentation

5.1.3.1 uint8_t I2cDiscreteIoExpander::digitalRead ( )

Retrieve discrete values from device.

Required Function Call this from within `loop()` in order to read from device.

Return values

0	success
1	length too long for buffer
2	address send, NACK received **(device not on bus)**
3	data send, NACK received
4	other twi error (lost bus arbitration, bus error, ...)

Usage:

```
...
I2cDiscreteIoExpander device;
...
uint8_t status = device.digitalRead();
if (TWI_SUCCESS == status)
{
  // do something with device.getPorts()
}
...
```

Examples:

examples/BareMinimum/BareMinimum.ino, and examples/MultipleDevices/MultipleDevices.ino.

```
{
  uint8_t hi, lo, status;

  Wire.beginTransmission(BASE_ADDRESS_ | address_);
  status = Wire.endTransmission();

  if (TWI_SUCCESS == status)
  {
    if (Wire.requestFrom(BASE_ADDRESS_ | address_, 2) == 2
      )
    {
      lo = Wire.read();
      hi = Wire.read();
      ports_ = shouldInvert_ ? word(~hi, ~lo) : word(hi, lo)
      ;
    }
    else
    {
      return TWI_ERROR;
    }
  }

  return status;
}
```

5.1.3.2 uint8_t I2cDiscreteIoExpander::digitalWrite (uint16_t *ports*)

Write discrete values to device.

Required Function Call this from within `loop()` in order to write to device.

Parameters

ports	word to be written to device (0x0000..0xFFFF)

Return values

0	success
1	length too long for buffer
2	address send, NACK received **(device not on bus)**
3	data send, NACK received

4	other twi error (lost bus arbitration, bus error, ...)

Usage:

```
...
I2cDiscreteIoExpander device;
...
uint8_t status = device.digitalWrite(0xFFFF);
if (TWI_SUCCESS == status)
{
  // do something
}
...
```

Examples:

examples/BareMinimum/BareMinimum.ino, and examples/MultipleDevices/MultipleDevices.ino.

```
{
  ports_ = shouldInvert_ ? ~ports : ports;
  Wire.beginTransmission(BASE_ADDRESS_ | address_);
  Wire.write(lowByte(ports_));
  Wire.write(highByte(ports_));
  // Wire.write(lowByte(shouldInvert_ ? ~ports_ : ports_));
  // Wire.write(highByte(shouldInvert_ ? ~ports_ : ports_));

  return Wire.endTransmission();
}
```

5.1.3.3 uint8_t I2cDiscreteIoExpander::getAddress ( )

Retrieve device address.

Optional Function (Troubleshooting) This function is for testing and troubleshooting.

Returns

address of device (0..7)

Usage:

```
...
I2cDiscreteIoExpander device;
...
address = device.getAddress();
...
```

Examples:

examples/BareMinimum/BareMinimum.ino.

```
{
  return address_;
}
```

5.1.3.4 uint16_t I2cDiscreteIoExpander::getPorts ( )

Retrieve ports 1 (P17..P10), 0 (P07..P00).

Required Function Call this from within `loop()` to retrieve ports.

Returns

ports of device (0x0000..0xFFFF)

Usage:

```
...
I2cDiscreteIoExpander device;
...
ports = device.getPorts();
...
```

Examples:

examples/BareMinimum/BareMinimum.ino.

```
{
  return ports_;
}
```

5.1.3.5 void I2cDiscreteIoExpander::enableBitwiseInversion ( )

Enable bitwise inversion.

All bits will be inverted prior to future read/write operations.

Usage:

```
...
I2cDiscreteIoExpander device;
...
device.enableBitwiseInversion();       // bits will now
       inverted
...
```

See Also

I2cDiscreteIoExpander::disableBitwiseInversion()
I2cDiscreteIoExpander::isInverted()

```
{
  ports_ = shouldInvert_ ? ports_ : ~ports_;
  shouldInvert_ = true;
}
```

5.1.3.6 void I2cDiscreteIoExpander::disableBitwiseInversion ( )

Disable bitwise inversion.

Bits will not be inverted prior to future read/write operations.

Usage:

```
...
I2cDiscreteIoExpander device;
...
device.disableBitwiseInversion();      // bits will no
       longer be inverted
...
```

See Also

I2cDiscreteIoExpander::enableBitwiseInversion()
I2cDiscreteIoExpander::isInverted()

```
{
  ports_ = shouldInvert_ ? ~ports_ : ports_;
  shouldInvert_ = false;
}
```

5.1.3.7 bool I2cDiscreteIoExpander::isInverted ( )

Indicate whether bitwise inversion is enabled.

Returns

status of bitwise inversion (false=not inverted, true=inverted)

Usage:

```
    ...
    I2cDiscreteIoExpander device;
    ...
    if (device.isInverted())
    {
      // do something
    }
    ...
```

See Also

I2cDiscreteIoExpander::enableBitwiseInversion()
I2cDiscreteIoExpander::disableBitwiseInversion()

```
{
  return shouldInvert_;
}
```

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

- I2cDiscreteIoExpander.h
- I2cDiscreteIoExpander.cpp

6 Example Documentation

6.1 examples/BareMinimum/BareMinimum.ino

```
/*

  BareMinimum.ino - example using I2cDiscreteIoExpander library

  This file is part of I2cDiscreteIoExpander.

  I2cDiscreteIoExpander is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  I2cDiscreteIoExpander is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.
  You should have received a copy of the GNU General Public License
  along with I2cDiscreteIoExpander.  If not, see <http://www.gnu.org/licenses/
      >.

  Written by Doc Walker (Rx)
  Copyright © 2009-2013 Doc Walker <4-20ma at wvfans dot net>

*/


#include <Wire.h>
#include <I2cDiscreteIoExpander.h>


// instantiate I2cDiscreteIoExpander object
I2cDiscreteIoExpander device;


void setup()
{
  // initialize i2c interface
  Wire.begin();

  // initialize serial interface
  Serial.begin(19200);
}


void loop()
{
  uint8_t status;
  static uint16_t i;

  // display device information on serial console
  Serial.print("Loop ");
  Serial.print(++i, DEC);
  Serial.print(", address ");
  Serial.print(device.getAddress(), DEC);
  Serial.print(", ");

  // attempt to write 16-bit word
  status = device.digitalWrite(i);
  if (TWI_SUCCESS == status)
  {
    // display success information on serial console
    Serial.print("write 0x");
    Serial.print(i, HEX);
    Serial.print(", ");
  }
  else
  {
    // display error information on serial console
    Serial.print("write error ");
    Serial.print(status, DEC);
    Serial.print(", ");
  }

  // attempt to read 16-bit word
  status = device.digitalRead();
  if (TWI_SUCCESS == status)
  {
    // display success information on serial console
    Serial.print("read 0x");
    Serial.print(device.getPorts(), HEX);
    Serial.println(".");
  }
  else
  {
    // display error information on serial console
    Serial.print("read error ");
    Serial.print(status, DEC);
    Serial.println(".");
  }

  delay(1000);
}
```

6.2 examples/MultipleDevices/MultipleDevices.ino

```
/*

  MultipleDevices.ino - example using I2cDiscreteIoExpander library

  This file is part of I2cDiscreteIoExpander.

  I2cDiscreteIoExpander is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  I2cDiscreteIoExpander is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with I2cDiscreteIoExpander.  If not, see <http://www.gnu.org/licenses/
      >.

  Written by Doc Walker (Rx)
  Copyright © 2009-2013 Doc Walker <4-20ma at wvfans dot net>

*/


#include <Wire.h>
#include <I2cDiscreteIoExpander.h>


// instantiate I2cDiscreteIoExpander objects
// device addresses purposely out of order to illustrate constructor
I2cDiscreteIoExpander device[8] = { 6, 4, 2, 0, 1, 3, 5, 7
       };


void setup()
{
  // initialize i2c interface
  Wire.begin();

  // initialize serial interface
  Serial.begin(19200);
}


void loop()
{
  uint8_t status, i;
  static uint16_t j;

  // loop to write/read each device
  for (i = 0; i < 8; i++)
  {
    // display device information on serial console
    Serial.print("Loop ");
    Serial.print(j, DEC);
    Serial.print(", device[");
    Serial.print(i, DEC);
    Serial.print("], address ");
    Serial.print(device[i].getAddress(), DEC);
    Serial.print(", ");

    // attempt to write 16-bit word
    status = device[i].digitalWrite(j);
    if (TWI_SUCCESS == status)
    {
      // display success information on serial console
      Serial.print("write 0x");
      Serial.print(j, HEX);
      Serial.print(", ");
    }
    else
    {
      // display error information on serial console
      Serial.print("write error ");
      Serial.print(status, DEC);
      Serial.print(", ");
    }

    // attempt to read 16-bit word
    status = device[i].digitalRead();
    if (TWI_SUCCESS == status)
    {
      // display success information on serial console
      Serial.print("read 0x");
      Serial.print(device[i].getPorts(), HEX);
      Serial.println(".");
    }
    else
    {
      // display error information on serial console
      Serial.print("read error ");
      Serial.print(status, DEC);
      Serial.println(".");
    }
  }

  j++;
  Serial.println("- - - - - - - - - - - - - - - - - - - - - - -");
  delay(1000);
}
```

Index