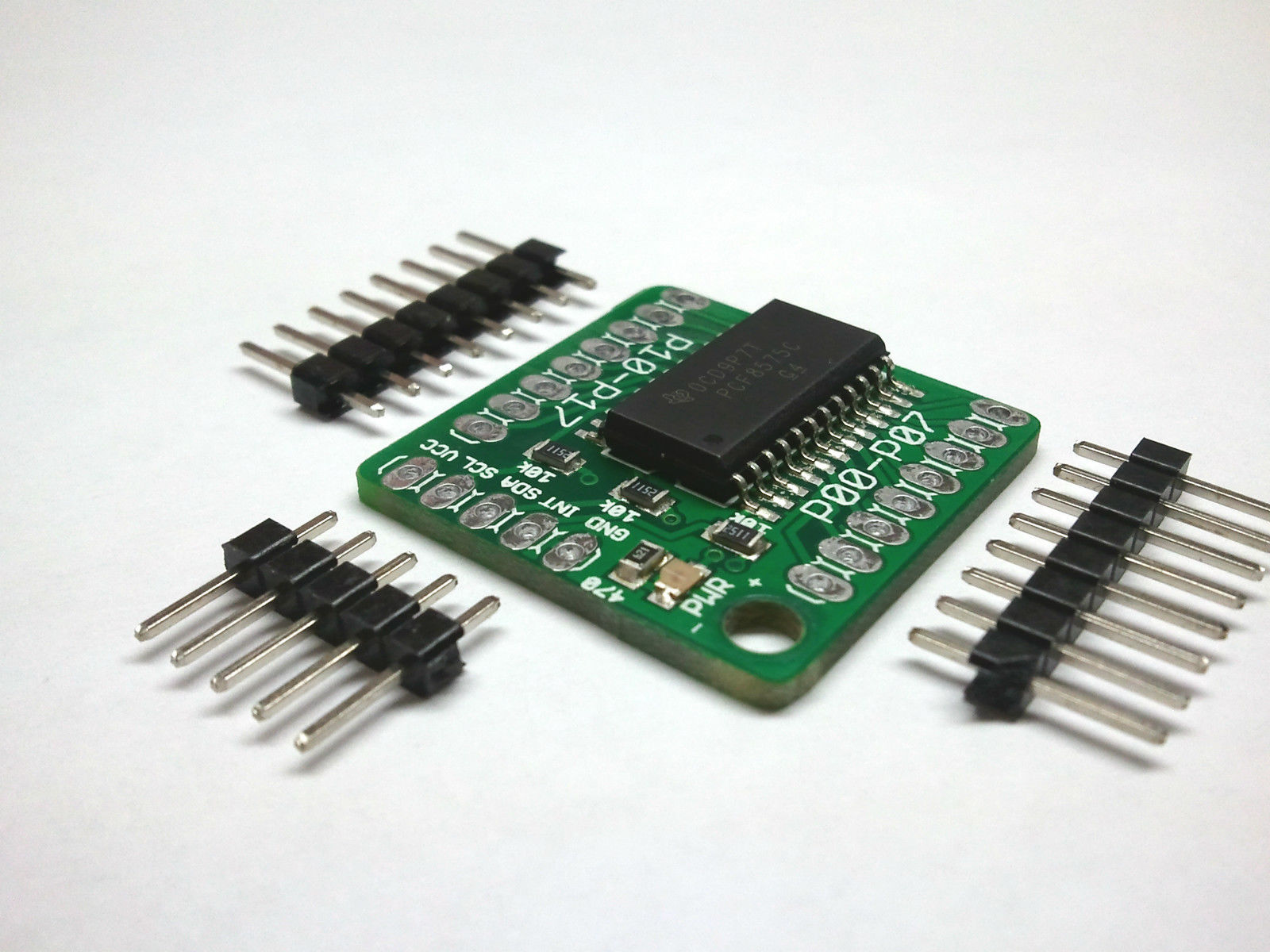
20140905 PCA9555D 16 Bit I/O Expander - I2C for Micro Controllers, Arduino, w/ LED & Pins

UNSUCCESSFUL IMPLEMENTATION



**9/15/2014**

**Used Arduino IO EXPANDER TUTORIAL AND CODE customized for the PCA9555. AudTutMod16LitesUp1**

**Notes:**

**Lights work on one side only. IO1\_0 thru 7**

**First address set to 20 is only way to see outputs.**

**When tie INT to display led & 12- 13 lamp blinks. looks like unit is working.**

* **Pull up (10K)on sclk sdat and int.**
* **12 to Int only no active lamps. (no led connected).**

**/\***

**\* A4 is SDA**

**\* A5 is SCL**

**\* Both need pull-up resistors.**

**\* The value of the resistors is not critical.**

**\* Anything from 1k8 (1800 ohms) to 47k (47000 ohms)**

**\* should work; 1k8, 4k7 and 10k are common values. Start**

**\* with 1k8 as this gives you the best performance. If**

**\* the resistors are missing, the SCL and SDA lines will**

**\* always be low - nearly 0 volts - and the I2C bus will**

**\* not work.**

**\***

**\* Connect a LED between Arduino pin D13 and Gnd so we can**

**\* show activity, and connect pin 13 (/INT) of the 8574 to**

**\* pin D12 on the Arduino. We use the /INT signal to tell**

**\* us when to read the data from the 8574.**

**\* (In a real application, you would probably want to**

**\* connect /INT to an Arduino interrupt (and associated**

**\* ISR routine) to avoid the polling loop...)**

**\*/**

**#include <Wire.h>**

**// 8574 Address range is 0x20-0x27**

**// 8574A Address range is 0x38-0x3F**

**// 9555 Address range is 0x20-0x27 (same as 8574, bummer)**

**#define INaddr 0x20 // 8574A addr 000**

**#define OUTaddr 0x18 // 8574A addr 001**

**#define NXP\_INPUT (0) // For NXP9555**

**#define NXP\_OUTPUT (2) // See data sheet**

**#define NXP\_INVERT (4) // for details...**

**#define NXP\_CONFIG (6)**

**void setup()**

**{**

**pinMode(12, INPUT); // to read /INT**

**pinMode(13, OUTPUT); // to show we are working**

**Wire.begin();**

**expanderSetInput16(INaddr, 0xFF);**

**}**

**// I2C routines to talk to 8574 and 8574A**

**///void expanderSetInput(int i2caddr, byte dir) {**

**/// Wire.beginTransmission(INaddr);**

**/// Wire.write(dir); // outputs high for input**

**/// Wire.endTransmission();**

**///}**

**///byte expanderRead(int i2caddr) {**

**/// int \_data = -1;**

**/// Wire.requestFrom(i2caddr, 1);**

**/// if(Wire.available()) {**

**/// \_data = Wire.read();**

**/// }**

**/// return \_data;**

**///}**

**///void expanderWrite(int i2caddr, byte data)**

**///{**

**/// Wire.beginTransmission(i2caddr);**

**/// Wire.write(data);**

**/// Wire.endTransmission();**

**///}**

**// I2C routines to talk to a PCA9555**

**void expanderSetInput16(int i2caddr, int dir) {**

**Wire.beginTransmission(i2caddr);**

**Wire.write(NXP\_CONFIG);**

**Wire.write(0xff & dir); // low byte**

**Wire.write(dir >> 8); // high byte**

**Wire.endTransmission();**

**}**

**void expanderWrite16(int i2caddr, int data) {**

**Wire.beginTransmission(i2caddr);**

**Wire.write(NXP\_OUTPUT);**

**Wire.write(0xff & data); // low byte**

**Wire.write(data >> 8); // high byte**

**Wire.endTransmission();**

**}**

**int expanderRead16(int i2caddr) {**

**int \_data = 0;**

**Wire.beginTransmission(i2caddr);**

**Wire.write(NXP\_INPUT);**

**Wire.endTransmission();**

**Wire.requestFrom(i2caddr, 2);**

**if(Wire.available()) {**

**\_data = Wire.read();**

**}**

**if(Wire.available()) {**

**\_data |= (Wire.read() << 8);**

**}**

**return \_data;**

**}**

**void loop()**

**{**

**while (digitalRead(12) == 1) { /\* wait for /INT to go low \*/ }**

**digitalWrite(13, 1);**

**delay(50); // Flash my bling**

**digitalWrite(13, 0);**

**int data = expanderRead16(INaddr);**

**if (data != -1) {**

**expanderWrite16(OUTaddr, (byte)data);**

**} else {**

**for (int x = 0; x <= 8; x++) { // flash error pattern**

**digitalWrite(13, 1); delay(100);**

**digitalWrite(13, 0); delay(400);**

**digitalWrite(13, 1); delay(400);**

**digitalWrite(13, 0); delay(100);**

**}**

**}**

**}**

[**I2C Tutorial**](http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html)

[**PCA9555 I2C IO EXPANDER DATA**](http://www.nxp.com/documents/data_sheet/PCA9555.pdf)

[**Arduino IO EXPANDER TUTORIAL and Code**](http://playground.arduino.cc/Code/I2CPortExpander8574)

[**PCA9555 PROJECT 2**](http://www.bristolwatch.com/arduino/arduino6.htm)

[**Step by Step Tutorial**](http://www.eng.utah.edu/~cs5968/handouts/Arduino%20I2C%20Expansion%20I_O%20.pdf)

[**Demo Code for the PCA9555**](http://www.kerrywong.com/2011/03/05/tca9555-library-for-arduino/)

[**Our Exact io expander**](https://www.sparkfun.com/products/8130)

[**13 on**](https://github.com/agenteaty007/PCA9555_Experiments/blob/master/Arduino/PCA9555_Blink/PCA9555_Blink.ino)

[**2 pcf8574**](http://hobbybotics.com/projects/hobbybotics-pcf8574a-i2c-io-expander/)

[**IOX 16**](http://www.unifiedmicro.com/IOX-16_Output_Demo.txt)

[**pca9555 direct wiring**](http://fritzing.org/projects/16-leds-controlled-by-a-pca9555-and-an-arduino)

[**Fritzing wiring and code pca9555 arduino**](http://fritzing.org/projects/16-leds-controlled-by-a-pca9555-and-an-arduino)

[**PCF8574 (DUAL) INPUT AND OUTPUT WIRING CODE**](http://garagelab.com/profiles/blogs/tutorial-arduino-i-o-port-expander-with-pcf8574)

[**pcf8574 part 2**](http://tronixstuff.com/2010/10/29/tutorial-arduino-and-the-i2c-bus-part-two/)

[**Port Expander I2C PCF8574**](http://blog.iharder.net/2014/08/08/msp430-or-arduino-library-for-pcf8574-i2c-port-expander/)

[**Port Expander PCF8574 CODE**](http://blog.iharder.net/2014/08/08/msp430-or-arduino-library-for-pcf8574-i2c-port-expander/)

**PCA9555D based I/O Expander with I2C interface for use with all microcontrollers!**

**PCA9555D Breakout Board Features:**

* **Add 16 additional inputs or outputs to your controller over an I2C interface**
* **Interrupt pin informs on input change**
* **Simple I2C interface**
* **On board LED to indicate power to module**
* **Mounting hole for 4-40 standoff**
* **On board address solder jumpers allow up to 8 devices for a total of 128 extra I/Os**
* **On board pull up resistors for INT, SDA, SCL with selectable solder jumpers**
* **Small footprint of just 25.4x25.4mm**
* **Header pins included**

**Features of the PCA9555D from the manufacturer:**

**1. General description**

**The PCA9555 is a 24-pin CMOS device that provides 16 bits of General Purpose parallel Input/Output (GPIO) expansion for I2C-bus/SMBus applications and was developed to enhance the NXP Semiconductors family of I2C-bus I/O expanders. The improvements include higher drive capability, 5 V I/O tolerance, lower supply current, individual I/O configuration, and smaller packaging. I/O expanders provide a simple solution when additional I/O is needed for ACPI power switches, sensors, push buttons, LEDs, fans, etc. The PCA9555 consists of two 8-bit Configuration (Input or Output selection); Input, Output and Polarity Inversion (active HIGH or active LOW operation) registers. The system master can enable the I/Os as either inputs or outputs by writing to the I/O configuration bits. The data for each Input or Output is kept in the corresponding Input or Output register. The polarity of the read register can be inverted with the Polarity Inversion register. All registers can be read by the system master. Although pin-to-pin and I2C-bus address compatible with the PCF8575, software changes are required due to the enhancements, and are discussed in Application Note AN469. The PCA9555 open-drain interrupt output is activated when any input state differs from its**

**corresponding input port register state and is used to indicate to the system master that an input state has changed. The power-on reset sets the registers to their default values and initializes the device state machine.**

**Three hardware pins (A0, A1, A2) vary the fixed I2C-bus address and allow up to eight devices to share the same I2C-bus/SMBus. The fixed I2C-bus address of the PCA9555 is**

**the same as the PCA9554, allowing up to eight of these devices in any combination to share the same I2C-bus/SMBus.**

**2. Features**

* **Operating power supply voltage range of 2.3 V to 5.5 V**
* **5 V tolerant I/Os**
* **Polarity Inversion register**
* **Active LOW interrupt output**
* **Low standby current**
* **Noise filter on SCL/SDA inputs**
* **No glitch on power-up**
* **Internal power-on reset**
* **16 I/O pins which default to 16 inputs**
* **0 Hz to 400 kHz clock frequency**
* **ESD protection exceeds 2000 V HBM per JESD22-A114, 200 V MM per**
* **JESD22-A115, and 1000 V CDM per JESD22-C101**

**Datasheet:**

**<http://www.nxp.com/documents/data_sheet/PCA9555.pdf>**

**Arduino Code:**

**<http://arduino.cc/playground/Code/I2CPortExpander8574>**

**---------------------------CODE BELOW----------------------**

# I2C Port expanders - 8574, 8574A, 8575, 9555 etc

**If you just don't have enough digital I/O pins on your Arduino to interface with all your sensors and controls, you might want to look at using the I2C bus to connect a few port expander chips:**

* **The 8574 and 8574A (TI and NXP) have 8 digital I/O bits and can be set to addresses 0x20-0x27 and 0x38-0x3F respectively. Cost is about $1.00**
* **The 8575 (TI and NXP) has 16 digital I/O bits at addresses 0x20-0x27. Cost is about $1.10**
* **The MCP23016 (Microchip) and PCA9555 (NXP) have 16 bits of digital I/O at adresses 0x20-0x27 (inexpensive at $1.30, but are more complicated to use)**
* **The PCA9698 from Phillips has 40 bits of digital I/O, can be addressed at 64 different addresses, is extremely powerful, but is a $4.00 56-pin surface-mount part.**

**Common among these chips is the feature that any of their digital I/O ports can be configured as an input or output, and that many (usually up to 8 of each type) can be used at the same time on the same bus. Using both 8574 and 8574A parts, you could have 16 independently addressable devices with 8 bits of I/O each - giving you a total of 128 additional I/O lines.**

**The I2C bus on the Arduino uses Analog pins 4 and 5, a couple of pull up resistors and the Wire library. The I2C bus is widely documented; I find** [**http://www.robot-electronics.co.uk/htm/using\_the\_i2c\_bus.htm**](http://www.robot-electronics.co.uk/htm/using_the_i2c_bus.htm) **to be a typically useful writeup. If you want to have more fun with I2C, check out the BlinkM (**[**http://blinkm.thingm.com/**](http://blinkm.thingm.com/)**) - a 3-color LED controlled via I2C.**

**Since the Wire library does almost all the work for you, there is very little to do in this example that uses one 8574A as an 8-input device and another as an 8-output. The code sample includes versions of the routines (unused here) to do the same for 16 bit PCA9555's.**

**sample8574A.pde**

| **/\*  \* A4 is SDA  \* A5 is SCL  \* Both need pull-up resistors.  \* The value of the resistors is not critical.   \* Anything from 1k8 (1800 ohms) to 47k (47000 ohms)  \* should work; 1k8, 4k7 and 10k are common values. Start  \* with 1k8 as this gives you the best performance. If  \* the resistors are missing, the SCL and SDA lines will  \* always be low - nearly 0 volts - and the I2C bus will  \* not work.   \*  \* Connect a LED between Arduino pin D13 and Gnd so we can  \* show activity, and connect pin 13 (/INT) of the 8574 to  \* pin D12 on the Arduino. We use the /INT signal to tell  \* us when to read the data from the 8574.  \* (In a real application, you would probably want to  \* connect /INT to an Arduino interrupt (and associated  \* ISR routine) to avoid the polling loop...)  \*/  #include <Wire.h> // 8574 Address range is 0x20-0x27 // 8574A Address range is 0x38-0x3F // 9555 Address range is 0x20-0x27 (same as 8574, bummer)  #define INaddr 0x38 // 8574A addr 000 #define OUTaddr 0x39 // 8574A addr 001  #define NXP\_INPUT (0) // For NXP9555 #define NXP\_OUTPUT (2) // See data sheet #define NXP\_INVERT (4) // for details... #define NXP\_CONFIG (6)  void setup() {  pinMode(12, INPUT); // to read /INT  pinMode(13, OUTPUT); // to show we are working  Wire.begin();   expanderSetInput(INaddr, 0xFF); }   // I2C routines to talk to 8574 and 8574A void expanderSetInput(int i2caddr, byte dir) {  Wire.beginTransmission(INaddr);  Wire.send(dir); // outputs high for input  Wire.endTransmission();  }  byte expanderRead(int i2caddr) {  int \_data = -1;  Wire.requestFrom(i2caddr, 1);  if(Wire.available()) {  \_data = Wire.receive();  }  return \_data; }  void expanderWrite(int i2caddr, byte data) {  Wire.beginTransmission(i2caddr);  Wire.send(data);  Wire.endTransmission();  }  // I2C routines to talk to a PCA9555 void expanderSetInput16(int i2caddr, int dir) {  Wire.beginTransmission(i2caddr);  Wire.send(NXP\_CONFIG);  Wire.send(0xff & dir); // low byte  Wire.send(dir >> 8); // high byte  Wire.endTransmission();  }  void expanderWrite16(int i2caddr, int data) {  Wire.beginTransmission(i2caddr);  Wire.send(NXP\_OUTPUT);  Wire.send(0xff & data); // low byte  Wire.send(data >> 8); // high byte  Wire.endTransmission();  }  int expanderRead16(int i2caddr) {  int \_data = 0;  Wire.beginTransmission(i2caddr);  Wire.send(NXP\_INPUT);  Wire.endTransmission();   Wire.requestFrom(i2caddr, 2);  if(Wire.available()) {  \_data = Wire.receive();  }  if(Wire.available()) {  \_data |= (Wire.receive() << 8);  }  return \_data; }   void loop() {  while (digitalRead(12) == 1) { /\* wait for /INT to go low \*/ }  digitalWrite(13, 1);  delay(50); // Flash my bling  digitalWrite(13, 0);   int data = expanderRead(INaddr);  if (data != -1) {  expanderWrite(OUTaddr, (byte)data);  } else {  for (int x = 0; x <= 8; x++) { // flash error pattern  digitalWrite(13, 1); delay(100);   digitalWrite(13, 0); delay(400);   digitalWrite(13, 1); delay(400);   digitalWrite(13, 0); delay(100);   }  } }** |
| --- |

**---------------------------CODE ABOVE--------------------**

2 ea pcf8574ap recd 9/17/2014

**item sold by[thaishine](http://www.ebay.com/usr/thaishine?_trksid=p2059210.m2749.l2754)**

| **[2 x PCF8574 I2C I/O Bus Expander IC - USA Seller - Free Shipping](http://www.ebay.com/usr/thaishine?_trksid=p2059210.m2749.l2754)****[2 x PCF8574 I2C I/O Bus Expander IC - USA Seller - Free Shipping](http://www.ebay.com/itm/221533525890?_trksid=p2059210.m2749.l2649&ssPageName=STRK%3AMEBIDX%3AIT)****( 221533525890 )****Estimated delivery Thu, Sep 18****Tracking number: 9400110200829321046589** |
| --- |