tronixstuff

tronixstuff fun and learning with electronics

Categorized | **arduino, education, I2C, lesson, MCP23017, microcontrollers, tutorial**

Tutorial: Maximising your Arduino's I/O ports with MCP23017

Posted on 26 August 2011. Tags: arduino, bus, control, duemilanove, expander, i/o, i2c, input, lesson, lessons, mcp23017, microchip, output, tronixstuff, tutorial, tutorials, uno

This is chapter forty-one of a series originally titled "Getting Started/Moving Forward with Arduino!" by John Boxall – a series of articles on the Arduino universe. The first chapter is here, the complete series is detailed here.

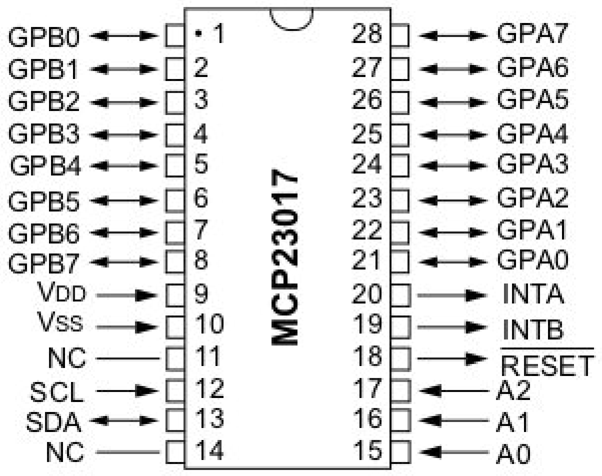
[Updated 04/12/2014]

In this article we discuss how to use the Microchip MCP23017 16-bit serial expander with I2C serial interface. This 28-pin IC offers sixteen inputs or outputs – and up to eight of the ICs can be used on one I2C bus… offering a maximum of 128 extra I/O ports. A few people may be thinking "Why not just get an Arduino Mega2560?" – a good question. However you may have a distance between the Arduino and the end-point of the I/O pins – so with these ICs you can run just four wires instead of a lot more; save board space with custom designs, and preserve precious digital I/O pins for other uses. Plus I think the I2C bus is underappreciated! So let's get started…

Here is our subject of the article in DIP form:

At this point you should also download yourself a copy of data sheet – it will be referred to several times, and very useful for reference and further reading. Furthermore if you are not familiar with Arduino and the I2C bus, please familiarise yourself with the I2C tutorials parts one and two. The MCP23017 can be quite simple or complex to understand, so the goal of this article is to try and make it as simple as possible. After reading this you should have the knowledge and confidence to move forward with using a MCP23017.

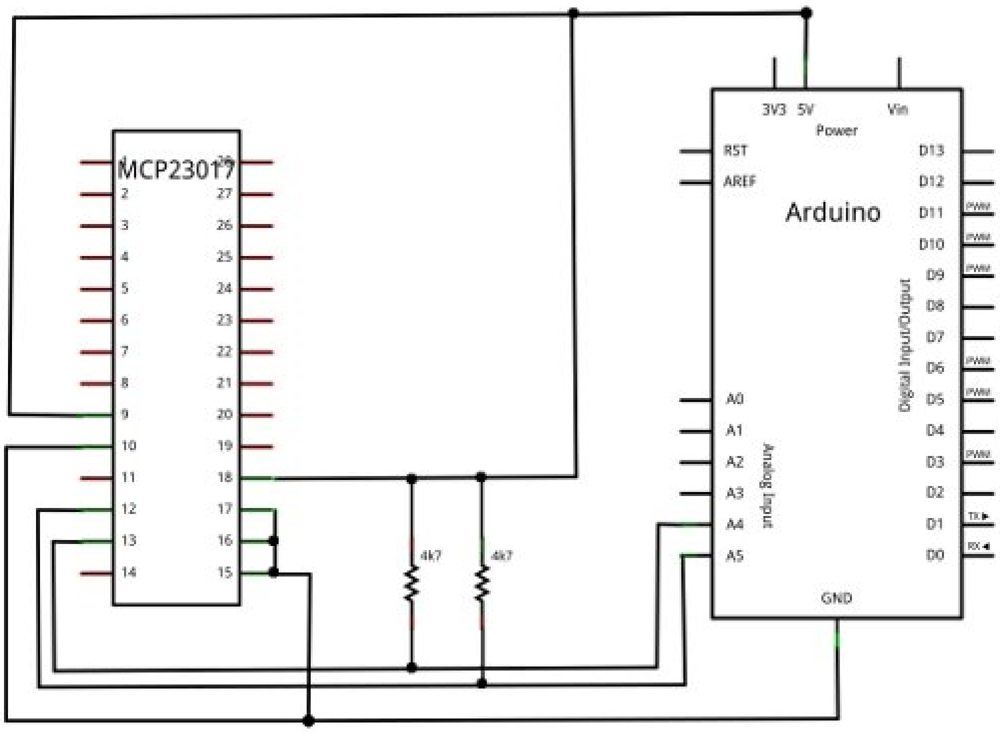
First, let’s look at the hardware basics of this IC. Consider the pinouts:



The sixteen I/O ports are separated into two ‘ports’ – A (on the right) and B (on the left. Pin 9 connects to 5V, 10 to GND, 11 isn’t used, 12 is the I2C bus clock line (Arduino Uno/Duemilanove analogue pin 5, Mega pin 21), and 13 is the I2C bus data line (Arduino Uno/Duemailnove analogue pin 4, Mega pin 20). External pull-up resistors should be used on the I2C bus – in our examples we use 4.7k ohm values. Pin 14 is unused, and we won’t be looking at interrupts, so ignore pins 19 and 20. Pin 18 is the reset pin, which is normally high – therefore you ground it to reset the IC. So connect it to 5V!

Finally we have the three hardware address pins 15~17. These are used to determine the I2C bus address for the chip. If you connect them all to GND, the address is 0x20. If you have other devices with that address or need to use multiple MCP23017s, see figure 1-2 on page eight of the data sheet. You can alter the address by connecting a combination of pins 15~17 to 5V (1) or GND (0). For example, if you connect 15~17 all to 5V, the control byte becomes 0100111 in binary, or 0x27 in hexadecimal.

Next, here is a basic schematic illustrating how to connect an MCP23017 to a typical Arduino board. It contains the minimum to use the IC, without any sensors or components on the I/O pins:



Now to examine how to use the IC in our sketches.

As you should know by now most I2C devices have several registers that can be addressed. Each address holds one byte of data that determines various options. So before using we need to set whether each port is an input or an output. First, we'll examine setting them as outputs. So to set port A to outputs, we use:

```
Wire.beginTransmission(0x20);
Wire.write(0x00); // IODIRA register
Wire.write(0x00); // set all of port A to outputs
Wire.endTransmission();
```

Then to set port B to outputs, we use:

```
Wire.beginTransmission(0x20);
Wire.write(0x01); // IODIRB register
Wire.write(0x00); // set all of port B to outputs
Wire.endTransmission();
```

So now we are in *void loop()* or a function of your own creation and want to control some output pins. To control port A, we use:

```
Wire.beginTransmission(0x20);
Wire.write(0x12); // address port A
Wire.write(??);  // value to send
Wire.endTransmission();
```

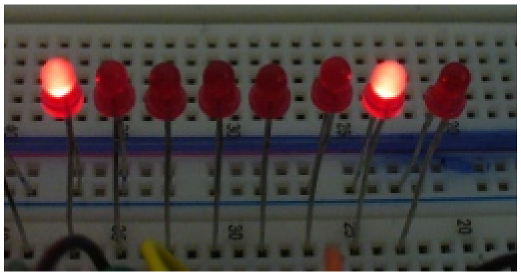
To control port B, we use:

```
Wire.beginTransmission(0x20);
Wire.write(0x13); // address port B
Wire.write(??);  // value to send
Wire.endTransmission();
```

… replacing ?? with the binary or equivalent hexadecimal or decimal value to send to the register.

To calculate the required number, consider each I/O pin from 7 to 0 matches one bit of a binary number – 1 for on, 0 for off. So you can insert a binary number representing the status of each output pin. Or if binary does your head in, convert it to hexadecimal. Or a decimal number. So for example, you want pins 7 and 1 on. In binary that would be 10000010, in hexadecimal that is 0x82, or 130 decimal. (Using decimals is convenient if you want to display values from an incrementing value or function result).

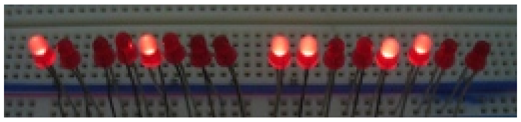
If you had some LEDs via resistors connected to the outputs, you would have this as a result of sending 0x82:



For example, we want port A to be 11001100 and port B to be 10001000 – so we send the following (note we converted the binary values to decimal):

```
Wire.beginTransmission(0x20);
Wire.write(0x12); // address port A
Wire.write(204); // value to send
Wire.endTransmission();
Wire.beginTransmission(0x20);
Wire.write(0x13); // address port B
Wire.write(136);     // value to send
Wire.endTransmission();
```

… with the results as such (port B on the left, port A on the right):



Now let's put all of this output knowledge into a more detailed example. From a hardware perspective we are using a circuit as described above, with the addition of a 560 ohm resistor followed by an LED thence to ground from on each of the sixteen outputs. Here is the sketch:

```
/*
 Example 41.1 - Microchip MCP23017 with Arduino
 http://tronixstuff.com/tutorials > chapter 41
 John Boxall | CC by-sa-nc
*/
// pins 15~17 to GND, I2C bus address is 0x20
#include "Wire.h"
void setup()
{
 Wire.begin(); // wake up I2C bus
// set I/O pins to outputs
 Wire.beginTransmission(0x20);
 Wire.write(0x00); // IODIRA register
 Wire.write(0x00); // set all of port A to outputs
 Wire.endTransmission();
Wire.beginTransmission(0x20);
 Wire.write(0x01); // IODIRB register
 Wire.write(0x00); // set all of port B to outputs
 Wire.endTransmission();
}
void binaryCount()
```

```
{
 for (byte a=0; a<256; a++)
 {
 Wire.beginTransmission(0x20);
 Wire.write(0x12); // GPIOA
 Wire.write(a); // port A
 Wire.endTransmission();
Wire.beginTransmission(0x20);
 Wire.write(0x13); // GPIOB
 Wire.write(a); // port B
 Wire.endTransmission();
 }
}
void loop()
{
 binaryCount();
 delay(500);
}
```

And here is the example blinking away:

Although that may have seemed like a simple demonstration, it was created show how the outputs can be used. So now you know how to control the I/O pins set as outputs. Note that you can’t source more than 25 mA of current from each pin, so if switching higher current loads use a transistor and an external power supply and so on.

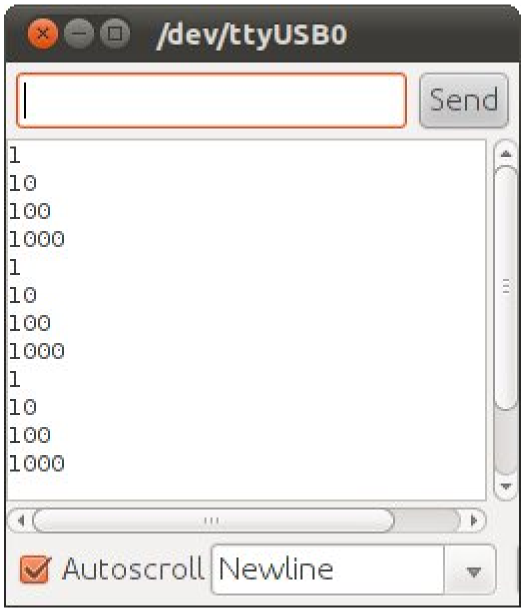
Now let’s turn the tables and work on using the I/O pins as **digital inputs**. The MCP23017 I/O pins default to input mode, so we just need to initiate the I2C bus. Then in the *void loop()* or other function all we do is set the address of the register to read and receive one byte of data.

For our next example, we have our basic sketch as described at the start of this article using four normally-open buttons (once again using the ‘button board‘) which are connected to port B inputs 0~3. Consider the first five lines of *void loop()* in the following example:

```
/*
 Example 41.2 - Microchip MCP23017 with Arduino
 http://tronixstuff.com/tutorials > chapter 41
 John Boxall | CC by-sa-nc
 */
// pins 15~17 to GND, I2C bus address is 0x20
```

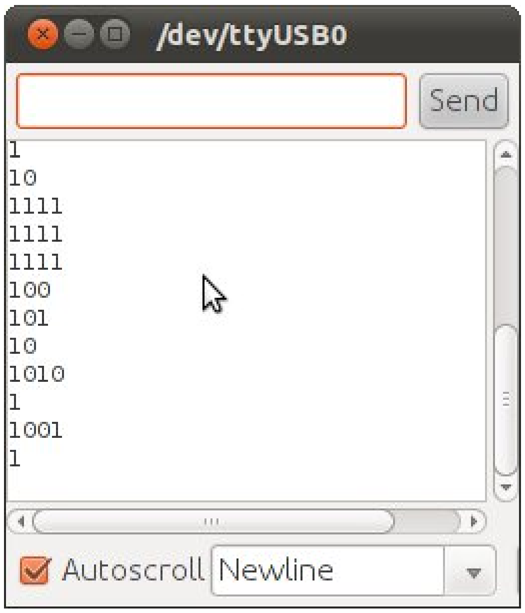
```
#include "Wire.h"
byte inputs=0;
void setup()
{
 Serial.begin(9600);
 Wire.begin(); // wake up I2C bus
}
void loop()
{
 Wire.beginTransmission(0x20);
 Wire.write(0x13); // set MCP23017 memory pointer to GPIOB address
 Wire.endTransmission();
 Wire.requestFrom(0x20, 1); // request one byte of data from MCP20317
 inputs=Wire.read(); // store the incoming byte into "inputs"
 if (inputs>0) // if a button was pressed
 {
 Serial.println(inputs, BIN); // display the contents of the GPIOB register in binary
 delay(200); // for debounce
 }
}
```

In this example *void loop()* sends the GPIOB address (0x13) to the IC. Then using *Wire.requestFrom()* it asks for one byte of data from the IC – the contents of the register at 0x13. This byte is stored in the variable *inputs*. Finally if *inputs* is greater than zero (i.e. a button has been pressed) the result is sent to the serial monitor window and displayed in binary. We display it in binary as this represents the state of the inputs 0~7. Here is an example of pressing the buttons 1, 2, 3 then 4 – three times:



```
1
10
100
1000
1
10
100
1000
1
10
100
1000
```

And as we are reading eight inputs at once – you can detect multiple keypresses. The following is an example of doing just that:



```
1
10
1111
1111
1111
100
101
10
1010
1
1001
1
```

As you can see pressing all four buttons returned 1111, or the first and third returned 101. Each combination of highs and lows on the inputs is a unique 8-bit number that can also be interpreted in decimal or hexadecimal. And if you wanted to read all sixteen inputs at once, just request and store two bytes of data instead of one.

For our last example – a demonstration of using port A as outputs and port B as inputs. Four LEDs with matching resistors are connected to port A outputs 0~3, with the buttons connected as per example 41.2. Here is the sketch:

```
/*
 Example 41.3 - Microchip MCP23017 with Arduino
 http://tronixstuff.wordpress.com/tutorials > chapter 41
 John Boxall | CC by-sa-nc
*/

// pins 15~17 to GND, I2C bus address is 0x20
#include "Wire.h"
byte inputs=0;

void setup()
{
  Serial.begin(9600);
  Wire.begin(); // wake up I2C bus

  Wire.beginTransmission(0x20);
  Wire.write(0x00); // IODIRA register
  Wire.write(0x00); // set all of bank A to outputs
  Wire.endTransmission();
}

void loop()
{
  // read the inputs of bank B
  Wire.beginTransmission(0x20);
  Wire.write(0x13);
  Wire.endTransmission();
  Wire.requestFrom(0x20, 1);
  inputs=Wire.read();

  // now send the input data to bank A
  Wire.beginTransmission(0x20);
  Wire.write(0x12); // GPIOA
  Wire.write(inputs);    // bank A
  Wire.endTransmission();
  delay(200); // for debounce
}
```

By now there shouldn't be any surprises in the last example – it receives a byte that represents port B, and sends that byte out to port A to turn on the matching outputs and LEDs. For the curious, here it is in action:

So there you have it… another way to massively increase the quantity of digital I/O pins on any Arduino system by using the I2C bus. You can get the MCP23017 from Tronixlabs.

tronixlabs

Have fun and keep checking into **tronixstuff.com**. Why not follow things on **twitter**, **Google+**, subscribe for email updates or RSS using the links on the right-hand column, or join our **forum** – dedicated to the projects and related items on this website.

Bio **Latest Posts**

John Boxall

Founder, owner and managing editor of tronixstuff.com.

Like this:

★ Like

6 bloggers like this.

← August 2011 Competition
Tutorial: Arduino and multiple thumbwheel switches →