***Categorized* | arduino, education, lesson, microcontrollers, port manipulation, tronixstuff, tutorial**

Tutorial: Arduino Port Manipulation

Posted on 22 October 2011. Tags: arduino, DDRB, DDRC, example, lesson, lessons, manipulation, PIND, port, PORTB, PORTC, PORTD, registers, tronixstuff, tutorial, tutorials

Control Arduino I/O pins faster and with less code in chapter forty-three of a series originally titled "Getting Started/Moving Forward with Arduino!" by John Boxall – a series of articles on the Arduino universe.

[Updated 19/01/13]

In this article we are going to revisit the I/O pins, and use what is called "Port Manipulation" to control them in a much faster manner than using digitalWrite()/digitalRead().

Why?

Speed! Using this method allows for much faster I/O control, and we can control or read groups of I/O pins simultaneously, not one at a time;

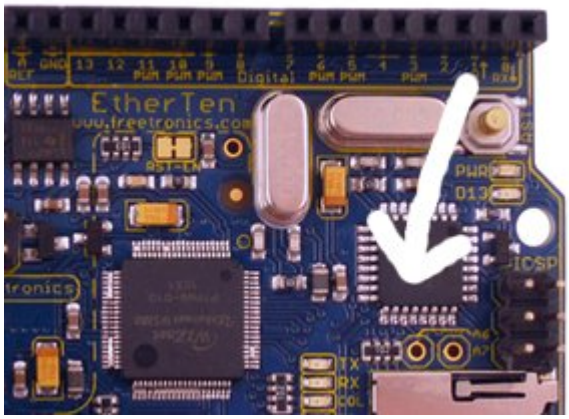
Memory! Using this method reduces the amount of memory your sketch will use.

Once again I will try and keep things as simple as possible. This article is written for Arduino boards that use the ATmega168 or ATmega328 microcontrollers (used in Arduino Duemilanove/Uno, Freetronics Eleven/EtherTen, etc).

First, we'll use the I/O as outputs. There are three *port registers* that we can alter to set the status of the digital and analogue I/O pins. A port register can be thought of as a special byte variable that we can change which is read by the microcontroller, therefore controlling the state of various I/O ports. We have three port registers to work with:

- D – for digital pins seven to zero (bank D)
- B – for digital pins thirteen to eight (bank B)
- C – for analogue pins five to zero (bank … C!)

Register C can control analogue pins seven to zero if using an Arduino with the TQFP style of ATmega328, such as the Nano or Freetronics EtherTen). For example:



It is very simple to do so. In void setup(), we use

```
DDRy = Bxxxxxxxx
```

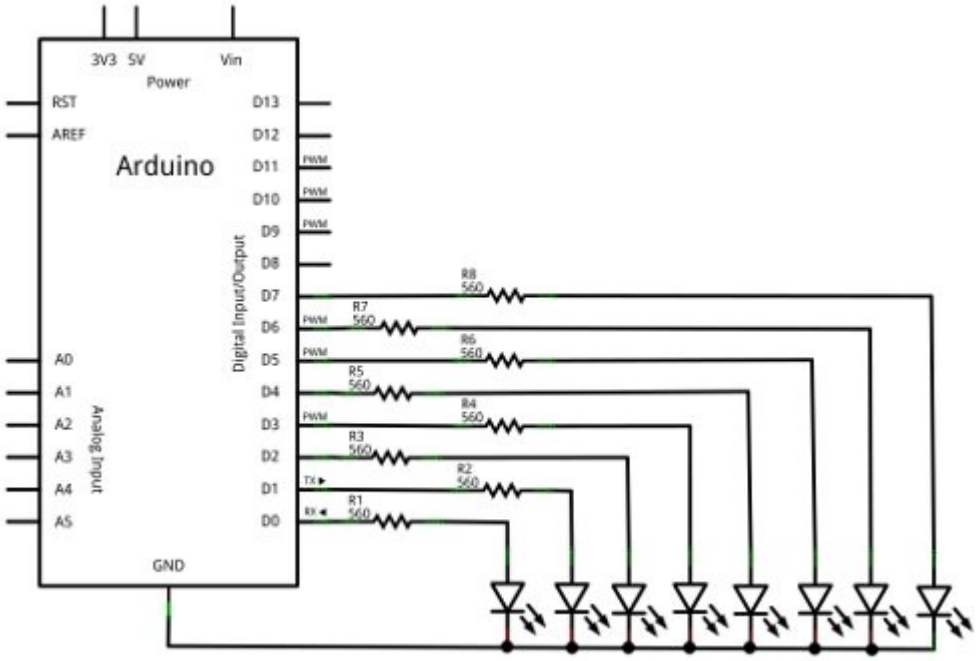
where y is the register type (B/C/D) and xxxxxxxx are eight bits that determine if a pin is to be an input or output. Use 0 for input, and 1 for output. The LSB (least-significant bit [the one on the right!]) is the lowest pin number for that register. Next, to control a bank of pins, use

```
PORTy = Bxxxxxxxx
```

where y is the register type (B/C/D) and xxxxxxxx are eight status bits – 1 for HIGH, 0 for LOW. This is demonstrated in the following example:

```
// Example 43.1
// tronixstuff.com/tutorials > chapter 43
// John Boxall - October 2011
// Digital 0~7 set to outputs, then on/off using port manipulation

void setup()
{
  DDRD = B11111111; // set PORTD (digital 7~0) to outputs
}

void loop()
{
  PORTD = B11110000; // digital 4~7 HIGH, digital 3~0 LOW
  delay(1000);
  PORTD = B00001111; // digital 4~7 LOW, digital 3~0 HIGH
  delay(1000);
}
```

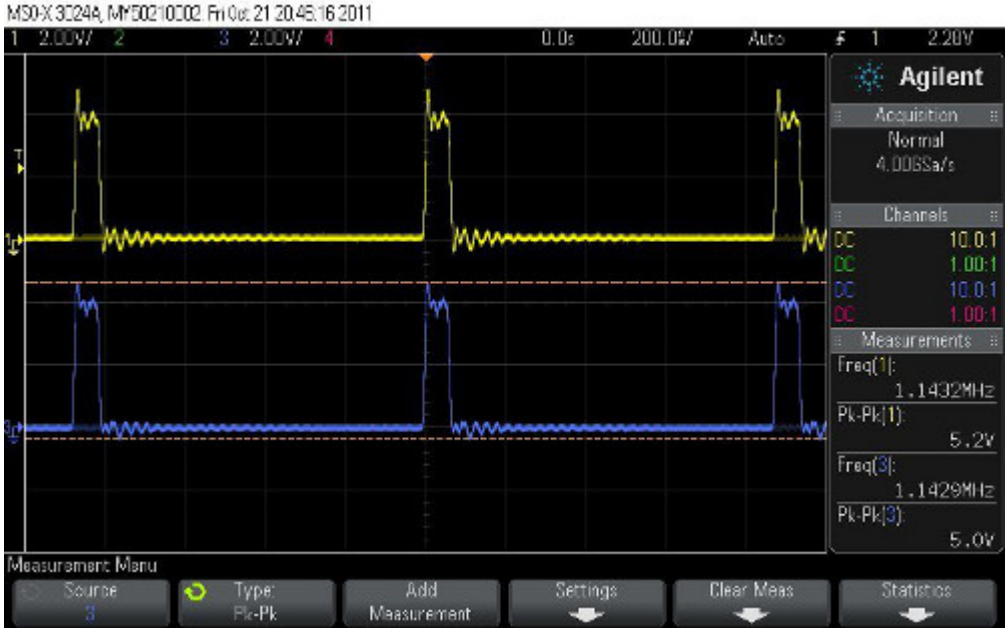
It sets digital pins 7~0 to output in void setup(). Then it alternates turning on and off alternating halves of digital pins 0~7. At the start I mentioned that using port manipulation was a lot faster than using regular Arduino I/O functions. How fast? To test the speed of port manipulation vs. using digitalWrite(), we will use the following circuit:



… and analyse the output at digital pins zero and seven using a digital storage oscilloscope. Our first test sketch turns on and off digital pins 0~7 without any delay between PORTD commands – in other words, as fast as possible. The sketch:

```
// Example 43.1.1
// tronixstuff.com/tutorials > chapter 43
// John Boxall - October 2011
// Digital 0~7 set to outputs, then on/off using port manipulation

void setup()
{
  DDRD = B11111111; // set PORTD (digital 7~0) to outputs
}

void loop()
{
  PORTD = B11111111;
  PORTD = B00000000;
}
```

In the image below, digital zero is channel one, and digital seven is channel three:

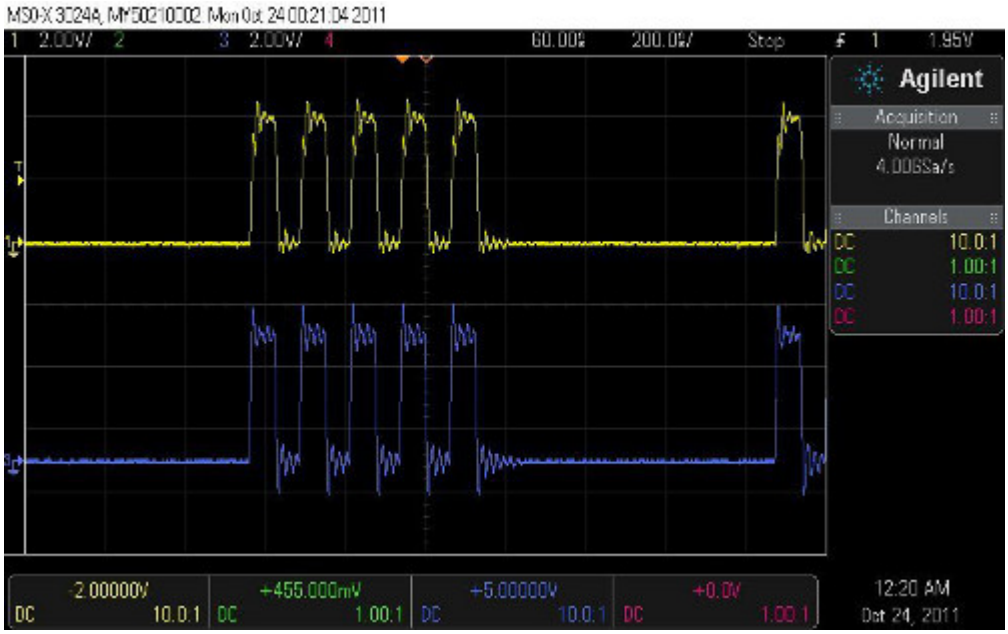


Wow – check the frequency measurements – 1.1432 MHz! Interesting to note the longer duration of time when the pins are low vs. high.

[Update] Well it turns out that the extra time in LOW includes the time for the Arduino to go back to the top of void loop(). This can be demonstrated in the following sketch. We turn the pins on and off five times instead of once:

```
// Example 43.1.2
// tronixstuff.com/tutorials > chapter 43
// John Boxall - October 2011

void setup()
{
  DDRD = B11111111; // set PORTD (digital 7~0) to outputs
}

void loop()
{
  PORTD = B11111111;
  PORTD = B00000000;
  PORTD = B11111111;
  PORTD = B00000000;
  PORTD = B11111111;
  PORTD = B00000000;
  PORTD = B11111111;
  PORTD = B00000000;
  PORTD = B11111111;
  PORTD = B00000000;
}
```

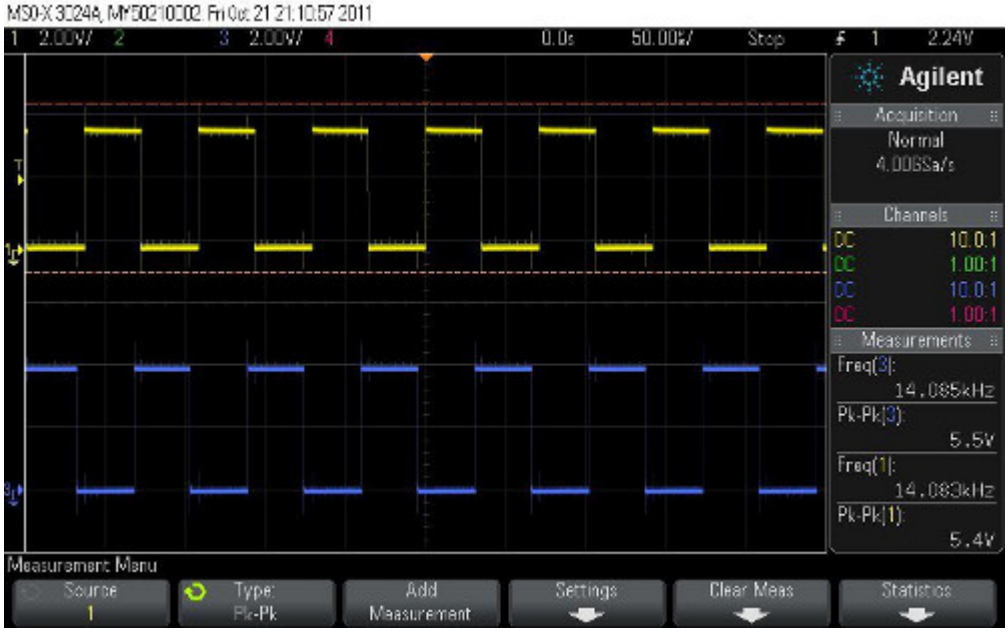
And the results from the MSO. You can see the duty cycle is much closer to 50% until the end of the sketch, at which point around 660 nanoseconds is the time used between the end of the last LOW period and the start of the next HIGH:



Next we do it the normal way, using this sketch:

```
// Example 43.2
// tronixstuff.com/tutorials > chapter 43
// John Boxall - October 2011
// Digital 0~7 set to outputs, then on/off using digitalWrite()

void setup()
{
  for (int a=0; a<8; a++)
  {
    pinMode(a, OUTPUT);
  }
}

void loop()
{
  for (int a=0; a<8; a++)
  {
    digitalWrite(a, HIGH);
  }
  for (int a=0; a<8; a++)
  {
    digitalWrite(a, LOW);
  }
}
```

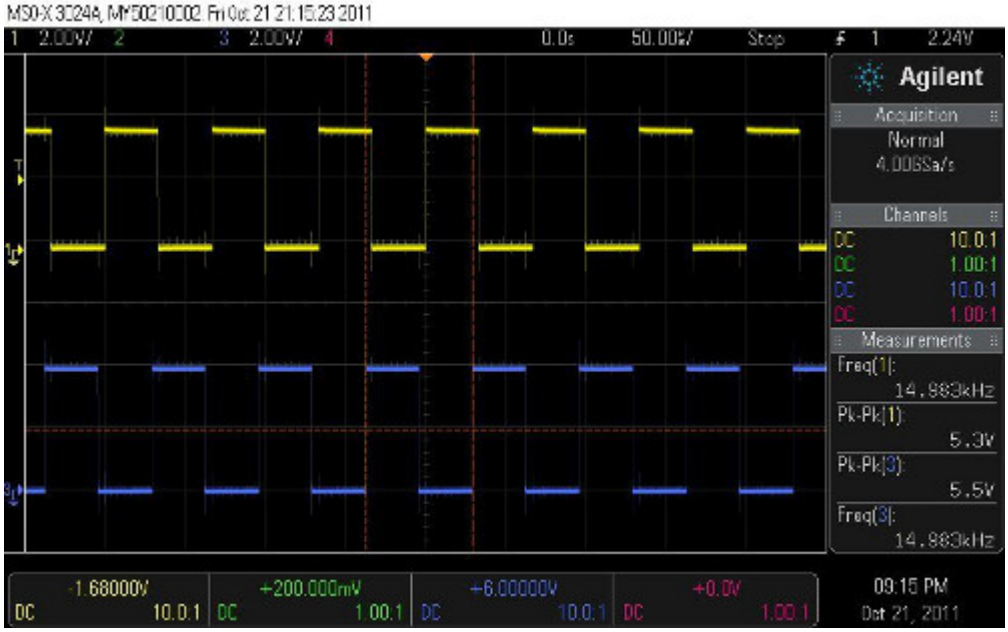
And the results:



That was a lot slower – we're down to 14.085 kHz, with a much neater square-wave output. Could some CPU time be saved by not using the for loop? We tested once more with the following sketch:

```
// Example 43.3
// tronixstuff.com/tutorials > chapter 43
// John Boxall - October 2011
// Digital 0~7 set to outputs, then on/off using individual digitalWrite()

void setup()
{
  for (int a=0; a<8; a++)
  {
    pinMode(a, OUTPUT);
  }
}

void loop()
{
  digitalWrite(0, HIGH);
  digitalWrite(1, HIGH);
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(0, LOW);
  digitalWrite(1, LOW);
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
}
```

and the results:



A small speed boost, the frequency has increased to 14.983 kHz. Hopefully you can now understand the benefits of using port manipulation. However there are a few things to take note of:

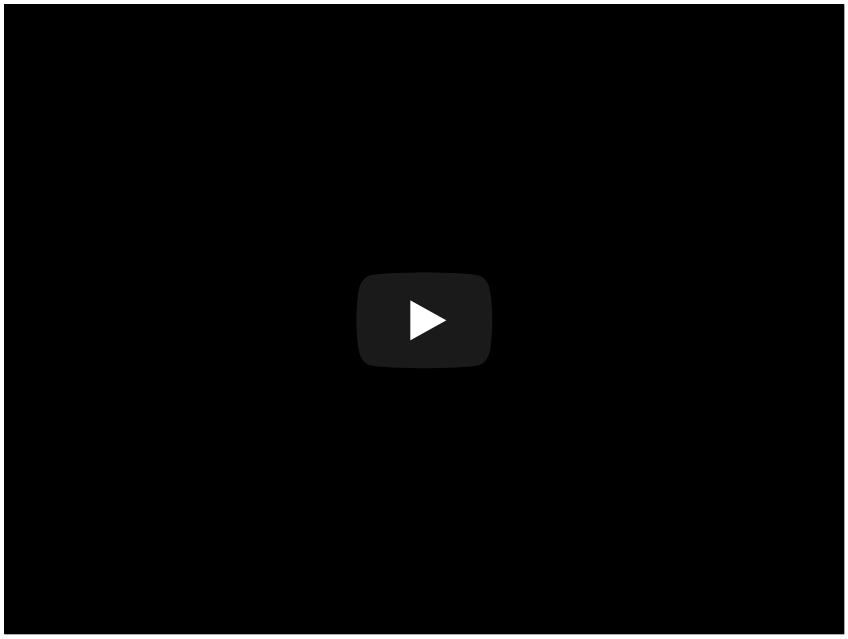
- You can't control digital pins 0 and 1 (in bank D) and use the serial monitor/port. For example if you set pin zero to output, it can't receive data!
- Always document your sketch – take pity on others who may need to review it later on and become puzzled about wchich bits are controlling or reading what!

Now to waste some electron flows by blinking LEDs. Using the circuit described earlier, the following sketch will create various effects for someone's enjoyment:

```
// Example 43.4
// tronixstuff.com/tutorials > chapter 43
// John Boxall - October 2011
// Fun with 8 LEDs on digital 7~0

void setup()
{
  DDRD = B11111111; // set PORTD (digital 7~0)
  //  to output
}

byte a = B11111111;
byte b = B00000001;
byte c = B10000000;
byte e = B10101010;

void krider()
{
  for (int k=0; k<5; k++)
  {
    for (int z=0; z<8; z++)
    {
      PORTD = b << z;
      delay(100);
    }

    for (int z=0; z<8; z++)
    {
      PORTD = c >> z;
      delay(100);
```

```
    }
  }
}

void onOff()
{
  for (int k=0; k<10; k++)
  {
    PORTD = a;
    delay(100);
    PORTD = 0;
    delay(100);
  }
}

void invBlink()
{
  for (int z=0; z<10; z++)
  {
    PORTD = e;
    delay(100);
    PORTD = ~e;
    delay(100);
  }
}

void binaryCount()
{
  for (int z=0; z<256; z++)
  {
    PORTD = z;
    delay(100);
  }
  PORTD=0;
}

void loop()
{
  invBlink();
  delay(500);
  binaryCount();
  delay(500);
  krider();
  delay(500);
  onOff();
}
```

And here it is in real life:



Now to use the I/O pins as inputs. Again, it is very simple to do so. In void setup(), we use

```
DDRy = Bxxxxxxxx
```

where y is the register type (B/C/D) and xxxxxxxx are eight bits that determine if a pin is to be an input or output. Use 0 for input. The LSB (least-significant bit [the one on the right!]) is the lowest pin number for that register. Next, to read the status of the pins we simply read the byte:

```
PINy
```

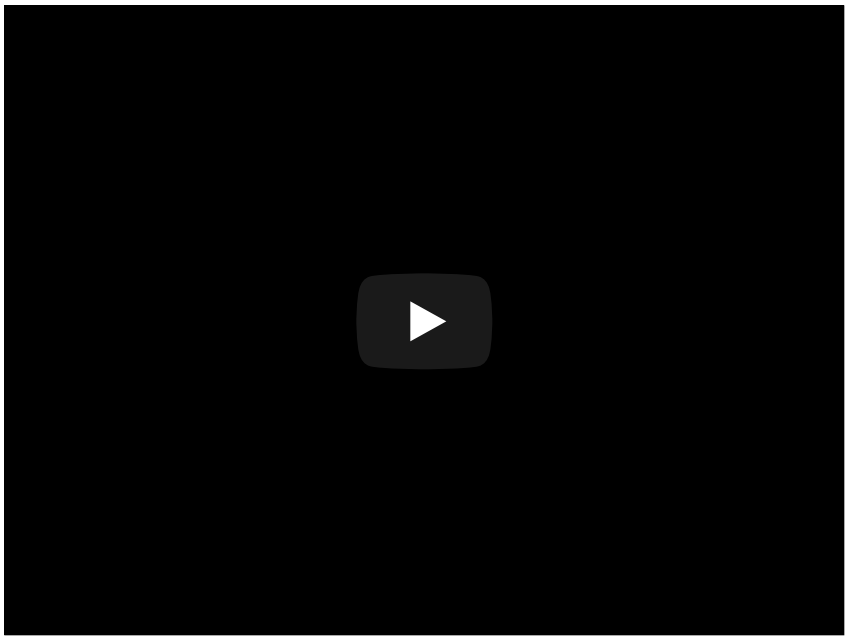
where y is the register type (B/C/D). So if you were using port B as inputs, and digital pins 8~10 were high, and 11~13 were low, PINB would be equal to B00000111. Really, that's it!

Now for another demonstration using both inputs and outputs. We will use a push-wheel switch from Chapter 40 on our inputs (digital pins 8~11), and a seven segment LED display for output (on digtal pins 7~0 – segments dp then a~f). The following sketch reads the input from the switch, which returns 0~9 in binary-coded decimal. This value is then used in the function void disp() to retrieve the matching byte from the array "segments", which contains the appropriate outputs to drive the seven segment LED display unit. Here is the sketch:

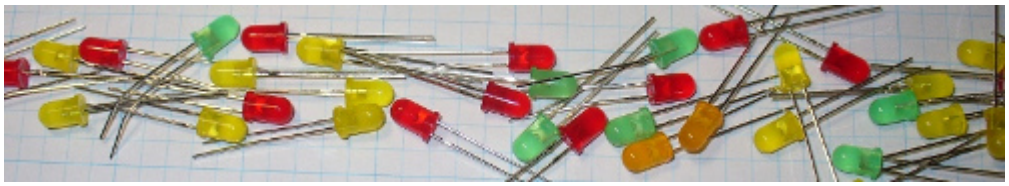
```
// Example 43.5
// tronixstuff.com/tutorials > chapter 43
// John Boxall - October 2011
// inputs and outputs

byte segments[] = {
  B01111110, B00110000, B01101101, B01111001, B00110011, B01011011, B01011111, B01110000, B01111
// digital pins 7~0 connected to display pins dp,a~g
void setup()
{
  DDRB = B00000000; // set PORTB (digital 13~8) to inputs
  DDRD = B11111111; // set PORTD (digital 7~0) to outputs
}

void disp(int z)
{
  PORTD = segments[z];
}

void loop()
{
```

```
22      disp(PINB);
23      delay(100);
24  }
```

And the ubiquitous demonstration video:



By now I hope you have an understanding of using port manipulation for your benefit. With a little effort your sketches can be more efficient in terms of speed and memory space, and also allow nifty simultaneous reading of input pins.



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 Google Group – dedicated to the projects and related items on this website. Sign up – it's free, helpful to each other – and we can all learn something.

Bio Latest Posts

John Boxall

Person. Author of http://arduinoworkshop.com Director of http://tronixlabs.com.au Rare updater of http://tronixstuff.com VK3FJBX

Like this:

Loading...

← October 2011 Competition
Review – Agilent Infiniivision MSO-X 3024A Mixed Signal Oscilloscope →

15 Responses to “Tutorial: Arduino Port Manipulation”

1. *Fritz Charleston (@fritzcharleston)* says:
October 22, 2011 at 4:45 pm

If you fine people can wrap your heads around this, then congratulations, your 100% closer to standalone AVR development!

Be wary of using PC6, this also controls the “reset button.” It can be used, you take note.

Also, here is some haxadecimal shorthand to take care of stuff like “00000011.” (Not that anything is wrong with it, just a preference thing.)

Binary | Hex
0000 | 0
0001 | 1
0010 | 2
0011 | 3
0100 | 4
0101 | 5
0110 | 6
0111 | 7
1000 | 8
1001 | 9
1010 | A
1011 | B
1100 | C
1101 | D
1110 | E
1111 | F
(precede with 0x)

As an example from example 43.4:

The code
```
void setup()
{
DDRD = B11111111; // set PORTD (digital 7~0)
// to output
}
```

could be written as:

```
void setup()
{
DDRD = 0xFF; // set PORTD (digital 7~0)
```

```
// to output
}
```

But either way is perfectly acceptable!

Reply

2. *John Boxall* says:
October 23, 2011 at 12:43 pm

Here is the source code for digitalWrite() – quite a lot of code to convert compared against port manipulation:

```
void digitalWrite(uint8_t pin, uint8_t val)
{
uint8_t timer = digitalPinToTimer(pin);
uint8_t bit = digitalPinToBitMask(pin);
uint8_t port = digitalPinToPort(pin);
volatile uint8_t *out;

if (port == NOT_A_PIN) return;

// If the pin that support PWM output, we need to turn it off
// before doing a digital write.
if (timer != NOT_ON_TIMER) turnOffPWM(timer);

out = portOutputRegister(port);

if (val == LOW) {
uint8_t oldSREG = SREG;
cli();
*out &= ~bit;
SREG = oldSREG;
} else {
uint8_t oldSREG = SREG;
cli();
*out |= bit;
SREG = oldSREG;
}
}
```

Reply

3. *Alastair D'Silva* says:
October 23, 2011 at 10:51 pm

Note that you can also twiddle a single pin using the _BV macro, while leaving the others untouched.

```
// pin high
PORTD |= _BV(3);

// pin low
PORTD &= ~_BV(3);
```

These evaluate to a single instruction which takes 2 clocks. Setting the full 8 bits on a port only takes 1 clock (assuming you are assigning a constant).

Reply

- *Ernest Guy* says:
  July 3, 2013 at 4:01 pm

  I am sorry to disagree. On an AVR using avr-gcc these commands each translate to a single assembler operation: sbi / cbi respectively, so everything in one clock cycle. 😀

  Reply
  - *Alastair D'Silva* says:
    July 3, 2013 at 4:54 pm

    You should be sorry 😉 Check the docs: http://www.atmel.com/Images/doc0856.pdf – on any ATMega, they are a single instruction that takes 2 clocks

4. *John Boxall* says:
   October 24, 2011 at 1:24 am

   Excellent question about the faster waveform and square-waves. I took some more measurements by setting the pins high then low five times in a row – and it is now apparent that the extra time in low is the time taken to restart void loop. Please see example 43.1.2 and the following MSO screen shot.
   cheers
   john

   Reply
5. *John Boxall* says:
   November 2, 2011 at 2:08 am

   When I used the TVout library it used digital pins 7 and 9 (http://tronixstuff.com/2011/05/30/tutorial-video-output-from-your-arduino/). Anyhow, I think that there may be a clash there between the library and the port manipulation. Frankly I would look at using a Mega or moving the I/O out to the I2C bus and an MCP23017 (for example).

   Reply
6. *Sebastian* says:
   November 2, 2011 at 7:37 pm

   Hi there.. First, thanks for the nice tutorial!
   but there is a question remaining for me. When I want to see the status of the pins in my serial monitor, what is the difference between using PORTy with in lnprint oder PINy. Essentially, I want to read out the binary numbers for the status of the pins, to execute some action in my code, depending on the status of the pin. What would be the best way to do that.

   Reading out PORTy or PINy?

   Thank you in advance
   Sebastian

   Reply
   - *John Boxall* says:
     November 9, 2011 at 12:13 pm

     Use PINy to read the status, you use PORTy to set the pins' status
     john

     Reply
7. *konturgestaltung* says:
   November 6, 2011 at 7:08 am

Nice tutorials!!

Reply

- *John Boxall* says:
  November 6, 2011 at 8:44 am

  Thank you

  Reply

8. *DTM22* says:
   November 17, 2011 at 7:31 am

   Hi, Nice tutorial! Definetly cleared things up alot! Im sill having trouble undertstanding how to use port manipulation to check the value of a a particular pin though. In my particular case, Im using pins 6 and 7 as INPUTS(DDRD=B00111100;)
   How do I check whether those pins read HIGH or LOW?

   Reply

   - *John Boxall* says:
     November 20, 2011 at 12:04 pm

     Just read them in the same manner as a digital bank.

     Reply

9. *Andy Tallack* says:
   July 6, 2013 at 6:00 am

   Hi

   Thanks – this clarifies some other posts I was reading on port manipulation. However, I've been struggling with PINC – do you have any experience reading the analog ports? They're obviously more than just 1/0 values, and I'm stumped!

   Thanks

   Reply

10. *mike* says:
    March 5, 2015 at 8:24 am

    Nice job on explaining this!!! Thanks

    Reply

Trackbacks/Pingbacks

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.

Visit tronixlabs.com

Helping you make it with Australia's best value for supported hobbyist electronics from adafruit, Altronics, DFRobot, Freetronics, Pololu and more!

Subscribe via email

Receive notifications of new posts by email.

Email Address

Subscribe

Search the site

Search

tronixstuff forum

Why not join our moderated discussion forum?

Arduino Tutorials

Click for Detailed Chapter Index

Chapters 0 1 2 3 4
Chapters 5 6 6a 7 8
Chapters 9 10 11 12 13
Ch. 14 - XBee
Ch. 15 - RFID - RDM-630
Ch. 16 - Ethernet
Ch. 17 - GPS - EM406A
Ch. 18 - RGB matrix - awaiting update

Ch. 19 - GPS - MediaTek 3329
Ch. 20 - I2C bus part I
Ch. 21 - I2C bus part II
Ch. 22 - AREF pin
Ch. 23 - Touch screen
Ch. 24 - Monochrome LCD
Ch. 25 - Analog buttons
Ch. 28 - Colour LCD
Ch. 29 - TFT LCD - coming soon...
Ch. 30 - Arduino + twitter
Ch. 31 - Inbuilt EEPROM
Ch. 32 - Infra-red control
Ch. 33 - Control AC via SMS
Ch. 34 - SPI bus part I
Ch. 35 - Video-out
Ch. 36 - SPI bus part II
Ch. 37 - Timing with millis()
Ch. 38 - Thermal Printer
Ch. 39 - NXP SAA1064
Ch. 40 - Push wheel switches
Ch. 40a - Wheel switches II
Ch. 41 - More digital I/O
Ch. 42 - Numeric keypads
Ch. 43 - Port Manipulation - Uno
Ch. 44 - ATtiny+Arduino
Ch. 45 - Ultrasonic Sensor
Ch. 46 - Analog + buttons II
Ch. 47 - Internet-controlled relays
Ch. 48 - MSGEQ7 Spectrum Analyzer
First look - Arduino Due
Ch. 49 - KTM-S1201 LCD modules
Ch. 50 - ILI9325 colour TFT LCD modules
Ch. 51 - MC14489 LED display driver IC
Ch. 52 - NXP PCF8591 ADC/DAC IC
Ch. 53 - TI ADS1110 16-bit ADC IC
Ch. 54 - NXP PCF8563 RTC
Ch. 56 - MAX7219 LED driver IC
Ch. 57 - TI TLC5940 LED driver IC
Ch. 58 - Serial PCF8574 LCD Backpacks
Ch. 59 - L298 Motor Control
Ch. 60 - DS1307 and DS3231 RTC part I

Australian Electronics!

Buy and support Silicon Chip - Australia's only Electronics Magazine.

Interesting Sites

Amazing Arduino Shield Directory
David L. Jones' eev blog
Silicon Chip magazine Always a great read!
Talking Electronics
Dangerous Prototypes

The Amp Hour podcast
Superhouse.tv High-tech home renovation

Use of our content…