arduino.cc

Arduino - PortManipulation

Reference Language | Libraries | Comparison | Changes

Port registers allow for lower-level and faster manipulation of the i/o pins of the microcontroller on an Arduino board. The chips used on the Arduino board (the ATmega8 and ATmega168) have three ports:

- B (digital pin 8 to 13)
- C (analog input pins)
- D (digital pins 0 to 7)

Each port is controlled by three registers, which are also defined variables in the arduino language. The DDR register, determines whether the pin is an INPUT or OUTPUT. The PORT register controls whether the pin is HIGH or LOW, and the PIN register reads the state of INPUT pins set to input with pinMode(). The maps of the ATmega8 and ATmega168 chips show the ports. The newer Atmega328p chip follows the pinout of the Atmega168 exactly.

DDR and PORT registers may be both written to, and read. PIN registers correspond to the state of inputs and may only be read.

PORTD maps to Arduino digital pins 0 to 7

DDRD - The Port D Data Direction Register - read/write

PORTD - The Port D Data Register - read/write

PIND - The Port D Input Pins Register - read only

PORTB maps to Arduino digital pins 8 to 13 The two high bits (6 & 7) map to the crystal pins and are not usable

DDRB - The Port B Data Direction Register - read/write

PORTB - The Port B Data Register - read/write

PINB - The Port B Input Pins Register - read only

PORTC maps to Arduino analog pins 0 to 5. Pins 6 & 7 are only accessible on the Arduino Mini

DDRC - The Port C Data Direction Register - read/write

PORTC - The Port C Data Register - read/write

PINC - The Port C Input Pins Register - read only

Each bit of these registers corresponds to a single pin; e.g. the low bit of DDRB, PORTB, and PINB refers to pin PB0 (digital pin 8). For a complete mapping of Arduino pin numbers to ports and bits, see the diagram for your chip: ATmega8, ATmega168. (Note that some bits of a port may be used for things other than i/o; be careful not to change the values of the register bits corresponding to them.)

Examples

Referring to the pin map above, the PortD registers control Arduino digital pins 0 to 7.

You should note, however, that pins 0 & 1 are used for serial communications for programming and debugging the Arduino, so changing these pins should usually be avoided unless needed for serial input or output functions. Be aware that this can interfere with program download or debugging.

DDRD is the direction register for Port D (Arduino digital pins 0-7). The bits in this register control whether the pins in PORTD are configured as inputs or outputs so, for example:

```
DDRD = B11111110;  // sets Arduino pins 1 to 7
as outputs, pin 0 as input
DDRD = DDRD | B11111100;  // this is safer as it
sets pins 2 to 7 as outputs
                          // without changing
the value of pins 0 & 1, which are RX & TX

//See the bitwise operators reference pages and
```
The Bitmath Tutorial in the Playground

PORTD is the register for the state of the outputs. For example;

```
PORTD = B10101000;  // sets digital pins 7,5,3
HIGH
```

You will only see 5 volts on these pins however if the pins have been set as outputs using the DDRD register or with pinMode().

PIND is the input register variable It will read all of the digital input pins at the same time.

Why use port manipulation?

From The Bitmath Tutorial

Generally speaking, doing this sort of thing is **not** a good idea. Why not? Here are a few reasons:

- The code is much more difficult for you to debug and maintain, and is a lot harder for other people to understand. It only takes a few microseconds for the processor to execute code, but it might take hours for you to figure out why it isn't working right and fix it! Your time is valuable, right? But the computer's time is very cheap, measured in the cost of the electricity you feed it. Usually it is much better to write code the most obvious way.

- The code is less portable. If you use digitalRead() and digitalWrite(), it is much easier to write code that will run on all of the Atmel microcontrollers, whereas the control and port registers can be different on each kind of microcontroller.

- It is a lot easier to cause unintentional malfunctions with direct port access. Notice how the line DDRD = B11111110; above mentions that it must leave pin 0 as an input pin. Pin 0 is the receive line (RX) on the serial port. It would be very easy to accidentally cause your serial port to stop working by changing pin 0 into an output pin! Now that would be very confusing when you suddenly are unable to receive serial data, wouldn't it?

So you might be saying to yourself, great, why would I ever want to use this stuff then? Here are some of the positive aspects of direct port access:

- You may need to be able to turn pins on and off very quickly, meaning within fractions of a microsecond. If you look at the source code in lib/targets/arduino/wiring.c, you will see that digitalRead() and digitalWrite() are each about a dozen or so lines of code, which get compiled into quite a few machine instructions. Each machine instruction requires one clock cycle at 16MHz, which can add up in time-sensitive applications. Direct port access can do the same job in a lot fewer clock cycles.

- Sometimes you might need to set multiple output pins at exactly the same time. Calling digitalWrite(10,HIGH); followed by digitalWrite(11,HIGH); will cause pin 10 to go HIGH several microseconds before pin 11, which may confuse certain time-sensitive external digital circuits you have hooked up. Alternatively, you could set both pins high at exactly the same moment in time using PORTB |= B1100;

- If you are running low on program memory, you can use these tricks to make your code smaller. It requires a lot fewer bytes of compiled code to simultaneously write a bunch of hardware pins simultaneously via the port registers than it would using a for loop to set each pin separately. In some cases, this might make the difference between your program fitting in flash memory or not!

See

- Pin Mapping of Atmega 168/328

Reference Home

Corrections, suggestions, and new documentation should be posted to the Forum.