# Manipulación directa de puertos. Usando los puertos digitales.

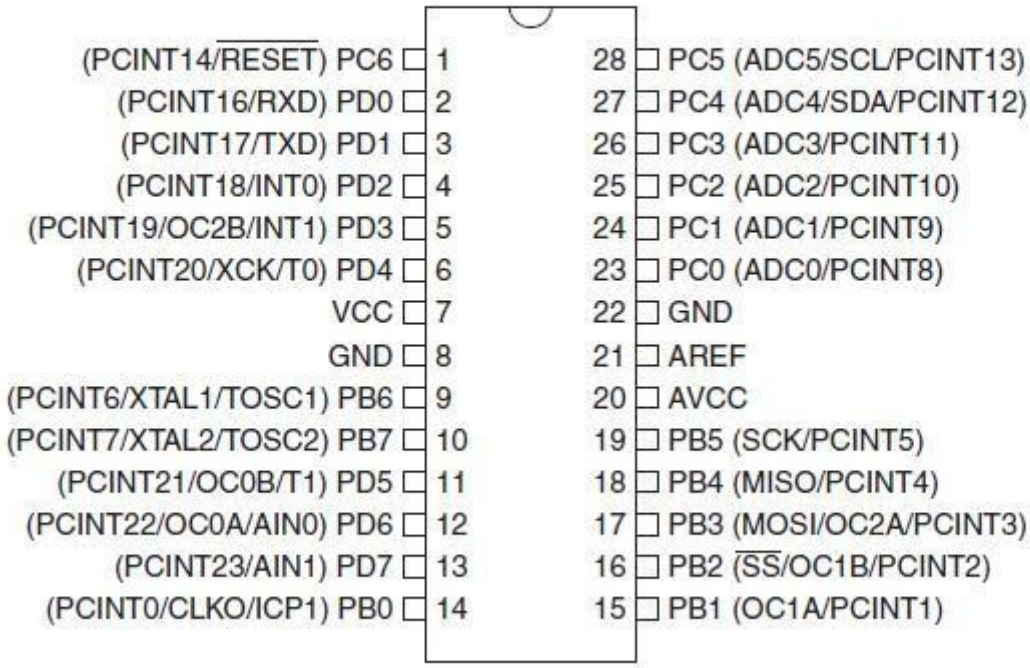
(Segunda traducción al castellano de los artículos escritos aqui: http://hekilledmywire.wordpress.com/2011/02/23/)

Hola amigos lectores, hoy voy a escribir un poco acerca de los pines digitales y cómo leer y escribir valores digitales para esto, es un tema sencillo pero de gran importancia, porque casi todo lo que nuestros microcontroladores hacen es utilizar sus entradas y salidas para hablar con leds, los drivers del motores, LCD, registros de desplazamiento, para leer los datos de los sensores digitales y un monton de otros usos, así que vamos a empezar a leer acerca de cómo hacerlo.

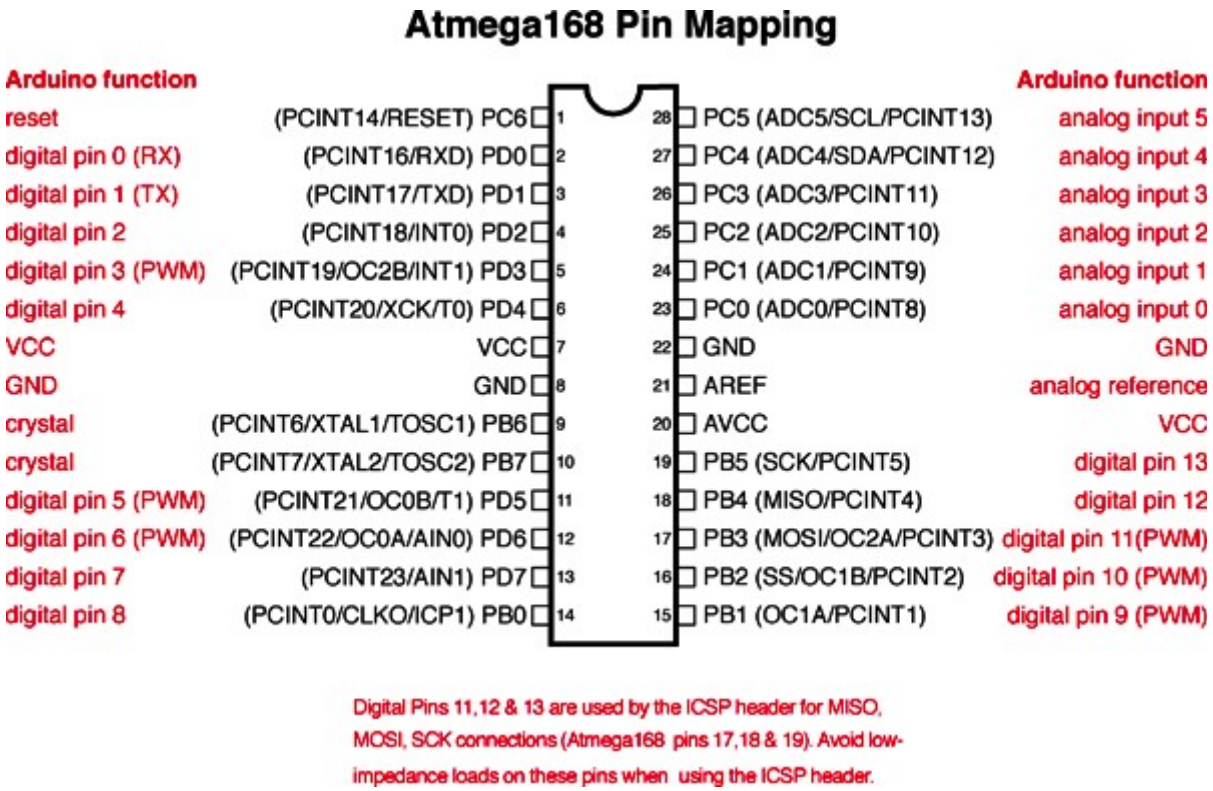
Nuestro microcontrolador, el ATmega328p tiene registros, estos registros están relacionados con los puertos de entrada/salida, cada puerto tiene un nombre específico y sus registros asociados, de hecho, nuestro Atmega tiene el puerto B, C y D, y cada puerto un diferente número de pines (Esta es una restricción del paquete de 28 pines PDIP y no desde el microcontrolador, ya que un PDIP 40 pines, por ejemplo, tiene 4 puertos con los 8 bits cada uno), el único puerto que tiene el total de sus 8 pines de entradas/salidas es PORTD.

Como usted ya sabe, cada pin puede tener múltiples funciones, como la generación de PWM, o las capacidades de ADC, los pines 6 y 7 del PORTB son los pines de entrada para el oscilador de cristal, y pin 6 del PORTC le corresponde al botón de reinicio.

En esta imagen se puede ver todas las funciones alternativas que cada pin puede tener en el chip en cuestión que es el ATmega328P.




Y aquí está la asignación entre los nombres de los puertos Arduino y su verdadero nombre:




Entonces, ¿cómo podemos interactuar con nuestros pines digitales?

Bueno, para empezar hay un registro dedicado para cada puerto que define si cada pin es una entrada o una salida, que es el registro de DDRX, donde x es la letra del puerto que queremos configurar, en el caso de la Arduino hay DDRB, DDRC y DDRD. Como toda variable lógica, cada bit en los registros DDRX puede ser 1 ó 0, es que poner un bit específicos de DDRX a 1 configura el pin como salida y ponerla a 0 configura el pin como una entrada, vamos a ver un pequeño ejemplo que configura pines 0,1,2,3 como entradas digitales y los pines 4,5,6,7 como salidas digitales:

```
DDRD = 0b11110000;
```

Y quiero todos los pin como salidas:

```
DDRD = 0b11111111;
```

Y si usted necesita todas las salidas? Compruébalo por ti mismo, o espera unos días que voy a liberar mi tutorial de de manipulación de bits.

Debe haber algún tipo de atención cuando se utiliza PORTD y Serial / USART porque los pines 0 y 1 del PORTD son los utilizados por la USART y si pones estos dos como entradas o salidas, la USART sera incapaz de leer o escribir datos en los pines.

Ya podemos decir al Atmega cómo serán utilizados sus pines, pero queremos saber cómo leer y escribir datos en dichos pines, de modo que para escribir datos en un determinado puerto, se utiliza el registro PORTx, éste es fácil de recordar, donde x es el nombre del puerto, y después de la configuración de un pin como salida es sólo una cuestión de poner 0 o 1 en el registro PORTx para controlar que el pin de este en estado alta o baja, respectivamente, vamos a ver algo de código para esto:

```
DDRD = 0b11111111;    // Todos los pines de PORTD son salidas.
PORTD = 0b11111111;    // Todos los pines de PORTD están en estado alto.
```

```
DDRD = 0b11111111;    // Todos los pines de PORTD son salidas.
PORTD = 0b00000000;    // Todos los pines de PORTD están estado bajo.
```

¿Y qué tal un patrón de encendido, apagado, encendido, ..?

```
DDRD = 0b11111111;    // Todos los pines de PORTD son salidas.
PORTD = 0b10101010;    // Un patrón de encendido, apagado, encendido, etc.
```

Ahora, lo único que queda es leer en el pin para poder leer los datos de los sensores o incluso cuando se pulsa un botón, para leer el estado de un pin digital configurado como entrada, vamos a utilizar un tercer registro llamado PINX, donde de nuevo x es el nombre del puerto donde se encuentra el pin, así que primero con DDRX decimos a la micro-controlador que queremos algunos pines como entradas digitales, y luego usando PINX leemos sus valores, parece fácil, por lo que permite profundizar en el código:

```
DDRD = 0b00000000;   // Todos los pines del PORTD son entradas
char my_var = 0;     // Creamos una variable para guardar la información leída en PORTD
my_var = PIND;       // Lee PORTD y pone la información en la variable
```

Es tan fácil como puede ser, y en comparación con el digitalWrite de Arduino y las funciones de leer, con acceso directo al puerto puede ahorrar espacio en el la memoria flash y también puede ganar mucha velocidad, porque las funciones Arduino puede tomar más de 40 ciclos de reloj para leer o escribir un solo bit en un puerto y al mismo tiempo para leer otra de un solo bit el código es bastante complejo con un montón de líneas que ocupan por lo menos unos 40 bytes, que podría ser un pequeño ahorro en flash, pero es un gran paso para acelerar cualquier programa, pero estos son fáciles de usar por las personas que no entienden mucho acerca de la programación/microcontroladores, por lo que cada aplicación tiene sus ventajas y desventajas, pero le permite continuar.

Es un poco raro que usted necesite leer o escribir en un puerto completo en cada momento, por ejemplo, si usted quiere encender un LED, o leer un botón sólo tendrá que utilizar un pin, y escribir todos los bits uno a uno cada vez que queremos cambiar un valor en un puerto, es una tarea aburrida, pero librería C de AVR tiene algunas pocas palabras definidas como Px0..7, donde x es de nuevo el puerto que desea utilizar y 0..7 es el valor del pin individual de dicho puerto, por lo que para iluminar un LED debemos hacer algo como esto:

```
DDRD = (1<<PD2);     // Configura el pin 2 de PORTD como salida.
PORTD = (1<<PD2);    // El pin 2 de PORTD tiene ahora un valor lógico 1.
```

O para leer el estado de un botón:

```
DDRD = 0b11111101;          /* Configura el pin 1 de PORTD como estrada y el resto como
                             * salida. */
char my_var = 0;            /* Crea una variable para guardar la información leída
                             * en PORTD. */
my_var = (PIND & (1<<PD1)); /* Le el pin 1 de PORTD y lo coloca en la variable. */
```

También puede utilizar la macro Px0..7 varias veces en una misma instrucción, por ejemplo, en este código, se ejectutara algo de código sólo si se pulsa dos botones al mismo tiempo:

```
DDRD = 0b11111100;         // Los pines 0 y 1 de PORTD son entradas, y el resto salidas.
if(PIND & ((1<<PD0) | (1<<PD1))) {
       /* Algún código dentro del if() que se ejecutara solo si los dos botones se
        * encuentran activados. */
}
```

Creo que vas a encontrar el punto, pero el tutorial de manipulación de bits ayudará un poco sobre esto temas.

Hay todavía algunas cosas más que podemos hacer con nuestra entrada y salida de los pines/puertos que sera muy útil para usar las interfaces I2C y por ejemplo, para utilizar los botones, estoy hablando de los pull-ups que nuestros microcontroladores tienen en su interior y te mostraré cómo se pueden habilitar y por qué debes utilizarlos cuando se utilizan los pulsadores.

Cuando usted tiene un botón que puede tener dos estados, uno es desconectado, y cuando usted lo presiona hará una conexión entre los pines del microcontrolador y permite por ejemplo, conectarse a tierra, pero cuando se desconecta, no hay nada que fuerce un valor estable en el pin de entrada, para el ojo inexperto es aceptable suponer que el pin va a leer un 1, porque cuando se pulsa el botón lee 0, pero la realidad es que el pin puede leer 1 ó 0 ya que el pin es muy sensible al ruido electromagnético, como una pequeña antena, por lo que puede resolver este problema de dos maneras similares, una es para conectar una resistencia de 10Kohms o más entre el Vcc (+5 v) y el pin de entrada, o sólo tienes que ahorrar algunas

monedas de un centavo con el uso de los pull-ups que nuestros micro-controlador que tienen integrados, también hace que nuestros circuitos un poco más simple y esto también es una buena idea.

Para habilitar las resistencias pull-ups tenemos que hacer algo que puede resultar un poco extraño a primera vista, porque no existe un registro dedicado para activar o desactivar el pull-ups, estos son activados o desactivados escribiendo, 1 o 0 respectivamente en el registro PORTx cuando el registro DDRX se configuran como entradas, vamos a ver algo de código para clarificar esto:

```
DDRD = 0b00000000;   // Todos los pines de PORTD son entradas.
PORTD = 0b00001111;  /* Habilito las Pull-ups de los pines 0,1,2 y 3 y lo deshabilito
                      * en los pines 4,5,6 y 7.
char my_var = 0;     // Creo una variable para guardar la información leída en PORTD.
my_var = PIND;       // Leo PORTD y coloco la información en la variable.
```

Si ejecuta este código, sin tener nada conectado a PORTD, los cuatro bits mas altos de la variable my_var puede ser 0 ó 1, cualquier combinación posible de ellos porque son flotantes (actúan como pequeñas antenas), pero los cuatro bits más bajos leerá todos un 1 debido a que el pull-ups imponen una señal de 5V débil que se lee como un valor lógico 1.

En un sentido básico esto es todo lo que necesita saber para dominar la manipulación directa de los puertos, pero el tutorial de manipulación de bits le enseñará cosas mas ingeniosas como las máscara de bits, las operaciones AND, OR, NOT y XOR y cómo configurar y limpiar los bits en un registro y algunos buenos trucos con los operaciones de desplazamiento derecho e izquierdo, todas cosas bueno a saber, ya que puede acelerar su programa y son muy útiles cuando se utilizan los puertos digitales.

Permitanme ahora hacer un pequeño programa de prueba que pueda hacer uso de las entradas y salidas digitales, así como el pull-ups, porque este tutorial a sido muy teórico y es agradable llegar al final haciendo parpadear algunos leds!.

Este es el pseudo-código de nuestro programa, su intención es la de leer un interruptor y cada vez que el interruptor es leído, se conmutara el estado de un LED, por lo que cuando se presione el interruptor el LED se iluminara, pulse de nuevo y el led se apagara, y de nuevo desde el principio, esto se podría hacer usando varias condicionales if(), pero se puede hacer en una sola línea usando el poderoso operador XOR(^)

```
Main {
      Configuración del puerto, en este código de ejemplo usar PORTD.
      bucle infinito {
            Leo el valor del botón
            Si el led esta encendido y el botón ==1: Apaga el led.
            Si el led esta apagado y el botón==1 enciende el led.
      }
}
```

Sólo una cosa más (sé que siempre voy un poco off-topic, pero este es importante) cuando se utiliza un botón conectado a una entrada digital, debemos ser conscientes de que un botón no da una buena y transición limpia entre 0 a 1 o de 1 a 0, pero en su lugar la señal puede tener problemas de rebote, esto es debido a las propiedades mecánicas del botón y no un defecto de diseño. Como el botón tiene una pequeño conector en el interior, y cuando presiona el conector se cierra y se cierra el circuito entre su entrada y salida, pero esta pata suele a oscilar un poco hasta que quede firme en su parada, lo que debemos tener cuidado de esto, y como siempre hay dos maneras, mediante un condensador pequeño cerca de la botón para el rebote del valor, o que esta eliminación de rebotes en el mismo código, que es más fácil de hacer cuando tenemos un montón de botones y de nuevo es más barato que la adición de una gran cantidad de componentes a nuestro circuito.

Si buscas en Google el termino “button debouncing” encontrara una gran variedad de formas para evitar el rebote de los botones usando código, la forma que voy a utilizar aquí es la más simple de todos, es sólo insertar un pequeño retraso entre las consecutivas lecturas de un botón, esto es de por supuesto, un método de bloqueo, porque nuestro microcontrolador se detendrá por algunos milisegundos, hay otras formas mas inteligentes que usar temporizadores, pero para proyectos de 2 ó 3 botones que no requieren una sincronización super precisa se trata de una método de uso común.

Para hacer este retraso voy a utilizar el incorporado en las rutinas de retraso proporcionado por AVR lib-c.

Así que vamos a empezar a programar, todo el código es muy sencillo si usted entendió todo lo que fue escrito anteriormente.

```
#include <avr/io.h>         /* Esta cabecera incluye las definiciones para todas las
                             * direcciones do los registros y otras cosas, casi siempre
                             * deberá incluirse. */

#define F_CPU 16000000UL    /* F_CPU le dice al compilador que usas un cristal de 16Mhz
                             * y asi puede generar los delay exactos, entonces debe ser
                             * declarado. */

#include <util/delay.h>     /* Contiene las funciones de delay que pueden generar delays
                             * exactos de ms o uS. */

uint8_t readButton(void);   // Declaración de la función readButton.

int main(void) {
       DDRD &= ~(1<<PD2);   // Configura el pin 2 del PORTD como una entrada.
       PORTD |= (1<<PD2);   // Activa el pull-ups en el pin 2 del PORTD.
       DDRB |= (1<<PB5);    /* Configura el pin 5 del PORTB como salida, este es el pin
                             * digital 13 en la placa Arduino que tiene el led
                             * integrado. */

       while(1) {           // Infinite loop
              if(readButton()==1) {       // Verifica el estado del botón.
                     PORTB ^=(1<<PB5);    /* Esta es la linea de código que mencione
                                           * anteriormente para intercambiar el estado
                                           * del led. */
              }
              _delay_ms(250);              // Delay entre consecutivas lecturas.
       }
}

uint8_t readButton(void) {
       if((PIND & (1<<PD2)) == 0) { // Si el botón esta presionado.
              _delay_ms(25);        // Retardo de entrada para el valor leído.
       }
       if((PIND & (1<<PD2)) == 0) { // Verifica que la la lectura sea correcta.
              Return 1;  // Si todavía es 0 es porque si teníamos pulsado el botón
       }
       else {
              return 0;  // Si el valor cambio la lectura es incorrecta.
       }
}
```

Creo que no hay necesidad de mostrar cómo crear el proyecto en AvrStudio en este punto, pero si usted tiene cualquier duda consulte los otros tutoriales o deje un comentario, y no se olvide de presionar el botón de reinicio cuando se quiere subir su código.

Tampoco tiene un esquema del circuito, pero puedo conseguir uno si alguien lo solicite, o cuando tengo el tiempo para hacer uno, el LED usado en es el LED incorporado que Arduino como en el pin digital 13, y el botón es también es fácil, se conecta una de las patas del botón situado a la pin PORTD2, que es el pin digital 2 de arduino y la otra pierna del interruptor debe conectarse a masa.

Hay aquí un pequeño video que muestra que este código en acción, el led no siempre cambia, pero esto es "normal" porque mi botón como ya se ha sufrido un poco en su vida y también puede pulsar el botón cuando el retraso de los 250 ms se está ejecutando, pero funciona y aquí está la prueba:

http://www.youtube.com/watch?feature=player_embedded&v=kuIz0EdIrn0

He hecho algunas correcciones al texto, y aquí está el código en un proyecto de AVR Studio listo para compilar y cargar:

http://code.google.com/p/avr-tutorials/downloads/detail?name=port.zip

Gracias por leer y no te olvides de comentar cualquier cosa que desee, y una buena programación!