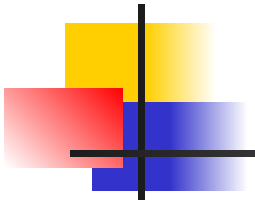
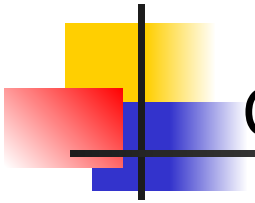


Primer Laboratorio

- C embebido
- Puertos
- Registros
- Polling
- Debouncing
- Temporizado y retardo
- Librería Serial



C embebido

Para los que tiene experiencia programando en C, verán que la estructura de un programa es exactamente la misma para un microcontrolador: inclusión de librerías, definición de constantes, variables, estructuras, etc., y la función principal main

C embebido

```
// INCLUSION DE LIBRERIAS
// DEFINICION DE VARIABLES, CONSTANTES, ESTRUCTURAS, etc.

int main(){
    // INICIALIZACIÓN DE PUERTOS VARIABLES Y PERIFERICOS
    // CICLO INFINITO
    while(1){
        // CODIGO A EJECUTAR INFINITAMENTE
    }
    // OTRA FORMA DE CICLO INFINITO
    for(;;){
        // CODIGO A EJECUTAR INFINITAMENTE
    }
}
```

C embebido

C

- Generalmente es usado en computadoras de escritorio.
- No necesito conocer el hardware subyacente.
- C usa los recursos de la computadora como memoria, OS, etc.
- Un programa es independiente del hardware.

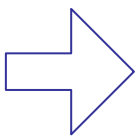
C embebido

- Para aplicaciones basadas en microcontroladores
- Uno necesita conocer el hardware subyacente, los nombres de los registros, etc
- Es usado con recursos limitados, tal como RAM,ROM I/Os sobre un procesador embebido
- Un programa es dependiente del hardware

C embebido

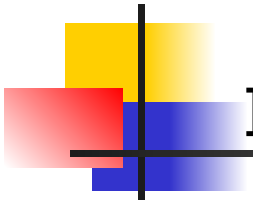
ATMEL STUDIO

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{   DDRB |= (1<< PB5);
    while (1)
    {   PORTB |= (1<<PB5);
        _delay_ms(1000);
        PORTB &= ~(1<<PB5);
        _delay_ms(1000);
    }
}
```



ARDUINO IDE

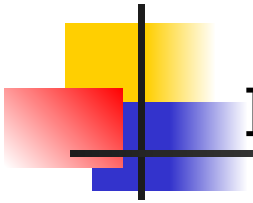
```
void setup() {
   pinMode(13, OUTPUT);
}
 void loop() {
   digitalWrite(13, HIGH);
   delay(1000);
   digitalWrite(13, LOW);
   delay(1000);
 }
```



I/O Devices

Un dispositivo INPUT envía información a un sistema para su procesamiento y un dispositivo OUTPUT reproduce o muestra los resultados de un procesamiento.

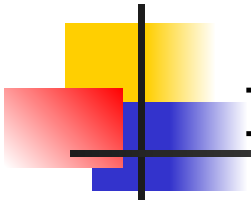
Input devices *only* allow for input of data and output devices *only* receive the output of data from another device



I/O Devices

Input devices:

- pulsador
- key pad
- ldr (sensor de luminosidad)
- joystick
- lm35 (sensor de temperatura)



I/O Devices

Output devices:

- led
- buzzer
- motor
- solenoide
- speaker
- relay

Puertos

El uso de los pines de los microcontroladores como entradas y salidas digitales.

Los pines normalmente vienen agrupados en grupos de 8 pines, a cada uno de estos grupos se les llama **puertos**, algunos puertos pueden tener menos de 8 pines

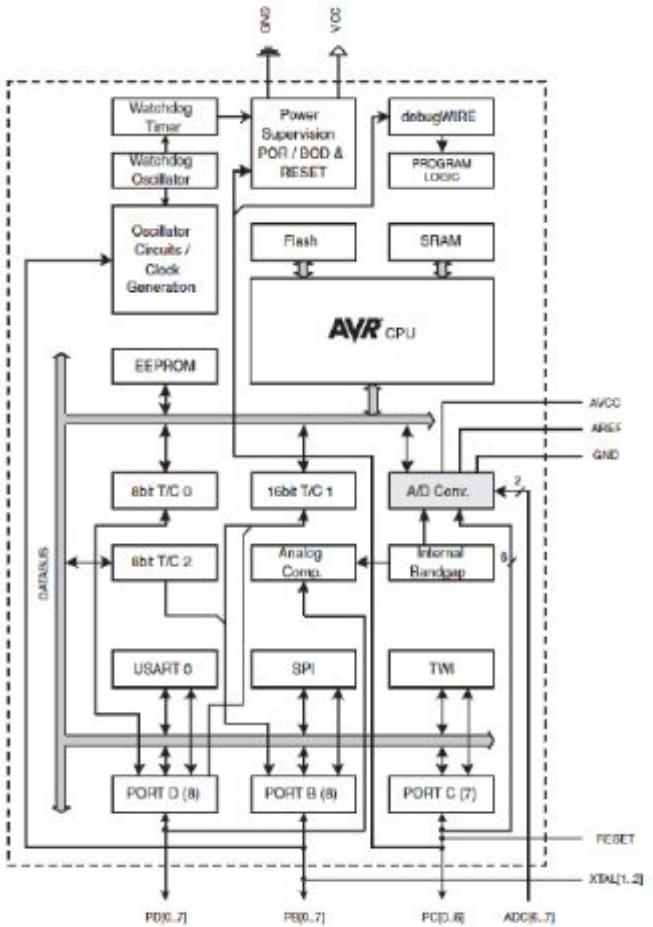
Para más detalles sobre cómo están mapeados los números de los pines de Arduino a los puertos y los bits observa la siguiente imagen:

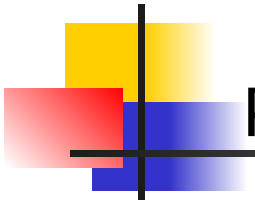
Puertos

ATMEGA328P-PU Chip to Arduino Pin Mapping

Arduino function					Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

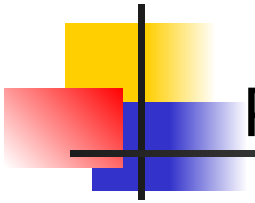
Digital Pins 11,12 & 13 are used by the ICSP header for MISO, MOSI, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.





Puertos

Cada uno de los **pines** del AVR a ser utilizados como entradas y salidas digitales tienen un **nombre propio relacionado con el puerto al que pertenecen** y al número de orden del bit con el cual se programará, por ejemplo para el puerto D se tienen los pines PD0, PD1, PD2, PD3, PD4, PD5, PD6, PD7.

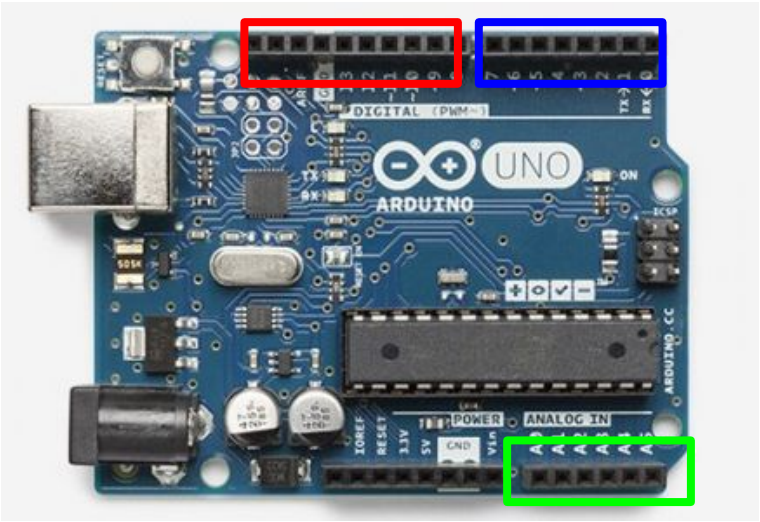


Puertos

Los puertos permiten manipular los pines de un microcontrolador a bajo nivel y de manera más rápida. El chip usado en la placa Arduino (ATmega328p) posee tres puertos:

- B (pines digitales del 8 al 13)
- C (entradas analógicas)
- D (pines digitales del 0 al 7)

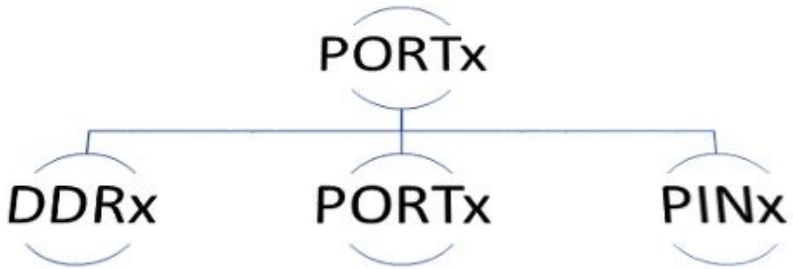
Puertos



- **PORT B (pines digitales del 8 al 13)**
- **PORT D (pines digitales del 0 al 7)**
- **PORT C (entradas analógicas)**

Registros de control de puerto

La forma de manipular un puerto es mediante DDRx, PORTx y PINx , donde x puede ser B,C o D.



Registros de control de puerto

DDRx (Data Direction Register x): es un registro para setear como entrada o como salida los pines del puerto x.

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Este registro lo podemos comparar en el entorno Arduino con pinMode();, ya que este registro se encarga de manipular la dirección de cada pin del puerto, esto significa que podemos indicar a cada pin del puerto x, que trabajen como escritura o lectura, salida o entrada, o como en Arduino pinMode(3,OUTPUT);

Registros de control de puerto

PORTx (Port Output Register x) : escribe los valores (0 ó 1) a los pines del puerto x en modo salida

Bit	7	6	5	4	3	2	1	0	
	PORTB 7	PORTB 6	PORTB 5	PORTB 4	PORTB 3	PORTB 2	PORTB 1	PORTB 0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

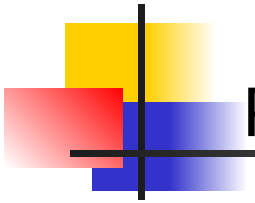
Cuando el DDR ha sido configurado como salida, si en el registro PORT guardamos un uno lógico, el pin del puerto sacará un uno (HIGH) por el puerto de salida. Sin embargo si en el registro PORT guardamos un cero lógico, el pin del puerto sacará un cero (LOW) por el puerto de salida. Este registro en Arduino tendría su similar con la función digitalWrite(nºpin, estado)

Registros de control de puerto

PINx (Port Input Register x): lee los datos de los pines del puerto x en modo entrada

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Este registro en Arduino tendría su similar con la función digitalRead(nºpin), excepto que esta función sólo lee un pin.



Registros de control de puerto

Estos son los tres grandes registros de manipulación de puertos GPI/O a grandes rasgos para manejar las entradas y salidas del microcontrolador. Para manipularnos es conveniente trabajar con el uso de las operaciones lógicas digitales básicas las cuales son AND ("&"), OR ("|") y NOT ("~"), XOR (^) y las operaciones shitfLeft y shiftRight "<< y >>"

Ejemplo1

Arduino

```
boolean estadoPin;

void setup()
{
 Serial.begin(9600);
 pinMode(7,INPUT);
}
void loop()
{
  estadoPin = digitalRead(7);
  Serial.println(estadoPin,BIN);
  delay(500);
}
```

Avr en Arduino

```
boolean estadoPin;

void setup()
{
 Serial.begin(9600);
 DDRD &= (0<<7);
}
void loop()
{
  estadoPin = PIND7;
  Serial.println(estadoPin,BIN);
  delay(500);

}
```

Ejemplo2

Avr en Arduino

```
void setup()
{
 DDRD |= (1<<2);
}
void loop()
{ PORTD |=  (1<<2);//
  delay(500);
  PORTD &= ~(1<<2);//
  delay(500);
}
```

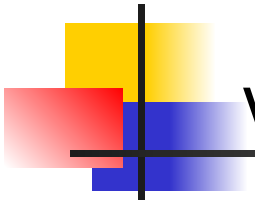
Arduino

```
void setup()
{
 pinMode(2,OUTPUT);
}
void loop()
{
  digitalWrite(2,HIGH);
  delay(500);
  digitalWrite(2,LOW);
  delay(500);
}
```

Ejemplo2

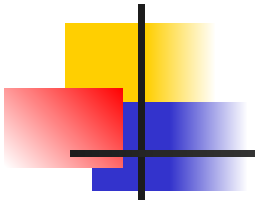
Avr en Arduino

```
void setup()
{ DDRD |= (1<<2);       //Pin 2 del puerto D declarado como salida
}
void loop()
{ PORTD |=  (1<<2);//
  portd =  00000000;
  (or) + 00000100;  // sumamos al registro anterior (1<<2)
       = 00000100;  // resultado pin 2 esta en estado HIGH
  delay(500);
  PORTD &= ~(1<<2);//
  portd =  00000100;
   (and)* 00000000;// resultado del ~(1<<2)
        = 00000000;//obtenemos del And que el pin 2 vuelve a un estado LOW

}
```



Ventajas

- Permite cambiar los estados de pines muy rápido.
- Algunas veces necesitar cambiar muchos pines exactamente al mismo tiempo.
- Si estas quedando sin memoria, puede usar estas operaciones para que el código sea más pequeño.



Desventajas

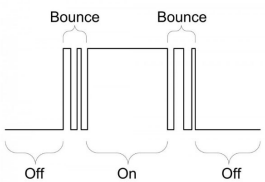
- El código puede ser más dificil de depurar y mantener.
- Se puede causar un mal funcionamiento no intencionado usando el acceso directo a los puertos.
- Es menos portable; los registros de control y puertos pueden ser distintos en diferentes microncontroladores.

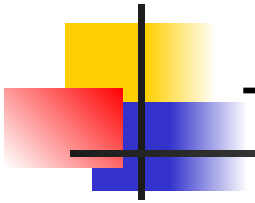
Técnica debouncing

Los rebotes son las falsas pulsaciones que se producen al hacer falsos contactos en el interruptor.

Para ello esperar un tiempo llamado “debounceDelay” para comprobar que el cambio de estado se mantiene y no son rebotes.

Para ello esperar un tiempo llamado debounceDelay para





Técnica debouncing

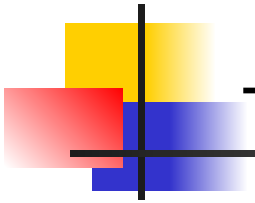
Comparar los códigos de button.c y flanco.c

- El button.c se enciende o apaga en función del nivel del voltaje.
- El flanco.c se enciende o apaga cuando el valor de dicho pin2 pasa de un valor lógico alto a un valor lógico bajo ó viceversa.

Temporizado

Escribir una rutina que permita cambiar el estado de un led y utilizar esa función permitiendo que el led trabaje a diferentes frecuencias.

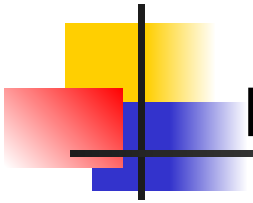
```
int main() {
//Inicializar el pin del pulsador como entrada DDRD &= ~(1<<DDD2);
//Se inicializa el pin digital como salida.  DDRB  |= (1<<DDB5);

int i=0;
while(1) {
//Leer el estado del pulsador
//Si no está presionado el pulsador (se detectó un keyup)...
…
...
if (!buttonState)
        {i = (i+1) % 6;
        }
        //Si hubo cambios válidos actualizo el valor de lectura anterior
        lastButtonState = buttonState;
        }
  modoLed(i);
}
}
```



Temporizado

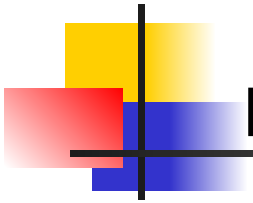
Analizar las ventajas y desventajas de un sistema de polling vs interrupciones.

Analizar ventajas y desventajas retardo por software vs retardos por hardware. (Timers ?)



Librerías de Arduino

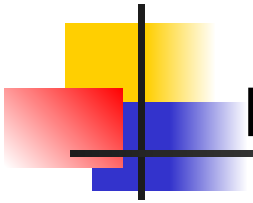
- Abrir el ejemplo de buttonArduino.cpp en Atmel Studio.
- Lo que se busca con esta actividad es como utilizar el código del IDE de Arduino en Atmel Studio, para ello deberá configurar en las propiedades del proyecto de Atmel con los pasos del punto 3.
- Qué hacen las funciones pinMode, digitalRead y digitalWrite en código AVR ?.



Librería serial

La comunicación serial en los pines TX / RX usa niveles lógicos TTL (5V o 3.3V dependiendo de la placa).

La serial se usa para la comunicación entre la placa Arduino y la computadora u otros dispositivos. Todas las placas Arduino tienen al menos un puerto serie (también conocido como UART o USART). Utilizan los pines digitales 0 (RX) y 1 (TX). Por lo tanto, si usa estas funciones, no puede usar también los pines 0 y 1 para entrada o salida digital.



Librería serial

https://www.arduino.cc/reference/en/language/functions/communication/serial/

Serial.begin (int speed): Inicializa el puerto serial. Es utilizado en el bloque setup de Arduino. speed corresponde a la velocidad de la conexión serie.

int Serial.available(): Devuelve el número de carácteres (bytes) disponibles para leer.

Serial.read(): devuelve el primer carácter disponible. (-1) si no hay nada.

Serial.print(): escribe los datos en la salida serie en formato ascii.

Serial.write(): escribe los datos en la salida serie en formato binarios. Los datos se envian como byte o como una serie de bytes. Para enviar los caracteres que representan los dígitos de un número, utilice la función print () en su lugar.

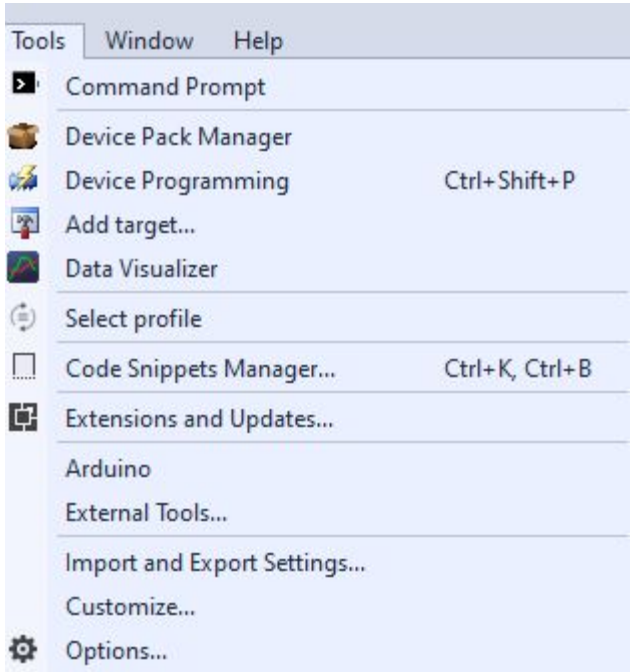
Librería serial

- Deberán utilizar el ejercicio de la actividad 4 y adaptarla leyendo del puerto serie, *números o caracteres* que crean convenientes y esto permita cambiar a diferentes estados.

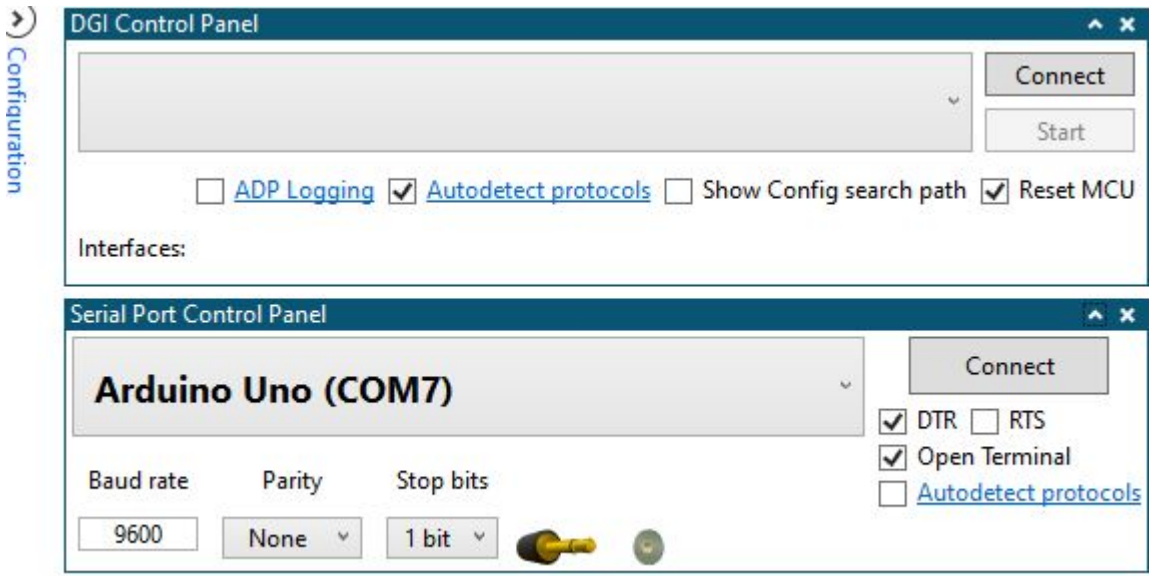
```
loop() {
 // read the sensor:
 if (Serial.available() > 0) {
      int cont = Serial.read();
      ….
      …..
}
  switch (cont)   {
           …
           …  case '5':
                Serial.println("Medio segundo encendido y un
segundo apagado")

                ...
```

Librería serial

Para conectarse a la serial mediante Atmel Studio



Librería serial



Librería serial