

Programación de Sistemas Embebidos

Clase – El primer programa embebido

Programa analítico de la asignatura

UNIDAD 2: Herramientas de desarrollo:

Toolchains (compilador, ensamblador, vinculador) .

Compilación cruzada (cross compiler). Debugger. Analizador de archivos objetos y ejecutables (disassembly). Automatización del ciclo de compilación (make).

UNIDAD 3: E/S de bajo nivel:

E/S paralela: GPIO.

UNIDAD 4: Programación de bajo nivel

Lenguaje C. Programación sobre hardware sin sistema (baremetal).

Temario

- **Ciclo de compilación de un hello world para AVR**
- **Make y makefiles**
- **Avrdude**
- **Entrada y Salida digital : GPIO en AVR**

Ciclo de compilación en lenguaje C

- **Etapa de desarrollo:**

Se desarrolla un proyecto de software en lenguaje C.

En un editor de texto plano se crean (se programan) los archivos fuentes (.c y .h de cabeceras) que componen toda la aplicación.

- **Etapa de compilación:**

Cada archivo fuente se compila por separado (se traducen) con el compilador de C. Por cada archivo fuente compilado se genera un archivo código objeto.

- **Etapa de vinculación:**

Se vinculan todos los archivos objetos y tal vez bibliotecas para formar un archivo ejecutable.

- **Etapa de transferencia (unicamente para sistemas embebidos):**

Se traduce el archivo ejecutable al formato aceptado por el sistema embebido, y se transfiere la imagen (firmware) al dispositivo.

- **Ejecución:**

Se ejecuta la aplicación (en sistemas embebidos sin S.O. la aplicación se ejecuta luego de un reset).

El primer programa embebido (AVR) – E/S bajo nivel

Pasos:

- 1- tener la hoja de datos (manual) del microprocesador / microcontrolador
- 2- tener el esquemático de la placa (PCB)
- 3- leer con atención si la E/S es mapeada en memoria o aislada
- 4- tener instalado el toolchain para el target
- 5- conocer como compilar (cross-compilar) un programa para la arquitectura destino (target)
- 6- realizar un programa utilizando punteros para operar sobre los registros de control/estado y datos del periférico de salida (GPIO):

Para esto se debe leer con atención el esquemático para observar donde se conectará el LED y a que pin del IC está “ruteado”.
Luego, se debe leer el manual para conocer el periférico GPIO relativo a ese pin (direcciones, configuración, uso).
- 7- compilar y transferir la imagen al target

Primer programa embebido: GPIO

De Wikipedia:

GPIO (General Purpose Input/Output, Entrada/Salida de Propósito General) es un pin de E/S genérico en un chip, cuyo comportamiento se puede controlar por software en tiempo de ejecución.

Los pines GPIO no tienen ningún propósito especial definido, y no se utilizan de forma predeterminada. La idea es que a veces, para el diseño de un sistema completo que utiliza el chip, podría ser útil contar con un puñado de líneas digitales de control adicionales, y tenerlas a disposición ahorra el tiempo de tener que organizar circuitos adicionales para proporcionarlos.

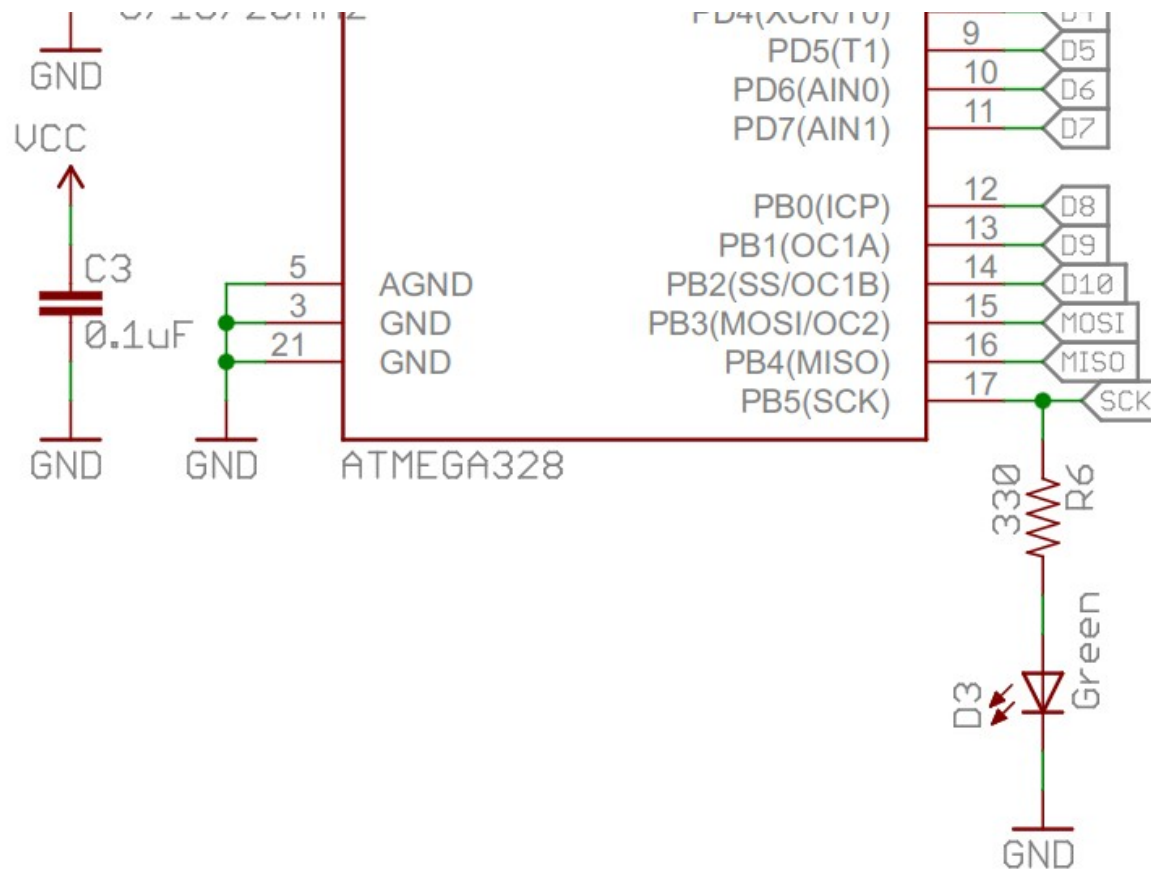
Ejemplo: los chips Realtek ALC260 (códec de audio) tienen 8 pines GPIO, que quedan sin utilizar de forma predeterminada. Algunos integradores de sistemas (Acer Inc. laptops) que emplean el ALC260 utilizan la primera GPIO (GPIO0) para poder encender por software el amplificador, utilizado para los altavoces internos y el conector de auriculares del ordenador portátil.

En los microcontroladores AVR atmega328p, los pines en un puerto GPIO son de acceso paralelos. Esto significa que se puede poner la señal en alto o bajo (on/off) de varios pines al mismo tiempo (al ejecutar una única instrucción de E/S en la CPU). Incluso, en el mismo momento, se pueden apagar algunas señales y encender otras.

Primer programa embebido: esquemático de la placa de circuito impreso (PCB)

Pasos:

0- Ubicación del LED (ver esquemático)



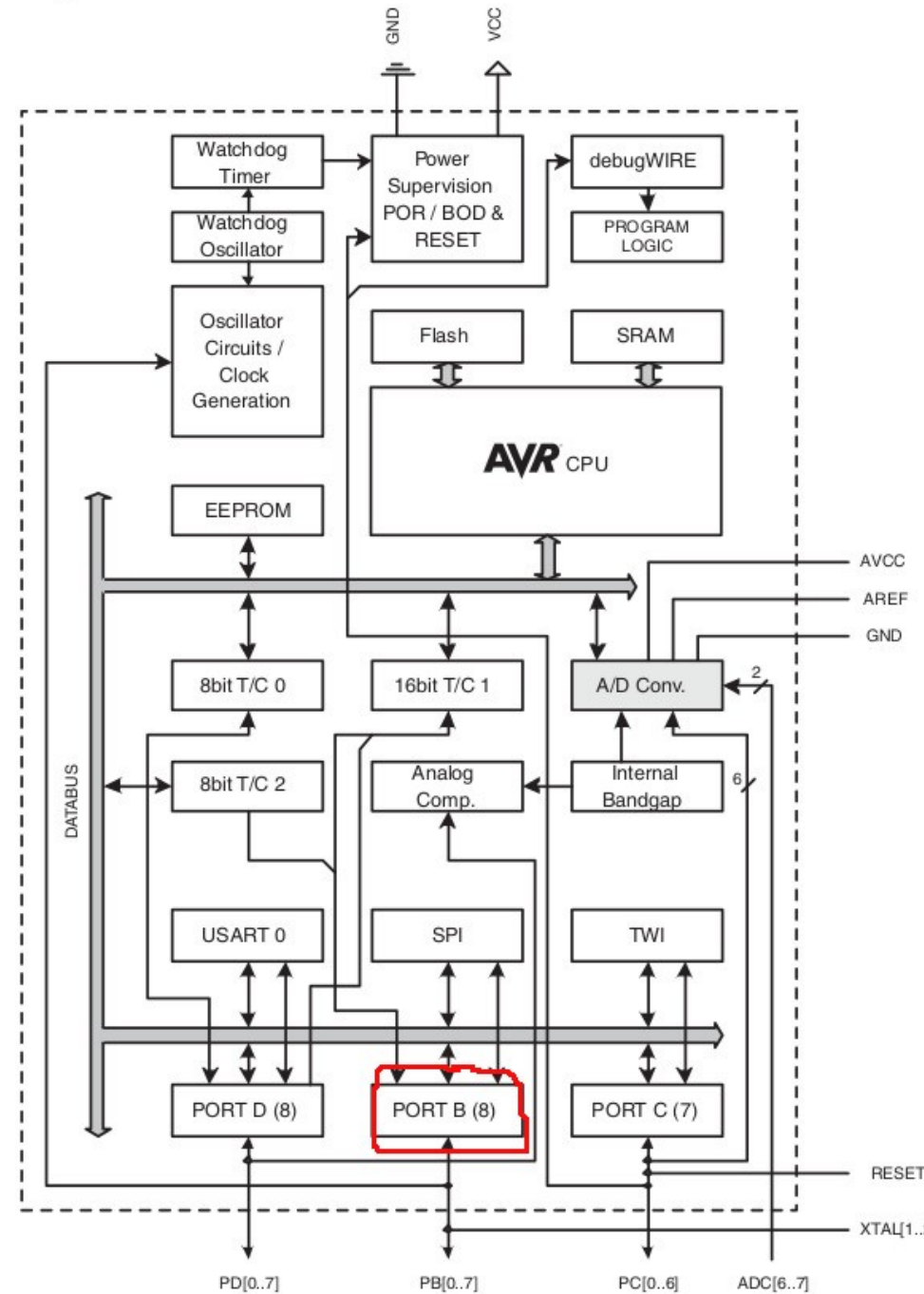
Primer programa embebido: Diagrama de bloques microcontrolador AVR 8-bit

Figure 2-1. Block Diagram

Pasos:

0- Ubicación del LED (ver esquemático)

1- Ubicación del PB5



Progr.

Primer programa embebido: Hoja de datos del AVR 8-bit

Pasos:

- 0- Ubicación del LED (ver esquemático)
- 1- Ubicación del PB5
- 2- Registros de PORT B
- 3- Direcciones de los registros

14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7 PORTB6 PORTB5 PORTB4 PORTB3 PORTB2 PORTB1 PORTB0								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	

14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7 DDB6 DDB5 DDB4 DDB3 DDB2 DDB1 DDB0								DDR B
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	

14.4.4 PINB – The Port B Input Pins Address⁽¹⁾

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	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	

Primer programa embebido: Hoja de datos del AVR 8-bit

Pasos:

0- Ubicación del LED (ver esquemático)

1- Ubicación del PB5

2- Registros de PORT B

3- Direcciones de los registros

4- Control de los registros

▶ 1. Pin Configurations	3
▶ 2. Overview	6
3. Resources	8
4. Data Retention	8
5. About Code Examples	8
6. Capacitive Touch Sensing	8
▶ 7. AVR CPU Core	9
▶ 8. AVR Memories	17
▶ 9. System Clock and Clock Options	27
▶ 10. Power Management and Sleep Modes	39
▶ 11. System Control and Reset	47
▶ 12. Interrupts	57
▶ 13. External Interrupts	70
▼ 14. I/O-Ports	75
14.1 Overview	75
▼ 14.2 Ports as General Digital I/O	76
14.2.1 Configuring the Pin	76
14.2.2 Toggling the Pin	76
14.2.3 Switching Between Input and Output	77
14.2.4 Reading the Pin Value	77
14.2.5 Digital Input Enable and Sleep Modes	79
14.2.6 Unconnected Pins	79
▼ 14.3 Alternate Port Functions	80
14.3.1 Alternate Functions of Port B	82
14.3.2 Alternate Functions of Port C	85
14.3.3 Alternate Functions of Port D	88
▼ 14.4 Register Description	91
14.4.1 MCUCR – MCU Control Register	91
14.4.2 PORTB – The Port B Data Register	91
14.4.3 DDRB – The Port B Data Direction Register	91
14.4.4 PINB – The Port B Input Pins Address(1)	91
14.4.5 PORTC – The Port C Data Register	91
14.4.6 DDRC – The Port C Data Direction Register	91
14.4.7 PINC – The Port C Input Pins Address(1)	92
14.4.8 PORTD – The Port D Data Register	92

14.2.1 Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in ["Register Description" on page 91](#), the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when reset condition becomes active, even if no clocks are running.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

14.2.2 Toggling the Pin

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn. Note that the SBI instruction can be used to toggle one single bit in a port.



14.2.3 Switching Between Input and Output

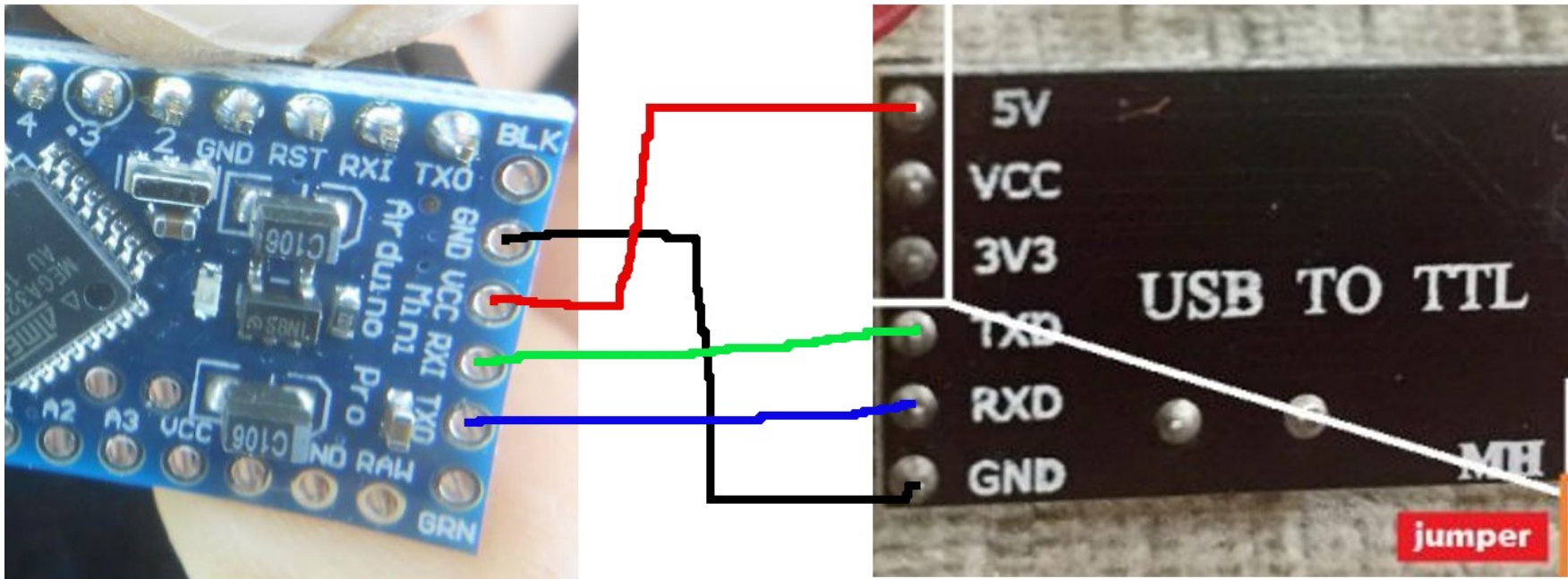
When switching between tri-state ($\{DDxn, PORTxn\} = 0b00$) and output high ($\{DDxn, PORTxn\} = 0b11$), an intermediate state with either pull-up enabled ($\{DDxn, PORTxn\} = 0b01$) or output low ($\{DDxn, PORTxn\} = 0b10$) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedance environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR Register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ($\{DDxn, PORTxn\} = 0b00$) or the output high state ($\{DDxn, PORTxn\} = 0b11$) as an intermediate step.

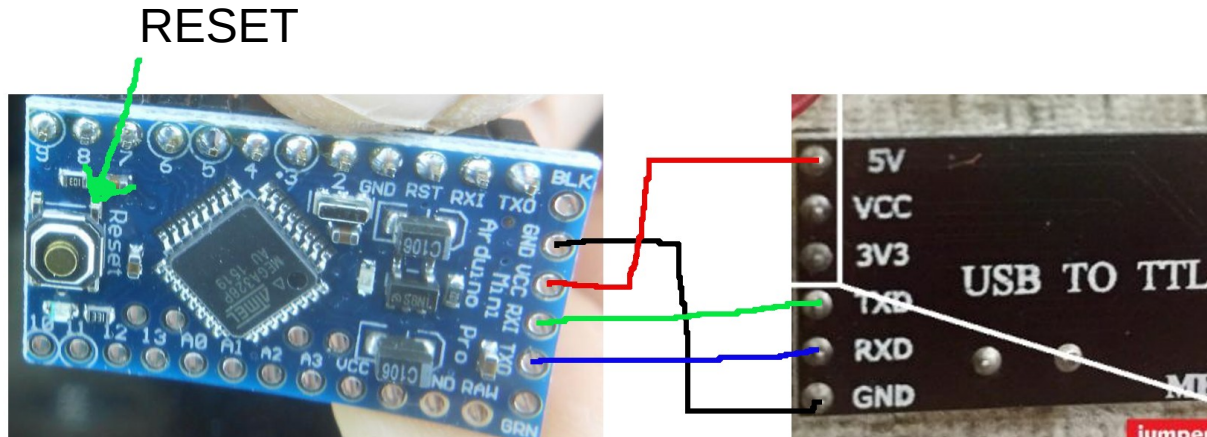
[Table 14-1](#) summarizes the control signals for the pin value.

Table 14-1. Port Pin Configurations

Primer programa embebido: Conexión AVR a PC (via USB-TTL)



Primer programa embebido: compilar y transferir el firmware (flash)



En la PC

```
# Compilar
```

```
make clean  
make
```

```
# Transferir firmware
```

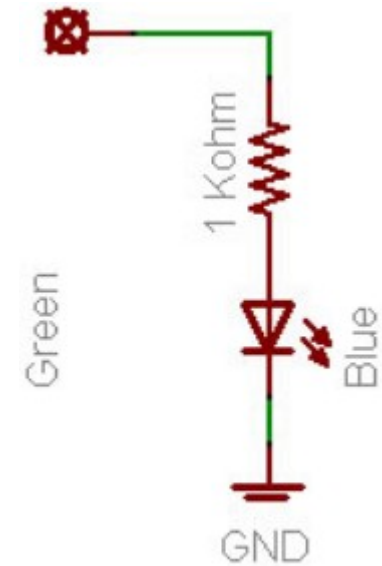
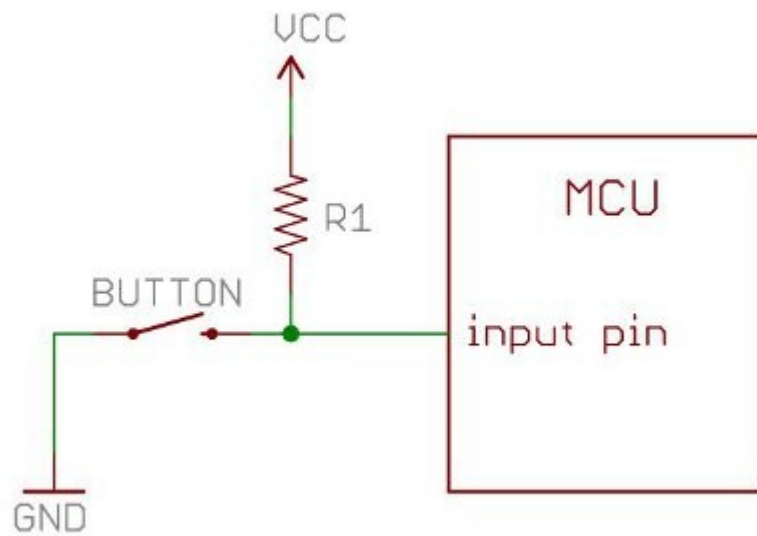
```
make flash [ENTER] <-- 2do.  
(si el comando falla por falta de permisos  
puede intentar: sudo make flash)
```

En la placa Arduino PRO MINI

1ro. Presionar botón RESET, dejar presionado
(tip: utilizar por ej. la goma de borrar de un lápiz con goma)

3ro. un segundo después de presionar ENTER en la PC
soltar el botón de RESET

Finalizando: esquemáticos de ENTRADA/SALIDA digital. (pull up resistor y LED)



Bibliografía

- **Programming Embedded Systems in C and C++, Michael Barr, O'Reilly, 1999, ISBN: 1-56593-354-5**
- **Designing Embedded Hardware, John Catsoulis, O'Reilly, 2003, ISBN: 1-596-00362-5**
- **Hoja de datos atmega328p, Atmel.**
- **Esquemático arduino pro mini**
- **Apunte: El primer programa embebido**