LSS 2016-17
Reti Logiche: introduzione ad Arduino

Piero Vicini (slides del Prof. A. Nigro)

A.A. 2016-2017

Introduzione: microprocessore e microcontrollore

Microprocessore

Un microprocessore integra sul chip la logica di elaborazione ma richiede sempre delle unità esterne (memorie, gestori di segnali e dispositivi periferici) per poter scambiare informazioni e interagire con l'esterno.

Microcontrollore

Un microcontrollore è invece un sistema completo, che integra in uno stesso chip il processore, la memoria permanente, la memoria volatile e i canali (pin) di I/O, oltre ad eventuali altri blocchi specializzati.
È progettato per interagire direttamente con il mondo esterno tramite un programma residente nella propria memoria interna e mediante l'uso di pin specializzati o configurabili dal programmatore. Sono disponibili in 3 fasce di capacità elaborativa (ampiezza del bus dati): 8 bit, 16 bit e 32 bit.

Introduzione (2)

Il primo microchip ottimizzato per applicazioni di controllo è stato il modello 8048 di Intel, uscito nel 1975, con RAM e ROM sullo stesso chip. Questo componente è stato utilizzato in più di un miliardo di tastiere per PC e numerose altre applicazioni.

Oggi si producono annualmente decine di miliardi di pezzi. Principali produttori:

- Renesas Technology (Giappone)
- Freescale Semiconductor (USA)
- Atmel (USA)
- Microchip Technology (USA)
- Infineon Technologies (Germania)
- Texas Instruments Incorporated (USA)
- Fujitsu (Giappone)
- NXP Semiconductors (Paesi Bassi)
- STMicroelectronics (Francia, Italia)
- Samsung Electronics (Corea del Sud)

Introduzione (3):

Componenti di un microcontrollore:

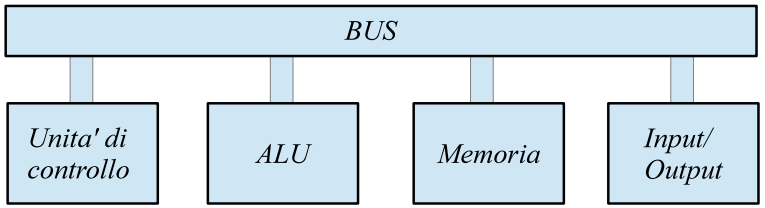
- CPU
- Memoria di programma: ROM, EPROM, FLASH
- Memoria dati: RAM, EPROM
- Oscillatore
- Porte di I/O
- Gestione di Interrupt
- Timer, contatori
- Moduli di comunicazione (SPI, I2C, USB....)
- ADC, DAC, PWM

Sistema di sviluppo:

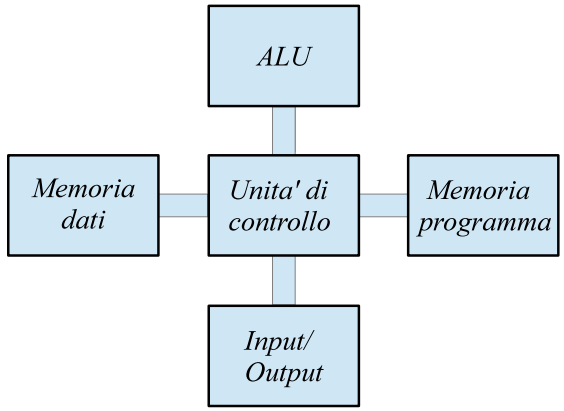
L'insieme di strumenti (hardware e software) necessari per generare il codice che deve essere eseguito dal processore, metterlo a punto e collaudarlo.

Introduzione (4): Architettura

- Von Neumann: Programma e dati sono ospitati sulla stessa memoria;
- Harvard: La memoria di programma e' separata da quella dei dati. Si utilizzano bus diversi.



(a)



(b)

Microcontrollori ATMEL AVR

AVR e' una famiglia di microcontrollori RISC ad architettura Harvard sviluppati dalla Atmel a partire dal 1996. L'AVR utilizza una memoria flash interna per il programma: questo permette di cancellare e riscrivere una nuova versione in pochi secondi e anche senza rimuovere il microcontrollore dalla scheda su cui è montato, velocizzando enormemente il processo di correzione e messa a punto del codice.

Caratteristiche:

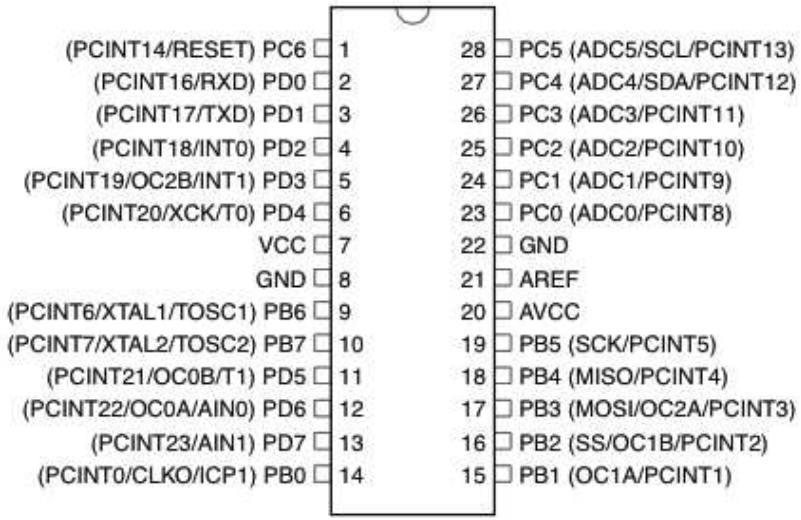
- Memoria di programma flash (riprogrammabile almeno 10000 volte)
- Memoria EEPROM (riscrivibile almeno 100000 volte)
- Memoria RAM statica
- Clock interno calibrato
- Porte di I/O
- Gestione di Interrupt
- Timer, contatori
- ADC, DAC, PWM

Ci sono 4 gruppi di microcontrollori, con caratteristiche differenti (ATtiny, ATMega, ATXMega, AT90).

ATMega328

Caratteristiche principali:

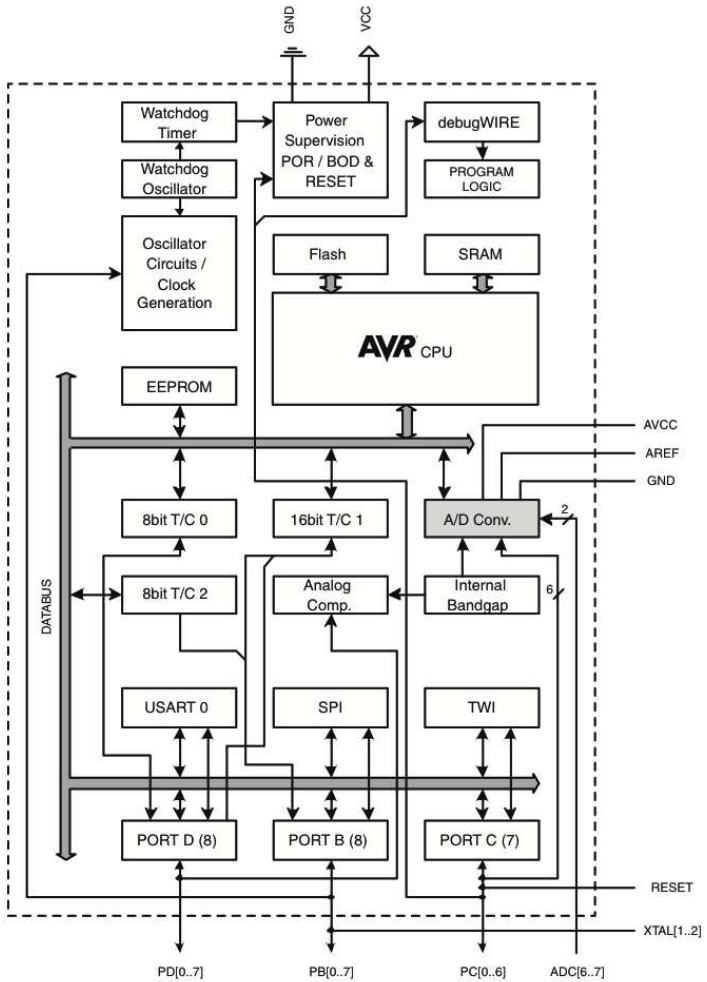
- Tecnologia CMOS
- Flash memory: 32KB
- EEPROM: 1 KB
- SRAM: 2KB
- Clock 16 MHz
- I/O digitali 14 (di cui 6 PWM)
- Ingressi analogici: 6
- Alimentazione: 5V



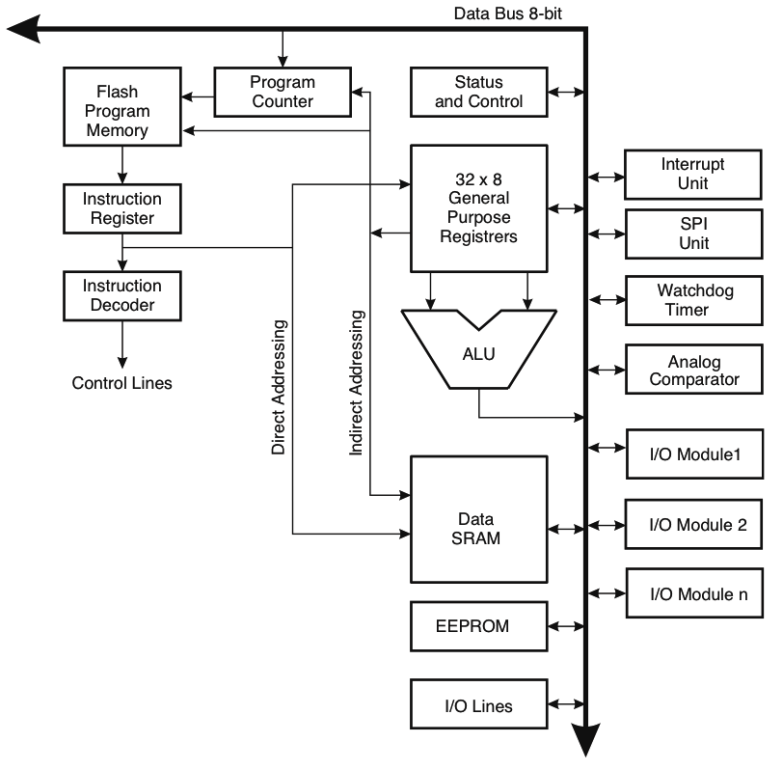
ATMega328: schema funzionale(1)

Il μC contiene inoltre:

- Circuiti per la comunicazione seriale;
- Timers, oscillatori;
- V_{ref} interna (1.1 V);
- Gestione dell'alimentazione;
- Gestione di interrupts;
- ....



ATMega328: schema funzionale(2)



ATMega328: Timing

Figure 6-4. The Parallel Instruction Fetches and Instruction Executions

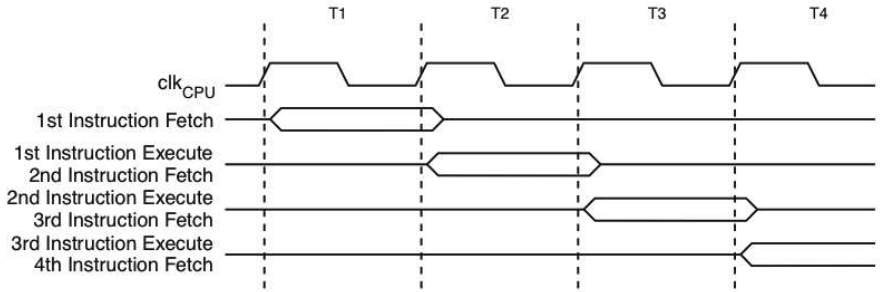
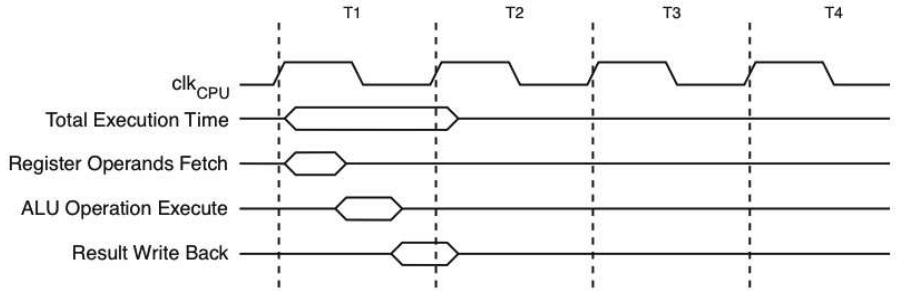


Figure 6-5 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 6-5. Single Cycle ALU Operation



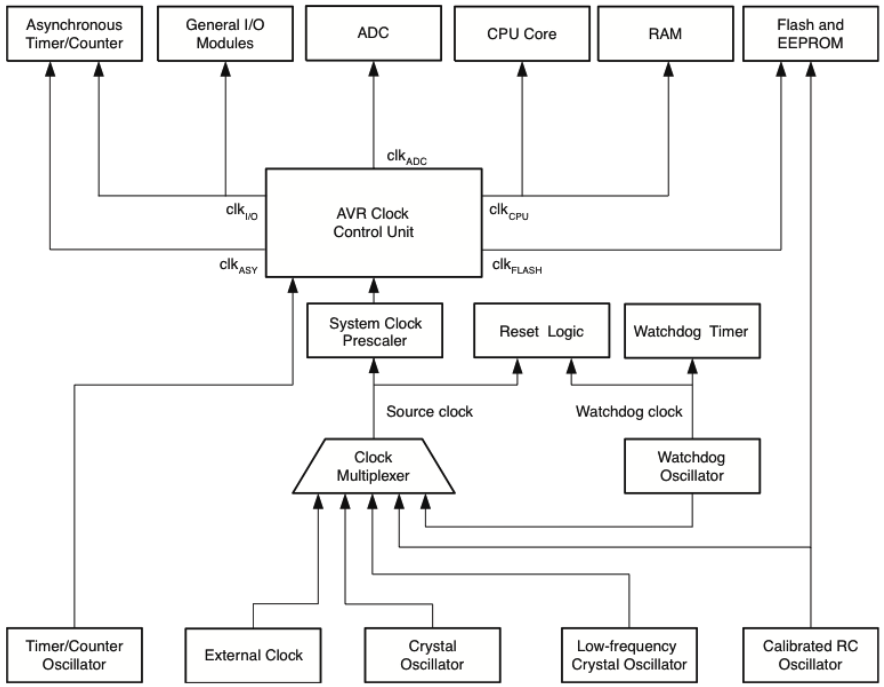
ATMega328: Interrupts

Ci sono varie possibili sorgenti di interrupts, interne o esterne e per ciascuna di esse un interrupt vector. Gli interrupt vectors sono contenuti nella memoria di programma e ordinati in base alla priorita'. Il Reset e' l'interrupt con priorita' piu' alta. Per ogni interrupt e' previsto 1 bit individuale di abilitazione, ma nello Status Register vi e' un bit (Global interrupt enable) di abilitazione globale. Quando arriva un interrupt il Global interrupt enable viene disabilitato e si avvia l'esecuzione della routine di servizio corrispondente. All'uscita di questa il Global bit viene nuovamente abilitato. Ci sono sostanzialmente due tipi di interrupt: nel primo tipo, l'evento viene memorizzato in un bit (interrupt flag) e viene servito appena possibile. Ovvero, se una o piu' condizioni di interrupts avvengono quando il Global enable e' disabilitato, non vengono perse e vengono servite non appena il Global enable e' abilitato di nuovo. Nel secondo tipo invece l'interrupt verra' servito solo se la condizione che lo ha causato persiste quando il Global enable e' abilitato di nuovo. Quando l'AVR esce da un interrupt ritorna sempre al programma principale ed esegue una istruzione prima di occuparsi di eventuali altri interrupt in attesa.

ATMega328: Clock

Sistema di gestione e distribuzione del clock:

Figure 8-1. Clock Distribution



Sono possibili diverse sorgenti di clock:

- interno (8 *MHz*, ma modificabile);
- esterno ($0 \div 20$ *MHz*), di vario tipo.

ATMega328: Alimentazione

- vari *sleep mode* selezionabili da programma per ridurre i consumi;
- *Brown out detector*: invia un segnale di reset quando la tensione di alimentazione scende sotto un livello prefissato;
- *Power on reset*: invia il segnale di reset finche' la tensione di alimentazione e' al di sotto di una certa soglia.

ATMega328: Watchdog

Il *watchdog* e' un mecccanismo di sicurezza che puo' essere utilizzato per evitare che il microcontrollore resti bloccato a causa di errori inaspettati nel programma o comunque altre cause. il sistema e' composto da un oscillatore a 128 *kHz* e un contatore con varie uscite. Sulla base di questo si puo' costruire una logica di *time-out* che puo' essere utilizzata per inviare un segnale di reset al microcontrollore.

ATMega328: Reset

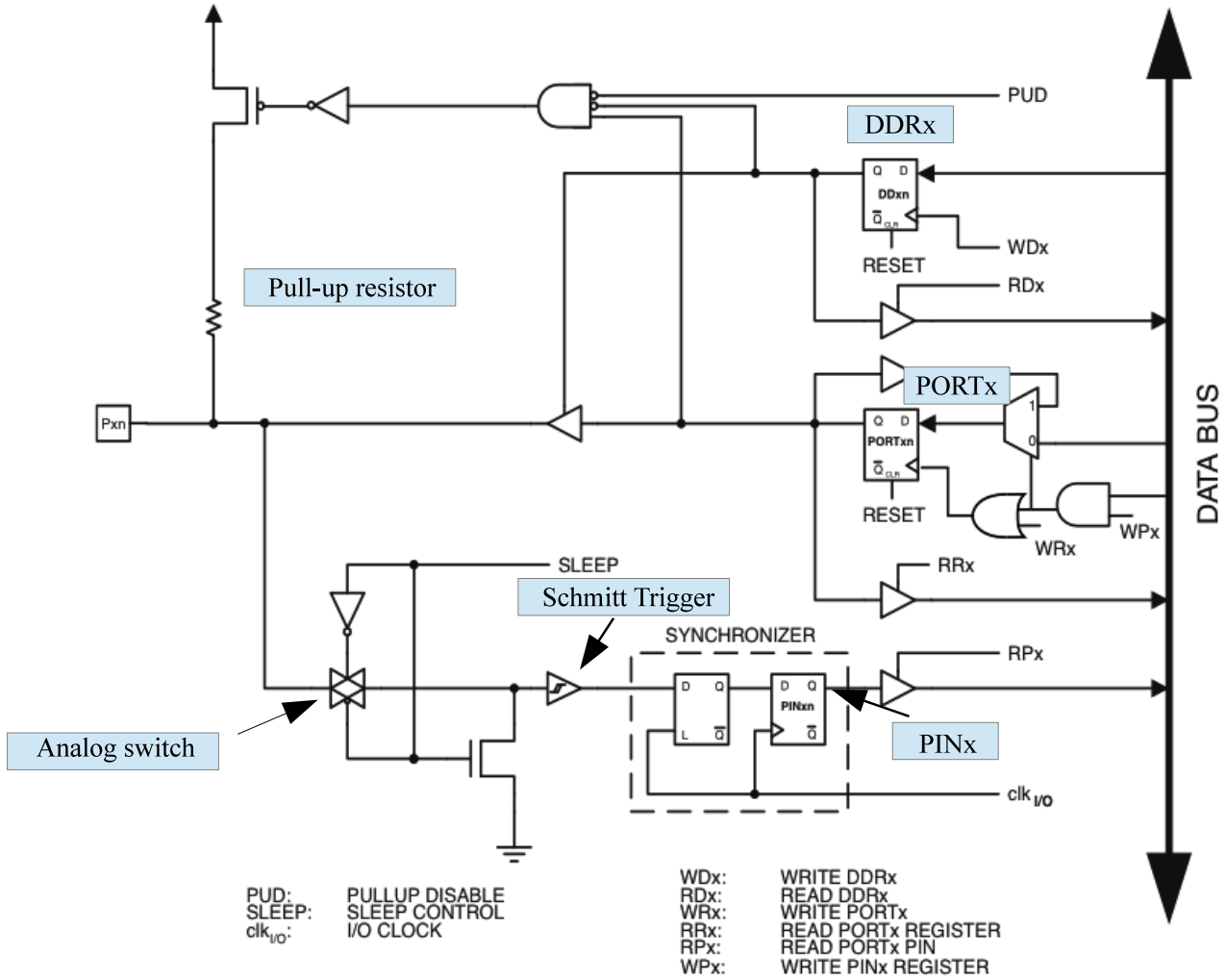
Ci sono quindi varie sorgenti che provocano il reset del microcontrollore:

- Un livello basso sul pin di reset;
- Power-on Reset;
- Brown-out-Reset;
- Watchdog Reset

ATMega328: Timers

Il microcontrollore comprende 2 timers a 8 bit e 1 timer a 16 bit che possono essere utilizzati in vari modi dal programma. Utilizzano un clock interno ma possono anche essere collegati ad un clock esterno.

ATMega328: Input-output digitale



ATMega328: Input-output digitale

Tutte le porte hanno le seguenti caratteristiche:

- resistore di pull-up selezionabile;
- logica 3-state;
- trigger di Schmitt in ingresso;
- sincronismo con il clock di sistema;
- in uscita possono erogare o assorbire corrente.

Tutte le porte degli AVR hanno una funzione di *true Read-Modify-Write* quando vengono usate come I/O digitali. Questo significa che la direzione di un singolo pin puo' essere cambiata al volo senza interferire con gli altri pin.

ATMega328: Input-output digitale

Il controllo di ogni porta e' effettuato con 3 registri:

- DDRx: Data Direction Register;
- PORTx: Output Register;
- PINx: Input Register

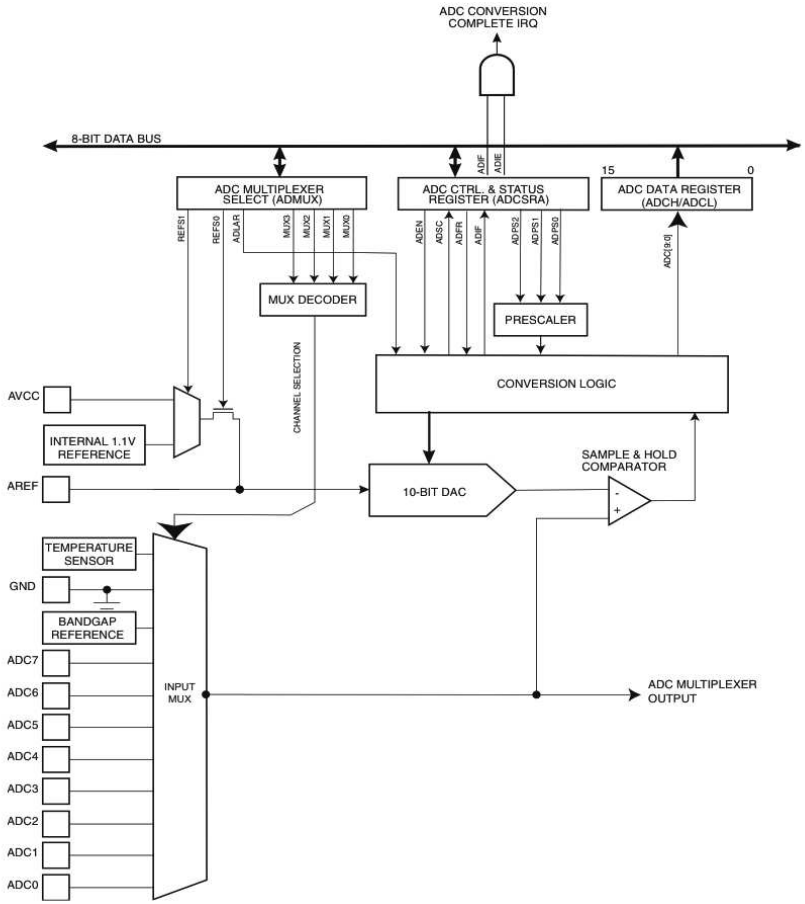
Ad esempio, PORTB3 contiene il bit in uscita per il pin 3 del Port B.

DDx	PORTx	PUD	I/O	Pull-up	Note
0	0	x	Input	No	Alta impedenza
0	1	0	Input	SI	
0	1	1	Input	No	Alta Impedenza
1	0	x	Output	No	
1	1	x	Output	No	

ATMega328: Uscite analogiche

Questa famiglia di microcontrollori non dispone di uscite analogiche. Tuttavia, l'uscita di alcuni pin digitali puo' essere modulata, ovvero utilizzata per emettere un'onda rettangolare (fra 0 e 5 V) con duty cycle variabile. Con un semplice integratore si puo' quindi avere una tensione *quasi* analogica.

ATMega328: Ingressi analogici



ATMega328: Ingressi analogici

Un ADC a 10 bit (sample and hold, successive approximation). Il tempo di conversione e'compreso tra 13 e 260 μs.
E' collegato tramite un multiplexer a 8 ingressi (solo 6 effettivamente utilizzabili, collegati ai pin $PC0 \div PC5$). L'ADC ha un'alimentazione separata (pin AV_{CC}) per migliore immunita' ai disturbi. L'uscita digitale e':

$$\text{Output} = \frac{2^{10} - 1}{V_{ref}} V_{in}$$

Ci sono 3 possibili opzioni per definire V_{ref}:

- AV_{CC};
- 1.1 V (Riferimento di tensione interno a bandgap);
- A_{ref} (Tensione sul pin A_{ref}).

La gestione dell'ADC e' delegata a 4 registri a 8 bit:

- ADMUX: Contiene le informazioni sulla scelta del canale d'ingresso e della V_{ref};
- ADCSRA: bits di status e controlli;
- ADCL e ADCH: contengono i 10 bit del risultato.

ATMega328: Comparatore analogico

Il comparatore analogico compara i valori in ingresso sull'ingresso positivo $AIN0$ e su quello negativo $AIN1$ (ovvero i pin $PD6$ e $PD7$); quando $AIN0 > AIN1$ l'uscita del comparatore ACO va ad 1. Questo segnale puo' essere usato per generare un Interrupt (sul fronte di salita o di discesa), oppure come trigger per il Timer/Counter 1. E' possibile anche utilizzare, all'ingresso non invertente, la tensione di riferimento interna (1.1 V).
E' anche possibile selezionare uno qualunque dei pin analogici di ingresso ($ADC0 \ldots ADC7$) come ingresso negativo del comparatore (poiche' utilizza lo stesso multiplexer dell'ADC, quest'ultimo deve essere off)

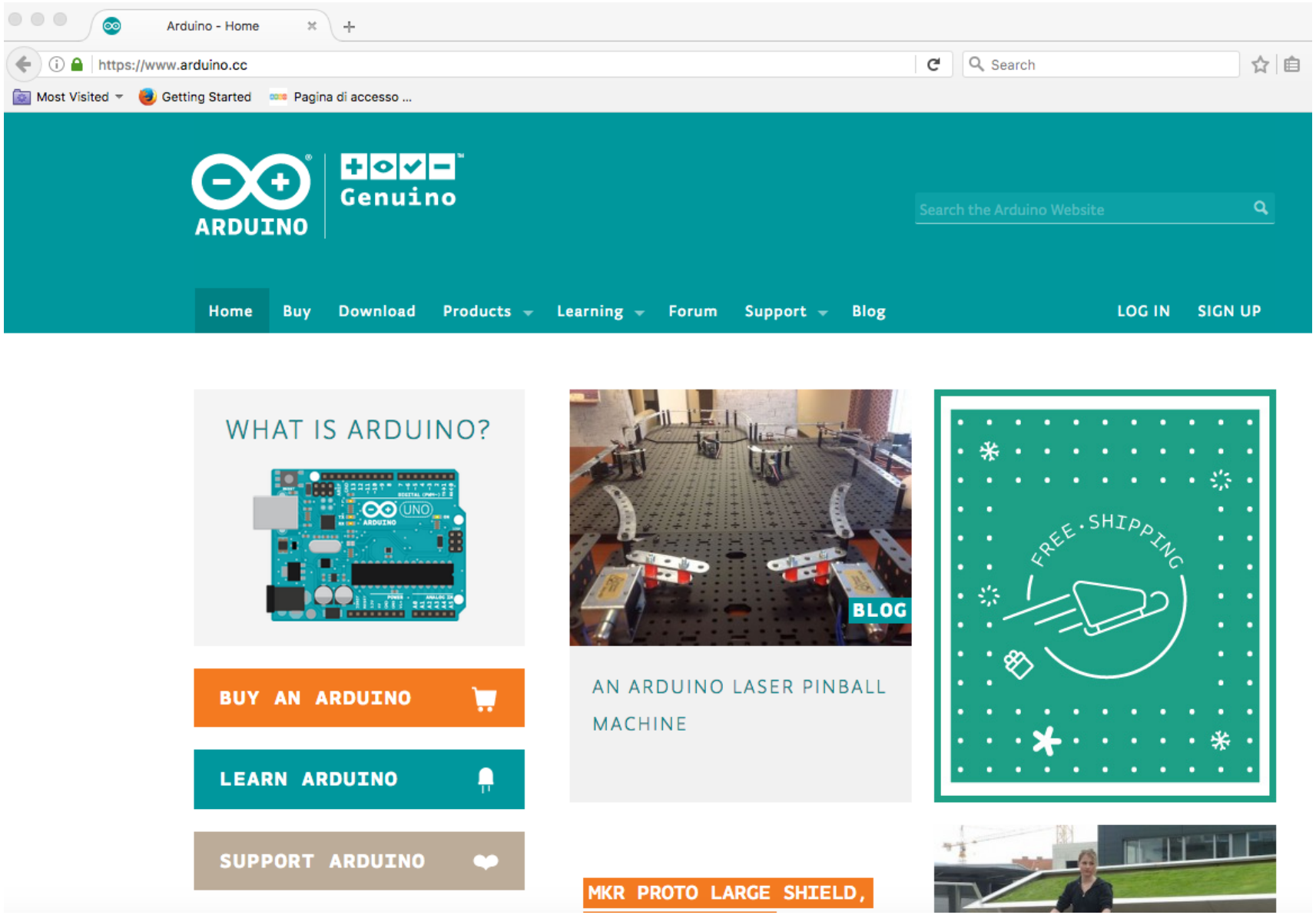
ATMega328: Comunicazione seriale

Il microcontrollore supporta vari meccanismi di comunicazione con dispositivi esterni:

- USART (Universal Synchronous Asynchronous Receive Transmit): Pin *PD*0 e *PD*1;
- SPI (Serial Peripheral Interface): Pin *PB*2, *PB*3, *PB*4, *PB*5;
- I2C/TWI (Inter Integrated Circuit / Two Wire Interface: Pin *PC*4 e *PC*5.

Sono anzitutto necessari per consentire all'utilizzatore di caricare e gestire il programma che deve essere eseguito sul microcontrollore, ma consentono anche di gestire dispositivi esterni (integrati) ovvero di trasferire dati verso l'esterno.
Ne discuteremo in dettaglio piu' avanti.

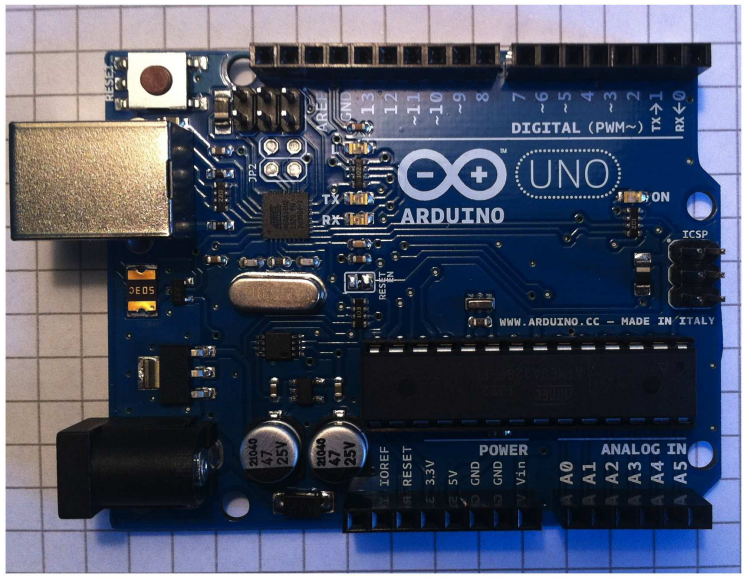
Tutte le info qui –> https://www.arduino.cc



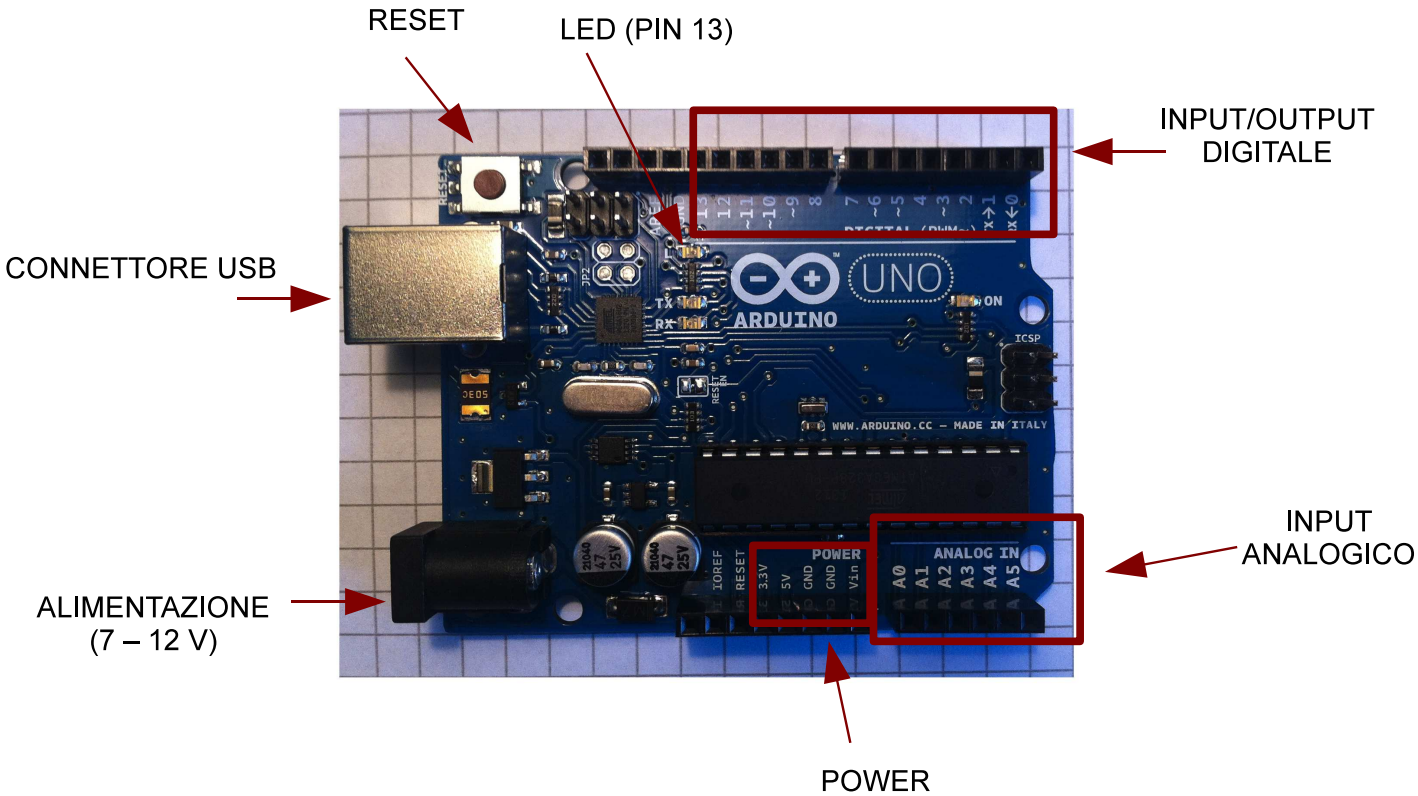
Arduino

E’ un sistema di sviluppo di piccole dimensioni basato su Microcontrollori ATMEL, adatto per sviluppo di prototipi e per scopi didattici.
E’ integrato con un ambiente di sviluppo software (open source) che permette un facile e rapido utilizzo del microcontrollore.
Esistono varie versioni, noi utilizzeremo la scheda Arduino Uno, basata sul microcontrollore ATMEL ATMEGA328.

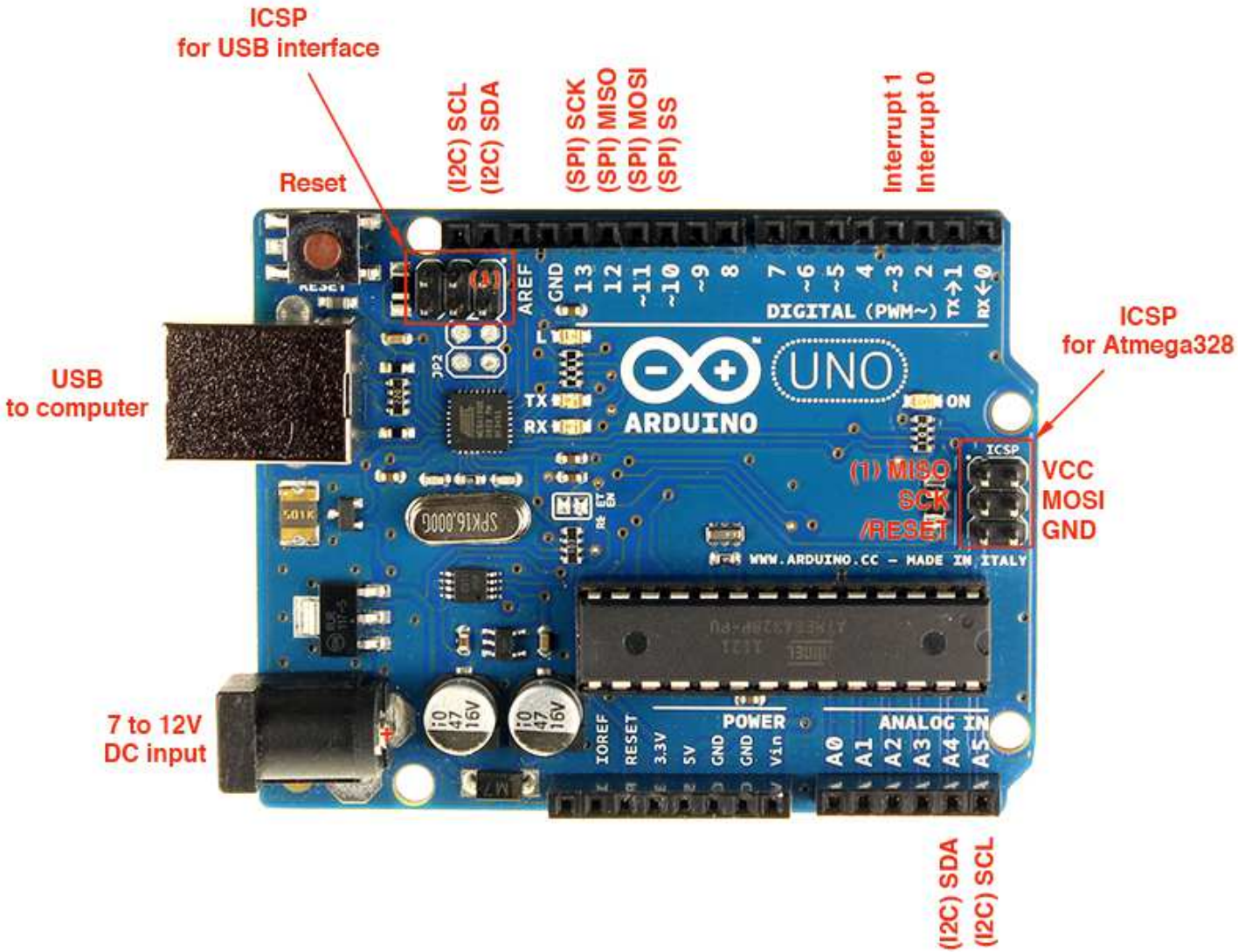
Arduino Uno: la scheda



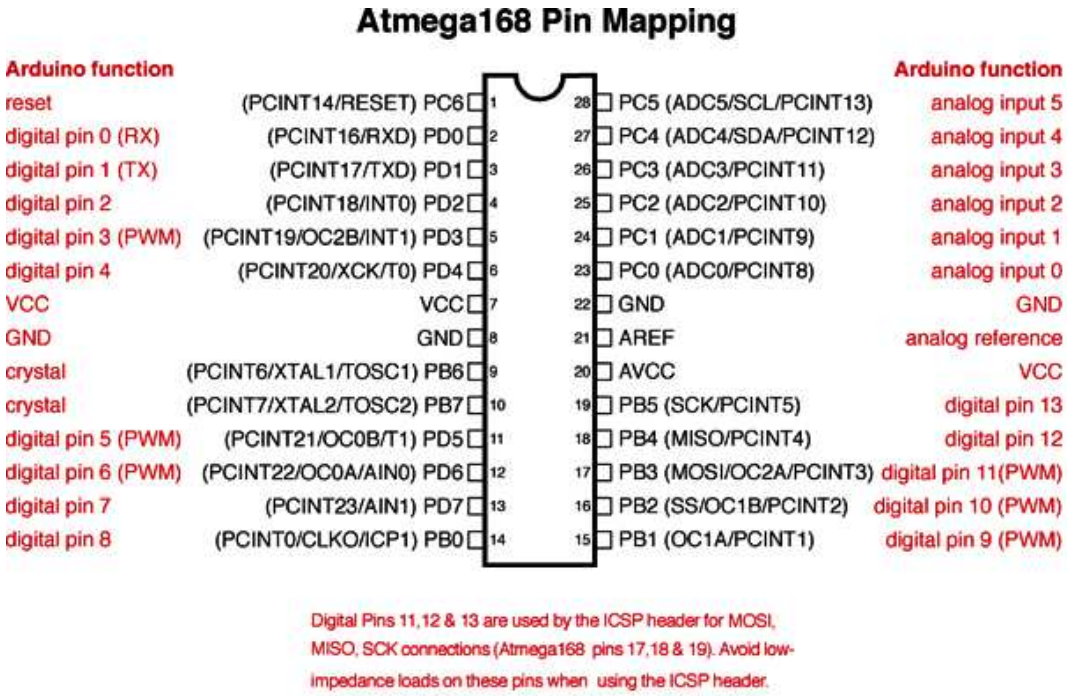
Arduino Uno: la scheda



Arduino Uno: la scheda



Arduino: pin mapping del ATMega328



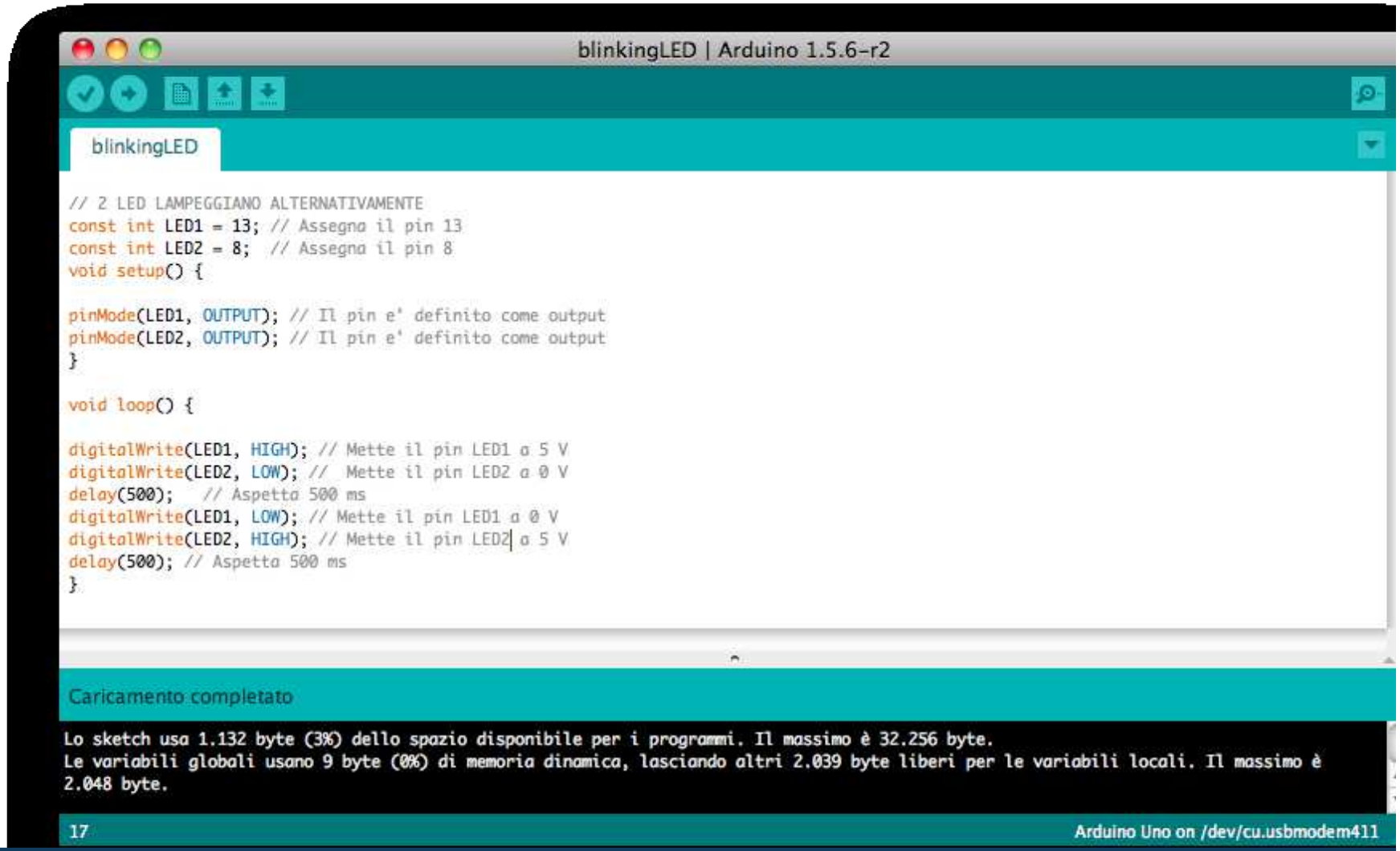
Ambiente di sviluppo

- Arduino dialoga con un computer host (PC Windows, Mac, Linux) tramite la porta USB;
- E' necessario installare sull'host il programma Arduino (open source). E' scritto in Java e quindi in grado di funzionare su molte piattaforme.
- La programmazione del microcontrollore e' in linguaggio C.

La finestra di scrittura del programma

Offre un *framework* per la scrittura dello *sketch* (come e' chiamato nel gergo di Arduino)

Semplice esempio



Semplice esempio (2)

Prima di compilare Arduino trasforma lo sketch creando il programma in *c* completo:

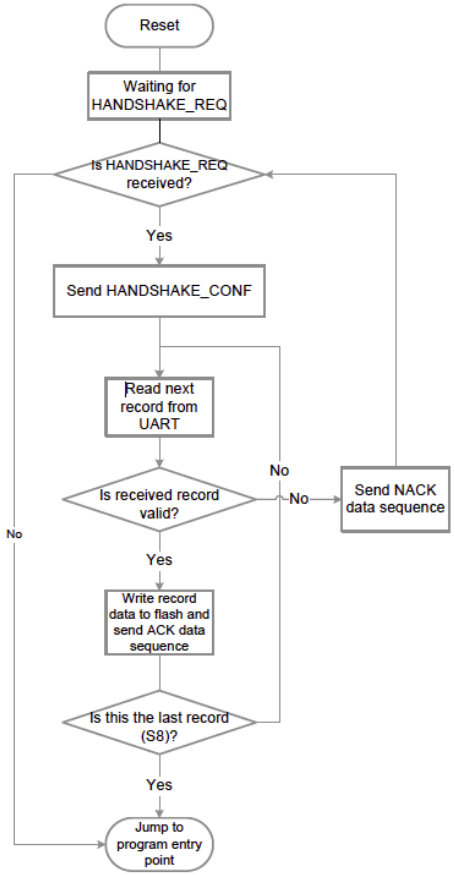
```
#include "Wprogram.h";
void setup();
void loop();
void setup() {
pinMode(LED1, OUTPUT); // Il pin e' definito come output
pinMode(LED2, OUTPUT); // Il pin e' definito come output
}
void loop() {
digitalWrite(LED1, HIGH); // Mette il pin LED1 a 5 V
digitalWrite(LED2, LOW); //  Mette il pin LED2 a 0 V
delay(500);    // Aspetta 500 ms
digitalWrite(LED1, LOW); // Mette il pin LED1 a 0 V
digitalWrite(LED2, HIGH); // Mette il pin LED2 a 5 V
delay(500); // Aspetta 500 ms
}
int main() {
setup();
for(;;) { loop();}
return 0;
}
```

Semplice esempio (3)

Il compilatore (residente sul PC) crea il codice eseguibile. Il codice viene poi trasferito sulla memoria flash del μC (attraverso la connessione USB) ed eseguito.
Dal lato del μC il trasferimento e la scrittura in memoria sono eseguiti da un programma residente (BootLoader) che occupa gli ultimi 512 bytes della memoria flash.
Il programma e' memorizzato permanentemente e viene eseguito ogni volta che si accende la scheda (anche senza aprire Arduino sul PC), finche' non e' sostituito da un altro programma.

Il bootloader

Al momento dell'accensione (o al reset) il bootloader si avvia e verifica se c'e' una richiesta di comunicazione proveniente dalla porta seriale.
Se non c'e' passa il controllo all'applicazione esistente.
Se invece c'e' avvia il dialogo tramite la porta seriale, scarica l'applicazione nuova e poi la avvia.
La presenza del bootloader non è indispensabile. Il programma da eseguire puo' essere caricato connettendo il μC ad una apposita scheda (programmatore).
Questa soluzione e' adatta per situazioni in cui non si prevede che l'applicazione debba subire modifiche o aggiornamenti.



Linguaggio

Un sommario del linguaggio di Arduino puo' essere reperito su:

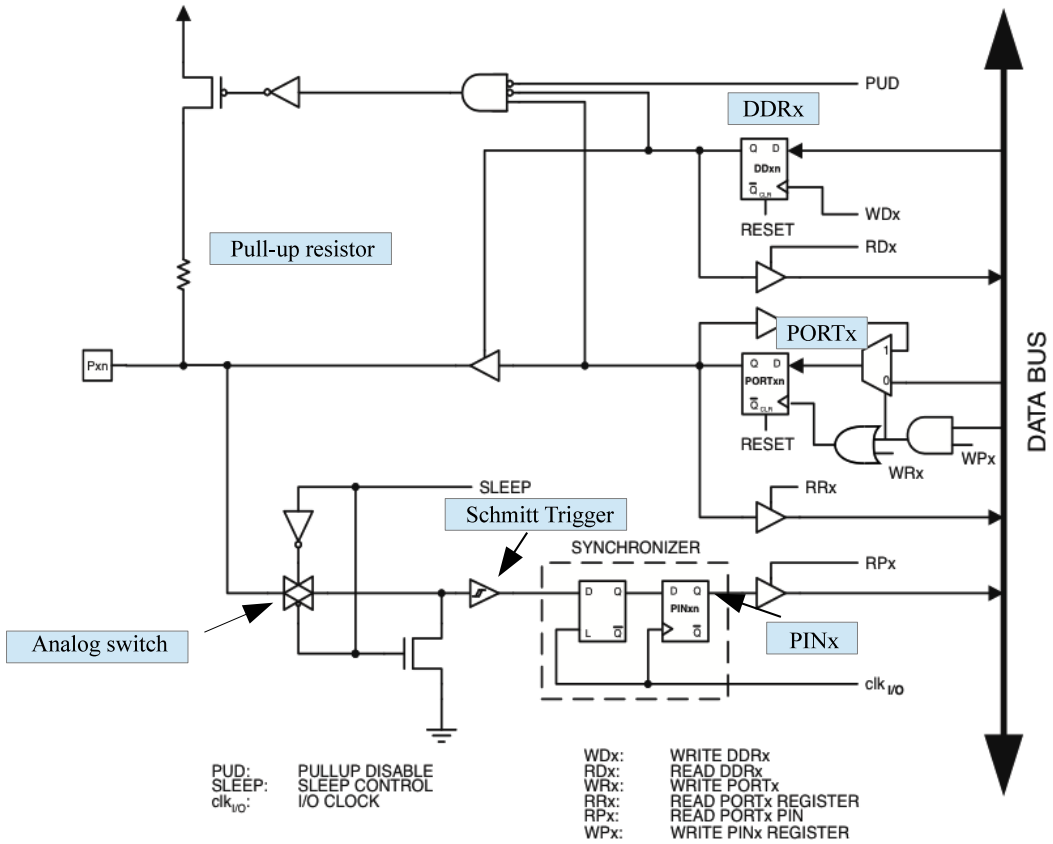
http://arduino.cc/en/Reference/HomePage

E' disponibile un'ampia libreria software per molteplici applicazioni.
Qui vedremo solo alcune cose fondamentali per iniziare.

I/O Digitale

- pinMode(pin,mode); (mode: *INPUT*, *OUTPUT*, *INPUT_PULLUP*)
- digitalWrite(pin, value); (value: *HIGH*, *LOW*)
- digitalRead(pin);

ATMega328: PULL UP



I/O Digitale: esempio

```
int ledPin = 13;
int inPin = 7;
int val = 0;
void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(inPin, INPUT);
}
void loop()
{
  val = digitalRead(inPin);
  digitalWrite(ledPin, val);
}
```

Output Analogico

I pin 3,5,6,9,10,11 possono, in uscita, essere utilizzati in modo “analogico” (PWM). (Pin 5,6: 980 Hz; Pin 3,9,10,11: 490 Hz)

Esempio:

```
int ledPin = 9;       // LED connected to digital pin 9
int val = 128;          // val: 0-255 (duty cycle)
void setup()
{
  pinMode(ledPin, OUTPUT);    // sets the pin as output
}
void loop()
{
  analogWrite(ledPin, val);   //
}
```

ADC

Il microcontroller contiene un ADC a 10 bit (sample and hold, successive approximation).
E' collegato tramite un multiplexer a 8 ingressi (6 effettivamente utilizzabili su Arduino).

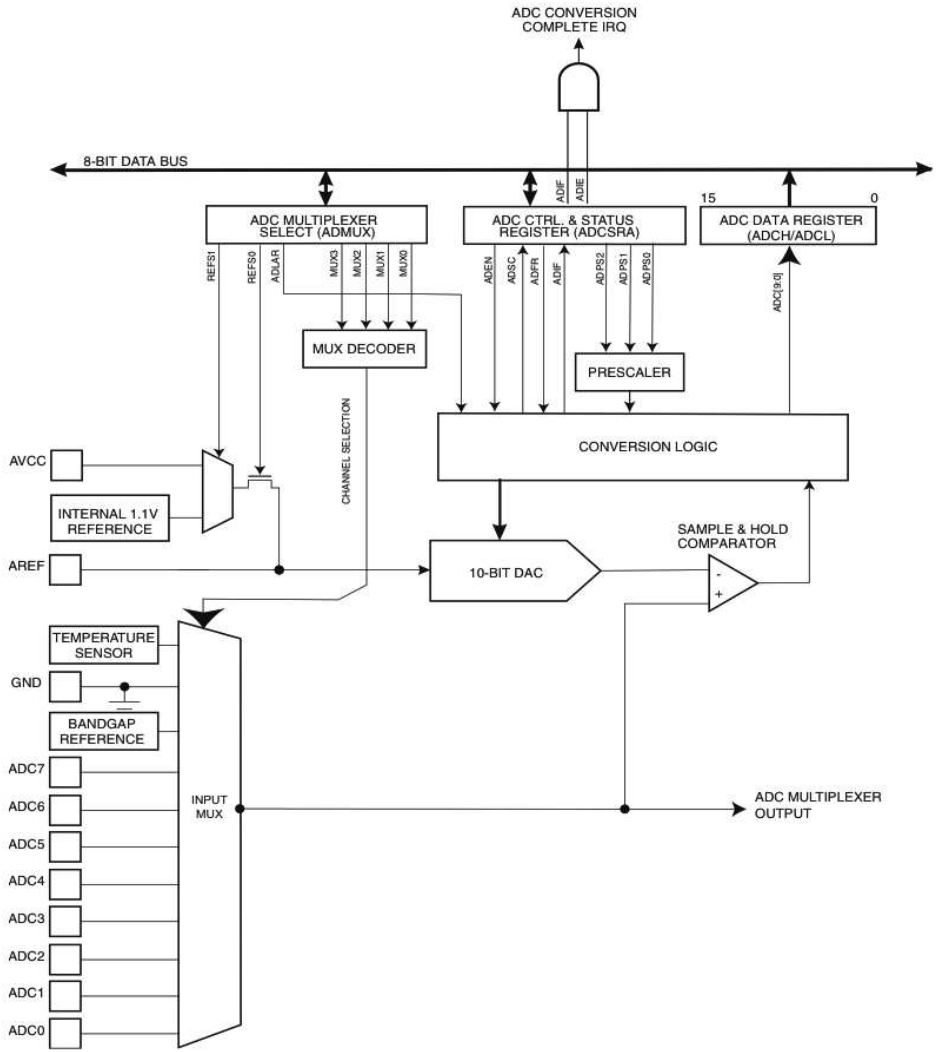
$$Output = \frac{2^{10}}{V_{ref}} V_{in}$$

3 possibili V_{ref}:

- 5 V (default)
- 1.1 V
- A_{ref} (esterna)

Nota bene:

$$0 \leq A_{ref} \leq +5\ V$$



Esempio

```
int analogPin = 3;      // analog pin 3
int val = 0;            // variable to store the value read
void setup()
{
  analogReference(EXTERNAL); // La Vref e' fornita esternamente
  Serial.begin(9600);        //  setup serial
}
void loop()
{
  val = analogRead(analogPin);    // read the input pin
  Serial.println(val);            // debug value
}
```

Comunicazione

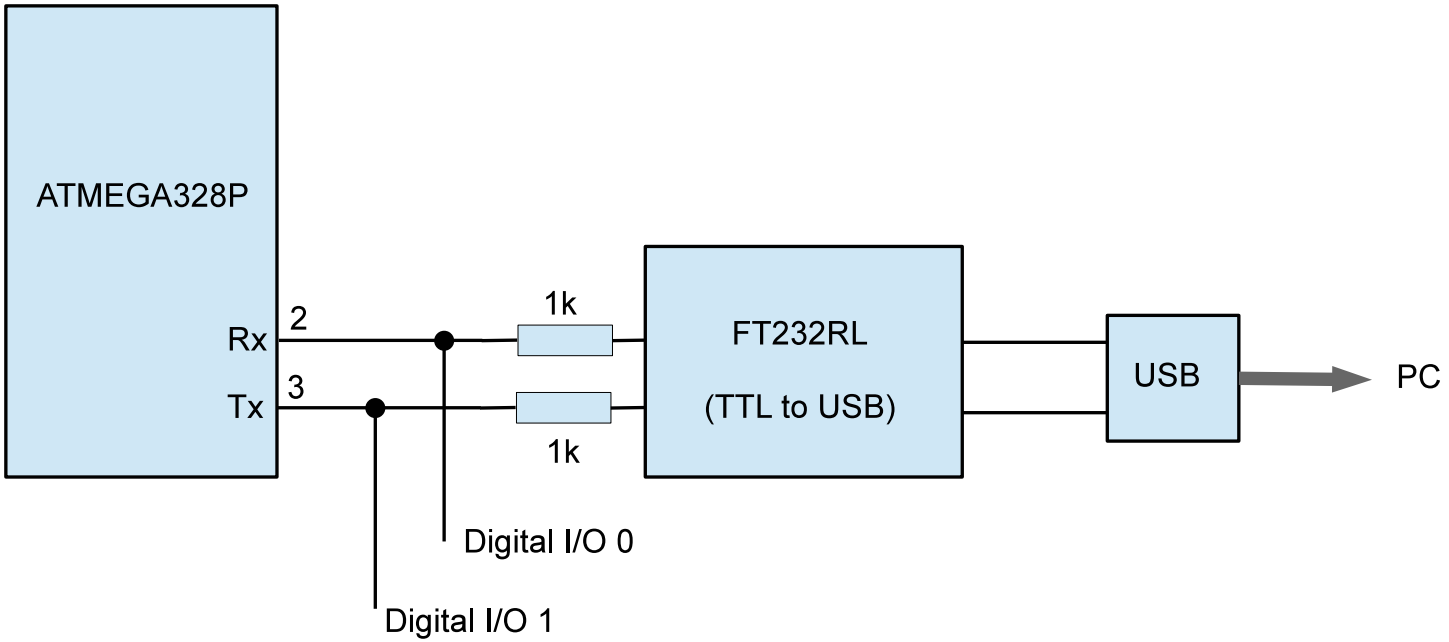
Il microcontrollore supporta vari meccanismi di comunicazione con dispositivi esterni:

- USART (Universal Synchronous Asynchronous Receive Transmit)
  Digital I/O 0 e 1
- SPI (Serial Peripheral Interface)
  Digital I/O 10, 11, 12, 13
- I2C/TWI (Inter Integrated Circuit / Two Wire Interface
  Analog pin A4, A5

Arduino fornisce librerie che facilitano l'uso di questi protocolli.

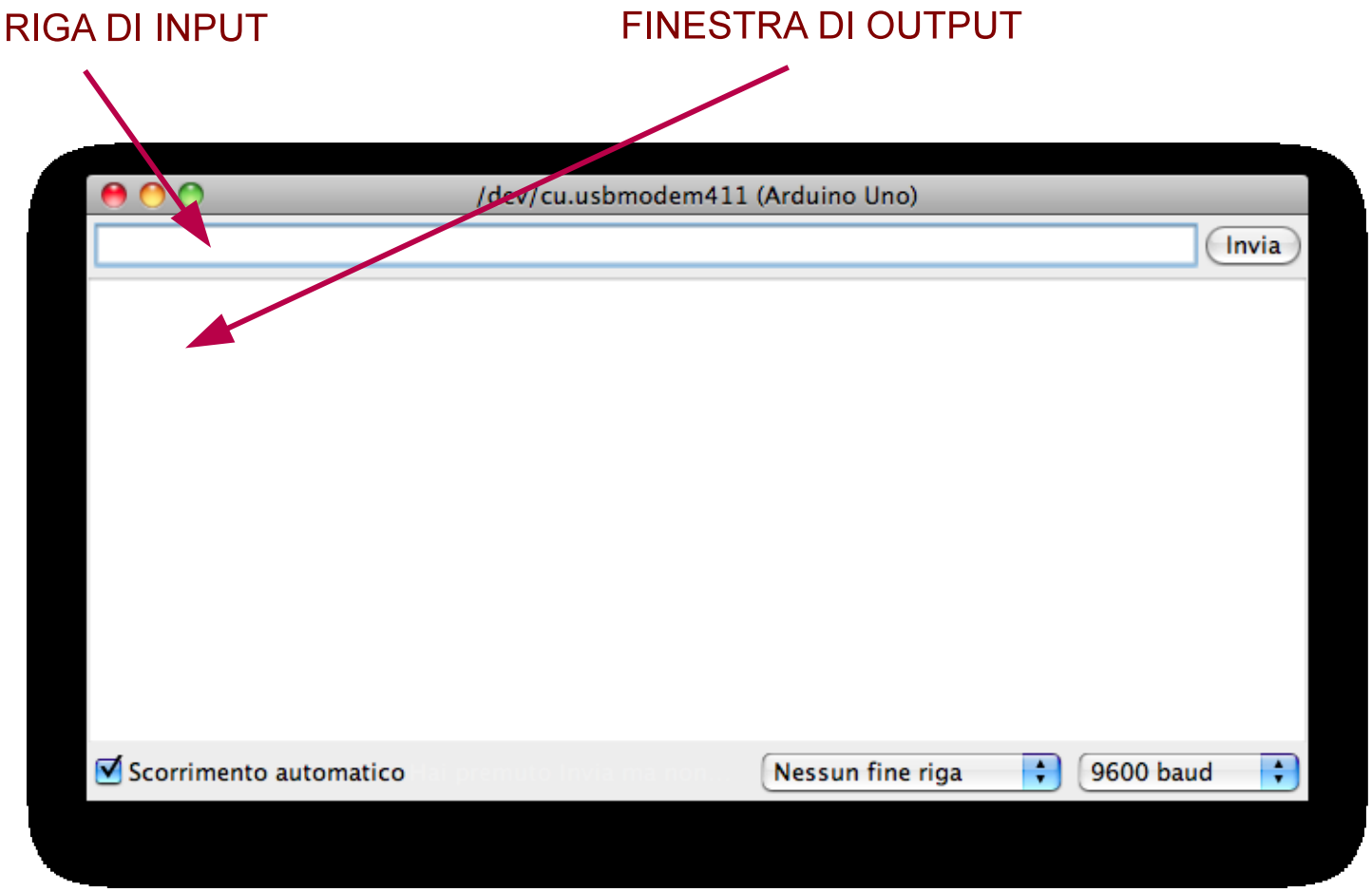
Comunicazione seriale USART

E' utilizzata da Arduino per connettere il μC al PC, tramite la porta USB: da qui viene fatto il download del programma.



Puo' liberamente essere poi utilizzata dall'utente.

Comunicazione seriale; finestra di monitor



Comunicazione seriale: esempio

```
int data = 0;
void setup() {
Serial.begin(9600); //Inizializzazione baud rate
}

void loop() {
  if(Serial.available() > 0) // verifica se esiste un INPUT
  {
data = Serial.parseInt();   // legge come numero intero
Serial.println(data);       // scrive sulla finestra di OUTPUT
delay(200);
  }
}
```

NOTA BENE:

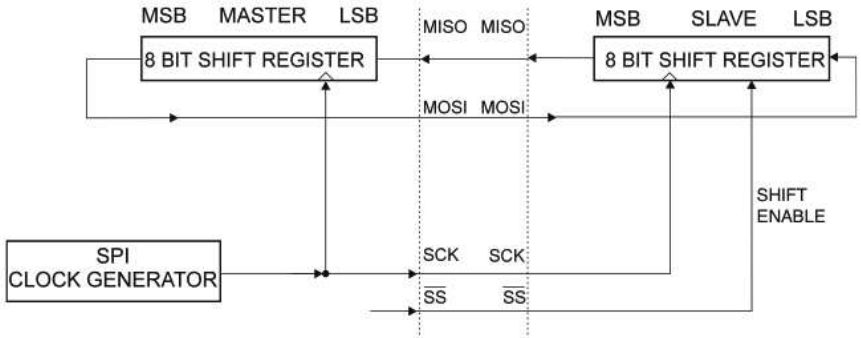
Se si usa la comunicazione seriale i pin I/O 0 *e* 1 *non possono essere usati per altri scopi!*

SPI (Serial Peripheral Interface)

E' un sistema di comunicazione tra un microcontrollore e altri circuiti integrati o tra più microcontrollori.
La trasmissione avviene tra un dispositivo detto master e uno o più slave. Il master controlla il bus, emette il segnale di clock, decide quando iniziare e terminare la comunicazione.

Richiede 4 linee di comunicazione:

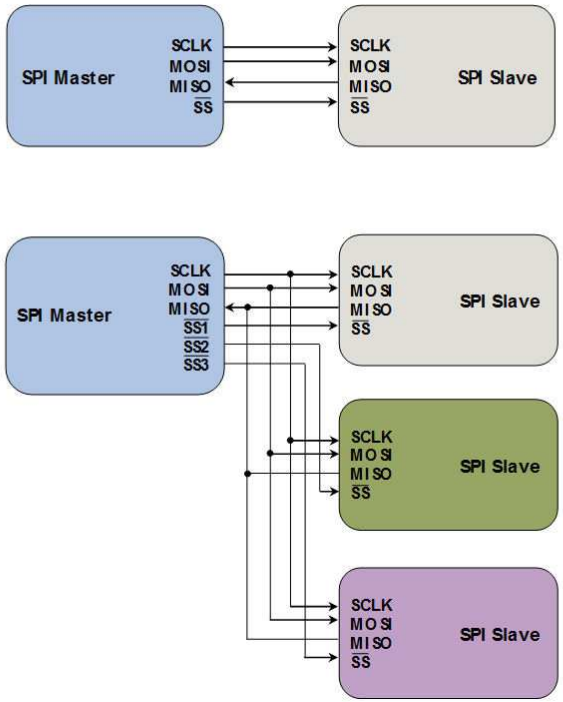
- SCK: Serial Clock;
- MOSI: Master Output - Slave Input;
- MISO: Master Input - Slave Ouput;
- SS: Slave Select



I vari dispositivi si scambiano dati utilizzando Shift Register.
(Libreria Arduino: SPI library)

SPI (Serial Peripheral Interface)-2

Esempi di connessione SPI



SPI (Serial Peripheral Interface)-3

Temporizzazione

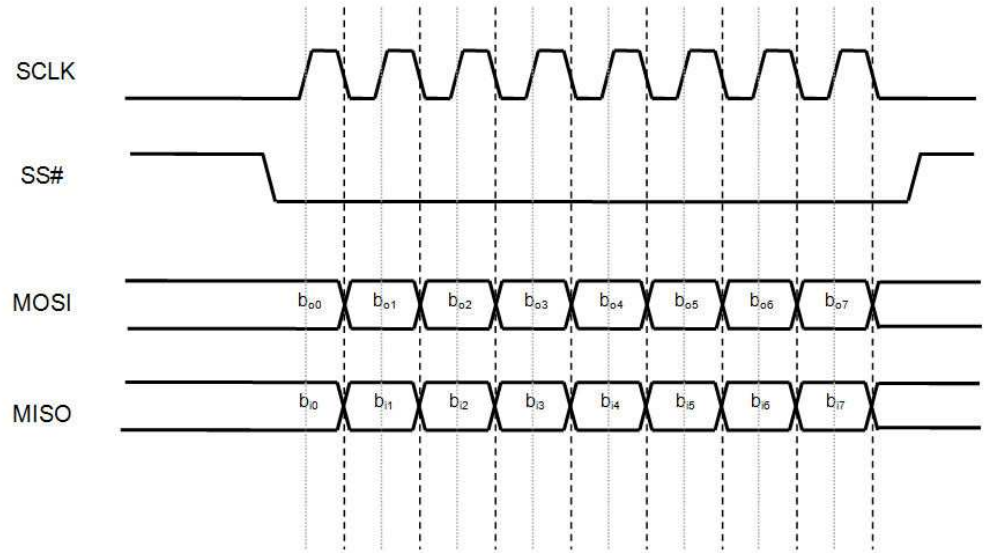


Figure 2 : A simple SPI communication. Data bits on MOSI and MISO toggle on the SCLK falling edge and are sampled on the SCLK rising edge. The SPI mode defines which SCLK edge is used for toggling data and which SCLK edge is used for sampling data.

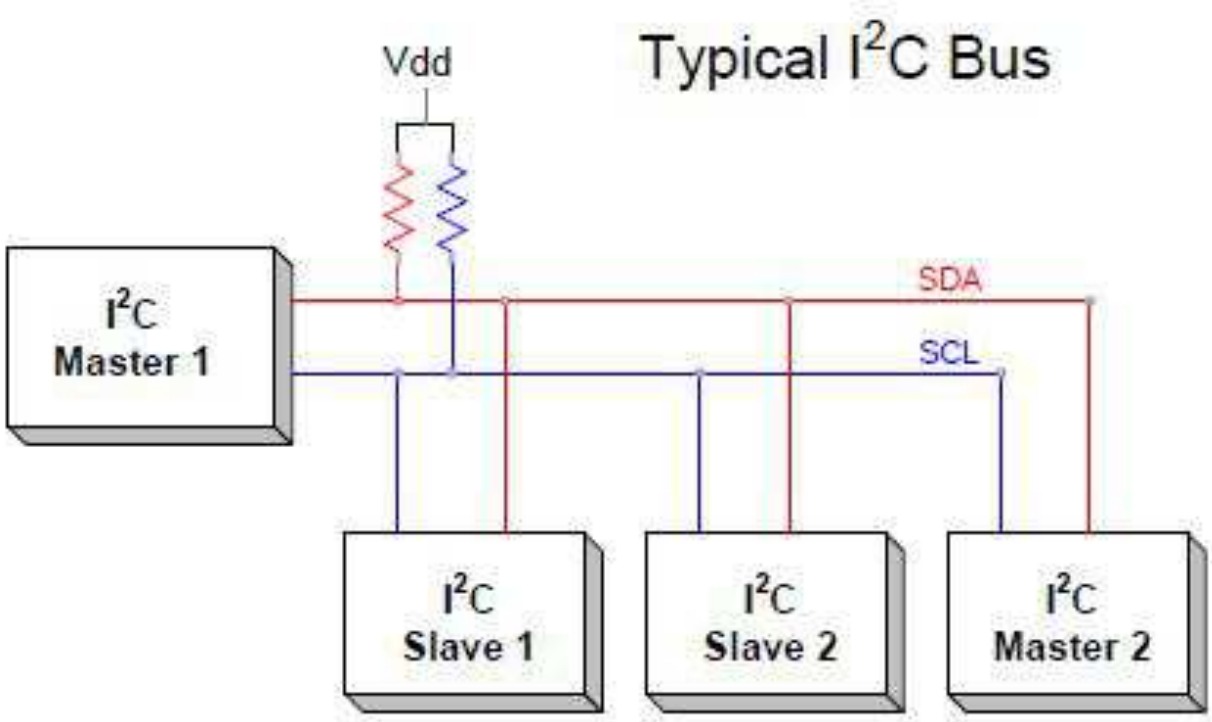
I^2C / TWI (Inter Integrated Circuit)

E' un sistema di comunicazione seriale bifilare utilizzato tra circuiti integrati.
Richiede un master e uno o piu' slave.
Linee

- SDA: Serial DAta Line, per i dati;
- SCL: Serial Clock Line, segnale di Clock (emesso dal master)

(Occorre poi anche una linea di GND comune e una linea di alimentazione per i pull-up).
Si possono collegare vari slave (ognuno con indirizzo diverso).
(Libreria Arduino: Wire library)

I^2C / TWI (Inter Integrated Circuit)-2



I^2C / TWI (Inter Integrated Circuit)-3

START	Slave address	Rd/nWr	ACK	Data	ACK	Data	ACK	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 1: writing 2 byte to a slave. The data put on the bus by the master are shaded.

START	Slave address	0	0	Data	0	Data	0	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 2: reading 2 bytes from a slave. The data put on the bus by the master are shaded.

START	Slave address	1	0	Data	0	Data	1	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Figure 5: Typical I²C transfer, with 2 bytes of data. The master initiates the transfer with a START condition, followed by the slave address and the transfer type (read or write) bit. The slave acknowledges its address. Each data byte is then transmitted and acknowledged by the receiver. When it receives data, the master can issue a not-acknowledge condition (NACK) when it has received enough data. The bus is released when the master issues a STOP condition.

Interrupts

La scheda Arduino Uno puo' gestire 2 fonti di interrupts

- Interrupt 0: Digital I/O 2;
- Interrupt 1: Digital I/O 3;

Quando arriva un interrupt viene eseguita una routine specificata. Alla fine il controllo torna al programma precedentemente in esecuzione.
Esempio:

```
void setup()
{
  Serial.begin(9600);
  attachInterrupt(0, myprog, CHANGE); //interrupt se I/O 2 cambia
}
void loop()
{
    ....
    ....
}
void myprog()
{
// Interrupt Service Routine
 Serial.println("E' arrivato un interrupt");
}
```

Interrupts:funzioni

`attachInterrupt(interrupt, ISR, mode);`

- *interrupt:* 0 *o* 1*;*
- *ISR: nome della routine di servizio;*
- *mode: LOW,CHANGE, RISING, FALLING*

`detachInterrupt(interrupt);`

Elimina l'interrupt precedentemente abilitato.

`noInterrupts();`

Disabilita tutti gli interrupts.

`interrupts();`

Abilita gli interrupts (precedentemente disabilitati).

Interrupts:caveat

Le routine a servizio degli interrupts hanno una particolare natura e specifiche peculiarità.
Leggere il manuale!

Esperienza 9: Familiarizzazione con Arduino

Esperienza 9: Semplici programmi con Arduino

- L'obiettivo di questa esperienza e' di familiarizzarsi con la scheda Arduino Uno, con il microcontrollore ATMEL ATMega328 e con il relativo software (vedi Appendice per maggiori dettagli).
- E' bene avere sempre disponibile per consultazione il sito internet di Arduino (http://www.arduino.cc) ed in particolare la pagina di reference della programmazione

  `https://www.arduino.cc/en/Reference/HomePage`
- I circuiti da utilizzare possono essere montati sulla consueta scheda sperimentale, connettendola ad Arduino Uno mediante opportuni ponticelli.

Il primo programma...

- "Hello, world!" (–> http://helloworldcollection.de/)
- Ricordarsi di far partire la *Finestra di Monitor* (in genere sotto tools/)

Comunicazione seriale

- Programmi di I/O verso la finestra di monitor.
- Misurare la velocita' di esecuzione del μC per varie istruzioni: operazioni aritmetiche, funzioni, operazioni di Input/Output.
- Funzioni di timer: `millis()` e `micros()`
- `delay()` in millisecondi

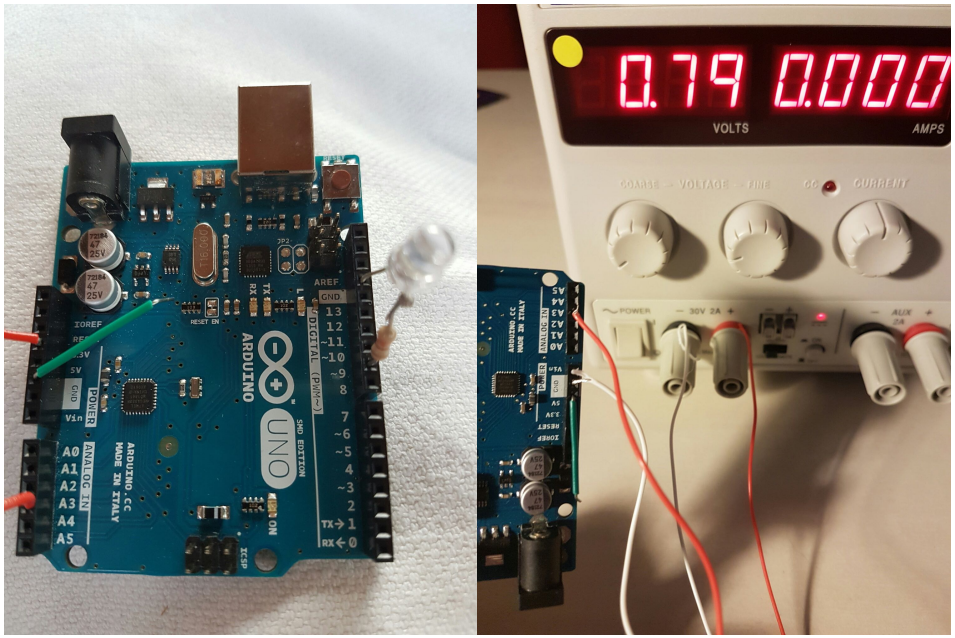
serial_io §

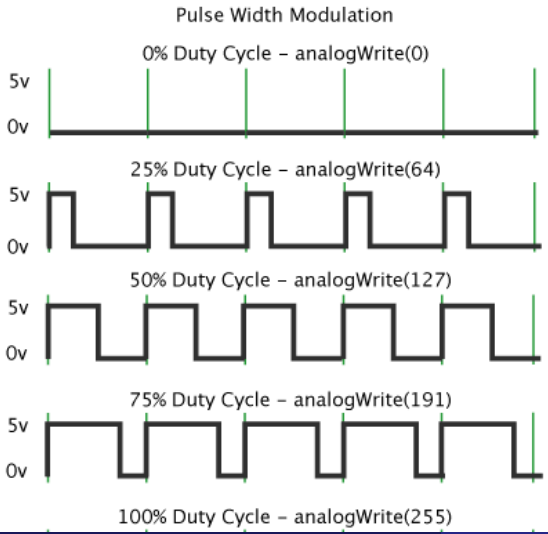
```
long int data=0;
long unsigned t0, t1; //time...

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  Serial.println("Here we go!");
  Serial.println(" ");
}

void loop() { // run over and over
  if (Serial.available() > 0) {
    data = Serial.parseInt();
    t0=micros();
//  | Serial.print("Hai scritto il numero ");
    Serial.println(data);
//    Serial.print("E questo e' moltiplicato per 10: ");
    Serial.println(data*10);
    t1=micros();
    Serial.print("durata: ");
    Serial.print(t0);
    Serial.print(" ");
    Serial.print(t1);
    Serial.print(" ");
    Serial.println(t1-t0);
    delay(100);
  }
}
```

Operazioni di I/O





- Leds per visualizzazione degli outputs
- Input forniti da generatore triplo o dai 5v (3.3V) di Arduino
- `analogRead(analogPin)`, `analogWrite(pin, value)`
- analogWrite usa modulazione PWM (*Pulse Width Modulation*)

AnalogWrite()

analog_write

```
int ledPin = 9;
int val = 128;

void setup() {
pinMode(ledPin, OUTPUT);
Serial.begin(9600);
Serial.println("Light your fire...");
Serial.println(" ");
}

void loop() {
  if (Serial.available() > 0) {
    val = Serial.parseInt();
 // check value in
    if (val >= 0 && val <= 255) {
     analogWrite(ledPin, val);
     Serial.print("Scritto valore :");
     Serial.println(val);
     }
    else {
     Serial.println("Valore fuori range (0:255)!!!");
     }
  }
}
```

Led_Rampa

```
int ledPin = 9;        // LED connected to digital pin 9
int val = 0;          // val: 0-255 (duty cycle)
int i = 0;
int millisecondi = 10;
void setup()
{
  Serial.begin(9600);
  Serial.println("This is an Arduino-based dimmer...");
  pinMode(ledPin, OUTPUT);
}
void loop()
{
for (i=0;i<255;i++)
  {
  val = i;
  analogWrite(ledPin, val);
  delay(millisecondi);
  }
for (i=0;i<255;i++)
  {
  val = 255-i;
  analogWrite(ledPin, val);
  delay(millisecondi);
  }
}
```

`AnalogRead()`

analog_input

```
int analogPin = 3; // analog pin 3
int RdVal = 0;
float Vx = 0.0;
void setup()
{
analogReference(DEFAULT); // Arduino 5 V
Serial.begin(9600);
Serial.println("Arduino Voltage meter");
Serial.println(" ");

void loop ()
{
RdVal = analogRead(analogPin);
Vx = float(5.0)*(float(RdVal)/float(1023));
delay(1000);
//Serial.print(RdVal);
Serial.print(" Tensione letta: ");
Serial.println(Vx);
}
```

Output analogico

- Verificare (oscilloscopio) le forme d'onda ottenute sui pin di Output Analogico (uno tra quelli a 490 Hz e uno tra quelli a 980 Hz)

Input analogico

1. Calibrazione dell'ADC del μC, usando un ingresso analogico.
   - Utilizzare come Analog Reference il default (5 V) e come tensione da convertire l'uscita dell'alimentatore triplo (0 - 5V).
   - Costruire il grafico della calibrazione.
2. Tensione sinusoidale e un'onda triangolare in ingresso.
   - Scrivere i risultati su una tabella in modo da poter poi fare un grafico con open Office e verificare la fedelta' del risultato ottenuto rispetto all'onda di partenza

ADC

ADC_1

```
int data;
int analogPin = 3;
int i = 0;
long unsigned t0, t1; //time...

const long interval = 2; // 2ms --> sample at 500Hz

void setup() {
  Serial.begin(9600);
  Serial.println("ADC");
}

void loop() {
  t0 = millis();
  for (i = 0; i < 100; i++) { // 100 samples...
    my_delay(interval);
    data = analogRead(analogPin);
    Serial.println(data);
  }
  t1 = millis();
  Serial.print("Durata: ");
  Serial.println(t1 - t0);
  delay(1000);
}

void my_delay(unsigned long interval)
{
  unsigned long MillisStart = millis();
  unsigned long currentMillis = MillisStart;
  //timer
  while (currentMillis - MillisStart <= interval) {
    currentMillis = millis();
  }
}
```