

# Outline

- Introduzione
- Breve introduzione alla piattaforma ClearSy
- Descrizione componenti e ciclo di sviluppo
- Breve introduzione ad Arduino
- Descrizione del sistema da realizzare



Link al materiale

# Organizzazione delle esercitazioni

- Il progetto sarà sviluppato in 4 incontri:  
20 maggio - 14:00-17:00 (presentazione dell'esercitazione e del materiale necessario)  
23 maggio - 14:00-16:00 (sviluppo)  
30 maggio - 14:00-16:00 (sviluppo)  
6 giugno - 14:00-16:00 (prove di integrazione e risultati)
- L'obiettivo dell'esercitazione è la realizzazione di un sistema industriale, a sua volta suddiviso in più sottosistemi
- Ogni gruppo lavorerà in maniera indipendente sviluppando un sottosistema che verrà integrato e testato con quelli degli altri gruppi

# Organizzazione delle esercitazioni

- Il progetto ha come primo scopo quello di vedere una semplice applicazione embedded, programmata in modo host-target
- Delle due piattaforme, una integra una nutrita serie di soluzioni specifiche per la safety. Il secondo scopo è quindi quello di studiare queste soluzioni
- Ogni gruppo lavorerà in maniera indipendente sviluppando il progetto su una delle due piattaforme assegnate (in modo casuale)
- Terzo scopo è di valutare la difficoltà comparata nell'utilizzo delle due piattaforme

# Organizzazione delle esercitazioni

- Le piattaforme proposte sono ClearSy SK0 e Arduino Due
- Verranno forniti i link per il download degli IDE necessari e tutorial
- Durante l'ultimo incontro presenterete risultati ottenuti e difficoltà riscontrate (in fase di sviluppo e integrazione) e ne discuteremo insieme

# La piattaforma ClearSy

La piattaforma ClearSy è costituita da una scheda elettronica e un IDE di sviluppo software (CSSP).

« Only inactive sequences can be added to the active sequences execution queue. »

Natural language  
requirement

```
activation_sequence = /* Activation d'une séquence non active */
PRE ~(sequences = sequences_actives) THEN
  ANY sequ WHERE
    sequ ∈ sequences - sequences_actives
  THEN
    sequences_actives := sequences_actives U {sequ}
  END
END;
```

```
vation_sequence = /* Activation d'une séquence non active */
sequ IN
sequ <-- indexSequenceInactive;
activeSequence(sequ)
```

```
0_activation_sequence(void)
X_SEQUENCES sequ:
sequence_manager_indexSequenceInactive((sequ);
sequence_manager_activeSequence(sequ);
```

```
F970 FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
F980 83C6 0C8D 1485 0000 0000 8D42 0883 F807
F990 7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
F9A0 83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3
```

B Specification

Proof (coherence)

Proof (refinement)

B Implementation

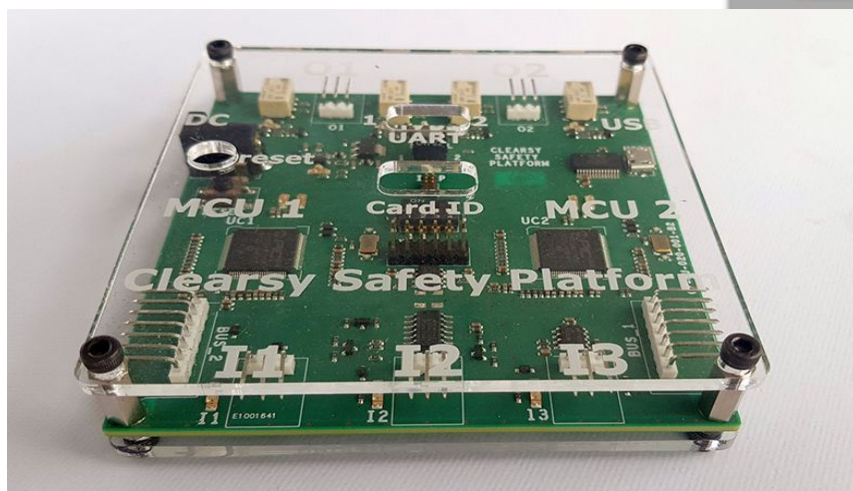
Proof (coherence)

C generated code

Binary code

B

+





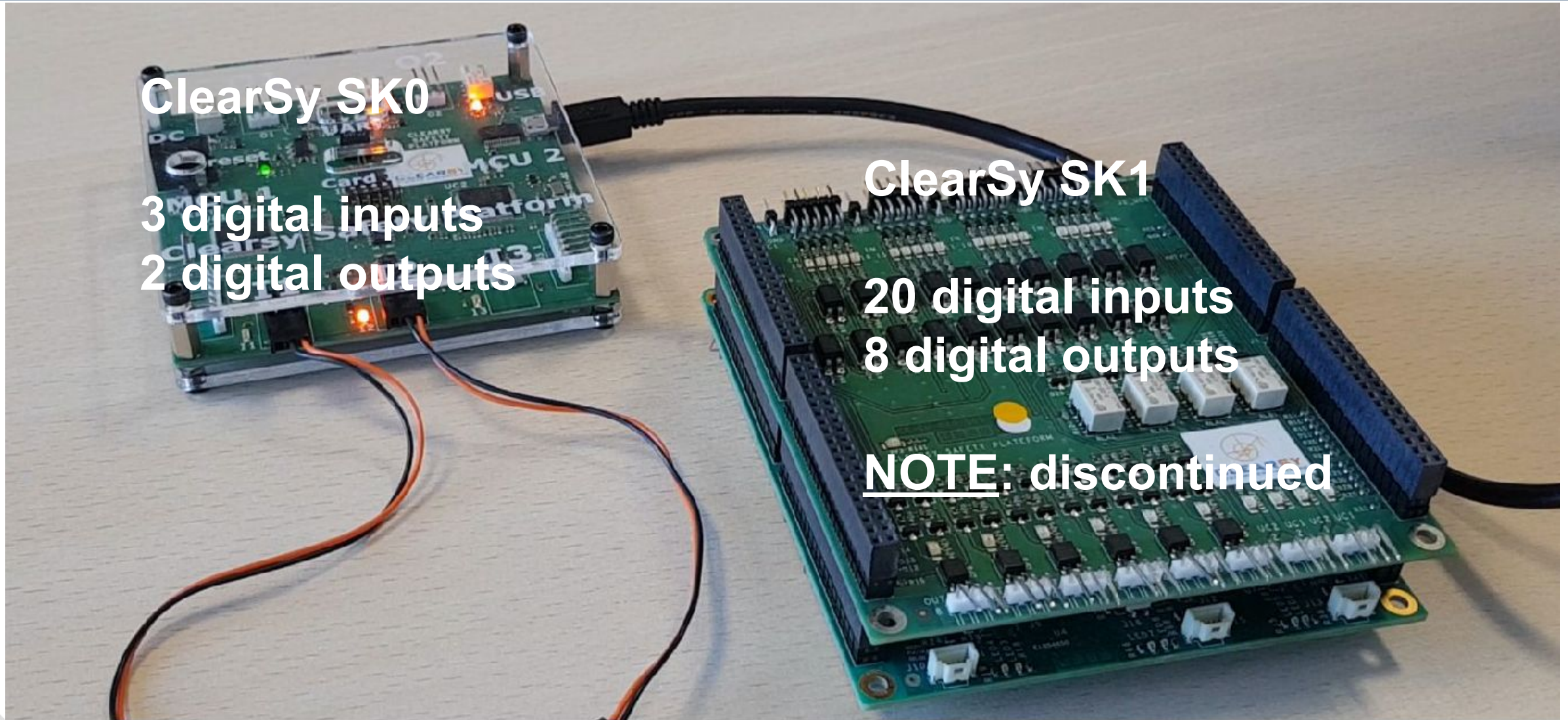
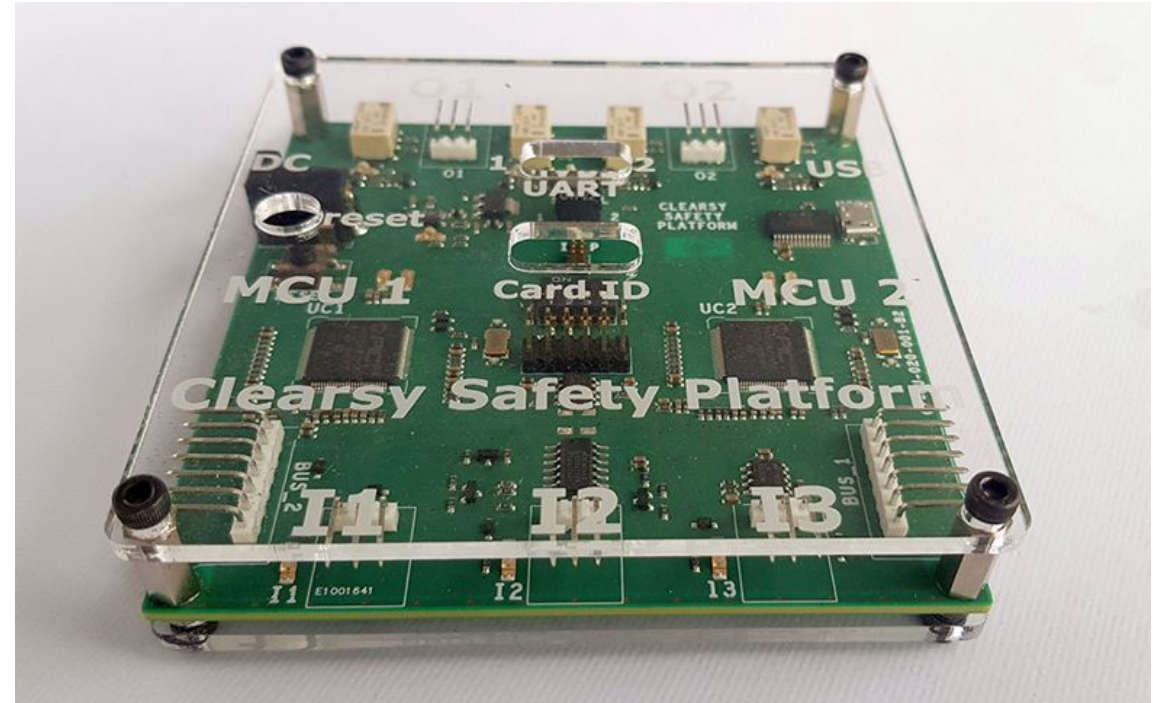


Figure 2.2: Starter kits SK<sub>0</sub> (left) and SK<sub>1</sub> (right)

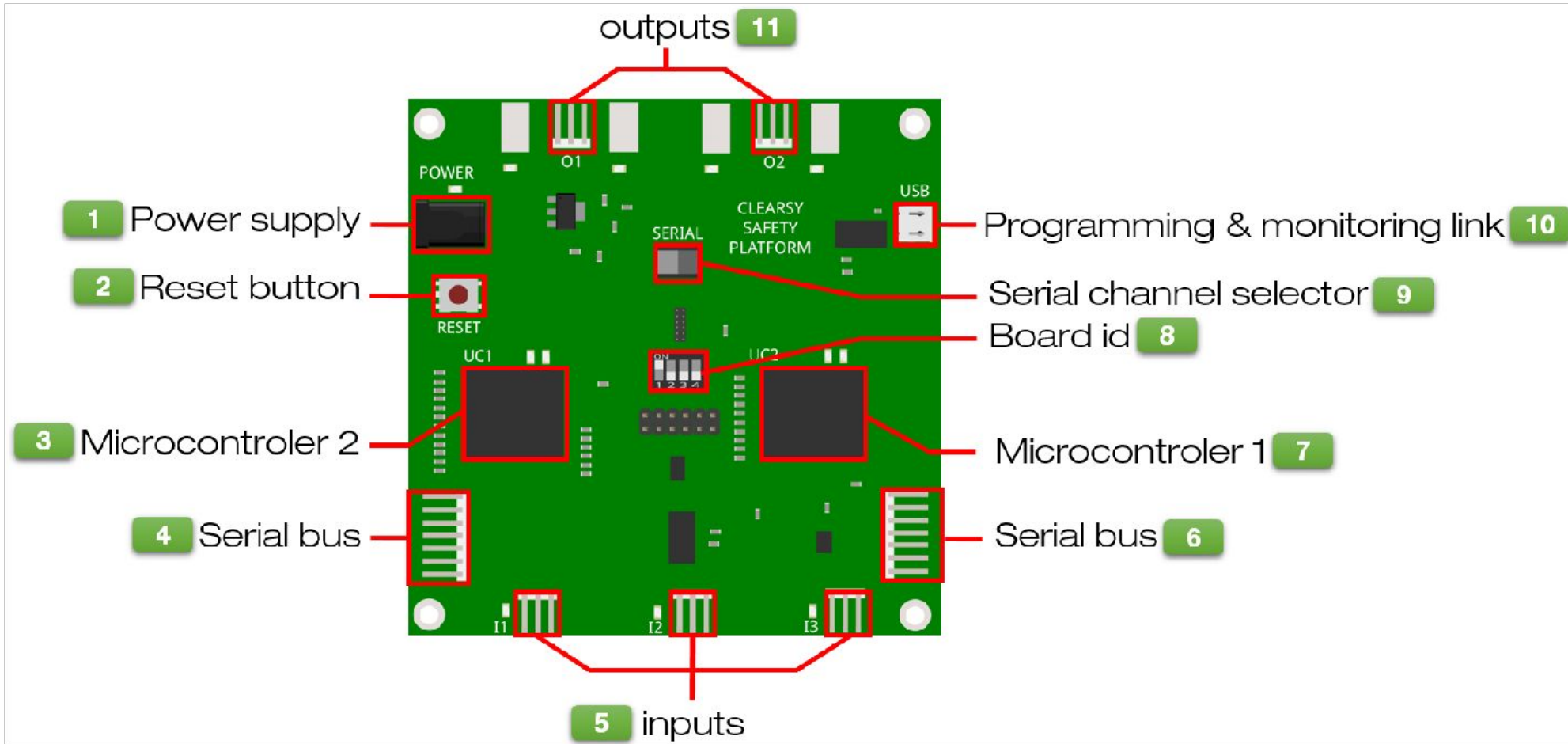
# ClearSy Safety Platform

- La scheda ClearSy SK0 ha 3 digital inputs e 2 digital outputs
- Ha due microprocessori PIC32
- È fornita di una porta USB

*ClearSy permette l'integrazione di metodi formali, generazione e compilazione di codice ridondanti e una piattaforma hardware che garantisce un'esecuzione sicura del software*



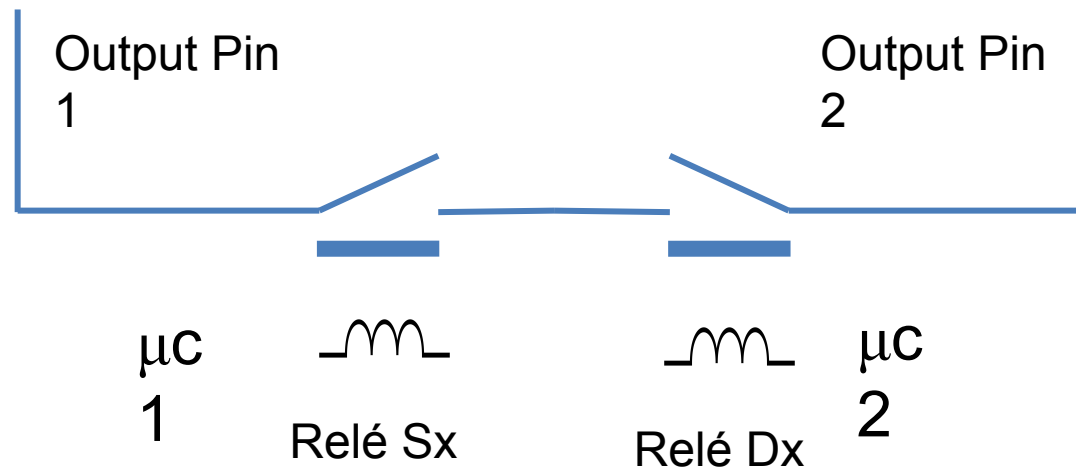




2 PIC32MX795F512L microcontrollers with 512KB Flash and 128 KB RAM, delivering 105 DMIPS at 80MHz.



# Uscita digitale «votata»



L'uscita è isolata  
dalla scheda

I due  $\mu C$  devono essere d'accordo nell'azionare il relè

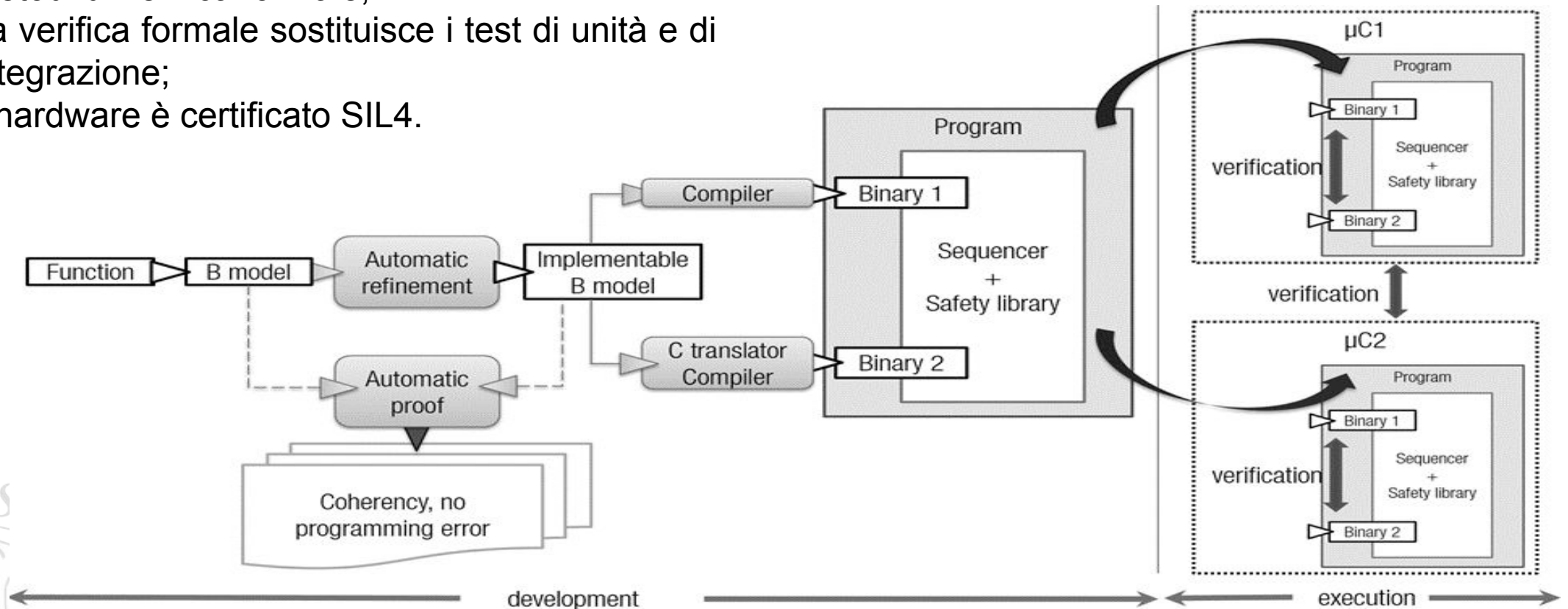
SK0: due uscite votate

# E i guasti software...?

- Metodo Formale B: raffinamento con prove di consistenza e correttezza formale (theorem proving)
- Safety Belt: run-time executive che include comparazione e verifica CRC
- Diversità a vari livelli

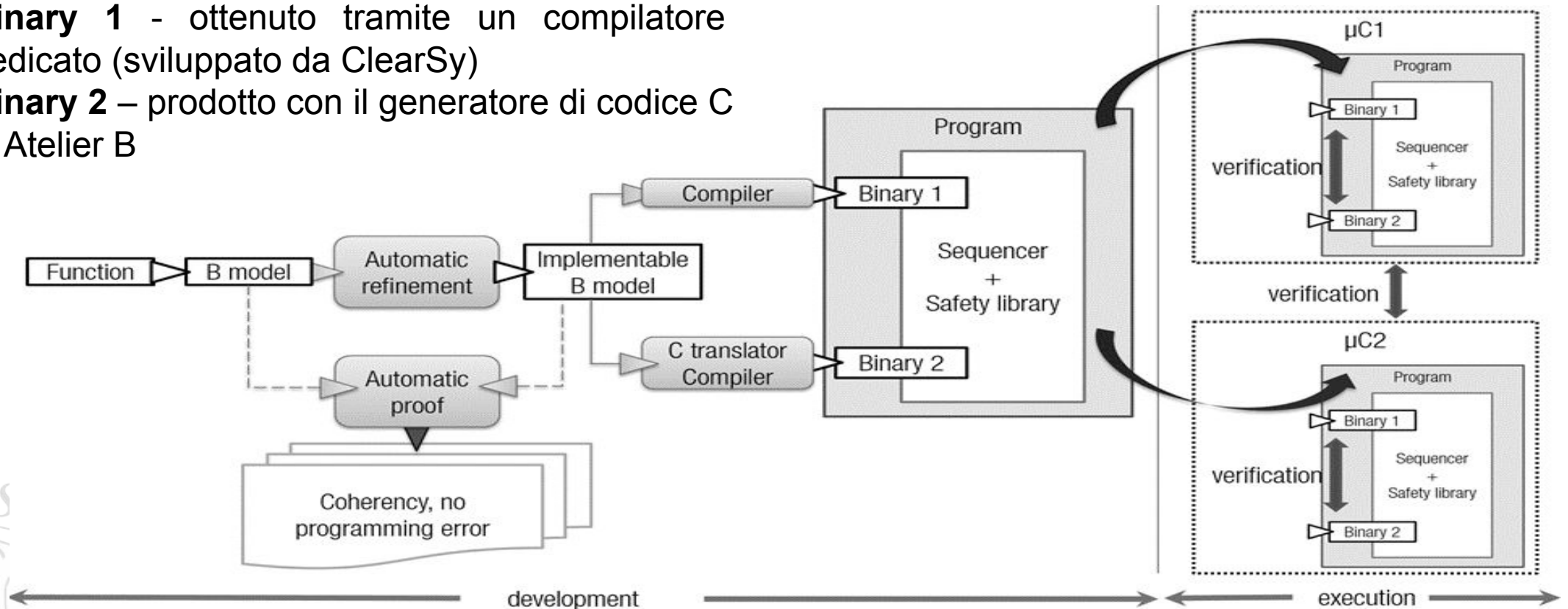
# Architettura

- ClearSy Safety Platform:
  - Copre l'intero ciclo di sviluppo;
  - Si basa su un linguaggio formale (B) e su metodi di verifica formale;
  - La verifica formale sostituisce i test di unità e di integrazione;
  - L'hardware è certificato SIL4.

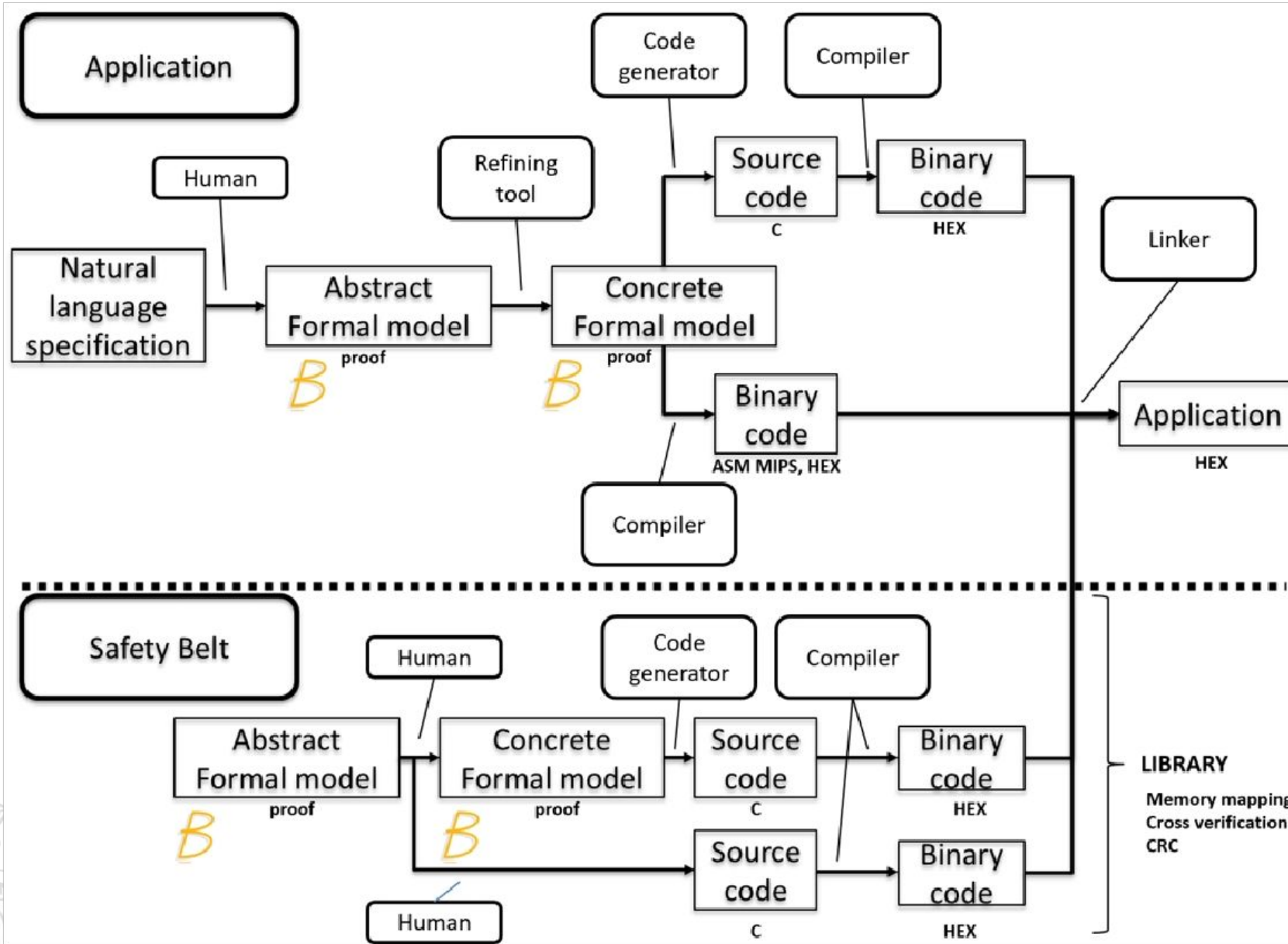


# Architettura

- Nella fase “Automatic proof” i modelli B vengono dimostrati (provati)
- Dal modello implementabile, vengono generati due binari:
  - **Binary 1** - ottenuto tramite un compilatore dedicato (sviluppato da ClearSy)
  - **Binary 2** – prodotto con il generatore di codice C di Atelier B







# Software Modeling

*E' necessario scrivere:*

- Una specifica;
- Un'implementazione;

*Il linguaggio usato sarà lo stesso per entrambi e sarà il linguaggio B.*

*La specifica contiene la struttura del modello*

*L'implementazione contiene invece le operazioni che devono essere compiute nel corso dell'esecuzione.*

*La specifica e l'implementazione sono verificati.*

*Se la correttezza è verificata viene generato il source code.*

« Only inactive sequences can be added to the active sequences execution queue. »

```
activation_sequence = /* Activation d'une séquence non active */
PRE ¬(sequences = sequences_actives) THEN
  ANY sequ WHERE
    sequ ∈ sequences - sequences_actives
  THEN
    sequences_actives := sequences_actives U {sequ}
  END
END;
```

```
activation_sequence = /* Activation d'une séquence non active */
VAR sequ IN
  sequ <-- indexSequenceInactive;
  activeSequence(sequ)
END;
```

```
void M0_activation_sequence(void)
{
  CTX_SEQUENCES sequ;

  sequence_manager_indexSequenceInactive(&sequ);
  sequence_manager_activeSequence(sequ);
}
```

0x01F970	FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980	83C6 0C8D 1485 0000 0000 8D42 0883 F807
0x01F990	7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0	83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3

Natural language  
requirement

B Specification

Proof (coherence)

Proof (refinement)

B Implementation

Proof (coherence)

C generated code

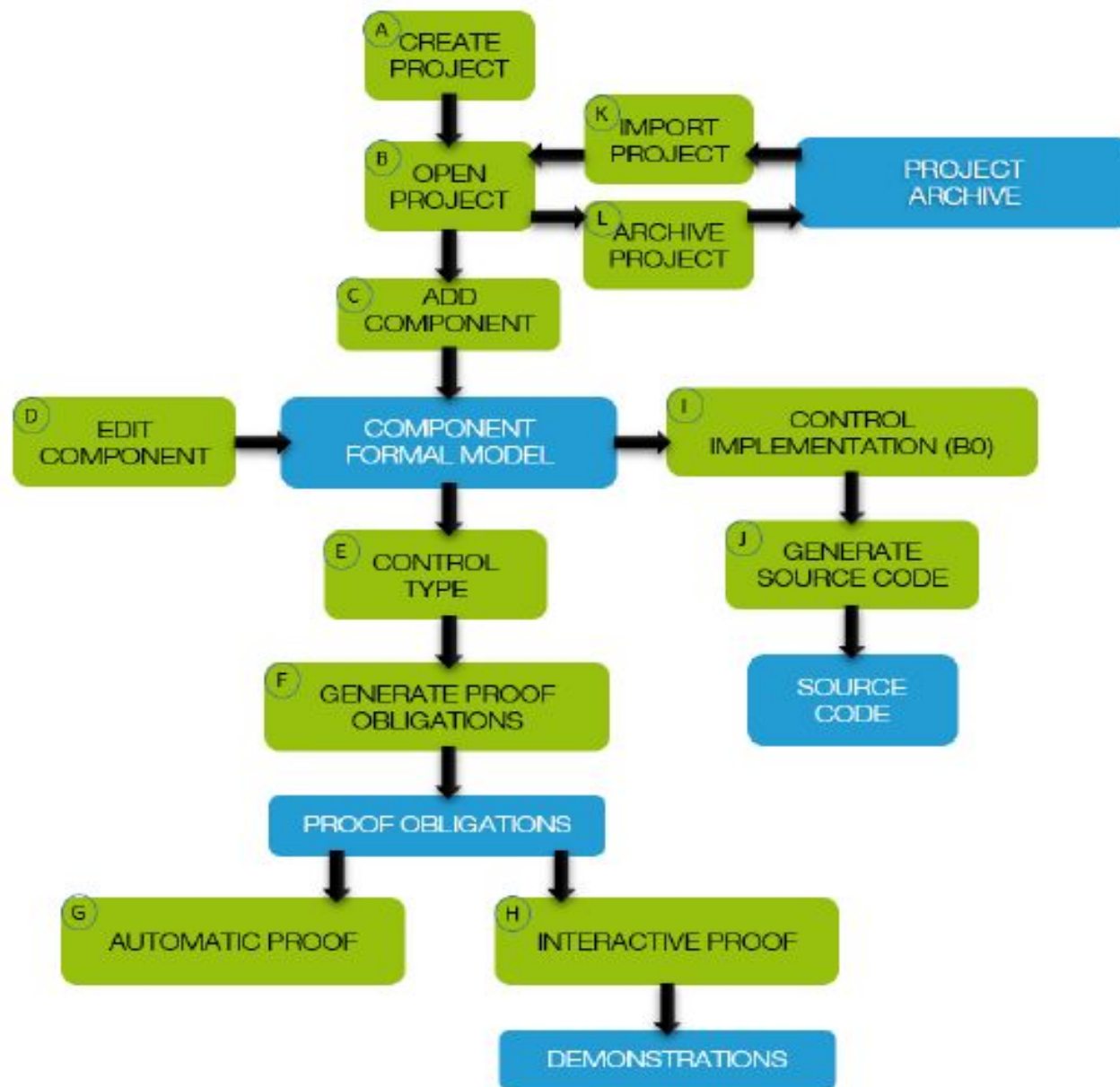
Binary code

B

+



# Project steps







# Acronimi e definizioni

*PO = Proof Obligation*

*POG = Proof Obligation Generator*

*INVARIANTE = Proprietà che deve essere sempre verificata nel corso dell'esecuzione.*



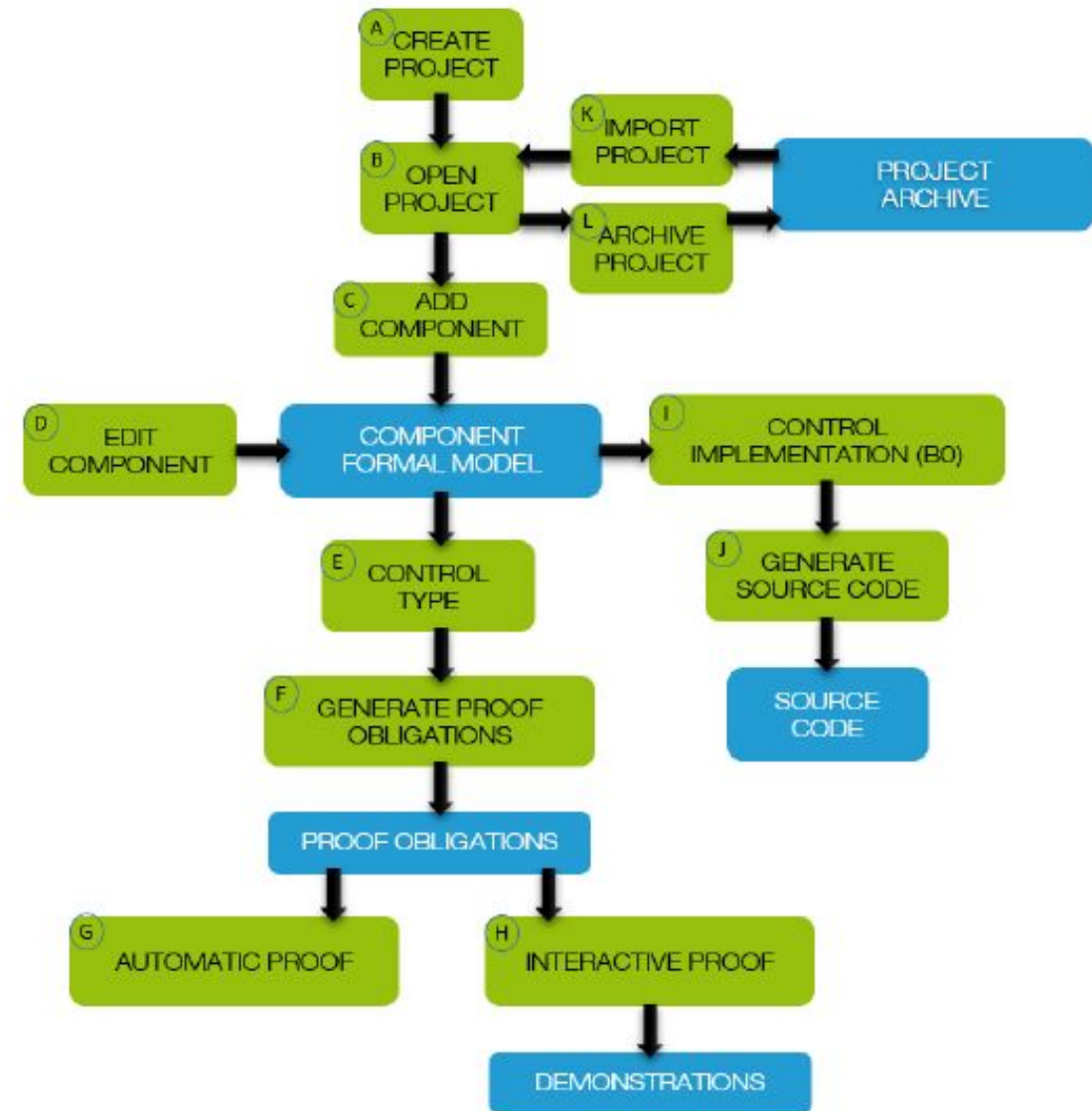
# Project creation

*E' necessario scrivere*

- Creare un nuovo progetto;
- Selezionare «Software development» e «Define as CSSP project»;
- Dare un nome al nuovo progetto;

*Il progetto viene creato e popolato sulla base della configurazione che abbiamo scelto per la nostra scheda in fase di installazione.*

*Per semplici applicazioni non c'è bisogno di scrivere nuovi componenti.*



# Componenti e architettura del progetto

*I componenti autogenerati sono:*

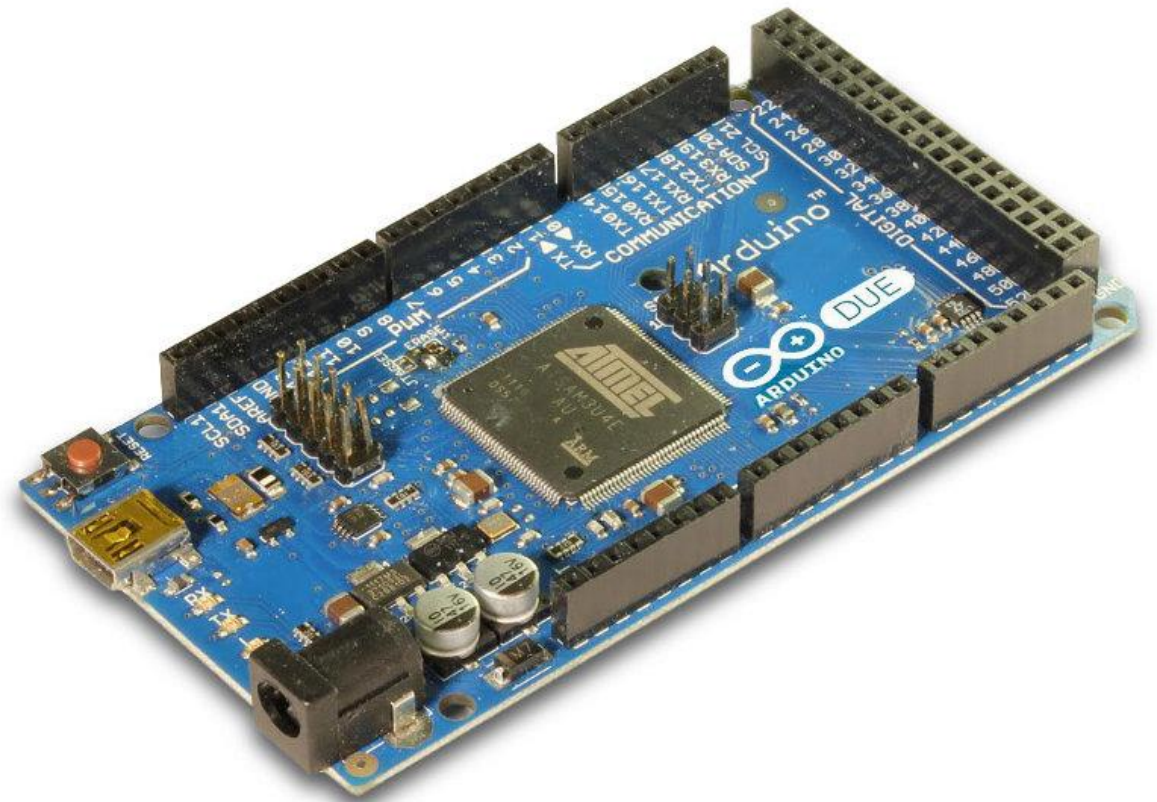
- **user\_ctx**: in cui è possibile aggiungere nuove costanti (CONCRETE\_CONSTANTS) a cui è possibile assegnare proprietà (PROPERTIES);
- **user\_ctx\_l**: che permette di fornire i valori alle costanti definite;
- **user\_logic**: in cui è possibile modificare la specifica delle operazioni definendo la logica vera e propria. Possono essere definiti invarianti (INVARIANT) e l'inizializzazione di variabili (INITIALISATION);
- **user\_logic\_l**: permette la definizione di variabili (CONCRETE\_VARIABLES) con la relativa inizializzazione (INITIALISATION), la definizione di invarianti (INVARIANT) e la definizione delle operazioni che devono essere compiute durante il ciclo di esecuzione (OPERATIONS).

*Una volta definiti i componenti è necessario passare alla prova formale dei modelli che può essere effettuata in maniera automatica (selezionando F0) o interattiva.*

*Una volta che il progetto è verificato è possibile generare il codice per la scheda e caricarlo.*

# La piattaforma Arduino

Arduino è una piattaforma hw. Ideata e sviluppata nel 2005 da Interaction Design Institute di Ivrea come strumento per la prototipazione rapida e per scopi hobbistici, didattici e professionali. Il nome deriva da quello del bar di Ivrea frequentato dai fondatori del progetto, nome che richiama a sua volta quello di Arduino d'Ivrea, Re d'Italia nel 1002.

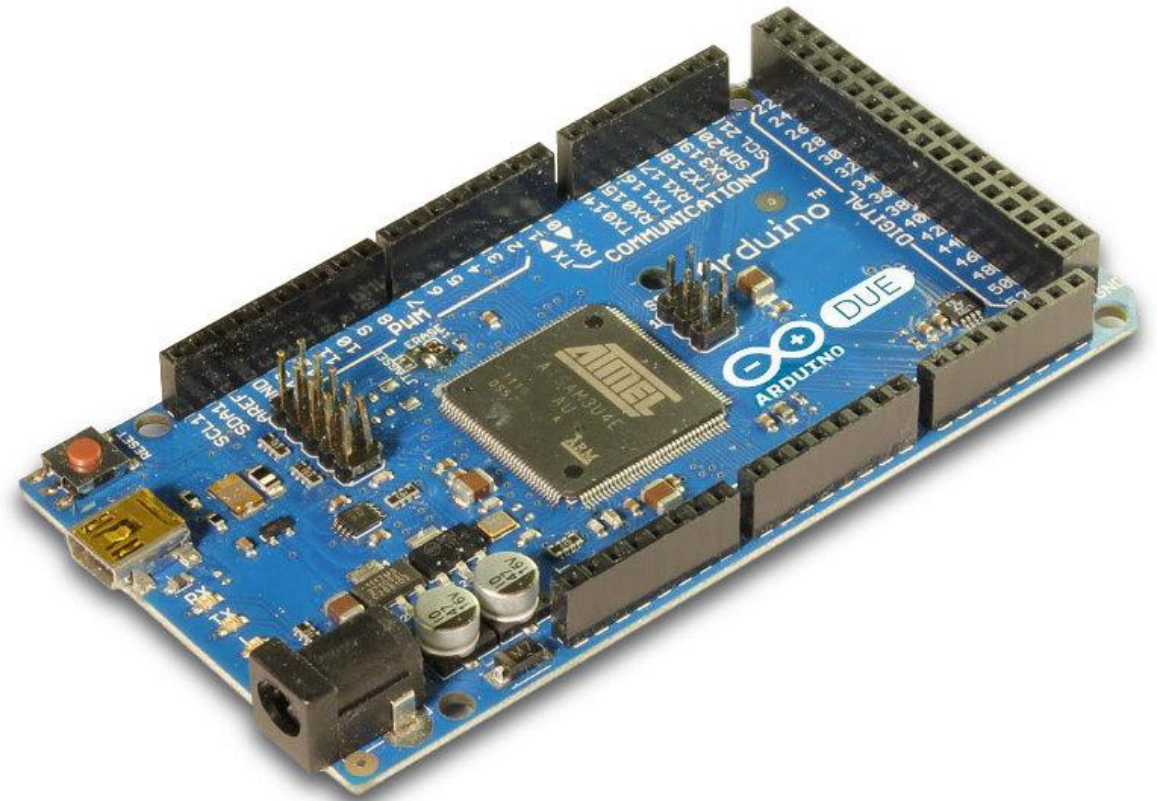




# La piattaforma Arduino

Con Arduino è facile realizzare dispositivi come controllori di luci, automatismi per il controllo della temperatura e dell'umidità etc..

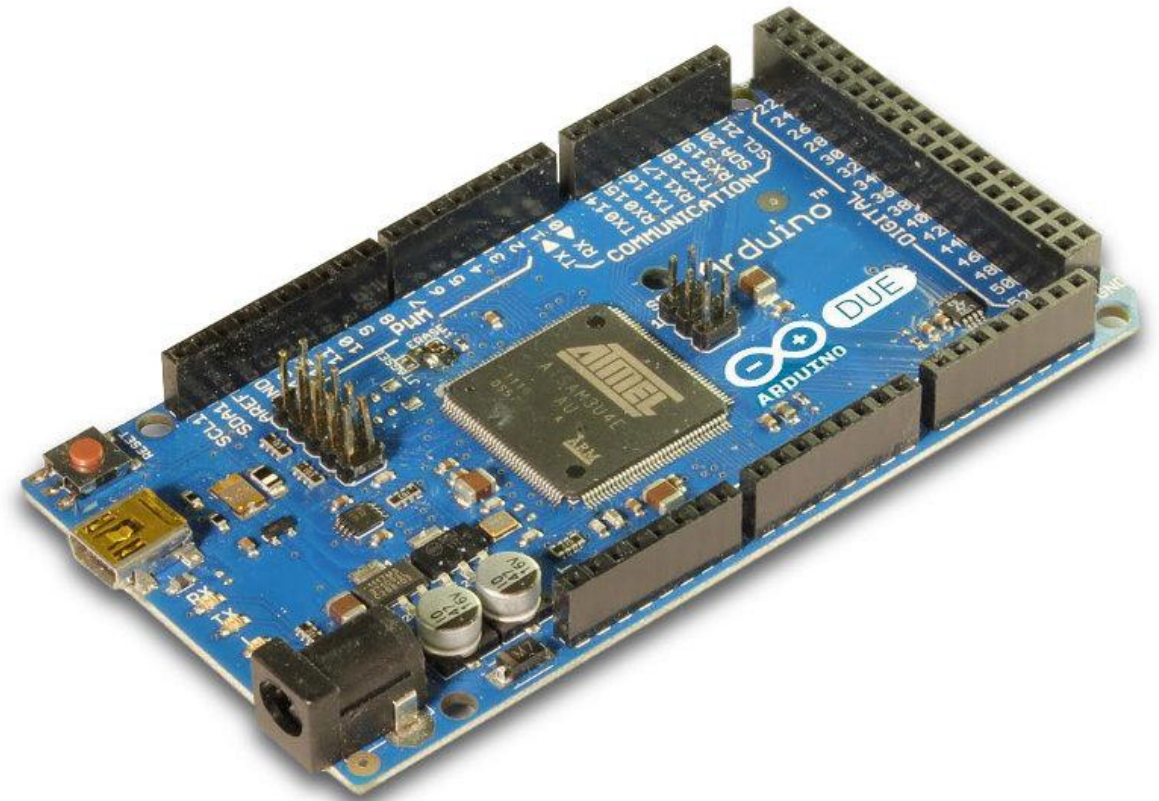
La scheda è abbinata a un IDE per la programmazione del microcontrollore. Tutto il software a corredo è libero, e gli schemi circuitali sono distribuiti come hardware libero.



# La piattaforma Arduino

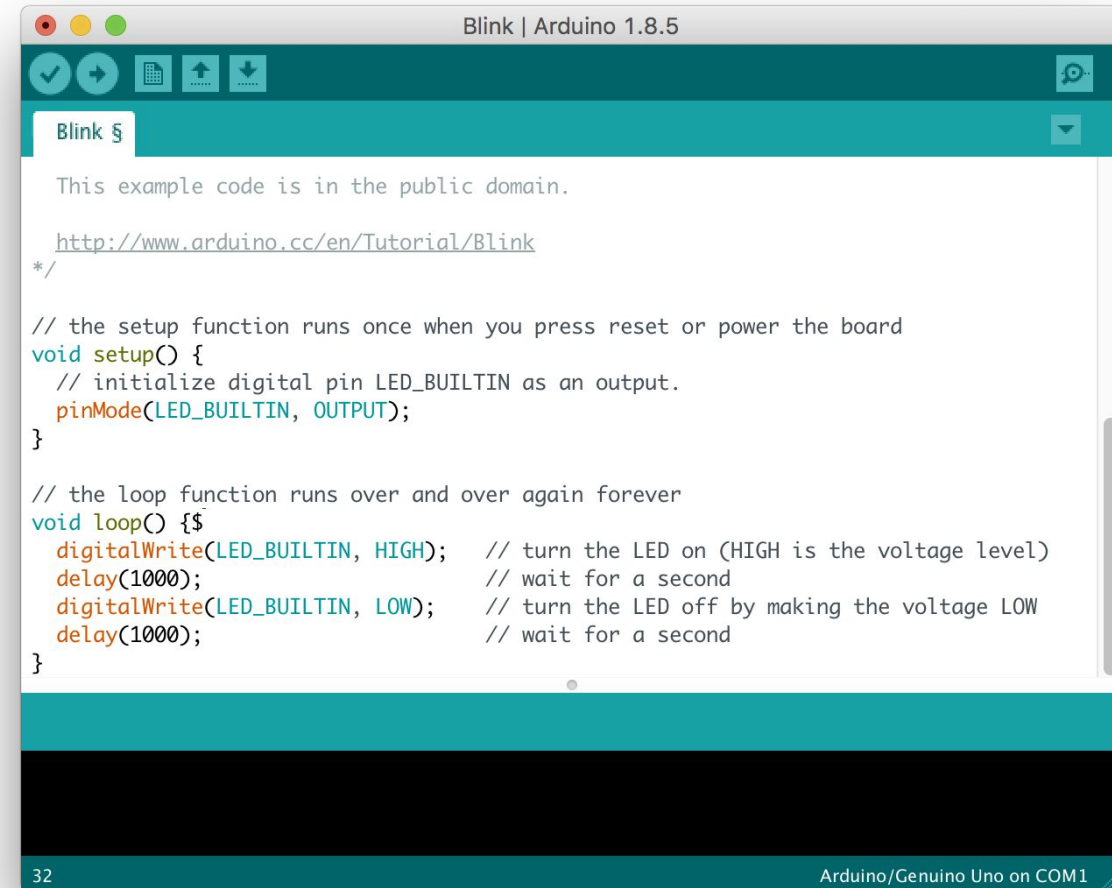
Esistono tantissime schede Arduino, differenti per microcontrollore (AVR o ARM), dimensioni, numero di pin...

Le schede hanno pin digitali (GPIO), ingressi analogici (ADC), uscite PWM, porte di comunicazione (seriale, SPI, I2C...).



# Arduino IDE

- Consente di installare i tool di supporto (compilazione) per le schede Arduino (di default supporta tutti i micro AVR)
- Consente di installare librerie di supporto già esistenti (sensori e attuatori, connettori...)
- Definire un nuovo progetto (file .ino)
  - Sviluppato in C/C++
  - Ogni progetto deve avere due funzioni:
    - setup: inizializzazione del programma
    - loop: ciclo di main chiamato continuamente
- Compilare un progetto
- Caricare un binario sulla scheda
- Collegarsi con la porta seriale (serial monitor)



The screenshot shows the Arduino IDE window titled "Blink | Arduino 1.8.5". The interface includes a toolbar with icons for file operations and a menu bar. The main text area displays the Blink example code, which is in the public domain. The code defines the setup and loop functions to blink the built-in LED. The status bar at the bottom indicates "32" and "Arduino/Genuino Uno on COM1".

```
This example code is in the public domain.  
  
http://www.arduino.cc/en/Tutorial/Blink  
*/  
  
// the setup function runs once when you press reset or power the board  
void setup() {  
  // initialize digital pin LED_BUILTIN as an output.  
  pinMode(LED_BUILTIN, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)  
  delay(1000); // wait for a second  
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW  
  delay(1000); // wait for a second  
}
```

<https://arduino.cc/en/software>

<https://docs.arduino.cc/software/ide/#ide-v2>

# Progetto per l'esercitazione

- *Individuare un sistema di semplice implementazione, per i tempi ristretti a disposizione*
- *Funzionalità limitata dal numero di ingressi/uscite della piattaforma ClearSy*
- *Esempio comunque realistico*

→ **PWM Detector + 2003 Voter**



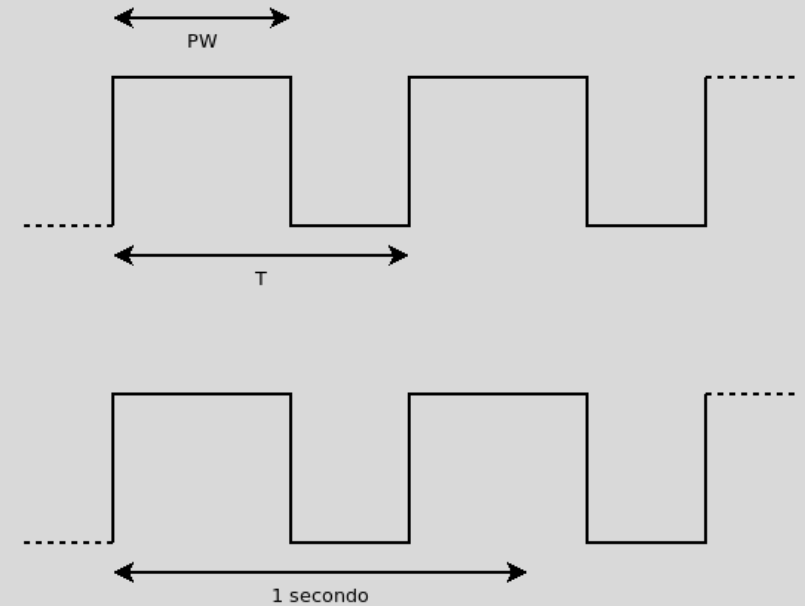
# Pulse Width Modulation (PWM)

Segnale a onda quadra ripetibile (periodico) con due fronti (edge)

- Fronte di salita (rising edge)
- Fronte di discesa (falling edge)

Due modi diversi (ma equivalenti) di descrivere una PWM

- Definire la durata temporale del periodo ( $T$ ) e la larghezza dell'impulso (PW oppure  $T_{ON}$ )  
→ **Meno usato, ma preferibile negli algoritmi**
- Definire la frequenza ( $f = 1/T$ ) come il reciproco del periodo (numero di periodi in un secondo) e il duty cycle ( $d = PW/T$ ) come la percentuale di attivazione del segnale  
→ **Più usato**

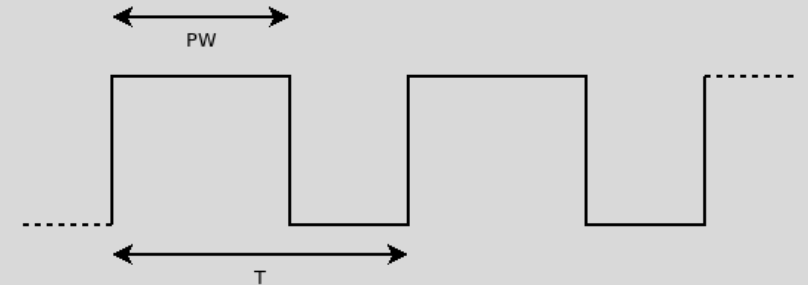


# PWM Detector

## Come si riconosce se ho avuto un fronte (di salita o di discesa)?

Lo stato corrente del segnale è cambiato rispetto al precedente

Abbiamo un fronte di salita se lo stato corrente è alto e precedente basso, di discesa altrimenti



## Come si riconosce una PWM?

1. Il tempo fra due fronti di salita (o due di discesa) deve essere uguale al periodo  $T$
2. Il tempo fra un fronte di salita e quello di discesa deve essere uguale alla larghezza dell'impulso  $PW$

## Semplice, giusto?

Sì... ma no. Questo algoritmo funziona nel mondo ideale, ma siamo ingegneri e il mondo ideale non esiste

1. Qualsiasi forma d'onda viene GENERATA sulla base di un clock che
  - 1.1. Non è perfetto → Genera una frequenza “vicina” a quella nominale
  - 1.2. Non è costante → La frequenza generata cambia con parametri ambientali, ad es. temperatura
2. Qualsiasi forma d'onda viene RICONOSCIUTA sulla base di un clock che
  - 2.1. Non è perfetto → Genera una frequenza “vicina” a quella nominale
  - 2.2. Non è costante → La frequenza generata cambia con parametri ambientali, ad es. temperatura

**DOBBIAMO CONSIDERARE LE TOLLERANZE**

## PWM Detector (*continua*)

### Come si riconosce una PWM (con le tolleranze)?

$f$  = frequenza nominale

$f_{\min} = f - \Delta f$  = frequenza minima da riconoscere

$d$  = duty cycle nominale

$d_{\min} = d - \Delta d$  = duty cycle minimo da riconoscere

$T_{\min} = 1/f_{\max}$  = minimo periodo da riconoscere

$T_{\max} = 1/f_{\min}$  = massimo periodo da riconoscere

$PW_{\min} = T_{\min} * d_{\min}$  = minima larghezza dell'impulso da riconoscere

$PW_{\max} = T_{\max} * d_{\max}$  = massima larghezza dell'impulso da riconoscere

$\Delta f$  = variazione sulla frequenza nominale

$f_{\max} = f + \Delta f$  = frequenza massima da riconoscere

$\Delta d$  = variazione sul duty cycle nominale

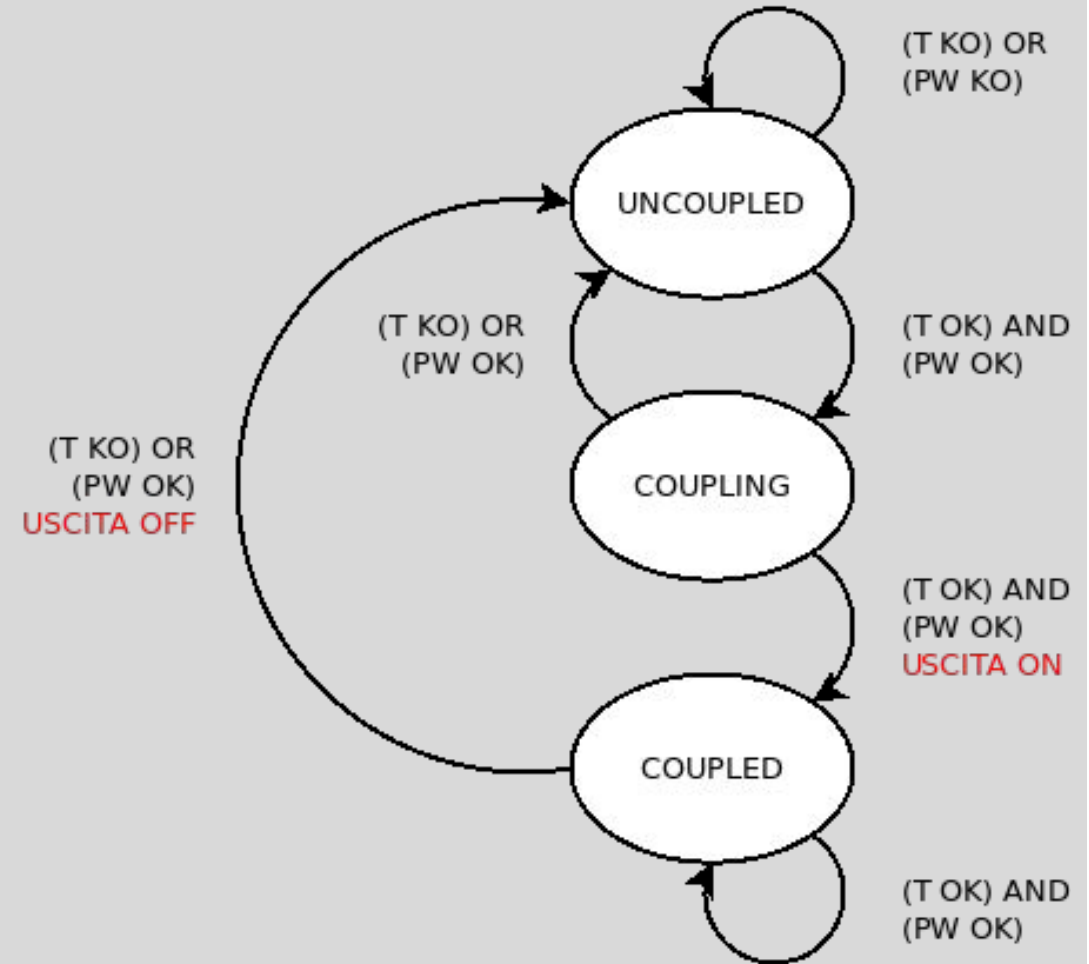
$d_{\max} = d + \Delta d$  = duty cycle massimo da riconoscere

1. Il tempo fra due fronti di salita (o due di discesa) deve essere compreso fra  $[T_{\min}, T_{\max}]$
2. Il tempo fra un fronte di salita e quello di discesa deve essere compreso fra  $[PW_{\min}, PW_{\max}]$

# PWM Detector (*continua*)

## PWM Detector e FSM

- UNCOUPLED: attesa di T valido con PW valido, quando succede ci spostiamo in COUPLING perché vogliamo essere sicuri che la frequenza sia valida
- COUPLING: se T o PW sono invalidi torniamo in UNCOUPLED, altrimenti attendiamo T valido con PW valido per andare in COUPLED. Sulla transizione verso COUPLED impostiamo l'uscita alta (ON)
- COUPLED: restiamo nello stato finché T valido con PW valido. Se T o PW sono invalidi torniamo in UNCOUPLED. Sulla transizione verso UNCOUPLED impostiamo l'uscita bassa (OFF)





## 2oo3 Voter

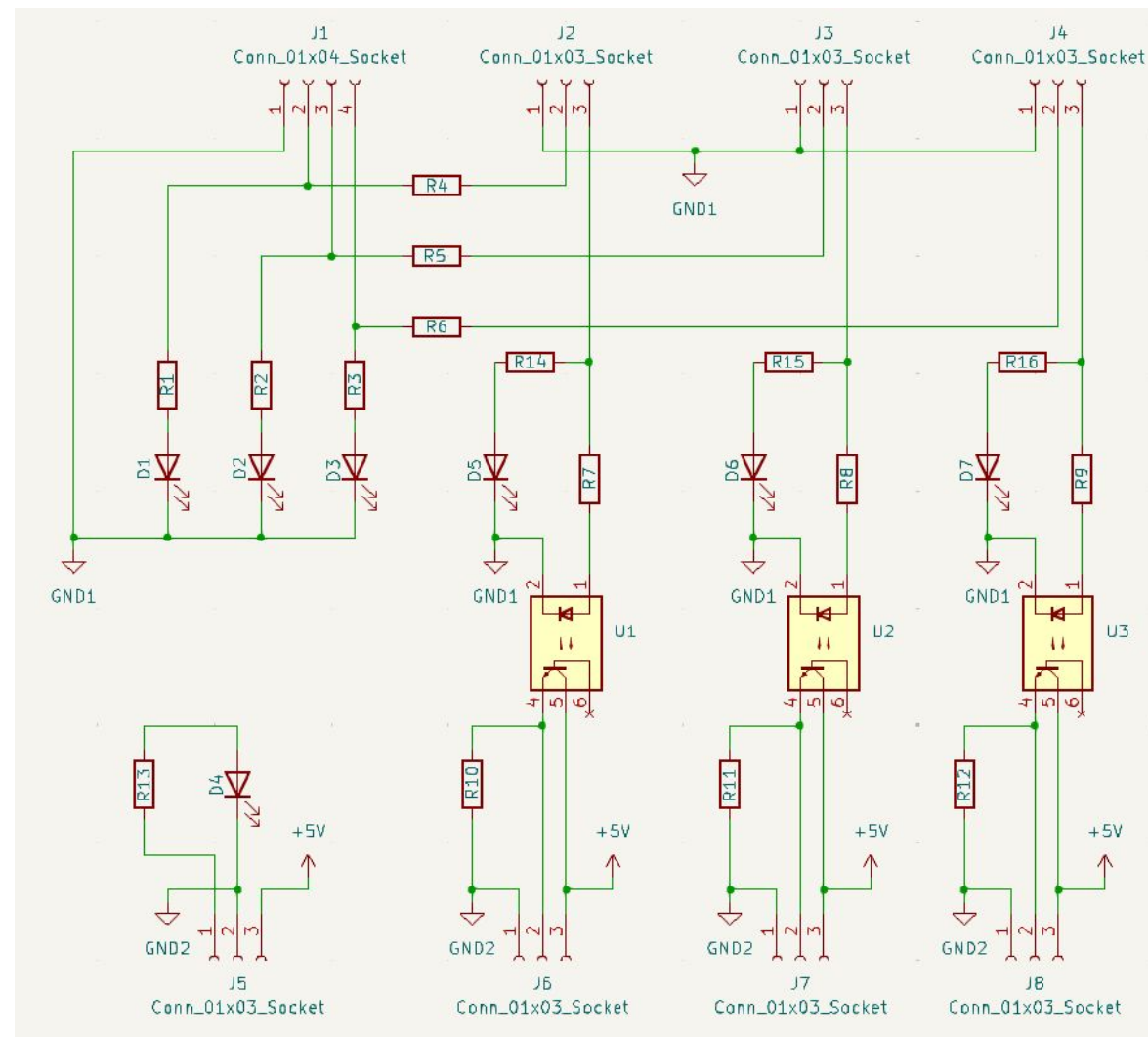
- Sistema di votazione a maggioranza
  - 3 ingressi
  - 1 uscita
- L'uscita è **alta** quando almeno due ingressi sono attivi, è **bassa** altrimenti
- Sistemi generalmente con alti requisiti di safety → Perfetti per essere implementati su una ClearSy

## OK ma... chi genera le PWM?

- I generatori di frequenza Direct Digital Synthesizer (DDS) possono generare le PWM, ma sono grossi, costosi, e generano un segnale (o al massimo due)
- Possiamo implementare una versione semplificata del DDS in software? Sì ma...
  - Il codice macchina deve avere certe caratteristiche (numero di colpi di clock necessari a fare un ciclo noto e costante, tempo su un if uguale al tempo su un else etc...) → L'unico linguaggio che rispetta queste caratteristiche è l'ASSEMBLY
  - Su una singola CPU possiamo gestire, limitando al minimo il jitter, una sola PWM
- Possiamo sfruttare un System on Chip (SoC) che integra delle periferiche PWM
  - Arduino Due (SoC ARM Cortex-M3 AT91SAM3X8E) contiene 8 canali PWM programmabili
  - Abbiamo una versione di Arduino Due per generare (fino a) 8 PWM suddivise (fino a) 2 tipologie di PWM diverse. Una tipologia è definita dalla coppia (f,d)

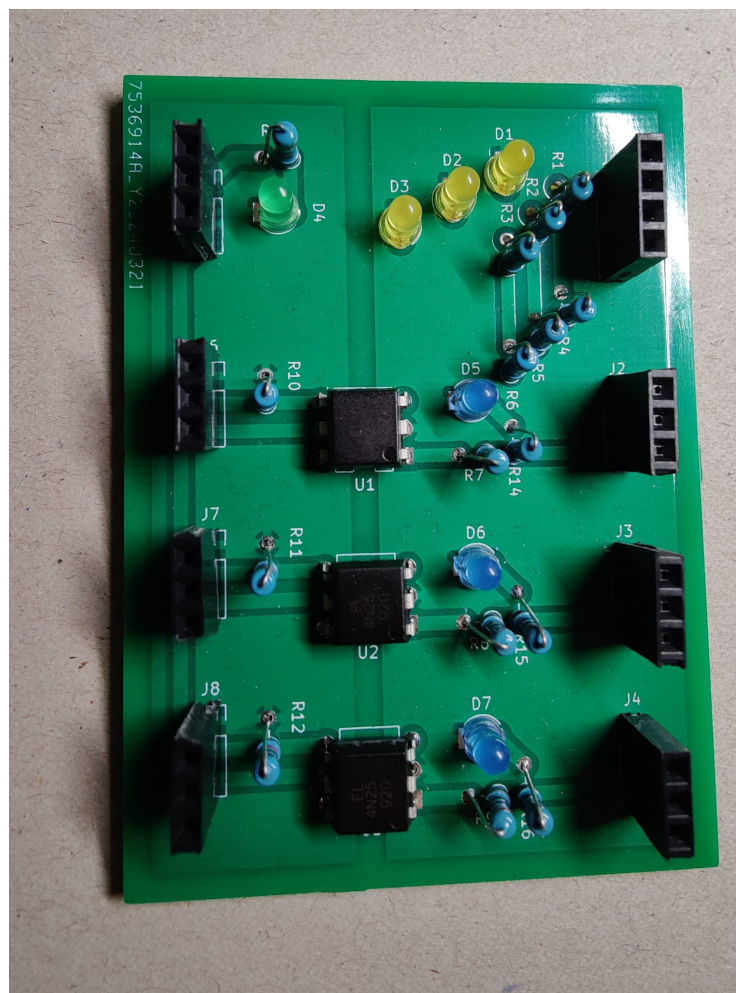
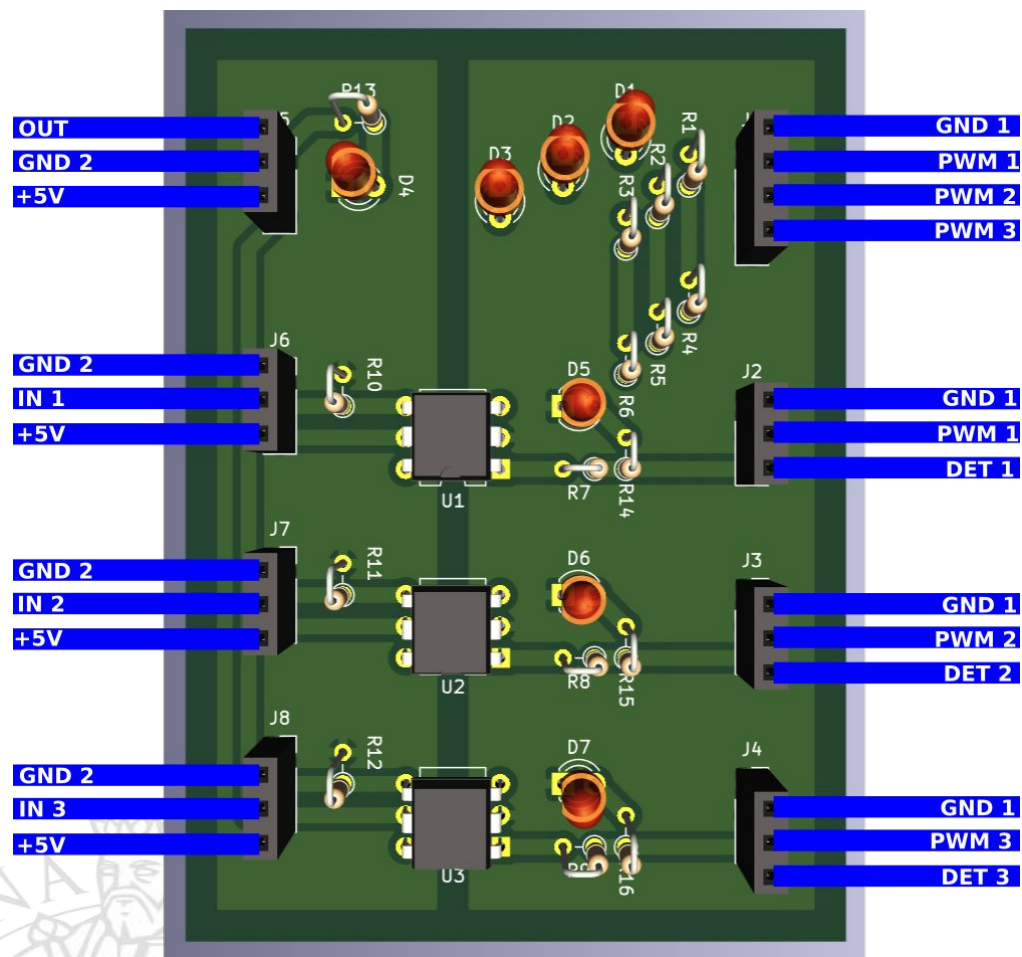
# Schematico di interconnessione

- Schematico di connessione fra tutti i dispositivi
  - Conn J1 → Arduino Due - PWM Generator
  - Conn J2, J3, J4 → Arduino Due - PWM Detector
  - Conn J5, J6, J7, J8 → ClearSy
- U1, U2, U3 sono degli optoisolatori (o fotoaccoppiatori)... Perché?
  - Il "mondo" della ClearSy è a safety più alta rispetto al "mondo" dei 4 Arduino Due, quindi vogliamo isolarla dal punto di vista elettrico (un guasto di un Arduino Due non deve avere impatto sulla ClearSy)
  - Il "mondo" dei 4 Arduino Due lavora con tensioni di 3V3, il "mondo" della ClearSy lavora con 5V, quindi è necessario disaccoppiare queste alimentazioni



# Scheda di interconnessione

- Lo schematico è stato portato su PCB e stampato per ottenere una scheda di interconnessione



- LED giallo: acceso durante PW della corrispondente PWM
- LED blu: acceso Arduino Due ha rilevato correttamente la PWM
- LED verde: uscita di sistema, acceso quando 2003 Arduino Due hanno rilevato la PWM