



**Makerspace mobili per promuovere l'educazione dei maker nelle scuole
democratizzare l'istruzione STEAM e lo sviluppo dell'innovazione per tutti gli
studenti**

IO2

Linee guida per gli insegnanti

Sistema di controllo semaforico con RPi Pico e Phyton

Livello di difficoltà: medio-alto

Età di riferimento: dai 16 ai 18 anni

Costo: attualmente molto basso

Realizzato da: IIS Leonardo da Vinci

Autori: Renato Gatti - Loris Ranalli - Giovanni De Rita



Erasmus+

GRANT NUMBER: 2020-1-PL01-KA201-081698

Dichiarazione

Questo file è stato preparato nell'ambito del progetto STEAM2GO. Laddove sono stati utilizzati altri materiali originali pubblicati e inediti, questi sono stati riconosciuti.

Diritto d'autore

© Copyright 2020 - 2022 il Consorzio [STEAM2GO](#)



Tutti i diritti riservati.



Questo documento è concesso in licenza al pubblico ai sensi di Creative Commons Attribution-Non Commercial - No Derivatives 4.0 International License.

Esclusione di responsabilità per il finanziamento

Questo progetto è stato finanziato con il sostegno della Commissione Europea. Questo rapporto riflette solo il punto di vista dell'autore e la Commissione non può essere ritenuta responsabile per qualsiasi uso che possa essere fatto delle informazioni in esso contenute.

Linee guida per gli insegnanti

Sistema di controllo semaforico con Raspberry Pico e Phytton

Contenuti

Linee guida per gli insegnanti	1
Note introduttive	4
Studio e realizzazione dei diagrammi temporali	5
Presentazione della scheda Raspberry Pi Pico	7
La breadboard	8
Creiamo il nostro programma "Semaforo" con Raspberry Pi Pico	9
Creiamo i nostri "Semafori" con Raspberry Pi Pico	10
I primi due programmi	12
Il terzo e il quarto programma	14
Disegno e realizzazione della planimetria di un incrocio a due strade	16
Note sui collegamenti di pulsanti e interruttori	20
Note sulle resistenze di pull-up e pull-down	21
File di supporto	22
Guida RPi Pico	22
Guida IDE Thonny	22

Note introduttive

Informazioni sul progetto

Inizieremo con la programmazione di un prototipo realizzato su singola bread board. Successivamente si potrà realizzare un modello dell'incrocio su carta o cartone che verrà appoggiato su un foglio di polistirolo dove saranno praticati dei fori rettangolari, di adeguate dimensioni, per alloggiare le quattro breadboard da 170 contatti. Queste verranno poi collegate alla breadboard con RPi Pico, pulsante e interruttore. I vari dispositivi andranno poi collegati ai relativi GPIO della scheda Raspberry Pi Pico e, infine, verrà programmato l'intero sistema.

Obiettivi formativi

Gli studenti coinvolti nel progetto avranno la possibilità di:

- Comprendere e implementare diagrammi di temporizzazione
- Comprendere dettagliatamente un circuito elettronico
- Creare circuiti utilizzando Raspberry Pi Pico e componenti elettronici
- Creare una planimetria di un impianto realizzando un modello in carta utilizzando materiali di uso comune
- Imparare i fondamenti della programmazione per implementare un modello proposto
- Analizzare e identificare i parametri e le funzioni da controllare in un sistema complesso
- Interfacciare, attraverso i GPIO di Raspberry Pi Pico, i segnali di I/O provenienti dai vari dispositivi
- Sviluppare, tramite il software Thonny per Raspberry Pi Pico, un complesso sistema di controllo
- Testare il prototipo
- Ottimizzare e definire la soluzione di lavoro ottimale condividendola tra i partecipanti

Collegamento ad altre aree tematiche e/o concetti STEM specifici

- Matematica
- Tecnologia
- Elettronica
- Informatica
- Realizzazione di manufatti relativi ad un progetto tecnico-scientifico

Hardware e software di cui potresti aver bisogno

Hardware	Software	Materiale artigianale
- 3-5 breadboard da 170 punti - https://amzn.eu/d/bcFYyKs - 1 x Raspberry Pi Pico - 1 pulsante - 1 interruttore SPDT - Jumper wires kit - Wiring cables kit (M/M and M/F) - Kit singolo 2 (o 4) semafori a led preassemblati: https://www.amazon.it/ZHITING-Display-semaforo-Mini-semaforo-elettronici/dp/B0859ZPM47	- Excel - Word - Thonny-4.0.1 IDE	- Alcuni fogli di carta bianca o cartoncino formato A4 o A3 - Un foglio di polistirolo espanso - Un foglio di polipropilene alveolare - 1 taglierino - 1 forbice - 1 colla - 1 nastro biadesivo - 1 matita - 1 temperamatite - Alcuni pennarelli indelebili di vari colori - Attrezzatura di cancelleria come matite, penne, fogli di carta

Le fasi di lavoro

1. Il docente può scegliere, in base al livello degli studenti, una delle opzioni presentate sia per la parte di programmazione che per la realizzazione del piano dell'intersezione stradale.
2. La fase iniziale sarà l'identificazione del corretto funzionamento di un controllo semaforico. Particolare attenzione dovrà essere prestata alla tempistica dei segnali e alle misure di sicurezza da adottare. Questo può essere fatto attraverso una rapida ricerca su Internet guidata dal docente.
3. Successivamente si dovrà studiare il diagramma temporale dell'impianto, inizialmente nel modo più semplice possibile e successivamente tenendo conto di un tempo di sovrapposizione della luce rossa, per garantire che l'incrocio sia libero da eventuali veicoli.
4. Configurazione della scheda Raspberry Pi Pico
5. Creazione del primo programma operativo utilizzando l'IDE "Thonny" per Raspberry Pi Pico
6. Effettuare tutte le interconnessioni necessarie tra i GPIO della scheda Raspberry Pi Pico e i vari componenti esterni
7. Trasferimento del programma dal PC alla scheda Raspberry Pi Pico
8. Progettare e realizzare una planimetria dell'intersezione stradale dove verranno posizionate le breadboard e i vari componenti
9. Testare il corretto funzionamento del sistema

Studio e realizzazione dei diagrammi temporali

Il primo passo per realizzare il progetto è analizzare e creare un diagramma con le temporizzazioni dei vari segnali di ogni semaforo. Per una corretta comprensione di ogni passaggio, è necessario partire dal diagramma temporale più semplice. Una volta completato, i pro e i contro di ognuno verranno portati all'attenzione degli studenti.

I diagrammi temporali possono essere realizzati su carta a quadretti oppure utilizzando un foglio di calcolo o un qualsiasi programma di grafica.

a) Diagramma di temporizzazione semplice, senza sovrapposizione delle luci rosse

La sequenza di ogni semaforo sarà, nell'esempio riportato in Fig.1:

- Verde: 4 sec.
- Giallo: 1 sec.
- Rosso: 5 sec. (il tempo del segnale rosso deve essere uguale alla somma del verde e del giallo del semaforo opposto)

In questo caso si assume che si accendano contemporaneamente il semaforo verde del semaforo [1], che chiameremo G1, e il semaforo rosso del semaforo opposto [2], che chiameremo R2.

Trascorso invece il tempo programmato, si accendono contemporaneamente la luce verde del semaforo [2], che chiameremo G2, e la luce rossa del semaforo opposto [1], che chiameremo R1. Fare riferimento alla **Figura 1**

b) Diagramma di temporizzazione con sovrapposizione delle luci rosse

In questo caso abbiamo considerato la necessità di concedere un po' di tempo alla sovrapposizione delle due luci rosse prima di attivare il verde. Questo per essere sicuri che, una volta acceso il verde, non ci siano più veicoli nell'incrocio.

Questo è il principale criterio di sicurezza, richiesto dalle norme stradali e dal codice della strada.

Ora la sequenza di ogni semaforo sarà, nell'esempio riportato in **Figura 2**:

- Verde: 4 sec.
- Giallo: 1 sec.
- Rosso: 7 sec. (il tempo del segnale rosso deve essere uguale alla somma del verde e del giallo del semaforo opposto più il tempo di sovrapposizione che abbiamo assunto in 1 sec per lato).

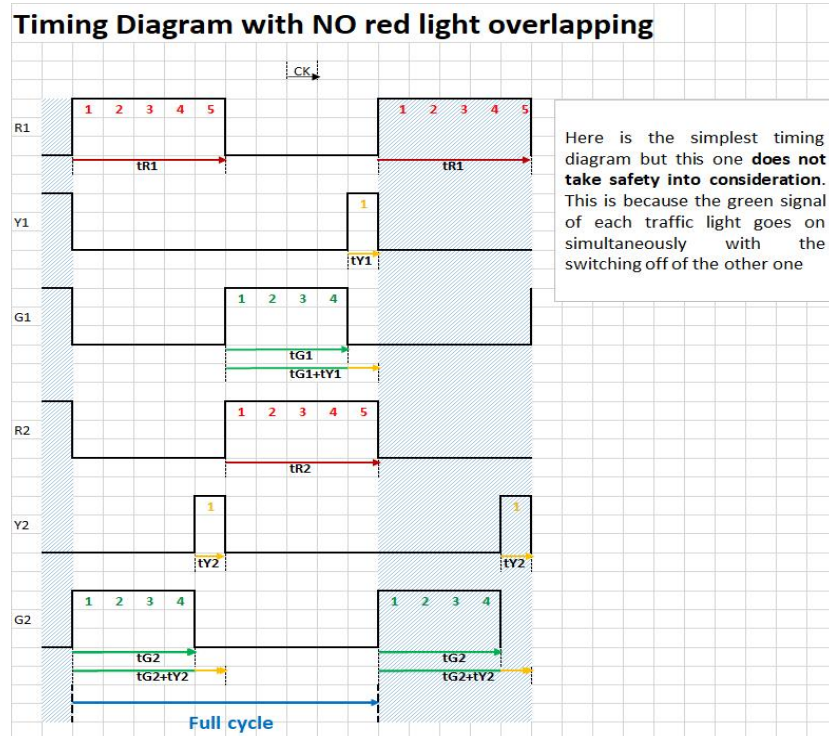


Figure 1

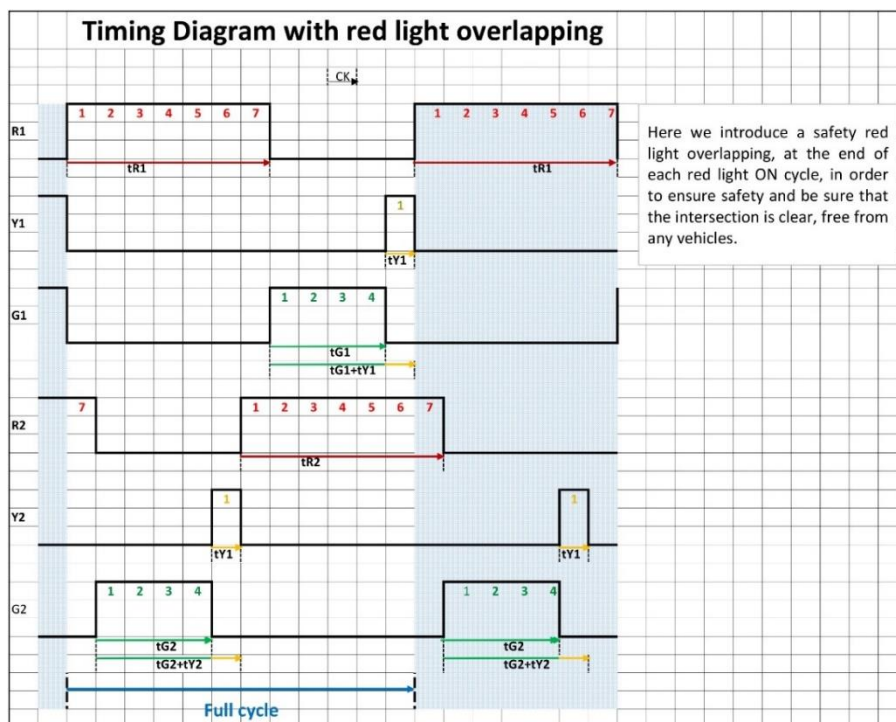


Figure 2

L'insegnante può intervenire sollevando questioni di sicurezza, ad esempio:

- È necessaria una sovrapposizione di luci rosse? Se è così, perché?
- Per quanto tempo le luci rosse dovrebbero sovrapporsi se necessario?
- In base a quali parametri calcolereste il tempo di sovrapposizione dei semafori rossi?
- Nel caso di un semaforo pedonale, come regolereste i vari orari del semaforo?
- Etc.

Cliccando sui link sottostanti è possibile scaricare i Diagrammi di Temporizzazione in vari formati:

Link ai Diagrammi temporali in formato MS Excel:

- [Diagramma di temporizzazione più semplice senza sovrapposizione di luci rosse](#)
- [Diagramma di temporizzazione con sovrapposizione di luci rosse](#)

Link ai diagrammi temporali in formato PDF:

- [Diagramma di temporizzazione più semplice senza sovrapposizione di luci rosse](#)
- [Diagramma di temporizzazione con sovrapposizione di luci rosse](#)

Link ai diagrammi temporali in formato JPG:

- [Diagramma di temporizzazione più semplice senza sovrapposizione di luci rosse](#)
- [Diagramma di temporizzazione con sovrapposizione di luci rosse](#)

Presentazione della scheda Raspberry Pi Pico

Cos'è Raspberry Pico?

Raspberry Pi Pico è una scheda microcontrollore a basso costo e ad alte prestazioni con interfacce digitali flessibili. Le caratteristiche principali includono:

- Chip microcontrollore RP2040 progettato da Raspberry Pi nel Regno Unito
- Processore Arm Cortex M0+ dual-core, clock flessibile fino a 133 MHz
- 264kB di SRAM e 2MB di memoria flash integrata
- USB 1.1 con supporto per dispositivi e host
- Programmazione drag-and-drop utilizzando l'archiviazione di massa tramite USB
- 26 pin GPIO multifunzione
- 2 × SPI, 2 × I2C, 2 × UART, 3 × ADC a 12 bit, 16 × canali PWM controllabili
- Orologio e timer accurati su chip
- Termometro
- 8 macchine con I/O programmabili (PIO) per il supporto di periferiche personalizzate

Il Raspberry Pi Pico si presenta come un piccolo modulo che consente la saldatura diretta strips di connessione, mentre il **Pico H** viene fornito con intestazioni pre-saldate (consigliato).

Il layout di Raspberry Pi Pico è mostrato nella **Figura 3** (anteriore) e **Figura 4** (posteriore)



Figura 3



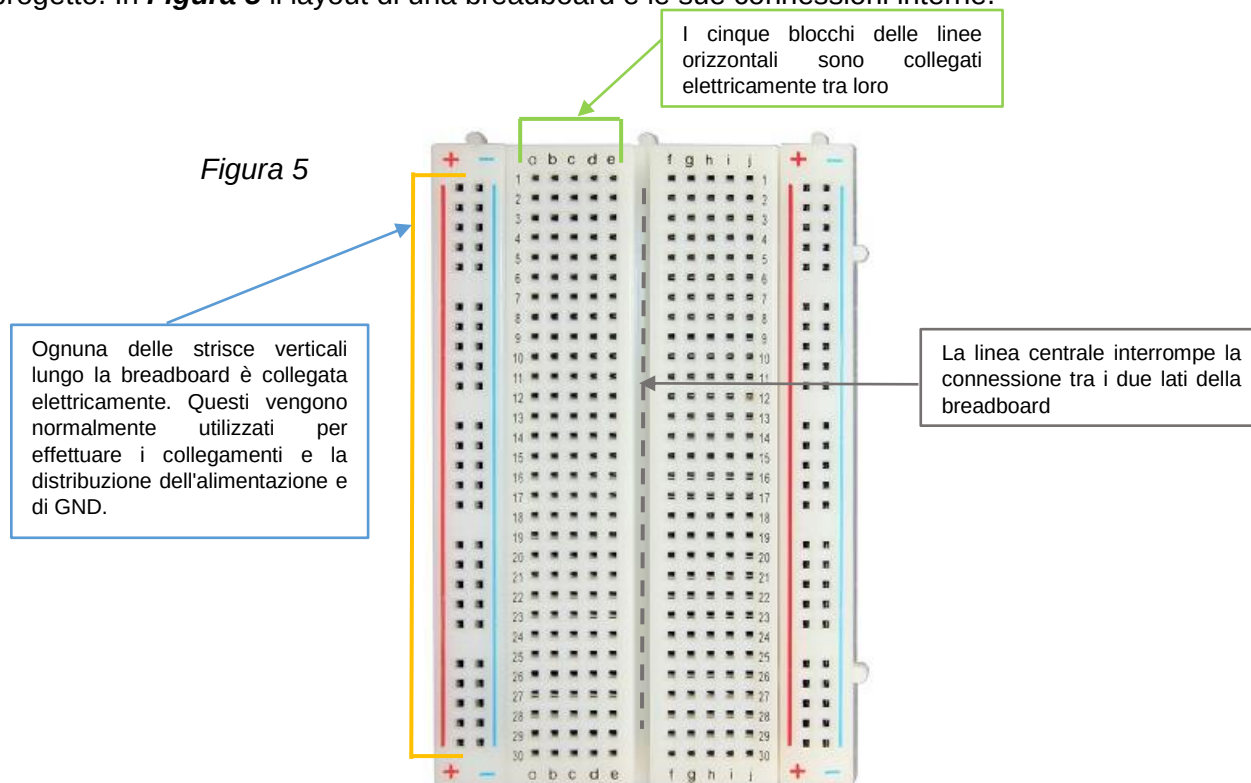
Figura 4

Per il corretto montaggio del pulsante e dell'interruttore è necessario verificare il funzionamento dei pin utilizzando il tester fornito con il Mobile Maker Space (si vedano le "Note sui collegamenti di pulsanti e interruttori" a pag. 20)

La breadboard

Breadboard è un modo per realizzare dispositivi elettronici senza dover utilizzare un saldatore. I componenti vengono inseriti nelle prese (fori) presenti sulla breadboard e quindi vengono collegati utilizzando fili "jumper" esterni per effettuare le connessioni tra i vari componenti.

Le breadboard sono fatte per effettuare prototipi o esperimenti veloci. Non sono adatte per mantenere in funzione i circuiti per molto tempo. Le connessioni, non essendo permanenti, sono facilmente removibili nel caso si commetta un errore o semplicemente iniziare da capo un nuovo progetto. In **Figura 5** il layout di una breadboard e le sue connessioni interne.



In **Figura 6** un esempio di come collegare un LED con una batteria e un pulsante

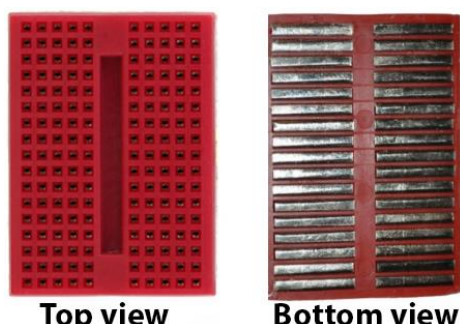
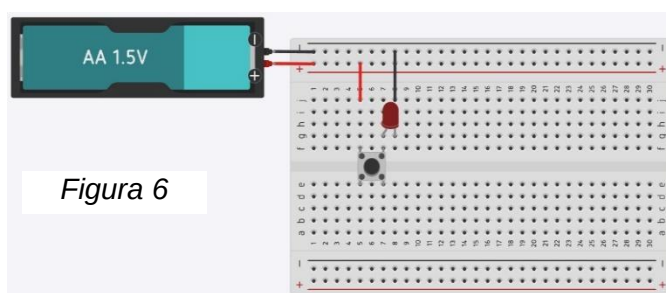


Figura 7

In **Figura 7** un altro tipo di breadboard, più semplice e più piccolo, che non ha le strisce verticali per i collegamenti e la distribuzione dell'alimentazione e GND

Programmare la scheda Raspberry Pi Pico

Per prima cosa è necessario scaricare e installare il software "Thonny" IDE (Integrated Development Environment) per Raspberry Pi Pico. Un IDE è un software che ti consente di scrivere direttamente il codice, compilarlo e scaricarlo sulla scheda Raspberry Pi Pico per farlo funzionare. Spesso fornisce anche altre funzionalità (come il debugger, la guida, l'evidenziazione delle parole chiave, ecc.).

Puoi scaricare l'IDE "Thonny" e visualizzare tutte le funzionalità qui:

<https://thonny.org/>

Creiamo il nostro programma "Semaforo" con Raspberry Pi Pico

Ora andiamo a predisporre le connessioni per poter utilizzare i GPIO della scheda Raspberry Pi Pico per collegare i dispositivi esterni, nel nostro caso due mini semafori preassemblati, un pulsante e un interruttore.

Collegamento dei GPIO Raspberry Pico a dispositivi esterni

Il passaggio successivo consiste nel collegare i GPIO Raspberry Pico alle breadboard. Per fare ciò è necessario utilizzare i cavi jumper, inclusi nel Mobile Maker Space. I GPIO Raspberry Pi Pico sono mostrati nella seguente **Figura 8**

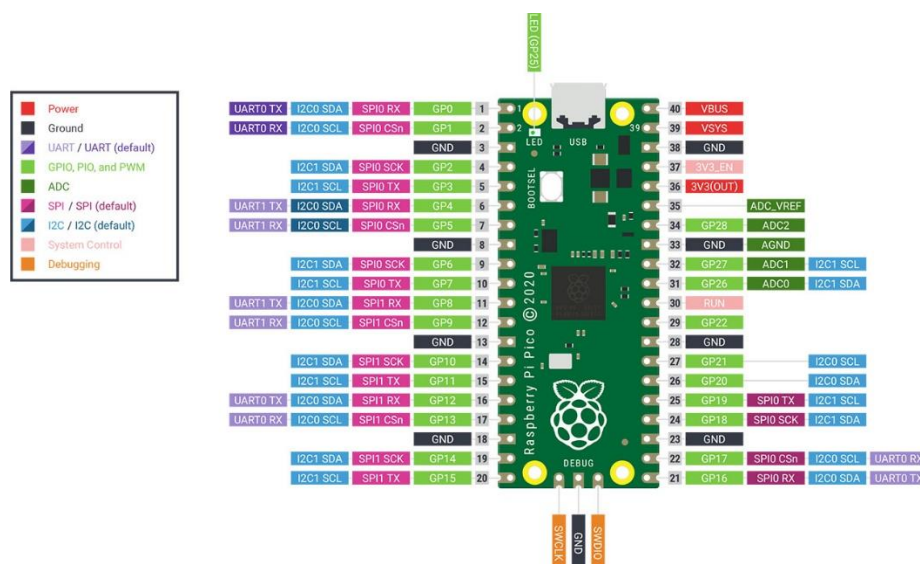


Figura 8 - Raspberry Pi Pico e relativi GPIO

È conveniente realizzare un foglio di calcolo con le interconnessioni tra i GPIO Raspberry Pi Pico e i vari semafori presenti nelle rispettive breadboard. Gli esempi sono in **Figura 9** (due semafori a LED) e in **Figura 10** (dispositivi di input).

Raspberry Pi pico		Led Traffic Lights		
GP #	PIN	Traffic Light n.	Colour	Pin
GP 12	16	1	Green 1	G
GP 11	15	1	Yellow 1	Y
GP 10	14	1	Red 1	R
Ground	13	1	---	GND
GP 4	6	2	Green 2	G
GP 3	5	2	Yellow 2	Y
GP 2	4	2	Red 2	R
Ground	3	2	---	GND

Figura 9 PinOut dei due semafori

Raspberry Pi pico		Input device
GP #	PIN	Type
GP 15	20	Start Push-button
GP 14	19	Maintenance switch
3V3	36	VCC

Figura 10 Dispositivi di input PinOut

Cliccando sui link sottostanti è possibile scaricare le tabelle dei PinOut in vari formati:

Link alla tabella PinOut in formato MS Excel:

- [Semaforo a due LED PinOut.xlsx](#)
- [Dispositivi di input PinOut.xlsx](#)

Link alla tabella PinOut in formato JPG:

- [Due semafori a LED PinOut.jpg](#)
- [Dispositivi di input PinOut.jpg](#)

Creiamo i nostri "Semafori" con Raspberry Pi Pico

Il sistema può essere implementato in due modi diversi:

- il più veloce è costruire un prototipo con un'unica breadboard su cui assemblare tutti i componenti. Questo sistema non prevede la realizzazione dei disegni di tracciato stradale e la realizzazione della piattaforma su cui posizionare i vari componenti (**Figura 11**)
- realizzando del disegno di tracciato stradale e della piattaforma su cui posizionare i vari componenti. Questo sistema richiede più tempo perché comporta anche una parte di realizzazione grafica e la costruzione del pannello su cui inserire i vari elementi (**Figura 12 e Figura 13**)

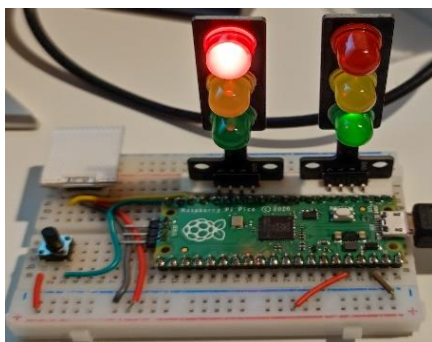


Figura 11

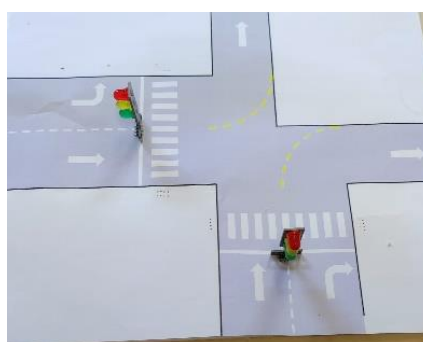


Figura 12

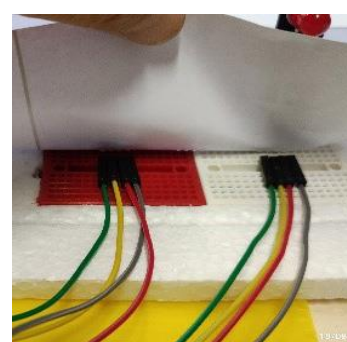


Figura 13

A questo punto è necessario effettuare tutti i collegamenti tra la scheda Raspberry Pico e la/e breadboard/i seguendo il PinOut indicato nelle **Figure 9 e 10**.

Una volta completate le connessioni possiamo iniziare a programmare il nostro Raspberry Pi Pico utilizzando il software IDE "Thonny". Si possono creare quattro diversi programmi, in ordine di difficoltà crescente:

I primi due programmi:

1. TL NO overlap NO buttons
2. TL overlap NO buttons

sono realizzati, per semplicità, senza utilizzare il pulsante e l'interruttore esterni. La sequenza semaforica inizierà automaticamente, senza l'intervento di dispositivi esterni (pulsante "Start" e switch "Maintenance").

Il terzo e il quarto sono più complessi in quanto utilizzeranno anche il pulsante "Start" e l'interruttore "Maintenance" per attivare le varie fasi del programma:

3. TL NO overlap Buttons
4. TL overlap Buttons

Prestare attenzione alle seguenti indicazioni:

- Una volta che la scheda Raspberry Pi Pico è stata collegata al PC tramite USB, sarà necessario scegliere la scheda corretta che si desidera programmare nell'ambiente di sviluppo "Thonny", nella scheda "Strumenti" - "Opzioni" - "Interprete" il nostro caso "MicroPython (Raspberry Pi Pico) (**Figura 14**)
- Python è un ambiente di programmazione particolare. Non usa le parentesi graffe per raggruppare le istruzioni ma usa solo la "indentazione", ovvero spostando le istruzioni subordinate a destra. Può succedere che il compilatore evidenzi errori dovuti a qualche errore di indentazione ma possono essere risolti attivando le funzioni "Assistente" (**Figura 15**) e "Shell" (**Figura 16**) nella scheda "Visualizza", che forniranno le informazioni per risolvere eventuali problemi
- All'inizio di ogni sequenza è importante impostare tutte le uscite GPIO a "0", in modo che tutti i LED siano spenti all'avvio del programma
- Ogni volta che si imposta un GPIO a "1" e si programma il tempo in cui quel led rimarrà acceso, dovremo anche spegnerlo riportandolo a "0" prima di accendere il successivo e viceversa.
- Nei commenti, preceduti dal cancelletto (#) sono presenti tutte le spiegazioni di ogni singolo comando.

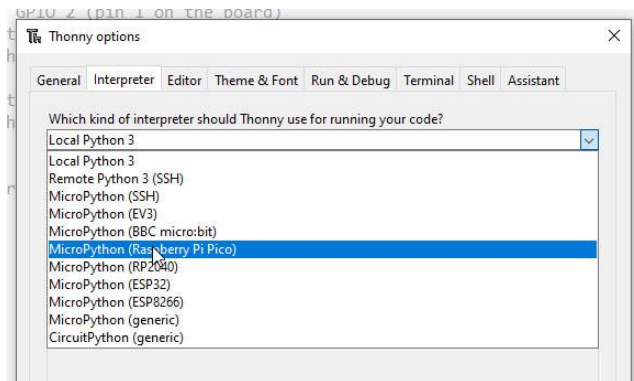


Figura 14

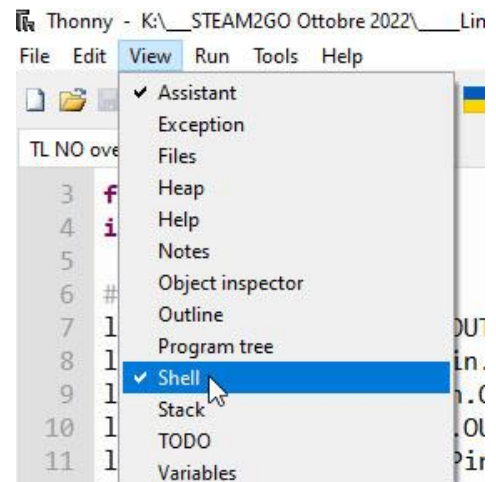


Figura 15

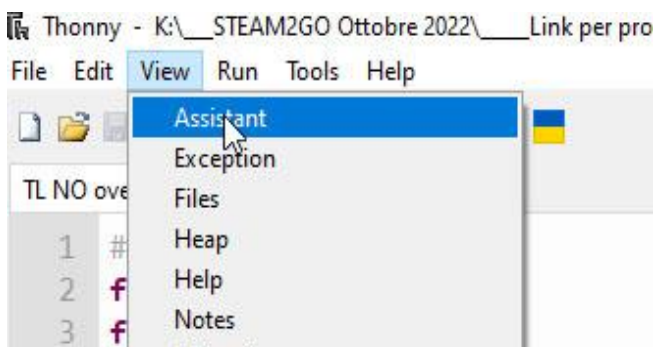


Figura 16

[Clicca qui per vedere il video del prototipo su singola breadboard realizzato con il programma con "4. TL overlap Buttons"](#)

Le pagine seguenti mostrano i quattro programmi realizzati con Raspberry Pi Pico e "Thonny":

1. TL NO overlap NO buttons
2. TL overlap NO buttons
3. TL NO overlap Buttons
4. TL overlap Buttons

I primi due programmi sono i più semplici, in quanto **non utilizzano i dispositivi di input, il pulsante e l'interruttore**:

1. TL NO overlap NO buttons

La **Figura 17** mostra il programma “TL NO overlap NO buttons” utilizzando solo i GPIO indicati in **Figura 9**.

```
TL NO overlap_NO Buttons.py ✕
1  #Importing libraries
2  from machine import Pin
3  from utime import sleep
4
5  #Definitions
6  led_red_1 = Pin(2, Pin.OUT)    #Assigns the variable "led_red_1" the GPIO 2 (pin 1 on the board)
7  led_yellow_1 = Pin(3, Pin.OUT) #Assigns the variable "led_yellow_1" the GPIO 3 (pin 2 on the board)
8  led_green_1 = Pin(4, Pin.OUT)  #Assigns the variable "led_green_1" the GPIO 4 (pin 4 on the board)
9  led_red_2 = Pin(10, Pin.OUT)   #Assigns the variable "led_red_2" the GPIO 10 (pin 5 on the board)
10 led_yellow_2 = Pin(11, Pin.OUT) #Assigns the variable "led_yellow_2" the GPIO 11 (pin 6 on the board)
11 led_green_2 = Pin(12, Pin.OUT) #Assigns the variable "led_green_2" the GPIO 12 (pin 7 on the board)
12
13 #Light Initialization
14 #All the LEDs are turned off during the initialization phase to be sure the sequence starts with all lights off
15 led_red_1.value(0)            #Turn OFF light RED_1
16 led_red_2.value(0)            #Turn OFF light RED_2
17 led_yellow_1.value(0)         #Turn OFF light YELLOW_1
18 led_yellow_2.value(0)        #Turn OFF light YELLOW_2
19 led_green_1.value(0)          #Turn OFF light GREEN_1
20 led_green_2.value(0)          #Turn OFF light GREEN_2
21 sleep(1)                      #Wait a second
22
23 for i in range (1,3):
24     led_yellow_1.value(1)      #Turn ON light YELLOW_1
25     led_yellow_2.value(1)      #Turn ON light YELLOW_2
26     sleep (0.5)               #For 0.5 sec
27     led_yellow_1.value(0)      #Turn OFF light YELLOW_1
28     led_yellow_2.value(0)      #Turn OFF light YELLOW_2
29     sleep (0.5)               #For 0.5 sec
30
31 while True: # Infinite Loop
32     #Phase 1
33     led_red_1.value(1)         #Turn ON light RED_1
34     led_green_2.value(1)       #Turn ON light GREEN_2
35     sleep(5)                   #For 5 sec
36     led_green_2.value(0)       #Turn OFF light GREEN_2
37     led_red_1.value(1)         #Turn ON light RED_1 while YELLOW_2 is ON
38     led_yellow_2.value(1)      #Turn ON light YELLOW_2
39     sleep(2)                   #For 2 sec
40     led_yellow_2.value(0)      #Turn OFF light YELLOW_2
41     led_red_1.value(0)         #Turn ON light RED_1
42     sleep(0)                   #For 2 sec overlap
43     #Phase 2
44     led_red_2.value(1)         #Turn ON light RED_2
45     led_green_1.value(1)       #Turn ON light GREEN_1
46     sleep(5)                   #For 5 secs.
47     led_green_1.value(0)       #Turn OFF light GREEN_1
48     led_red_2.value(1)         #Turn ON light RED_2 while YELLOW_1 is ON
49     led_yellow_1.value(1)      #Turn ON light YELLOW_1
50     sleep(2)                   #For 2 sec
51     led_yellow_1.value(0)      #Turn OFF light YELLOW_2
52     led_red_2.value(0)         #Turn ON light RED_1
53     sleep(0)                   #Immediately
```

Figura 17 TL NO overlap NO buttons

Qui si può scaricare il primo programma Raspberry Pico Code:

1. [TL NO overlap NO buttons](#)

2. TL overlap NO buttons

La **Figura 18** mostra il programma “TL overlap NO buttons” utilizzando solo i GPIO indicati in **Figura 9**.

```
TL overlap_NO Buttons.py ✕
1 #Importing libraries
2 from machine import Pin
3 from utime import sleep
4
5 #Definitions
6 led_red_1 = Pin(2, Pin.OUT) #Assigns the variable "led_red_1" the GPIO 2 (pin 1 on the board)
7 led_yellow_1 = Pin(3, Pin.OUT) #Assigns the variable "led_yellow_1" the GPIO 3 (pin 2 on the board)
8 led_green_1 = Pin(4, Pin.OUT) #Assigns the variable "led_green_1" the GPIO 4 (pin 4 on the board)
9 led_red_2 = Pin(10, Pin.OUT) #Assigns the variable "led_red_2" the GPIO 10 (pin 5 on the board)
10 led_yellow_2 = Pin(11, Pin.OUT) #Assigns the variable "led_yellow_2" the GPIO 11 (pin 6 on the board)
11 led_green_2 = Pin(12, Pin.OUT) #Assigns the variable "led_green_2" the GPIO 12 (pin 7 on the board)
12
13 #Light Initialization
14 #All the LEDs are turned off during the initialization phase to be sure the sequence starts with all lights off
15 led_red_1.value(0) #Turn OFF light RED_1
16 led_red_2.value(0) #Turn OFF light RED_2
17 led_yellow_1.value(0) #Turn OFF light YELLOW_1
18 led_yellow_2.value(0) #Turn OFF light YELLOW_2
19 led_green_1.value(0) #Turn OFF light GREEN_1
20 led_green_2.value(0) #Turn OFF light GREEN_2
21 sleep(1) #Wait a second
22
23 for i in range (1,3):
24     led_yellow_1.value(1) #Turn ON light YELLOW_1
25     led_yellow_2.value(1) #Turn ON light YELLOW_2
26     sleep (0.5) #For 0.5 sec
27     led_yellow_1.value(0) #Turn OFF light YELLOW_1
28     led_yellow_2.value(0) #Turn OFF light YELLOW_2
29     sleep (0.5) #For 0.5 sec
30
31 while True: # Infinite Loop
32     #Phase 1
33     led_red_1.value(1) #Turn ON light RED_1
34     led_green_2.value(1) #Turn ON light GREEN_2
35     sleep(5) #For 5 sec
36     led_green_2.value(0) #Turn OFF light GREEN_2
37     led_red_1.value(1) #Turn ON light RED_1 while YELLOW_2 is ON
38     led_yellow_2.value(1) #Turn ON light YELLOW_2
39     sleep(2) #For 2 sec
40     led_yellow_2.value(0) #Turn OFF light YELLOW_2
41     led_red_2.value(1) #Turn ON light RED_1
42     sleep(2) #For 2 sec overlap
43     led_red_1.value(0) #Turn OFF light RED_1
44     led_red_2.value(0) #Turn OFF light RED_2
45     sleep(0) #Immediately
46     #Phase 2
47     led_red_2.value(1) #Turn ON light RED_2
48     led_green_1.value(1) #Turn ON light GREEN_1
49     sleep(5) #For 7 secs.
50     led_green_1.value(0) #Turn OFF light GREEN_1
51     led_red_2.value(1) #Turn ON light RED_2 while YELLOW_1 is ON
52     led_yellow_1.value(1) #Turn ON light YELLOW_1
53     sleep(2) #For 2 sec
54     led_yellow_1.value(0) #Turn OFF light YELLOW_2
55     led_red_1.value(1) #Turn ON light RED_1
56     sleep(2) #For 2 sec overlap
57     led_red_1.value(0) #Turn OFF light RED_1
58     led_red_2.value(0) #Turn OFF light RED_2
59     sleep(0) #Immediately
```

Figura 18 TL overlap NO buttons

Qui puoi scaricare i secondi programmi Raspberry Pico Code:

2. [TL overlap NO buttons](#)

Il terzo e il quarto programma sono un po' più complessi, in quanto usano i dispositivi di input, pulsante e interruttore:

In **Figura 19** è mostrato il programma “TL NO overlap buttons” utilizzando tutti i GPIO indicati nelle **Figure 9 e 10**.

3. TL NO overlap buttons

```

TL NO overlap Buttons.py
1 #Importing libraries
2 from machine import Pin
3 from utime import sleep
4
5 #Definitions
6 led_red_1 = Pin(2, Pin.OUT) #Assigns the variable "led_red_1" the GPIO 2 (pin 1 on the board)
7 led_yellow_1 = Pin(3, Pin.OUT) #Assigns the variable "led_yellow_1" the GPIO 3 (pin 2 on the board)
8 led_green_1 = Pin(4, Pin.OUT) #Assigns the variable "led_green_1" the GPIO 4 (pin 4 on the board)
9 led_red_2 = Pin(10, Pin.OUT) #Assigns the variable "led_red_2" the GPIO 10 (pin 5 on the board)
10 led_yellow_2 = Pin(11, Pin.OUT) #Assigns the variable "led_yellow_2" the GPIO 11 (pin 6 on the board)
11 led_green_2 = Pin(12, Pin.OUT) #Assigns the variable "led_green_2" the GPIO 12 (pin 7 on the board)
12 start = Pin(15, Pin.IN, machine.Pin.PULL_DOWN) #Assigns the variable "start" the GPIO 15 (pin 20 on the board)
13 service = Pin(14, Pin.IN, machine.Pin.PULL_DOWN) #Assigns the variable "service" the GPIO 14 (pin 19 on the board)
14 START_STATE = 0 #Set START_STATE variable to "0"
15 SERVICE_STATE = 0 #Set SERVICE_STATE variable to "0"
16
17 #Light Initialization
18 #All the LEDs are turned off during the initialization phase to be sure the sequence starts with all lights off.
19 led_red_1.value(0) #Turn OFF light RED_1
20 led_red_2.value(0) #Turn OFF light RED_2
21 led_yellow_1.value(0) #Turn OFF light YELLOW_1
22 led_yellow_2.value(0) #Turn OFF light YELLOW_2
23 led_green_1.value(0) #Turn OFF light GREEN_1
24 led_green_2.value(0) #Turn OFF light GREEN_2
25 sleep(1) #Wait a second
26
27 def on():
28     #Phase 1
29     led_red_1.value(1) #Turn ON light RED_1
30     led_green_2.value(1) #Turn ON light GREEN_2
31     sleep(5) #For 5 sec
32     led_green_2.value(0) #Turn OFF light GREEN_2
33     led_red_1.value(1) #Turn ON light RED_1 while YELLOW_2 is ON
34     led_yellow_2.value(1) #Turn ON light YELLOW_2
35     sleep(2) #For 2 sec
36     led_yellow_2.value(0) #Turn OFF light YELLOW_2
37     led_red_1.value(0) #Turn ON light RED_1
38     sleep(0) #For 2 sec overlap
39     #Phase 2
40     led_red_2.value(1) #Turn ON light RED_2
41     led_green_1.value(1) #Turn ON light GREEN_1
42     sleep(5) #For 5 sec
43     led_green_1.value(0) #Turn OFF light GREEN_1
44     led_red_2.value(1) #Turn ON light RED_2 while YELLOW_1 is ON
45     led_yellow_1.value(1) #Turn ON light YELLOW_1
46     sleep(2) #For 2 sec
47     led_yellow_1.value(0) #Turn OFF light YELLOW_1
48     led_red_2.value(0) #Turn ON light RED_1
49     sleep(0) #Immediately
50
51 def maintenance():
52     led_yellow_1.value(1) #Turn ON light YELLOW_1
53     led_yellow_2.value(1) #Turn ON light YELLOW_2
54     sleep(.25) #For 0.25 sec
55     led_yellow_1.value(0) #Turn OFF light YELLOW_1
56     led_yellow_2.value(0) #Turn OFF light YELLOW_2
57     sleep(0.25) #For 0.25 sec
58
59 while True: # Infinite Loop
60     START_STATE=start.value()
61     if START_STATE==1:
62         for i in range (1,3):
63             led_yellow_1.value(1) #Turn ON light YELLOW_1
64             led_yellow_2.value(1) #Turn ON light YELLOW_2
65             sleep (0.5) #For 0.5 sec
66             led_yellow_1.value(0) #Turn OFF light YELLOW_1
67             led_yellow_2.value(0) #Turn OFF light YELLOW_2
68             sleep (0.5) #For 0.5 sec
69         while START_STATE==1:
70             #If the state of the START_STATE variable is at level 1 ("start" button pressed), the following instructions are executed
71             print ("START %s - SERVICE %s" % (START_STATE, SERVICE_STATE)) #This is a check that can be useful during the programming phase to see the status of the variables
72             if SERVICE_STATE==0:
73                 #If the state of the SERVICE_STATE variable is at level 0 ("service" switch OFF), the program hands the control to the "on()" function:
74                 on()
75             #Otherwise, if the status of the SERVICE_STATE variable is at level 1 ("service" switch ON), the program hands the control to the "maintenance()" function:
76             else:
77                 maintenance()
78             #Where the "maintenance()" function is executed if the "service" switch was pressed in the previous instruction
79             SERVICE_STATE=service.value()
80             #Where, at the end of the while loop, the status of the "service" switch (Pin 15) is read and sent to the "SERVICE_STATE" variable.
81             #This variable is evaluated in the "if" statement in line 71 and if its value is at 0 then the "on" function is called.
82             #Otherwise if its value is at 1, the "maintenance" function is called.
83             #Note that in the meantime START_STATE variable remained at 1.
84         #Keep in mind that Python does not use curly brackets to contain instructions or functions but uses "indents".
85         #I.e. it reads the position on the listing with respect to the left margin.

```

Figura 19 TL NO overlap buttons

Qui si può scaricare il codice del terzo programma per Raspberry Pi Pico:

3. [TL NO overlap buttons](#)

La **Figura 20** mostra il programma “TL NO overlap buttons” utilizzando tutti i GPIO indicati nelle **Figure 9 e 10**.

4. TL NO overlap buttons

```

TL overlap Buttons.py
1  #Importing libraries
2  from machine import Pin
3  from utime import sleep
4
5  #Definitions
6  led_red_1 = Pin(2, Pin.OUT) #Assigns the variable "led_red_1" the GPIO 2 (pin 1 on the board)
7  led_yellow_1 = Pin(3, Pin.OUT) #Assigns the variable "led_yellow_1" the GPIO 3 (pin 2 on the board)
8  led_green_1 = Pin(4, Pin.OUT) #Assigns the variable "led_green_1" the GPIO 4 (pin 4 on the board)
9  led_red_2 = Pin(10, Pin.OUT) #Assigns the variable "led_red_2" the GPIO 10 (pin 5 on the board)
10 led_yellow_2 = Pin(11, Pin.OUT) #Assigns the variable "led_yellow_2" the GPIO 11 (pin 6 on the board)
11 led_green_2 = Pin(12, Pin.OUT) #Assigns the variable "led_green_2" the GPIO 12 (pin 7 on the board)
12 start = Pin(15, Pin.IN, machine.Pin.PULL_DOWN) #Assigns the variable "start" the GPIO 15 (pin 28 on the board)
13 service = Pin(14, Pin.IN, machine.Pin.PULL_DOWN) #Assigns the variable "service" the GPIO 14 (pin 19 on the board)
14 START_STATE = 0 #Set START_STATE variable to "0"
15 SERVICE_STATE = 0 #Set SERVICE_STATE variable to "0"
16
17 #Light Initialization
18 #All the LEDs are turned off during the initialization phase to be sure the sequence starts with all lights off
19 led_red_1.value(0) #Turn OFF light RED_1
20 led_red_2.value(0) #Turn OFF light RED_2
21 led_yellow_1.value(0) #Turn OFF light YELLOW_1
22 led_yellow_2.value(0) #Turn OFF light YELLOW_2
23 led_green_1.value(0) #Turn OFF light GREEN_1
24 led_green_2.value(0) #Turn OFF light GREEN_2
25 sleep(1) #Wait 1 second
26
27 def on():
28     #Phase 1
29     led_red_1.value(1) #Turn ON light RED_1
30     led_green_2.value(1) #Turn ON light GREEN_2
31     sleep(5) #For 5 sec
32     led_green_2.value(0) #Turn OFF light GREEN_2
33     led_red_1.value(1) #Turn ON light RED_1 while YELLOW_2 is ON
34     led_yellow_2.value(1) #Turn ON light YELLOW_2
35     sleep(2) #For 2 sec
36     led_yellow_2.value(0) #Turn OFF light YELLOW_2
37     led_red_2.value(1) #Turn ON light RED_2
38     sleep(2) #For 2 sec overlap
39     led_red_1.value(0) #Turn OFF light RED_1
40     led_red_2.value(0) #Turn OFF light RED_2
41     sleep(0) #Immediately
42     #Phase 2
43     led_red_2.value(1) #Turn ON light RED_2
44     led_green_1.value(1) #Turn ON light GREEN_1
45     sleep(5) #For 7 secs.
46     led_green_1.value(0) #Turn OFF light GREEN_1
47     led_red_2.value(1) #Turn ON light RED_2 while YELLOW_1 is ON
48     led_yellow_1.value(1) #Turn ON light YELLOW_1
49     sleep(2) #For 2 sec
50     led_yellow_1.value(0) #Turn OFF light YELLOW_1
51     led_red_1.value(1) #Turn ON light RED_1
52     sleep(2) #For 2 sec overlap
53     led_red_1.value(0) #Turn OFF light RED_1
54     led_red_2.value(0) #Turn OFF light RED_2
55     sleep(0) #Immediately
56
57 def maintenance():
58     led_yellow_1.value(1) #Turn ON light YELLOW_1
59     led_yellow_2.value(1) #Turn ON light YELLOW_2
60     sleep(.25) #For 0.25 sec
61     led_yellow_1.value(0) #Turn OFF light YELLOW_1
62     led_yellow_2.value(0) #Turn OFF light YELLOW_2
63     sleep(0.25) #For 0.25 sec
64
65 while True: # Infinite loop
66     START_STATE=start.value()
67     if START_STATE==1:
68         for i in range(1,3):
69             led_yellow_1.value(1) #Turn ON light YELLOW_1
70             led_yellow_2.value(1) #Turn ON light YELLOW_2
71             sleep(.5) #For 0.25 sec
72             led_yellow_1.value(0) #Turn OFF light YELLOW_1
73             led_yellow_2.value(0) #Turn OFF light YELLOW_2
74             sleep(0.5) #For 0.25 sec
75         while START_STATE==1:
76             #If the state of the START_STATE variable is at level 1 ("start" button pressed), the following instructions are executed
77             print ("START %s - SERVICE %s" % (START_STATE, SERVICE_STATE)) #This is a check that can be useful during the programming phase to see the status of the variables
78             if SERVICE_STATE==0:
79                 #If the state of the SERVICE_STATE variable is at level 0 ("service" switch OFF), the program hands the control to the "on()" function:
80                 on()
81                 #Where the "on()" function is executed if the "service" switch was at level 0 (OFF) in the previous instruction
82             else:
83                 #Otherwise, if the status of the SERVICE_STATE variable is at level 1 ("service" switch ON), the program hands the control to the "maintenance()" function:
84                 maintenance()
85                 #Where the "maintenance()" function is executed if the "service" switch was pressed in the previous instruction
86             SERVICE_STATE=service.value()
87             #Where, at the end of the while loop, the status of the "service" switch (Pin 15) is read and sent to the "SERVICE_STATE" variable.
88             #This variable is evaluated in the "if" statement in line 71 and if its value is at 0 then the "on" function is called.
89             #Otherwise if its value is at 1, the "maintenance" function is called.
90             #Note that in the meantime START_STATE variable remained at 1.
91
92 #Keep in mind that Python does not use curly brackets to contain instructions or functions but uses "Indents".
93 #i.e. it reads the position on the listing with respect to the left margin |

```

Figura 20 TL NO overlap buttons

Qui si può scaricare il codice del quarto programma Raspberry Pico:

4. [TL NO overlap buttons](#)

Terminata la programmazione si può passare alla fase di test, utilizzando il prototipo con un'unica breadboard (**Figura 11**) oppure realizzare i disegni del tracciato stradale e la costruzione della piattaforma (**Figure 12 e 13**) come indicato di seguito.

Disegno e realizzazione della planimetria di un incrocio a due strade

La planimetria dell'incrocio semaforico può essere realizzata con un foglio di carta delle dimensioni, come nel nostro caso, 28x34 cm., oppure utilizzando un semplice foglio A4, come indicato nelle **Figure 21-22-23**

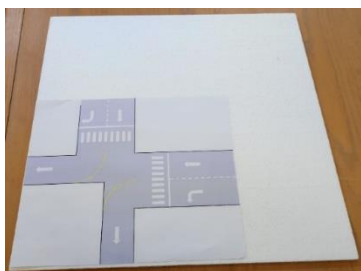


Figura 21

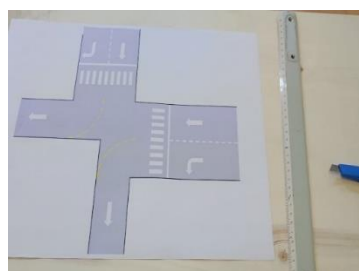


Figura 22



Figura 23

Il progetto può essere realizzato a mano con pennarelli o matite colorate oppure utilizzando un programma di disegno e stampandolo; nel nostro caso abbiamo utilizzato MS Word. Di seguito sono mostrati tre diversi layout:

- “Draft Layout_2 luci” **Figura 24**
- “Draft Layout_4 luci” **Figura 25**
- “Layout a semaforo per il montaggio” **Figura 26**

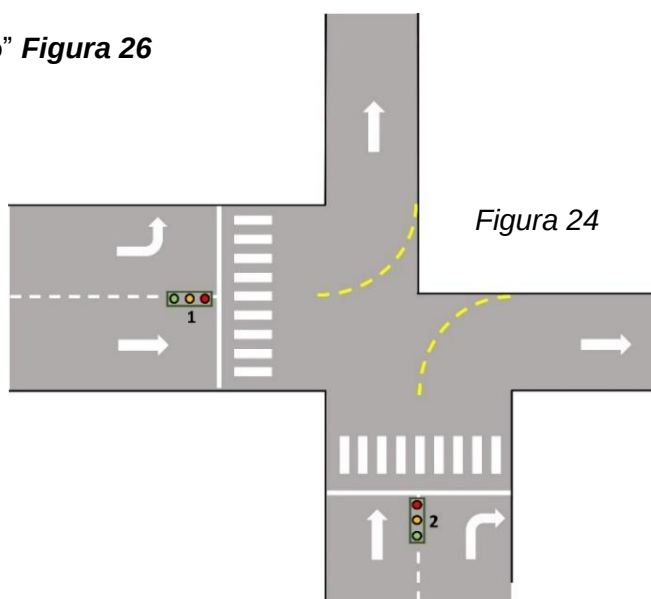


Figura 24

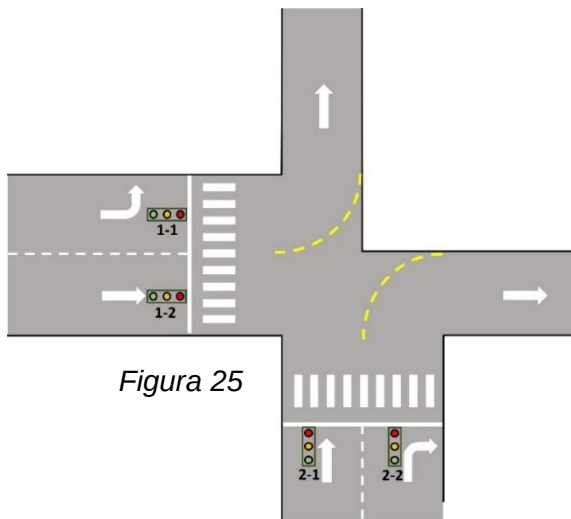


Figura 25

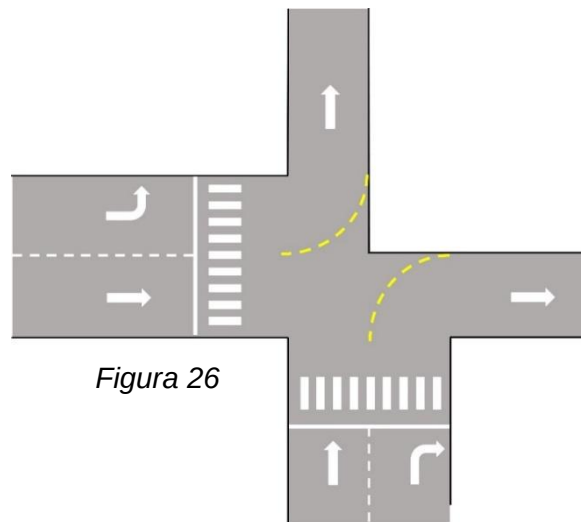


Figura 26

Cliccando sui link sottostanti è possibile scaricare il Layout in vari formati:

Link al Layout realizzato con MS Word:

- [Bozza Layout 2 luci](#)
- [Bozza Layout 4 luci](#)
- [Disposizione del semaforo per il montaggio](#)

Link al Layout realizzato in formato PDF:

- [Bozza Layout 2 luci](#)
- [Bozza Layout 4 luci](#)
- [Disposizione del semaforo per il montaggio](#)

Link al Layout realizzato in formato JPG:

- [Bozza Layout 2 luci](#)
- [Bozza Layout 4 luci](#)
- [Disposizione del semaforo per il montaggio](#)

Dopo aver creato il layout cartaceo sarà necessario realizzare una base in polistirolo alta 1 cm dove andranno inserite le breadboard. Il foglio di polistirolo deve avere le dimensioni del foglio di carta (si può usare un foglio A4) usato per creare il layout grafico o leggermente più grande (**Figure 27-28**)



Figura 27

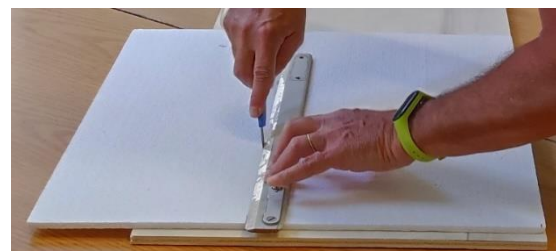


Figura 28

Il layout e i fogli di polistirolo tagliati a misura sono mostrati in **Figura 29**.

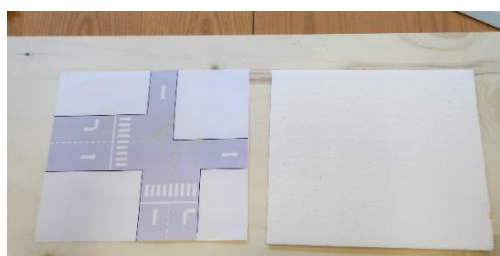


Figura 29

Il prossimo passo sarà segnare la posizione esatta delle breadboard su un disegno dell'incrocio utilizzando il layout creato in precedenza **Figura 30-31**.

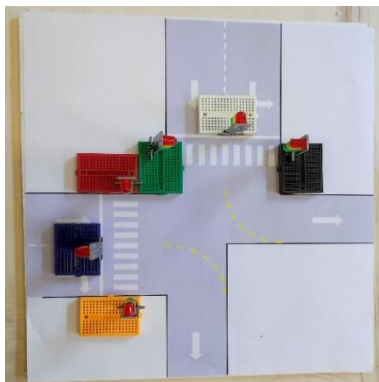


Figura 30

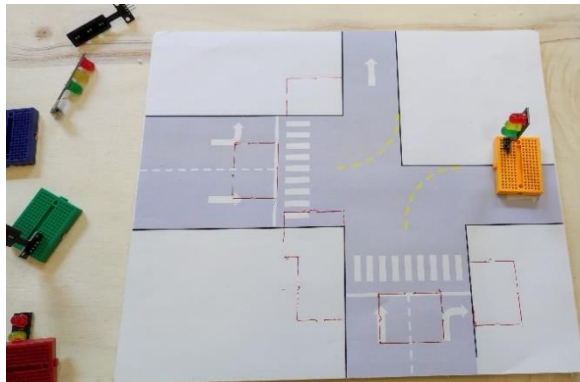


Figura 31

Una volta segnati sul disegno i contorni esatti delle breadboard, dovremo ritagliare i contorni utilizzando un taglierino **Figura 32**

Adesso dobbiamo tracciare i contorni sulla lastra di polistirolo usando il disegno che abbiamo appena modellato come riferimento e guida **Figura 33**

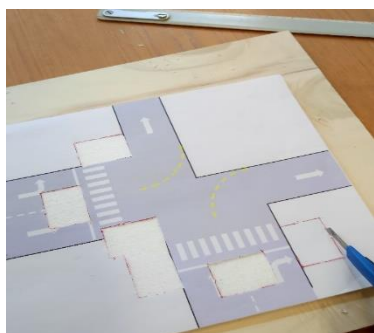


Figura 32

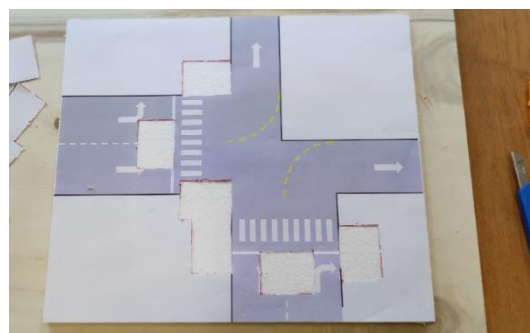


Figura 33

Ora siamo pronti per ritagliare le fessure dove troveranno alloggio le breadboard nel foglio di polistirolo **Figura 34-35-36**.

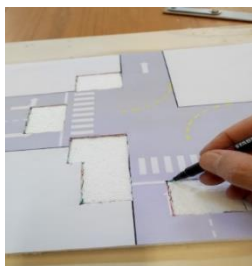


Figura 34

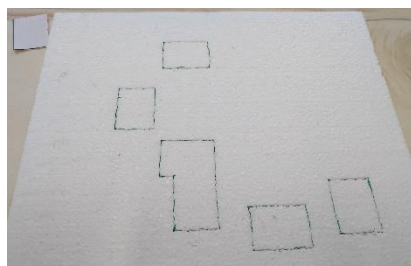


Figura 35

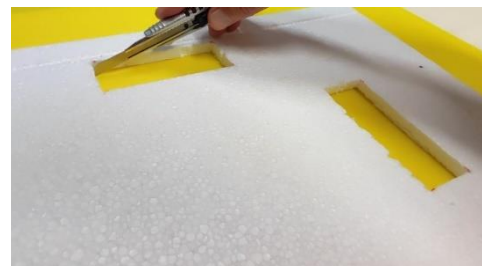


Figura 36

Sarà necessario prestare un po' di attenzione nel sagomare perfettamente le fessure nella lastra di polistirolo, in modo che le breadboard vi si incastrino perfettamente.

Una volta ritagliate le fessure nei fogli di polistirolo, si dovranno inserire le breadboard, ognuna nel proprio alloggiamento, come mostrato in **Figura 37**.

Il risultato di questo passaggio intermedio è mostrato in **Figura 38**.

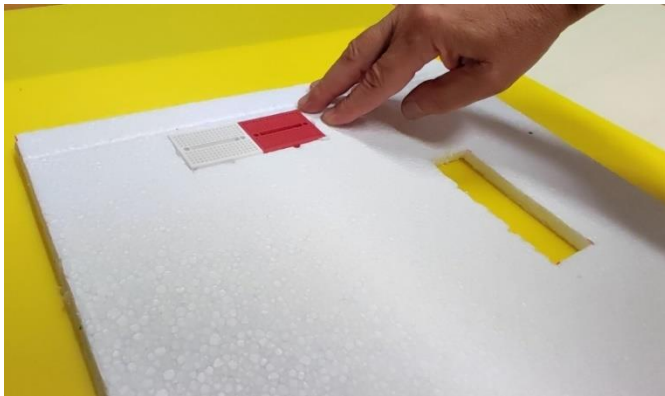


Figura 37

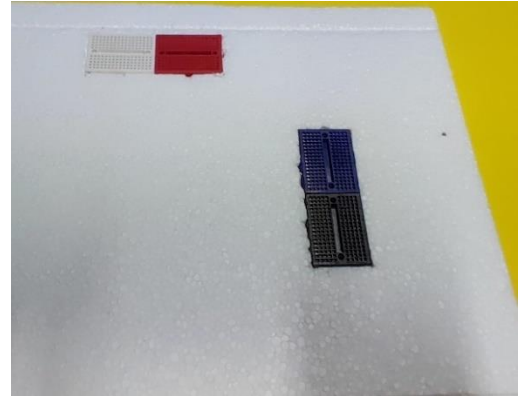


Figura 38

Possiamo ora appoggiare il layout cartaceo precedentemente realizzato sulla base di polistirolo e procedere all'inserimento dei mini semafori a led sulle relative breadboard, seguendo i passaggi mostrati nelle **Figure 39 e 40**.

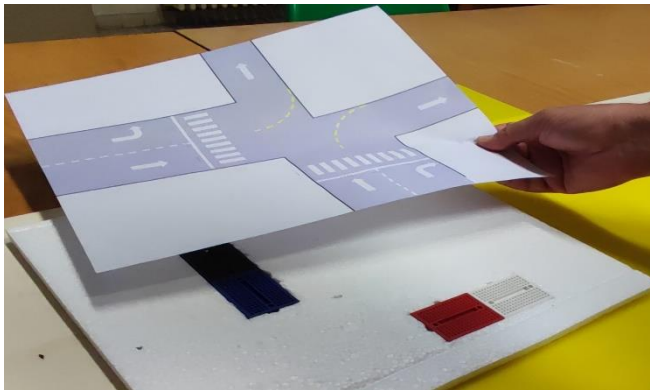


Figura 39

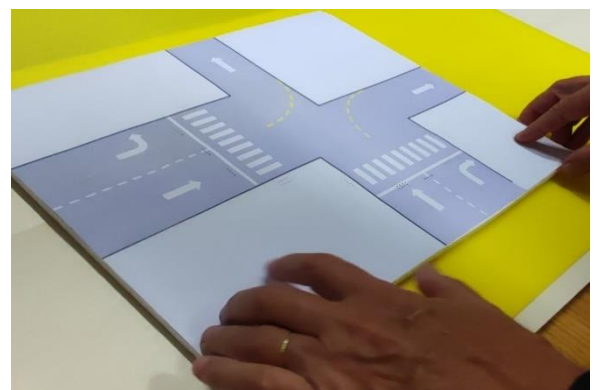


Figura 40

Sotto il sistema completo con i due semafori a LED inseriti nelle breadboard (**Figura 41**).

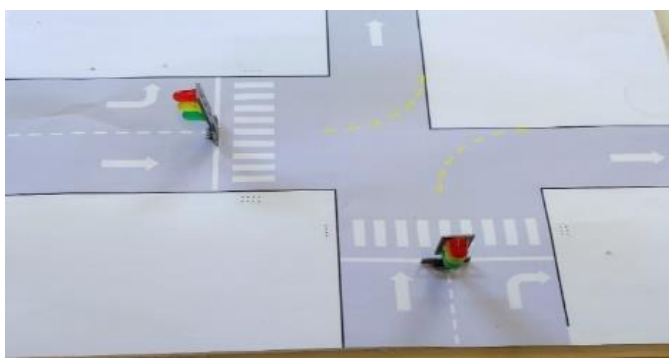


Figura 41

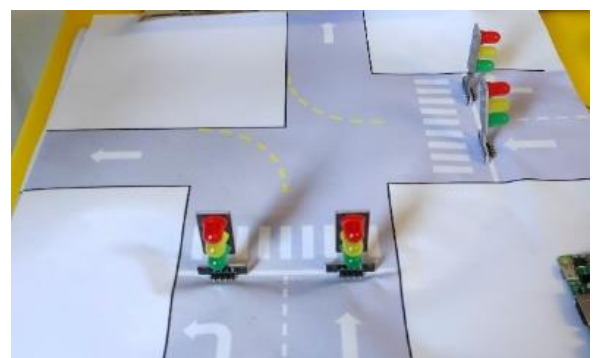


Figura 42

In **Figura 42** è mostrato un altro esempio dello stesso incrocio ma con due semafori per ogni strada.

Ora possiamo collegare i dispositivi LED agli I/O della scheda RPi Pico.

Sarà necessario piegare di 90° i piedini dei cavetti da inserire nella breadboard, in modo da poterli ricoprire con il disegno planimetrico, come mostrato nelle **Figure 43-44-45**.



Figura 43



Figura 44

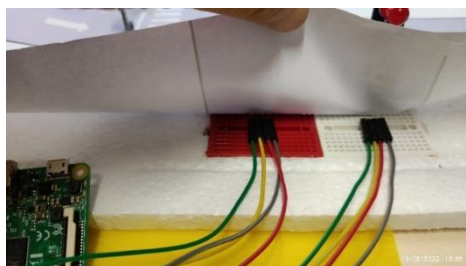


Figura 45

In **Figura 46** un esempio del sistema completo finale.

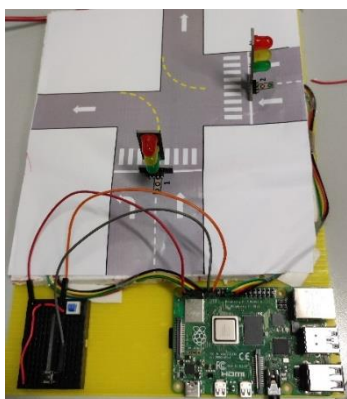


Figura 46

Note sui collegamenti di pulsanti e interruttori

Il PinOut del pulsante utilizzato nel prototipo è mostrato in **Figura 47**:



Figura 47

Si noti che i pin 1-3 (blu nella figura) sono collegati tra loro, così come i pin 2-4 (rossi nella figura). La connessione tra i pin 1-2 è NORMALMENTE APERTA e si chiuderà solo alla pressione del pulsante. Lo stesso vale per i pin 3-4.

Il PinOut dell'interruttore utilizzato nel prototipo è mostrato in **Figura 48**:

Si usino le barre verticali su entrambi i lati come riferimento (su un lato ha solo una barra, l'altro lato ha due barre)

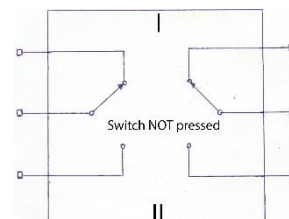


Figura 48

Note sulle resistenze di pull-up e pull-down

I resistori di pull-up (o pull-down) sono resistori a valore fisso, tra l'alimentazione (o massa) e un particolare pin in un circuito logico digitale. Comunemente abbinato agli interruttori, il suo scopo è garantire che la tensione tra massa (GND) e Vcc sia controllata attivamente quando l'interruttore è aperto. Senza il resistore di pull-up lo stato logico dell'interruttore aperto è flottante, quindi potrebbe essere uno '0' o un '1', perché non essendo connesso a un potenziale stabile, si comporta come un'antenna e la tensione ai suoi capi è fluttuante, indeterminata.

Di seguito alcuni esempi di come collegare la resistenza di pull-up (o pull-down).

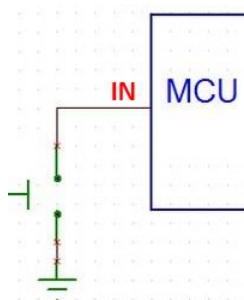


Figure 49
Senza
resistore

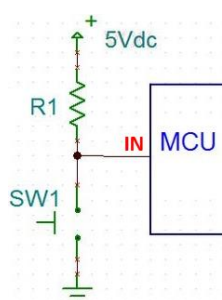


Figure 50
Resistore Pull-Up
Switch non premuto

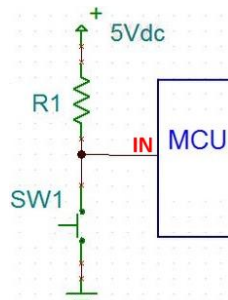


Figure 51
Resistore Pull-Up
Switch premuto

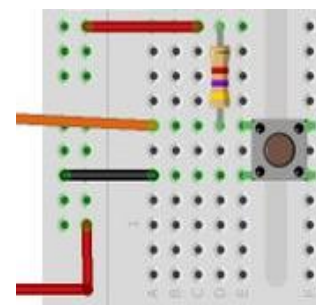


Figure 52
Resistore Pull-Up
sulla breadboard

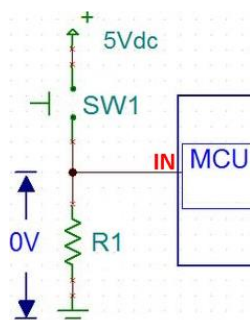


Figure 53
Resistore Pull-Down
Switch non premuto

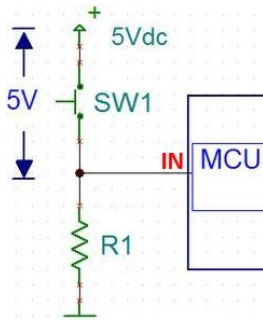


Figure 54
Resistore Pull-Down
Switch premuto

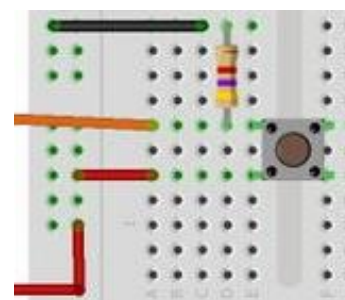


Figure 55
Pull-Down resistor
on the breadboard

Quale dovrebbe essere il valore della resistenza?

Supponiamo di voler limitare la corrente a 1mA. Ad esempio se $V_{cc} = 5V$, utilizzando la legge di Ohm: $R = V/I$ otteniamo $R = 5V/1mA$ (oppure $R = 5000mV/1mA$) quindi $R = 5000\Omega = 5k$. In questo caso possiamo utilizzare una resistenza da 4.7k Ω . Normalmente è possibile utilizzare, in generale, un resistore da 10k Ω , questo richiederà solo 0,5mA.

Normalmente in tutti i sistemi a microcontrollore le resistenze di pull-up e pull-down sono già integrate all'interno della scheda stessa. I pin di ingresso possono essere programmati per essere configurati come ingressi pull-up o pull-down con semplici istruzioni.

Esempio di Pull-down: `Start = Pin(15, Pin.IN, machine.Pin.PULL_DOWN)`

Esempio di Pull-up: `Start = Pin(15, Pin.IN, machine.Pin.PULL_UP)`

Si veda la Guida RPi Pico per ulteriori informazioni.

File di supporto

Foglio di lavoro per gli studenti

[Il foglio di lavoro per lo studente](#)

Programmi

[La cartella con tutti i programmi](#)

Figure

[La cartella con tutte le figure](#)

Video

[La cartella con tutti i video](#)

Guida RPi Pico

Iniziare con RPi Pico:

<https://projects.raspberrypi.org/en/projects/iniziare-con-il-pico>

Documentazione RPi Pico:

<https://www.raspberrypi.com/documentation/microcontrollori/raspberry-pi-pico.html>

Guida IDE Thonny

Documentazione dell'IDE Thonny:

<https://thonny.org/>