ITT G. CHILESOTTI

LASERTAG

Anno scolastico 2017-2018

Mazzaro Roberto 5^B

Indice:

1. Introduzione

Cerchiamo innanzitutto di capire cos'è il lasertag.

Il lasertag è un gioco di squadra basato sulla simulazione militare.
L'attività simula, tramite apposite apparecchiature, combattimenti tra squadre avversarie. Gli obbiettivi sono vari, ma fondamentalmente lo scopo del gioco è colpire l'avversario fino alla sua eliminazione.
Ogni giocatore è munito di un kit a raggi infrarossi, totalmente innocui, collegato a dei sensori, posti generalmente sul capo o sul corpo, che devono essere colpiti.
È una tecnologia di derivazione militare, infatti è nata per l'addestramento delle forze militari.

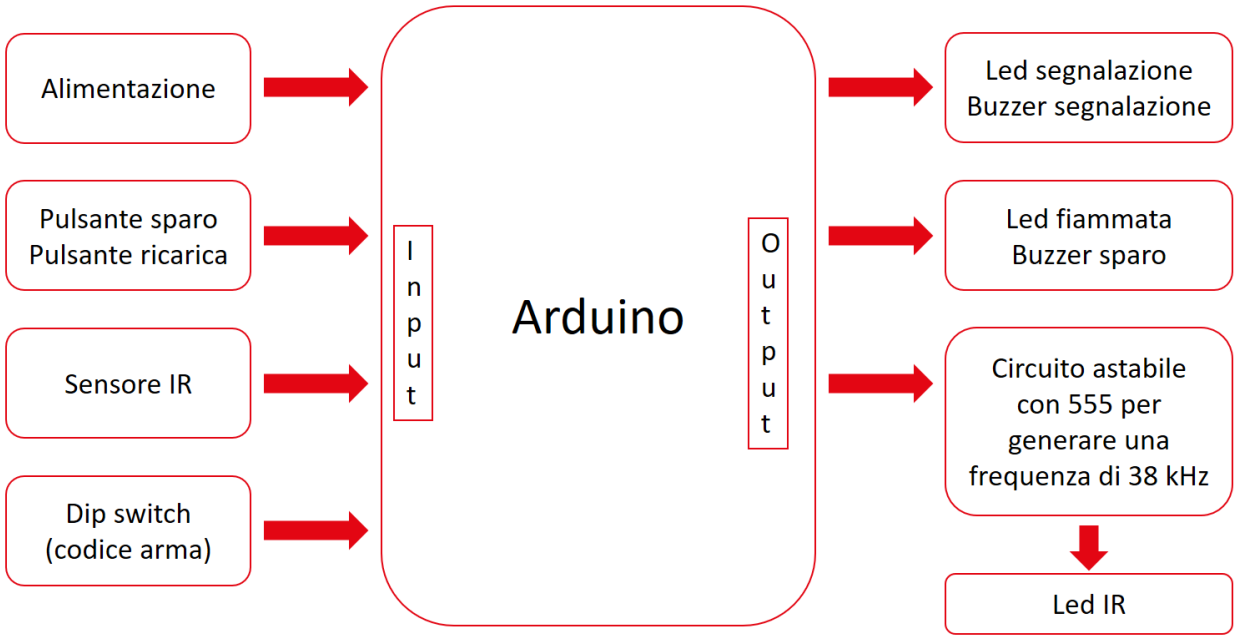
2. Fasi del progetto

Il principale problema da risolvere per far funzionare questo progetto è capire come far comunicare due Arduino, ovvero il controllore utilizzato per sviluppare questo progetto, tramite un fascio infrarosso.
Risolto questo problema si può pensare al resto
Le fasi del progetto sono le seguenti

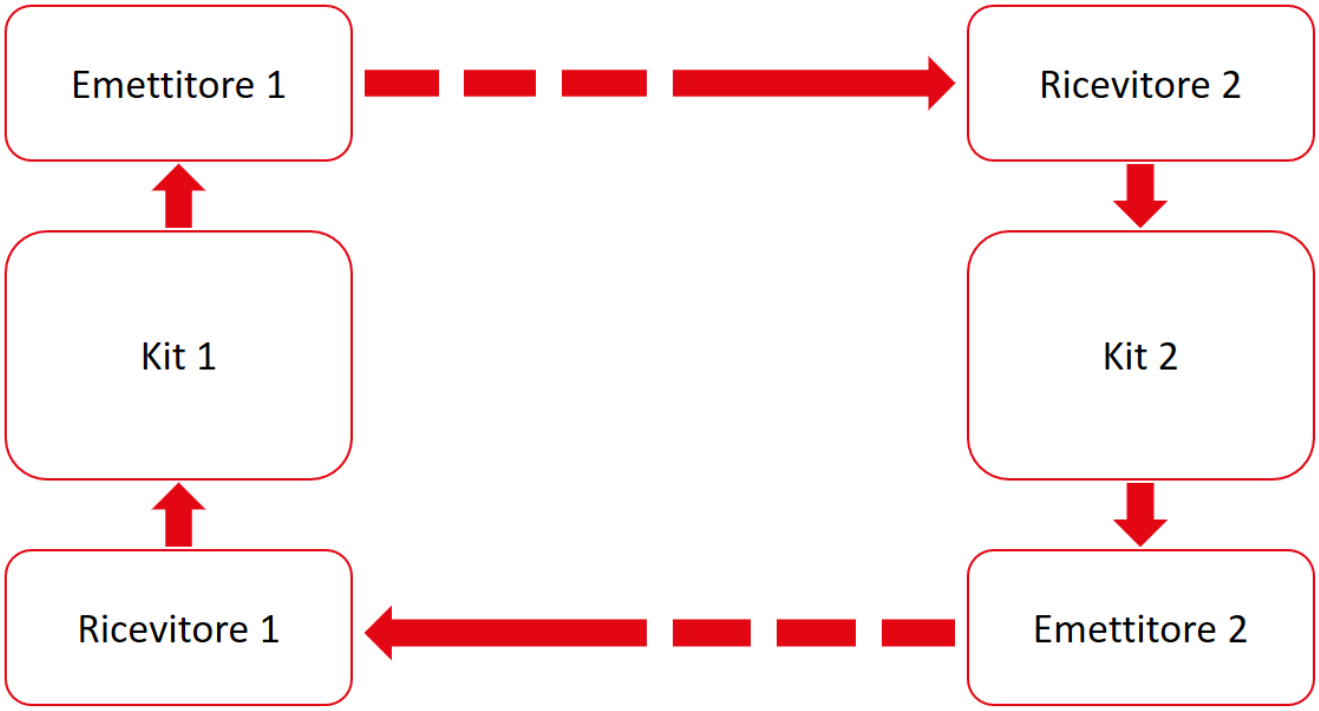
- Capire come far comunicare due Arduino tramite la funzione PulseIn()
- Capire come inserire la comunicazione con l'infrarosso
- Montare un prototipo su breadboard
- Scrivere le singole funzioni utilizzate dal software in linguaggio C
- Disegnare, sviluppare e montare il PCB
- Mettere a punto il programma generale
- Disegnare e realizzare la struttura con la stampante 3D

3. Schema a blocchi

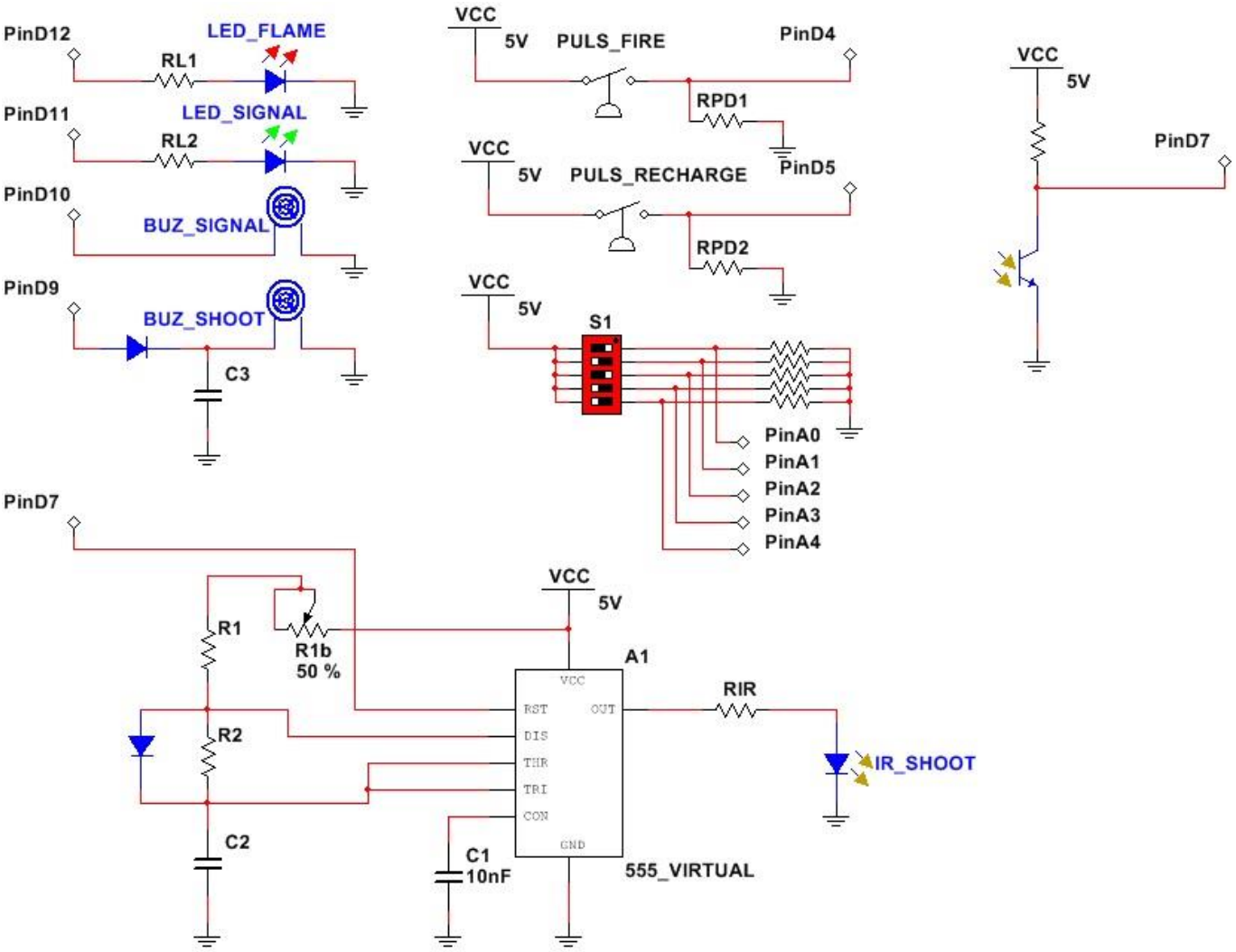
3.1 Schema a blocchi di un kit



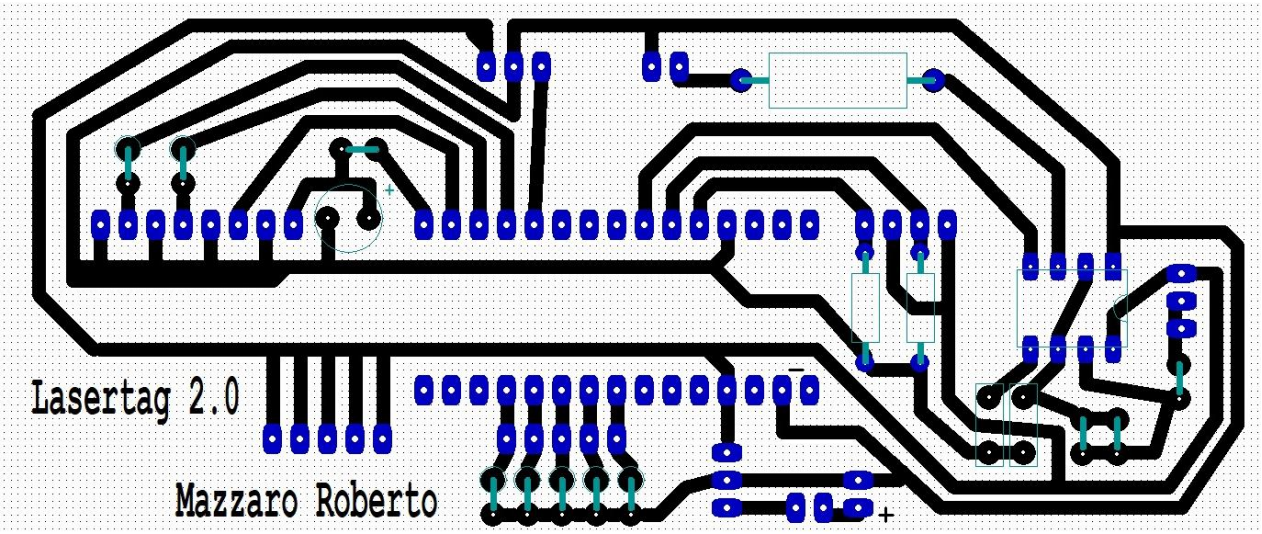
3.2 Schema a blocchi comunicazione tra due kit

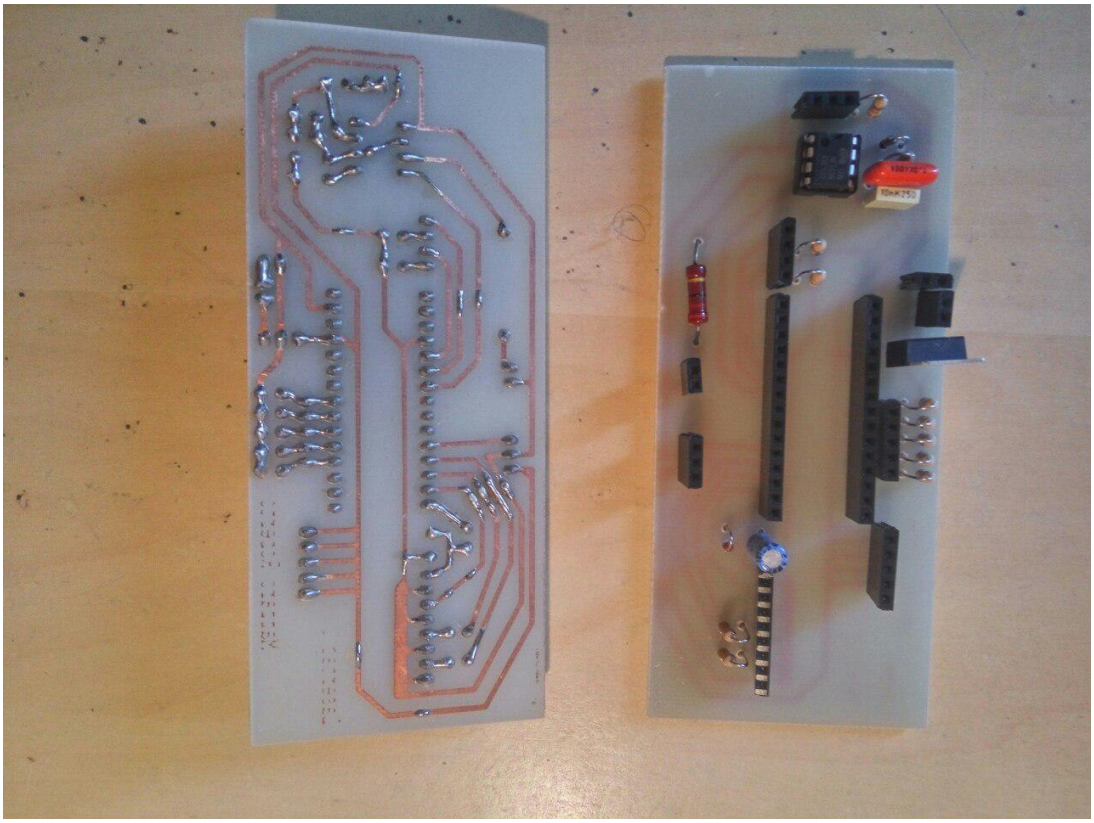


4. Schema elettrico



5.PCB





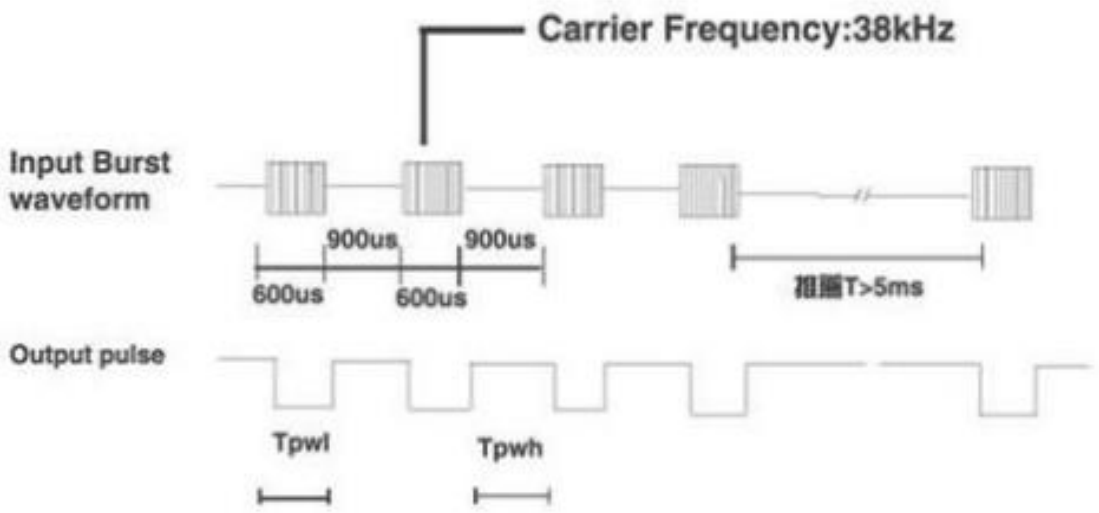
6. Hardware

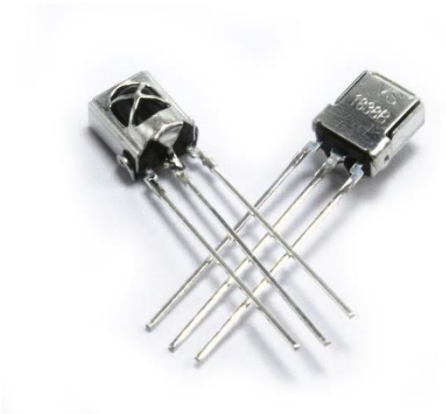
Di seguito vengono analizzate le parti principali dell'hardware.

6.1 Sensore IR

Il sensore infrarosso impiegato in ricezione è il TL1838.
È stato scelto perché è quello che si trova più facilmente in commercio.
Le caratteristiche peculiari di questo sensore sono:

- Frequenza di lavoro di 38kHz
- La durata minima di un impulso affinchè possa essere letto è di 600µS
- Quando riceve segnale in uscita fornisce 0V
- Quando non riceve il segnale adeguato in uscita fornisce 5V





6.2 Hardware trasmissione

Dato che il sensore per funzionare correttamente necessita che gli venga fornito un segnale con una portante a 38kHz e dato che tutti i timer che mette a disposizione Arduino erano già impiegati per svolgere altre funzioni la frequenza di 38kHz viene generata tramite un 555 in configurazione astabile. Andando a pilotare il piedino di reset del 555 tramite Arduino si decide quando generare la frequenza di 38kHz che verrà spedita al kit avversario tramite un led IR collegato con un'opportuna resistenza al piedino d'uscita del 555.

6.3 Dimensionamento circuito astabile

f= 38kHz

$T = \frac{1}{f} = 26.3\mu S$

$T = 0.7 * [(R_1 + R_2) * C]$

Scelgo C= 10nF

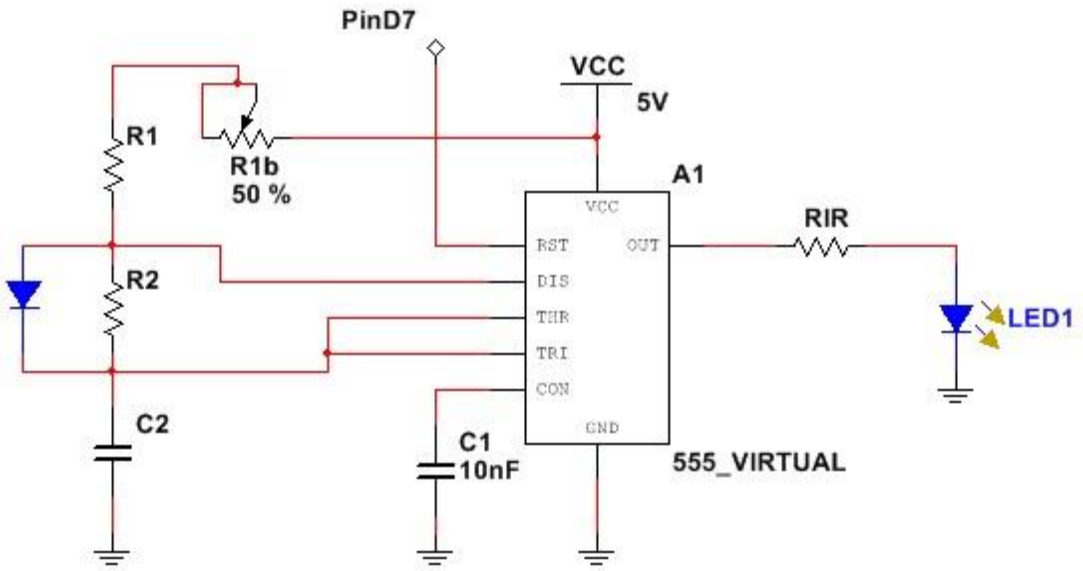
$R_1 + R_2 = \frac{T}{0.7*C} = 37571\Omega$

Scelgo R_2= 15kΩ

R_1= 12kΩ

R_{1B}= trimmer da 10kΩ (per la regolazione dei 38kHz)

6.4 Schema elettrico circuito astabile con 555



7. Software

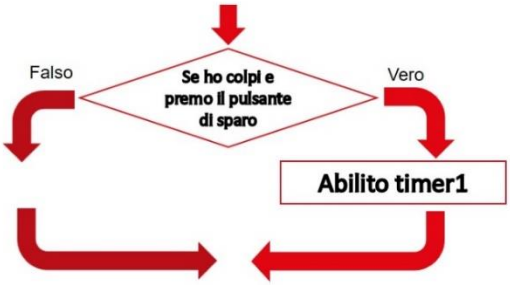
Di seguito viene analizzato il software nelle sue componenti principali cercando di spiegare l'idea su cui esso si basa.

7.1: Acquisizione codice kit

Questa parte del programma viene eseguita una volta sola all'accensione perché si trova nel ´void setup()´.
Si va a fare la lettura analogica di 5 piedini che vanno da A0 ad A4,
questi piedini sono collegati ad un dip switch tramite il quale si può scegliere il codice del kit.
Vengono utilizzati dei piedini analogici per fare una lettura digitale perché erano inutilizzati e in Arduino ci sono a disposizione pochi piedini digitali.
Dato che al momento sono stati acquisiti dei valori analogici questi devono essere convertiti in digitale, fatto ciò si ricava il codice della squadra e il codice dell'arma del kit, i primi due bit indicano il valore della squadra, mentre i restanti tre indicano il valore dell'arma.
In base al codice dell'arma si impostano le sue caratteristiche quali la vita, il rateo di fuoco, il numero di caricatori, i colpi per caricatore.

7.2 Funzione di sparo

La funzione di sparo è la funzione che si occupa di spedire il codice del kit al kit avversario.
È gestita con il timer1 che genera una frequenza che dipende dal tipo di arma selezionata, ogni volta che c'è il fronte di salita si spedisce il codice.
Per spedire il codice si agisce sul piedino di reset del 555, quando lo poniamo allo stato alto il 555 genera la frequenza di 38kHz.
Ad inizio trasmissione viene spedito il bit di start e per far ciò il reset del 555 viene abilitato per 1000µS.
Successivamente si spediscono il codice della squadra e quello dell'arma; per spedire un bit a 1 si abilita il reset del 555 per 800µS, mentre per spedire un bit a 0 lo si abilita per 600µS.
Dato che sia che si stia spedendo il bit di start uno 0 o un 1 si sta sempre spedendo un segnale infrarosso a 38kHz, solo che con diversa durata, per separare un bit dall'altro non si spedisce niente per 600µS.
Ogni volta che si spara, oltre a spedire il codice con il metodo precedentemente spiegato, si attivano anche il buzzer di sparo ed il led di fiammata e si scala un colpo dai colpi nel caricatore.
Gestire questa funzione con un interrupt è importante perché il programma deve essere sempre pronto a riceve un colpo avversario e per far ciò il programma non deve 'bloccarsi', quindi non ci possono essere delay inutili o eccessivamente lunghi.



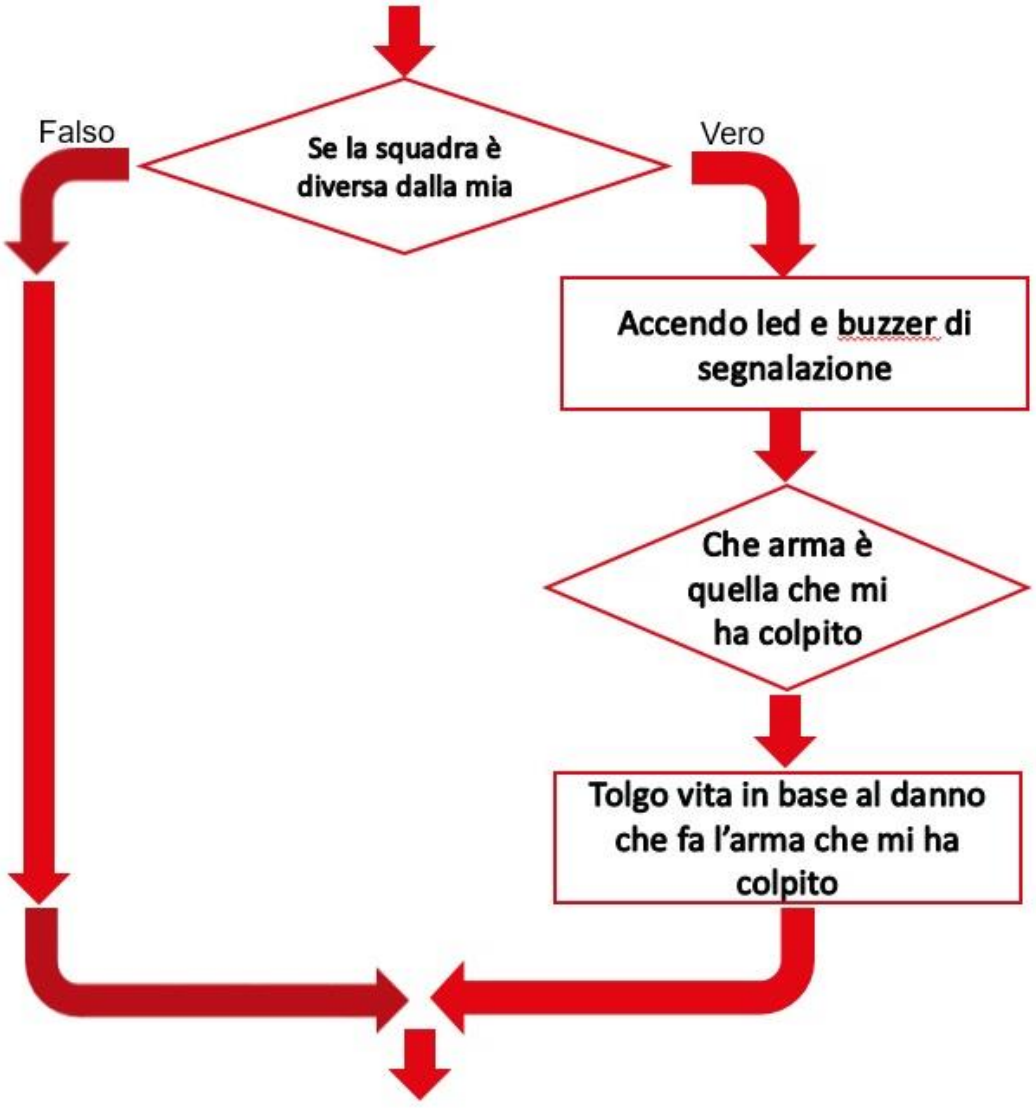
7.3 Funzione di ricezione

La funzione di ricezione si basa sulla funzione pulseIn(), che è una funzione già esistente nelle librerie di Arduino.
PulseIn() è una funzione che si occupa di restituire il valore in µS della durata dello stato alto o di quello basso in base al fatto che si decida di prendere come riferimento il fronte di salita o quello di discesa.
Viene utilizzata la funzione pulseIn() con l´opzione, già presente in essa, di inserire un timeout ovvero un tempo massimo entro il quale se non arriva il fronte desiderato la funzione pulseIn() procede oltre e ci restituisce come risultato 0µS.
I timeout sono necessari perché il programma non può restare ad aspettare un colpo che non si sa quando arriva e dobbiamo poter svolgere anche le altre azioni.
I timeout, se non vengono impostati, di base sono ad 1S.

Per ricevere il codice bisogna innanzitutto individuare il bit di start, per far ciò c'è un ciclo while che lavora fino a quando pulseIn() non rileva uno stato basso della durata di almeno 950µS, questo perché il bit di start dura 1000µS e ci vuole un po' di margine;
all'interno del ciclo while si trova il resto del programma.
Rilevato il bit di start si esce dal ciclo while e tramite altri 5 pulseIn() si vanno ad acquisire le durate degli altri impulsi che successivamente vengono convertiti in 0 e 1, fatto ciò si ricavano il codice della squadra e dell'arma di chi ci ha colpito e si richiama la funzione di aggiornamento vita.

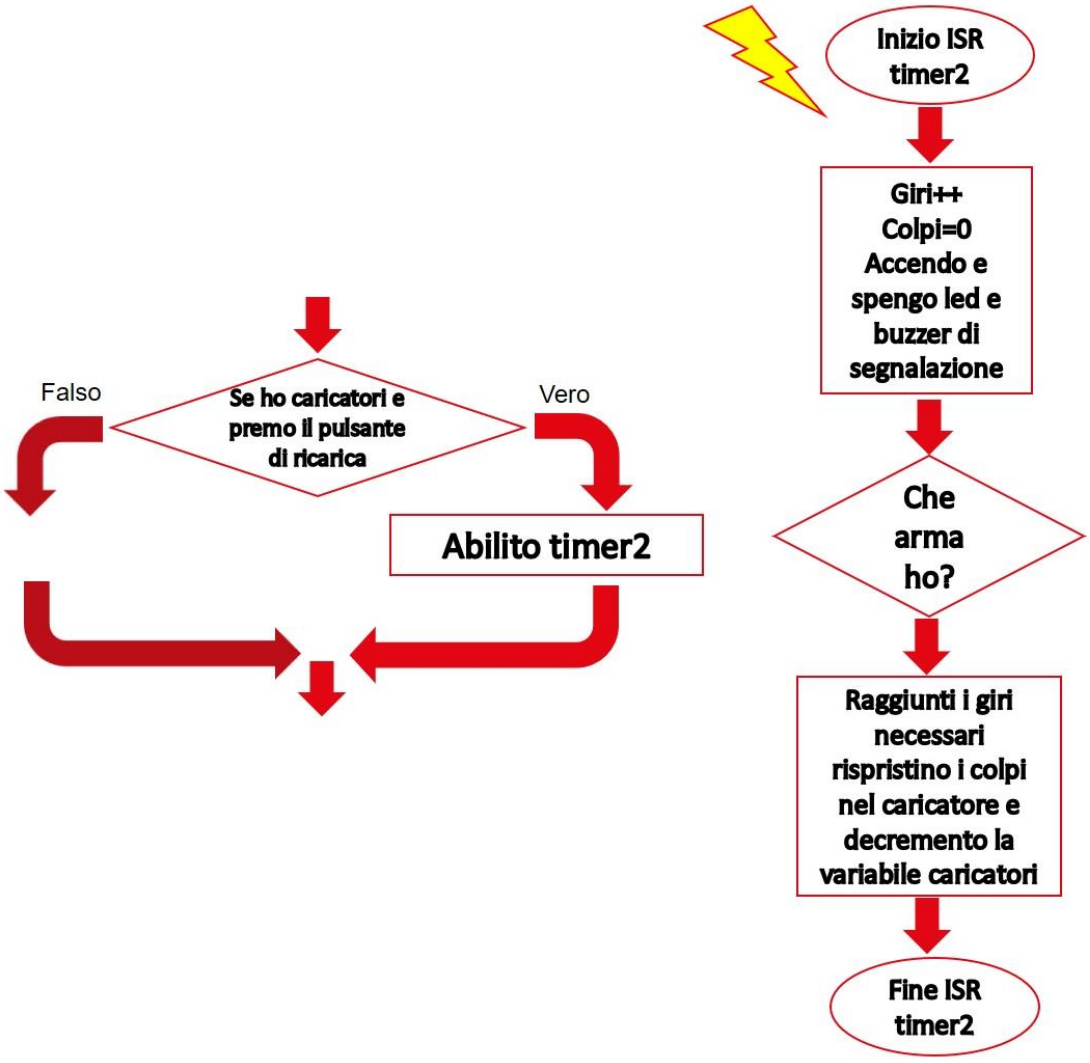
7.4 Funzione aggiornamento vita

Questa funzione viene eseguita dopo che si è stati colpiti e che i codici della squadra e dell'arma di colui che ci ha colpito sono stati ricavati.
Innanzitutto si fa un controllo per verificare che la squadra del kit che ha ricevuto il colpo sia diversa da quella del kit che l'ha spedito, se sono diverse allora, in base al codice dell'arma che ci ha colpito, si decrementa la variabile vita in base al danno che fa quell'arma, inoltre si accende il led ed il buzzer di segnalazione per segnalare che siamo stati colpiti.
Nel caso in cui le due squadre siano uguali il colpo non viene registrato e non si perde vita; è stato deciso di disabilitare il fuoco amico perché dato che il colpo è spedito tramite un fascio infrarosso che può essere riflesso c'era il rischio di colpirsi da soli.



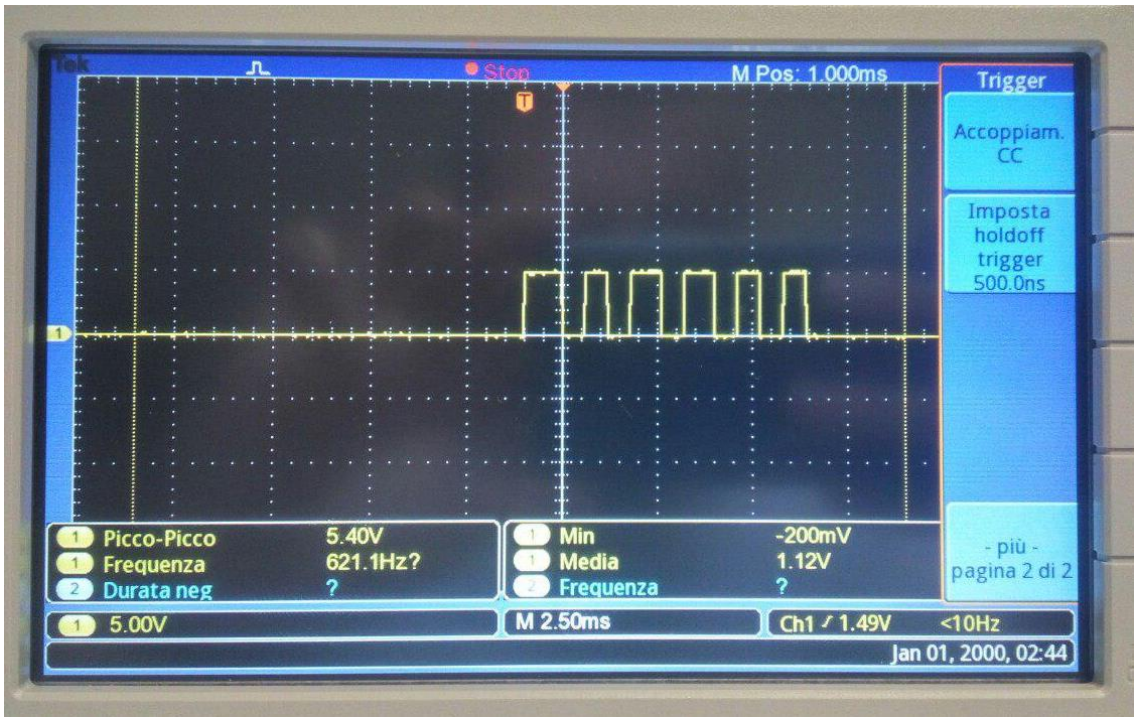
7.5 Funzione di ricarica

La funzione di ricarica viene eseguita quando si preme il pulsante di ricarica se si hanno ancora caricatori a disposizione.
Appena premuto il pulsante di ricarica si azzera la variabile colpi, così non è possibile sparare, inoltre si accendono e si spengono il led ed il buzzer di segnalazione.
Questa funzione è gestita con il timer2 impostato ad una frequenza fissa di 2kHz, per ottenere i vari tempi di ricarica richiesti dalle varie armi si aspetta che vengano eseguiti un certo numero di cicli dal timer, quindi c'è una variabile che tiene conto di quanti giri fa il timer e raggiunto il numero di cicli necessari si rispristinano i colpi nel caricatore e si decrementa la variabile caricatori.
È stato necessario adottare questa soluzione per il conteggio del tempo di ricarica perché il timer2 è un timer a soli 8 bit e con esso non è possibile ottenere i periodi lunghi necessari in questa funzione.

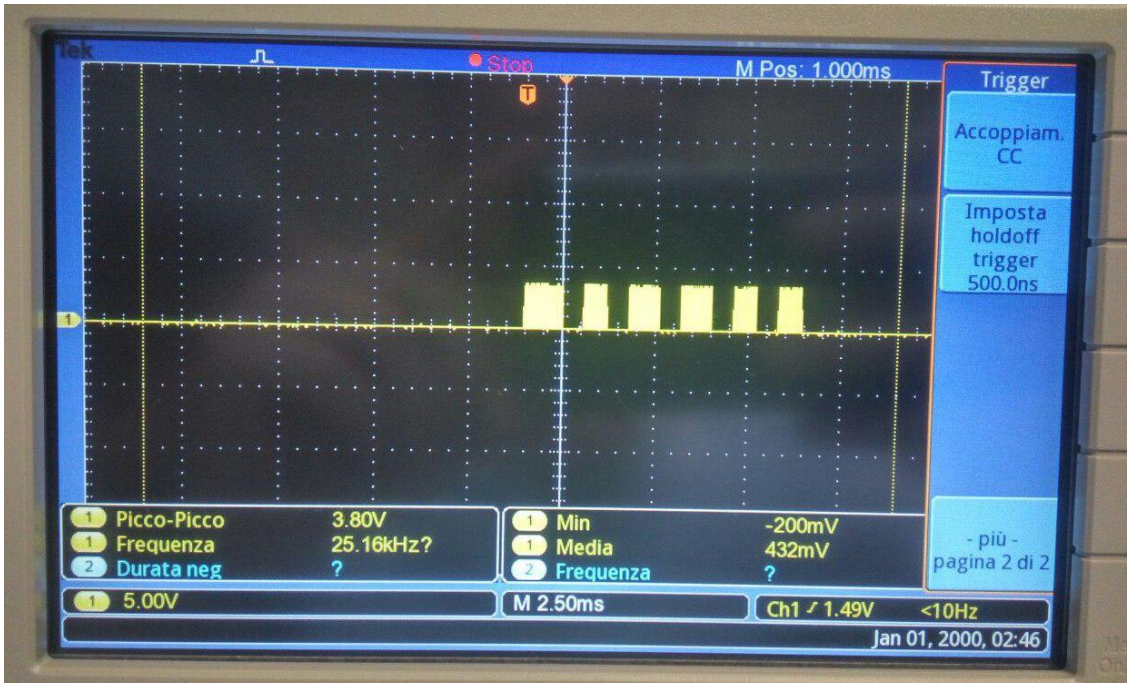


8. Misure con oscilloscopio

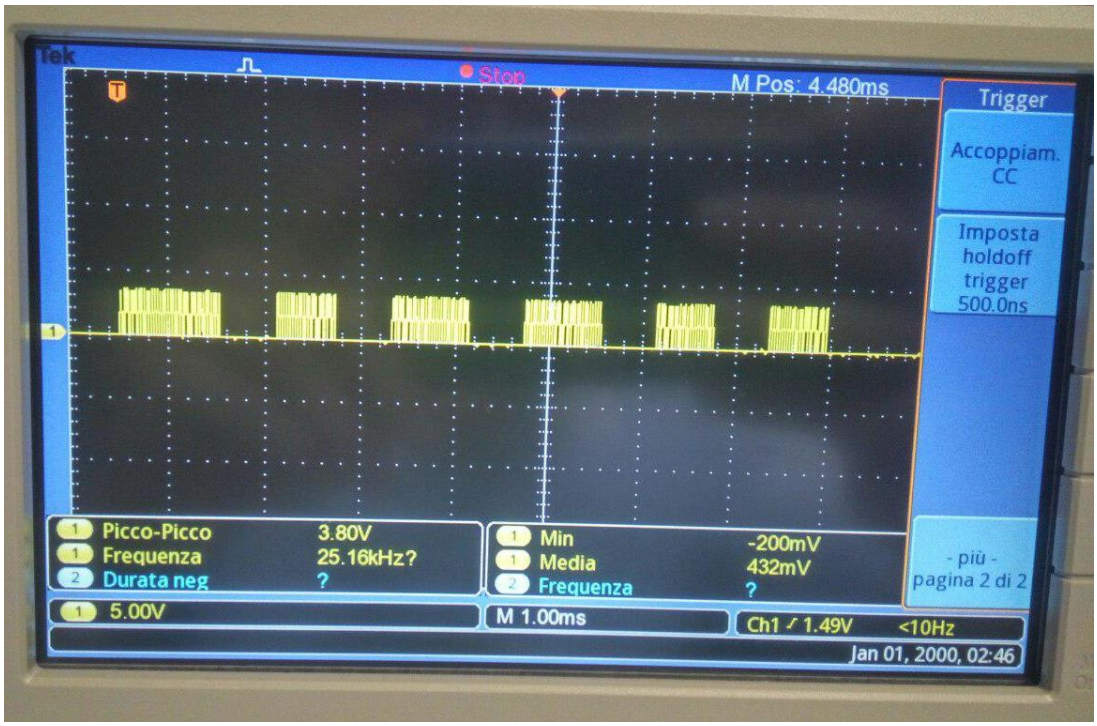
Segnale in uscita da Arduino che pilota il reset del 555



Segnale in uscita dal 555 che pilota il led IR



Segnale trasmesso dal led IR

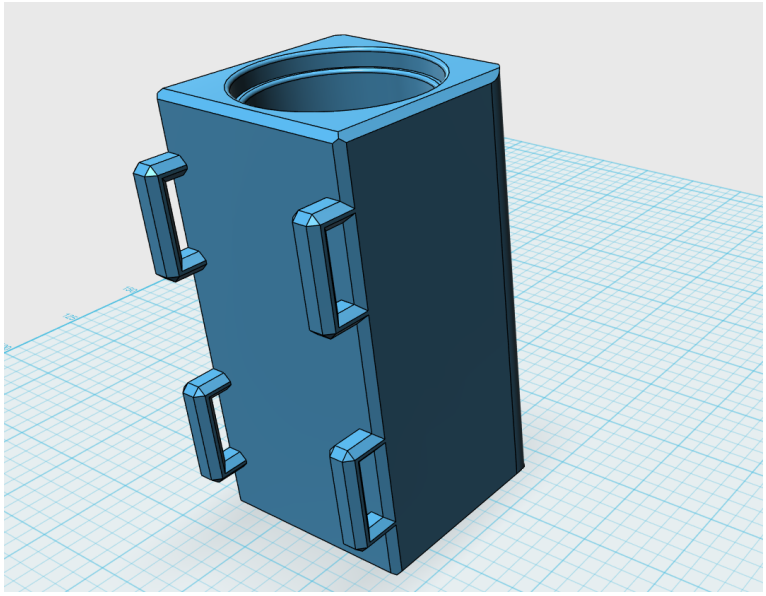


Segnale in uscita dal sensore IR

9. Struttura esterna

Il progetto della struttura esterna è stato realizzato con un programma di disegno 3D adatto alla creazione di file STL, ovvero il formato richiesto per la stampa in 3D, infatti la struttura esterna è stata successivamente realizzata con la stampante 3D.

Progetto in cad dell'ottica



Ottica realizzata

10. Approfondimento timer in ATMega 328

L'ATMega 328, il microcontrollore presente in Arduino nano, mette a disposizione tre timer, di cui due a 8 bit (timer0 e timer2) e uno a 16 bit (timer1).

I registri più importanti di un timer sono:

TCNTn: è il registro di conteggio
OCRn: è il registro di comparazione, quando TCNTn raggiunge il valore impostato in OCRn succede qualcosa in base alla modalità con cui stiamo utilizzando il timer
OCn: è l'uscita del timer
TCCRnA TCCRnB: servono per impostare la modalità di funzionamento del timer

I timer possono essere utilizzati in quattro diverse modalità:

Normal mode: la direzione del conteggio è sempre in avanti, si incrementa TCNTn ogni volta che arriva un impulso e quando si arriva al numero massimo si riparte a contare da 0.

Fast PWM mode: si basa su un meccanismo a singola rampa, il contatore conta dal minimo al massimo e poi riparte dal minimo.
Quando TCNTn e OCRn sono uguali OCn viene resettato e successivamente torna allo stato alto quando TCNTn raggiunge il valore massimo.
Grazie al suo funzionamento a singola rampa questa modalità consente di avere una frequenza doppia in confronto alla modalità Phase correct PWM e ciò la rende adatta per regolazioni di potenza.

Phase correct PWM mode: si basa su un meccanismo a doppia rampa, il contatore continua a contare dal minimo al massimo e dal massimo al minimo, quando TCNTn e OCRn diventano uguali l'uscita OCn cambia stato.
Viene preferito per il controllo dei motori.

Clear time on compare match (CTC) mode: è la modalità utilizzata per ottenere una frequenza.
TCNTn conta partendo dal minimo e quando raggiunge il volare di OCRn TCNTn si azzera e OCn cambia di stato.

Per questo progetto i timer sono stati utilizzati in modalità CTC quindi di seguito è illustrato come si impostano i registri per farli funzionare in tale modalità.

La frequenza del timer si calcola tramite la seguente formula:

$$f_{OCn} = \frac{f_{clk}}{2N(1 + OCRn)}$$

In Arduino f_{clk} è 16MHz
N è il prescaler e può valere (1, 8, 64, 256, 1024)

Tramite la formula inversa si può ricavare il valore di OCRn per ottenere la frequenza che desideriamo.

$$OCRn = \frac{f_{clk}}{2N * f_{OCn}} - 1$$

Questo valore deve essere minore di 256 nel caso in cui il timer sia ad 8 bit o minore di 65536 con il timer a 16 bit, tuttavia si cerca sempre di avere il valore più alto tra quelli consentiti per avere una maggior precisone.

Per abilitare il CTC mode si impostano i bit WGM00 WGM01 WGM02 come da tabella

TCCRnA

Bit	7	6	5	4	3	2	1	0
	COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

TCCRnB

Bit	7	6	5	4	3	2	1	0
	FOC0A	FOC0B			WGM02	CS0[2:0]		
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCR0x at	TOV Flag Set on[1][2]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Per impostare che OCn commuti al compare match si impostano i bit COMA0 COMA1 come da tabella.

Table 19-3. Compare Output Mode, non-PWM

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match.
1	1	Set OC0A on Compare Match .

Per impostare il prescaler ci sono tre bit CAn2, CAn1, CSn0 che vanno impostati come da tabella

CA02	CA01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

11. Listato programma

Inizializzazioni

```
int ledsgn = 10;    //led di segnalazione, si accende quando si muore o quando si e` senza colpi
int ledfiam = 9;   //led di fiammata si accende quando si spara per simulare la luce rossa che esce dalla canna di un fucile quando si spara
int buzsp = 12;     //buzzer di sparo si accende quando si spara
int buzsgn = 11;     //buzzer di segnalazione, si accende quando si e` colpiti o a ricarica avvenuta
int IR555 = 4;      //segnale che va a pilotare il reset di un 555 la cui uscita e` collegata ad un led ir
int pulssp = 3;     //pulsante di sparo
int pulsric = 2;     //pulsante di ricarica
int sensIR = 8;     //sensori infrarosso che ricevono il colpo

//ingressi impostazione arma e codice
int s0 = 0;         //i bit s impostano la squadra
int s1 = 1;
int a0 = 2;         //i bit a impostano l'arma
int a1 = 3;
int a2 = 4;
//variabili impostazione arma e codice
int DATO[5] = {0,0,0,0,0};    //variabile che contiene il codice

int data[5]; //i 5 impulsi che arrivano all'IR vengono inseriti nell'array data[ ] contengono il codice dell'arma avversaria

//variabili che inpostano l'arma
int vita = 0;          //variabile che indica i punti vita, si decrementa ogni volta che si viene colpiti, aumenta se si viene curati
int colpi = 0;         //variabile che indica il numero di colpi per caricatore, decrementa ogni volta che si spara un colpo
int caricatori = 0;    //variabile che indica il numero di caricatori, decrementa ogni volta che si ricarica
int SQUADRA = 0;       //variabile che indica la squadra della propria arma
int ARMA = 0;          //variabile che indica il tipo della propria arma

int squadranem = 0;
int armanem = 0;

int base2 = 1;         //variabili di supporto per la conversione in binario del codice arma squadra
int BASE2 =1;

volatile int iric = 0;          //variabili di supporto usate nella ISR ricarica
volatile int iledsgn = 0;
volatile int flagledsgn = 0;

//variabili per codice in uscita
int BitStart = 1000;    //variabile che indica il valore del bit di start
int bin_1 = 800;       //variabile che indica quando voglio spedire un bit 1
int bin_0 = 600;       //variabile che indica quando voglio spedire un bit 0
int basso = 600;       //variabile che indica il valore di stacco tra un bit e l'altro

int BitStartRic = 950;    //variabile che indica il valore che devo avere per iniziare la ricezione del codice
int bin_1Ric = 750;       //variabile che indica il valore che devo avere per avere un bit a 1
int Statoh = LOW;         //variabile che mi indica se ho un bit a 1 con stato basso o stato alto
int timeoutstart = 5000;  //10000 funziona
int timeoutbit = 4000;    //9000 funziona

int i = 0;
int flagtimer0 = 0;

int k = 0;
int flagbuzin = 0;
int flaggiriin = 0;

int FUNZIONESPARO();
int funzioneaggiornamentovita();
int sparo=0;
```

Void setup()

```
void setup( )
{
  pinMode(ledsgn, OUTPUT);
  pinMode(ledfiam, OUTPUT);
  pinMode(buzsp, OUTPUT);
  pinMode(buzsgn, OUTPUT);
  pinMode(IR555, OUTPUT);
  pinMode(pulssp, INPUT);
  pinMode(pulsric, INPUT);
  pinMode(sensIR, INPUT);

  Serial.begin(9600);

  cli();//stop interrupts

  //set timer2 interrupt at 2kHz                                    //timer usato per la ricarica
  TCCR2A = 0;// set entire TCCR2A register to 0
  TCCR2B = 0;// same for TCCR2B
  TCNT2  = 0;// initialize counter value to 0
  // set compare match register for 2khz increments
  OCR2A = 124;// = (16*10^6) / (2000*64) - 1 (must be <256)
  // turn on CTC mode
  TCCR2A |= (1 << WGM21);
  // Set CS01 and CS00 bits for 64 prescaler
  TCCR2B |= (1 << CS21) | (1 << CS20);

  TIMSK2 &= (0 << OCIE2A);



                                                                  //  timer usato per lo sparo
  TCCR1A = 0;// set entire TCCR1A register to 0
  TCCR1B = 0;// same for TCCR1B
  TCNT1  = 0;//initialize counter value to 0
  // set compare match register for 1hz increments
  //OCR1A viene scelto nel case arma
  // turn on CTC mode
  TCCR1B |= (1 << WGM12);
  // Set CS12 and CS10 bits for 1024 prescaler
  TCCR1B |= (1 << CS12) | (1 << CS10);

  TIMSK1 &= (0 << OCIE1A);


 //set timer0 interrupt at 2kHz
 TCCR0A = 0;// set entire TCCR2A register to 0
 TCCR0B = 0;// same for TCCR2B
 TCNT0  = 0;//initialize counter value to 0
 // set compare match register for 2khz increments
 OCR0A = 124;// = (16*10^6) / (2000*64) - 1 (must be <256)
 // turn on CTC mode
 TCCR0A |= (1 << WGM01);
 // Set CS01 and CS00 bits for 64 prescaler
 TCCR0B |= (1 << CS01) | (1 << CS00);

 TIMSK0 &= (0 << OCIE0A);

 sei();//allow interrupts
```

```
DATO[0]=analogRead(s0);
DATO[1]=analogRead(s1);
DATO[2]=analogRead(a0);
DATO[3]=analogRead(a1);
DATO[4]=analogRead(a2);
for(i=0;i<5;i++)
{
  Serial.print("dato: ");
  Serial.println(DATO[i]);
  if(DATO[i]>= 800)
  {
    DATO[i]=1;
  }
  else
  {
    DATO[i]=0;
  }
  Serial.print("DATO: ");
  Serial.println(DATO[i]);
}

for(int i=0;i<5;i++)
 {
    if(i<2)
    {
      if(DATO[i] == 1)
      {
        SQUADRA=SQUADRA+base2;
      }
      base2 = base2 * 2; // questa istruzione permette il calcolo del peso in base 2 necessario per la conversione
    }
     else
     {
        if(DATO[i] == 1)
       {
         ARMA=ARMA+BASE2;
       }
       BASE2 = BASE2 * 2; // questa istruzione permette il calcolo del peso in base 2 necessario per la conversione
     }
  }
  Serial.print("Squadra: ");
  Serial.println(SQUADRA);
  Serial.print("Arma: ");
  Serial.println(ARMA);

  switch (ARMA)
  {
    case 1:  vita = 100;        //m4
               colpi = 30;
               caricatori = 5;
               OCR1A = 2232;    // = (16*10^6) / (1*1024) - 1 (must be <65536)  //spara a 7 hz       //400 colpi al minuto
               //danno 25
               //tempo di ricarica 4s
               Serial.println("arma 1");
               break;

    case 2:  vita = 100;        //uzzi
               colpi = 15;
               caricatori = 15;
               OCR1A = 1953; //500 colpi al minuto
               //danno 25
               //tempo di ricarica 3s
               Serial.println("arma 2");
               break;

    case 3:  vita = 100;        //cecchino
               colpi = 1;
               caricatori = 15;
               OCR1A = 2232;
               //danno 100
               //tempo di ricarica 7s
               Serial.println("arma 3");
               break;
```

```
    case 4:  vita = 400;        //minigun
             colpi = 200;
             caricatori = 3;
             OCR1A = 1563;     //600 colpi minuto
             //danno 10
             //tempo di ricarica 10s
             Serial.println("arma 4");
             break;

    case 5:  vita = 400;        //bazooka
             colpi = 1;
             caricatori = 7;
             OCR1A = 2232;
             //danno 100
             //tempo di ricarica 10s
             Serial.println("arma 3");
             break;

    case 6:  vita = 200;        //pompa
             colpi = 7;
             caricatori = 5;
             OCR1A = 5200;
             //danno 50
             //tempo di ricarica 5s
             Serial.println("arma 2");
             break;

  /*  case 7:  vita = 100;        //libero
             colpi = 1;
             caricatori = 15;
             OCR1A = 2232;
             //danno 100
             //tempo di ricarica 7s
             Serial.println("arma 3");
             break;*/

   default:  vita = 0;
             colpi = 0;
             caricatori = 0;
             Serial.println("errore arma");
             break;
  }
  digitalWrite(ledsgn, LOW);
  digitalWrite(ledfiam, LOW);
  digitalWrite(buzsp, LOW);
  TIMSK0 |= (1 << OCIE0A);
}
```

void loop()

```
void loop()
{
  if(vita > 0)
  {
     BASE2 = 1;
     base2 = 1;
     squadranem = 0;
     armanem = 0;
     while(pulseIn(sensIR, LOW, timeoutstart) < BitStartRic)
     {
            if(colpi != 0)
            {
               digitalWrite(ledsgn, LOW); //togliere se ci son problemi con il led di segnalazione
             // Serial.println("ho colpi da sparare");
              if(digitalRead(pulssp) == HIGH)
              {
               // Serial.println("pulsante sparo ON");
                TIMSK1 |= (1 << OCIE1A);
                if(sparo == 1)
                {
                  FUNZIONESPARO();
                }
              }
              else
              {
              //  Serial.println("pulsante sparo OFF");
                TIMSK1 &= (0 << OCIE1A);
                digitalWrite(IR555, LOW);
              }
            }
            else
            {
              TIMSK1 &= (0 << OCIE1A);
              //Serial.println("NON ho colpi da sparare");
              if(iric == 0)
              {
                digitalWrite(ledsgn, HIGH);
              }
            }
            if((digitalRead(pulsric) == HIGH)  && caricatori != 0)
            {
              //Serial.println("Ricarica");
              colpi=0;
              TIMSK2 |= (1 << OCIE2A);
            }
```

```
    }
    data[0] = pulseIn(sensIR, Statoh, timeoutbit); // inizia la misura della durata
    data[1] = pulseIn(sensIR, Statoh, timeoutbit); // degli impulsi alti a partire
    data[2] = pulseIn(sensIR, Statoh, timeoutbit); // dal trigger alto-basso
    data[3] = pulseIn(sensIR, Statoh, timeoutbit);
    data[4] = pulseIn(sensIR, Statoh, timeoutbit);
    for(i=0;i<5;i++)
    { // verifica tutti i dati
      if(data[i] > bin_1Ric)
      {
        data[i] = 1 ;
      }
      else
      {
        data[i] = 0;
      }
    }
    for(int i=0;i<5;i++)
    {
       if(i<2)
       {
         if(data[i] == 1)
         {
           squadranem=squadranem+base2;
         }
         base2 = base2 * 2; // questa istruzione permette il calcolo del peso in base 2 necessario per la conversione
       }
       else
       {
          if(data[i] == 1)
         {
           armanem=armanem+BASE2;
         }
         BASE2 = BASE2 * 2; // questa istruzione permette il calcolo del peso in base 2 necessario per la conversione
       }
    }
    funzioneaggiornamentovita();
  }
  else
  {
    colpi = 0;
    caricatori = 0;
    digitalWrite(buzsgn, HIGH);
    digitalWrite(ledsgn, HIGH);
    digitalWrite(ledsgn, HIGH);
  }
}
```

ISR sparo

```
//ISR azione sparo
ISR(TIMER1_COMPA_vect)
{
    //Serial.println("ISR SPARO");
    sparo=1;
}
```

Funzione di sparo

```
int FUNZIONESPARO()
{
    //Serial.println("funzione sparo");
    digitalWrite(ledfiam, HIGH);
    digitalWrite(buzsp, HIGH);
    digitalWrite(IR555, HIGH);
    delayMicroseconds(BitStart);
    digitalWrite(IR555, LOW);
    delayMicroseconds(basso);
    switch (SQUADRA)
    {
      case 0: digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_0);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_0);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                break;

      case 1: digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_1);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_0);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                break;

      case 2: digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_0);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_1);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                break;

      case 3: digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_1);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_1);

                 digitalWrite(IR555, LOW);
                 delayMicroseconds(basso);
                 break;
    }
```

```
switch (ARMA)
{
  case 1:  digitalWrite(IR555, HIGH);  // m4
             delayMicroseconds(bin_1);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_0);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_0);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             break;

  case 2:  digitalWrite(IR555, HIGH);  // uzzi
             delayMicroseconds(bin_0);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_1);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_0);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             break;

  case 3:  digitalWrite(IR555, HIGH);  // cecchino
             delayMicroseconds(bin_1);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_1);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_0);
             digitalWrite(IR555, LOW);

             break;

  case 4:  digitalWrite(IR555, HIGH);  // minigun
             delayMicroseconds(bin_0);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_0);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_1);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             break;

  case 5:  digitalWrite(IR555, HIGH);  // bazooka
             delayMicroseconds(bin_1);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_0);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_1);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             break;

  case 6:  digitalWrite(IR555, HIGH);  // pompa
             delayMicroseconds(bin_0);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_1);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             digitalWrite(IR555, HIGH);
             delayMicroseconds(bin_1);
             digitalWrite(IR555, LOW);
             delayMicroseconds(basso);
             break;
```

```
      case 7:  digitalWrite(IR555, HIGH);  // libero
                delayMicroseconds(bin_1);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_1);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_1);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                break;

    default:  digitalWrite(IR555, HIGH); //colpo di striscio
                delayMicroseconds(bin_0);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_0);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                digitalWrite(IR555, HIGH);
                delayMicroseconds(bin_0);
                digitalWrite(IR555, LOW);
                delayMicroseconds(basso);
                break;
    }
    colpi--;
    digitalWrite(ledfiam, LOW);
    digitalWrite(buzsp, LOW);
    sparo=0;
}
```

Funzione aggiornamento vita

```
int funzioneaggiornamentovita()
{
  if(squadranem != SQUADRA && squadranem != 0)
  {
    switch (armanem)
    {
      case 1:    flagtimer0 = 1;
                 vita=vita-25;             //m4
                 TIMSK0 |= (1 << OCIE0A);
                 break;

      case 2:    flagtimer0 = 1;
                 vita=vita-25;             //uzzi
                 TIMSK0 |= (1 << OCIE0A);
                 break;

      case 3:    flagtimer0 = 1;
                 vita=vita-100;          //cecchino
                 TIMSK0 |= (1 << OCIE0A);
                 break;

      case 4:    flagtimer0 = 1;
                 vita=vita-10;             //minigun
                 TIMSK0 |= (1 << OCIE0A);
                 break;

      case 5:    flagtimer0 = 1;
                 vita=vita-100;             //bazooka
                 TIMSK0 |= (1 << OCIE0A);
                 break;

      case 6:    flagtimer0 = 1;
                 vita=vita-50;          //pompa
                 TIMSK0 |= (1 << OCIE0A);
                 break;

     /* case 7:    flagtimer0 = 1;
                 vita=vita-100;          //cecchino
                 TIMSK0 |= (1 << OCIE0A);
                 break;*/

    default:     flagtimer0 = 2;
                 TIMSK0 |= (1 << OCIE0A);  //colpo di striscio
                 break;
    }
  }
}
```

```
ISR(TIMER0_COMPA_vect)
{
  k++;
  if(flagtimer0 == 0)
  {
    if(k >= 200 && flaggiriin < 4)
    {
      flagbuzin = 1-flagbuzin;
      digitalWrite(buzsgn, flagbuzin);
      k=0;
      flaggiriin++;
    }
    else if(flaggiriin >=4)
    {
      flagtimer0 = 1;
      digitalWrite(buzsgn, LOW);
      k=0;
      TIMSK0 &= (0 << OCIE0A);
    }
  }
  else if (flagtimer0 == 1)
  {
    digitalWrite(buzsgn, HIGH);
    digitalWrite(ledsgn, HIGH);
    if(k >= 50)
    {
      digitalWrite(buzsgn, LOW);
      digitalWrite(ledsgn, LOW);
      k=0;
      TIMSK0 &= (0 << OCIE0A);
    }
  }
  else if (flagtimer0 == 2)
  {
    if(k >= 25 && flaggiriin < 4)
    {
      flagbuzin = 1-flagbuzin;
      digitalWrite(buzsgn, flagbuzin);
      k=0;
      flaggiriin++;
    }

    else if(flaggiriin >= 4)
    {
      digitalWrite(buzsgn, LOW);
      k=0;
      TIMSK0 &= (0 << OCIE0A);
    }
  }
}
```

Funzione di ricarica

```
//ISR per la ricarica
ISR(TIMER2_COMPA_vect)
{
  iric++;
  iledsgn++;
  if(iledsgn >= 1000)
  {
    flagledsgn=1-flagledsgn;
    digitalWrite(ledsgn, flagledsgn);
    digitalWrite(buzsgn, flagledsgn);
    iledsgn=0;
   // Serial.println("ISR RICARICA");
  }
  switch (ARMA)
    {
      case 1:  if(iric >= 8000)             // m4
               {
                  colpi = 30;
                  caricatori--;
                  TIMSK2 &= (0 << OCIE2A);
                  iric=0;
                  digitalWrite(ledsgn, LOW);
                  digitalWrite(buzsgn, LOW);
               }
                break;

      case 2:    if(iric >= 6000)             // uzzi
               {
                  colpi = 15;
                  caricatori--;
                  TIMSK2 &= (0 << OCIE2A);
                  iric=0;
                  digitalWrite(ledsgn, LOW);
                  digitalWrite(buzsgn, LOW);
               }
                break;

      case 3:     if(iric >= 14000)             // cecchino
               {
                  colpi = 1;
                  caricatori--;
                  TIMSK2 &= (0 << OCIE2A);
                  iric=0;
                  digitalWrite(ledsgn, LOW);
                  digitalWrite(buzsgn, LOW);
               }


                break;

     case 4:  if(iric >= 20000)             // minigun
               {
                  colpi = 200;
                  caricatori--;
                  TIMSK2 &= (0 << OCIE2A);
                  iric=0;
                  digitalWrite(ledsgn, LOW);
                  digitalWrite(buzsgn, LOW);
               }
                break;

     case 5:    if(iric >= 20000)             // bazooka
               {
                  colpi = 1;
                  caricatori--;
                  TIMSK2 &= (0 << OCIE2A);
                  iric=0;
                  digitalWrite(ledsgn, LOW);
                  digitalWrite(buzsgn, LOW);
               }
                break;

     case 6:     if(iric >= 10000)             // pompa
               {
                  colpi = 7;
                  caricatori--;
                  TIMSK2 &= (0 << OCIE2A);
                  iric=0;
                  digitalWrite(ledsgn, LOW);
                  digitalWrite(buzsgn, LOW);
               }

                break;
```

```
    default:     if(iric >= 10)
                {
                   colpi = 0;
                   caricatori = 0;
                   TIMSK2 &= (0 << OCIE2A);
                   iric=0;
                   digitalWrite(ledsgn, LOW);
                   digitalWrite(buzsgn, LOW);
                }
                   break;
    }
}
```