

Progetto Allarme Domestico

INTRODUZIONE	1
Testo del progetto	1
Obiettivi	2
ANALISI E SVILUPPO	3
Analisi dei componenti	3
Sensore di movimento	3
IR Receiver e IR Remote	3
LED RGB	6
Buzzer	6
Analisi delle problematiche	6
Gestione dei Timer	6
Decodifica dei segnali IR	7
Gestione funzionamento Buzzer	7
ANALISI DEL CODICE	7
Componenti utilizzati	7
Variabili Globali	8
Configurazioni Timer 1 e Timer 2	8
Configurazioni Interrupt	8
Funzioni Custom	8

Link al progetto: [Allarme Domestico](#)

INTRODUZIONE

Testo del progetto

Realizzare un sistema di allarme da appartamento così composto:

- 1) n. 2 sensori di movimento <https://docs.wokwi.com/parts/wokwi-pir-motion-sensor>
- 2) n. 2 sensori "Finestra" - simulati da due Pulsanti

3) Un ricevitore IR <https://docs.wokwi.com/parts/wokwi-ir-receiver> per attivare e disattivare l'allarme

4) Un led RGB che indica lo stato dell'allarme

5) Un buzzer <https://docs.wokwi.com/parts/wokwi-buzzer>

Il sistema ha le seguenti modalità di lavoro:

a) Se disattivato, il led indica il colore verde fisso;

b) Quando attivato, dal tasto rosso del telecomando:

b1) Il led lampeggia con colore rosso

b2) In caso arrivi un segnale di intrusione da uno dei due sensori, attende 10 secondi e quindi, se non viene disattivato, attiva l'allarme sonoro a doppia tonalità: 1000 e 500 Hz per 5 minuti. Al termine riprende i controlli

b3) Per disattivare l'allarme occorre digitare, entro i 10 secondi un codice di sicurezza da 4 cifre "predefinito"

Il sistema deve essere modulare ovvero prevedere almeno i seguenti moduli di elaborazione in background:

1) Lettura periodica (attivata da INT del clock) dello stato dei sensori

2) Gestione del telecomando tramite interrupt e riconoscimento solo dei tasti "Rosso" e numerici

3) Gestione lampeggio led, attivato anch'esso da interrupt del clock

La funzione loop() (foreground) sarà quindi nulla.

Per tutte le funzioni in background si dovranno prevedere accessi diretti ai registri o ai comandi del display, limitando al minimo il "ts". Per questo non sarà possibile usare la libreria per la gestione del telecomando e per gli ingressi/uscite.

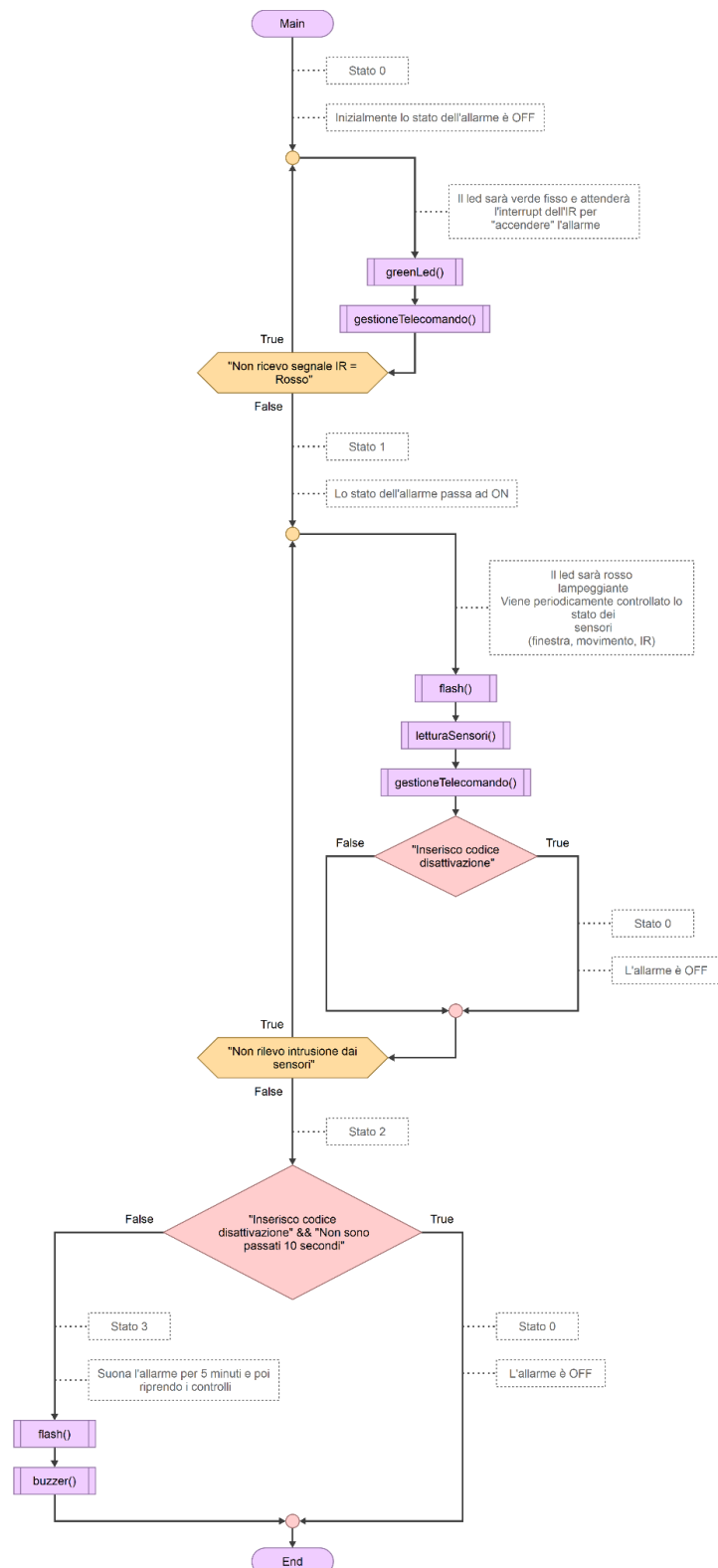
Obiettivi

Per realizzare un sistema di allarme domestico così composto, è necessario suddividere il suo funzionamento in "fasi" o "stati" in cui l'allarme può entrare. Nel caso iniziale (fase 0), l'allarme è disattivato ed è indicato da una luce verde fissa, in attesa di ricevere l'attivazione da parte del telecomando. Una volta attivato attraverso il pulsante "Rosso" del telecomando, l'allarme entrerà nella fase 1 ed avrà luce lampeggiante rossa. In questo caso l'allarme è attivo ed è in attesa della sua disattivazione tramite telecomando, nel frattempo controlla lo stato dei sensori in caso di intrusione. In quest'ultimo caso si entra nella fase 2, in cui l'allarme attende per 10 secondi la sua

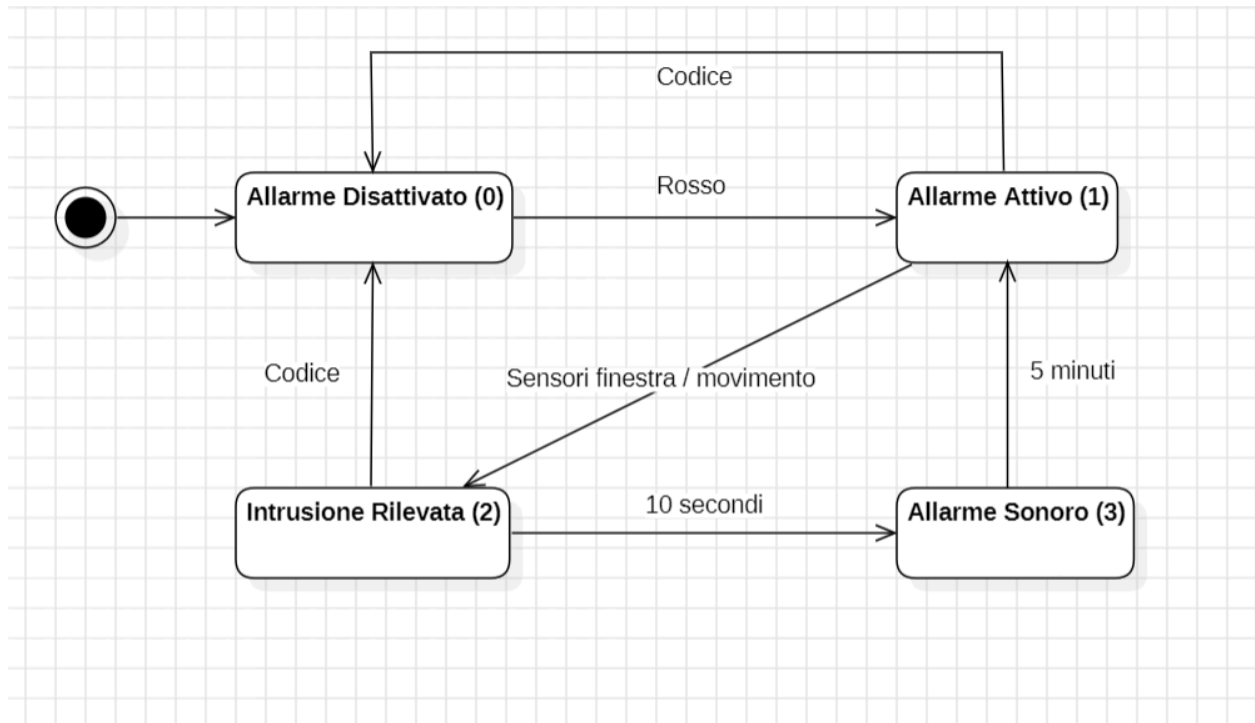
disattivazione tramite telecomando. Nel caso in cui non avvenisse, si entrerebbe nella fase 3 in cui i controlli vengono disattivati e viene azionato il buzzer a doppia tonalità per 5 minuti. Una volta terminato il segnale sonoro, l'allarme riprende i propri controlli, tornando al primo stato.

La progettazione di questo sistema è basata principalmente sulla temporizzazione, per cui ogni componente è gestito da interrupt del clock: dal lampeggio del led fino alla lettura dei sensori "finestra" e di movimento.

Una primissima bozza di flowchart che si avvicina alla progettazione del sistema può essere la seguente:



Una più precisa rappresentazione può essere fatta attraverso uno statechart:

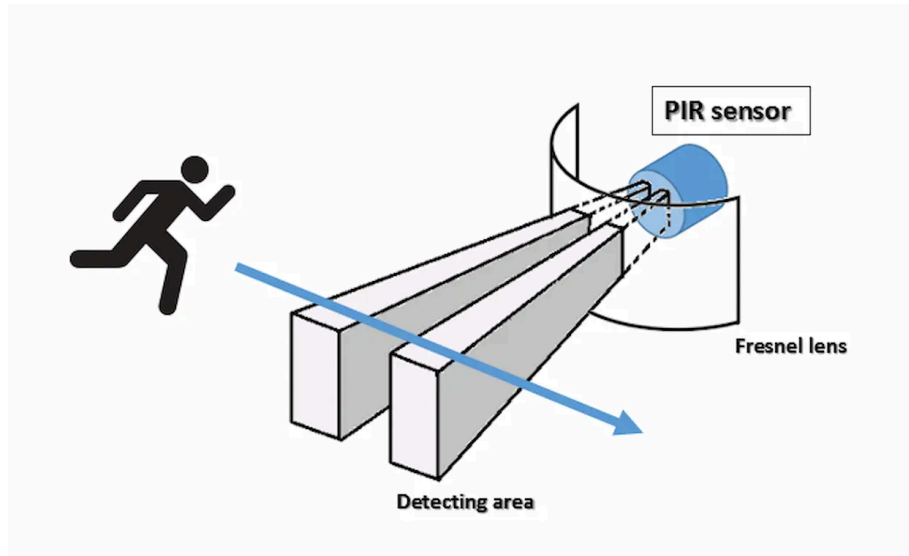


ANALISI E SVILUPPO

Analisi dei componenti

Sensore di movimento

Un sensore di movimento PIR (Passive Infrared) è un dispositivo elettronico progettato per rilevare i cambiamenti di temperatura nell'ambiente circostante e utilizzarli per identificare il movimento. Il cuore del sensore è una lente speciale che divide la zona monitorata in una serie di aree. Ciascuna di essa è collegata ad un sensore PIR, che rileva le variazioni di temperatura nell'area assegnata. Infatti, il sensore indica la presenza di un movimento generando un segnale elettrico, dopo aver rilevato una differenza di temperatura tra l'oggetto in movimento e lo sfondo.



Il sensore di movimento è quindi un dispositivo di input dotato di 3 pin: GND (Ground), VCC (Alimentazione 5V) e OUT (Output Digitale).

Pulsante

Un pulsante (pushbutton) è un dispositivo di input che può essere utilizzato per rilevare lo stato momentaneo di un interruttore. Il pulsante ha due serie di pin (contatti), 1 e 2. Quando il pulsante viene premuto, il circuito si chiude, permettendo il flusso di corrente.

Ciascun contatto ha un pin sul lato sinistro e un altro pin sul lato destro. Dato che entrambi appartengono allo stesso contatto, sono sempre collegati, anche quando il pulsante non è premuto. Si collega quindi un pin di un contatto a un pin digitale, e l'altro contatto a terra.

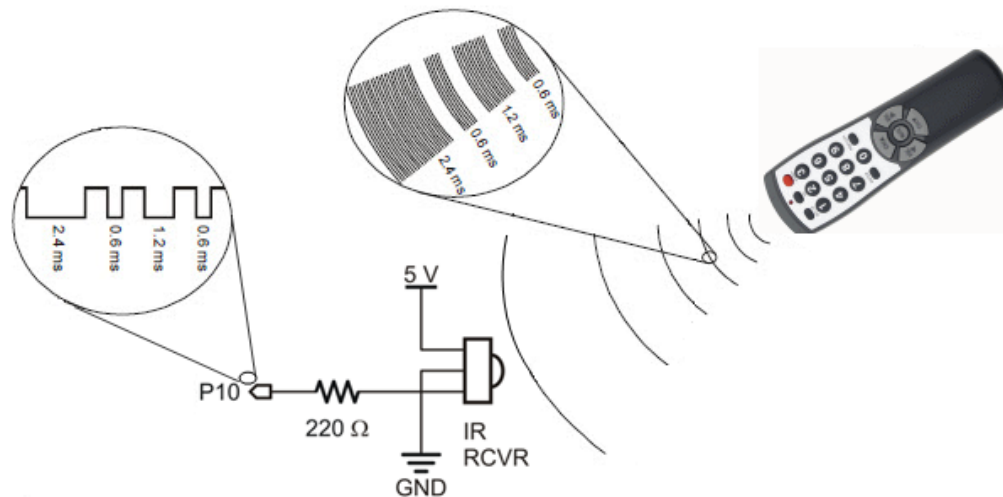
Quando si preme il pulsante fisico, il circuito si apre e si chiude decine o centinaia di volte. Questo fenomeno è chiamato rimbalzo. Ciò accade a causa della natura meccanica dei pulsanti: quando i contatti metallici si uniscono, c'è un breve periodo in cui il contatto non è perfetto, che provoca una serie di rapide transizioni di apertura/chiusura.

IR Receiver e IR Remote

Un IR Receiver (Ricevitore ad Infrarossi) e un IR Remote (Telecomando ad Infrarossi) sono due componenti utilizzati nella comunicazione a infrarossi tra dispositivi elettronici.

L'IR Receiver è progettato per rilevare i segnali trasmessi dal telecomando, convertendoli in impulsi elettrici, che vengono quindi inviati al dispositivo "centrale" per la decodificazione/l'elaborazione.

L'IR Remote è progettato per inviare segnali attraverso un LED a infrarossi che viene attivato dai pulsanti del telecomando stesso. Ogni pulsante è associato ad un codice specifico che rappresenta una determinata funzione. Ovviamente telecomando e ricevitore devono essere compatibili per poter comunicare tra loro. Nel nostro caso entrambi i dispositivi lavorano su una frequenza di 38KHz.



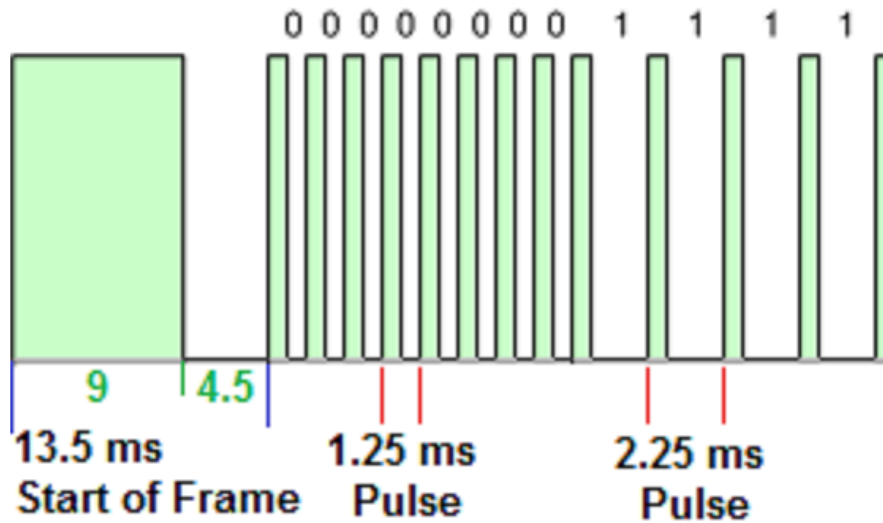
L'IR Receiver è quindi un dispositivo di input dotato di 3 pin: GND (Ground), VCC (Alimentazione 5V) e DAT (Output Digitale). Nel nostro caso, il protocollo impiegato per la codifica dei segnali a infrarossi è il NEC. Questo protocollo codifica i pulsanti usando un frame a 32 bit così composto:

NEC Frame Format			
Address	Complement of Address	Command	Complement of Command
LSB-MSB(0-7)	LSB-MSB(8-15)	LSB-MSB(16-23)	LSB-MSB(24-31)

Ogni bit viene trasmesso secondo precise pulsazioni per cui:

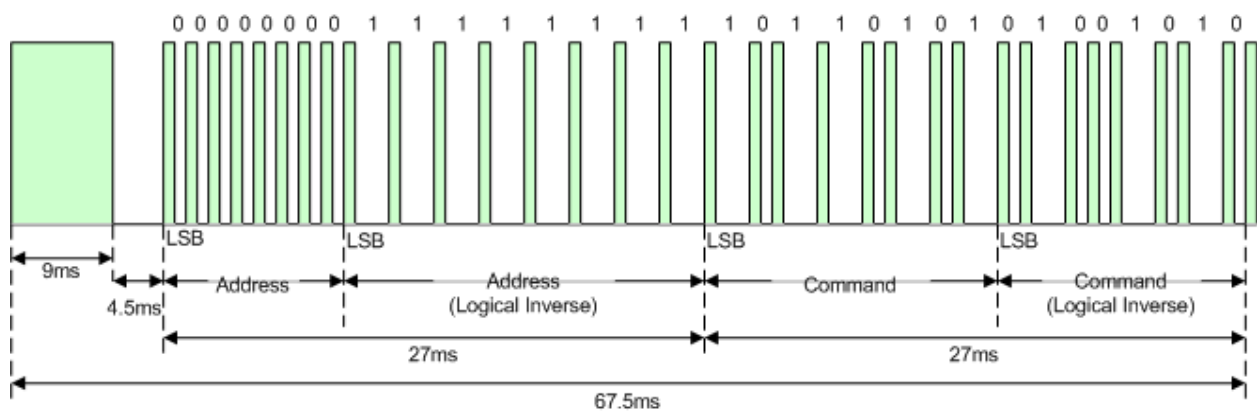
'0' Logico: un burst di impulsi da 562,5µs seguito da uno spazio di 562,5µs per un totale di 1,125ms

'1' Logico: un burst di impulsi da 562,5µs seguito da uno spazio di 1,6875ms per un totale di 2,25ms



Il messaggio trasmesso consiste in queste fasi:

1. Un burst di impulsi in iniziale di 9ms
2. Uno spazio di 4,5ms
3. 8 bit di indirizzo
4. 8 bit del complemento dell'indirizzo
5. 8 bit del comando
6. 8 bit del complemento del comando
7. Un burst finale da 562,5μs per indicare la fine del messaggio trasmesso



Nel nostro caso, sappiamo già che il campo indirizzi equivale sempre a 0, per cui l'inverso logico sarà 256.

LED RGB

Un LED RGB è un tipo di diodo che è in grado di riprodurre più colori attraverso la combinazioni di luci rosse, verdi e blu. Questo LED è costituito a sua volta da 3 LED (Rosso, Verde e Blu) che possono essere controllati indipendentemente.

Il LED RGB è quindi un dispositivo di output composto da 4 pin: R (LED Rosso), G (LED Verde), B (LED Blu) e COM (Common pin).

Buzzer

Un buzzer è un dispositivo elettronico in grado di produrre suoni o segnali acustici. Funziona mediante la vibrazione di una membrana o di un elemento piezoelettrico a cui viene applicata una tensione. La frequenza e l'intensità del suono prodotto dipendono da vari fattori, come la frequenza della tensione applicata e il design del buzzer stesso.

Il buzzer quindi è un dispositivo di output composto da 2 pin: 1 (Negativo) e 2 (Positivo).

Analisi delle problematiche

Per poter realizzare il sistema quindi, il progetto è stato suddiviso in tanti moduli per farli funzionare indipendentemente prima di farli lavorare insieme.

Gestione dei Timer

In Arduino, i timer interni sono componenti hardware che possono essere programmati per contare il tempo, generare impulsi di clock o generare segnali di interruzione a intervalli regolari.

Il chip ATmega328p su cui è basato Arduino Uno, dispone di tre timer hardware:

- Timer0 a 8 bit, usato per gestire il tempo (come funzioni *delay* e *millis*)
- Timer1 a 16 bit, usato dalla libreria servo per i motori
- Timer2 a 8 bit, usato dalla funzione *tone*

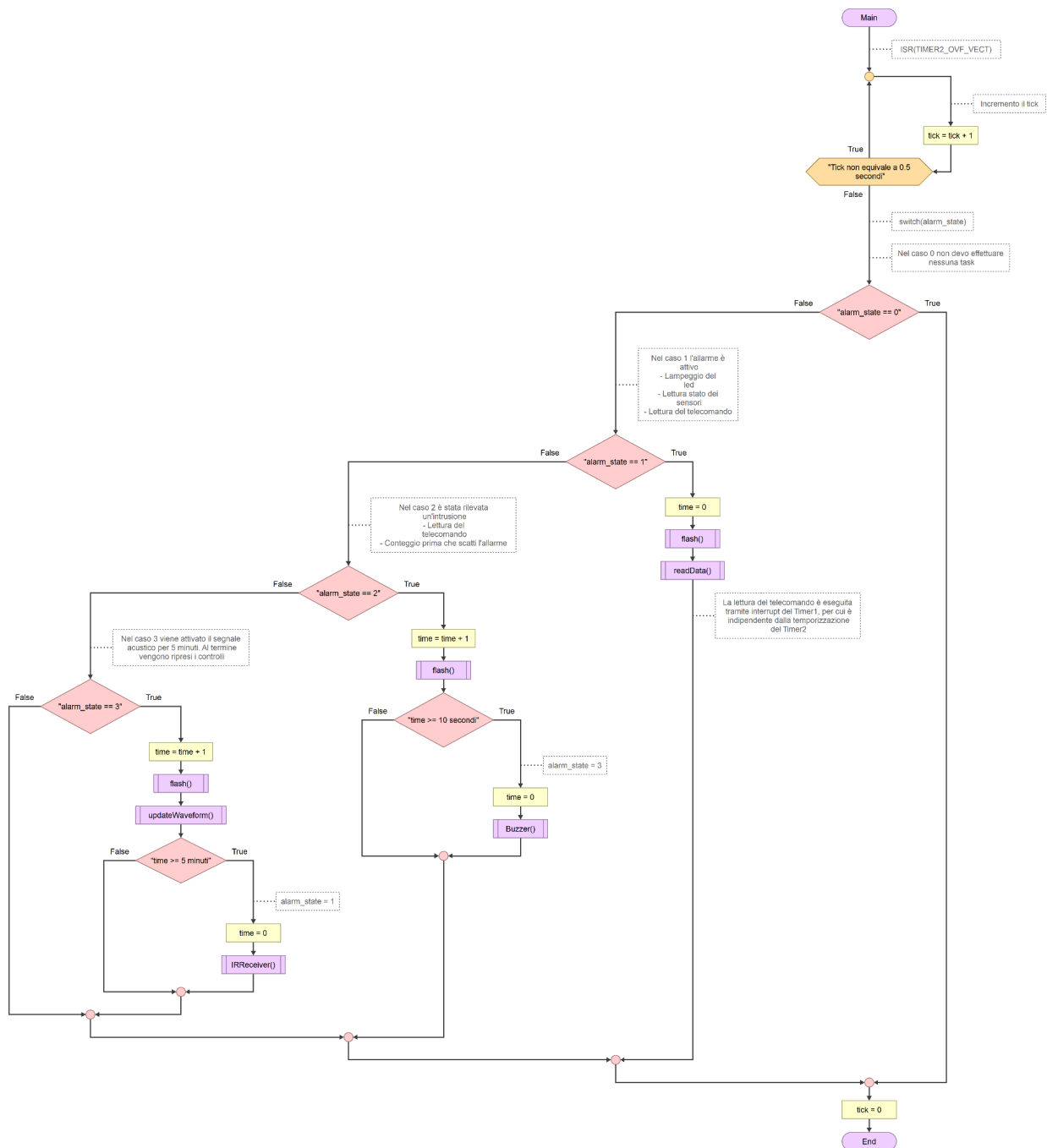
Il chip lavora con una frequenza di 16MHz, ovvero è in grado di eseguire 16 milioni di 'istruzioni' al secondo. Per rendere più comodo l'uso dei timer, si può ricorrere ad un divisore del clock detto *prescaler*. Quest'ultimo consente di regolare la frequenza con cui il timer conta o genera impulsi e può essere impostato per alcuni valori: 1, 8, 64, 256 o 1024.

I registri dei timer di Arduino sono registri speciali nei quali è possibile scrivere o leggere per configurare e controllare il funzionamento. I principali registri associati ai timer sono:

-
- Registri di controllo (TCCRnA e TCCRnB): sono utilizzati per impostare le modalità di funzionamento e le opzioni di controllo, come la modalità di conteggio, la modalità di uscita, il prescaler e altre opzioni.
 - Registri di conteggio (TCNTn): contengono il valore corrente del contatore del timer. Leggendo o scrivendo in questo registro, è possibile accedere al valore corrente del contatore.
 - Registri di confronto (OCRnA e OCRnB): sono utilizzati per impostare valori di confronto. Il timer confronta il valore corrente del contatore con questi registri e può generare un'azione quando il contatore raggiunge il valore specificato.
 - Registri di maschera di interruzione (TIMSKn): sono utilizzati per abilitare o disabilitare le interruzioni associate al timer n, come l'interruzione di overflow o di confronto.

I timer impiegati all'interno del progetto sono:

- Timer2, impiegato per l'esecuzione delle task attraverso un interrupt generato ogni millisecondo.
 - Ad ogni interrupt viene aumentato il tick. Quando il tick raggiunge mezzo secondo, verranno eseguite le determinate task, a seconda dello stato dell'allarme.



- Timer1, impiegato per la decodifica dei segnali ad infrarossi e per la gestione dell'allarme sonoro tramite il buzzer
 - Inizialmente viene impostato per la gestione dei segnali IR. Quando viene raggiunto lo stato 3 dell'allarme, il Timer1 viene configurato per la gestione del buzzer. Una volta terminato l'allarme sonoro, vengono ripresi i controlli.

Decodifica dei segnali IR

La gestione e la decodifica dei segnali IR NEC nel nostro sistema sono implementate attraverso l'uso combinato dell'interrupt del Timer 1 e l'interrupt esterno dell'IR Receiver.

In questo caso sono stati utilizzati external interrupt sui pin 2 e 3 (INT0 e INT1).

External Interrupt (Interrupt Esterno):

- Si riferisce a un interrupt generato da una sorgente esterna al microcontrollore Arduino, come un sensore di movimento o un segnale proveniente da un altro dispositivo.
- L'interrupt esterno è associato a specifici pin hardware del microcontrollore, come ad esempio i pin 2 e 3 su Arduino Uno. Quando si verifica un cambiamento di stato su uno di questi pin, come un fronte di salita o discesa, viene generato un interrupt che interrompe immediatamente l'esecuzione del programma principale per gestire l'evento.

Digital Interrupt (Interrupt Digitale):

- Si riferisce a un interrupt generato da un cambiamento di stato su un pin digitale specifico di Arduino.
- Questo tipo di interrupt può essere configurato per monitorare l'input di un pin digitale e attivare un'azione quando viene rilevato un cambiamento di stato, ad esempio da HIGH a LOW o viceversa.

L'IR Receiver è quindi collegato a un pin di interrupt esterno del microcontrollore e configurato per generare un interrupt ogni volta che viene rilevato un segnale IR.

La funzione interrupt esterna che viene chiamata ogni volta che viene rilevato un segnale IR dal modulo IR Receiver esegue le seguenti istruzioni:

1. Salva il valore corrente del Timer 1 (contatore) per misurare la durata degli impulsi e degli spazi del segnale IR.
2. In base allo stato corrente del segnale IR, ne gestisce la decodifica, comprese la rilevazione degli impulsi e degli spazi e la determinazione del valore binario dei bit trasmessi.
3. Gestisce il cambiamento di stato del segnale IR e le transizioni tra impulsi e spazi.
4. Aggiorna il valore corrente del contatore del Timer 1 e imposta nuovamente il prescaler per misurare correttamente la durata degli impulsi e degli spazi.

[Flowchart INTO](#)

Il contatore del Timer 1 incrementa di due ogni $1\ \mu\text{s}$ ($\frac{1}{8}$ prescaler). Sono stati usati quindi degli intervalli per scandire le fasi della decodifica:

-
- [9500µs - 8500µs] per l'impulso da 9ms
 - [5000µs - 4000µs] per lo spazio da 4,5ms
 - [700µs - 400µs] per l'impulso da 562,5µs
 - [1800µs - 400µs] per lo spazio da 1687,5µs o 562,5µs
 - 1000µs per identificare uno spazio corto (0 logico) da uno spazio lungo (1 logico)

I ticks necessari sono quindi:

- [9500µs - 8500µs] = [19000 - 17000] ticks
- [5000µs - 4000µs] = [10000 - 8000] ticks
- [700µs - 400µs] = [1400 - 800] ticks
- [1800µs - 400µs] = [3600 - 800] ticks
- 1000µs = 2000 ticks

Il Timer 1 è configurato per generare un interrupt di overflow regolare che viene utilizzato per sincronizzare e controllare il processo di decodifica dei segnali IR NEC.

1. Interrompe il processo di decodifica del segnale IR impostando lo stato corrente su 0 e disabilitando il Timer 1.
2. Se il segnale IR è stato decodificato correttamente, gestisce il comando ricevuto eseguendo azioni specifiche in base al comando.
3. Effettua il reset dell'external interrupt per il rilevamento del segnale IR, consentendo al sistema di rimanere in attesa di nuovi segnali IR.

[Flowchart Timer 1](#)

Gestione funzionamento Buzzer

Il buzzer nel progetto è gestito utilizzando il Timer 1 dell'Arduino Uno. Sono state implementate una generazione di onde quadre con cambio di frequenza ad intervalli regolari per produrre suoni a diverse tonalità.

Sono state create due funzioni principali per gestire il cambio di frequenza del buzzer:

1. **'updatePWMFrequency(unsigned long frequency)'**: questa funzione calcola il periodo necessario per una data frequenza e imposta il periodo nel registro ICR1 (Input Capture Register 1) del Timer 1. Il periodo viene calcolato utilizzando la formula $period = F_{CPU} / (64UL * frequency) - 1$, dove F_{CPU} è la frequenza del clock del microcontrollore e 64UL è il prescaler utilizzato per il Timer 1. Questa funzione viene chiamata ogni volta che desideriamo cambiare la frequenza del buzzer.

-
2. **'updateWaveform()'**: questa funzione gestisce il cambio di frequenza del buzzer. Se un certo flag è impostato, la funzione imposta la frequenza del buzzer a 1000 Hz utilizzando `updatePWMFrequency(1000)`. In caso contrario, la frequenza del buzzer viene impostata a 500 Hz utilizzando `updatePWMFrequency(500)`. Questo flag viene modificato ad intervalli regolari tramite l'uso di un interrupt del Timer 2.

ANALISI DEL CODICE

Componenti utilizzati

- IR Receiver collegato al PIN 2
- Pulsanti collegati al PIN 3
- LED RGB: PIN Verde 8, PIN Rosso 9
- Motion Sensor 1 collegato al PIN 12
- Motion Sensor 2 collegato al PIN 11
- Buzzer collegato al PIN 10

Variabili Globali

- **'count'**: contatore dei tick del Timer 2
- **'alarm_state'**: stato dell'allarme, può assumere i valori 0 (disattivato), 1 (attivato), 2 (intrusione rilevata), 3 (allarme sonoro attivo)
- **'time'**: variabile di temporizzazione delle task
- **'i', 'nec_state', 'command', 'nec_code'**: variabili utilizzate per decodifica del segnale infrarosso NEC
- **'code'**: codice di sicurezza predefinito
- **'ins_code'**: codice inserito dall'utente per disattivare l'allarme

```

1  /* VARIABILI GLOBALI */
2  volatile uint16_t tick; // Tick del Timer2
3  volatile uint8_t alarm_state = 0; // Stato dell'allarme [0-3]
4  volatile uint8_t time = 0; // Temporizzazione delle task
5
6  // Decodifica segnale NEC
7  volatile byte i, nec_state = 0, command;
8  volatile unsigned long nec_code;
9
10 // Codice Allarme
11 const String code = "1234";
12 volatile String ins_code = "";
13
14 // Macro per gestire il registro General Purpose
15 // Al bit 0 viene gestito il cambio frequenza del Buzzer
16 // Al bit 1 viene gestito il processo di decodifica del segnale NEC
17 // Al bit 2 viene gestita la lettura dei Pulsanti
18 #define SMY_FLAG(n, mode) (mode == 1 ? (GPIO0 |= _BV(n)) : (GPIO0 &= ~_BV(n)))

```

Configurazioni Timer 1 e Timer 2

Questa serie di funzioni gestisce le configurazioni dei Timer 1 e Timer 2 su Arduino.

- **Funzione TIMER1_reset:** questa funzione è responsabile del reset completo delle impostazioni del Timer 1. Viene impostato il registro di controllo A (TCCR1A) a 0, il registro di controllo B (TCCR1B) a 0 e il registro del conteggio del timer (TCNT1) a 0 per assicurare che il timer sia completamente azzerato. Inoltre, viene disabilitato l'interrupt del timer (TIMSK1) e l'interrupt esterno (INT0) impostando i registri a 0. Infine, viene fatto il reset dei flag relativi alla frequenza del buzzer e alla lettura dei pulsanti.
- **Funzione TIMER1_IR_setup:** questa funzione è specificamente progettata per configurare il Timer 1 per il riconoscimento dei segnali IR. Come la funzione precedente, essa effettua il reset dei registri TCCR1A, TCCR1B e TCNT1. Successivamente, abilita l'interrupt di overflow del timer 1 (TOIE1) impostando il registro TIMSK1 e riabilita l'interrupt esterno (EIMSK) per INT0.
- **Funzione TIMER1_buzzer_setup:** questa funzione configura il Timer 1 per generare un segnale PWM rapido, adatto per il funzionamento di un buzzer. Viene abilitata la modalità Fast PWM impostando i bit appropriati nei registri TCCR1A e TCCR1B. Inoltre, viene impostato il prescaler a 64. Infine, viene chiamata la funzione `updatePWMFrequency(1000)` per configurare la frequenza del buzzer a 1000 Hz.

```

1  /* CONFIGURAZIONI TIMER 1 */
2  void TIMER1_reset(){
3      TCCR1A = 0; // Reset del registro
4      TCCR1B = 0; // Reset del registro
5      TCNT1 = 0; // Reset del registro
6      TIMSK1 = 0; // Reset del registro
7      SMY_FLAG(0,0); // Reset del bit per la frequenza del buzzer
8      SMY_FLAG(2,0); // Reset del bit per la lettura dei pulsanti
9      EIMSK &= ~(1 << INT0); // Disattivo External Interrupt IR Receiver (INT0)
10 }
11
12 void TIMER1_IR_setup(){
13     TCCR1A = 0; // Reset del registro
14     TCCR1B = 0; // Reset del registro
15     TCNT1 = 0; // Reset del registro
16     TIMSK1 = 1 << TOIE1; // Abilito interrupt TIMER1_OVF
17     EIMSK |= 1 << INT0; // Riattivo External Interrupt IR Receiver (INT0)
18 }
19
20 void TIMER1_buzzer_setup(){
21     // Configura il Timer 1 in modalità Fast PWM, prescaler a 64
22     TCCR1A = (1 << WGM11) | (1 << COM1B1);
23     TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11) | (1 << CS10);
24     // Chiamo la funzione che mi configura la frequenza
25     updatePWMFrequency(1000);
26 }

```

- **Funzione TIMER2_setup:** questa funzione è responsabile della configurazione del Timer 2. Viene fatto il reset dei registri TCCR2A e TCCR2B, quindi il prescaler viene impostato a 64 attraverso il registro TCCR2B. Infine, viene abilitato l'interrupt di overflow del Timer 2 impostando il bit appropriato nel registro TIMSK2.

```

1  /* CONFIGURAZIONE TIMER 2 */
2  void TIMER2_setup(){
3      TCCR2A = 0; // Reset del registro
4      TCCR2B = 0; // Reset del registro
5      TCCR2B = 1 << CS22; // Prescaler a 64
6      TIMSK2 = 1 << TOIE2; // Abilito interrupt TIMER2_OVF
7  }

```

Setup

Questa funzione setup() è una parte essenziale di un programma Arduino, poiché viene eseguita una sola volta all'avvio del dispositivo.

-
- **Disabilitazione degli Interrupt:** viene chiamata la funzione `cli()` per disabilitare tutti gli interrupt. Questo è un passo comune all'inizio di `setup()` per evitare che gli interrupt interferiscano con la configurazione iniziale.
 - **Configurazione del LED a "Verde":** viene chiamata la funzione `greenLED()` per configurare il LED in modo che sia acceso di colore verde.
 - **Configurazione degli Interrupt INTO e INT1:** viene configurato il registro `EICRA` per abilitare gli interrupt `INT0` e `INT1` su cambi di stato logici (fronte di salita). Successivamente, gli interrupt `INT0` e `INT1` vengono attivati abilitando i bit corrispondenti nel registro `EIMSK`.
 - **Configurazione del Registro `DDRB`:** viene configurato il registro `DDRB` per impostare i pin 0, 1 e 2 come output, che controllano il LED RGB e il buzzer. Inoltre, vengono impostati i pin 11 e 12 come input, che sono utilizzati per il sensore di movimento.
 - **Abilitazione degli Interrupt:** viene chiamata la funzione `sei()` per abilitare nuovamente tutti gli interrupt. Questo avviene dopo che tutte le configurazioni sono state completate e il sistema è pronto per interagire con l'ambiente esterno.


```

1 void setup(){
2
3   // Disabilito interrupt
4   cli();
5
6   // All'avvio configuro il Timer1 per la decodifica dei segnali IR
7   TIMER1_IR_setup();
8   // e configuro il Timer2 per eseguire le task
9   TIMER2_setup();
10
11  // Configuro il LED a "Verde"
12  greenLED();
13
14  // Interrupt INT0 e INT1
15  EICRA |= 1 << ISC00 | 1 << ISC10;
16  EIMSK |= 1 << INT0 | 1 << INT1;
17
18  // Configuro il registro DDRB
19  DDRB = 1 << PINB0 | 1 << PINB1 | 1 << PINB2;
20  // PIN8 - PIN9 - PIN10 ad output: RGB LED e Buzzer
21  // PIN11 e PIN12 ad input: Motion Sensor
22
23  // DDRD = 0 - Pushbuttons e IR Receiver ad input
24
25  // Abilito gli interrupt
26  sei();
27
28 }

```

Configurazioni Interrupt

Questa funzione è un'ISR (Interrupt Service Routine) che gestisce l'overflow del Timer 2. Il timer è configurato in modo che l'overflow scatti ogni 1.024 millisecondi. La funzione viene eseguita ad ogni overflow del Timer 2 e svolge diverse operazioni in base allo stato dell'allarme.

- **Incremento del Tick:** ad ogni chiamata della funzione ISR, viene incrementato il contatore tick. Questo contatore tiene traccia del tempo trascorso in millisecondi.
- **Esecuzione delle Task Ogni Mezzo Secondo:** se il contatore raggiunge o supera il valore di 488 (che corrisponde a mezzo secondo), vengono eseguite delle task specifiche in base allo stato dell'allarme.
- **Controllo dello Stato dell'Allarme:** viene eseguito un controllo sullo stato dell'allarme tramite uno switch-case. Le possibili condizioni sono:

-
- **Allarme Disattivato** (case 0): in questa condizione, non vengono eseguite task.
 - **Allarme Attivato** (case 1):
 - Reset del tempo (time)
 - Lampeggio del LED
 - Lettura dei sensori
 - **Attesa Spegnimento o Rilevazione Intrusione** (case 2):
 - Incremento del tempo
 - Lampeggio del LED
 - Se il tempo trascorso è maggiore o uguale a 10 secondi, lo stato dell'allarme passa a 3 (Allarme Sonoro)
 - **Allarme Sonoro** (case 3):
 - Incremento del tempo
 - Lampeggio del LED
 - Cambio della frequenza del Buzzer
 - Se il tempo trascorso è maggiore o uguale a 5 minuti (300 secondi), lo stato dell'allarme passa nuovamente a 1 (Allarme Attivato)
 - **Azzero il Tick**: alla fine delle operazioni, il contatore tick viene azzerato per prepararsi al prossimo ciclo di 500 millisecondi.



sketch.ino

```
1 // Timer 2 - Overflow scatta ogni 1.024ms
2 ISR(TIMER2_OVF_vect){
3
4     // Ogni ms aggiorno il tick
5     tick++;
6
7     // Ogni mezzo secondo eseguo le task
8     if(tick ≥ 488){ // 976 = 1sec, 488 = 0.5sec
9
10        // Controllo lo stato dell'allarme
11        switch(alarm_state){
12
13            // Allarme disattivato
14            case 0:
15                break;
16
17            // Allarme attivato
18            case 1:
19                time = 0; // Reset del tempo
20                flash(); // Lampeggio del LED
21                read_data(); // Lettura dei sensori
22                break;
23
24            // Attesa spegnimento o attivazione allarme sonoro
25            case 2:
26                time++; // Incremento il tempo
27                flash(); // Lampeggio del LED
28
29                // Dopo 10 secondi passo allo stato successivo
30                if((time/2) ≥ 10){
31                    alarm_state = 3;
32                    TIMER1_reset(); // Reset del Timer1
33                    TIMER1_buzzer_setup(); // Configuro il Timer1 per il Buzzer
34                    time = 0; // Reset del tempo
35                }
36                break;
37
38            // Allarme sonoro
39            case 3:
40                time++; // Incremento il tempo
41                flash(); // Lampeggio del LED
42                updateWaveform(); // Cambio frequenza del Buzzer
43
44                // Dopo 5 minuti di allarme, riprende i controlli
45                if((time/2) ≥ 300){
46                    alarm_state = 1;
47                    TIMER1_reset(); // Reset del Timer1
48                    TIMER1_IR_setup(); // Configuro il Timer1 per i segnali IR
49                    time = 0; // Reset del tempo
50                }
51                break;
52
53            default:
54                break;
55        }
56
57        // Azzerò il tick
58        tick = 0;
59    }
60 }
61 }
```

Questa funzione gestisce l'interrupt dell'IR Receiver (INT0).

- **Salvataggio del Timer Value:** viene salvato il valore corrente del Timer 1 (TCNT1) in una variabile locale chiamata `timer_value`. Questo valore viene utilizzato per misurare la durata degli impulsi e degli spazi del segnale IR.
- **Switch Case per lo Stato NEC:** la funzione utilizza una variabile di stato chiamata `nec_state` per tenere traccia della fase attuale della decodifica del segnale IR. Viene eseguito un switch case basato sullo stato corrente per determinare l'azione appropriata in base alla fase della decodifica.
 - **Stato 0 (Inizio Ricezione Segnale):** se lo stato è 0, il segnale IR appena ricevuto viene riconosciuto come l'inizio della ricezione del segnale, quindi viene fatto il reset del Timer 1 e viene avviato per misurare la durata dell'impulso iniziale (9 ms).
 - **Stato 1 (Inizio Spazio da 4.5 ms):** se lo stato è 1, viene controllato se la durata del timer (misurata nell'istruzione precedente) è valida per identificare l'inizio dello spazio di 4.5 ms. Se la durata è valida, lo stato viene aggiornato per indicare l'inizio dell'impulso successivo (562 μ s).
 - **Stato 2 (Inizio Impulso da 562 μ s):** in questo stato, viene verificato se la durata del timer è valida per identificare l'inizio dell'impulso da 562 μ s. Se la durata è valida, lo stato viene aggiornato per indicare l'inizio dello spazio successivo (562 μ s o 1687 μ s).
 - **Stato 3 (Inizio Spazio da 562 μ s o 1687 μ s):** in questo stato, viene verificato se la durata del timer è valida per identificare l'inizio dello spazio da 562 μ s o 1687 μ s. Se la durata è valida, lo stato viene aggiornato di conseguenza.
 - **Stato 4 (Decodifica Bit):** in questo stato, viene determinato se lo spazio è lungo o corto. Se lo spazio è lungo (> 1 ms), viene impostato un bit a 1 nel codice ricevuto `nec_code`; altrimenti, viene impostato a 0. Viene quindi incrementato l'indice `i` per passare al bit successivo. Se tutti i bit sono stati ricevuti, il processo di decodifica viene considerato completato, e il flag di decodifica `SMY_FLAG` viene impostato. Infine, l'interrupt esterno (INT0) viene disabilitato per evitare ulteriori interruzioni durante il processo di decodifica.

```

1 // External Interrupt IR Receiver
2 ISR(INT0_vect){
3     // Valore del timer
4     unsigned int timer_value;
5
6     if(nec_state != 0){
7         timer_value = TCNT1; // Salvo il contenuto del registro
8         TCNT1 = 0;           // Reset Timer1
9     }
10
11     switch(nec_state){
12         // Inizio a ricevere il segnale (inizio dell'impulso da 9ms)
13         case 0:
14             TCNT1 = 0;           // Reset del contatore
15             TCCR1B = 1 << CS11; // Abilito il prescaler a 8 (2 tick ogni 1 us)
16             nec_state = 1;       // Prossimo stato: fine dell'impulso da 9ms (inizio dello spazio da 4.5ms)
17             i = 0;
18             return;
19
20         // Inizio dello spazio da 4.5ms
21         case 1:
22             // Intervallo non valido ==> stop decodifica e reset
23             if((timer_value > 19000) || (timer_value < 17000)){
24                 TCCR1B = 0;
25                 nec_state = 0;
26             }
27             else
28                 nec_state = 2; // Prossimo stato: fine dello spazio di 4.5ms (inizio impulso da 562µs)
29             return;
30
31         // Inizio impulso da 562µs
32         case 2:
33             if((timer_value > 10000) || (timer_value < 8000)){
34                 TCCR1B = 0;
35                 nec_state = 0;
36             }
37             else
38                 nec_state = 3; // Prossimo stato: fine dell'impulso da 562µs (inizio dello spazio da 562µs o 1687µs)
39             return;
40
41         // Inizio dello spazio da 562µs o 1687µs
42         case 3:
43             if((timer_value > 1400) || (timer_value < 800)){
44                 TCCR1B = 0;
45                 nec_state = 0;
46             }
47             else
48                 nec_state = 4; // Prossimo stato: fine dello spazio da 562µs o 1687µs
49             return;
50
51         case 4:
52             if((timer_value > 3600) || (timer_value < 800)){
53                 TCCR1B = 0;
54                 nec_state = 0;
55                 return;
56             }
57
58             // Se l'ampiezza dello spazio è > 1ms (spazio lungo)
59             if(timer_value > 2000)
60                 nec_code |= 1UL << (31 - i); // Scrivo 1 al bit (31 - i)
61             else // Se l'ampiezza dello spazio è < 1ms (spazio corto)
62                 nec_code &= ~(1UL << (31 - i)); // Scrivo 0 al bit (31 - i)
63
64             i++; // Incremento i
65
66             // Se tutti i bit sono stati ricevuti
67             if(i > 31){
68                 SMY_FLAG(1,1); // Processo di decodifica OK
69                 EIMSK &= ~(1 << INT0); // Disabilito external interrupt (INT0)
70                 return;
71             }
72             nec_state = 3; // Prossimo stato: fine dell'impulso da 562µs (inizio dello spazio da 562µs o 1687µs)
73     }
74 }

```

Questa funzione gestisce l'overflow del Timer 1.

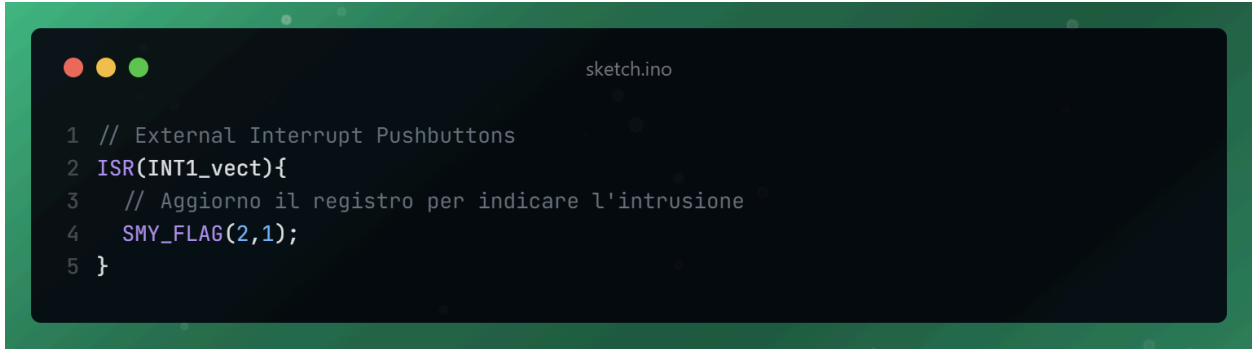
- **Reset del Processo di Decodifica:** viene fatto il reset dello stato corrente del processo di decodifica (`nec_state`) impostandolo a 0.
- **Disabilitazione del Timer 1:** viene disabilitato il modulo del Timer 1 (`TCCR1B = 0`) per interrompere la misurazione del tempo e la decodifica dei segnali IR.
- **Controllo del Messaggio Ricevuto:** viene verificato se il messaggio IR è stato ricevuto con successo attraverso il flag `GPOR0`.
- **Reset del Processo di Decodifica:** se il messaggio IR è stato ricevuto correttamente, viene fatto il reset del processo di decodifica impostando il flag di decodifica (`SMY_FLAG(1,0)`).
- **Salvataggio del Codice del Comando:** viene salvato il codice del comando IR ricevuto `bitSwap(nec_code >> 8)` nella variabile `command`.
- **Controllo dello Stato dell'Allarme:** viene effettuato un controllo sullo stato dell'allarme. Se lo stato è 0 e il comando ricevuto corrisponde al codice per il tasto "Rosso", viene disattivato il LED verde e lo stato dell'allarme viene impostato a 1.
- **Riconoscimento per Sequenza:** se lo stato dell'allarme non è 0 e non è 3 (indicando che l'allarme non è disattivato e non è in uno stato particolare), viene eseguito il riconoscimento per sequenza dei comandi IR ricevuti. In base al comando ricevuto, viene aggiornato il codice inserito (`ins_code`) in modo appropriato.
- **Spegnimento dell'Allarme:** se il codice inserito corrisponde al codice corretto (`code`), l'allarme viene disattivato impostandolo a 0. Si azzerà il codice inserito. Viene inoltre attivato il LED verde.
- **Riattivazione dell'Interrupt Esterno:** viene riattivato l'interrupt esterno su `INT0` per consentire la ricezione di nuovi segnali IR.

```

1 // Interrupt Timer 1 per riconoscimento segnali IR
2 ISR(TIMER1_OVF_vect) {
3
4     nec_state = 0; // Reset del processo di decodifica
5     TCCR1B = 0; // Disabilito il modulo
6
7     // Se il messaggio è stato ricevuto con successo
8     if(GPIOR0 & _BV(1)){
9         SMY_FLAG(1,0); // Reset del processo di decodifica
10        command = bitSwap(nec_code >> 8); // Salvo il codice del comando
11
12        // Controllo stato dell'allarme
13        if(alarm_state == 0){
14            if(command == 162){ //Premuto "Rosso"
15                PORTB &= ~(1 << PORTB0); // Disattivo LED verde
16                alarm_state = 1;
17            }
18        }else if(alarm_state == 0 && alarm_state == 3){
19            //Riconoscimento per sequenza...
20            switch(command){
21                case 104: ins_code += "0"; break;
22                case 48: ins_code += "1"; break;
23                case 24: ins_code += "2"; break;
24                case 122: ins_code += "3"; break;
25                case 16: ins_code += "4"; break;
26                case 56: ins_code += "5"; break;
27                case 90: ins_code += "6"; break;
28                case 66: ins_code += "7"; break;
29                case 74: ins_code += "8"; break;
30                case 82: ins_code += "9"; break;
31                case 176: ins_code = ""; break;
32                default: break;
33            }
34
35            // Spegnimento allarme
36            if(ins_code == code){
37                alarm_state = 0;
38                ins_code = "";
39                greenLED();
40            } // Reset codice inserito
41            else if(ins_code.length() == 4){
42                ins_code = "";
43            }
44        }
45
46        //Riattivo external interrupt su INT0 - IR Receiver
47        EIMSK |= 1 << INT0;
48    }
49 }

```

Questa funzione gestisce l'interrupt dei Pushbuttons (INT1).



```
1 // External Interrupt Pushbuttons
2 ISR(INT1_vect){
3   // Aggiorno il registro per indicare l'intrusione
4   SMY_FLAG(2,1);
5 }
```

Funzioni Custom

1. **'greenLED()'**: imposta il LED verde.
2. **'flash()'**: imposta il LED rosso lampeggiante.
3. **'read_data()'**: legge i dati dai sensori di movimento e imposta lo stato dell'allarme in caso di esito positivo (intrusione).
4. **'bitSwap(command)'**: consente di invertire l'ordine dei bit all'interno di un byte per la corretta decodifica del comando. Il segnale IR NEC invia per primo il bit meno significativo. Ad esempio, il pulsante "Rosso" viene ricevuto in questo modo:
0100 0101 = 0x45. La sua vera decodifica è in realtà: 1010 0010 = 0xA2.
5. **'updatePWMFrequency(frequency)'**: imposta la frequenza del buzzer in base al parametro passato.
6. **'updateWaveform()'**: aggiorna la lunghezza d'onda del buzzer in base allo stato del flag.


```

1  /* FUNZIONI CUSTOM */
2
3  // Setup LED verde
4  void greenLED(){
5      // Imposto a 0 il bit PORTB1 (rosso)
6      PORTB &= ~(1 << PORTB1);
7      // Imposto a 1 il bit PORTB0 (verde)
8      PORTB |= 1 << PORTB0;
9  }
10
11 // Setup LED rosso lampeggiante
12 void flash(){
13     //PORTB &= ~(1 << PORTB0);
14     if(PINB & _BV(PINB1)){
15         PORTB &= ~(1 << PORTB1); // Spengo
16     }else{
17         PORTB |= 1 << PORTB1; // Accendo
18     }
19 }
20
21 // Lettura per rilevamento intrusione
22 void read_data(){
23     // Controllo i pin dei sensori di movimento e il registro dei pulsanti
24     if((PINB & _BV(PINB4)) || (PINB & _BV(PINB3)) || (GPION0 & _BV(2))){
25         alarm_state = 2;
26     }
27 }

```

```

1 // Swap dei bit per decodifica corretta dei segnali NEC
2 byte bitSwap(byte command){
3     command = ((command & 0x01) << 7) |
4               ((command & 0x02) << 5) |
5               ((command & 0x04) << 3) |
6               ((command & 0x08) << 1) |
7               ((command & 0x10) >> 1) |
8               ((command & 0x20) >> 3) |
9               ((command & 0x40) >> 5) |
10              ((command & 0x80) >> 7);
11     return command;
12 }

```

sketch.ino

```
1 // Tempo per ogni cambio di lunghezza d'onda del Buzzer
2 void updatePWMFrequency(unsigned long frequency) {
3     // Calcola il nuovo periodo per la frequenza data
4     unsigned long period = F_CPU / (64UL * frequency) - 1;
5
6     // Imposta il nuovo periodo nel registro ICR1
7     ICR1 = period;
8 }
9
10 // Cambio della lunghezza d'onda del Buzzer
11 void updateWaveform() {
12     if (GPBIO0 & _BV(0)) {
13         updatePWMFrequency(1000); // Imposto frequenza a 1000Hz
14         SMY_FLAG(0,0);
15     } else {
16         updatePWMFrequency(500); // Imposto frequenza a 500Hz
17         SMY_FLAG(0,1);
18     }
19 }
```