ANALIZZATORE TELECOMANDI

UNICAM - Architettura degli elaboratori ST1213

Massimo Trojani

20/02/2024

Appello parziale modulo Lab AE

Panoramica

In questo progetto analizzeremo il processo della comunicazione telecomandata, concentrandoci in particolare sul riconoscimento del tasto premuto su un telecomando.

Obiettivi

1. **Identificare i segnali provenienti da un telecomando ed escludere i disturbi esterni**
2. **Decifrare il pacchetto ricevuto seguendo il protocollo appropriato**
3. **Salvare il valore del tasto premuto sul telecomando**

Tappe intermedie

I. Analisi del problema

Tradurre le specifiche del progetto in un flusso logico.
Individuare i componenti logici.

II. Statechart

Costruire il grafo delle transizioni dello stato.

III. Consolidamento dei componenti

Identificare i componenti funzionali.
Spiegare il funzionamento di ciascun componente.

IV. Sintesi

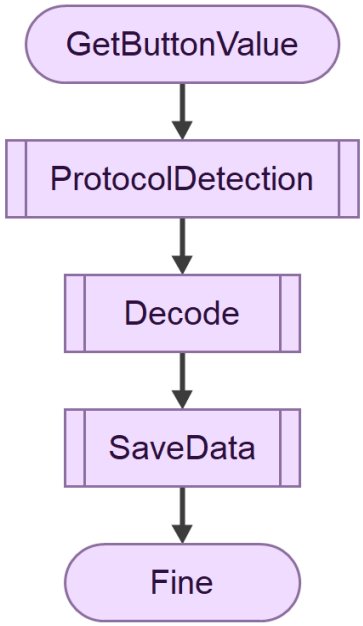
Realizzare i componenti identificati nella fase precedente.

V. Implementazione & Verifica

Assemblare i componenti creati in una struttura interconnessa.
Verificare il buon funzionamento del progetto.

I. ANALISI DEL PROBLEMA

Si suddivide il problema di “Riconoscere il tasto premuto su un telecomando” in 3 sotto-problemi principali:



1. **Riconoscere il protocollo del telecomando**
2. **Utilizzare il protocollo riconosciuto per decodificare il valore del tasto premuto**
3. **Salvare il valore in memoria**

Queste 3 steps devono essere eseguite ogni volta che premiamo un qualsiasi tasto del nostro telecomando. Per realizzare una soluzione a questi problemi, è necessario comprendere innanzitutto la natura della comunicazione tramite telecomando.

In questo progetto tratteremo la comunicazione via **onde infrarosse (IR)** *[700nm - 1mm]*, e come in ogni tipo di comunicazione, ci sono 2 agenti principali: un **Emittente** e un **Ricevitore**. Nel nostro caso, il *Telecomando IR* funge da emittente e il *Ricevitore IR* sarà il dispositivo di ricezione. L'implementazione dell'emittente non sarà oggetto di questa trattazione; ci concentreremo invece sullo sviluppo della logica alla base del funzionamento del ricevitore.

- **FUNZIONALITÀ CHIAVE :**

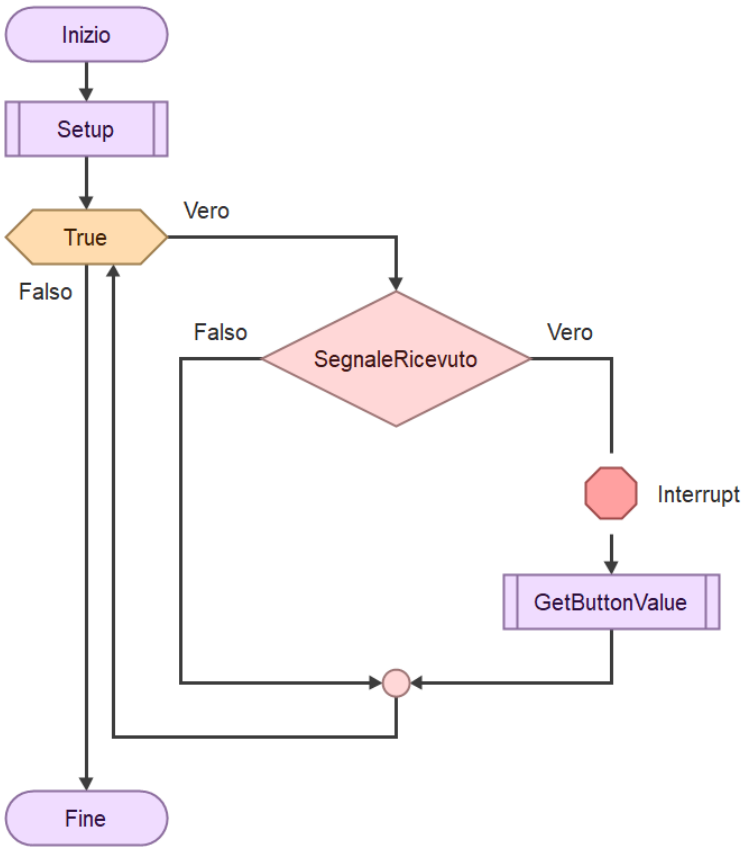
Osserviamo che un telecomando IR generico (come quello della TV), ha due funzionalità di base: ***comunicare che un tasto singolo è stato premuto*** e ***comunicare che un tasto è premuto continuamente***. Inoltre, un telecomando è un prodotto commerciale che tipicamente accompagna un dispositivo controllabile da esso. Di conseguenza, ogni azienda produce i propri telecomandi dedicati ai propri dispositivi. Da qui nasce il concetto di ***protocollo di codifica*** rappresenta il modo in cui l'informazione imitata viene decodificata nel segnale emesso. Questo protocollo varia per ogni produttore di telecomandi.

- **OBIETTIVO FINALE :**

Dunque, il nostro ricevitore deve essere capace di estrarre e salvare il valore di un tasto qualsiasi, premuto in qualsiasi modo, su qualsiasi telecomando.

Il ricevitore deve eseguire la logica ***GetButtonValue*** ogni volta che un nuovo tasto viene premuto. Pertanto, il ricevitore deve sempre costantemente monitorare se un nuovo segnale è stato ricevuto. Per fare ciò, è necessario utilizzare ***un ricevitore infrarosso (IR-RCVR)*** che sia in grado di escludere i disturbi esterni, ossia *(Le onde IR che non rientrano nel range delle frequenze supportate, che è di circa 38 kHz)*.
L'output di questo dispositivo verrà considerato come un evento di **Interrupt**.

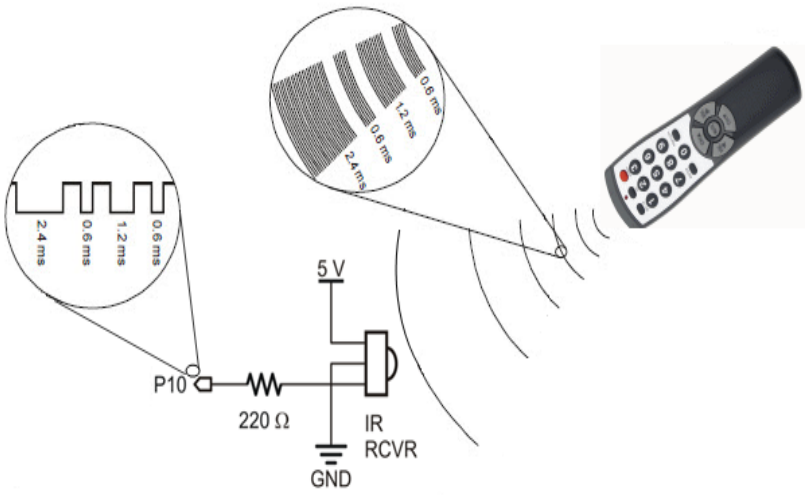
Per garantire il corretto funzionamento del sistema, sarà necessaria una fase di ***Setup*** durante la quale verrà eseguita l'inizializzazione del processo.



- **INTERRUPT :**

Un interrupt rappresenta un cambiamento di stato logico dell'output del ***ricevitore infrarosso (IR-RCVR)***, indicando la ricezione di un ***Impulso.*** Generalmente ogni interrupt segna l'inizio di un impulso e la fine di un altro.

Di conseguenza, ogni impulso ha una ***durata di tempo*** e uno ***stato logico***.



- **IMPULSO:**

Il segnale ricevuto assume la forma di un ***Pacchetto***, il quale consiste in una sequenza di impulsi consecutivi. L'informazione è codificata nella durata e nello stato logico ***(segnale modulato ricevuto*** o ***spazio vuoto)*** di questi impulsi, in conformità con ***il protocollo di codifica*** utilizzato dal telecomando.

1. PROTOCOL DETECTION

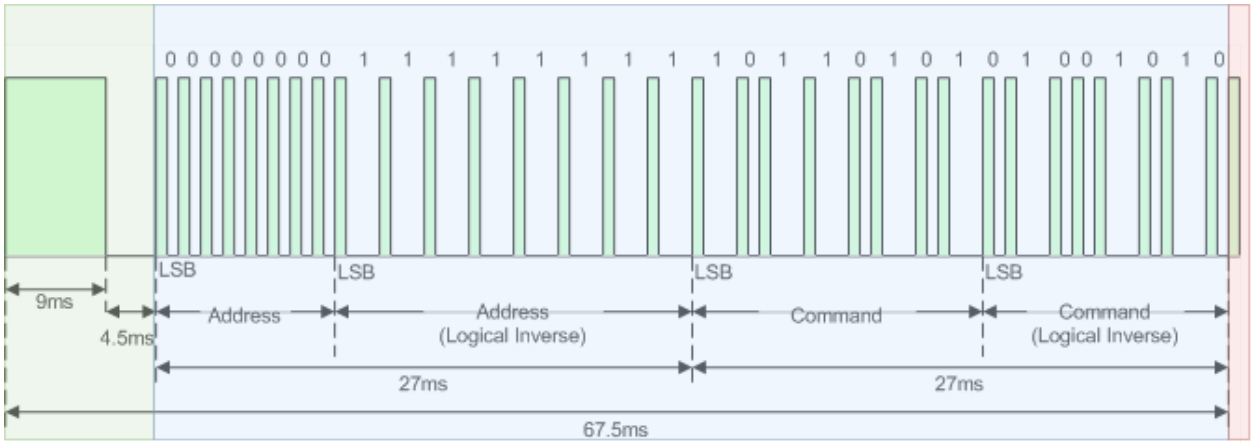
Un protocollo di codifica è uno schema standardizzato per codificare i ***dati*** che desideriamo trasmettere sotto forma di impulsi. Per un ricevitore, il protocollo non è altro che un ***insieme di regole*** che definisce come estrarre i dati dal pacchetto ricevuto.

Esistono molti protocolli, tra i più comuni troviamo: ***NEC, SIRCS, RC5, JAPAN, SAMSUNG, ...***

Per convenzione, un protocollo è definito da 3 campi principali: ***START OF FRAME***, ***DATA***, ***END OF FRAME***

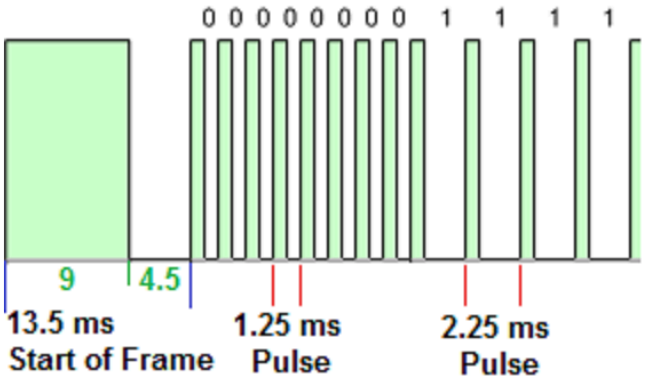
- **START OF FRAME :** Serve per identificare il protocollo utilizzato.
- **DATA :** Questa porzione contiene l'informazione da comunicare.
- **END OF FRAME :** Serve per indicare la fine della trasmissione dei dati.

Ogni protocollo è una variazione di questa struttura. Prendiamo ad 'esempio il protocollo ***NEC.***



Si nota rapidamente che la durata degli impulsi ha una caratteristica in comune. Le durate sono multipli di un certo numero, che per protocollo ***NEC*** è di ***562,5 µs***.

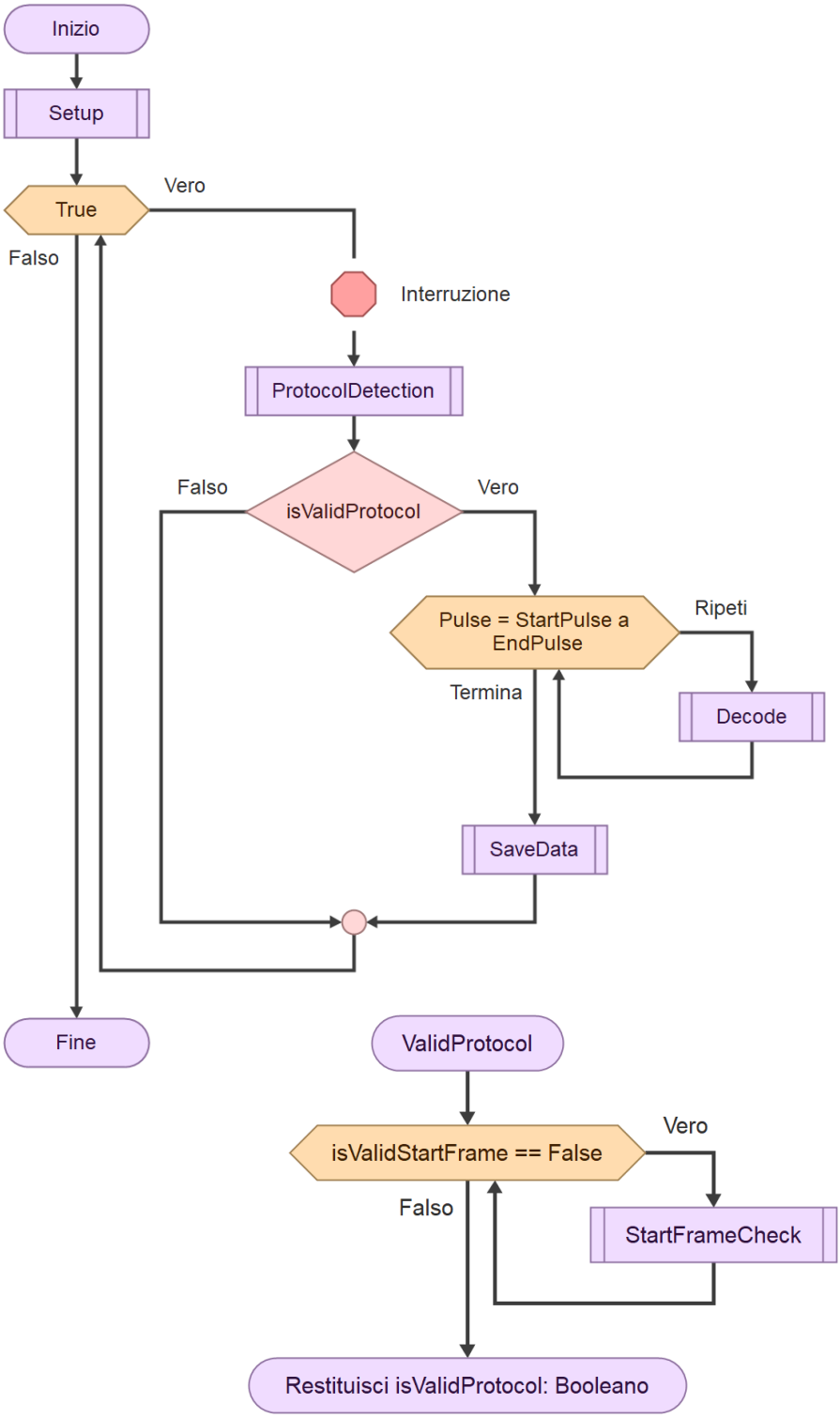
Questo numero rappresenta ***il tempo base (T)***, specifico per ogni protocollo.

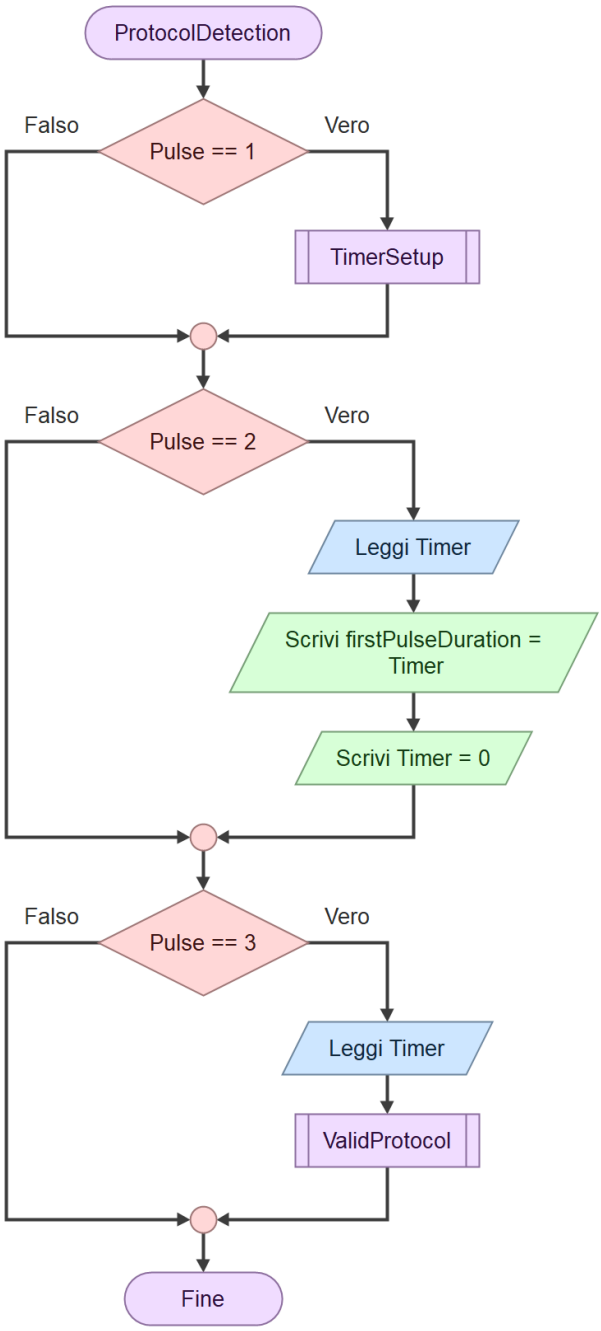


Allora, le regole del protocollo NEC si presenteranno sotto forma di:

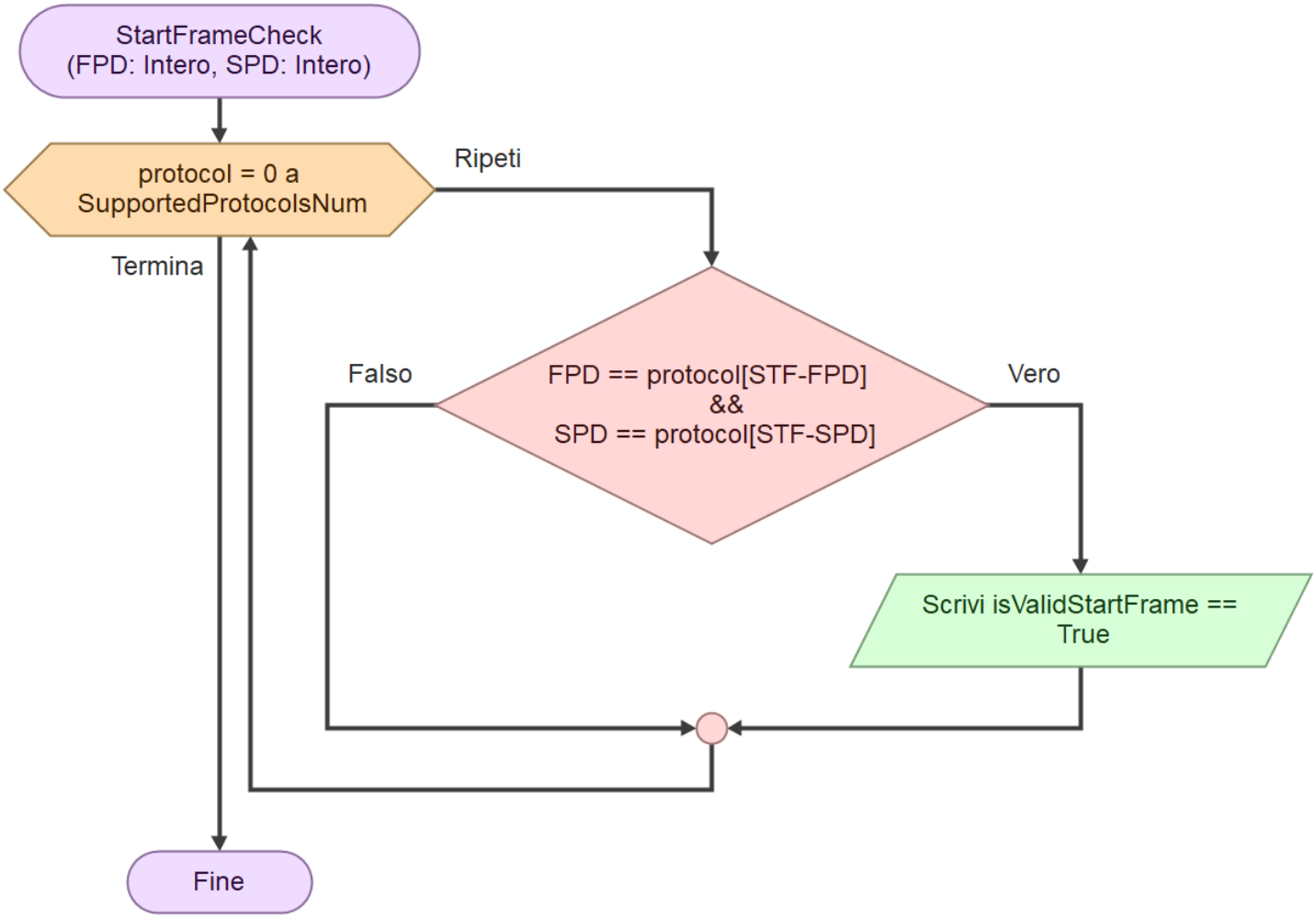
START OF FRAME : 16T [HIGH] - 8T [LOW] | **DIGITAL 0 : 1T [HIGH] - 1T [LOW]**

END OF FRAME : 1T [HIGH] - 1T [LOW] | **DIGITAL 1 : 1T [HIGH] - 3T [LOW]**





** ad ogni interrupt "Pulse" viene incrementato di uno*



Secondo questo flusso logico, saremo in grado di identificare il protocollo utilizzato al terzo interrupt, che indica la fine del ***START OF FRAME (STF)***. È necessario salvare le specifiche dei protocolli, *quindi saremo capaci di identificare solo i protocolli per i quali abbiamo le relative regole,* così chiamati ***Supported Protocols.***

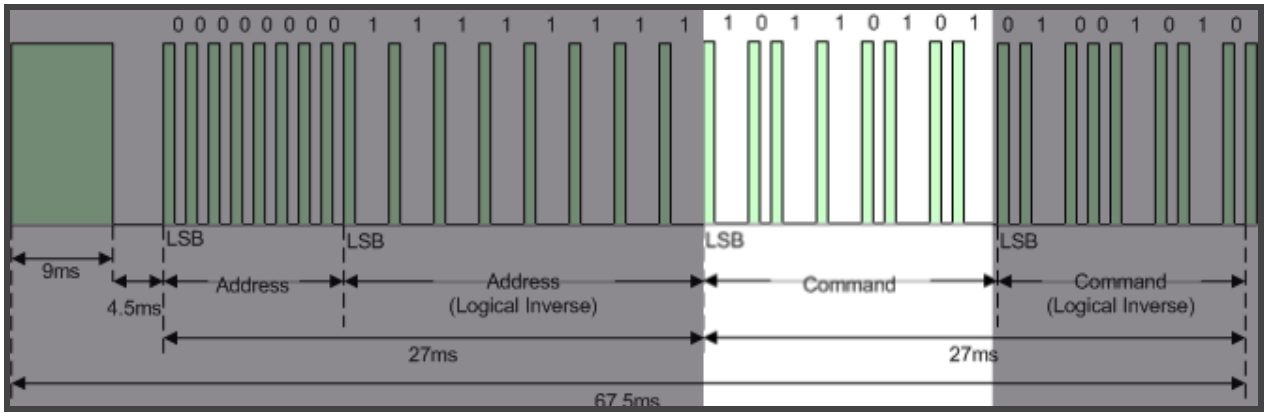
Dopo aver identificato il protocollo, sapremo come saranno presentati i dati successivi, più importante ancora, dove si trova il valore desiderato ***(campo COMMAND)*** e da ***quanti bit sarà composto.*** Al fine di ottenere correttamente il valore del tasto premuto.

Aggiungiamo alle regole ***(NEC)*** :

TOTAL LENGTH	**: 68 Pulses**	**DATA STARTING PULSE**	**: Pulse 35**
FPD PIN-STATE	**: HIGH**	**DATA LENGTH**	**: 8 bit**

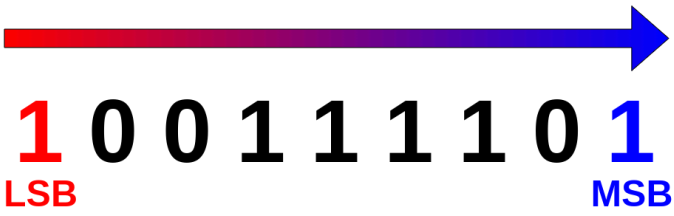
FPD : First Pulse Duration*

2. DECODE DATA



Nel caso del protocollo NEC, il campo comando è costituito da un ***byte non segnato (unsigned),*** dove il primo bit ricevuto è il ***LSB (Least Significant Bit)*** e l'ultimo è il ***MSB (Most Significant Bit).*** consentendo quindi la codifica di ***256 valori distinti.***

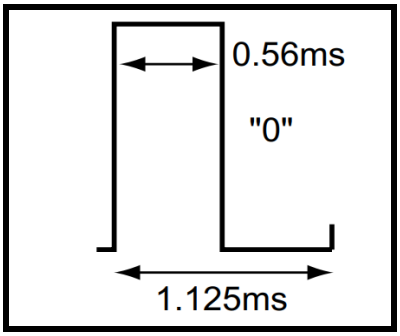
Di conseguenza, un telecomando NEC può avere al massimo ***256 possibili pulsanti.***

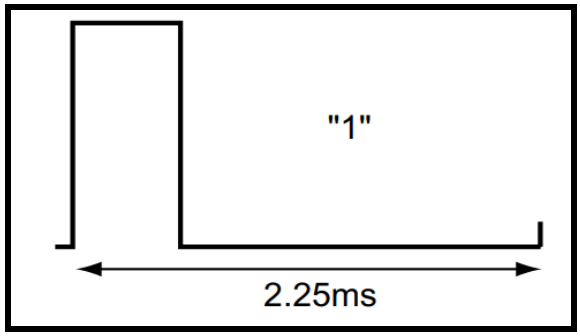


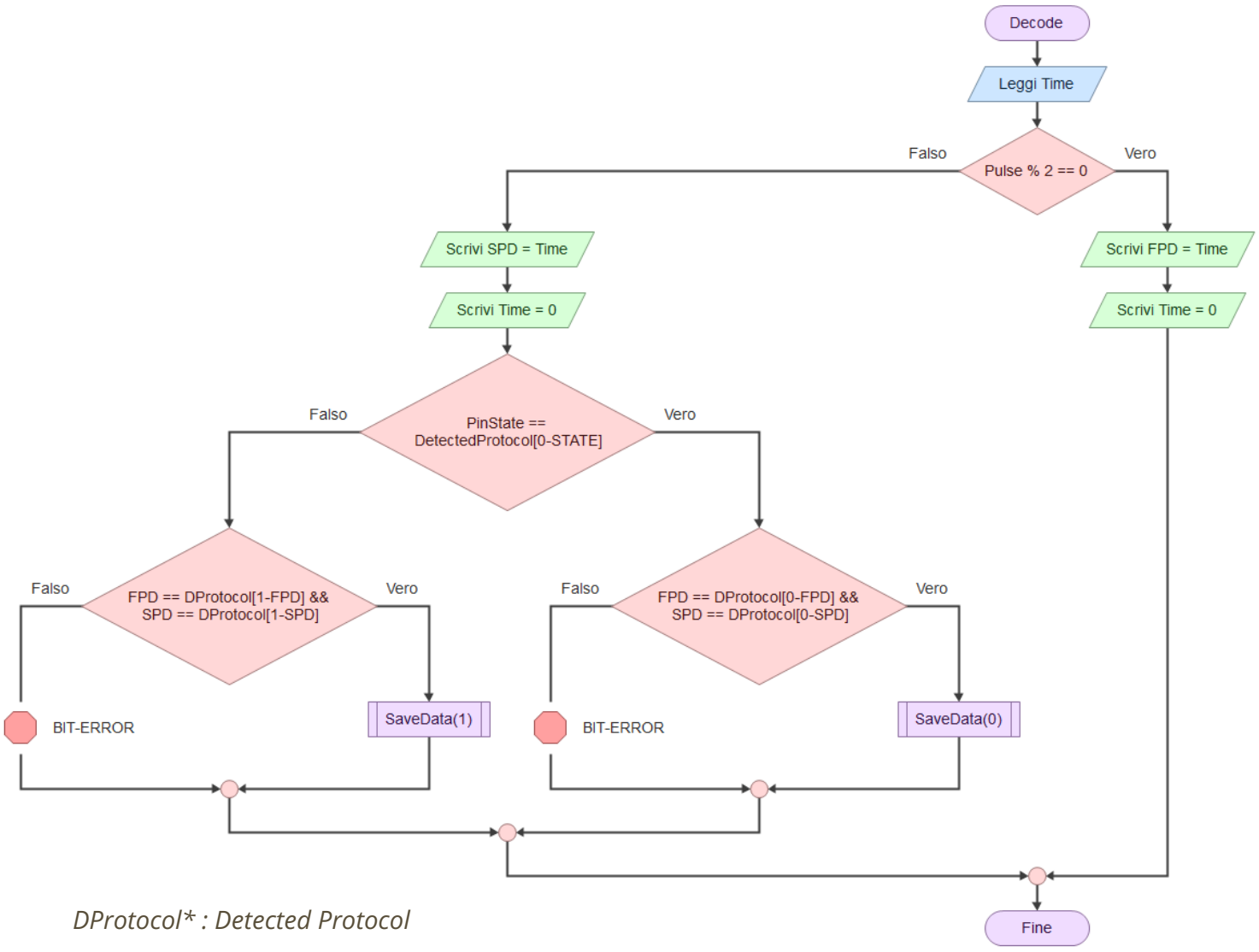
Per decodificare il campo COMMAND, è necessario registrare la durata degli impulsi per determinare se si tratta di un ***bit 0*** o ***1.*** A tal fine, dobbiamo memorizzare il tempo ad ogni interrupt. Tuttavia, nel secondo impulso, dobbiamo confrontare la durata del ***First Pulse Duration (FPD)*** e ***del Second Pulse Duration (SPD)*** con quelle specificate dal protocollo identificato.

- **ESEMPIO:**

Ad esempio, se il **bit** ricevuto è **0**, ci aspettiamo che ***FPD*** e ***SPD*** siano entrambi pari a **1T**, con lo stato logico **HIGH** al tempo di controllo.

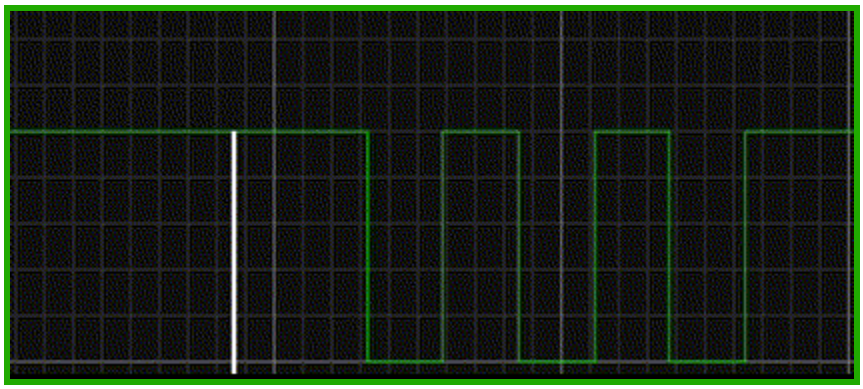






DProtocol : Detected Protocol*

— VISUALIZZAZIONE DATA (NEC) —



Questo metodo di decodifica verrà utilizzato solo se è stato premuto un nuovo tasto. Tuttavia, se il tasto viene premuto continuamente, il telecomando emetterà lo stesso segnale a ***intervalli regolari,*** detti ***intervallo tasto ripetuto.*** In questo caso, ci basterà verificare il tempo tra due segnali di *Start of Frame (STF)* per registrare il valore che sarà uguale al Data precedente.

3. SAVE DATA

Adesso che siamo in grado di identificare i singoli bit del ***campo COMMAND,*** dobbiamo salvarli correttamente. Per fare ciò, è necessario dichiarare una variabile temporanea per memorizzare i bit durante la fase di decodifica. È importante notare che questi bit sono ricevuti in ordine inverso rispetto all'orientamento tipico di un byte, dove il ***bit meno significativo (LSB)*** è il bit nella posizione più a sinistra. Quindi, durante la decodifica, dovremo salvarli nell'ordine corretto.

1. **Esempio: SaveData(1)**

   bitValue = 1

   Data = 00000000

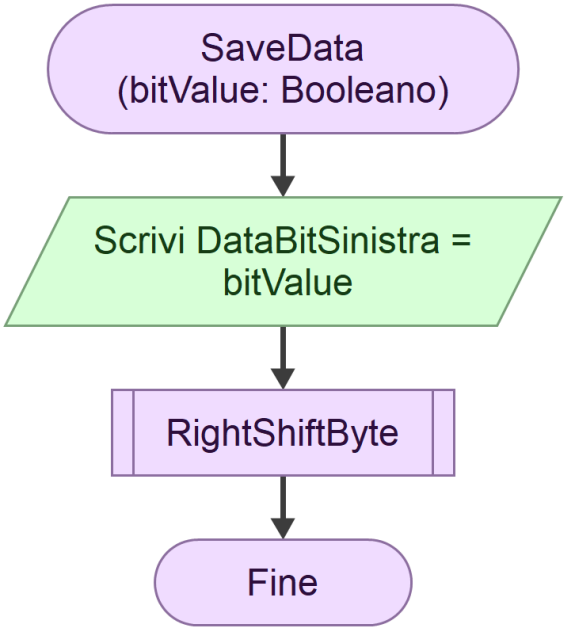
   Data = 10000000

2. **Esempio: SaveData(0)**

   bitValue = 0

   Data = 10000000

   Data = 01000000



Una volta decodificato l'intero campo COMMAND e quindi aver correttamente ottenuto il valore del pulsante, salvandolo in un ***BYTE*** nel formato corretto, dobbiamo memorizzare l'intero byte nella memoria di massa. Nel nostro caso, questa memoria è lunga ***8 BYTE*** ed è ***allineata a destra.*** Pertanto, è necessario spostare tutti i dati salvati ***(SAVED_DATA)*** a sinistra per fare spazio a un nuovo byte, che occuperà il posto liberato.

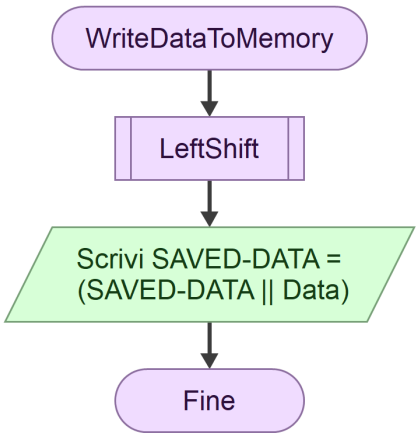
Esempio: salvare (100) su la memoria dopo aver già salvato (240)

SAVED_DATA = ... 00000000 11110000

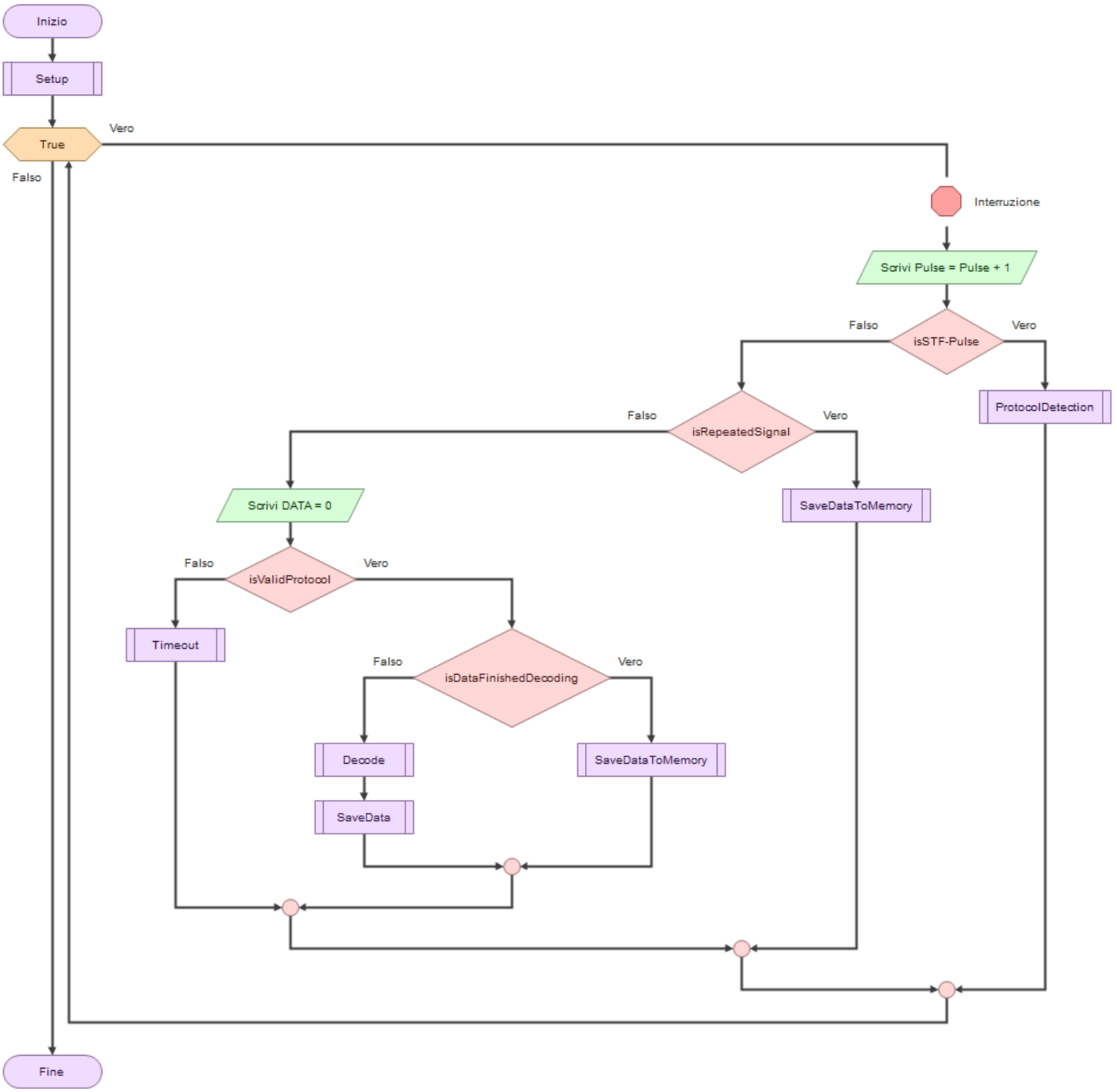
DATA = 01100100 (100 dec)

SAVED_DATA = ... 11110000 00000000

SAVED_DATA = ... 11110000 01100100

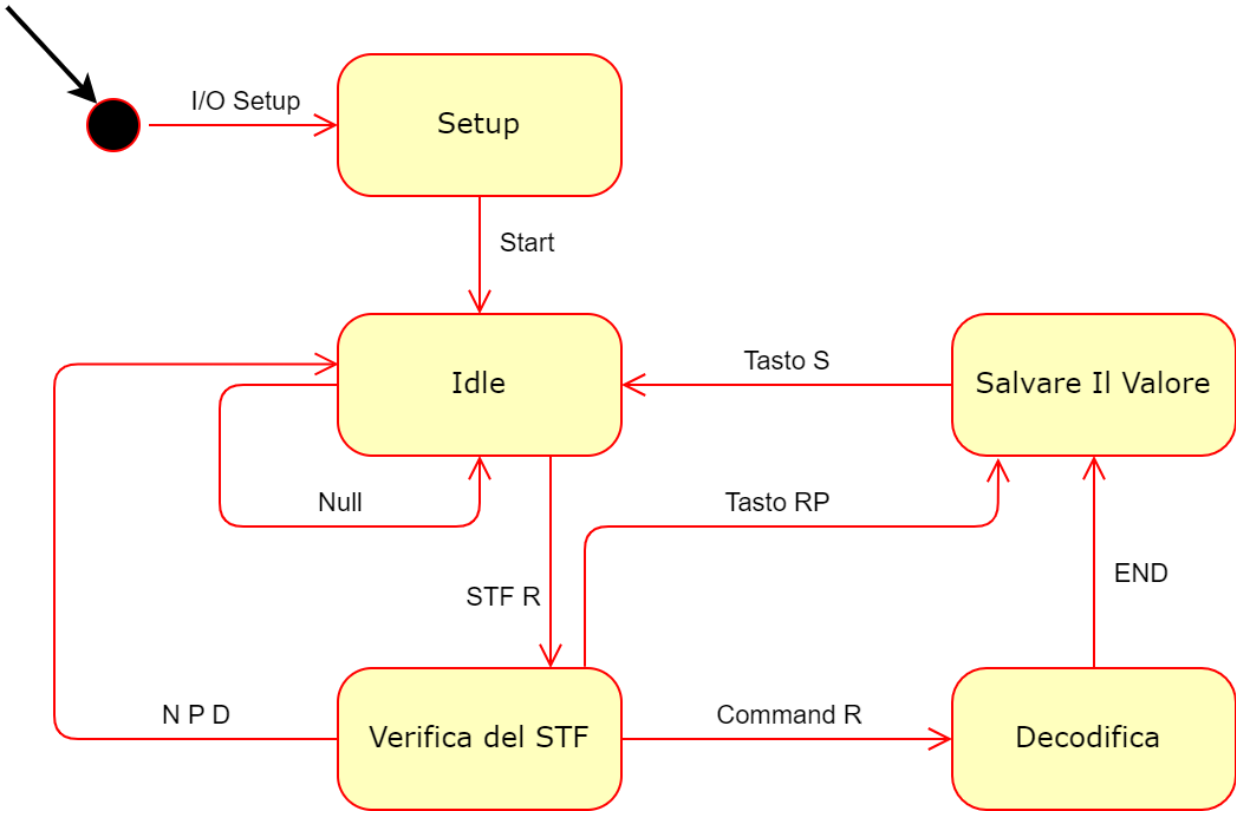


4. FLOWCHART GENERALE



II. STATECHART

Un statechart è un diagramma che visualizza lo stato di un sistema e le transizioni tra di essi. È composto da stati, eventi e azioni, offrendo una panoramica chiara del comportamento del sistema. È uno strumento essenziale per progettare e analizzare sistemi complessi.



**Legenda nella pagina successiva*

1. STATI

Setup: Inizializzazione dell'automa ***(ricevere interrupt, tempo, protocolli riconosciuti, …)***.

Idle: Stato di base dell'automa, nessun interrupt avvenuto.

Verifica del STF: Verifica se si tratta di un STF di un protocollo ***accettato dall'automa*** e se il tempo trascorso dallo STF corrente al precedente è uguale all'intervallo per i tasti ripetuti.

Decodifica: Estrazione dei singoli bit dal campo COMMAND

Salvare il Valore: Salvataggio del valore codificato in memoria, ordinati per rilevanza.

2. TRANSIZIONI

I/O Setup: Setup dei sensori esterni

Start: Fine del Setup ***(pronto per ricevere i pacchetti IR)***

Null: Nessun Interrupt rilevato

STF R (Start Of Frame Ricevuto): Primo interrupt ricevuto

N P D (No Protocol Detected): Il STF ricevuto non è accettato dall'automa

Tasto RP (Tasto Ripetuto): Il tempo trascorso dallo STF precedente indica che si tratta di un tasto premuto continuamente

Command R (Command Ricevuto): Primo impulso del campo command rilevato

END: Ultimo impulso ricevuto

Tasto S (Tasto Salvato): Valore del campo command salvato in memoria correttamente

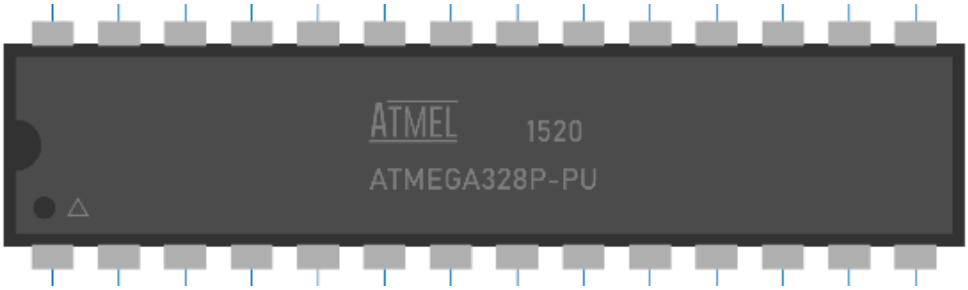
III. CONSOLIDAMENTO DEI COMPONENTI

Per realizzare l'automa descritto, ci servirebbero componenti per implementare le funzioni logiche in questione. Dalle informazioni fornite nel flusso logico della prima parte, si deduce che per garantire il corretto funzionamento del progetto sarà essenziale avere:

1. **Ricevitore IR:** Per ricevere i segnali infrarossi modulati.
2. **Timer:** Per misurare la durata degli impulsi e il tempo trascorso tra due STF.
3. **Detettore di protocollo:** Per verificare se il protocollo è riconosciuto dall'automa.
4. **Decodificatore:** Per estrarre e ottenere il valore immagazzinato nel segnale.
5. **Gestore di memoria:** Per salvare e gestire i dati ottenuti precedentemente.

I componenti descritti sono utili solo se interconnessi in una struttura *(come un elaboratore)* che fornisca il supporto necessario per le funzionalità richieste. Una delle scelte più convenienti in tal senso è una ***MCU (Microcontroller Unit)*** come ad esempio **Arduino Uno® Rev3,** dotato di un **ATmega328P,** che offre: ***Pin I/O, timers, supporto agli interrupt*** e ***memoria d'archivio***.

Questa struttura interconnessa dei componenti permetterà al sistema di operare in modo coordinato ed efficiente, consentendo l'esecuzione del flusso logico del programma .



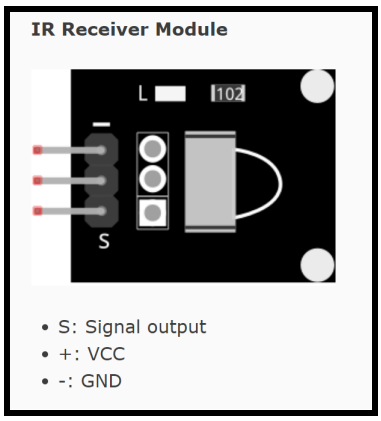
IV. SINTESI

Dopo aver identificato i componenti necessari per l'automa, è giunto il momento di realizzarli, un componente alla volta.

1. RICEVITORE IR

Il funzionamento preciso del ricevitore IR è fuori dall'ambito di questo progetto. Ci basta un modulo che sia in grado di rilevare ***l'onda modulata (portante il segnale)*** e tradurla in un semplice output logico. In questo modo, se viene ricevuto il segnale, il modulo genererà un impulso LOW, altrimenti genererà un impulso HIGH.

Per ciò scelgo il ***HX1838*** che opera a ***38kHz,*** perfetto per il protocollo NEC *(il protocollo più usato sul mercato).* L'output del modulo ***[Pin S]*** dovrà essere collegato al ***[PIN Digital 2 o 3]*** del ***Arduino,*** perché sono gli unici pin che sopportano gli ***interrupt esterni.***



HX1838

2. TIMER

Un Timer è un dispositivo che conta il tempo. Per fare ciò, il timer deve avere un ***evento di riferimento di base,*** chiamato ***T-base,*** di cui è noto il tempo trascorso. Nel nostro caso si tratta di un oscillatore a cristallo, che fornisce una frequenza di clock pari a *16 MHz,* ***(Tb = 1/16µs).*** Il timer incrementa il ***valore salvato (TCNT)*** in base a questo evento, il che significa che il tempo del timer è in funzione di ***T-base.*** Per ottenere un maggior controllo sul time step, ogni **TIMER** dispone di un **PRESCALER,** il quale moltiplica il numero di occorrenze dell'evento di riferimento necessarie per incrementare il valore effettivo del ***TCNT.*** Il tempo del timer può quindi essere calcolato utilizzando la seguente formula:

$$Time = Tb \times Prescaler \times TCNT$$

**L'unità di Time è quella di T-base (µs), essendo TimeValue solo un intero .*

Nel nostro caso utilizzeremo il ***TIMER1*** del nostro MCU, che è un *timer a 16 bit.* ***TCNT1,*** o ***Timer/Counter Register 1,*** è il registro principale associato al TIMER1. Esso tiene traccia del conteggio del timer, aumentando di **uno ad ogni *256* cicli di clock.** Utilizzeremo questo timer per misurare con precisione il tempo tra gli impulsi e gestire le operazioni di temporizzazione necessarie per il nostro progetto. *[1 in TCNT1 == 16µs realmente)]*

3. DETETTORE DI PROTOCOLLO

Prima di decodificare il campo command, ***l'automa deve essere in grado di interpretare correttamente gli impulsi di questo campo.*** Questa informazione è specifica per ogni protocollo, quindi è necessario identificare il protocollo corretto prima di procedere con la decodifica. Questo processo avviene analizzando il *Start Of Frame (STF).*

Una volta ricevuto il STF, l'automa confronta il segnale ricevuto con i protocolli supportati. Se il STF ricevuto corrisponde a uno dei protocolli nella lista dei protocolli supportati, l'automa dichiara che il protocollo è stato rilevato e procede alla decodifica del campo command. In caso contrario, se il STF non è riconosciuto tra i protocolli supportati, l'automa annulla l'operazione di analisi dei dati ricevuti, poiché sarebbe incapace di decodificare correttamente il pacchetto senza conoscere il protocollo corretto.

I dati dei protocolli sono salvati nella memoria del nostro MCU. Una volta riconosciuto il protocollo, i dati vengono resi accessibili globalmente, il che significa che diventano disponibili anche per gli altri componenti del sistema.

```
NUM_SUPPORTED_PROTOCOLS 5

NEC_PULSE_TIME      560
SAMSUNG_PULSE_TIME  560
JAPAN_PULSE_TIME    420
SIRCS_PULSE_TIME    600
RC5_PULSE_TIME      889
```

**Il tempo base (T) dei protocolli supportati*

4. DECODIFICATORE

L'obiettivo di questo componente è di rilevare il valore del tasto premuto, immagazzinato del campo command del pacchetto seguendo le regole del protocollo identificato precedentemente.

Dopo aver identificato quale interrupt del pin corrisponde all'inizio del campo desiderato, ad ogni fine della trasmissione di un ***bit*** *(gli interrupt dispari periodici, ad esempio 3, 5, 7, ...),* avviene il controllo della *durata dei due impulsi* e *dello stato logico attuale.* Ripetendo questa operazione per l'intero campo, si ottiene il valore del tasto premuto. *È importante allineare correttamente i bit ricevuti in una variabile d'appoggio,* la quale sarà poi salvata nella memoria dedicata da parte del gestore di memoria principale.

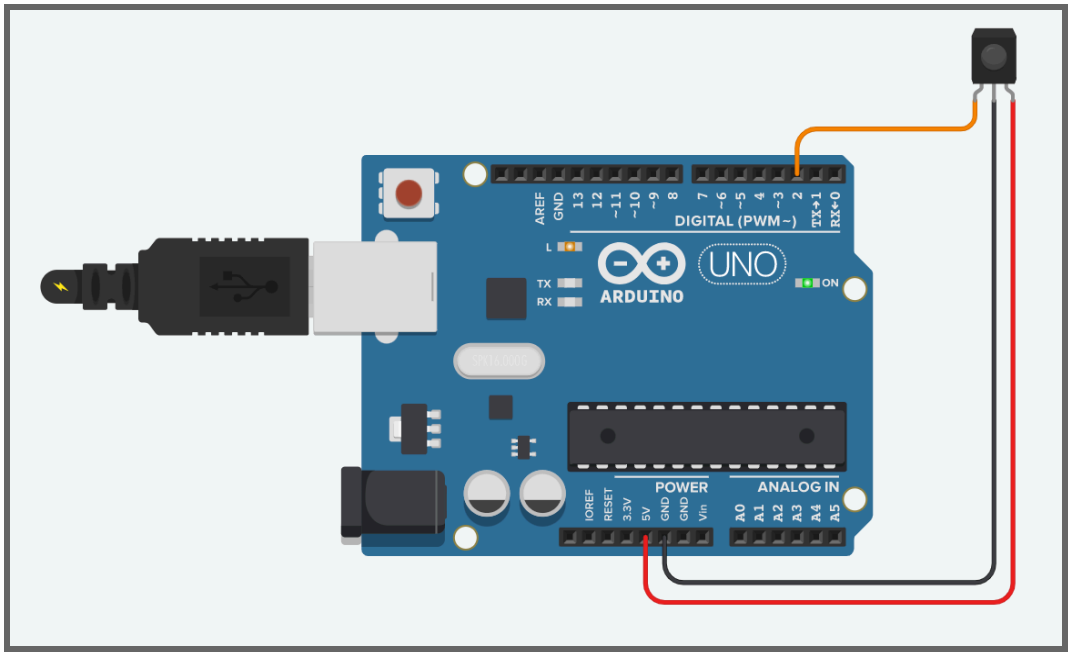
5. GESTORE DI MEMORIA

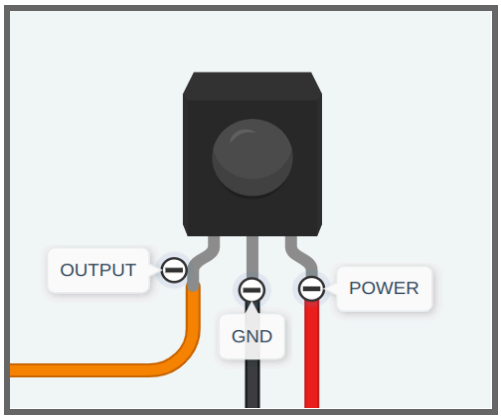
L'ultimo componente serve a salvare il valore ottenuto dal decodificatore in memoria. Il tipo di dato più grande a nostra disposizione sul MCU è il ***uint64_t (unsigned int 64 bit),*** che occupa ***8 byte di memoria.*** Pertanto, la memoria può contenere i dati relativi agli ultimi 8 tasti premuti. Il tasto più recente sarà sempre nella parte meno significativa del ***uint64_t,*** allineato a destra.

Nel caso di un tasto premuto continuamente, il valore di ***DATA*** precedentemente memorizzato nella memoria temporanea viene salvato automaticamente, senza la necessità di decodificare nuovamente il pacchetto.

V. IMPLEMENTAZIONE & VERIFICA

Finalmente abbiamo la comprensione necessaria per integrare tutti i componenti in ***Arduino*** e tradurre il flusso logico in ***codice C eseguibile.*** Per questo utilizzerò il simulatore ***WOKWI,*** ecco il PROGETTO.





POWER: 5V

OUTPUT: PIN 2

Utilizzando ***GDB (GNU Debugger)*** è possibile verificare che il progetto funzioni come previsto, usando print _SAVED_DATA_ alla fine della trasmissione di ogni pacchetto.