Interfaccia SPI - ILI9341

INDICE

1.Analisi

1.1 Testo assegnato

Facendo riferimento all'interfaccia SPI di ATMEGA 328P (cap. 18 -SPI – Serial Peripheral Interface), realizzare le seguenti funzioni primitive, con I/O Wait gestito tramite Interrupt:

- void SPI_MasterOpen(uint8_t clkdiv);

Imposta l'interfaccia SPI come master ed un clock pari fsck/clkdiv (clkdiv=2/4/8.../128

- void SPI_MasterTx(uint8_t data);

Trasmette il byte "data" senza attendere il completamento

- boolean SPI_MasterReady();

Ritorna true o false a seconda se sia in corso o meno la trasmissione di un byte

Per verificare il funzionamento di tali primitive, realizzare un programma che scriva un pixel in una posizione qualsiasi dello schermo LCD https://docs.wokwi.com/parts/wokwi-ili9341

ovvero realizzare:

- void ILI_sendCmd(uint8_t cmd);

Invia un comando, tramite SPI, al processore di ILI9341

- void ILI_sendData(uint8_t data);

Invia un dato da 8 bit, tramite SPI, al processore di ILI9341

- void ILI_sendData16(uint16_t data);

Invia un dato da 16 bit, tramite SPI, al processore di ILI9341

- void ILI_init();

Inizializza lo schermo con le seguenti caratteristiche:

Orientation: Portrait: XMAX=240, YMAX=320

altri parametri di inizializzazione, avendo come riferimento la libreria la libreria Adafruit_ILI9341 e quindi il vettore di comandi e dati: static const uint8_t PROGMEM initcmd[]

```
 static const uint8_t PROGMEM initcmd[] = {
  0xEF, 3, 0x03, 0x80, 0x02,
  0xCF, 3, 0x00, 0xC1, 0x30,
  0xED, 4, 0x64, 0x03, 0x12, 0x81,
  0xE8, 3, 0x85, 0x00, 0x78,
  0xCB, 5, 0x39, 0x2C, 0x00, 0x34, 0x02,
  0xF7, 1, 0x20,
  0xEA, 2, 0x00, 0x00,
  ILI9341_PWCTR1  , 1, 0x23,           // Power control VRH[5:0]
  ILI9341_PWCTR2  , 1, 0x10,           // Power control SAP[2:0];BT[3:0]
  ILI9341_VMCTR1  , 2, 0x3e, 0x28,      // VCM control
  ILI9341_VMCTR2  , 1, 0x86,           // VCM control2
  ILI9341_MADCTL  , 1, 0x48,           // Memory Access Control
  ILI9341_VSCRSADD, 1, 0x00,            // Vertical scroll zero
  ILI9341_PIXFMT  , 1, 0x55,
  ILI9341_FRMCTR1 , 2, 0x00, 0x18,
  ILI9341_DFUNCTR , 3, 0x08, 0x82, 0x27, // Display Function Control
  0xF2, 1, 0x00,                    // 3Gamma Function Disable
  ILI9341_GAMMASET , 1, 0x01,           // Gamma curve selected
  ILI9341_GMCTRP1 , 15, 0x0F, 0x31, 0x2B, 0x0C, 0x0E, 0x08, // Set Gamma
    0x4E, 0xF1, 0x37, 0x07, 0x10, 0x03, 0x0E, 0x09, 0x00,
  ILI9341_GMCTRN1 , 15, 0x00, 0x0E, 0x14, 0x03, 0x11, 0x07, // Set Gamma
    0x31, 0xC1, 0x48, 0x08, 0x0F, 0x0C, 0x31, 0x36, 0x0F,
  ILI9341_SLPOUT  , 0x80,              // Exit Sleep
  ILI9341_DISPON  , 0x80,              // Display on
  0x00                          // End of list
 };
```

1.2 Analisi del problema

Il sistema deve poter essere in grado di trasferire dei dati utilizzando l'interfaccia SPI e mediante l'utilizzo di uno schermo LCD sarà possibile verficare il corretto funzionamento dell'interfaccia scrivendo un pixel in un qualsiasi punto dello schermo.

A partire dal testo fornito emergono alcune definizioni e stati.

***Definizioni**:*

- *Cmd : Comando da inviare allo schermo*
- *Data: Dato da inviare*
  - *Data1 : 8 bit*
  - *Data2: 16 bit*

Stati

- *Schermo = Sleeping, Ready*

Il flusso di esecuzione che descrive il problema può essere rappresentato in questo modo:

1- Seleziona Schermo

2- Se statoSchermo = Sleeping allora

3- statoSchermo = inviaComando(Ready)

4- InviaComando(IniziaTrasmissione)

5- Fino a quando ci sono dati da inviare

6- inviaData(DataN)

7- mostraDatoRicevutoSuSchermo()

cicla 5

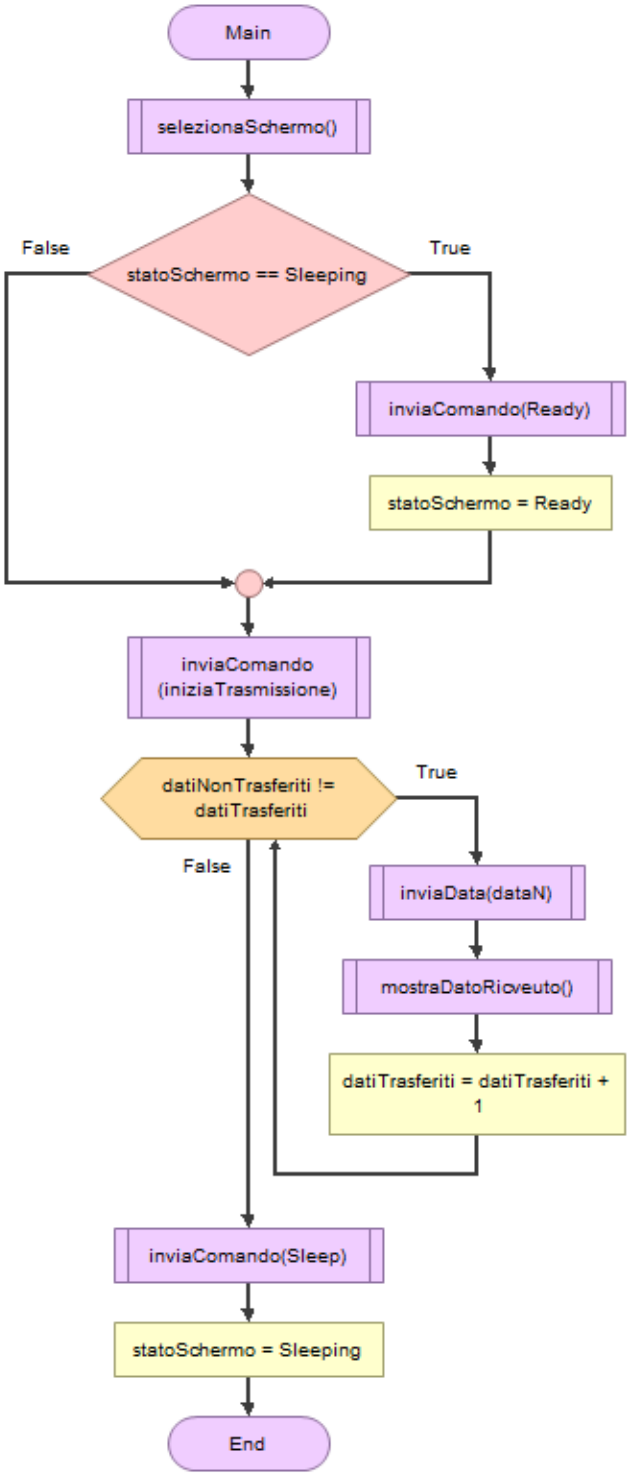
8- statoSchermo = inviaComando(Sleep)

FINE

Il punto 8 descrive un passaggio in cui lo schermo dopo un certo periodo di inattività va in modalità Sleeping.

L'immagine seguente mostra in modo strutturato tramite flowgorithm il semplice flusso di esecuzione che si è derivato da una prima analisi del testo.

La dichiarazione e definizione delle variabili è stata omessa al fine di mantenere il diagramma di un formato leggibile. Tutti i diagrammi riportati nella seguente relazione saranno disponibili in uno zip allegato. I diagrammi contrassegnati con NE sono stati realizzati allo scopo di mostrare il funzionamento ma non sono eseguibili.



***1.1** Diagramma flusso di esecuzione del sistema*

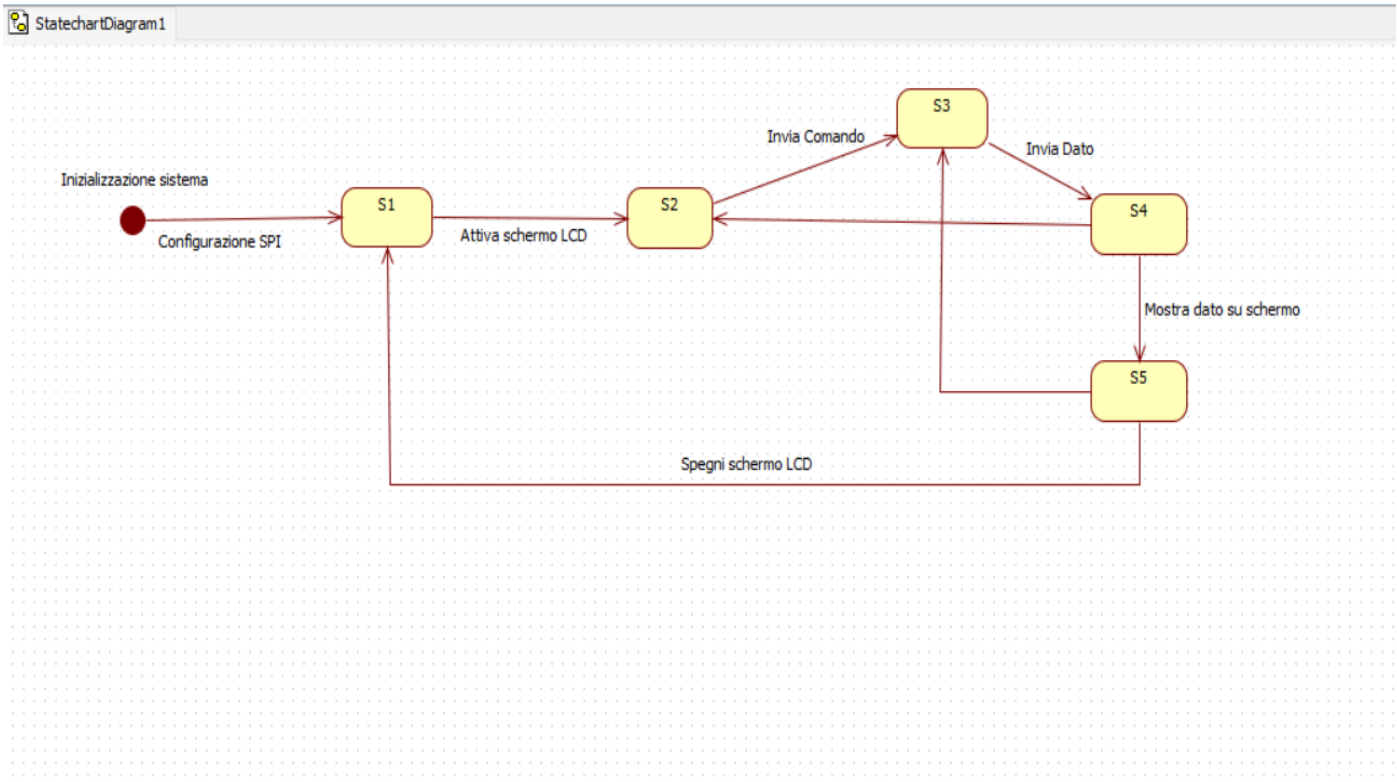
Moduli di elaborazione

Emergono due moduli di elaborazione principali, un modulo che si occuperà dell'invio dei dati e quindi legato all'interfaccia SPI, e un altro modulo invece legato

alla visualizzazione dei pixel sullo schermo LCD.

Possiamo quindi esprimere attraverso i diagrammi di stato il funzionamento del sistema.

In questa fase iniziale dell'analisi sono stati omessi gli input/output sulle transizioni in quanto strettamente collegati all'implentazione della soluzione per l'utilizzo dell'interfaccia SPI e dello schermo LCD. Successivamente tali diagrammi verranno rivisiti e completati



***1.2** Diagramma flusso di esecuzione del sistema*

STATO S1: *Configurazione SPI*

- *Descrizione: L'interfaccia SPI è configurata*
- *Transizioni:*

  Input: Il sistema riceve la configurazione per l'interfaccia SPI

  Output: L'interfaccia SPI è configurata e si può attivare lo schermo

STATO S2: *Attivaizone schermo LCD*

- *Descrizione: Lo schermo LCD è acceso(attivo)*
- *Input: Il sistema riceve il comando di accendere lo schermo*

- *Output: Una volta acceso lo schermo è possibile inviare comandi e dati*

STATO S3: ***Invio comando***

- *Descrizione: Invio di un comando*
- *Input: Il sistema invia un comando*
- *Output: Ricevuto il comando sarà possibile inviare il dato*

STATO S4: ***Invio dato***

- *Descrizione: Invio del dato*
- *Input: Il sistema invia il dato*
- *Output: Ricevuto il dato lo schermo potrà mostrarlo oppure inviare un nuovo comando*

STATO S5: ***Mostra dati su schermo***

- *Descrizione:Vengono mostrati i dati ricevuti sullo schermo*
- *Input: Lo schermo mostra i dati ricevuti*
- *Output: Il sistema può continuare ad inviare dati oppure spegnere lo schermo*

1.3 Analisi della soluzione adottata

Avendo definito l'architettura funzionale del sistema in modo astratto, si focalizza ora l'attenzione sull'analisi dettagliata dei moduli di elaborazione, valutando le loro specifiche tecniche e le interfacce necessarie per garantire il corretto funzionamento dell'intero sistema.

1.3.1 Modulo di elaborazione: Invio dei Dati tramite SPI

Il modulo per l'invio dei dati tramite interfaccia SPI si occupa di gestire la comunicazione sincrona tra l'MCU , configurato come master e un dispositivo slave. Il modulo sarà quindi responsabile della trasmissione di dati e comandi dal master verso lo slave in modo sincrono e senza errori.
Dal datasheet del ATMega328P emerge che per avviare la trasmmissione dei dati deve essere portato ad uno stato basso ossia impostato a 0 il bit per lo slave select, che rappresenta lo slave selezionato per la trasmissione. Successivamente il master genera un impulso di clock e il dato viene shiftato dal registro del master a quello

dello slave. Terminato l'invio del byte viene interotto l'impulso di clock generato dal master e impostato a 1 la flag che indica il termine della trasmissione.
Tale flusso può essere schematizzato in questo modo

1. ***Configurazione dell'Interfaccia SPI:***
    - *Il microcontrollore viene configurato come master.*
    - *Viene impostata la frequenza di clock SPI.*
    - *Viene abilitata la gestione degli interrupt per il completamento della trasmissione.*

2. ***Preparazione dei Dati da Trasmettere:***
    - *I dati da inviare vengono caricati nel registro dati SPI.*

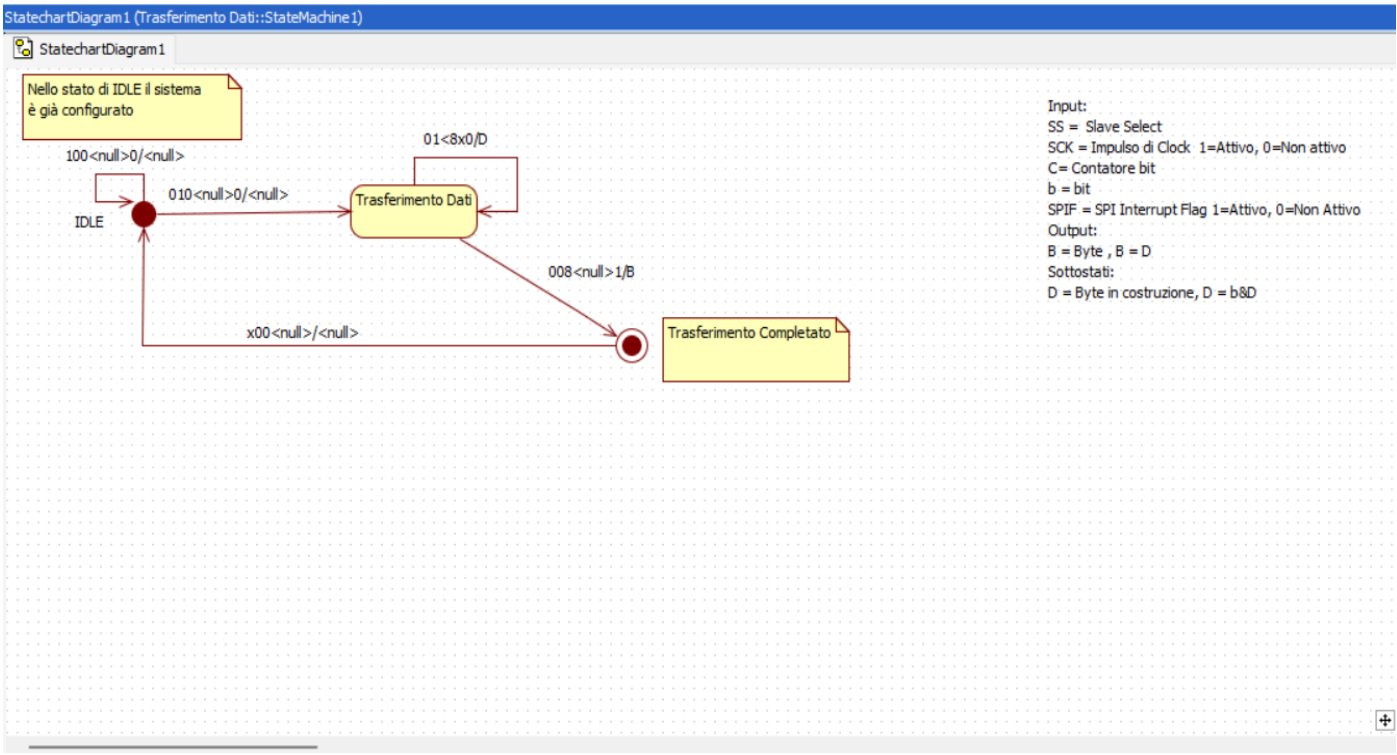
3. ***Inizio della Trasmissione:***
    - *La trasmissione dei dati inizia automaticamente quando il registro dati viene scritto.*
    - *Il modulo SPI invia i bit sul bus MOSI (Master Out, Slave In), sincronizzandosi con il clock generato sul pin SCK.*

4. ***Attesa del Completamento della Trasmissione:***
    - *Il microcontrollore attende il completamento della trasmissione o esegue altre operazioni fino a quando non viene generato un interrupt che segnala la fine della trasmissione.*

5. ***Gestione dell'Interrupt:***
    - *All'avvenuta trasmissione, il modulo genera un interrupt che viene gestito dalla routine di servizio ISR.*
    - *Il microcontrollore può verificare lo stato di completamento della trasmissione e decidere se inviare altri dati o terminare la comunicazione.*

6. ***Conclusione della Comunicazione:***
    - *Una volta completata la trasmissione dei dati necessari, il microcontrollore può disabilitare l'interfaccia SPI o lasciare la linea SS (Slave Select) alta per terminare la sessione di comunicazione.*

Il seguente diagramma a stati presenta quanto descritto utilizzando un formato in cui gli input e output nelle transizioni sono ordinati secondo la descrizione nella legenda.



***1.3** Diagramma a stati del modulo di elaborazione Trasferimento dati SPI*

STATO INIZIALE:

Descrizione: Nello stati iniziale l"interfaccia SPI è configurata

Transizioni:

Input: Il sistema rimane in attesa che venga impostato a 0 lo slave select

Output: lo slave viene selezionato, viene generato un impulso di clock e il contatore di bit settato a 0

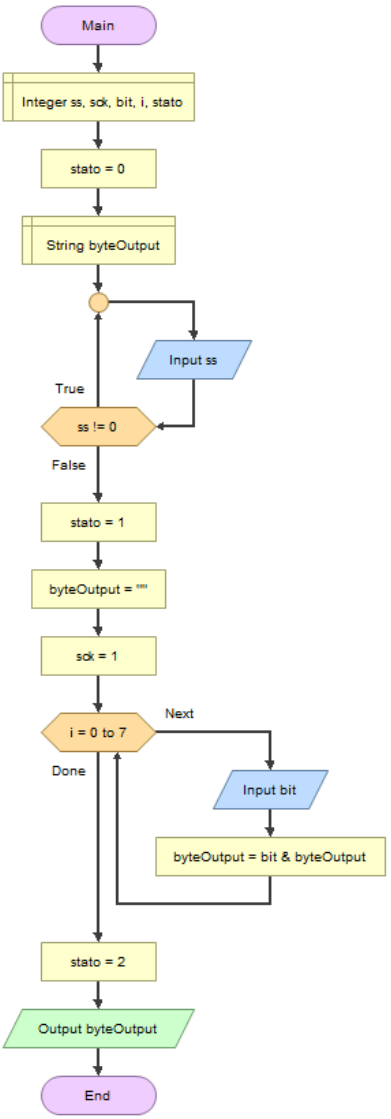
STATO S1: *Trasferimento Dati*

- *Descrizione: I dati vengono trasferiti dal master allo slave*
- *Input: lo slave viene selezionato, viene generato un impulso di clock e il contatore di bit settato a 0*

- *Output: vengono shiftati i bit dal master allo slave,terminata questa fase il byte è stato trasferito completamente nello slave*

STATO Finale:

Descrizione: Trasferimento completato, viene generato un interrupt al completamento del trasferimento e il sistema può tornare allo stato iniziale e procedere con l'invio di un nuovo byte.



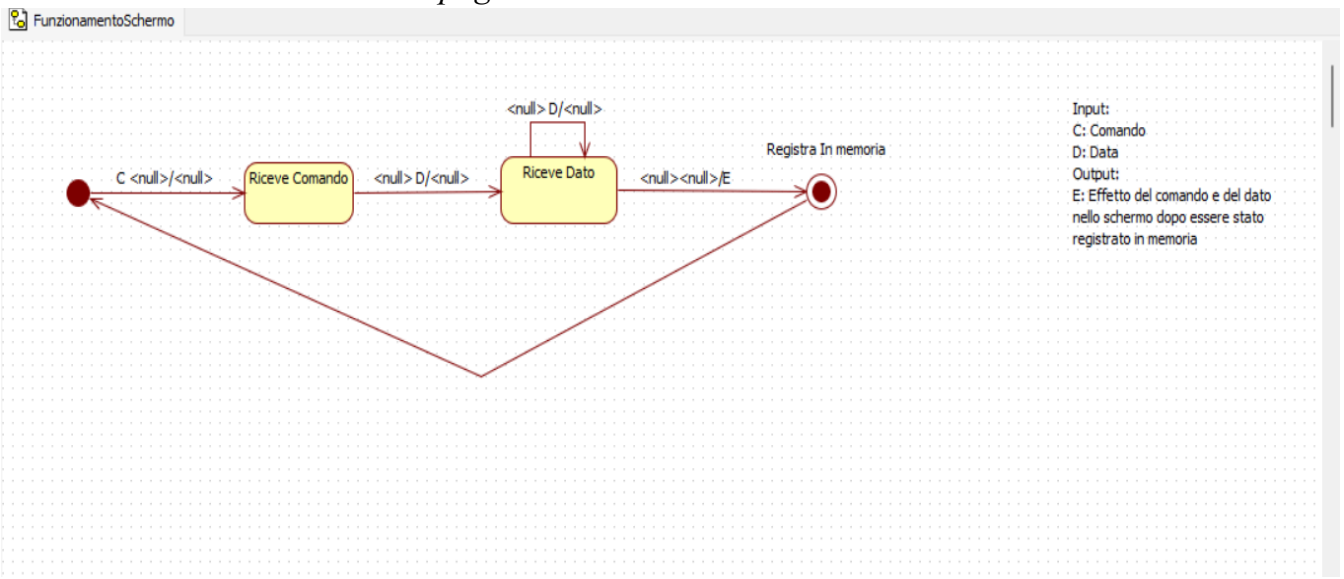
***1.4** Diagramma a stati del modulo di elaborazione Trasferimento dati SPI*

1.3.1 Modulo di elaborazione: Visualizzazione Pixel su schermo

Il modulo di elaborazione per il disegno dei pixel sullo schermo si occupa di gestire la visualizzazione grafica inviando le coordinate e i colori dei pixel al display LCD.
L'analisi di questo modulo avrà come riferimento lo schermo LCD ILI9341, dalla lettura del datasheet emerge che lo schermo separa l'invio di comandi dai dati.
Questo avviene settando a 1 la flag D/C (Data/Command) che indica se l'informazione in arrivo è un comando o un dato.

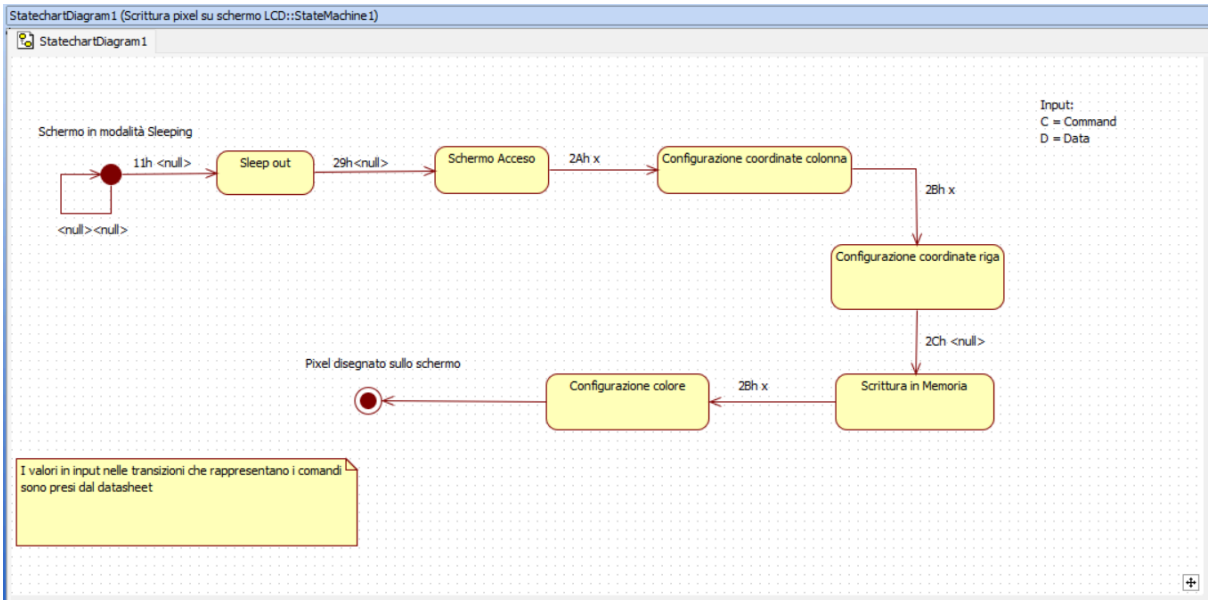
- ***Comandi:*** *Sono utilizzati per configurare e istruire il display*
- ***Dati:*** *Dopo aver inviato il comando si inviano i dati ad esso associati come le coordinate, il colore o i parametri di configurazione*

L'ILI9341 segue un formato di trasmissione in cui prima viene inviato il comando e poi uno o piu byte che rappresentano i dati di tale comando
La scrittura del pixel sullo schermo avviene modificando il contenuto della memoria interna dell'ILI9341 che rappresenta i pixel sullo schermo settando le coordinate tramite i comandi di colonna e pagina.



1.5 *Diagramma a stati modulo di elaborazione Visualizzazione Pixel su schermo funzionamento generale*

Tale diagramma mostra il funzionamento generale descritto precedentemente in cui dallo stato di ricezione del comando si transiziona allo stato di ricezione del dato e successivamente allo stato finale che rappresenta lo stato di registrazione di dato e comando nel sistema



***1.6** Diagramma a stati modulo di elaborazione Visualizzazione Pixel su schermo*

Lo scopo di tale diagramma è quello di rappresentare il funzionamento del modulo rispetto all'operazione di scrittura di un pixel, mettendo in evidenza il codice in esadecimale del comando corrispondente, tali codici sono presi dal datasheet.

2. Sintesi

2.1 Sintesi componente Invio dati tramite SPI

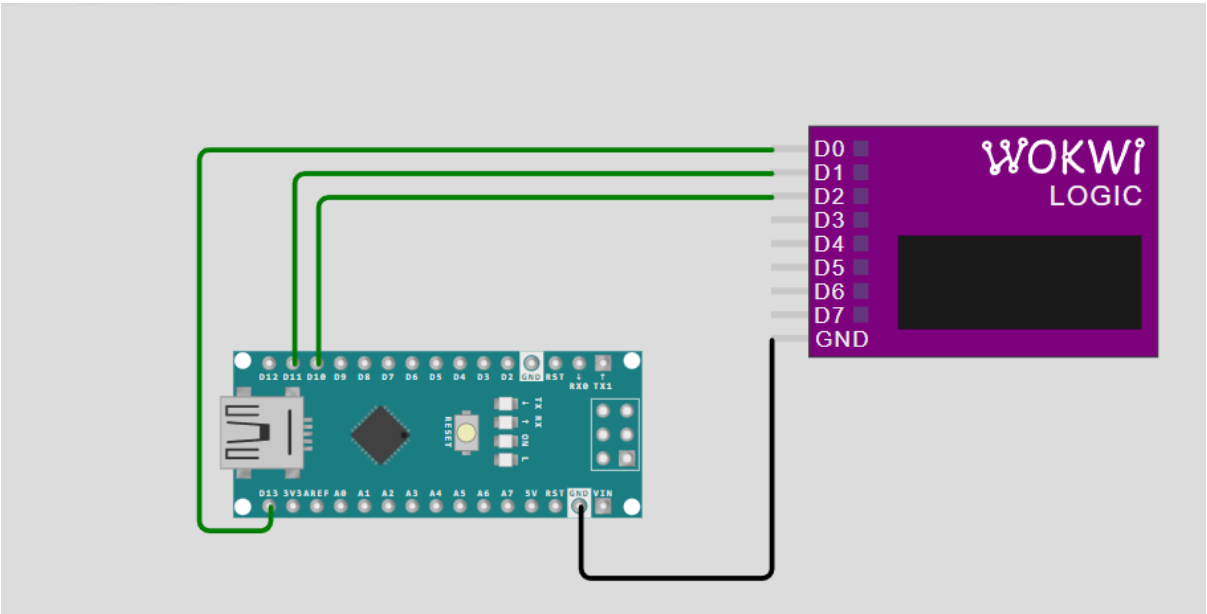
Il primo passaggio per realizzare tale modulo è configurare l'interfaccia SPI.

Si abilita l'interfaccia SPI, imposta come Master la mcu e si abilitano come output i pin della porta B corrispondenti per il MOSI, SS, SCK rispettivamente 3,2,5.

```
1   void setup() {
2   //Abilitazione SPI , configurazione mcu come Master, per ora non si abilità l'interrupt
3     SPCR |= (1<< 6) | (1 << 4);
4   //Configurazione MOSI, SCK, SS
5     DDRB |= (1 << DDB3) | (1<<DDB5) | (1 << DDB2);
6   }
```

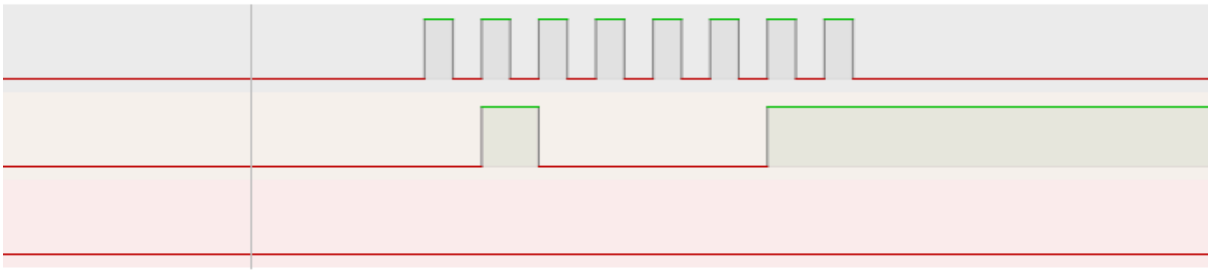
Dopo aver configurato l'interfaccia sarà possibile testarne il funzionamento inviando un byte. Attraverso l'utilizzo di un analizzatore logico sarà possibile verificare che i bit sono stati trasmessi correttamente, si configura il campionamento sul fronte di salita del clock, la polarità, si simula il funzionamento della linea SS manualmente portando la linea ad uno stato basso quando inizia la trasmissione e poi allo stato alto una volta terminata. Nel codice è stato anche inserito un delay di 1 ms e alcuni print ma solo ai fini di testing e verranno entrambi rimossi nei passaggi successivi e non saranno presenti nel codice finale.

```ino
char data[4] = {'C', 'I', 'A','O'};

void setup() {
    SPIMasterInit();
  Serial.begin(9600);
}

void SPIMasterInit(){
  //Configurazione MOSI, SCK, SS
  DDRB = (1 << DDB3) | (1<<DDB5) | (1 << DDB2);
  //Abilitazione SPI , configurazione mcu come Master,
  // divisore clock=16,campionamento fronte di salita ,idle clock=basso)
  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0) | (1<< SPR1) | (0 << CPHA) | (0 << CPOL);
}

void trasmettiByte(uint8_t data) {
  SPDR = data;
  while(!(SPSR & (1 << SPIF)));
  Serial.print("Valore trasmesso in decimale: ");
  Serial.println(data);
}

void loop() {
  for(int i = 0; i < 4; i++){
    PORTB &= !(1 << 2);
    trasmettiByte(data[i]);
    PORTB |= (1 << 2);
    delay(1);
  }
}
```

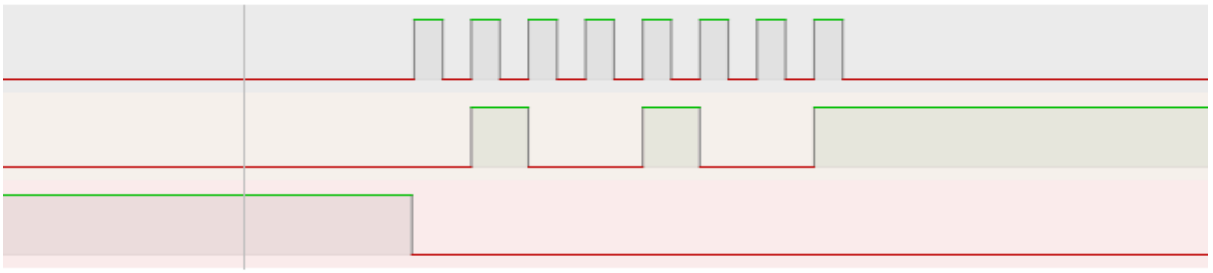


Come si può vedere direttamente da PulseView il primo byte trasmesso rappresenta il

carattere 'C' in binario 01000011, dall'immagine troviamo rispettivamente il clock, il dato inviato , e la linea SS



Cosi anche per il carattere 'I' in binario 01001001



A questo punto si può quindi implementare completamente il metodo SPI_MasterOpen(uint8_t clkdiv) necessario per abilitare l'interfaccia, impostare come master e settare una frequenza di clock custom attraverso un divisore a seconda delle necessità in questo modo:

```
57  void SPI_MasterOpen(uint8_t clkdiv)(){
58    //Configurazione MOSI, SCK, SS
59    DDRB = (1 << DDB3) | (1<<DDB5) | (1 << DDB2);
60    //Abilitazione SPI , configurazione mcu come Master,
61    // campionamento fronte di salita ,idle clock=basso)
62    SPCR = (1 << SPE) | (1 << MSTR) | (0 << CPHA) | (0 << CPOL);
63    // Gestione del divisore di clock
64    if (clkdiv == 2 || clkdiv == 8 || clkdiv == 32) {
65      SPSR |= (1 << SPI2X);  // Imposta SPI2X per divisori con doppia velocità
66    } else {
67      SPSR &= ~(1 << SPI2X);  // Disabilita SPI2X per gli altri divisori
68    }
69    // Gestione dei bit SPR1 e SPR0 per i restatni divisori
70    switch (clkdiv) {
71      case 2:  // 2 = SPI2X = 1, SPR1 = 0, SPR0 = 0
72      case 4:  // 4 = SPI2X = 0, SPR1 = 0, SPR0 = 0
73        SPCR &= ~((1 << SPR1) | (1 << SPR0));  // SPR1 = 0, SPR0 = 0
74        break;
75      case 8:  // 8 = SPI2X = 1, SPR1 = 0, SPR0 = 1
76      case 16: // 16 = SPI2X = 0, SPR1 = 0, SPR0 = 1
77        SPCR = (SPCR & ~(1 << SPR1)) | (1 << SPR0);  // SPR1 = 0, SPR0 = 1
78        break;
79      case 32: // 32 = SPI2X = 1, SPR1 = 1, SPR0 = 0
80      case 64: // 64 = SPI2X = 0, SPR1 = 1, SPR0 = 0
81        SPCR = (SPCR & ~(1 << SPR0)) | (1 << SPR1);  // SPR1 = 1, SPR0 = 0
82        break;
83      case 128: // 128 = SPI2X = 0, SPR1 = 1, SPR0 = 1
84        SPCR |= (1 << SPR1) | (1 << SPR0);  // SPR1 = 1, SPR0 = 1
85        break;
86      default:
87        // Default a 4 se clkdiv non è valido
88        SPCR &= ~((1 << SPR1) | (1 << SPR0));  // SPR1 = 0, SPR0 = 0
89        SPSR &= ~(1 << SPI2X);  // Assicura che SPI2X sia disabilitato
90        break;
```

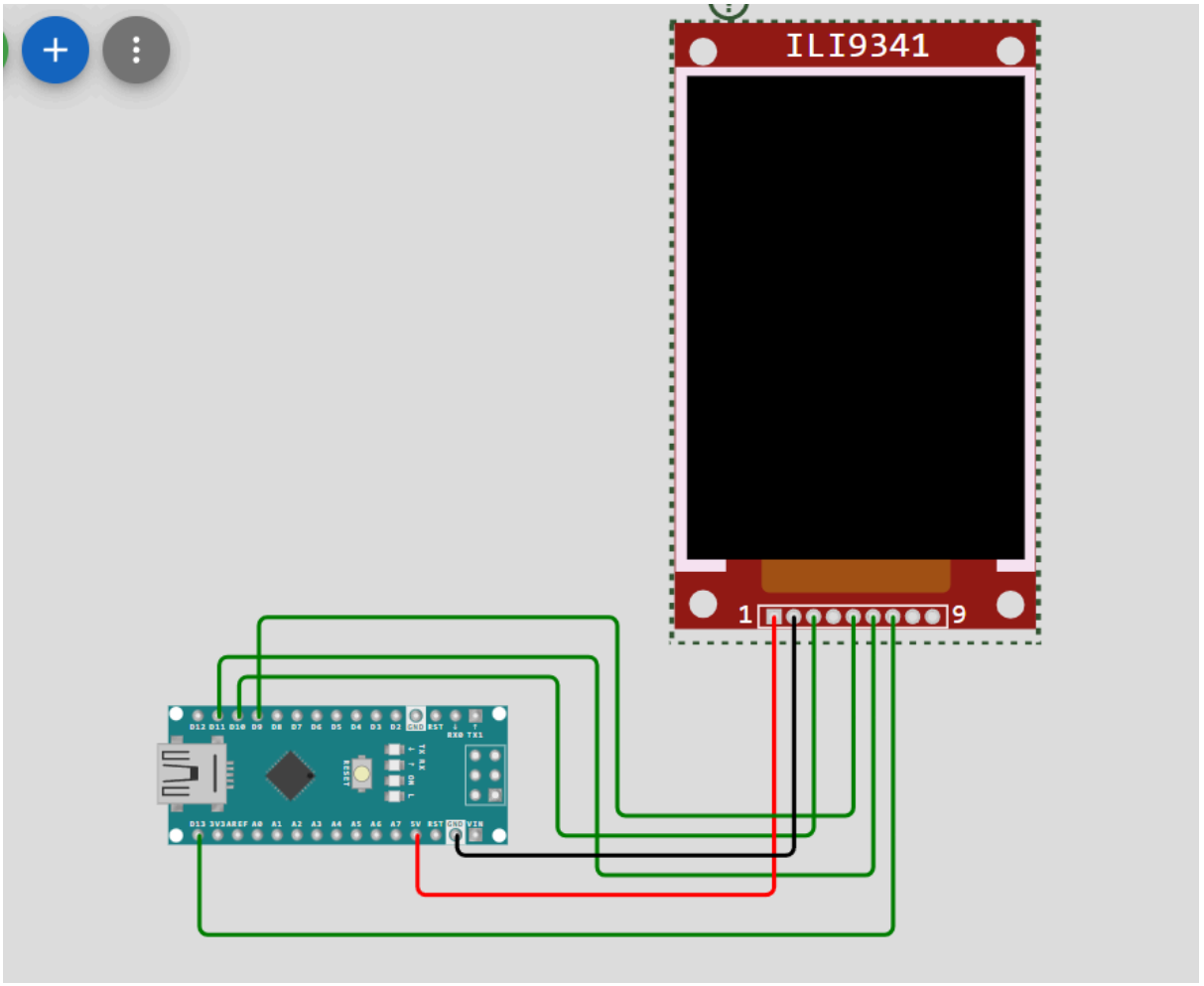
Si verifica se il divisore del clock è 2,8 o 32 e in tal caso il bit SPI2X viene settato a 1 e poi si settano per i rispettivi casi i bit SPR1 e SPR0.

Inoltre fino ad ora è stato utilizzato un meccanismo di polling nella trasmissione dei dati in quanto la verifica del completamento della trasmissione avveniva attraverso un ciclo while come mostrato nel metodo trasmettiByte, tale meccanismo di controllo verrà rimpiazzato dall'utilizzo di un interrupt.

2.2 Sintesi componente Visualizzazione Pixel su schermo

La realizzazione di questo componente coinvolge l'utilizzo dello schermo ILI 3941, come già visto durante la fase di analisi del componente sarà necessario gestire l'invio del comando e del dato associato(piu dati possono essere associati al componente).

Il primo passo sarà quello di collegare la mcu con lo schermo come mostrato in figura e definire i pin.




```
1   // Definizioni Pin
2   #define PINB_COMMAND_DATA 1
3   #define PINB_MOSI 3
4   #define PINB_SCK 5
5   #define PINB_SS 2
6   //
```

Si definiscono i comandi per l'inizializzazione del display

```
#define ILI9341_PWCTR1 0xC0 ///< Power Control 1
#define ILI9341_PWCTR2 0xC1 ///< Power Control 2
#define ILI9341_VMCTR1 0xC5 ///< VCOM Control 1
#define ILI9341_VMCTR2 0xC7 ///< VCOM Control 2
#define ILI9341_MADCTL 0x36   ///< Memory Access Control
#define ILI9341_VSCRSADD 0x37 ///< Vertical Scrolling Start Address
#define ILI9341_PIXFMT 0x3A   ///< COLMOD: Pixel Format Set
#define ILI9341_FRMCTR1 0xB1 ///< Frame Rate Control (In Normal Mode/Full Colors)
#define ILI9341_DFUNCTR 0xB6 ///< Display Function Control
#define ILI9341_GAMMASET 0x26 ///< Gamma Set
#define ILI9341_GMCTRP1 0xE0 ///< Positive Gamma Correction
#define ILI9341_GMCTRN1 0xE1 ///< Negative Gamma Correction
#define ILI9341_SLPOUT 0x11 ///< Sleep Out
#define ILI9341_DISPON 0x29   ///< Display ON

//Vettore dei comandi per l'inizializzazione dello schermo
static const uint8_t PROGMEM initcmd[] = {
  0xEF, 3, 0x03, 0x80, 0x02,
  0xCF, 3, 0x00, 0xC1, 0x30,
  0xED, 4, 0x64, 0x03, 0x12, 0x81,
  0xE8, 3, 0x85, 0x00, 0x78,
  0xCB, 5, 0x39, 0x2C, 0x00, 0x34, 0x02,
  0xF7, 1, 0x20,
  0xEA, 2, 0x00, 0x00,
  ILI9341_PWCTR1  , 1, 0x23,             // Power control VRH[5:0]
  ILI9341_PWCTR2  , 1, 0x10,             // Power control SAP[2:0];BT[3:0]
  ILI9341_VMCTR1  , 2, 0x3e, 0x28,       // VCM control
  ILI9341_VMCTR2  , 1, 0x86,             // VCM control2
  ILI9341_MADCTL  , 1, 0x48,             // Memory Access Control
  ILI9341_VSCRSADD, 1, 0x00,             // Vertical scroll zero
  ILI9341_PIXFMT  , 1, 0x55,
  ILI9341_FRMCTR1 , 2, 0x00, 0x18,
  ILI9341_DFUNCTR , 3, 0x08, 0x82, 0x27, // Display Function Control
  0xF2, 1, 0x00,                         // 3Gamma Function Disable
  ILI9341_GAMMASET , 1, 0x01,             // Gamma curve selected
  ILI9341_GMCTRP1 , 15, 0x0F, 0x31, 0x2B, 0x0C, 0x0E, 0x08, // Set Gamma
    0x4E, 0xF1, 0x37, 0x07, 0x10, 0x03, 0x0E, 0x09, 0x00,
  ILI9341_GMCTRN1 , 15, 0x00, 0x0E, 0x14, 0x03, 0x11, 0x07, // Set Gamma
    0x31, 0xC1, 0x48, 0x08, 0x0F, 0x0C, 0x31, 0x36, 0x0F,
  ILI9341_SLPOUT  , 0x80,                // Exit Sleep
  ILI9341_DISPON  , 0x80,                // Display on
  0x00                                   // End of list
};
// Termine definizioni
```

Successivamente sarà necessario costruire la funzione per l'invio dei comandi e la funzione per l'invio dei dati, per l'invio del comando sarà necessario attivare la modalità comando e portare a 0 il pin della linea SS.

```
/*Metodo per inviare un comando*/
void ILI_sendCmd(uint8_t cmd) {
  PORTB &= ~((1 << PINB_COMMAND_DATA) | (1 << PINB_SS ));  // Attivo la linea comando, e
  masterTransmit(cmd);
  PORTB |= (1 << PINB_SS);
}
/*Metodo per inviare un dato associato al comando*/
void ILI_sendData(uint8_t data) {
  PORTB |= (1 << PINB_COMMAND_DATA);   // Modalità dati
  PORTB &= (~(1 << PINB_SS));
  masterTransmit(data);
  PORTB |= (1 << PINB_SS);
}
```

Visto che il vettore necessario all'inizializzazione dello schermo contiene molti comandi con multipli dati associati sarà necessario creare una funzione che sarà in grado di inviare un comando con piu data associati.

```
/* Tale metodo serve per inviare un comando che ha piu dati ad esso associati*/
void sendCommand(uint8_t cmd, uint8_t *dati, uint8_t numDati){ // 1 dato = byte
 if ((1 << PINB_SS) & PINB){      // Se il pin è alto
   PORTB |= (1 << 2);             // Seleziona lo slave
  }
  PORTB |= (1 << 1);             // Modalità comando
  ILI_sendCmd(cmd);             //Invia il comando
  for (int i = 0; i < numDati; i++) {
      ILI_sendData(*dati); // Send the data bytes
      dati++;
    }
```

L'inizializzazione dello schermo avviene tramite l'esecuzione del vettore dei comandi precedentemente definito initcmd[] come in figura.

```
/**Inizializzazione schermo**/
void ILI9341_Init() {
  // Invia la sequenza di inizializzazione
  ILI_sendCmd(0x01); // Reset dello schermo poichè non fornito nella simulazione da wokwi
  delay(200);
  uint8_t cmd, x, numArgs;          // Comando, ritardo(settimo bit), numero argomenti comando
  const uint8_t *addr = initcmd;    // Puntatore ai comandi per l'inizializzazione dello schermo
  uint8_t *dataAddr;                //Puntatore temporaneo per i dati associati al comando
  while ((cmd = pgm_read_byte(addr++)) > 0) { // fino a quando ci sono dei comandi
    //Serial.print("Comando: ");
    //Serial.println(cmd,HEX);

    dataAddr = addr;
    x = pgm_read_byte(addr++);
    numArgs = x & 0x7F;                 // estrai il numero di argomenti associati al comando
    sendCommand(cmd, dataAddr+1, numArgs);
    addr += numArgs;
    if (x & 0x80)
      delay(150);
  }

}
```

Dopo l'invio del primo comando che è quello di reset sarà necessario un delay di 200 ms.

Per inviare un dato da 16 bit.

```
/**Metodo per trasmettere un dato da 16 bit allo schemro**/
void ILI_sendData16(uint16_t data){
  PORTB |= (1 << PINB_COMMAND_DATA);  // Modalità dati
  PORTB &= (~( 1 << PINB_SS));        // Linea SS bassa seleziono lo slave
  ILI_sendData(data >> 8);             // Invio gli 8 bit piu significativi
  ILI_sendData(data & 0xFF);             // Invio gli 8 bit meno significativi
}

```

Eseguita la configurazione e i metodi per trasmettere comandi si possono quindi creare i metodi per scrivere un pixel sullo schermo. Va quindi impostata la finestra di scrittura selezionando la colonna e la riga/pagina.

```
// Funzione per impostare una finestra di disegno (area) sul display
void ILI_setAddrWindow(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1) {
  // Imposta la finestra Y (colonne)
  ILI_sendCmd(ILI9341_CASET);     // Column Address Set
  ILI_sendData16(x0);             // Inizio colonna
  ILI_sendData16(x1);             // Fine colonna
  // Imposta la finestra X (righe)
  ILI_sendCmd(ILI9341_PASET);     // Page Address Set
  ILI_sendData16(y0);             // Inizio riga
  ILI_sendData16(y1);             // Fine riga

}
```

```
// Funzione per disegnare un pixel in una posizione specifica
void ILI_drawPixel(uint16_t x, uint16_t y, uint16_t color) {
  // Imposta la finestra di disegno come un singolo pixel
  ILI_setAddrWindow(x, y, x+1, y+1);
  // Invia il comando per iniziare a scrivere in memoria
  ILI_sendCmd(ILI9341_RAMWR);
  // Invia il colore del pixel
  ILI_sendData16(color);
}

// Funzione per disegnare un pixel al centro dello schermo
void ILI_drawPixelAtCenter() {
  uint16_t centerY = LUNGHEZZA / 2;
  uint16_t centerX = LARGHEZZA / 2;

  // Disegna un pixel rosso al centro dello schermo
  ILI_drawPixel(centerX, centerY, RED);
}
```

3. Sistema complessivo

Il link al progetto è il seguente:
https://wokwi.com/projects/408767389861249025
Saranno comunque allegati in uno zip i diagrammi e il codice.

3.1 Manuale utilizzo

Nel codice sono stati aggiunti un metodo per disegnare un pixel al centro dello schermo e un metodo per disegnare un pixel in qualsiasi punto.

Tuttavia risultano esserci problemi in quanto in alcuni casi il pixel non viene disegnato o viene disegnato di un altro colore rispetto a quello fornito e solo dopo diversi riavvii il codice funziona.

3.2 Considerazioni finali

Per le problematiche sopra citate e per il tempo speso cercando di risolvere tali problemi non ho avuto modo di eliminare il polling e rimpiazzarlo con la gestione tramite interrupt.