Bus di Comunicazione: SPI e I2C

Luigi Coppolino, Giovanni Mazzeo

i Bus di Comunicazione 1/3

- Un Bus è un canale di comunicazione che permette a periferiche e componenti di I/O di comunicare tra loro scambiandosi informazioni attraverso la trasmissione di segnali.
- Possono essere di tipo **seriale** o **parallelo** a seconda del numero di linee utilizzate per la trasmissione dei dati (una o molteplici)
- Affinché due o più dispositivi possano "parlare tra di loro e capirsi" si fa uso di protocolli di bus
- Tra i più importanti si annoverano: AMBA, SPI, I2C, USB, CAN

i Bus di Comunicazione 2/3

- Solitamente i bus comunicano in modalità Master/Slave. Indipendentemente dal tipo di protocollo, è possibile delineare un modo di funzionamento generico:
  - Per effettuare una comunicazione sul bus:
    - un dispositivo **Master** prende il controllo del bus
    - invia una richiesta ad un secondo dispositivo (uno **Slave**)
    - terminata la comunicazione il bus viene liberato per eseguirne una nuova.
- Il ruolo di un dispositivo (se master o slave) può cambiare nel tempo; un dispositivo può comportarsi da master o da slave in contesti differenti. Lo standard che definisce il bus deve fornire le regole per gestire tali condizioni o vietare conflitti.

i Bus di Comunicazione 3/3

- Esistono due diversi meccanismi di temporizzazione dei segnali:
  - **Protocollo asincrono**: tutta la temporizzazione della comunicazione è gestita dal protocollo stesso attraverso lo scambio dei messaggi.
  - **Protocollo sincrono**: è previsto un segnale di clock che permette di gestire la temporizzazione delle comunicazioni.

Protocollo Asincrono – 1/2

- Un esempio di protocollo asincrono particolarmente semplice è il cosiddetto **handshaking a due variabili.** In un ciclo di lettura:

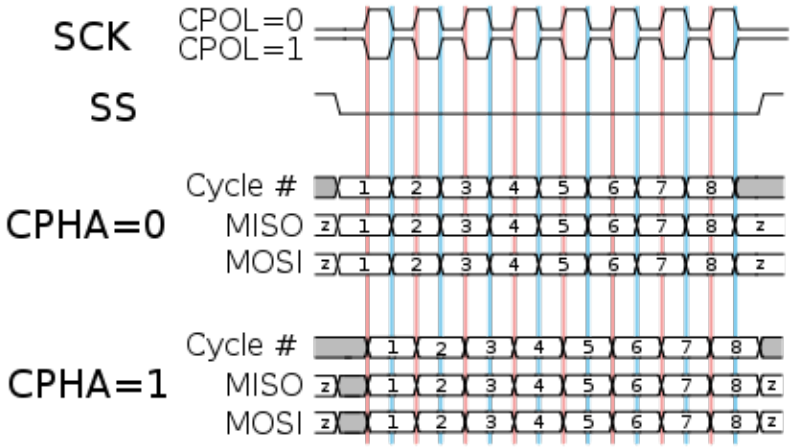
  - **M=S=0 (riposo)** Il Master in qualsiasi momento, dopo aver verificato che S=0, può decidere di iniziare la comunicazione. Lo Slave controlla continuamente il valore di M in attesa della transizione a 1.
  - **M=1, S=0 (inizio lettura)** Il Master, dopo aver preparato l'indirizzo, invia l'ordine di inizio lettura; attende dunque la risposta dello Slave, ossia il passaggio di S al valore 1. Lo Slave riconosce l'ordine e da inizio all'esecuzione del comando; lo Slave non risponde (ossia mantiene S=0) fin quando non ha terminato l'esecuzione del comando.

Protocollo Asincrono – 2/2

- **M=1, S=1 (fine lettura)** Il Master legge dal bus il valore dello Slave; mantiene M=1 fin quando non ha "consumato" completamente i dati. Lo Slave ha risposto al Master dopo aver fornito il risultato della lettura sul bus; mantiene questi valori e l'indicazione S=1 fino a quando non si accorge del passaggio di M a 0.
- **M=0, S=1 (conferma lettura)** Il Master ha dato conferma allo Slave dell'avvenuta lettura, ponendo M=0; in questa condizione il Master può procedere con altre attività, ma non può mandare un nuovo comando allo Slave. Lo Slave verificando che M=0 si accorge che la lettura é terminata, e stacca la sua connessione in uscita sul bus; solo dopo aver staccato la propria connessione passa allo stato successivo.
- **M=0, S=0 (bus libero)** Lo Slave segnala di aver rilasciato il bus ponendo S=0, quindi si riporta nello stato iniziale di "attesa di ordini".

Protocollo Sincrono

- Come osservato, il protocollo Asincrono è particolarmente complesso e richiede un rilevante quantità di scambi di messagi tra Master e Slave per la sola sincronizzazione
- Per questo motivo, i bus Sincroni sono preferiti nella maggior parte dei contesti di utilizzo.
- L'idea é quella di sincronizzare le attività del Master e dello Slave sulle variazioni periodiche di un'unica variabile di riferimento: il Clock.

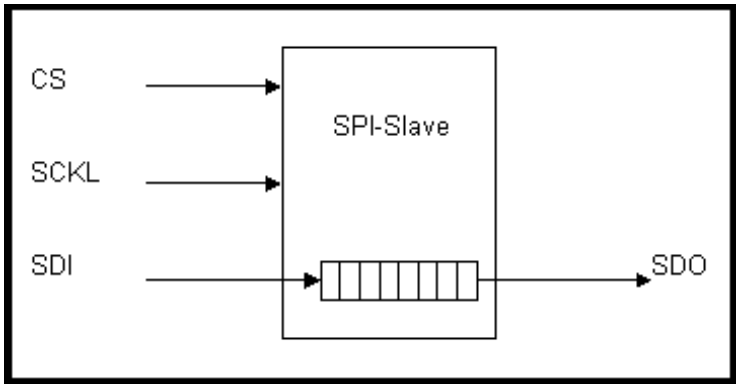


SPI

- Il **Serial Peripheral Interface (SPI)** è un protocollo di comunicazione sviluppato da Motorola
- E' un protocollo **Sincrono e Seriale**
- E' solitamente utilizzato per comunicazioni seriali tra un processore e le periferiche
- Funziona in configurazione **Master/Slave**. Il Master potrebbe essere il microcontrollore, e lo Slave la periferica.
- **Vantaggi**: E' un bus che ha un protocollo di comunicazione molto semplice e dunque un overhead sul processore basso; è tra i più veloci nella sua categoria
- **Svantaggi**: Più crescono le periferiche collegate e più aumentano le linee del bus -> maggiore area di occupazione delle schede

SPI: Operazioni - 1/2

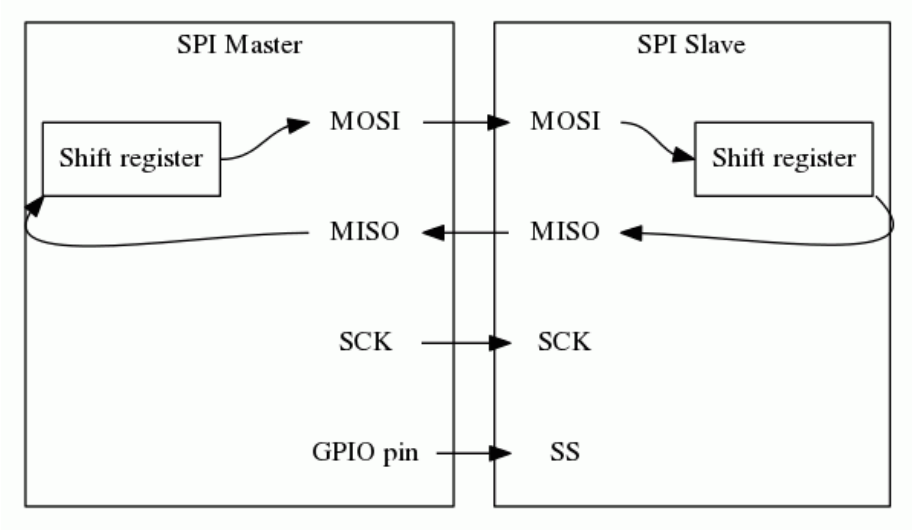
- Il bus SPI fa uso di 4 linee:
  - **Serial Clock (SCLK)**
  - **Chip Select (CS)**
  - **Serial Data In (SDI) o Master input Slave Output (MISO)**
  - **Serial Data Out (SDO) o Master Output Slave Input (MOSI)**

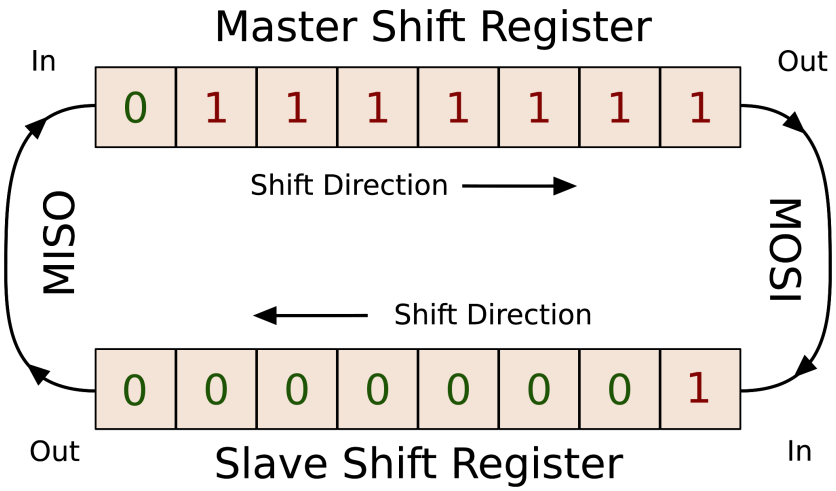


- Il bus permette che solo un Master possa esserci durante una comunicazione
- Il numero di Slaves (o periferiche) dipende dal numero di linee Chip Select (CS) in uscita dal Master
- La frequenza di funzionamento di SPI è solitamente nell'intorno di 1-2 MHz

SPI: Operazioni - 2/2

- Ogni periferica (sia Master che Slave) presentano al loro interno dei **Shift Registers** collegati alle linee MOSI (SDO) e MISO (SDI)
- Ad ogni colpo di clock l'ultimo bit viene inviato dallo shift register del master a quello dello slave. Dopo 8 colpi di clock, il dato sarà trasmesso completamente.

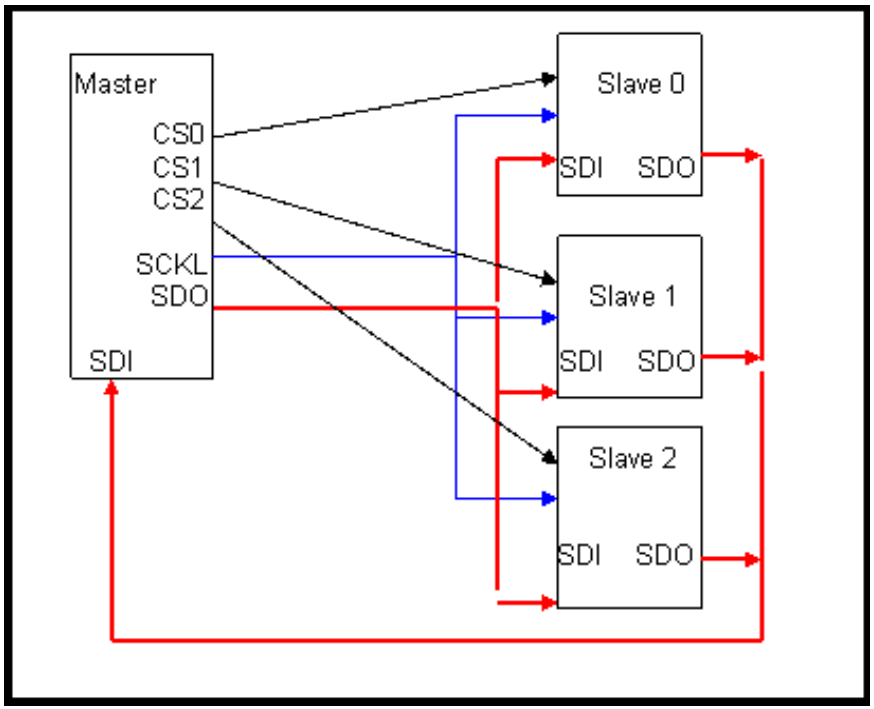




Master-Slave Setup – 1/2

- Un Master per iniziare una comunicazione attiva il clock sulla linea SCKL e seleziona lo slave col quale vuole parlare attraverto l'opportuna linea CS
- Osservare che le linee SDO sono collegate alle SDI. Ciò vale per entrambe le connessioni Master-Slave e Slave-Master

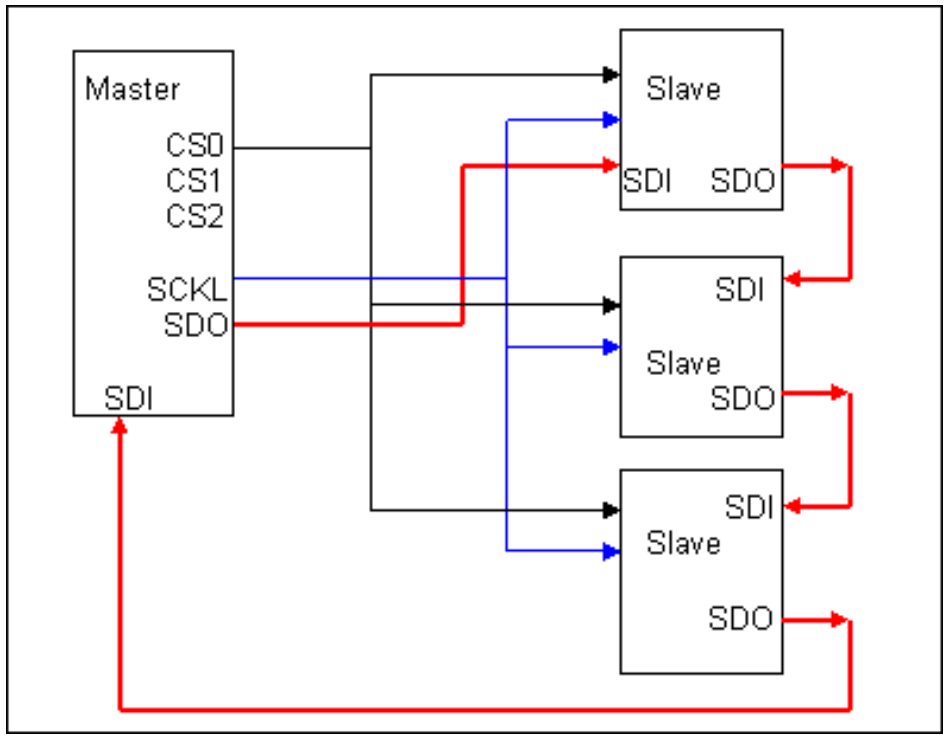
Multiple Independent Slave Configuration



Master-Slave Setup – 2/2

- In questo caso gli Slave sono collegati in cascata tra di loro
- L'uscita di uno andrà in ingresso dell'altro
- In questo caso, gli Slave sono trattati come un solo Slave e connessi alla stessa linea CS

Multiple Slave Cascaded



I2C

- Il Bus **Inter Integrated Circuit** (**I2C**) (si legge ”I square si”) è un bus di comunicazione **seriale**, **sincrono, bifilare** basato su pattern Master/Slave
- E’ stato sviluppato da Philips nel 1982 per impiegarlo all’interno di televisori
- E’ spesso utilizzato nei microcontrollori per connettere *low-speed devices* come, EEPROMs, A/D and D/A converters, I/O interfaces ed altre periferiche simili dei sistemi embedded.
- I2C è un bus molto popolare per la sua bassa occupazione di area: ci possono essere uno o più master che controllano un numero illimitato di dispositivi di I/O con soli due fili

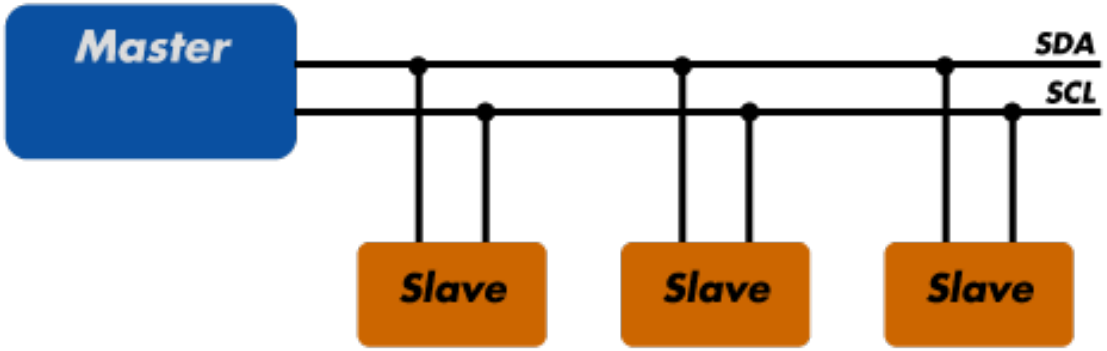
I2C

- I2C, a differenza di SPI, è un protocollo multi-master
- Permette l'utilizzo del protocollo in tre modalità aventi data rate diversi: 100 kbps (standard mode), 400 kbps (fast mode) and 3.4 Mbps (high-speed mode)
- Il bus è composto da 2 sole linee (**SDA** per i dati ed **SCL** per il clock)
- Ogni slave device è caratterizza da un indirizzo di 7 bit che lo identifica in modo univoco

I2C: Operazioni

- Il *Master* invia un segnale di START che informerà tutti gli *Slave* che devono ascoltare sulla linea dati
- Il master dunque invia l ADDRESS dello slave col quale vuole parlare ed un *flag* che indica se vuole fare un'operazione di Read o Write
- Lo Slave con quell'inidirizzo risponderà con un ACK
- La comunicazione quindi avrà luogo tra il master e lo slave. Entrambi possono ricevere o trasmettere dati a seconda di come è stata impostata la comunicazione precedenemtenete (se il flag è read o write)
- Ad ogni colpo di clock un bit sarà inviato allo slave
- Quando la comunicazione è completata il master invia uno STOP che indica la terminazione.

I2C


Master
SDA
SCL
Slave
Slave
Slave


START
READ/
WRITE
ACK
ACK
Slave Address
Data
Data
STOP

I2C

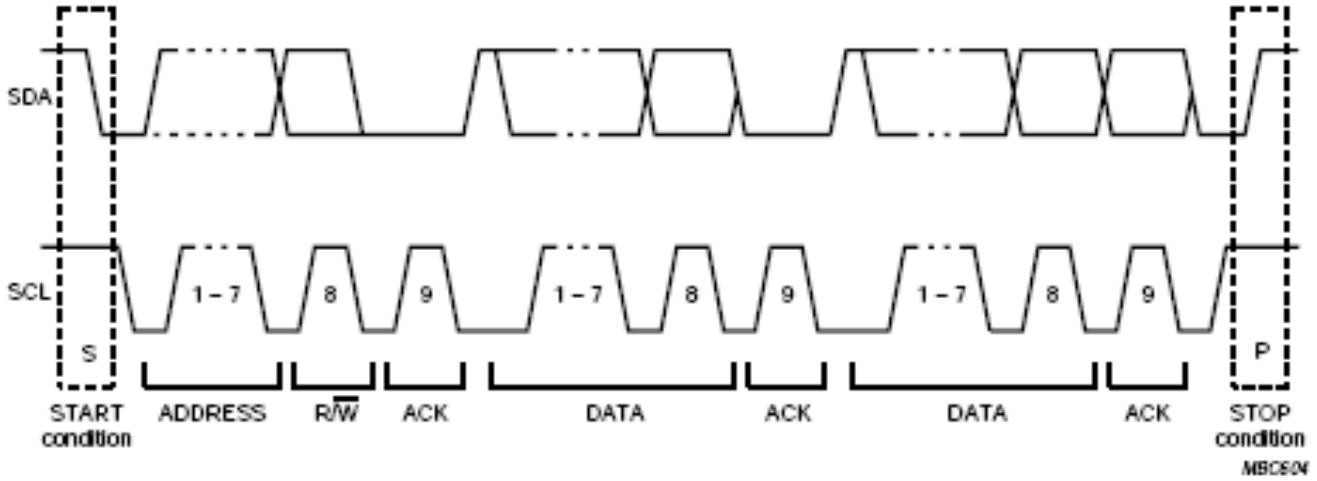
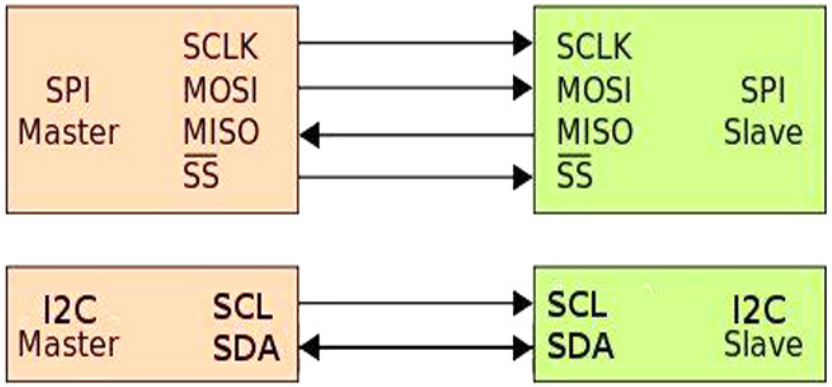


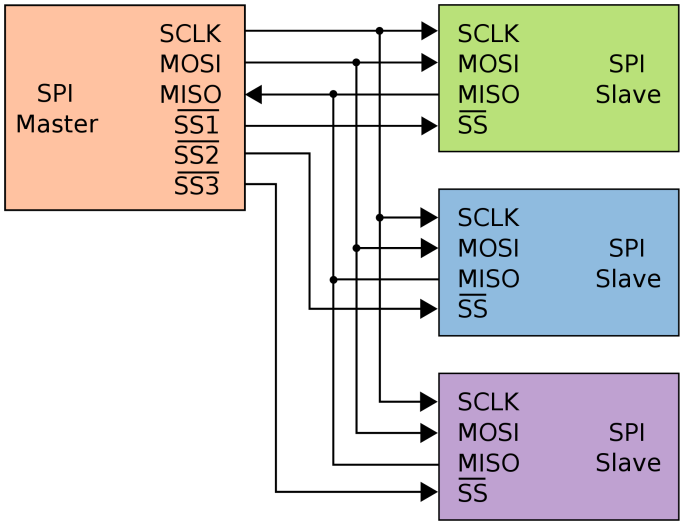
Fig.10 A complete data transfer.

SPI vs I2C

- I due bus possono essere comparati sulla base di:
  - **Topologia**
    - I2C richiede solo 2 lineed, mentre SPI almeno 4 che crescono al variare del numero di slave. Ciò ha ovviamente implicazioni sull'occupazione d'area dei chip
    - L'unico svantaggio di I2C è che ha uno spazio di indirizzi limitato (7 bit)
  - **Throughput / Speed:**
    - In caso di trasferimenti ad alta velocità, SPI è migliore di I2C. SPI è full-duplex mentre I2C non lo è. SPI non presenta limiti di velocità. Alcune implementazioni toccano i 10Mbps. I2C è invece limitato.
  - **Ease-of-Implementation:**
    - SPI è molto più semplice rispetto a I2C da implementare

SPI vs I2C

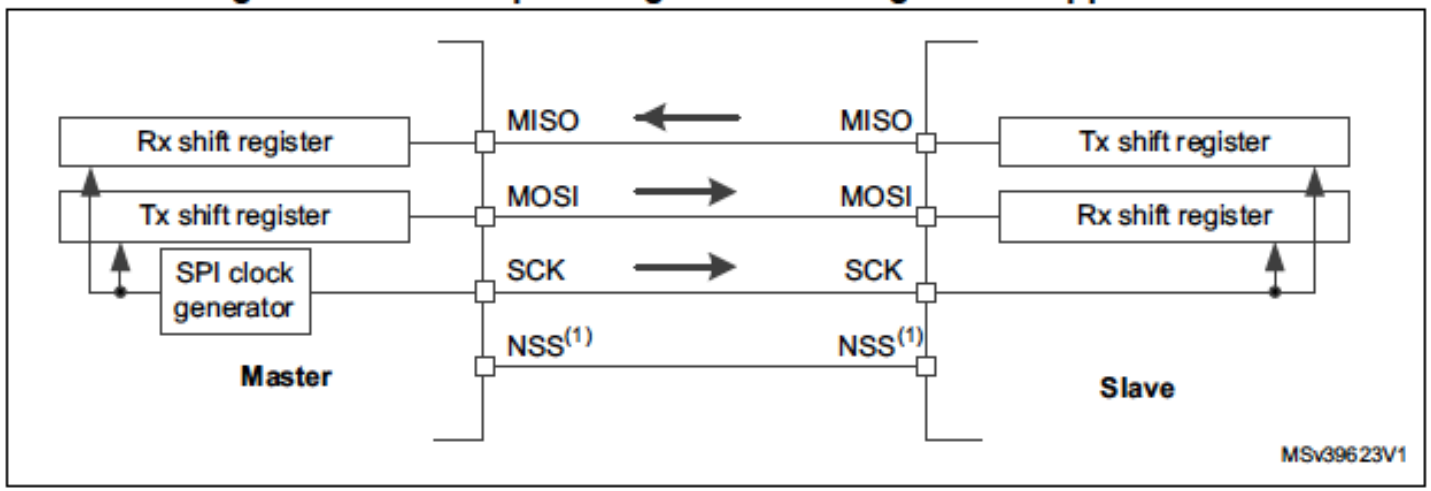

SPI
Master
SCLK
MOSI
MISO
SS
SCLK
MOSI
MISO
SS
SPI
Slave
I2C
Master
SCL
SDA
SCL
SDA
I2C
Slave


SPI
Master
SCLK
MOSI
MISO
SS1
SS2
SS3
SCLK
MOSI
MISO
SS
SPI
Slave
SCLK
MOSI
MISO
SS
SPI
Slave
SCLK
MOSI
MISO
SS
SPI
Slave

SPI sulla STM32F3 - 1

- La STM32F3 presenta 4 interfacce SPI utilizzabili dall'utente
- Tali interfacce è possibile configurarle in tre modi:
  - Full-Duplex (default): *"In this configuration, the shift registers of the master and slave are linked using two unidirectional lines between the MOSI and the MISO pins"*

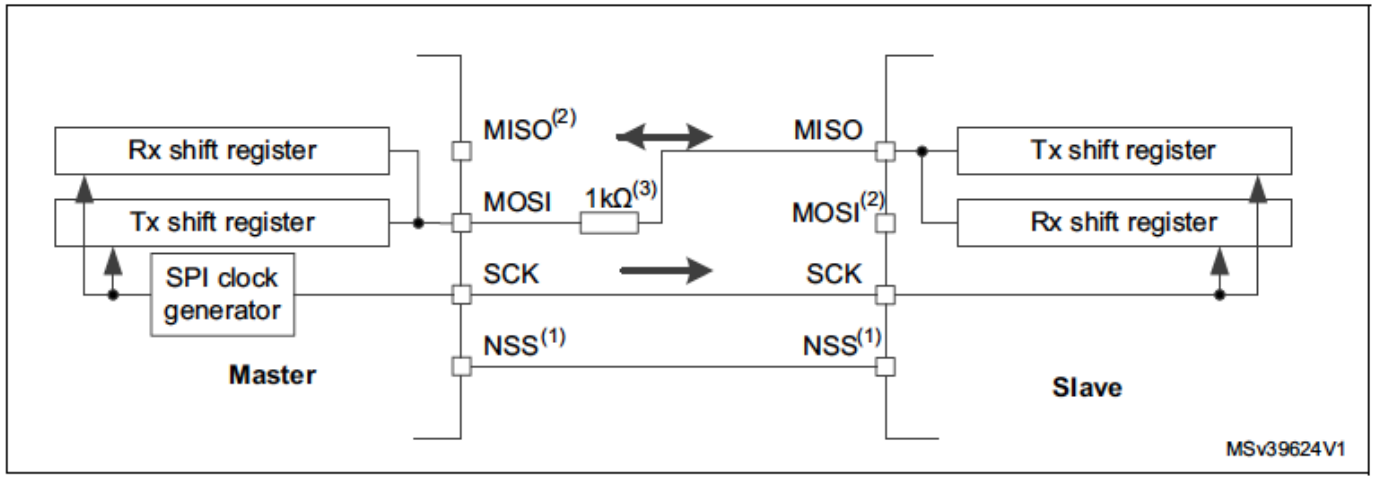
Figure 349. Full-duplex single master/ single slave application



SPI sulla STM32F3 - 2

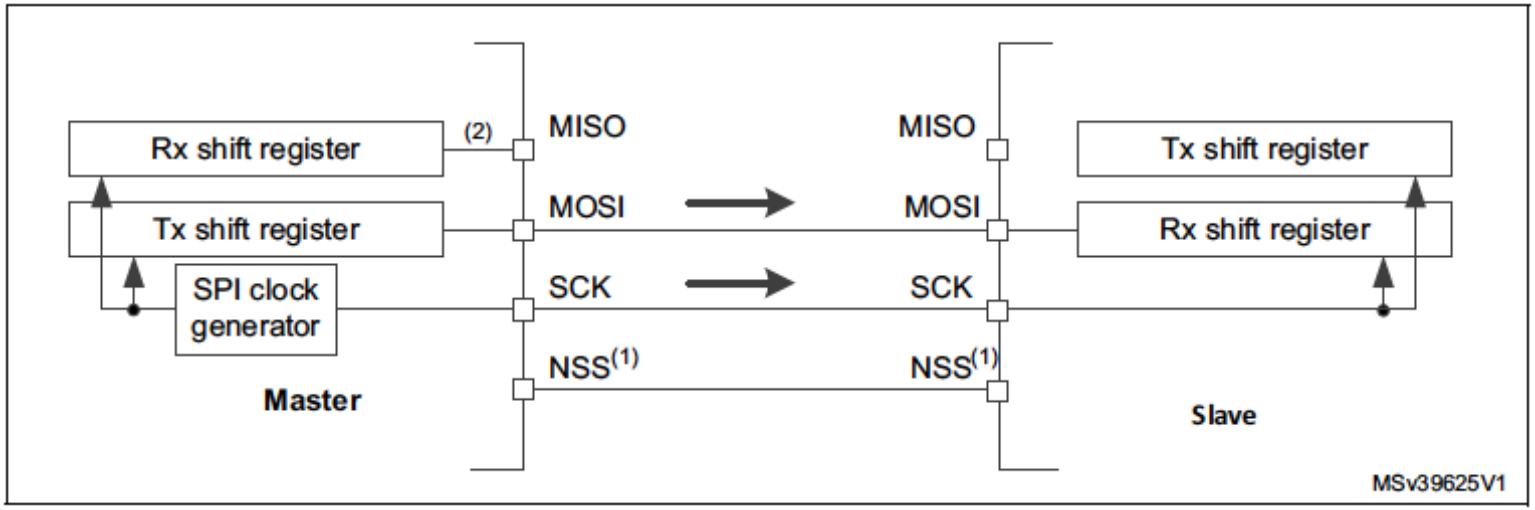
- Half-Duplex: *" In this configuration, one single cross connection line is used to link the shift registers of the master and slave together"*

Figure 350. Half-duplex single master/ single slave application



SPI sulla STM32F3 - 3

- Simplex: *" In this configuration, one single cross connection line is used to link the shift registers of the master and slave together"*



Modello di Programmazione SPI

- Come per qualsiasi altra periferica già vista, per utilizzare il bus SPI dovremo innanzitutto abilitare il clock su quella periferica ed in seguito definire:
  - Quale dei 4 SPI utilizzeremo: *SpiHandle.Instance = SPI2;*
  - Il modo di comunicazione (Full-Duplex, Half-…)
    - *SpiHandle.Init.Direction = SPI_DIRECTION_2LINES;*
  - La modalità di funzionameno (Master/Slave):
    - *SpiHandle.Init.Mode = SPI_MODE_SLAVE;*
  - La polarità e la fase del Clock:
    - *SpiHandle.Init.CLKPhase = SPI_PHASE_1EDGE;*
    - *SpiHandle.Init.CLKPolarity = SPI_POLARITY_HIGH;*
  - La dimensione dei dati:
    - *SpiHandle.Init.DataSize = SPI_DATASIZE_8BIT;*
  - Se controllare il NSS (Lo slave select) via Software
    - *SpiHandle.Init.NSS = SPI_NSS_SOFT;*

Esercizio: Comunicazione tra due schede STM32F3 attraverso SPI

- Realizzeremo ora un esercizio in cui due schede STM32F3 “parleranno” tra loro attraverso il bus SPI. L’utente dovrà premere lo “User Button” per avviare la trasmissione
- Collegheremo fisicamente le due schede con gli opportuni pin GPIO e realizzeremo il driver che permette loro di comunicare
  - Passo 1 – Decidere l’interfaccia SPI del microcontrollore da utilizzare (SPI1, SPI2, SPI3, o SPI4)
  - Passo 2 – Vedere sullo ”User Manual” quali sono i pin GPIO collegati alle linee dello SPIx selezionato (CLK, SDI, SDO, SS)
  - Passo 3 – Inizializzare dunque le linee di GPIO specifiche per lo SPIx in Alternate Functions. Ciò servirà ad abilitare i pin per la comunicazione tra le due schede.
  - Passo 4 – Inizializzare l’interfaccia SPI
  - Passo 5 – Scrivere il main cui effettuare la comunicazione

Riferimenti

- L'esercizio che fa uso dell'interfaccia SPI (sulla STM32F4) è reperibile su GitHub:
- https://github.com/fboris/STM32Cube_FW_F4/tree/master/Projects/STM32F4-Discovery/Examples/SPI/SPI_FullDuplex_ComPolling
- Attenzione che tale esempio è per la scheda STM32F4 ed inoltre utilizza alcune librerie per l'utilizzo di LED e buttons diverse dalle nostre
- Dunque utilizzarlo come punto di partenza per lo sviluppo del proprio codice