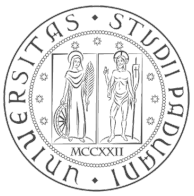


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Elettronica

Sistema di test per la caratterizzazione di dispositivi integrati.

Realizzazione di un'interfaccia per la generazione di comunicazioni seriali programmabili e sincronizzabili con segnali digitali

Laureando

Alberto Trentin

Relatore

Ch.ma Prof.ssa Giada Giorgi

Correlatore

Ing. Enrico Mottin

Anno Accademico 2016/2017

..nessuno muore sulla terra finché
vive nel cuore di chi resta..

A papà

Indice

Capitolo 1

Introduzione

Durante lo sviluppo di un dispositivo integrato su silicio, la caratterizzazione, ovvero la verifica sul prototipo fisico delle funzionalità e caratteristiche definite in fase di design secondo le specifiche di progetto, rimane uno dei passaggi fondamentali nonostante la continua evoluzione e aggiornamento degli strumenti e dei modelli di simulazione.
Le specifiche di un prodotto sono sintetizzate nel datasheet, reso disponibile all'utilizzatore finale.
La caratterizzazione si avvale di strumenti di misura tra i quali generatori di forme d'onda, multimetri e oscilloscopi per forzare e monitorare gli ingressi e le uscite dei dispositivi da testare (DUT), verificando che il comportamento corrisponda a quanto desiderato. Il dispositivo viene testato nelle diverse condizioni di lavoro specificate nel datasheet per individuare eventuali malfunzionamenti, affinché si possa procedere con un redesign o a una modifica del datasheet, prima di rilasciare il prodotto sul mercato.
Lo scopo di questo progetto è realizzare un'interfaccia che, integrata in un ampio sistema di test, permetta di forzare gli ingressi mediante comunicazioni seriali programmabili e sincronizzabili con segnali digitali.

1.1 Panoramica del progetto

Il progetto è nato per rispondere alle esigenze del team di product engineer della sezione body power di Infineon Italia (Padova), che si occupa della caratterizzazione e del controllo post-produzione di dispositivi LED driver, come automotive LED driver IC e multichannel relay LED driver.
La maggior parte di questi prodotti permette all'utilizzatore di interfacciarsi tramite pin digitali di ingresso oppure mediante moduli di comunicazione seriali basati sui protocolli SPI e SCI.

1.1.1 Richieste

In fase di caratterizzazione è necessario forzare gli ingressi del DUT in modo da verificarne il corretto funzionamento e ricavare informazioni sulle tempistiche di lavoro che andranno poi a definire le performance del dispositivo.
Il compito del sistema di test è quello di validare i dati stimati mediante simulazioni svolte durante la fase di design e individuare eventuali deviazioni o malfunzionamenti.

Per fare ciò, è richiesta un'interfaccia in grado di inviare pattern digitali: un insieme di comunicazioni seriali temporizzate e sincronizzate con segnali digitali.
In sostanza, è richiesta la possibilità di gestire con elevata risoluzione la distanza temporale tra i vari comandi che compongono una comunicazione, la quale può essere generalmente composta da comandi SPI, SCI e commutazioni di linee digitali. Per quanto riguarda i singoli comandi è richiesta la possibilità di personalizzare i parametri principali dei vari protocolli di comunicazione per esempio frequenza, numero di bit ecc.

1.1.2 Esempio di caretterizzazione di un dispositivo

Prendendo in considerazione un driver per il controllo di relè e LED basato su comunicazioni SPI, Infineon TLE750008-EMD [1], possiamo capire quali sono alcune delle principali funzioni richieste all'interfaccia di comunicazione.
Il dispositivo mette a disposizione dell'utente 4 modalità operative: Sleep mode, Idle mode, Active mode ed Limp Home mode. Le transizioni tra modalità operative sono determinate da commutazioni dei pin di ingresso (IDLE pin, INn pin), oppure aggiornando i registri di stato mediante comunicazioni SPI.
In fase di caratterizzazione è necessario misurare il tempo impiegato dal chip par cambiare modalità operativa, in modo da verificare le informazioni contenute nel datasheet, come mostrato in Figura 1.1. Per esempio, per misurare $t_{ACTIVE2LH}$, ovvero il tempo necessario per passare da Active mode ad Limp Home mode, bisogna eseguire sequenzialmente le seguenti operazioni:

- set IDLE pin: passare da Sleep mode ad Idle mode mediante commutazione low to high della linea digitale Idle;
- write Idle to Active: passare da Idle mode ad Active mode aggiornando l'opportuno registro con comunicazione SPI;
- set IN1 pin: attivare un'uscita mediante commutazione low to high della linea digitale IN1;
- reset IDLE pin: passare da Active mode a Limp Home mode mediante commutazione high to low della linea digitale Idle;
- read Standard Diagnosis: leggere mediante comunicazione SPI il registro Standard Diagnosis per verificare il cambiamento di modalità operativa.

In questo esempio, per caratterizzare $t_{ACTIVE2LH}$ è fondamentale gestire la distanza tra il reset di IDLE pin e l'invio del comando di lettura Standard Diagnosis. Quest'ultimo registro, infatti, contiene informazioni sull'attuale modalità operativa del DUT. Ripetendo la sequenza di operazioni appena descritte e variando la distanza tra i due, attraverso la lettura di Standard Diagnosis si può facilmente misurare il tempo necessario per cambiare modalità operativa. L'interfaccia di comunicazione deve quindi mettere a disposizione dell'utente interfacce di comunicazione seriale (SPI) sincronizzabili con uscite digitali con le quali inviare pattern in ingresso al DUT.

Table 8 Electrical Characteristics Power Supply (cont'd)

V_{DD} = 3 V to 5.5 V, V_S = 7 V to 18 V, T_J = -40 °C to +150 °C, all voltages with respect to ground, positive currents flowing as described in Figure 2 (unless otherwise specified)
Typical values: V_{DD} = 5 V, V_S = 13.5 V, T_J = 25 °C

Parameter	Symbol	Values			Unit	Note / Test Condition	Number
		Min.	Typ.	Max.			
Idle to Active delay	$t_{IDLE2ACTIVE}$	–	100	200	µs	1) from INn or CSN pins to MODE = 10_B	P_6.3.55
Active to Idle delay	$t_{ACTIVE2IDLE}$	–	100	200	µs	1) from INn or CSN pins to MODE = 11_B	P_6.3.56
Sleep to Limp Home delay	$t_{SLEEP2LH}$	–	300 +t_{ON}	600 +t_{ON}	µs	1) from INn pins to V_{DS} = 10% V_S	P_6.3.57
Limp Home to Sleep delay	$t_{LH2SLEEP}$	–	200 +t_{OFF}	400 +t_{OFF}	µs	1) from INn pins to Standard Diagnosis = 0000_H (see Chapter 10.6.1 for details). External pull-down SO to GND required	P_6.3.58
Limp Home to Active delay	$t_{LH2ACTIVE}$	–	50	100	µs	1) from IDLE pin to MODE = 10_B	P_6.3.59
Active to Limp Home delay	$t_{ACTIVE2LH}$	–	50	100	µs	1) from IDLE pin to TER + INST register = 8683_H (IN0 = IN1 = "high") or 8682_H(IN1 = "high", IN0 = "low") or 8681_H (IN1 = "low", IN0 = "high") (see Chapter 10.5 for details)	P_6.3.60

Figura 1.1: Esempio, electrical characteristics power supply TLE750008-EMD.

1.1.3 Motivazioni

Per svolgere questo tipo di misure, le comunicazioni seriali e le commutazioni digitali sono attualmente generate utilizzando dei moduli DAQ di acquisizione dati multifunzione, mentre per misure specifiche si ricorre a moduli di comunicazione personalizzati basati su FPGA o microcontrollore appositamente programmato.
Queste soluzioni, tuttavia, risultano avere alcune limiti: l'utilizzo del DAQ permette di generare pattern relativamente piccoli, garantendo l'invio di pochi comandi e poche commutazioni digitali; inoltre, la risoluzione temporale del pattern è nell'ordine di alcuni microsecondi. L'utilizzo di FPGA e microcontrollori richiede tempo e competenze speci-

fiche per la loro programmazione.
Inoltre un singolo sistema di test può contenere più dispositivi e un singolo dispositivo può avere molteplici ingressi seriali e digitali, richiedendo più interfacce di comunicazione, per le quali un dispositivo DAQ o FPGA può non essere sufficiente, con un conseguente aumento dei costi.
Sul mercato questi dispositivi verranno utilizzati in sistemi le cui interfacce di comunicazione e i segnali digitali sono gestiti da microcontrollori. Uno degli aspetti che si prende in considerazione è la verifica del dispositivo il più vicino possibile alle condizioni di lavoro sul campo. Per questo si preferisce l'utilizzo, in fase di caratterizzazione, di un'interfaccia a microcontrollore rispetto alla soluzione con DAQ. Lo scopo del progetto è sviluppare un'interfaccia che tratti i problemi appena descritti, ovvero:

- semplice da usare;
- con (elevata risoluzione temporale);
- economico;
- prossimo all'ambiente d'utilizzo finale.

1.1.4 Caratteristiche

Dal punto di vista hardware l'interfaccia di comunicazione deve essere implementata su microcontrollore e deve avere delle uscite digitali che permettano di forzare gli ingressi digitali dei DUT, analogamente, delle interfacce di comunicazione seriale (SPI, SCI) per leggere e scrivere i registri.
Per quanto riguarda il software, per prima cosa deve essere sviluppato il firmware del microcontrollore, e successivamente deve essere sviluppata una GUI che permetta all'utente di descrivere il pattern da generare in modo semplice ed intuitivo, garantendo la possibilità di personalizzare i parametri delle comunicazioni seriali.
Un sistema di misura e caratterizzazione è composto da molteplici strumenti coordinati tra di loro, in questo caso gestito dall'ambiente di sviluppo Labview, che permette di automatizzare le misure. Deve quindi essere sviluppato un driver (Labview) che permetta l'integrazione semplice e veloce dell'interfaccia nel sistema di test. Riassumendo, le attività principali che caratterizzano il progetto e lo sviluppo dell'interfaccia sono:

- selezione dell'hardware;
- programmazione del firmware;
- programmazione del driver (Labview).

1.2 Struttura dell'elaborato

L'elaborato è suddiviso in otto capitoli. Dopo questa prima parte introduttiva, si passa a descrivere i vari passaggi che hanno portato alla progettazione e sviluppo dell'interfaccia.

Nel secondo capitolo vengono presentate le specifiche di progetto. In primo luogo, viene

fornita una panoramica sui protocolli di comunicazione seriale integrati nell'interfaccia. Successivamente, si descrivono nel dettaglio i requisiti di progetto.

Il terzo capitolo illustra i passaggi che hanno portato alla scelta dell'hardware. Dopo una breve presentazione dello studio di fattibilità, si passa alla descrizione dei criteri di scelta del"hardware, in particolare del microcontrollore. In seguito, vengono descritte le soluzioni adottate.

Il quarto capitolo è dedicato alla presentazione del firmware, e alla descrizione dettagliata delle periferiche del microcontrollore, utilizzate in questo. Dopo aver illustrato brevemente il software di sviluppo, viene fornita la struttura del firmware. In primo luogo, sono descritte le interazioni tra moduli e CPU, che permettono la sincronizzazione di commutazioni digitali e comunicazioni seriali. Successivamente, viene mostrato il codice, con particolare attenzione al protocollo e alle funzioni di comunicazione con il driver. Infine, è mostrata la mappa per l'individuazione dei pin di I/O dell'interfaccia.

Il quinto capitolo mostra la tabella di definizione del pattern. Le informazioni contenute in questo capitolo servono all'utilizzatore finale per gestire correttamente l'interfaccia. Viene descritto il formato standard da utilizzare, ovvero l'esatta collocazione dei valori di configurazione all'interno della tabella. In seguito, vengono delineate le regole per definire il pattern.

Il sesto capitolo fornisce una panoramica sul driver Labview. Più che alla struttura, si è data importanza alla descrizione dell'uso del driver e delle funzioni da esso svolte, al fine di creare un manuale di utilizzo per l'utente finale, attraverso il quale integrare l'interfaccia in un sistema di test. Vengono quindi presentate le funzioni (VI) di gestione e comunicazione con l'interfaccia, con particolare attenzione al formato degli input/output di ogni VI.

Il settimo capitolo mostra due esempi di utilizzo dell,interfaccia. Nel primo esempio viene misurato il tempo necessario per cambiare modalità operativa di un driver per il controllo di relè. Nel secondo esempio viene verificato il corretto funzionamento di una routine di controllo per le uscite di un convertitore H-Bridge DC/DC.

Capitolo 2

Specifiche di progetto

Lo scopo principale dell'interfaccia di comunicazione è garantire la sincronizzazione tra comunicazioni seriali e linee digitali. È importante poter personalizzare le principali caratteristiche delle comunicazioni seriali e gestire con elevata risoluzione le temporizzazioni tra gli elementi che compongo il pattern.
Per comprendere nel dettaglio i requisiti di progetto è necessario avere una panoramica delle comunicazioni seriali richieste (SPI/SCI) e del loro ruolo nella gestione dei dispositivi.

2.1 Comunicazione serali

La trasmissione seriale è una modalità di comunicazione tra dispositivi digitali nella quale i bit sono comunicati uno di seguito all'altro e giungono sequenzialmente al ricevente nello stesso ordine in cui sono stati trasmessi dal mittente. Nonostante i moduli di comunicazione siano più complessi rispetto alla trasmissione parallela, la modalità seriale è una delle più diffuse soprattutto nelle comunicazioni tra chip che devono comunicare fra loro a grandi distanze, perché:

- richiede un minor numero di fili con conseguente riduzione dei costi;
- è più tollerante rispetto alle interferenze e agli errori di trasmissione.

Esistono due categorie di trasmissione seriale: trasmissione asincrona e sincrona.

Trasmissione asincrona

Nelle trasmissioni asincrone non si trasmette il clock, ma il ricevitore genera un clock locale della stessa frequenza del trasmettitore. Affinché i due clock risultino in fase, occorre che il ricevitore sappia quando ha inizio la trasmissione di un carattere, in modo da sincronizzare la lettura dei vari bit. In pratica, un carattere in trasmissione è preceduto da un bit di start e seguito da uno o più bit di stop.

Trasmissione sincrona

Nelle trasmissioni sincrone il trasmettitore invia degli impulsi di clock contemporaneamente ai bit di informazione, in modo da consentire al ricevitore la corretta lettura dei dati in arrivo a intervalli regolari di tempo scanditi dal trasmettitore.

2.1.1 SPI (Serial Peripheral Interface)

Il principale tipo di comunicazione seriale implementato nei dispositivi da caratterizzare è il Serial Peripheral Interface o SPI, un bus standard di comunicazione ideato dalla Motorola Inc. "SPI Block Guide V03.06" [2], uno dei più usati per comunicazione inter-chip a medio-basse velocità di trasferimento [3].
La trasmissione avviene tra un dispositivo detto Master e uno o più Slave. Il Master controlla il bus, emette il segnale di clock e decide quando iniziare e terminare la comunicazione. Il bus SPI si definisce:

- sincrono, per la presenza di un clock che coordina la trasmissione e ricezione dei singoli bit che determina la velocità di trasmissione;
- full-duplex, in quanto il "colloquio" può avvenire contemporaneamente in trasmissione e ricezione.

Tale bus è basato su quattro segnali [4]:

- CLK/SCK: inviato dal master agli slave; tutti i segnali SPI sono sincronizzati con questo segnale di clock;
- CS/SS: usato per selezionare lo slave con il quale il Master intende comunciare;
- MOSI/SIMO: Master Output Slave Input, linea di dati dal Master allo Slave;
- MISO/SOMI: Master Iutput Slave Output, linea di dati dallo Slave al Master.

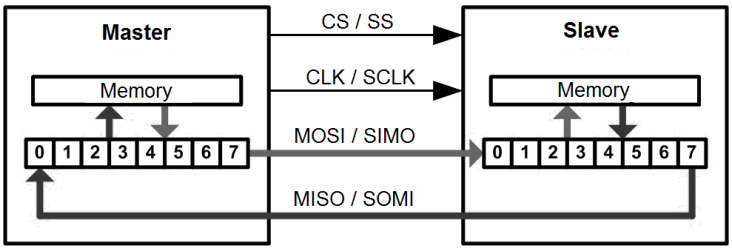


Figura 2.1: Registri e linee dati SPI.

Ogni device possiede uno shift register contenente i dati e il trasferimento prevede lo scambio del loro contenuto. Il Master, tramite il segnale di clock, gestisce tale trasferimento con cui il contenuto del suo shift register viene inviato allo Slave attraverso il segnale MOSI, mentre il contenuto dello shift register dello Slave viene inviato al Master attraverso il segnale MISO (Figura 2.1). Ad ogni impulso di clock, entrambi i dispositivi emettono un bit dal loro registro interno rimpiazzandolo con un bit emesso dall'altro interlocutore. La sincronizzazione è fatta sui fronti di clock di salita o di discesa. [5]
La comunicazione viene intrapresa sempre su iniziativa del dispositivo Master, che abilita lo Slave tramite CS e successivamente impone il clock sulla linea dedicata. Con questa procedura ha inizio lo scambio dei bit tra i due registri.
Alla fine di ogni parola trasmessa il contenuto del registro dello Slave sarà passato al Master e viceversa. Con opportune parole identificative si possono inviare comandi al

dispositivo ricevente, che potrà effettuare l'elaborazione assegnata ponendo poi nel suo shift-register il dato richiesto, che al prossimo ciclo di trasmissione verrà inviato al richiedente.
Esistono varie tipologie di comunicazione SPI che differiscono tra loro per polarità di CS e CLK. Per quanto riguarda CS, abbiamo quattro diverse opzioni:

- attivo basso: la comunicazione ha inizio con una transizione high to low della linea di CS, finisce con una transizione low to high e durante tutta la comunicazione permane lo stato logico basso;
- attivo alto: la comunicazione ha inizio con una transizione low to high della linea di CS, finisce con una transizione high to low e durante tutto la comunicazione permane lo stato logico alto;
- sempre basso: durante tutta la comunicazione CS permane lo stato logico basso. Inzio e fine commutazione sono gestiti dal segnale di CLK;
- sempre alto: durante tutta la comunicazione CS permane lo stato logico alto. Inzio e fine commutazione sono gestiti dal segnale di CLK.

La sincronizzazione è regolata da due parametri che possono essere impostati dall'utente: CPOL e CPHA [6].
CPOL regola la polarità del clock, ovvero discrimina lo stato normale di riposo cui si porta la linea di clock quando non è attiva. Quando CPOL è impostato a 0, il clock, nel suo stato di riposo, si porta a livello logico basso; viceversa, il clock si porta a livello logico alto durante il tempo di inattività se CPOL è impostato a 1.
CPHA regola la fase del clock, ovvero il fronte di clock in cui il ricevente campiona il segnale in ingresso. Se CPOL=0 allora con CPHA possiamo scegliere di campionare il dato sul fronte di salita del segnale di clock, impostando CPHA=0, oppure sul fronte di discesa impostando CPHA a 1. L'inverso accade se CPOL è settato a 1 (Figura 2.2).
Riassumendo, abbiamo quindi quattro diverse configurazioni possibili per il segnale di sincronizzazione CLK:

- CPOL=0, CPHA=0: CLK inattivo basso, scrittura sui fronti di discesa e lettura su fronti di salita di CLK;
- CPOL=0, CPHA=1: CLK inattivo basso, scrittura sui fronti di salita e lettura su fronti di discesa di CLK;
- CPOL=1, CPHA=0: CLK inattivo alto, scrittura sui fronti di salita e lettura su fronti di discesa di CLK;
- CPOL=1, CPHA=1: CLK inattivo alto, scrittura sui fronti di discesa e lettura su fronti di salita di CLK.

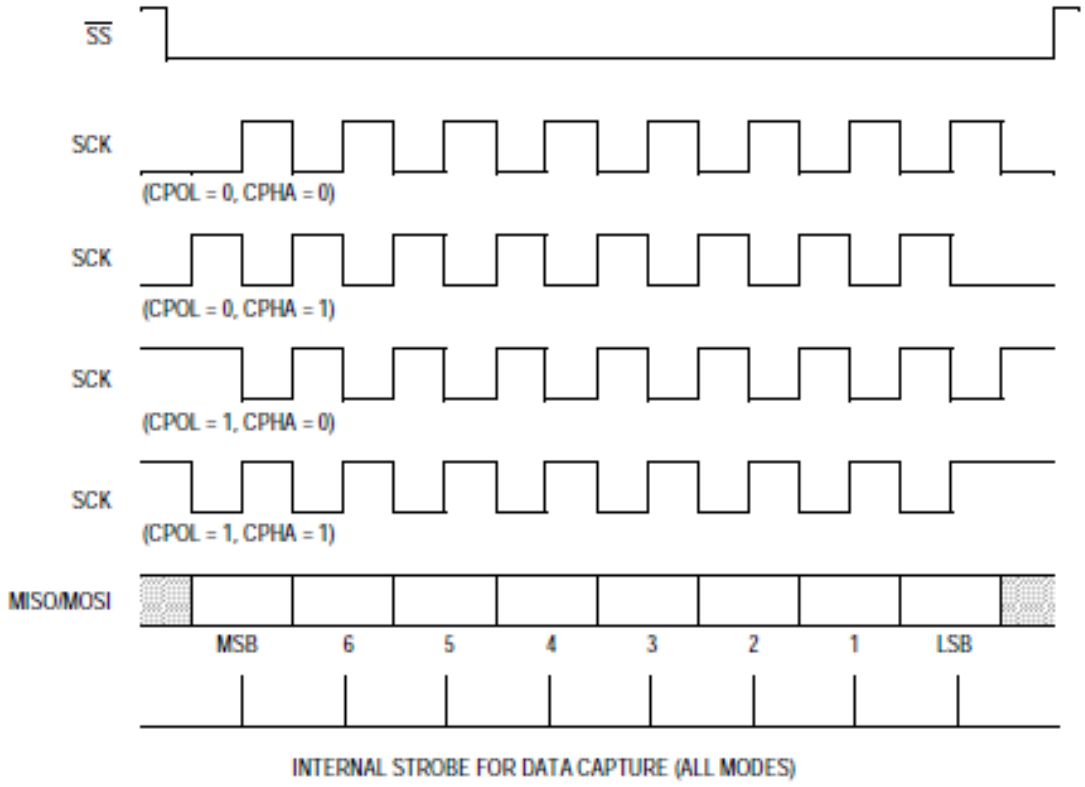


Figura 2.2: Possibili configurazioni della comunicazione SPI al variare di CPOL e CPHA.

Nel caso di collegamento con più Slave, qualora ci sia un unico Master, quest'ultimo ha il compito di selezionare lo Slave con il quale avviare la comunicazione. Per fare ciò esistono due soluzioni[7].

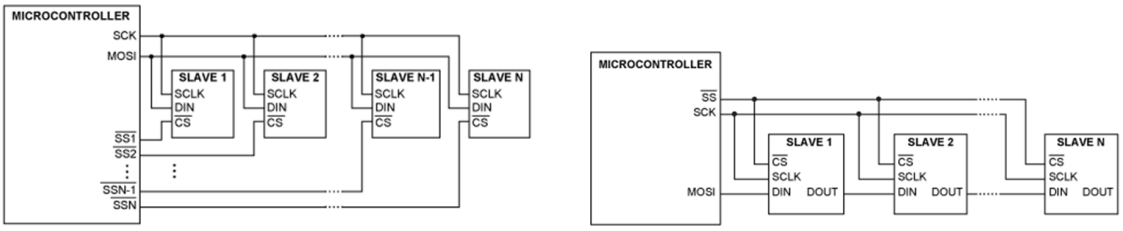
Dispositivi slave controllati singolarmente

Il Master è provvisto di diverse linee di selezione CS, ognuna dedicata a un singolo Slave. La linea CS, normalmente attiva bassa, in caso di disabilitazione (livello logico alto) lascia il dispositivo slave con uscita in alta impedenza e quindi completamente isolato dal bus, indifferentemente dall'esistenza del segnale di clock (Figura 2.3(a)).

Dispositivi slave connessi in catena (daisy chain)

In questa configurazione, vari dispositivi sono attivati dallo stesso CS. La linea MOSI di ogni dispositivo è connessa alla linea MISO di un altro dispositivo in modo da formare una catena. La fine della catena di Slave è connessa direttamente al Master.
Durante una comunicazione, se CS risulta attivo e lo shift register è stato riempito completamente dai dati in ingresso, eventuali nuovi bit in ingresso causano lo shift del messaggio

lungo la catena. In questo caso, per comunicare con tutti i dispositivi il Master deve essere in grado di generare comandi SPI di dimensioni multiple rispetto alla lunghezza dello shift register di un singolo device.
Per esempio, collegando 4 dispositivi con shift register di 16-bit, il Master deve generare un comando SPI lungo 64-bit per poter raggiungere l'ultimo elemento della catena (Figura 2.3(b)).



(a) Dispositivi slave controllati singolarmente

(b) Dispositivi slave connessi in catena (daisy chain)

Figura 2.3: Collegamento di Slave multipli

2.1.2 Tempisitche SPI

La caratterizzazione ha come obiettivo la valutazione dei parametri e delle risposte del dispositivo alle sollecitazioni variabili nel range operativo definito nel datasheet e in alcuni casi anche oltre per verificarne i limiti. Occorre quindi avere a disposizione strumenti in grado di variare i parametri dei segnali forzati al DUT.
Il più semplice esempio di caratterizzazione per una comunicazione seriale sincrona è quello di variare la frequenza del segnale di clock per poter definire un limite entro il quale il corretto funzionamento del dispositivo è assicurato e oltre il quale il produttore non garantisce affidabilità.

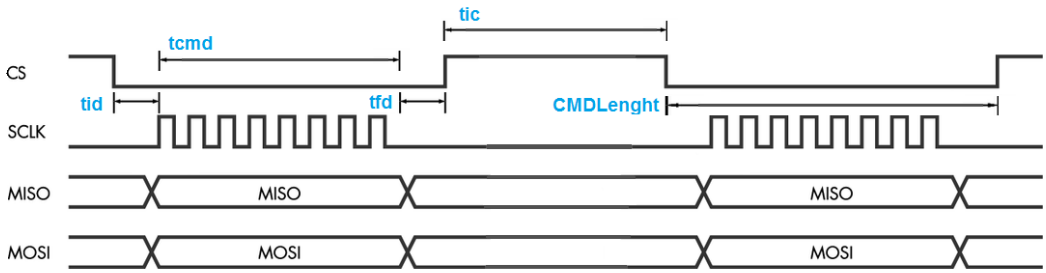


Figura 2.4: Parametri temporali comunicazione SPI.

In Figura 2.4 è rappresentata una comunicazione SPI con CS attivo basso. In questo caso, una transizione high-low della linea, imposta dal Master è interpretata dallo Slave come l'inizio di una comunicazione con il conseguente scambio di dati attraverso le linee MOSI e MISO e la generazione del segnale di clock. Una transizione low-high rappresenta invece la fine della comunicazione. In gergo, la porzione di comunicazione compresa tra le due

transizioni di CS è detta comando (CMD).
In figura, quindi, è rappresentata una comunicazione composta da due comandi adiacenti. Le transazioni di CS sono gli estremi della comunicazione e rappresentano segnali importanti per l'elaborazione. Molto spesso, infatti, attraverso un comando SPI si impostano registri dei dispositivi e la commutazione di fine comando è interpretata dal chip come un segnale di start per l'elaborazione. La commutazione di inizio comando indica al dispositivo che deve prepararsi a comunicare. Questo duplice effetto delle transizioni di CS spiega perché, in fase di caratterizzazione, è fondamentale poter gestire con elevata risoluzione la distanza tra le commutazioni di CS.
Riprendendo in considerazione l'esempio della sez. 1.1.2, una transizione di modalità operativa del dispositivo è comunicata tramite la scrittura di un registro. L'effettiva elaborazione, invece, ha inizio solamente con l'avvenuta transizione di fine comando del CS. Un eventuale invio di un altro comando, per esempio di Standard Diagnosis, determina la lettura dello stato del dispositivo, non appena avviene la commutazione di inizio comando.
Avvalendoci di quanto mostrato in Figura 2.4, possiamo definire quali sono i parametri di una comunicazione SPI da considerare in fase di caratterizzazione del dispositivo:

- tid: distanza tra la commutazione di inizio comando del CS e l'inizio della trasmissione;
- tif: distanza tra la commutazione di fine comando del CS e la fine della trasmissione;
- $freq = \frac{1}{tp}$: frequenza del segnale di clock CLK;
- tic: distanza tra le commutazione del segnale CS di fine comando ed inizio del comando successivo.

2.1.3 SCI

Oltre al protocollo SPI è richiesta la possibilità di generare comunicazioni basate sul protocollo SCI adattato al dispositivo in sviluppo(Figura 2.5). Questo bus di sola scrittura è utilizzato per flussi di dati con frequenze superiori a quelle della comunicazione SPI, si compone di tre linee di comunicazione: un segnale di sincronizzazione CLK, una linea dati per comandi a lunghezza fissa 264-bit (256-bit di dati, seguito da 8-bit di checksum) e un secondo segnale di sincronizzazione SYNC. Il trasmettitore commuta il segnale sulla linea data in corrispondenza del fronte di salita del segnale di CLK. Il ricevitore campiona il segnale sui fronti di discesa. Il segnale di sincronizzazione SYNC è attivo basso (livello logico=0) a partire dal primo fronte di salita del CLK associato all'invio del primo bit della comunicazione (nel caso considerato bit data 255) e permane in questo stato fino al fronte di salita associato all'ultimo (bit 0) dove commuta ed è disattivato (livello logico 1) fino all'invio del prossimo comando.

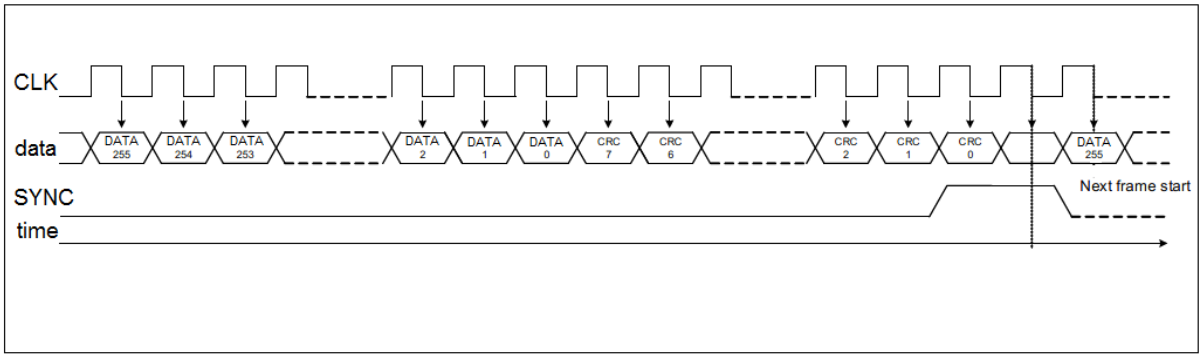


Figura 2.5: Protocollo SCI.

2.2 Requisiti di progetto

Le specifiche di progetto definiscono quali funzioni debbano essere implementate nell'interfaccia. Alcune di queste specifiche sono opzionali e talvolta sovradimensionate rispetto alle effettive necessità dell'utilizzatore e alle possibilità della piattaforma hardware/software scelta. Tali richieste possono quindi essere rinegoziate in fase di sviluppo. Altre invece sono imprescindibili e hanno assoluta priorità rispetto alle altre.
I requisiti di progetto dell'interfaccia sono riassunti in (tab. 2.1).

Programmable digital lines interface module

	Requisiti di progetto
Hardware features	tensione di input/output variabile da 3 a 5,5V (opzionale: inferiore a 2V)
	opzionale: protezione da cortocircuiti e sovratensioni
	opzionale: livello di tensione dell'interfaccia SPI/SCI e delle linee digitali configurabile individualmente
	opzionale: disconnessione meccanica delle connessioni
	opzionale: interfaccia di comunicazione LVDS
SPI interface features	interfacce comunicazione seriale: 1 SPI, 1 SCI (linee CS/SI/CLK/SO) (opzionale più di 1)
	lettura del TER bit
	numero di bit da 1 a 1024
	frequenza CLK fino a 10MHz (opzionale fino a 50MHz)
	distanza tra comandi SPI programmabile (sequenze di comandi SPI fino a 100 elementi)
	possibilità di inviare sequenze di comandi SPI in loop
	ritardi (tid, tfd, tnf, ...) programmabili
	possibilità di scegliere polarità de segnali SPI (CS, CLK edge, ...)
Digital outputs features	8 segnali digitali programmabili e sincronizzabili con l'interfaccia SPI/SCI
	2 canali di generazione segnali PWM (freq/duty cycle programmabili)
Trigger IN	1 linea digitali di input(opzionale 2): un segnale di tigger in ingresso che permetta di attivare una comunicazione SPI o commutazioni di linee digitali
Trigger OUT	1 linea digitale di output: un segnale di trigger in uscita attivato da eventi (fine di comunicazione SPI o lettura programmabile dei comandi SPI ricevuti)
Connection to PC	comunicazione con PC attraverso l'interfaccia USB
Software	creazione di una GUI
	creazione di driver per Labview e Matlab

Tabella 2.1: Requisiti di progetto.

2.2.1 Specifiche dell'hardware

Definiscono le caratteristiche elettriche dell'interfaccia. Nello specifico, è molto importante poter variare le tensioni di ingresso e uscita in modo da caratterizzare i livelli di tensione che garantiscono la corretta comunicazione. Sono richiesti anche alcuni meccanismi di protezione per eventuali cortocircuiti o sovratensioni che possono essere inseriti direttamente nell'interfaccia o in alternativa sulle board di caratterizzazione.

2.2.2 Specifiche dell'interfaccia SPI

Definiscono quali sono i parametri dei protocolli di comunicazione seriale che devono essere configurabili. È richiesta la personalizzazione di tutti i parametri visti in sez. 2.1.2, in particolare la gestione del *tic* e il raggiungimento di elevate frequenze di CLK. Ulteriore richiesta è la gestione delle polarità di CLK e CS viste in sez. 2.1.1 per garantire all'interfaccia di comunicare con qualsiasi tipologia di comunicazione SPI. Anche il numero di bit di un singolo comando deve essere configurabile: un elevato numero di bit è necessario per gestire un eventuale connessione daisy chain di più dispositivi, mentre la possibilità di inviare comandi con lunghezza arbitraria è utile per verificare il comportamento dei dispositivi qualora vengano inviati comandi volutamente errati, per esempio privi di alcuni bit oppure più lunghi rispetto alla lunghezza nominale. L'interfaccia deve prevedere la possibilità di inviare comandi in loop che possono terminare previa lettura di uno specifico comando SPI definito dall'utente.

2.2.3 Specifiche delle uscite digitali

Definiscono le caratteristiche delle linee digitali. In sostanza, è richiesto di poter commutare linee digitali sincronizzate con comandi SPI e di poter quindi gestire le distanze tra commutazioni delle linee di inizio e fine comandi.
Considerando l'esempio in sez. 1.1.2, cambiamenti di modalità operativa possono avvenire tramite variazione dei pin di ingresso. In questo caso, la gestione delle distanze tra commutazione del pin e inizio SPI permette di caratterizzare al meglio i tempi necessari per il cambio di modalità operativa.
È necessario anche un pin che svolga la funzione di trigger di ingresso, ovvero la possibilità di inviare un pattern di comunicazione in risposta alla commutazione della linea di trigger. Deve essere possibile commutare linee digitali alla fine di un loop di comandi e, infine, sono richiesti due generatori d'onda quadra (PWM) con frequenza e duty cycle programmabili.

2.2.4 Interfaccia grafica (GUI)

Una richiesta è poter integrare l'interfaccia in un ampio sistema di test basato sull'ambiente di sviluppo Labview. È necessario quindi fornire all'utente un driver per gestire le funzionalità e una semplice ed intuitiva GUI che permetta la descrizione del pattern da inviare mediante formato tabellare simile a quello in tab. 2.2[1].
I dati vengono organizzati in due sotto-matrici: la prima contiene i parametri della comunicazione, ovvero le caratteristiche dei comandi seriali per esempio (freq, tid, polarità

[1]Il formato finale della tabella è descritto nel capitolo 5

CS,CLK, ecc.); nell'altra, invece, è definito il pattern vero e proprio. Ogni riga di questa sotto-matrice fa riferimento all'istante temporale definito nella colonna TIME. Nella colonna CMD è definito l'eventuale invio di un comando, e nelle colonne Digital-(1-5) eventuali commutazioni delle linee digitali.
Prendendo come esempio la tab. 2.2, dalla prima sotto-matrice possiamo ricavare le caratteristiche dei comandi (freq=5 MHz, tid=100ns, ecc.), mentre la seconda sotto-matrice descrive invece il seguente pattern:

- time 0: istante di inizio pattern; invio del comando esadecimale FFCC e commutazione delle linee digitali Digital_1=0 e Digital_2=1;
- time 5μs: dopo 5μs dall'istante iniziale commutazione della linea digitale Digital_1=1;
- time 20μs: dopo 20μs dall'istante iniziale invio del comando esadecimale FFAA e commutazione della linea digitale Digital_2=0.

freq	5 MHz					
tid	100ns					
CS pol	1					
CLK pol	0					
TIME(μs)	CMD	Digital_1	Digital_2	Digital_3	Digital_4	Digital_5
0	FFCC	0	1			
5		1				
20	FFAA		0			

Tabella 2.2: Descrizone pattern in formato tabellare

Capitolo 3

Scelta dell'hardware

Scegliere l'hardware più idoneo a soddisfare le specifiche è uno degli step principali in fase di progetto. Per quanto riguarda il microcontrollore i parametri che influenzano la scelta sono molteplici: performance, quantità e tipo di periferiche disponibili molto spesso non bastano se non sono correlate a un buon ambiente di sviluppo del firmware e una chiara documentazione del dispositivo.

3.1 Studio di fattibilità

Per prendere confidenza con il problema da risolvere e individuare la soluzione migliore si è deciso di effettuare uno studio di fattibilità basandosi sull'evaluation board Infineon x166 attualmente in uso, previa specifica programmazione, per i test di caratterizzazione più complessi. Dopo un attento studio di periferiche e CPU si sono delineate due possibili soluzioni:

- Interrupt: questa soluzione prevede la generazione di un segnale di temporizzazione periodico attraverso l'utilizzo del modulo timer, che permette la generazione di un interrupt a distanze temporali definite dalle impostazioni del timer. A ogni istanza di interrupt si può agire sui pin di GPIO modificando le uscite. In questo modo, ogni linea (Digital ed SPI) deve essere gestita attraverso degli array numerici di commutazioni preimpostati a seconda del pattern da generare. Tutta la gestione dell'interfaccia è quindi lasciata alla CPU, e questa soluzione necessita di molta memoria in quanto, per ogni istanza di interrupt, deve esistere un array contenente i valori di ogni linea d'uscita.
  I limiti di questo approccio sono evidenti: la generazione di un interrupt a partire da un flag del modulo timer richiede alcune centinaia di ns che, sommati al tempo necessario per impostare i registri di uscita GPIO, permettono di utilizzare periodi del timer di alcuni μs, ovvero di generare comunicazioni seriali con frequenza pari a qualche centinaia di kHz. È chiaro che questo approccio al problema non è compatibile con i requisiti di progetto. A ogni modo questo tentativo si è rivelato utile come primo approccio alla programmazione del microcontrollore e alcuni aspetti di quanto descritto sono stati utilizzati per ottenere la soluzione finale descritta nei prossimi capitoli.

- Modulo SPI: un approccio software, considerando la frequenza massima di lavoro dei microcontrollori (200-300MHz) e le limitate capacità di calcolo, è impensabile. Per poter garantire comunicazioni seriali con frequenze nell'ordine dei MHz è necessario utilizzare un apposito modulo SPI.
  I moduli SPI consentono di raggiungere elevate prestazione utilizzando un apposito hardware basato sull'utilizzo di buffer per immagazzinare i dati in trasmissione e ricezione, e di shift register per l'erogazione dei dati.
  L' architettura di un modulo di gestione SPI (Figura 3.1 tratta dal documento "SPI Block Guide V03.06" [2]) mostra che la complessità del modulo non si limita a un solo shift register, ma comprende una complessa struttura di gestione delle polarità del clock per consentire una comunicazione interamente automatizzata anche nel caso di un full-duplex. I moduli integrati nei microcontrollori ricalcano questa struttura, ma possono non essere esattamente identici, per cui occorrerà documentarsi sui fogli informativi prima di usarli.

La soluzione basata su interrupt non può essere adottata in quanto il limite di qualche centinaia di kHz non è compatibile con il requisito fondamentale di alte frequenze (10MHz). L'interfaccia, quindi, deve essere sviluppata basandosi sul modulo di comunicazione SPI presente nel microcontrollore. Una difficoltà del progetto diventa trovare un metodo per sincronizzare i segnali SPI con linee digitali e trovare un dispositivo con un modulo SPI altamente configurabile.
I moduli che agiscono direttamente sui pin di uscita e possono essere utilizzati per commutazioni digitali sono il modulo GPIO o il modulo PWM.
Da un colloquio avuto con alcuni ingegneri di Infineon specializzati nel settore dei microcontrollori, è emerso che l'unico modo per garantire la sincronizzazione ad alte frequenze dei segnali prodotti del modulo SPI con commutazioni digitali dei pin è sfruttare al meglio le varie periferiche e le interconnessioni tra di loro e ridurre al minimo il numero di istruzioni che devono essere svolte dalla CPU.

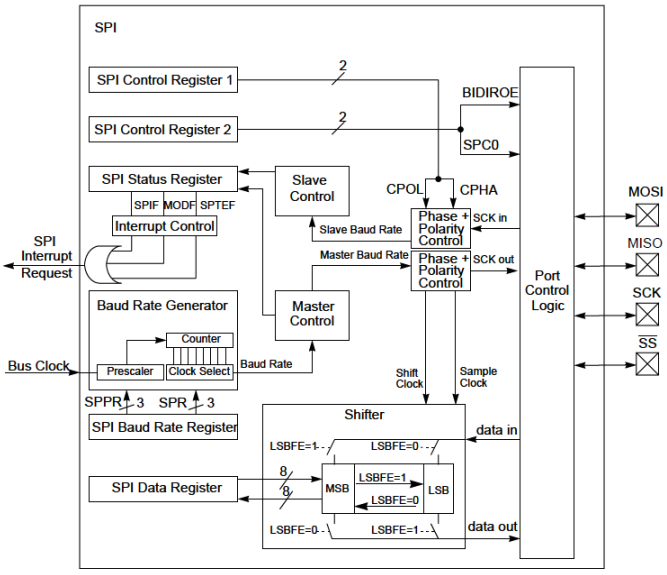


Figura 3.1: Tipica struttura modulo SPI.

3.2 Microcontrollore

In questo caso, considerando tipo e quantità dei requisiti di progetto e il tempo a disposizione, è chiaro che lo sviluppo di un sistema completo e dedicato che preveda la creazione di una board partendo dal singolo microcontrollre non è la soluzione adatta.
La scelta ricade quindi sull'utilizzo delle cosiddette development board, ovvero circuiti stampati dove, oltre al microcontrollore, sono presenti interfacce di comunicazione (USB), connettori di alimentazione, debugger (USB) e connettori di accesso ai pin di GPIO. In questo modo, l'utilizzatore si deve occupare solamente del progetto del firmware.
Sempre considerando le tempistiche di sviluppo del progetto, risulta oneroso programmare il firmware con linguaggio macchina (assembler). Rinunciando alle performance, si può programmare il microcontrollore con linguaggi di programmazione ad alto livello. Ogni produttore, infatti, mette a disposizione un software per la programmazione con linguaggi simil-C che vengono poi tradotti in linguaggio macchina attraverso compilatori. Questi software contengono spesso librerie che implementano funzioni standard per la gestione del dispositivo.

3.2.1 Architettura del microcontrollore

Il primo passo verso la scelta del microcontrollore è studiarne la struttura interna per poter comprendere quali sono le specifiche tecniche utili al progetto che si intende sviluppare. La tipica architettura di un microcontrollore è descritta in Figura 3.2 dove si possono individuare tre blocchi principali:

- CPU: rappresenta il cuore del microcontrollore, dove sono implementati i registri di controllo e le unità di calcolo. Al giorno d'oggi, quasi l'80% dei dispositivi (microcontrollore) sono basati su architettura Arm Cortex, e ciò garantisce la possibilità di reperire facilmente documentazione e guide su gestione e utilizzo.
  Le principali specifiche che caratterizzano la CPU sono numero di bit del set di istruzioni, registri e frequenza del clock interno, che definisce il numero di istruzioni eseguite in un secondo.

- Memorie: statiche e dinamiche. La dimensione delle memorie definisce la lunghezza massima del firmware e la quantità di dati che possono essere memorizzati.

- Periferiche: moduli di comunicazione (UART, SPI, I2C), timer, generatori di segnale PWM, ADC e general pourpose input/output (GPIO), che sono gestiti mediante appositi registri nel blocco memoria configurabili dalla CPU. La presenza di periferiche permette al chip di interagire con il mondo esterno, ed è uno degli aspetti che differenziano un microcontrollore da un microprocessore, garantendone autosufficienza funzionale. Quantità e specifiche di ogni singola periferica caratterizzano il microcontrollore indirizzandolo verso applicazioni dedicate. Un microcontrollore con un elevato numero di moduli PWM, per esempio, è indicato per applicazioni di automazione e controllo.

Per scegliere quale sia il microcontrollore adatto per ogni specifico progetto, bisogna capire quali sono le risorse necessarie, quanta potenza di calcolo è richiesta, quanti dati devono essere immagazzinati ma, soprattutto, numero e tipo di periferiche che verranno utilizzate.

Inoltre, è molto importante scegliere un dispositivo provvisto di un buon software di programmazione con una libreria per la gestione di periferiche e CPU.

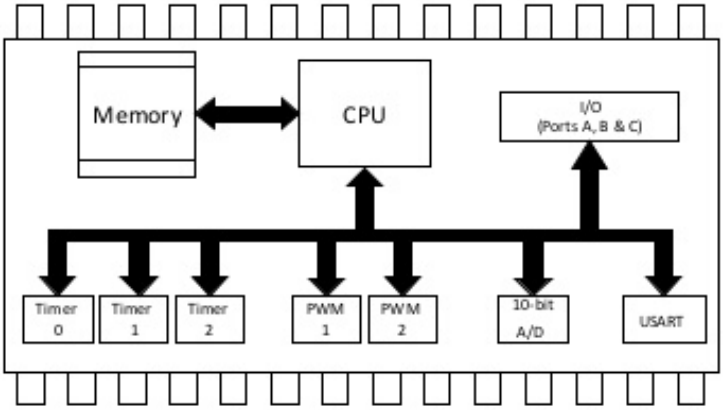


Figura 3.2: Tipica architettura microcontrollore.

3.2.2 Scelta del microcontrollore e della development board

Considerando quanto visto in precedenza, il parametro principale nella scelta del microcontrollore è il modulo SPI, che deve garantire la massima personalizzazione dei parametri che caratterizzano la comunicazione, in linea con le richieste di tab. 2.1. In secondo luogo occorre tener conto dei moduli PWM e GPIO che agiscono direttamente sulle commutazioni dei pin digitali. Come già ribadito, la presenza di un buon software di programmazione con relativa libreria di gestione del microcontrollore e una buona documentazione sono essenziali per agevolare lo sviluppo del firmware. Infine, bisogna tener conto delle specifiche tecniche dell'evaluation board, in particolare il numero di pin accessibili e la presenza di un debugger on-board.
Visto l'ambito di applicazione e i requisti di progetto, si è deciso di valutare dispositivi realizzati dai principali produttori, selezionando i migliori in termini di prestazioni, con particolare attenzione a: frequenza di clock, numero di bit del set di istruzioni, memoria dati, memoria ram, ma soprattutto tipo e numero di periferiche (SPI, GPIO, PWM).
Si sono così individuati quattro differenti prodotti da confrontare tra loro (Figura 3.3).

Microcontrollore	Microchip PIC32MZ2048ECH144	NXP LPC4327JBD144	ST STM32F769AI	Infineon XMC4700-F144K2048AA
bit	32	32(ARM)	32(ARM)	32(ARM)
f(MHz)	200	200	216	144
Ram(KB)	512	136	512	352
Progaram memory(KB)	2048	1024	2048	2048
pin GPIO	120	80	160	119
Internal Oscillator	8MHz,32KHz		16MHz,32KHz	24MHz,32KHz
Moduli PWM	9		4	16
Modulo SPI	Standard 8-16 bit	Standard 8-16 bit	Standard 8-16 bit	Configurabile
Clock Managment	PLL	PLL	PLL	PLL
USB	HS Device/Host/OTG	HS Device/Host/OTG	HS Device/Host/OTG	FULL SPEED/Host/OTG
Package	LQFP,TQFP	LQFP	LQFP	LQFP
Software	MPLAB	LPCOpen	COLDE	DAVE
Materiale di supporto	Ottimo			
Alimentazione(V)	3,3	3,3	3,3	3,3
Evaluation Board				

Figura 3.3: Riassunto caratteristiche principali dei microcontrollori.

Dopo un'attenta analisi dei dispositivi, e lo studio dei loro datasheet si è scelto di implementare l'interfaccia utilizzando l'evaluation board KIT XMC47_RELAX_V1[8] basato su microcontrollore XMC4700 [9] di Infineon Technologies.
La presenza di un particolare modulo SPI altamente configurabile e di un elevato numero di moduli PWM rende questo dispositivo il più adatto per questo progetto.
Il modulo SPI di questa famiglia di microcontrollori (XMC) permette la completa configurazione di: lunghezza del numero di bit, frequenza CLK, polarità di CLK e CS, tid, tfd.
Le altre soluzioni analizzate utilizzano invece moduli di comunicazione SPI standard, in cui la lunghezza dei comandi è fissa (8 o 16 bit) e le frequenza di CLK può variare tra pochi valori predefiniti.
XMC4700 è un prodotto pensato per applicazioni automotive, e di conseguenza fa dei moduli di comunicazione il suo punto di forza. Molte applicazioni, quali appunto LED driving, richiedono il continuo dialogo tra dispositivi e microcontrollore, rendendo necessario un modulo di comunicazione flessibile che si adatti a ogni configurazione possibile di SPI. Inoltre, la famiglia di microcontrollori XMC è provvista della piattaforma di sviluppo DAVE^{TM} che, grazie ad un compilatore, permette la progrmmazione del codice in linguaggio simil-C ed è provvisto della libreria XMC_{LIB} per la gestione della CPU e delle periferiche.

3.2.3 Development board XMC47_RELAX_V1

Il microcontrollore scelto (XMC4700 Infineon Technologies)[9] e in particolare i moduli SPI, PWM e GPIO, verranno presentati in dettaglio nel prossimo capitolo. In questa sezione, invece, viene analizzata l'evaluation board, evidenziando, le principali caratteristiche [1].

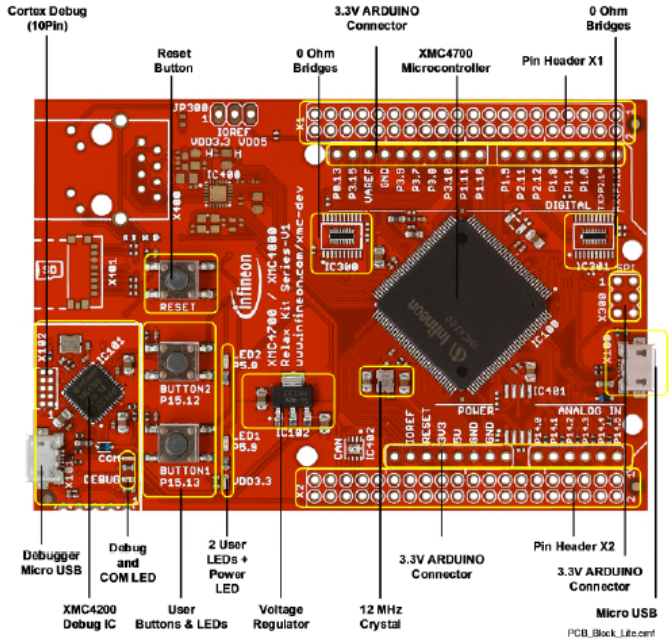


Figura 3.4: Board XMC47_RELAX_V1.

- on-board debugger: permette la comunicazione tra la piattaforma di sviluppo DAVE™ e il microcontrollore attraverso connessione USB (protocollo UART). Sfruttando il driver Segger's Jlink, si può facilmente caricare il firmware creato con DAVE™ e inoltre sono implementate funzioni di debug;

- XMC4700 o XMC4800: sono i microcontrollori che possono essere montati sulla board;

- pin header 40 bit: permette l'accesso ai pin più significativi del controllore per facilitare eventuali connessioni;

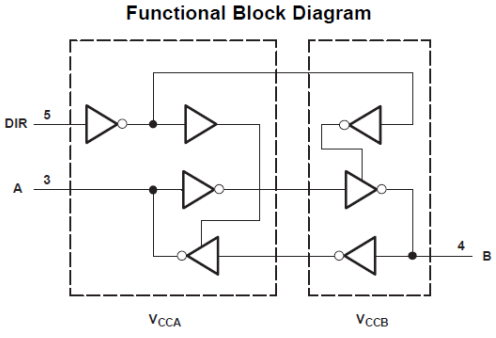
- on board power generation: trasformazione e distribuzione della tensione in ingresso alle porte USB (5V-3,3V);

- connettori micro USB: permettono di ottenere l'alimentazione (+5V) e la connessione fisica per comunicazioni basati su USB. La porta di connessione Debugguer Micro USB in Figura 3.4 permette la connessione tra debugger e l'ambiente di sviluppo.

[1] Per ulteriori informazioni fare riferimento al file: EvaluationBoard For XMC4000 Family, Board User's Manual[8].

3.3 Voltage level shifter

Sebbene il focus principale del progetto sia lo sviluppo della parte software (firmware e driver), in questa fase sono state soddisfatte anche le richieste hardware alcune delle quali opzionali. L'evaluation board non soddisfa i requisiti di possibilità di disconnessione fisica delle linee, protezione dell'interfaccia da corto circuiti e sovratensioni, e soprattutto non garantisce la variazione della tensione di uscita delle linee, che invece sono fisse a 3,3V.
La disconnessione fisica può essere implementata tramite appositi relè mentre, per variare la tensione di I/O si è scelto di utilizzare un voltage level shifter, ovvero un dispositivo usato in circuiti digitali con multiple tensioni di lavoro che permette la comunicazione tra domini di tensione diversi cambiando la tensione del livello logico dei segnali [11].
In questo caso si è scelto di usare il voltage level shifter SN74LVC1T45 (Texas Instrument) [12], in Figura 3.5(a). Questo dispositivo utilizza due canali di alimentazione separati e configurabili. La porta A lavora in base alla tensione VCCA, che può variare tra 1,65V e 5,5V. La porta B lavora in base alla tensione VCCB e anch'essa puo varaiare tra 1,65V e 5,5V. Il pin di direction control DIR attiva la porta A oppure B come uscita. Il dispositivo trasmette dati dalla porta A alla porta B se la porta B è impostata come output altrimenti trasmette dati dalla porta B alla porta A se la porta A è impostata come output. La particolarità di questo prodotto è la possibilità di traslare il livello logico di ingresso in un livello logico con tensione maggiore ma soprattutto minore rispetto a quella d'ingresso.
La porte di I/O del level shifter collegate al microcontrollore devono essere alimentate a 3,3V, mentre la porta collegata al DUT può assumere tensione arbitraria. In questo modo possono essere caratterizzati i livelli di tensione dei vari ingressi.
Relè e level shifter possono essere montati direttamente sulla board di caratterizzazione, oppure può essere prevista un'ulteriore board di interfacciamento. È compito dell'utilizzatore decidere come e quando impiegare level shifter e relè.
In Figura 3.5(b) sono riportati i tempi di switching per tensione di ingresso 3,3V.



(a) Struttura interna, SN74LVC1T45.

7.8 Switching Characteristics (V_{CCA} = 3.3 V ± 0.3 V)

over recommended operating free-air temperature range, V_{CCA} = 3.3 V ± 0.3 V (see Figure 9)

PARAMETER	FROM (INPUT)	TO (OUTPUT)	V_{CCB} = 1.8 V ±0.15 V		V_{CCB} = 2.5 V ±0.2 V		V_{CCB} = 3.3 V ±0.3 V		V_{CCB} = 5 V ±0.5 V		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	
t_{PLH}	A	B	2.1	15.5	1.4	8	0.7	5.8	0.7	4.4	ns
t_{PHL}			2	12.6	1.3	7	0.8	5	0.7	4	
t_{PLH}	B	A	1.7	8.3	1.3	6.4	0.7	5.8	0.6	5.4	ns
t_{PHL}			1.8	7.1	1.3	5.4	0.8	5	0.7	4.5	
t_{PHZ}	DIR	A	2.9	7.3	3	7.3	2.8	7.3	3.4	7.3	ns
t_{PLZ}			1.8	5.6	1.6	5.6	2.2	5.7	2.2	5.7	
t_{PHZ}	DIR	B	5.4	20.5	3.9	10.1	2.9	8.8	2.4	6.8	ns
t_{PLZ}			3.3	14.5	2.9	7.8	2.4	7.1	1.7	4.9	
t_{PZH} (1)	DIR	A		22.8		14.2		12.9		10.3	ns
t_{PZL} (1)				27.6		15.5		13.8		11.3	
t_{PZH} (1)	DIR	B		21.1		13.6		11.5		10.1	ns
t_{PZL} (1)				19.9		14.3		12.3		11.3	

(b) Tempi di switching, SN74LVC1T45.

Figura 3.5: Voltage level shifter, SN74LVC1T45, Texas Instrument.

3.4 Interfaccia LVDS

Con il continuo aumentare della banda richiesta per le comunicazioni vi è la necessità di trovare interfacce fisiche sempre più efficienti in termini di velocità di trasmissione, consumo di potenza e riduzione del rumore. Lo standard LVDS è un cosidetto sistema differntial signaling, che trasmette informazioni attraverso la differenza di potenziale di due conduttori [13].
In una tipica implementazione (Figura 3.6) il trasmettitore inietta una corrente nei conduttori e la direzione della corrente determina il livello logico. Se gli ingressi del ricevitore sono ad alta impedenza, la corrente trasmessa passa attraverso la resistenza e genera una differenza di potenziale ai suoi capi. Il ricevitore determina il livello logico a seconda della polarità di questa tensione.
Il vantaggio di LVDS è la riduzione dei disturbi elettromagnetici dovuta al fatto che la corrente scorre nei due conduttori paralleli con verso opposto generando campi elettromagnetici che tendono a cancellarsi. LVDS può quindi essere utile per comunicazioni seriali, qualora vengano generate comunicazioni ad alte frequenze, soprattutto se i chip sono distanti tra di loro, per garantire l'integrtità del segnale.

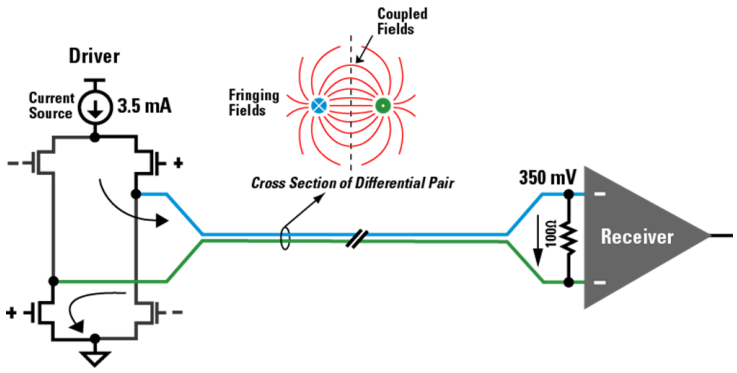
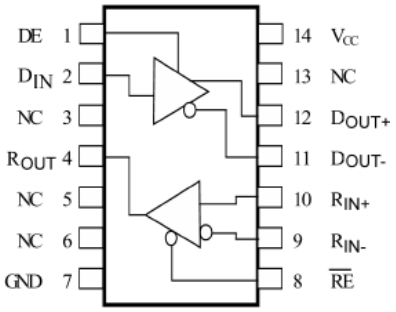


Figura 3.6: Tipica struttura di trasmissione/ricezione LVDS.

Purtroppo il microcontrollore scelto non dispone di un modulo integrato LVDS. È comunque sufficiente utilizzare dispositivi Driver/Receiver come FIN1019 di ON Semiconductor[14] (Figura 3.7(a)) che ha il compito di tradurre classici segnali TTL in tipici segnali LVDS (driver) e viceversa (receiver).
Ad esempio, nel caso del segnale CLK di una classica comunicazione SPI possiamo utilizzare due dispositivi FIN1019. Il primo dispositivo (driver) è utilizzato connettendo l'ingresso DIN direttamente al segnale CLK in uscita dal microcontrollore (Master). Il secondo dispositivo (receiver), invece, prevede la connesione del pin di uscita Rout all'ingresso del segnale di clock dello Slave. Connettendo tra loro i pin Dout+, Rin+ e Dout-, Rin- si ottiene una completa trasmissione LVDS.
Anche in questo caso è compito dell'utilizzatore prevedere l'inserimento delle interfacce LVDS nella board di caratterizzazione o in un'apposita board di interfacciamento.



(a) Struttura interna FIN1019.

Pin Name	Description
D_{IN}	LVTTL Data Input
D_{OUT+}	Non-inverting LVDS Output
D_{OUT-}	Inverting LVDS Output
DE	Driver Enable (LVTTL, Active HIGH)
R_{IN+}	Non-Inverting LVDS Input
R_{IN-}	Inverting LVDS Input
R_{OUT}	LVTTL Receiver Output
$\overline{RE}$	Receiver Enable (LVTTL, Active LOW)
V_{CC}	Power Supply
GND	Ground
NC	No Connect

(b) Nomeclatura pin FIN1019.

Figura 3.7: Modulo Driver/Receiver LVDS, FIN1019, ON Semiconductor.

Capitolo 4

Firmware

In questo capitolo sono trattate le soluzioni firmware e una panoramica delle funzionalità del microcontrollore e dei suoi moduli. In particolare sono descritte le interazioni tra questi moduli e la CPU che permettono di sincronizzare commutazioni digitali e comunicazioni seriali. Nella parte finale è descritto l'apposito protocollo di scambio dati progettato per la comunicazione con il driver.

4.1 Microcontrollore XMC4700

La scelta di utilizzare il microcontrollore XMC4700 è stata fortemente influenzata dalla presenza di un elevato numero di moduli PWM (CCU8) e del modulo di comunicazione seriale (USIC) che permette di configurazione buona parte dei parametri richiesti dalle specifiche. L'ambiente di sviluppo DAVE^{TM} assicura un approccio semplice e intuitivo alla programmazione, e la libreria XMC_{lib} fornisce all'utente le principali funzioni di gestione delle periferiche e della CPU.
Nei prossimi paragrafi sono descritte in dettaglio solamente le caratteristiche e le funzionalità delle periferiche e della CPU utili allo sviluppo dell'interfaccia[1].
XMC4700 è un dispositivo della famiglia di microcontrollori XMC4000 basati sull'architettura ARM Cortex-M4 (Figura 4.1), e le sue caratteristche prinicipali sono:

- set di istruzioni Thumb2 16-bit e 32-bit;
- frequenza di clock 144MHz;
- Nested Vectored Interrupt Controller (NVIC): sistema programmabile di gestione degli iunterrupt. Prevede 64 nodi di interrupt e nodi dedicati alle più importanti sorgenti di interrupt;
- 2048 KB flash memory per il firmware e per la memorizzazione di dati;
- USB 2.0 full-Speed;
- 6 moduli Unversal Serial Interface Channel (USIC): un'interfaccia flessibile per implementare molti tipi di comunicazione seriale, come UART, SPI, I2C;
- 4 ADC;
- 2 DAC;
- 2 Capture/Compare Units 8 (CCU8) per la generazione di PWM;

[1]Per avere maggiori informazioni, si invita il lettore alla consultazione del manuale di utilizzo del microcontrollore [10].

- 2 Capture/Compare Units 4 (CCU4) per la generazione di PWM;
- porte programmabili con accesso ai singoli bit.

La CPU gestisce e configura le periferiche impostando gli appositi registri[2].

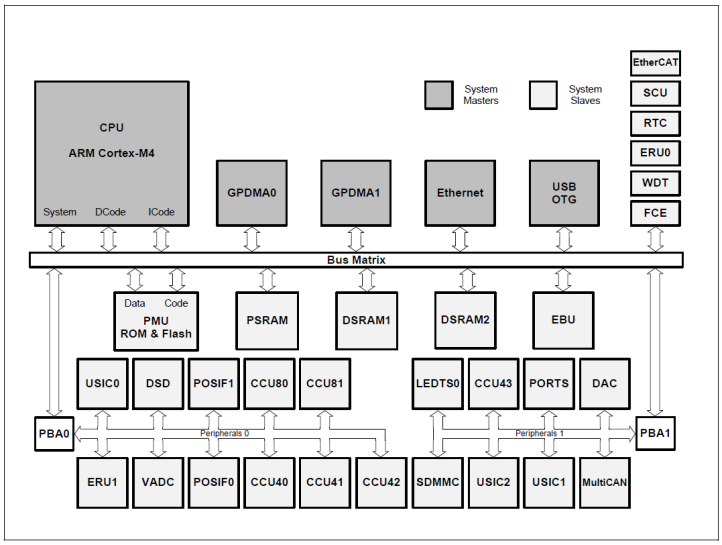


Figura 4.1: Architettura XMC4700.

4.2 Modulo universale per comunicazioni seriali: USIC

USIC è un modulo flessibile per la generazione di comunicazioni seriali, composto da due canali indipendenti (USICx_CH0 e USICx_CH1) che può essere programmato run-time definendo il protocollo e pin di I/O.
Tra i vari protocolli gestiti da USIC troviamo SSC/SPI, che permette di inviare comandi con più di 64 bit, gestendo il trasferimento dei dati con eventi di trigger esterni al modulo per temporizzare la trasmissione. Ciò può avvenire, per esempio, tramite un pin di input o un modulo timer.
Il modulo USIC (Figura 4.2) contiene due canali, ognuno dei quali provvisto di: unità di data shift e buffering, pre processore per la scelta del protocollo, baud rate generator e, opzionalmente, un FIFO data buffer.

[2]In questo documento vengono usati i nomi dei registri dei registri come definito dallo standard CMSIS.

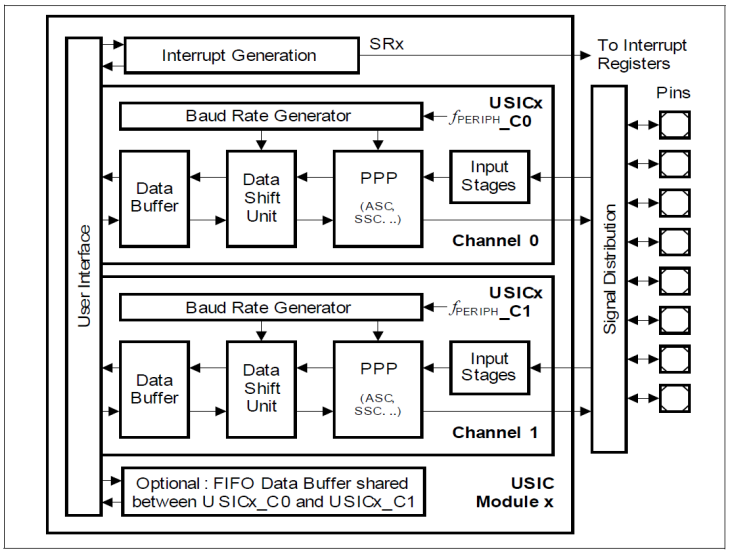


Figura 4.2: Modulo USIC.

4.2.1 Baud Rate Generator

È la struttura per la generazione dei segnali di sincronizzazione per i vari protocolli di comunicazione, composto da (Figura 4.3(a)):

- Fractional Divider (Figura 4.3(a)): permette di generare un segnale di clock a frequenza f_{FD} partendo dal segnale di ingresso del modulo $f_{PERHIP} = 144MHz$ e dividendolo per un fattore n (Normal divider mode) o moltiplicandolo per un fattore n/1024 (Fractional divider mode),

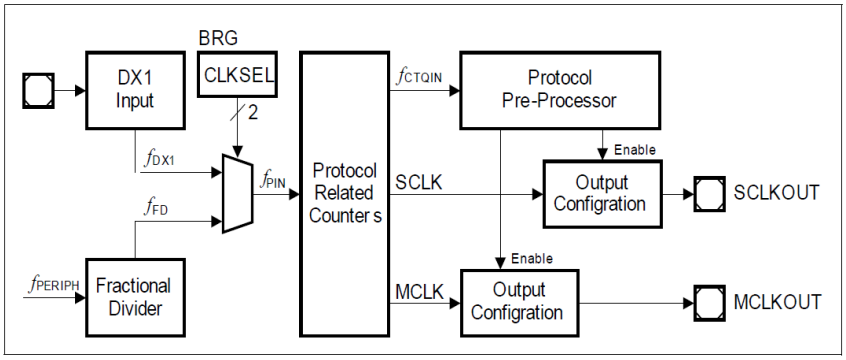
  - si ottiene Normal diveder mode impostando il registro FDR.DM=01 e in questo caso $f_{FD} = f_{PERIPH} \times \frac{1}{n}$, con n=1024-FDR.STEP;
  - si ottiene Fractional divider mode impostando FDR.DM=10 mentre in questo caso $f_{FD} = f_{PERIPH} \times \frac{n}{1024}$, con n=FDR.STEP.

- Divider mode counter (Figura 4.3(b)): è uno dei due contatori presenti nel blocco protocol-related counters (Figura 4.3(a)). In base al protocollo usato provvede alla generazione dei segnali di master clock MCLK e shift clock SCLK ed è controllato attraverso il registro BRG.
  Il segnale di ingresso al counter f_{PIN} può essere scelto mediante l'impostazione di BRG.CLKSEL tra f_{FD} e un segnale esterno proveniente da un pin di ingresso f_{DX1}.
  Un divisore per due impone la generazione del segnale $f_{MCLK} = \frac{f_{PIN}}{2}$.
  Successivamente viene generato il segnale f_{PDIV} dividendo grazie al registro PDIV il segnale di ingresso scelto tra f_{MCLK} e f_{PIN}, per mezzo del registro BRG.PPPEN.

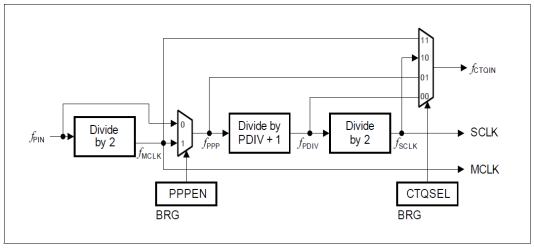
  - se PPPEN=0: $f_{PDIV} = f_{PIN} \times \frac{1}{PDIV+1}$;
  - se PPPEN=1: $f_{PDIV} = f_{MCLK} \times \frac{1}{PDIV+1}$.

  Infine, con un ultimo divisore per 2 si ottiene: $f_{SCLK} = \frac{f_{PDIV}}{2}$.

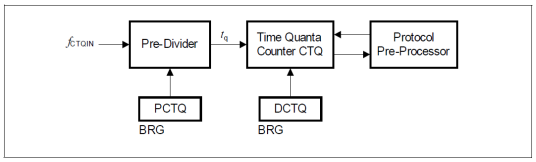
- Time quanta counter (Figura 4.3(c)): è l'altro contatore del blocco protocol-related counters associato al pre processore di scelta del protocollo che definisce le specifiche temporizzazioni dei protocolli. Come vedremo in seguito (sez. 4.3.3), nel protocollo SSC/SPI, time quanta counter gestisce in particolare tid e tfd. Time quanta counter permette di generare intervalli di tempo utilizzati dai vari protocolli per specifiche applicazioni. Attraverso il registro BRG.PCTQ, viene impostato un divisore che genera il segnale t_q a partire dal segnale di ingresso f_{CTQIN}, che può essere scelto tra f_{MCLK}, f_{SCLK}, f_{PPP} e f_{PDIV} grazie al registro BRG.CTQSEL.
  Un counter impostabile grazie al registro BRG.DCTQ permette la generazione di intervalli temporali multipli di t_q.
  Ogni protocollo di comunicazione può utilizzare questo timer per gestire le temporizzazioni specifiche dei segnali.



(a) Panoramica.



(b) Diveder mode counter.



(c) Time quanta counter.

Figura 4.3: Baudrate generator.

4.2.2 Data shift e buffering

L'unità di datashift e buffering è composta da shift registers a 16-bit (TSR) e un buffer con registri a 16-bit (TBUF).
I parametri del protocollo, come lunghezza del comando e direzione dei shift registers, sono controllati mediante il registro di controllo SCTR. Tramite il registro di controllo TCSR, invece, sono gestiti i dati da inviare ed è monitorato lo stato della comunicazione. Cambiamenti dei segnali di output legati agli shift register avvengono solamente in corrispondenza di fronti del segnale di clock, e il livello logico dell'ultimo bit inviato è mantenuto costante fino al successivo fronte di clock (SCLK). L'unità di data shift e buffering si occupa di:

- trasmissione e ricezione dei dati con shift registers (TSR);
- memorizzazione dei dati in attesa di essere inviati nei registri di buffering TBUF. Anche i dati in ricezione vengono memorizzati nei registri di buffering TBUF in attesa di essere gestiti dalla CPU;
- triggering della trasmissione con eventi esterni;
- interfacciamento con la CPU per la gestione dei dati con eventi di interrupt o segnali di controllo generati della comunicazione.

Sono di particolare interesse i campi word length control (WLEMD) e frame length control (FLEMD) del registro TCSR, che gestiscono le lunghezze dei comandi da inviare. In generale, un frame (comando) è composto da una o più parole (word):

- se TCSR.WLED=1: attraverso il campo SCTR.WLE si può impostare la lunghezza delle parole che costituiscono un comando tra 1-bit e 16-bit;
- se TCSR.FLEMD=1: attraverso il campo SCTR.FLE si può impostare la lunghezza del comando. Generalmente un comando è composto da una o più parole. Analizzeremo in (sez. 4.3.1) l'importante ruolo dei registri SCTR.WLE e SCTR.FLE per la definizione delle lunghezze dei comandi nel protocollo di comunicazione SSC/SPI.

Fondamentale è lo schema di validazione dei dati da inviare (Figura 4.4), che permette il triggering della trasmissione con eventi esterni al modulo.
Per esempio, è possibile condizionare l'invio di un comando rispetto al valore di una linea in ingresso a un pin di input. Questa caratteristica risulterà essenziale per risolvere il problema della sincronizzazione tra comandi SPI e linee digitali.
Una parola da trasmettere è contenuta nel registro di buffering TBUF e il suo invio è condizionato dal bit di validazione TCSR.TDV. Una combinazione di eventi e condizioni programmabili attraverso hardware o software definiscono il valore di TCSR.TDV, ovvero definiscono quando un messaggio può essere inviato. Esiste anche una logica di validazione dell'invio gestita dal registro TCSR che comprende segnali di ingresso: DX2T e DX2S. Per esempio, attraverso i registri TCSR.TDEN=11 e TCSR.TDSSM=0 possiamo condizionare l'invio delle parole al valore del segnale di ingresso DX2S=1 e al valore di TCSR.TDV=1.

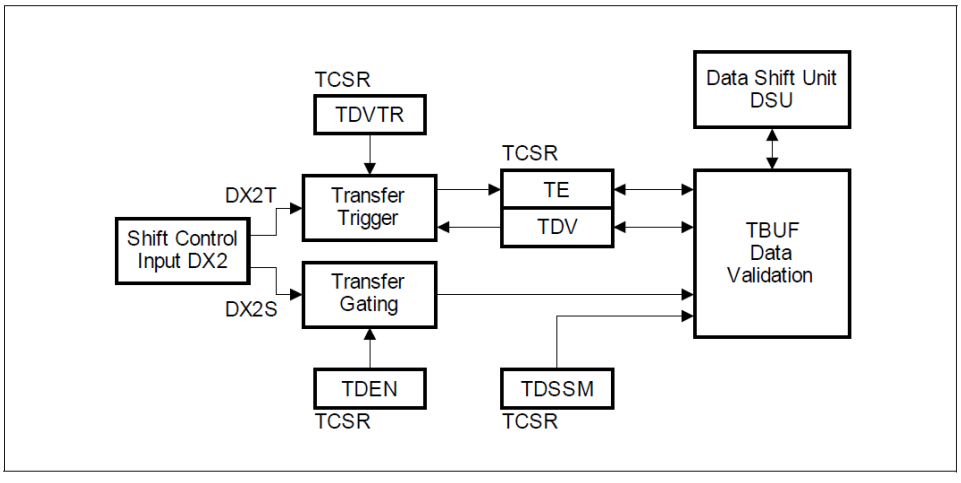


Figura 4.4: Schema di validazione della trasmissione.

4.3 Protocollo di comunicazione SPI: SSC

SSC è un protocollo di comunicazione del modulo USIC per implementare comunicazioni SPI, che può gestire ricezione e trasmissione sincronizzate di comandi tra un Master e uno o più Slave. Come visto in sez. 2.1.1, SSC provvede alla generazione di una tipica comunicazione SPI, composta da: un segnale di clock generato dal Master (CLK), due segnali per lo scambio dei dati (MOSI e MISO) e un segnale di inizio e fine comunicazione CS, anch'esso generato dal Master.
Il modulo USIC permette l'implementazione del protocollo SPI come Master, ovvero occupandosi della generazione di CLK e CS, e anche come Slave. Per il progetto dell'interfaccia saranno utilizzate solamente interfacce configurate come Master.

4.3.1 Lunghezza dei comandi

In una comunicazione SSC, un comando (frame) è composto da una serie consecutiva di parole (word) che possono essere eventualmente separate tra di loro da un cosiddetto inter-word delay. Senza inter-word delay, le varie parole consecutive si uniscono, formando una singola parola più lunga detta appunto comando.
La lunghezza delle parole e dei comandi sono gestite dai registri SCTR.WLE e SCTR.FLE visti in sez. 4.2, e può essere definita in due diversi modi:

- qualora si vogliano inviare comandi con meno di 64 bit, si può impostare la lunghezza del comando tramite SCTR.FLE. Appena è stato inviato il numero di bit definito nel registro SCTR.FLE, la comunicazione termina.
  Per esempio, impostando lunghezza delle parole 8-bit SCTR.WLE=8 e lunghezza del frame 12-bit SCTR.FLE=12, nel buffer vengono inserite due parole da 8-bit ciascuna. Dopo l'invio del quarto bit della seconda parola, ovvero del 12-bit del comando, la comunicazione si interrompe poiché è stata raggiunta la lunghezza di bit definita dal registro SCTR.FLE.

- qualora si vogliano inviare comandi con più di 64 bit, si può impostare la lunghezza del messaggio da inviare direttamente attraverso i registri di buffer TBUF. Impostando SCTR.FLE=63 e il bit PCR.FEM=1, si assume che la fine di un comando avvenga quando nei registri TBUF non sono presenti dati da inviare.
  Per esempio, impostando SCTR.FLE=63 e SCTR.WLE=8 e utilizzando la struttura FIFO per riempire TBUF, possiamo inviare 33 parole ognuna delle quali a 8-bit. Una volta inviate tutte e 33 le parole, la comunicazione termina poiché la struttura FIFO è vuota e quindi il buffer non contiene più dati. In questo modo possiamo generare un comando di lunghezza 264-bit. Questa modalità è utile quando abbiamo molti Slave SPI collegati assieme in daisy chain mode. Inoltre, sfruttando questa caratteristica, possiamo utilizzare il protocollo SSC per la generazione dei comandi SCI[3], utilizzando CLK e MOSI (data) di una comunicazione SPI con 264-bit, simile a quella appena vista in esempio. Per implementare una completa comunicazione SCI bisogna generare anche il segnale SYNC, come vedremo in dettaglio nelle prossime sezioni.

[3]SCI può essere generato applicando qualche modifica alla comunicazione SPI.

4.3.2 Baud rate generator e data shift unit

La baud rate della comunicazione determina la lunghezza temporale dei singoli bit, ossia la frequenza di lavoro della comunicazione seriale.
Il segnale di sincronizzazione utilizzato dal protocollo SSC per la generazione del CLK è il segnale SCLK con frequenza f_{SCLK} generato dal baud rate generator del modulo USIC descritto in sez. 4.2.
Vista la moltitudine di configurazioni possibili per il segnale di sincronizzazione CLK, esiste l'unità data shift che si occupa di generare il vero e proprio segnale di clock inviato dal Master allo Slave (SCLKOUT in Figura 4.5) e che gestisce gli istanti di scrittura e lettura delle linee dati. Attraverso il registro BRG.SCLKCFG è possibile implementare le quattro configurazioni di SPI caratterizzate dai parametri CPOL e CPHA viste in sez. 2.1.1:

- BRG.SCLKCGFG=00: livello logico passivo del SCLKOUT=0. La scrittura del primo bit avviene al primo fronte di salita di SCLKOUT e la lettura avviene al primo fronte di discesa (CPOL=0, CPHA=1);

- BRG.SCLKCGFG=01: livello logico passivo del SCLKOUT=1. La scrittura del primo bit avviene al primo fronte di discesa di SCLKOUT e la lettura avviene al primo fronte di salita (CPOL=1, CPHA=1);

- BRG.SCLKCGFG=10: livello logico passivo del SCLKOUT=0. La scrittura del primo bit avviene 1/2 periodo in anticipo rispetto al primo fronte di salita di SCLKOUT e la lettura avviene 1/2 periodo in anticipo rispetto al primo fronte di discesa (CPOL=0, CPHA=0);

- BRG.SCLKCGFG=11: livello logico passivo del SCLKOUT=1. La scrittura del primo bit avviene 1/2 periodo in anticipo rispetto al fronte di discesa di SCLKOUT e la lettura avviene 1/2 periodo in anticipo rispetto al primo fronte di salita (CPOL=1, CPHA=0).

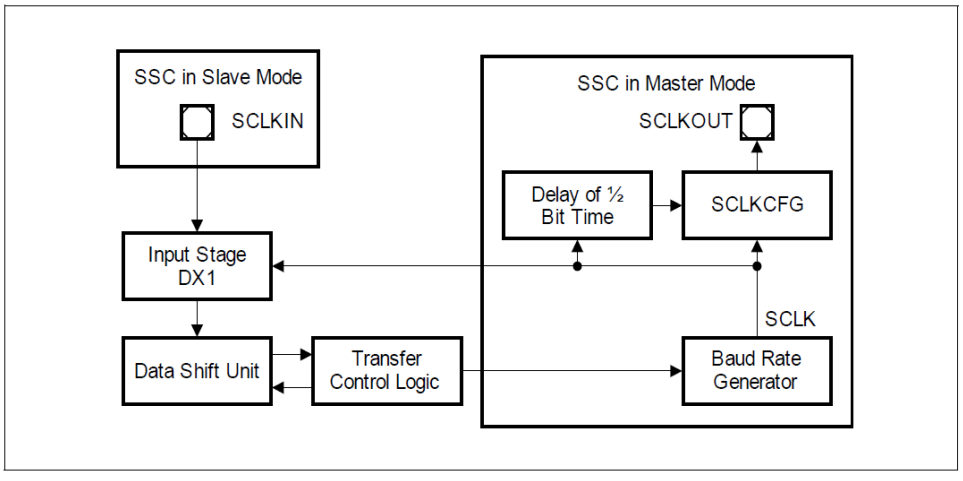


Figura 4.5: Data shift unit.

4.3.3 CS delay generation

Il segnale CS determina inizio e fine della comunicazione SPI. Come mostrato in Figura 4.6, l'inizio della trasmissione dei dati e del segnale di CLK ha inizio un tempo programmabile *tid* dopo l'attivazione dello slave, necessario per preparare il dispositivo alla comunicazione. Se più comandi sono inviati nella stessa istanza di comunicazione, possiamo gestire la distanza tra i comandi con un tempo programmabile tnf.
Se un comando è composto da più di una parola possiamo gestire la distanza tra di esse attraverso il tempo programmabile *tiw* impostato nel registro PCR.TIWEN. Se PCR.TIWEN=0 le parole si uniscono formando un unica parola.
Tutti questi tempi sono gestiti dal time quanta counter, sono programmabili e multipli di un quanto di tempo definito dal periodo del segnale di ingresso al counter: f_{CTQIN}.

- $tid = tfd$, hanno la stessa lunghezza che può essere programmata attraverso i seguenti registri:
  - BRG.CTQSEL per definire la frequenza di input f_{CTQIN};
  - BRG.PCTQ per definire la lunghezza di un singolo quanto: divisione di f_{CTQIN} per 1, 2, 3 o 4;
  - BRG.DCTQ per definire il numero di quanti temporali: da 1 a 32.
- $tnf = tiw$, hanno la stessa lunghezza ma *tiw* può essere disabilitato. Per la loro programmazione devono essere impostati i seguenti registri:
  - PCR.CTQSEL1 per definire la frequenza di input f_{CTQIN};
  - PCR.PCTQ1 per definire la lunghezza di un singolo quanto: divisione di f_{CTQIN} per 1, 2, 3 o 4;
  - PCR.DCTQ1 per definire il numero di quanti temporali: da 1 a 32;
  - PCR.TIWEN per abilitare o disabilitare *tiw*.

Grazie alla flessibilità sulla scelta della frequenza di input f_{CTQIN}, possiamo avere tempi molto corti (inferiori alla lunghezza di un singolo bit) oppure molto lunghi (fino a 128 volte la lunghezza di un singolo bit). I ritardi vengono calcolati come segue:

$$tid = tfd = \frac{(PCTQ + 1) \times (DCTQ + 1)}{f_{CTQIN}}$$

$$tnd = tiw = \frac{(PCTQ1 + 1) \times (DCTQ1 + 1)}{f_{CTQIN}}$$

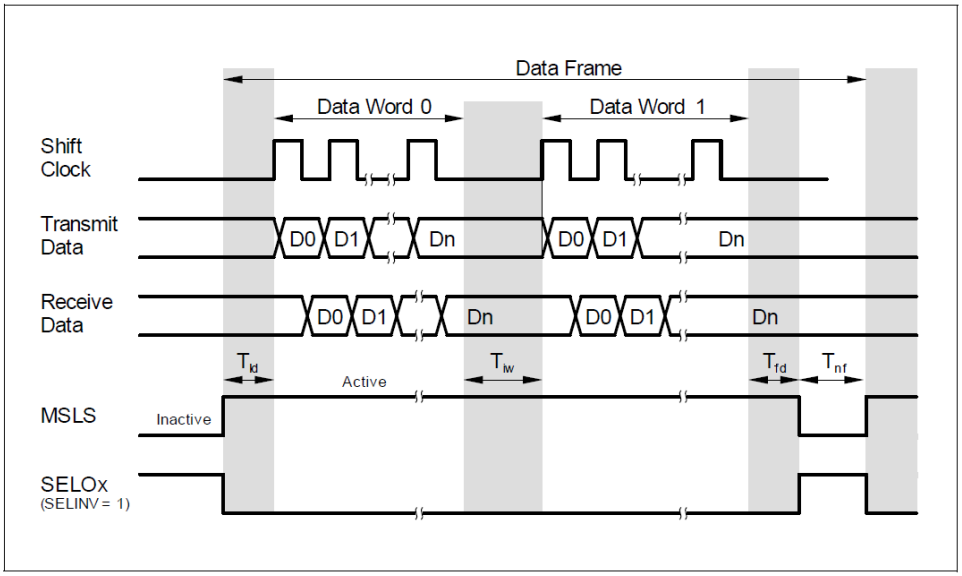


Figura 4.6: Ritardi programmabili USIC.SSC.

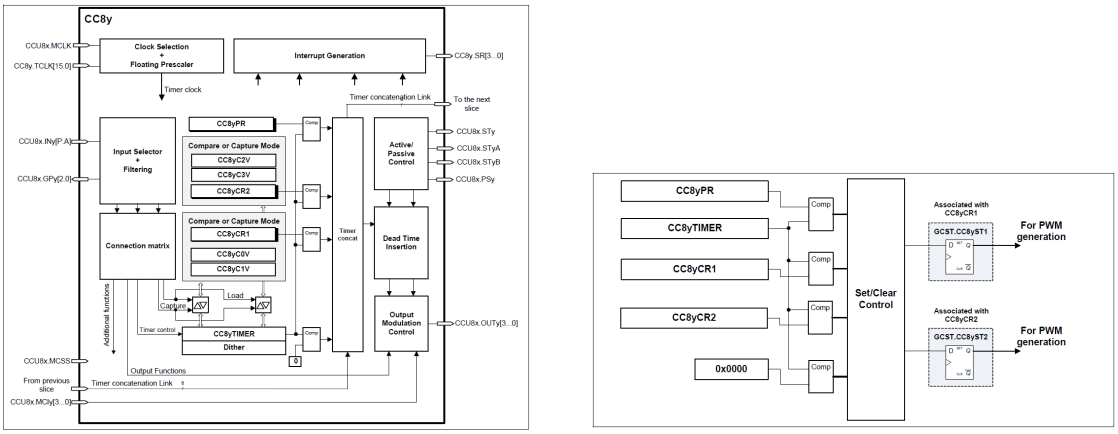
4.4 Moduli CCU8: PWM

Il modulo Capture/Compare CCU8 permette la generazione di complessi segnali PWM. Il microcontrollore XMC4700 dispone di due istanze del modulo CCU8x, ognuna delle quali dispone di quattro timer a 16-bit CC8y. In compare mode ogni timer dispone di due canali di comparazione (compare channel) che permettono la generazione di due segnali PWM. In totale, quindi, si possono avere 16 distinte PWM.
Le principali caratteristiche del modulo sono:

- timer a 16 bit;
- registri per l'aggiornamento dinamico del periodo e dei canali di comparazione (shadow transfer);
- generazione PWM simmetriche e asimmetriche;
- tre differenti modalità: cenetr aligned, edge aligned, single shot;
- periodo e duty cycle delle PWM programmabile;
- start e stop delle PWM controllabile da eventi esterni.

Il cuore del timer è costituito da un contatore a 16-bit, un registro del periodo e due canali di comparazione. Il clock può essere selezionato individualmente per ogni timer del modulo (Figura 4.7(a)).
Il registro CC8yPR del periodo (period register) regola il massimo valore a cui può arrivare il contatore, mentre i due comparatori CC8yCR1 e CC8yCR2 (compare register) sono usati per controllare lo stato (attivo/passivo) delle linee di output (Figura 4.7(b)).
In edge aligned mode il contatore viene resettato al valore 0000_{HEX} ogni volta che viene raggiunto il valore definito dal registro di periodo. I registri di periodo e comparazione

possono essere cambiati dinamicamente durante l'esecuzione del programma permettendo la variazione di periodo e duty cycle delle PWM.
Start e stop del timer possono avvenire attraverso comandi di set/clear software oppure attraverso pin programmabili di input.



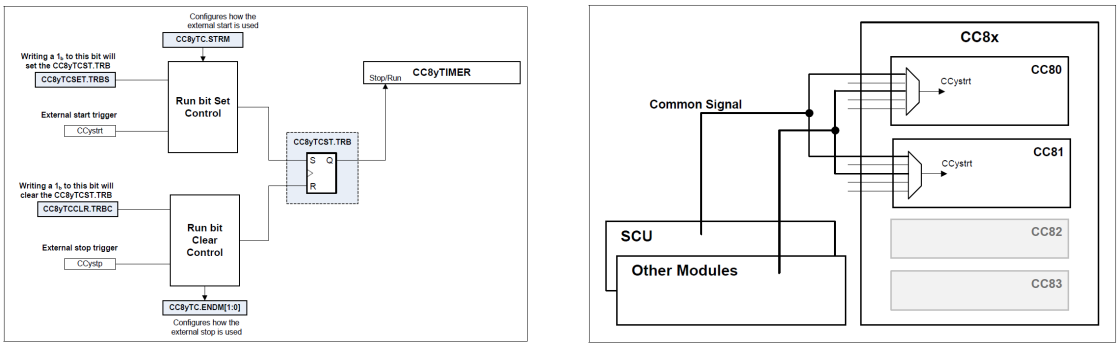
(a) Struttura timer CC8y.

(b) Registri di generazione PWM

Figura 4.7: Timer CC8y.

4.4.1 Start e stop

Start e stop del timer possono avvenire tramite comandi software oppure tramite eventi esterni (hardware) associati ai fronti di salita o discesa dei segnali di ingresso ai pin di input CCystrt e CCystp (Figura 4.8(a)). Selezionare lo start con eventi esterni non implica l'uso di eventi esterni per lo stop, e viceversa.
Il modulo fornisce anche la possibilità di sincronizzare l'avvio e lo stop di tutti i timer dell'unità CCU8 via software. Esiste infatti un pin dedicato che deve essere configurato come evento esterno di start, controllato dalla CPU per mezzo del registro CCUCON e connesso a tutti i timer dell'unità CCU8 (Figura 4.8(b)).



(a) External start/stop.

(b) Sincronizzazione start/stop (CUCCON)

Figura 4.8: Start e stop timer.

4.4.2 Shadow transfer

Ogni canale di comparazione è provvisto di un bit di stato (GCST.CC8yST1 per il canale 1 e GCST.CC8yST2 per il canale 2) che indica lo stato attivo o passivo del canale, e tali valori sono utilizzati per la generazione dei segnali PWM. Il set/clear di questi status bit è imposto dal valore del period register e dei compare register del timer che possono essere aggiornati dinamicamente permettendo l'aggiornamento del periodo e del duty cycle a ogni ciclo della PWM.
Ogni timer è associato ai cosiddetti shadow registers (Figura 4.9), che facilitano l'aggiornamento run time via software dei parametri del segnale PWM (periodo e duty cycle) attraverso la variazione dei registri compare (CC8yCR1, CC8yCR2) e period (CC8yPR). Inoltre, esiste un shadow register che permette di selezionare il livello passivo (CC8yPSL) della linea.
L'aggiornamento di questi registri può avvenire solamente scrivendo i nuovi valori nel shadow register associato al registro da aggiornare e attendere che i nuovi valori vengano trasferiti. Ogni gruppo di shadow registers relativi a un timer possiede un bit GCST.SySS per l'abilitazione del trasferimento dei nuovi valori che deve essere abilitato via software. Questo bit è automaticamente disabilitato via hardware alla fine del trasferimento.
Il trasferimento dei nuovi dati contenuti negli shadow registers dentro ai registri veri e propri può avvenire solamente dopo aver abilitato il bit GCST.SySS e in determinate condizioni:

- nel ciclo di clock successivo a un period match, ovvero quando il contatore del timer raggiunge il valore contenuto nel period register;
- immediatamente, se il timer viene stoppato.

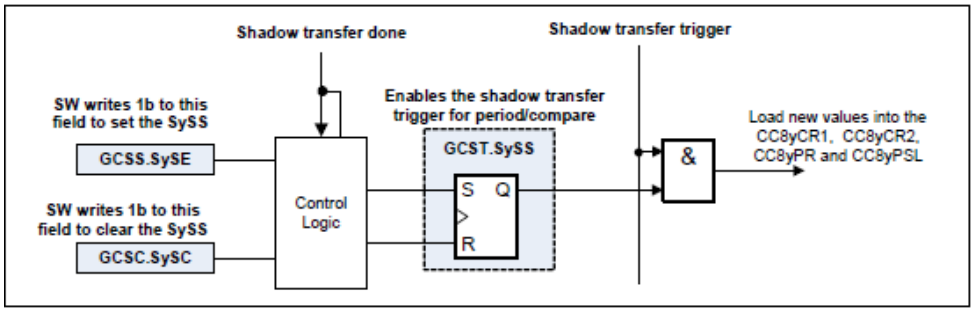


Figura 4.9: Struttura shadow transfer.

4.4.3 Edge Aligned Mode

In questa modalità il timer è incrementato a ogni colpo di clock fino a quando non viene raggiunto il valore impostato nel period register CC8yPR (period match). Una volta raggiunto tale valore il timer viene resettato e ricomincia ad incrementare.
In questa configurazione i valori di period register e compare register possono essere aggiornati via software in corrispondenza di un period match attraverso un shadow transfer. Esistono due configurazioni dell'Edge Aligned Mode:

- Standard Edge Aligned Mode: i due canali di comparazione lavorano indipendentemente e possono essere programmati con differenti valori generando due segnali PWM con stesso periodo e differente duty cycle.
  Il bit di stato GCST.CC8ySTx utilizzato per la generazione delle PWM viene abilitato un ciclo di clock dopo il raggiungimento del valore impostato nel compare register e disabilitato un ciclo di clock dopo l'occorrenza di un period match (Figura 4.10).

- Asymmetrical Edge Aligned Mode: i due canali di comparazione possono essere combinati assieme generando una singola PWM assimetrica. Il canale di uscita GCST.CC8yST2 è disabilitato e permane al valore passivo.
  Il bit di stato GCST.CC8yST1 viene abilitato quando il timer raggiunge il valore del compare1 register (CC8yCR1.CR1) e disabilitato quando viene raggiunto il valore del compare2 register (CC8yCR2.CR2). Quando CC8yCR2.CR2 è programmato con un valore minore rispetto a CC8yCR1.CR1, il bit di stato GCST.CC8yST1 permane disabilitato.

In entrambi i casi, la generazione del segnale PWM in uscita ai pin di output avviene combinando le informazioni contenute nei bit di stato (GCST.CC8yST1) e nel bit passive level (CC8yPSL).
Qualora il livello logico passivo della linea sia impostato a 0 (CC8yPSL=0), l'abilitazione del bit di stato (GCST.CC8ySTx=1) impone al segnale di uscita (PWM) di assumere il valore logico 1. La disabilitazione (GCST.CC8ySTx=0), invece, impone alla linea lo stato logico 0.
Viceversa, qualora il livello logico passivo della linea sia impostato a 1 (CC8yPSL=1), l'abilitazione del bit di stato (GCST.CC8ySTx=1) impone al segnale di uscita (PWM) di assumere il valore logico 0. La disabilitazione (GCST.CC8ySTx=0), invece, impone lo stato logico 1. In questo modo, si possono ottenere due segnali PWM complementari solamente cambiando il registro CC8yPSL.

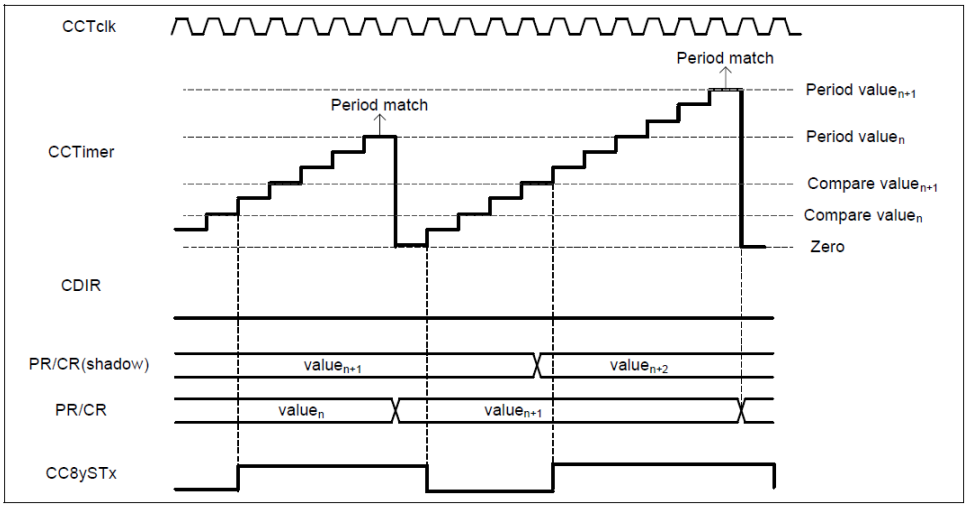


Figura 4.10: Standard Edge Aligned Mode.

4.4.4 Generazione di segnali PWM

CCU8 offre la possibilità di generare PWM con un range di periodo e duty cycle molto flessibile. Il registri di gestione del timer (period e compare) sono registri a 16-bit, e quindi possono contenere valori numerici che vanno da 0 a 65535.
La frequenza del segnale di clock in ingresso a ogni contatore è gestita attraverso un prescaler a 4 bit impostabile per mezzo del registro CC8yPSC.PSIV.
In Normal Prescaler Mode possiamo divedere la frequenza di ingresso al timer $f_{CCU8} = 144MHz$ (CPU internal clock), come mostrato in Figura 4.11.

- In Edge Aligned Mode, per ottenere un segnale con periodo t_{PER}, occorre impostare period register come segue:

$$<periodValue> = \frac{t_{PER}}{t_{CCU8}}, \qquad dove \qquad t_{CCU8} = \frac{1}{f_{CCU8}}$$

  Ad esempio, per ottenere un segnale PWM in Normal Edge Aligned Mode con periodo 20μs e duty cycle 70%, possiamo agire come segue:

  - impostare prescaler=0000 ottenendo $f_{CCU8} = 144MHz$ ovvero $t_{CCU8} = 6,944ns$.;
  - impostare il period pegister: $<periodValue> = \frac{t_{PER}}{tCCU8} = \frac{20\mu s}{6.944ns} = 2880,184$ che deve essere arrotondato all'intero decimale più vicino (2880);
  - per ottenere duty cycle 70% bisogna impostare il comparatore con il valore decimale 864, ovvero il 30% del period register.

  Così facendo, quando il contatore raggiunge il valore del period register (2880), il bit di stato viene disabilitato, il contatore reimpostato al valore iniziale 0000_{HEX} e il conteggio riparte.
  Successivamente, quando viene raggiunto il valore del compare register 864, il bit di stato viene abilitato ed il conteggio continua fino al nuovo raggiungimento del period register.
  Impostando $f_{CCU8} = 144MHz$ sappiamo che il counter incrementa il suo valore ogni $t_{CCU8} = 6,944ns$. In sostanza, il bit di stato permane al valore logico 0 per $864 \times 6,944ns = 6\mu s$, e in seguito assume il valore logico 1 per $(2880 - 864) \times 6,944ns = 14\mu s$, garantendo così la generazione del segnale PWM desiderato.
  Chiaramente, i registri possono contenere solamente valori interi, limitando così la risoluzione temporale del segnale a $t_{CCU8} = 6,944ns$. Il valore massimo contenuto in un registro a 16-bit è 65535, garantendo così, nel caso in cui $t_{CCU8} = 6,9944ns$, la possibilità di generare segnali PWM con periodo massimo $65535 \times 6,944 = 455\mu s$.
  Per generare PWM con duty cycle 100% occorre impostare compare register CC8yCRx=0, mentre per avere duty cycle 0 % occorre avere CC8yCRx=CC8yPR+1.

- In Asymmetrical Edge Aligned Mode, abilitazione e disabilitazione del bit di stato sono imposti dal raggiungimento dei valori contenuti da compare register1 (abilitazione) e compare register2 (disabilitazione).

CC8yPSC.PSIV	Resulting clock
0000_B	f_{ccu8}
0001_B	$f_{ccu8}/2$
0010_B	$f_{ccu8}/4$
0011_B	$f_{ccu8}/8$
0100_B	$f_{ccu8}/16$
0101_B	$f_{ccu8}/32$
0110_B	$f_{ccu8}/64$
0111_B	$f_{ccu8}/128$
1000_B	$f_{ccu8}/256$
1001_B	$f_{ccu8}/512$
1010_B	$f_{ccu8}/1024$
1011_B	$f_{ccu8}/2048$
1100_B	$f_{ccu8}/4096$
1101_B	$f_{ccu8}/8192$
1110_B	$f_{ccu8}/16384$
1111_B	$f_{ccu8}/32768$

Figura 4.11: Normal Prescaler Mode.

4.5 Interrupt

L'architettura Cortex-M4 supporta interrupt ed eccezioni grazie alla CPU e a un apposito modulo NVIC che fornisce fino a 64 livelli di priorità di interrupt. NVIC provvede alla veloce esecuzione della routine di servizio di interrupt (ISR) riducendo drasticamente la latenza. Ciò è ottenuto grazie alla sovrapposizione hardware dei registri e alla possibilità di sospendere e memorizzare operazioni multiple.
NVIC è sensibile alle variazioni delle apposite linee di comunicazione SRx (service request output) con le periferiche. Il valore di queste linee determina l'avvio delle routine di interrupt. Ogni periferica dispone di strutture per la generazione e la gestione dei cosidetti eventi di interrupt, ovvero è in grado di selezionare e comunicare alla CPU attraverso SRx quali eventi devono generare interrupt.

4.5.1 Interrupt del modulo USIC

La struttura del generatore di eventi interrupt è mostrata in Figura 4.12. Se una condizione definita è soddisfatta, un flag di indicazione viene abilitato (Event infication flag) e, se abilitata (interrupt enable), può essere generata una richiesta di interrupt attraverso i segnali SRx.
Questi segnali sono connessi direttamente ai registri di controllo interrupt e al modulo NVIC e permettono alla CPU di reagire alla richiesta di interrupt. Gli eventi di interrupt generati dal trasferimento di dati sono basati sulla trasmissione e sulla ricezione. Ogni protocollo possiede specifici eventi, mentre il modulo USIC fornisce i seguenti eventi che possono essere abilitati/disabilitati individualmente:

- receive event: genera un evento quando una parola è stata ricevuta;
- receive start event: genera un evento quando la ricezione di una parola ha inizio;
- transmit shift event: genera un evento quando una parola è stata trasmessa;

- transimit buffer event: genera un evento quando la trasmissione di una parola ha inizio;
- data lost event: genera un evento che indica la perdita dell'utlima parola ricevuta.

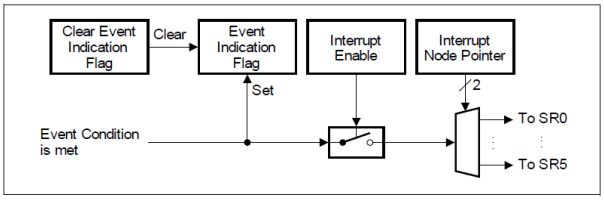


Figura 4.12: Struttura generatore eventi di interrupt USIC.

4.5.2 Interrupt del modulo PWM

Ogni modulo CCU8 possiede una struttura per la generazione degli eventi di interrupt, come si può in Figura 4.13. Il registro CC8yINTS è il registro di stato degli interrupt, e tramite i registri CC8ySWS e CC8ySWR si può modificare via software il valore della sorgente di eventi. Ogni sorgente può essere abilitata/disabilitata per mezzo del registro CC8yINTE. Una sorgente abilitata continua a generare eventi sulla linea di richiesta SRx fino a quando CC8yINTE non è disabilitato. Le più importanti sorgenti di eventi sono:

- period match: un evento di interrupt che indica quando il counter raggiunge il valore contenuto nel period register CC8yPR;
- comapre match: un evento di interrupt che indica quando il counter raggiunge il valore contenuto nei comapre registers CC8yCRx.;
- external event match: un evento di interrupt che indica quando avvengono determinate variazioni sulle linee di ingresso per eventi esterni.

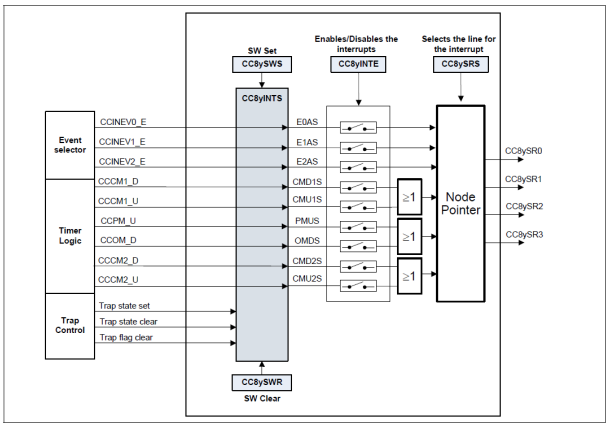
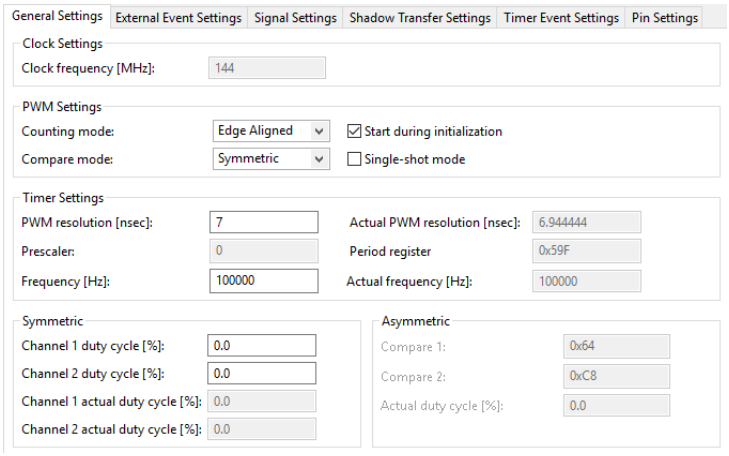


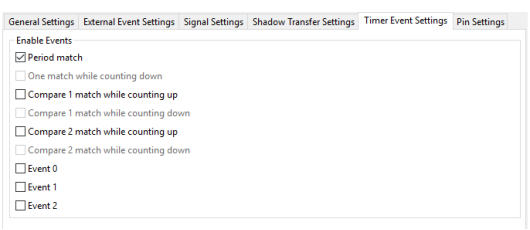
Figura 4.13: Struttura generatore eventi di interrupt CCU8.

4.6 DAVE™

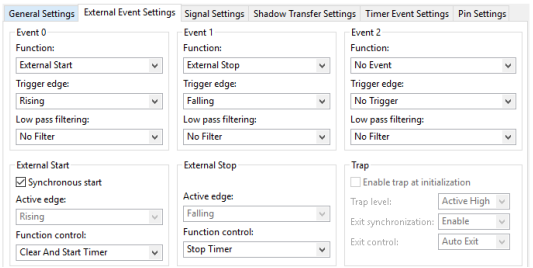
Gestire le periferiche di un microcontrollore significa avere ottima conoscenza di tutti i registri di configurazione e delle varie interconnessioni con CPU e altre periferiche. È chiaro che, soprattutto per progetti lunghi e complessi, è impensabile programmare il software agendo direttamente sui registri. Per questo il produttore (Infineon) mette a disposizione dell'utente uno strumento di sviluppo per la generazione automatica del codice macchina a partire da un linguaggio simil-C.
Per la famiglia di microcontrollori XMC esiste un'appostita libreria XMC_{lib} per la configurazione e la gestione delle periferiche. Inoltre, la configurazione può essere gestita attraverso modelli standard personalizzabili per mezzo di interfacce grafiche: le cosiddette APP. Queste APP permettono di configurare le periferiche con una semplice e intuitiva interfaccia grafica, dove le varie funzionalità sono gestite per mezzo di text box o check box. Il software si occupa di convertire le impostazioni fornite dall'utente tramite APP in linguaggio C utilizzando le funzioni della libreria XMC_{lib}. Successivamente, il compilatore si occupa di tradurre il codice simil-C nel vero e proprio codice macchina.
Un esempio di utilizzo di APP, presentato in (Figura 4.14(a)), mostra la configurazione di un segnale PWM generato del modulo CCU8. L'APP richiede all'utente i valori di frequenza (Hz) e duty-cycle (%) della PWM occupandosi poi di calcolare e impostare i valori dei registri di period e compare. Anche l'abilitazione delle sorgenti dei segnali interrupt (Figura 4.14(b)), come la gestione delle funzioni associate agli eventi esterni (Figura 4.14(c)) e svariate altre impostazioni possono essere gestite attraverso l'APP.



(a) Configurazione periodo e duty cycle.



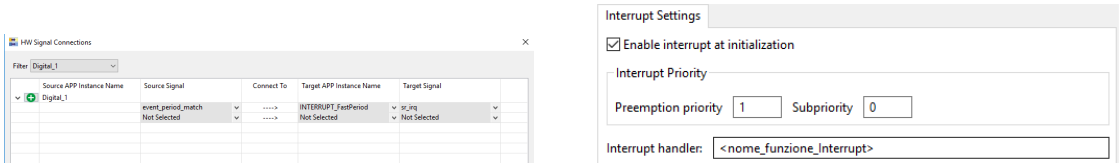
(b) Configurazione eventi di interrupt.



(c) Configurazione eventi esterni

Figura 4.14: APP configurazione CCU8(PWM).

Le informazioni raccolte vengono usate dal software per modificare i registri attraverso le funzioni della libreria XMC_{lib} che sono utilizzate anche per la gestione della periferica durante lo svolgimento del programma. Ad esempio, esiste una funzione per la modifica di frequenza e duty cycle delle PWM che agisce direttamente sui registri period, compare dopo aver calcolato i valori a partire dai dati numerici in ingresso e può essere utilizzata durante l'esecuzione del programma.
DAVE^{TM} fornisce all'utente la possibilità di gestire per mezzo di semplici interfacce anche la mappatura dei pin di uscita, la gestione degli interrupt (Figura 4.15(b)) e le connessioni interne tra CPU e periferiche (Figura 4.15(a)), facilitando di molto la programmazione.
In particolare, la gestione degli interrupt è facilitata dalla presenza di un APP apposita (Figura 4.15(b)). L'utilizzatore è tenuto solamente a indicare qual è il segnale di ingresso generatore (SRx) dell'interrupt e le operazioni che devono essere svolte durante l'esecuzione della routine di interrupt, mentre l'APP si occupa della gestione dei registri e della connessione con il sistema di gestione delle interrupt (NVIC).
A ogni interrupt è associata una funzione che deve essere sviluppata dall'utente e viene richiamata ogni volta che ha inizio la routine di interrupt.
Tornando all'esempio appena discusso, per collegare il segnale di evento period match del modulo PWM a un interrupt, è sufficiente utilizzare l'apposita interfaccia (HW Signal Connections, Figura 4.15(a)) per indicare il segnale generatore dell'evento (SRx) e specificare la funzione da richiamare a ogni occorrenza dell'interrupt. Nelle impostazioni dell'APP (Interrupt) si devono indicare solamente il livello di priorità dell'interrupt e il nome della funzione che sarà implementata dall'utente e richiamata a ogni interrupt.



(a) HW Signal Connections. (b) APP Interrupt.

Figura 4.15: Interfacce DAVE^{TM}

Le APP forniscono modelli standard per la configurazione delle periferiche, ma non sempre garantiscono di sfruttare al massimo le potenzialità del modulo. Per applicazioni complesse è necessario utilizzare le funzioni della libreria XMC_{lib}, e talvolta occorre modificare direttamente i registri.
A volte, alcune funzioni di gestione delle periferiche della libreria contengono istruzioni inutili allo scopo dell'utilizzatore, ma necessarie per garantire generalità alla funzione e quindi riutilizzo in svariate applicazioni. Questa necessità influenza (,a volte anche pesantemente) le prestazioni delle periferiche come vedremo in seguito. L'utente in cerca di elevate prestazioni può comunque generare specifiche funzioni con nuove istruzioni oppure riutilizzando parte di codice presente nelle funzioni predefinite. Agire direttamente sui registri di configurazione e gestione garantisce massime prestazioni.
Una volta controllato e compilato il firmware, DAVE^{TM} si occupa, attraverso un apposito tool (Sagger Jlink), di interagire con i debugger del microcontrollore per il caricamento del firmware e, qualora sia implementato, permette il debug del codice.
In questo progetto, per la gestione delle periferiche si è scelto (,quando possibile), di uti-

lizzare le funzioni predefinite messe a disposizione dalla libreria XMC_{lib} per evitare errori e minimizzare il tempo di lavoro. La configurazione base delle periferiche si basa sull'utilizzo delle APP, mentre per implementare alcune funzionalità specifiche dei protocolli non gestite dai modelli standard e per garantire elevate prestazioni di gestione delle funzionalità dei moduli si è ricorso allo sviluppo di apposite funzioni che talvolta agiscono direttamente sui registri.

4.7 Struttura del firmware

Come visto nelle sez. 4.3 e sez. 4.2 il modulo USIC con il protocollo SSC garantiscono di soddisfare quasi completamente le richieste di progetto relative alla personalizzazione delle comunicazioni seriali (SPI, SCI):

- baud rate generator (sez. 4.2.1): permette di generare segnali di sincronizzazione (CLK) con frequenza superiore ai 10MHz e personalizzabile;
- l'unitò di data shift e buffering (sez. 4.3.1): permette la configurazione del numero di bit di un comando, da 1 a 512 bit;
- time quanta counter (sez. 4.3.3), permette la generazione programmabile delle tempistiche che caratterizzano il protocollo SPI (tid, tfd, tiw);
- interrupt (sez. 4.5.1): permette la generazione di eventi di interrupt legati alla trasmissione dei dati;
- l'unità di data shift (sez. 4.2.2 e sez. 4.3.2): permette di configurare polarità e fase del segnale di sincronizzazione (CPOL, CPHA) della comunicazione SPI.
  Attraverso lo schema di validazione dei dati (Figura 4.4), permette il triggering della trasmissione con eventi esterni al modulo associando l'invio del comando ad un evento esterno. Per esempio, al valore logico di un segnale digitale.

Il modulo CCU8 (PWM, sez. 4.4) può essere utilizzato per la generazione dei segnali digitali da sincronizzare con le comunicazioni seriali. Le principali caratteristiche del modulo sono:

- start e stop (sez. 4.4.1): permette di sincronizzare l'avvio e stop di tutti i segnali PWM;
- shadow transfer (sez. 4.4.2): permette di modificare dinamicamente i registri di periodo e comparazione durante l'esecuzione del programma;
- interrupt (sez. 4.5.2): permette generazione di eventi di interrupt legati ai segnali PWM.

La possibilità di sincronizzare start e stop delle PWM e la possibilità di aggiornare dinamicamente periodo e duty cycle di ognuna di loro permettono di ottenere pattern di commutazioni digitali. Per mantenere la sincronizzazione, ogni PWM deve avere lo stesso periodo a ogni aggiornamento dei registri, mentre il valore del duty cycle può variare tra una PWM e l'altra.

Impostando il segnale in ingresso ai timer $f_{CCU8} = 144MHz$ ($t_{CCU8} = 6,944ns$), la gestione combinata di periodo e duty cycle (da 0% a 100%) permette di collocare le commutazioni del bit di stato delle PWM con risoluzione pari a 7ns (sez. 4.4.4).
Ad esempio, in Figura 4.16 sono descritti i due segnali s1 e s2 in uscita rispettivamente dal timer1 e timer2.
Il pattern generato ha inizio con lo start sincronizzato (CCUCON) dei timer all'istante 0. Per ottenere il primo segmento di pattern (0-10μs) dobbiamo impostare per entrambi i timer lo stesso valore nel registro di period = $\frac{t_{PER}}{t_{CCU8}} = \frac{10\mu s}{6,944ns} = 1440$. Per tutta la durata del periodo s1 rimane basso, quindi per il timer1 dobbiamo impostare duty cycle=0% (compare_s1 =period+1=1441). Impostando per il segnale s2 il timer2 con duty cycle=50% (compare_s2=720) si ottiene invece la commutazione low to high dopo 5μs dallo start dei segnali.
Analogamente, il secondo segmento di pattern (10-30) si ottiene impostando per entrambi i segnali nuovi valori nei registri: periodo=20μs (period=2880), duty cycle=90% (compare_s1=2880) per il timer1 e duty cycle=100% (compare_s2=0) per il timer2.

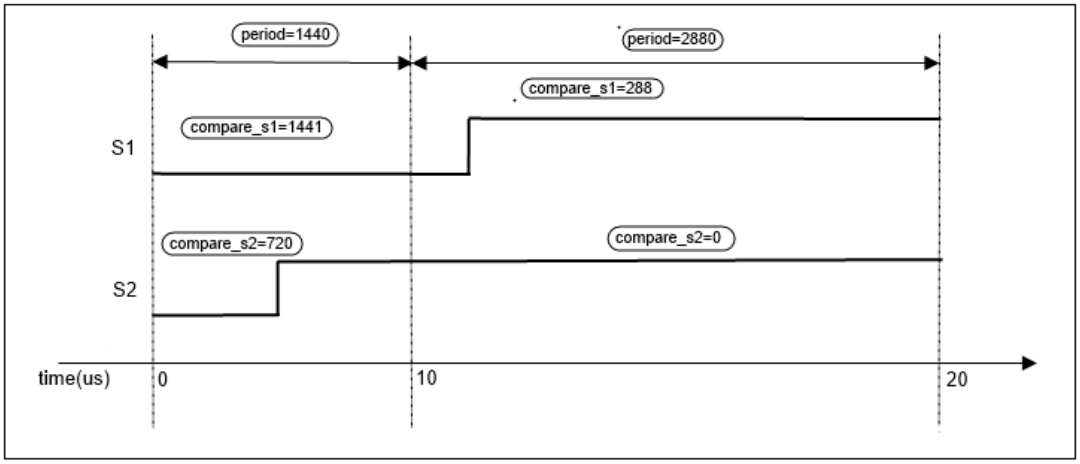


Figura 4.16: Esempio generazione pattern con PWM.

Utilizzando il modulo in modalità Edge Aligne Mode (sez. 4.4.3) abbiamo due possibili configurazioni:

- Standard Edge Aligned Mode: in un periodo del segnale PWM può avvenire una sola commutazione del bit di stato low to high, dovuta al registro compare1 (duty cycle), e una commutazione di fine periodo high to low, dovuta al registro period. Se nel periodo successivo è impostato duty cycle=100%, la commutazione di fine periodo non viene eseguita (segnale s2 in Figura 4.16).
- Assymmetrical Edge Aligned Mode: in un periodo del segnale PWM possono avvenire due distinte commutazioni del bit di stato low to high e high to low, dovute rispettivamente ai registri compare1 e compare2. Per non avere commutazione high to low è sufficiente impostare compare2=period+1. Analogamente, per non aver commutazione low to high occorre impostare compare1=period+1.

É bene ricordare che le uscite finali delle PWM sono definite dalla combinazioni del valore del bit di stato e del registro PSL (sez. 4.4.3).

Per aggiornare dinamicamente i registri di periodo e comparazione si possono sfruttare le caratteristiche di shadow transfer e interrupt del modulo CCU8. Per mezzo dell'evento di interrupt period match (sez. 4.5.2) è possibile avviare una routine di interrupt alla fine di ogni periodo della PWM. All'interno di questa routine si possono modificare i registri di shadow transfer e abilitare il trasferimento dei nuovi valori (GCST.SySS) così da modificare i registri period e compare. L'effettivo cambiamento avverrà al successivo evento di period match.
Ricapitolando, a ogni evento di period match è avviata una routine di interrupt per modificare i registri di shadow transfer, e i valori dei registri compare e period vengono modificati con i valori di shadow transfer definiti nella routine di interrupt generata dall'evento di period match precedente.
Queste operazioni richiedono tempo, che varia da circa 1μs a 4μs, a seconda del numero di PWM coinvolte. Caricare i nuovi valori nei registri di shadow transfer richiede infatti circa 300ns per ogni registro. Per garantire il corretto funzionamento della procedura ed evitare l'avvio di una nuova routine di period match prima che si sia conclusa la precedente, è necessario impostare PWM con periodi maggiori di 2μs o 5μs, anche in questo caso a seconda del numero di PWM coinvolte.

Impostanto i registri TCSR.TDSSM=0 e TCSR.TDEN=1, lo schema di validazione dati del modulo USIC (sez. 4.2.2) permette la trasmissione dei dati solamente quando TCSR.TDV=1 e il segnale di input DX2S=1.
Se TCSR.TDSSM=0, il contenuto del buffer è sempre considerato valido, cioè quando nuovi dati vengono inseriti nel buffer abbiamo TCS.TDV=1. In questo modo l'invio della parola contenuta nel buffer avviene solamente quando il segnale DX2S=1.
La risposta del modulo USIC alla variazione del segnale di ingresso è pressoché immediata. Tra una commutazione low to high del segnale DX2S e l'effetivo inizio di una trasmissione SPI (commutazione del segnale di CS) trascorrono in media 40/50 ns (CSdelay).

Unendo le caratteristiche del modulo PWM e del modulo USIC(SSC) appena descritte, possiamo ottenere la sincronizzazione del pattern di linee digitali con le comunicazioni seriali. Infatti, possiamo sostituire il segnale CS prodotto internamente dall'interfaccia con un segnale digitale sincronizzato prodotto da un timer PWM utilizzandolo sia come CS vero e proprio della comunicazione, sia come segnale di ingresso DX2S per la validazione dei dati.
Così facendo, attraverso la gestione di periodo e duty cycle, si deve garantire che il segnale PWM CS durante una comunicazione SPI si comporti esattamente come il segnale CS.
Una comunicazione SCI può essere implementata attraverso una comunicazione SPI, aggiungendo il segnale di SYNC prodotto utilizzando un'altra PWM e trascurando i segnali di MISO e CS. Quindi, con il protocollo SCI, il segnale CS è utile solo come ingresso DX2S per la validazione dei dati. I vantaggi di questo approccio sono:

- sincronizzazione tra linee digitali e CS della comunicazione digitale;
- personalizzazione della distanza tra due comandi *tic* (il CS è ora gestito come segnale digitale dal modulo PWM, quindi possiamo gestire le distanze con 7ns di risoluzione).

Queste caratteristiche, assieme alla personalizzazione dei comandi seriali permessa dal modulo USIC:SPI, garantiscono di ottenere la sincronizzazione tra linee digitali e comunicazioni seriali.
Questa soluzione, tuttavia, presenta limiti per l'utilizzatore e svantaggi per la programmazione del firmware e del driver:

- distanza minima tra due comandi e lunghezza minima di un comando, in alcuni casi, sono limitati dalla lunghezza minima del periodo PWM (da 2 a 5μs);
- la generazione del pattern che avviene attraverso i valori dei registri compare e register, richiede un'importante fase di elaborazione e traduzione per fornire all'utente finale un'interfaccia di definizione del pattern semplice e intuitiva, come in figura..

4.7.1 Configurazione del pattern digitale

Il pattern digitale è composto da 5 segnali digitali e 2 CS realtivi a due distinte interfacce SPI. Per i segnali di CS sono utilizzati timer in Edge Aligned Mode che permette una commutazione low to high del bit di stato definita dal compare register e una transizione high to low definita dal period register. Con il registr CC8y.PSL=0 (passive level) e con lo schema di validazione della trasmissione (TCSR.TDSSM=0 e TCSR.TDEN=1) dei segnali di CS, si impone che la trasmissione sia abilitata quando il bit di stato=1 e disabilitata quando il bit di stato=0.
Durante una comunicazione seriale ogni periodo dei segnali PWM (CS) rappresenta la distanza $tic + CMDLenght$ (Figura 4.17), dove tic rappresenta la distanza tra la fine di un comando e l'inizio del successivo (CS inattivo), mentre $CMDLenght$ rappresenta il tempo durante il quale avviene la comunicazione (CS attivo).
Ricapitolando, alla fine di ogni periodo il timer si azzera, il bit di stato si resetta, CS diventa inattivo e la trasmissione è disattivata (tic). Quando il timer raggiunge compare register il bit di stato si alza, CS diventa attivo e anche la trasmissione dei dati è attiva $CMDLenght$. Al raggiungimento del nuovo period register il ciclo ricomincia.
Così facendo, period register e compare register ad ogni periodo definiscono rispettivamente le distanze temporali $tic + CMDLenght$ e tic.
Durante un pattern di comunicazione il modulo permette di inviare solamente comandi della stessa lunghezza, pertanto $CMDLenght$ deve rimanere invariato, ma la modifica dinamica di period e compare permettono la variazione del tic tra un comando e l'altro.
Per mantenere la sincronizzazione ogni timer deve essere impostato con lo stesso period register a ogni aggiornamento dei registri.
I segnali digitali sono implementati utilizzando timer in Assymmetrical Edge Aligned Mode permettendo una commutazione low to high definita dal compare1 e una commutazione high to low definita dal compare2 durante un periodo $tic + CMDLenght$. Se compare2<compare1 non avvengono commutazioni. Non sono quindi permesse commutazioni high to low seguite da commutazioni low to high.
Con una commutazione low to high seguita da una commutazione high to low è possibile generare il segale SYNC di UPD. Per mantenere le linee a un valore logico costante durante tutto un periodo si devono impostare i registri come descritto:

- linea CS=1: compare=0 (duty cycle 100%);
- linea CS=0: compare=period+1 (duty cycle 0%);

- linea digitale=1: compare1=0, compare2=period (duty cycle 100%);
- linea digitale=0: compare1=period+1, compare2=period (duty cycle 0%).

Generalmente, un pattern è costituito da una prima fase di sole commutazioni digitali, seguito da una fase di commutazioni digitali con comunicazioni seriali e, infine un'altra fase di sole commutazioni digitali.
Durante la fase di sole commutazioni digitali il pattern è così definito:

- registro period: rappresenta la distanza temporale tra due commutazioni digitali;
- registro compare (CS), compare1 e compare2 (Digital): sono impostati in modo da mantenere le linee a un valore logico costante durante tutto il periodo (duty cycle 100% o %0). Eventali commutazioni possono avvenire solamente a fine periodo.

Durante la fase di commutazioni digitali e comunicazioni seriali il pattern è così definito:

- registro period: rappresenta la distanza temporale $tic + CMDLenght$;
- registro compare (CS): rappresenta la distanza temprale tic;
- registro compare1 (Digital): rappresenta l'istante di commutazione high to low delle linee digitali;
- registro compare2 (Digital): rappresenta l'istante di commutazione low to high delle linee digitali.

Per generare un pattern bisogna definire per ogni periodo i valori dei registri compare di ogni PWM. Sfruttando le caratteristiche di shadow transfer e interrupt delle PWM i registri devono essere cambiati dinamicamente. La combinazione dei registri period e compare permette all'utente di definire le commutazioni con risoluzione fino a 7ns (sez. 4.4.4).

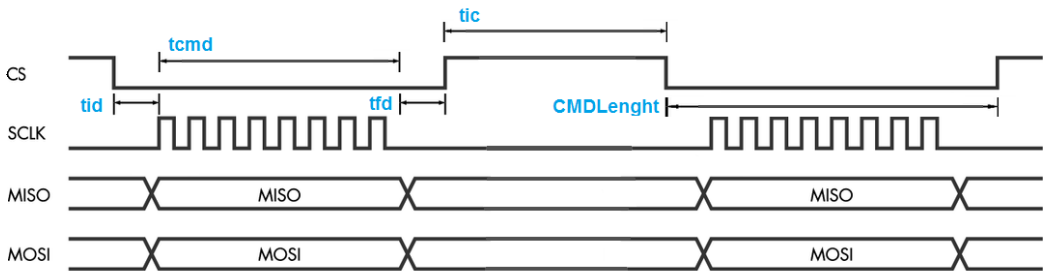


Figura 4.17: Parametri temporali comunicazione SPI.

Dal punto di vista software la descrizione del pattern avviene per mezzo di un array di elementi struct (`PWMdata`), che deve essere interamente definito prima di avviare i timer.

```
typedef struct{
      uint32_t period;
      uint32_t compareCS;
      uint32_t compareCSoff;
      uint32_t compare11;
      uint32_t compare21;
      uint32_t compare12;
      uint32_t compare22;
      uint32_t compare13;
      uint32_t compare23;
      uint32_t compare14;
      uint32_t compare24;
      uint32_t compare15;
      uint32_t compare25;
}PWMdata;
```

Ogni elemento `PWMdata` contiene i valori dei registri period (uguale per tutti i timer) e compare delle varie PWM.
A ogni period match delle PWM è avviata una routine di interrupt[4] che provvede ad aggiornare i registri con i valori contenuti in una struttura `PWMdata`, per esempio la funzione `periodPWM`.

```
void periodPWM(void){
    contPer++;
    if(contPer<length){
        writeDigitalLine(PWMCSdata,contPer,CS,CSoff,&Digital_1,
                         &Digital_2,&Digital_3,&Digital_4,&Digital_5);
    }else{
        if(contPer==length){
            stopAllPWM();
            read=true;
        }
    }
}
```

In questa funzione, la variabile `conPer` inizializzata a 0, incrementa a ogni istanza dell'interrupt, ed è utilizzata per indicizzare l'array `PWMCSdata` di strutture `PWMdata`. La lunghezza del pattern è definita dalla variabile `length`, che di fatto rappresenta il numero di periodi che compongo il pattern.
Se `contPer < lenght` il pattern non è concluso e quindi, attraverso la funzione `writeDigitalLine`, vengono scritti i registri shadow register con i nuovi valori che saranno trasferiti nei registri compare e period al prossimo period match.
Quando `contPer=lenght` il pattern è concluso e, grazie alla funzione `stopAllPWM`, vengono stoppati contemporaneamente i timer garantendo che le linee permangano al valore definito nell'ultimo periodo.

```
void writeDigitalLine(PWMdata* data,uint8_t index,PWM_CCU8_t* ptrCS,
                      PWM_CCU8_t* ptrCSoff,PWM_CCU8_t* ptrD1,PWM_CCU8_t*
                      ptrD2,PWM_CCU8_t* ptrD3,PWM_CCU8_t* ptrD4
                      ,PWM_CCU8_t* ptrD5){
```

[4]L'evento di period match può essere generato da uno qualsiasi dei timer coinvolti, in quanto, period register è lo stesso per ogni timer a ogni aggiornamento.

```
    ptrCS->ccu8_slice_ptr->PRS = data[index].period;
    ptrCS->ccu8_slice_ptr->CR1S =  data[index].compareCS;

    ptrCSoff->ccu8_slice_ptr->PRS=data[index].period;
    ptrCSoff->ccu8_slice_ptr->CR1S=data[index].compareCSoff;


    ptrD1->ccu8_slice_ptr->PRS =  data[index].period;
    ptrD1->ccu8_slice_ptr->CR1S =  data[index].compare11;
    ptrD1->ccu8_slice_ptr->CR2S =  data[index].compare21;
    .
    .
    .
    ptrD5->ccu8_slice_ptr->PRS = data[index].period;
    ptrD5->ccu8_slice_ptr->CR1S =  data[index].compare15;
    ptrD5->ccu8_slice_ptr->CR2S = data[index].compare25;

    ptrD1->ccu8_module_ptr->GCSS = 4369;//CCU80
    ptrD5->ccu8_module_ptr->GCSS = 4369;//CCU81
}
```

L'aggiornamento dei registri avviene per mezzo di funzioni come `writeDigitalLine`. Il numero di PWM coinvolte nell'aggiornamento incide sulla durata delle funzioni di interrupt richiamate dagli eventi di period match e di conseguenza sulla lunghezza del minimo periodo che può essere impostato. Un periodo troppo breve non consente infatti il completo aggiornamento dei registri prima del successivo period match.
Le funzioni messe a disposizione da XMC_{lib} calcolano automaticamente il valore dei registri period e compare a partire dal periodo e duty cycle richiesto, ma richiedono tempi superiori a 20μs.
Per ridurre al minimo questo tempo, si è deciso di impostare direttamente i registri di shadow register (period e compare) lasciando al driver[5] il compito di calcolare i valori traducendo il pattern definito in formato tabellare. Così facendo, l'aggiornamento di tutte e 5 le linee digitali richiede l'uso di periodi superiori a 5μs, che possono essere ridotti a 2μs utilizzando una sola linea digitale.
Un'ulteriore struttura (`Pattern_Setting`) è utilizzata per contenere informazioni sul pattern utili nella definizione dei comandi da inviare.

```
typedef struct{
      uint8_t numWordCMD;
      uint8_t numCMD;
      uint16_t numCMDLoop;
      uint8_t numberOfBitToRead;
      uint8_t readMode;
}Pattern_Setting;
```

4.7.2 Configurazione del modulo SPI

Il pattern digitale, gestito dal modulo CCU8 (PWM), si occupa di temporizzare l'invio dei comandi utilizzando un segnale PWM (CS), che svolge la duplice funzione di ingresso per lo schema di validazione della trasmissione e vero e proprio segnale CS di uscita

[5]Il compito di calcolare i valori compare e period è affidato al driver per non appesantire il firmware del controllore.

dall'interfaccia.
La lunghezza dei comandi e le varie tempistiche devono essere conosciute prima di definire il pattern per poter generare correttamente il CS. Le caratteristiche delle comunicazioni seriali devono quindi essere impostate prima di avviare il pattern. A questo scopo sono state sviluppate le seguenti funzioni di configurazione del modulo USIC(SSC):

```
void setBaudrateMCLK40(SPI_MASTER_t* spi,uint32_t baudRate);
```

`setbaudrateMCLK40`: agisce sui parametri BRG.FDR, BRG.PPPEN, BRG.PDIV (sez. 4.2.1) e imposta MCLK $f_{MCLK} = 40MHz$.
Il parametro baudrate (1-2048) permette di scegliere la frequenza di SCLK $f_{SCLK} = \frac{40MHz}{baudrate}$.

```
void setTid(SPI_MASTER_t* ptr,uint32_t DCTQ,uint32_t PCTQ,uint32_t CTQSEL);
```

`setTid`: agisce sui parametri BRG.DCTQ, BRG.PCTQ, BRG.CTQSEL (sez. 4.2.1, sez. 4.3.3) per impostare i valori di $tid = tfd$.

- BRG.CTQSEL=11: imposta $f_{CTQIN} = f_{MCLK} = 40MHz$ e quindi tq=25ns;
- BRG.CTQSEL=10: imposta $f_{CTQIN} = f_{SCLK}$ e tq=$\frac{1}{f_{SCLK}}$.

BRG.DCTQ(0-3), BRG.PCTQ(1-32) definisco il numero di quanti temporali tq che compongono Tid.

```
void setCLKPolarity(SPI_MASTER_t* spi,uint32_t mode);
```

`setCLKPolarity`: agisce sul registro BRG.SCLKCGFG (sez. 4.3.2) e definisce la polarità del CLK (sez. 2.1.1) grazie al parametro mode (0-3):

- mode=0: CPOL=0, CPHA=1;
- mode=1: CPOL=0, CPHA=0;
- mode=2: CPOL=1, CPHA=0;
- mode=3: CPOL=1, CPHA=1.

```
void setTic(SPI_MASTER_t* ptr,uint32_t DCTQ1,uint32_t PCTQ1,uint32_t CTQSEL1);
```

`setTic:` imposta il valore di tnf attraverso i parametri PCR.DCTQ=11, PCR.PCTQ=0, PCR.CTQSEL1=11 (sez. 4.2.1, sez. 4.3.3). In questo modo, il modulo USIC può inviare un comando già dopo tnf=75ns dal precedente.

```
void setCSPolarity(PWM_CCU8_t* ptrCS,uint32_t passiveLevel,SPI_MASTER_t* spi);
```

`setCSpolarity`: come ribadito in precedenza, il segnale CS è utilizzato sia per la validazione della trasmissione (sez. 4.2.2), sia come vero e proprio segnale CS di uscita dall'interfaccia.
Il segnale di validazione deve essere attivo (1) durante la trasmissione (CMDlenght) inattivo (0) al di fuori della comunicazione (TIC) . Fortunatamente in Edge Aligne Mode sono previste due distinte uscite PWM che si basano sullo stesso bit di stato[6]. La prima uscita

[6]Le uscite PWM sono definite dalla combinazione del valore del bit di stato e del valore del registri PSL.

del timer con PSL=0 (sempre) è collegata all'ingresso DX2S dello schema di validazione. La seconda uscita, invece, è utilizzata come segnale di CS dell'interfaccia e la sua polarità può essere gestita modificando il bit PSL. É bene ricordare che i due segnali sono esattamente uguali a meno dell'eventuale inversione di polarità. Così facendo, si garantisce l'indipendenza dello schema di validazione dalla polarità del CS, che può anche assumere le configurazioni sempre alto e sempre basso.
La funzione `setCSPolarity` agisce sul registro PSL della seconda uscita del timer e definisce la polarità del CS grazie al parametro passiveLevel (0-3) (sez. 2.1.1):

- passiveLevel=0: attivo basso;
- passiveLevel=1: attivo alto;
- passiveLevel=2: sempre basso;
- passiveLevel=3: sempre alto.

```
void setExternalStartMode(SPI_MASTER_t* spi);
```

`setExternalStartMode`: attiva lo schema validazione trasmissione. La funzione agisce sul registro TCSR.TDEN e sul registro DXCR (sez. 4.3.2) per impostare il pin di ingresso collegato al segnale DX2S. Il parametro spi definisce quale delle interfacce SPI è utilizzata.

```
void XMC_SPI_CH_DisableInterwordDelay(XMC_USIC_CH_t * channel);
```

`XMC_SPI_CH_DisableInterwordDelay`: disabilita tiw (sez. 4.3.2).

```
void XMC_SPI_CH_SetWordLength(XMC_USIC_CH_t * channel, uint8_t word_length);
```

`XMC_SPI_CH_SetWordLength`: agisce sul registro SCTR.WLE (1-8) (sez. 4.3.2) e definisce la lunghezza delle parole grazie al parametro word_length. La connessione con il driver avviene attraverso tipiche trasmissioni seriali che utilizzano pacchetti di 8-bit. Si è quindi deciso di limitare la massima lunghezza delle parole che compongo i comandi da inviare a 8-bit, per garantire che ogni parola possa essere contenuta in un singolo pacchetto di comunicazione e per facilitare la memorizzazione.

```
void XMC_SPI_CH_SetFrameLength(XMC_USIC_CH_t * channel, uint8_t frame_length)
```

`XMC_SPI_CH_SetFrameLength` agisce sul registro SCTR.WLE (1-64)(sez. 4.3.2) e definisce la lunghezza de comando grazie al parametro frame_legth.

Tutte queste funzioni sono richiamate per mezzo della funzione `initSPI`, prima dell'invio del pattern, nella cosiddetta fase di configurazione SPI.

```
void initSPI(SPI_MASTER_t* spi,PWM_CCU8_t* pwm,SPI_Setting* setSPI)
```

Tutti i parametri di ingresso alle varie funzioni sono contenuti in un unica struttura dati, `SPI_Setting`.

```
typedef struct{
      uint8_t CSpolarity;
      uint8_t CLKpolarity;
      uint8_t wordLength;
```

```
    uint8_t frameLength;
    uint8_t PCTQ;
    uint8_t DCTQ;
    uint8_t CTQSEL;
    uint32_t baudRate;
    uint8_t baudRateMode;
    bool enable;
}SPI_Setting;
```

4.7.3 Modalità di generazione del pattern

In base al numero e al tipo di comandi possiamo distinguere quattro diverse modalità di generazione del pattern: fast pattern, normal pattern, long pattern e loop pattern. Ogni modalità è caratterizzata da specifiche routine di interrupt.

Fast mode

Fast mode è utilizzata per pattern con:

- comandi seriali con lunghezza minore di 64-bit;
- numero masssimo di bit dell'intera comunicazione 512-bit;
- pattern con 1 linea digitale sincronizzata con il CS della comunicazione.

Prima di avviare il pattern digitale bisogna caricare tutti i comandi da inviare nel buffer attraverso l'apposita funzione transmit, fornita dalla libreria XMC che si occupa autonomamente di trasmissione e ricezione dei messaggi SPI contenuti ngli array monodimensionali writeSPI e readSPI.
Caricando i dati nel buffer TCSR.TDV=1, l'invio dei comandi avviene quando DX2S=1, ovvero CS=1, con l'impostazione di $tnf = 75ns$ e $tiw = 0$ si garantisce che il modulo USIC sia pronto a inviare il prossimo comando dopo il tempo tns. L'invio dei comandi è però determinato dal valore del segnale CS, che quindi avviene solamente dopo il tempo tic imposto dal pattern.
La presenza di una sola linea digitale permette di utilizzare PWM con periodi di minimo 2μs. Il pattern digitale, per come è costruito, obbliga ad avere distanza $tic + CMDLenght > 2\mu s$. Purtroppo, soprattutto ad alte frequenze e per comandi con pochi bit, possiamo avere comandi con lunghezza CMDLenght inferiore ai 2μs e quindi, in questo caso, tic deve essere abbastanza lungo per garantire $tic + CMDLenght > 2\mu s$.
L'unica funzione di interrupt coinvolta in questa modalità è `fastPeriodPWM` che è avviata da un evento di period match del pattern:

```
void fastPeriodPWM(void);
```

All'interno di `fastPeriodPWM` avviene la scrittura dei registri di shadow transfer grazie alla funzione:

```
void writeDigitalLineFast(PWMdata* data,uint8_t index,PWM_CCU8_t*
                          ptrCS,PWM_CCU8_t* ptrCSoff,PWM_CCU8_t* ptrD1);
```

Normal mode

Normal mode è utilizzata per pattern con:

- comandi seriali con lunghezza minore di 64-bit;
- numero masssimo di bit dell'intera comunicazione 512-bit;
- pattern con 5 linee digitali sincronizzate con il CS della comunicazione.

Il funzionamento è il medesimo di Fast mode, ma la presenza di 5 linee digitali permette di utilizzare PWM con periodi di minimo 5μs. Il pattern digitale, per come è costruito, obbliga ad avere distanza $tic + CMDLenght > 5\mu s$. Purtroppo, soprattutto ad alte frequenze e per comandi con pochi bit, possiamo avere comandi con lunghezza CMDLenght inferiore ai 5μs e quindi, in questo caso, tic deve essere abbastanza lungo per garantire $tic + CMDLenght > 5\mu s$. L'unica funzione di interrupt coinvolta in questa modalità è `periodPWM` che è avviata da un evento di period match del pattern:

```
void periodPWM(void);
```

All'interno di `periodPWM` avviene la scrittura dei registri di shadow trnasfer grazie alla funzione:

```
void writeDigitalLine(PWMdata* data,uint8_t index,PWM_CCU8_t* ptrCS,
                      PWM_CCU8_t* ptrCSoff,PWM_CCU8_t* ptrD1,
                      PWM_CCU8_t* ptrD2,PWM_CCU8_t* ptrD3,PWM_CCU8_t*
                      ptrD4,PWM_CCU8_t* ptrD5);
```

Long mode

Long mode è utilizzata per pattern con:

- comandi seriali con lunghezza maggiore di 64-bit (multipli di 8-bit) oppure numero di bit dell'intera comunicazione maggiore di 512-bit;
- pattern con 5 linee digitali sincronizzate con il CS della comunicazione.

Il buffer può contenere al massimo 512-bit e inoltre, per terminare comandi con più di 64-bit, è necessario svuotare completamente il buffer (sez. 4.3.1). Non è quindi possibile pre-caricare il buffer con l'intera comunicazione prima dell'avvio del pattern come nelle altre modalità, ma bisogna ricaricare il buffer con nuovi dati alla fine di ogni comando. Per fare ciò si ricorre all'utilizzo degli eventi di interrupt che possono essere generati alla fine di una comunicazione.
Ogni volta che un comando termina, con la lettura dell'ultimo bit può essere generato un evento di interrupt al quale può essere associata una routine. All'interno della routine il buffer viene riempito con il nuovo comando grazie alla la funzione transmit. Come per la generazione del pattern, anche per contenere i dati da inviare che saranno inseriti nel buffer si utilizza un array multidimensionale (`**writeSPILong`) e una variabile di indicizzazione `contCMD`, aggiornata ad ogni istanza di interrupt per tenere traccia della posizione dei dati da inserire nel buffer.
Il numero di parole di un comando e il numero dei comandi coinvolti nel pattern sono definiti nella struttura `Pattern_Setting`. Grazie allo schema di validazione dati, l'invio

del comando è determinato dal valore di DX2S=1 ovvero CS=1. L'aggiornamento del buffer, l'avvio della routine di interrupt e la preparazione del modulo USIC all'invio (funzione transmit) del nuovo comando richiedono circa 10-15μs. Per garantire il corretto funzionamento in ogni situazione si è deciso di mantenere una distanza tra la fine di un comando e l'inzio del successivo maggiore di 30μs (tic). Questa scelta per tic esclude i problemi di lunghezza minima della distanza $tic + CMDLenght$ visti nelle precedenti modalità.
La funzione di period match è `periodPWM`, la stessa di Normal mode, mentre la funzione di interrupt avviata da un evento di fine lettura dove viene riempito il buffer è:

```
void endOfSPI_Long();
```

Loop mode

Loop mode è caratterizzata da:

- comandi seriali con lunghezza qualsiasi (maggiore di 64-bit solo multipli di 8-bit) oppure numero di bit dell'intera comunicazione maggiore di 512-bit;
- pattern con 5 linee digitali sincronizzate con il CS della comunicazione;
- possibilità di inviare un comandi in loop;
- possibilità di leggere un comando durante l'esecuzione del pattern.

Il funzionamento è il medesimo di Long mode ma, in questo caso, alla fine del pattern l'ultimo comando viene ripetuto in loop. Questo loop termina al raggiungimento di un numero prefissato dall'utente di ripetizioni, oppure quando il comando letto MISO corrisponde ad uno specifico comando, anch'esso definito dall'utente.
In caso di uscita dal loop dovuta alla lettura di uno specifico comando, è possibile commutare le linee digitali. Per fare ciò è sufficiente stoppare i timer, modificare i registri tramite shadow register e riavviare i timer. Queste operazioni sono gestite aggiungendo alla routine di interrput avviata dall'evento di fine comando delle funzioni di confronto per i comandi ricevuti. Il numero di parole da inserire nel buffer, il numero di ripetizioni e il tipo di confronto dei dati (AND o OR) sono definiti nella struttura `Pattern_Setting`.
La funzione di period match è `periodPWM`, la stessa di Normal mode, mentre la funzione di interrupt avviata da un evento di fine lettura, dove viene riempito il buffer e viene confrontato il comando ricevuto con il comando di fine loop, è:

```
void endOfSPI(void);
```

In questo evento di fine lettura sono presenti due funzioni per la lettura e il confronto dei dati ricevuti durante la comunicazioni, che si differenziano per il tipo di comparazione:

- OR: verifica che un comando o parte di un comando,sia anche solo parzialmente uguale a quello definito dall'utente;

```
bool DigitalLineAnswerReadSO_OR(readWord* ptrRW ,uint8_t* read,
                                Pattern_Setting* setSPI);
```

- AND: verfica che un comando o parte di un comando, sia esattamente uguale a quello definito dall'utente.

```
bool DigitalLineAnswerReadSO_AND(readWord* ptrRW ,uint8_t* read,
                                 Pattern_Setting* setSPI);
```

I dati di comparazione sono defininti dall'utente e gestiti dalla struttura `readWord`.

```
typedef struct {
   uint8_t mask;
   uint8_t pos;
   uint8_t valore;
   uint8_t word_n;
}readWord;
```

4.7.4 Il main

Riassumendo, la generazione del pattern digitale e delle comunicazioni seriali è affidata ai moduli USIC e CCU8:PWM, e l'aggiornamento dinamico dei registri del modulo CCU8 è gestito dalle routine di interrupt.
Come suggerito nella fase di studio di fattibilità (sez. 3.1), sono state sfruttate il più possibile le periferiche e si è cercato di ridurre al minimo il codice gestito dalla CPU. In questo caso, la funzione principale `main` invocata all'inizio del programma si occupa della comunicazione con il driver di gestione dell'interfaccia.

```
int main(void){
    %Inizializzazione
      DAVE_STATUS_t status;

      status = DAVE_Init();

      %Start USB_COM connection
      if(USBD_VCOM_Connect() != USBD_VCOM_STATUS_SUCCESS){
          return -1;
      }
      while(!USBD_VCOM_IsEnumDone());

      %Inizializzazione  pwm
      initPWM(&staticDigitalValue,CS,CSoff,&Digital_1,&Digital_2,
              &Digital_3,&Digital_4,&Digital_5);
```

All'accensione del dispositivo la funzione `main` viene richiamata dalla CPU e le prime operazioni eseguite si occupano di inizializzare le periferiche. In fase di sviluppo del firmware il modulo CCU8, il modulo USIC e le varie routine routine di interrupt sono stati impostati attraverso le APP di configurazione fornite dall'ambiente di sviluppo DAVE™ (sez. 4.6. Questo software si occupa di convertire le impostazioni fornite dall'utente tramite le interfacce grafiche delle APP in linguaggio simil-C utilizzando le funzioni della libreria XMC_{lib}. Queste funzioni di configurazione vengono raggruppate in un unica funzione: `Dave_Init`, richiamata ad ogni avvio del programma.
Nel `main` è gestita la comunicazione tra il microcontrollore e il mondo esterno grazie ad una APP che permette di implementare un'interfaccia virtuale (COM port) attraverso USB, per porte seriali virtuali di comunicazione (VCOM). Quest'interfaccia è gestita autonomamente dal microcontrollore e l'utilizzatore si deve occupare solamente dell'avvio

della connessione (`USBD_VCOM_Connect()`), della lettura e della scrittura dei dati, attraverso apposite funzioni.
Infine, vengono inizializzati tutti i registri delle PWM con valori di default che impostano tutte le linee (CS e digitali) del pattern al valore logico 0 (`initPWM`).

```
%Lettura
  while(1U)
  {
      if(read){

          bytesRecived = USBD_VCOM_BytesReceived();

          if(bytesRecived){
              USBD_VCOM_ReceiveData(temp_rx_buffer, 4);

              operation=(uint8_t) temp_rx_buffer[0];
              patternMode=(uint8_t) temp_rx_buffer[1];
              bytes=(uint8_t) temp_rx_buffer[2] |
                     (uint8_t) temp_rx_buffer[3]<<8;

              for(int k=0;k<bytes;k++){
                  USBD_VCOM_ReceiveData(&rx_buffer[k],1);
              }

              CDC_Device_USBTask(&USBD_VCOM_cdc_interface);
```

Dopo la fase di inizializzazione ha inizio il vero e proprio corpo del programma. All'interno di un ciclo while infinito, avviene la continua lettura della porta seriale di comunicazione virtuale (`USBD_VCOM_ReceiveData`) in attesa di appositi comandi dall'esterno che definiscano quali funzioni devono essere svolte.

```
              switch (operation) {
                    case 1:
                        configureSPI(patternMode);
                        break;
                    case 2:
                        configurePattern(patternMode);
                        break;
                    case 3:
                        startPatternTS(patternMode,(uint8_t)rx_buffer[0]);
                        break;
                    case 4:
                        configureStaticDigitalLine();
                        break;
                    case 5:
                        resetSystem();
                        break;
                    case 6:
                        readPatternData(patternMode);
                        break;
                    case 7:
                        pwmStart_Stop();
                        break;
                    case 8:
                        readStaticDigitalLine();
                        break;
                    default:
                        break;
                }
          }
```

```
            CDC_Device_USBTask(&USBD_VCOM_cdc_interface);
        }
    }
}
```

Le funzioni messe a disposizione dell'utente permettono di gestire l'interfaccia e tutte le funzionalità a essa associate. Attraverso degli appositi comandi l'utente può avviare funzioni che si occupano di configurazione del pattern, generazione del pattern, gestione static delle linee, lettura dei dati e gestione dei segnali PWM. Nella prossima sezione saranno analizzate in dettaglio queste funzioni con particolare attenzione al protocollo per lo scambio di dati.

4.8 Protocollo di comunicazione con il driver

Per poter comunicare con il microcontrollore si è definito un semplice protocollo[7], dove i primi 4 bytes di ogni comando ricevuto corrispondo a:

- OP[0]: identifica quale funzione deve essere eseguita e viene memorizzato nella variabile `operation`;
- PM[1]: identifica la modalità di generazione del pattern (sez. 4.7.3) e viene memorizzato nella variabile `pattern mode`;
- BY[2][3]: identifica quanti bytes compongono il resto del comando ricevuto (DATA) e viene memorizzato nella variabile `bytes`.

DATA[n] identifica i dati che assumono un ruolo specifico a seconda della funzione richiamata e che vengono inseriti in un buffer (`rx_buffer`) in attesa di essere letti e utilizzati .

0	1	2 3	n
OP	PM	BY	DATA
1-8	0-3	0-65535	

Tabella 4.1: Esempio formato comandi.

Il primo byte (OP) di ogni comunicazione identifica tramite un numero intero (1-8) le operazioni che possono essere svolte dal micrcontrollore per mezzo di specifiche funzioni messe a disposizione dell'utente, che permettono di gestire tutte le fasi della generazione di un pattern, e funzioni implementate dall'interfaccia. Queste funzioni si occupano principalmente di:

- leggere i dati di configurazione contenuti nell'rx_buffer e di inserirli in apposite strutture dati;

[7]La prima riga rappresenta la posizione dei byte, la seconda riga il nome del campo e la terza riga il valore decimale che può essere contenuto nel dato.

- scrivere sulla porta di comunicazione i dati richiesti dall'utente, per esempi i comandi ricevuti MISO in un pattern SPI;
- impostare il valore statico delle linee digitali o dei segnali PWM configurabili;
- gestire le procedure di configurazione e avvio del pattern.

Per poter comunicare correttamente con il microcontrollore sono stati definiti alcuni formati che regolano la costruzione dei pacchetti di dati in ingresso (`rx_buffer`) e in uscita.

4.8.1 configureSPI

```
void configureSPI(uint8_t patternMode);
```

Questa funzione permette di configurare il modulo USIC. Per prima cosa vengono letti i dati di configurazione del modulo USIC e inseriti una struttura di tipo `SPI_Setting`. Successivamente si configura il modulo grazie alla funzione `initSPI` (sez. 4.7.2). Il formato dei comandi di configurazione è il seguente:

0	1	2	3	4	5	6	7
OP	PM	BY		CSPOL	CLKPOL	WL	FL
1	0-3	13		0-3	0-3	1-8	1-16

8	9	10	11	12	13	14	15
PCTQ	DCTQ	CTQSEL	BR				BRM
0-3	1-32	2-3	4-2048				0-1

16	
EN	
0-2	

Tabella 4.2: Formato configureSPI

- CSPOL, polarità CS (sez. 2.1.2):
  - 0: attivo basso,
  - 1: attivo alto,
  - 2: sempre alto,
  - 3: sempre basso.
- CLKPOL, polarità CLK (sez. 2.1.2):
  - 0: CPOL=0, CPHA=1;
  - 1: CPOL=0, CPHA=0;
  - 2: CPOL=1, CPHA=0;
  - 3: CPOL=1, CPHA=1.

- WL(1-8): lunghezza delle parole (sez. 4.3.1);
- FL(1-64): lunghezza dei comandi (sez. 4.3.1);
- PCTQ(0-3) e DCTQ(1-32): impostazioni del numero di quanti tq per generare $tid = tfd$ (sez. 4.3.3);
- CTQSEL, impostazione del quanto temporale tq (sez. 4.3.3):
  - 2: $tq = \frac{BR}{40MHz}$;
  - 3: $tq = 25ns$;
- BR(4-2048): impostazione della frequenza di SCLK (sez. 4.3.2);
- BRM: sempre a 1;
- EN: selezione dell'interfaccia di comunicazione:
  - 0: nessuna interfaccia;
  - 1: interfaccia1;
  - 2: interfaccia2.

4.8.2 configurePattern

```
void configurePattern(uint8_t patternMode);
```

Questa funzione permette di inizializzare le strutture dati `Pattern_setting`, `PWMdata` (sez. 4.7.1), e `readWord` (sez. 4.7.3). Tutte le operazioni di traduzione del pattern definito in formato tabellare sono svolte dal driver, e quindi tutti i dati in arrivo alla funzione `configurePattern` sono pronti per essere inseriti nelle strutture dati. Il formato dei comandi di configurazione dati è il seguente:

0	1	2	3	4	5	6	7
OP	PM	BY		NWC	NC	NCL	
2	0-3	0-65535		1-32	0-255	0-65535	

8	9	10	11	12	13	14	15
RM	NBR	ALD1	ALD2	ALD3	ALD4	ALD5	
0-1	0-255	0-1	0-1	0-1	0-1	0-1	

		x	x+1			y	y+1
DATAPATT			DATACMD				

		z	z+1	z+2	z+3	z+4	z+5
LSCMD			SLVD1	SLVD2	SLVD3	SLVD4	SLVD5
			0-2	0-2	0-2	0-2	0-2

Tabella 4.3: Formato configurePattern

- PM, modalità di generazione del pattern (sez. 4.7.3):

  - 0: Fast mode;
  - 1: Normal mode;
  - 2: Long mode;
  - 3: Loop mode;

- NWC(1-32): numero di parole che compongo un comando (sez. 4.3.1);
- NC(0-255): numero di comandi da inviare nel pattern (sez. 4.3.1);
- NCL(0-65535): numero di ripetizioni del comando in loop utile solo quando PM=3 (sez. 4.7.3);
- RM: tipo di funzione di confronto del comando di fine loop (PM=3) (sez. 4.7.3):
  - 0: confronto OR;
  - 1: confronto AND;
- NBR(0-255): numero di bit coinvolti nel confronto di fine loop (PM=3);
- ALD1-5: definisce quali delle rispettive linee digitali Digital1-5 è coinvolta nel pattern;
  - 0: coinvolta;
  - 1: non coinvolta;
- DATAPATT(15-x): contiene i valori di period e compare register che devono essere inseriti in un array di strutture `PWMdata`;
- DATACMD(x+1-y): contiene le parole che compongono i messaggi SPI che devono essere inseriti in apposite strutture;
- LSCMD(y+1-z): contiene i dati per descrivere il comando di fine loop che devono essere inseriti in un array di strutture `readWord` ed è inviato solo se PM=3;
- SLVD1-5: definisce il valore che devono assumere le linee digitali in caso di fine loop dovuta alle funzioni di confronto ed è inviato solo se PM=3:
  - 0: imposta la linea al valore logico 0;
  - 1: imposta la linea al valore logico 1;
  - 2: il valore della linea rimane invariato.

Le lunghezze dei campi DATAPATT, DATACMD e LSCMD dipendono dal pattern. In ogni caso, è sufficiente indicare nel campo (BY) il numero totale di byte inviati dopo il quarto byte, e attraverso il valore dei campi [4-14] è possibile ricavare le lunghezze di ognuno dei tre campi per suddividere correttamente i dati ricevuti.

DATAPATT

Contiene i valori (2-byte[8]) dei period e compare register che andranno a definire il pattern e saranno inseriti in un array di strutture `PWMdata`. Il formato dei dati è il seguente:

0 1	2 3	4 5	6 7
PR	CS	C1DX_1	C2DX_1
0-65535	0-65535	0-65535	0-65535
		n n+1	n+2 n+3
...	...	PR2	CS2
0-65535	0-65535	0-65535	0-65535
n+4 n+5	n+6 n+7		
C1DX_2	C2DX_2	...	...
0-65535	0-65535	0-65535	0-65535

Tabella 4.4: Formato DATAPATT

- PR: primo valore del registro period, uguale per tutti i timer;
- CS: primo valore del registro compare del segnale CS;
- C1DX_1: primo valore del registro compare 1 della linea Digital_X;
- C2DX_1: primo valore del registro compare 2 della linea Digital_X;
- PR2: secondo valore del registro period, uguale per tutti i timer;
- CS2: secondo valore del registro compare del segnale CS;
- C1DX_2: secondo valore del registro compare 1 della linea Digital_X;
- C2DX_2: secondo valore del registro compare 2 della linea Digital_X;

Il numero di elementi varia a seconda del pattern generato e i dati sono suddivisi in pacchetti che contengono sempre PR e CS, mentre C1DX e C2DX sono definiti solamente per le linee coinvolte nel pattern. Ad esempio, qualora siano coinvolte solamente due linee, saranno definiti solamente i valori di PR, CS, C1D1_1, C2D1_1,
C1D2_1, C2D2_1, PR2, CS2, ecc.
L'ordine di definizione dei valori è quello descritto dal formato: prima vengono definiti PR e CS (sempre presenti), e successivamente i valori dei compare1 e comapre2 solamente delle linee attivate (descritto da AL1-5) e in ordine numerico cresente (prima Digital1-5). Alla fine di un pacchetto si riparte con i nuovi valori di PR, CS, ecc, seguendo sempre la stessa regola.
La funzione `configurePattern`, avvalendosi dei dati contenuti in AL1-5 riempie la struttura `PWMdata` curandosi di mantenere invariato il valore delle linee non coinvolte nel pattern.

[8]I registri compare e period sono a 16-bit e quindi devono essere definiti da 2-byte contigui.

DATACMD

Contiene i dati che definiscono i comandi da inviare e vengono inseriti in appositi array a seconda della modalità di generazione pattern utilizzata. Vengono inviati seguendo il formato:

0	1		n				
W00	W10	...	Wn0	W01	W11	...	Wn1
0-255	0-255		0-255	0-255	0-255		0-255

...	W0c	W1c		Wnc
	0-255	0-255		0-255

Tabella 4.5: Formato DATACMD

- Wnc: definisce la parola numero n=NWC che compone il comando numero c=NC.

LSCMD

Contiene i dati che definiscono il messaggio di stop loop ed è inviato solamente se PM=3. I dati ricevuti vengono inseriti in un array di strutture `readWord` e sono inviati seguendo il formato:

0	1	2	3	4	5	6	7
MK	POS	VU	WN	MK1	POS1	VU	WN1
$2^{0\text{-}7}$	0-7	0-1	0-32	$2^{0\text{-}7}$	0-7	0-1	0-32

	n	n+1	n+2	n+3
...	MKn	POSn	VUn	WNn
	$2^{0\text{-}7}$	0-7	0-1	0-32

Tabella 4.6: Formato LSCMD

- MK: maschera per svolgere le operazioni di AND logico per l'estrazione del bit desiderato dall'intera parola di confronto;
- POS: indica la posizione del bit desiderato all'interno della parola;
- VU: valore di confronto del bit desiderato;
- WN: indica in quale parola è contenuto il bit da verificare.

In una comando possono essere confrontati più di un bit per implementare le funzioni logiche AND o OR (sez. 4.7.3) che definiscono la fine del loop. I dati sono suddivisi in pacchetti di 4 byte. Ogni pacchetto viene memorizzato in un elemento `readWord` di un array. Il nuemro di elementi è definito in NBR.

4.8.3 startPatternTS

```
void startPatternTS(uint8_t patternMode);
```

Come richiesto dalla specifiche di progetto (sez. 2.2), l'avvio del pattern può avvenire in due modi: tramite l'invio di un apposito comando da parte del driver o tramite un evento su un apposito segnale di input triggering. La funzione `startPatternTs` definisce la modalità di avvio del pattern:

0	1	2	3	4	
OP	PM	BY		SM	
3	0-3	1		0-3	

Tabella 4.7: Formato startPattern

- SM, modalità di start:
  - 0: immediata, la ricezione del comando dà immediatamente avvio al pattern;
  - 1: in attesa di un fronte di salita del segnale di trigger-in;
  - 2: in attesa di un fronte di discesa del segnale di trigger-in;
  - 3: in attesa di un fronte di salita o discesa del segnale di trigger-in;
- PM, modalità di generazione del pattern (sez. 4.3.3):
  - 0: Fast mode;
  - 1: Normal mode;
  - 2: Long mode;
  - 3: Loop mode;

L'avvio vero e proprio del pattern è gestito dalla funzione `startPattern`, richiamata immediatamente con SM=0. Con SM=1,2,3 `startPattern` è inserita all'interno di un'apposita funzione (`startFormPin`) che viene richiamata in risposta ad un PIN_interrupt. Un PIN_interrupt è un particolare tipo di interrupt gestito facilmente con l'apposita APP di DAVE™ che viene generato in risposta alle commutazioni di un apposito pin di trigger-in definito dall'utente.

```
void startPattern(uint8_t patternMode);
```

La funzione `startPattern` si occupa di:

- resettare tutte le variabili di indicizzazione degli array di pattern e comandi;
- abilitare le corrette funzioni di interrupt associate alla modalità di generazione del pattern definita in PMW;
- inizializzazione dei timer del modulo CCU8 (`Initpatterndata`;,
- aggiornare la variabile static digital value con il valore che sarà assunto dalle linee a fine pattern;
- inizializzare il buffer del modulo USIC con i comandi da inviare;
- avvio sincronizzato dei timer.

L'inizializzazione del pattern prevede prima dell'avvio uno slot di tempo (due periodi dei segnali PWM da 15μs ciascuno), in cui tutte le linee digitali mantengono il valore precedente allo start mentre il CS viene portato allo stato inattivo. Questa fase è richiesta per poter caricare il buffer assicurandosi che il segnale di abilitazione della trasmissione DX2S=CS sia inattivo.

4.8.4 configureStaticDigitalLine

```
void configureStaticDigitalLine();
```

Questa funzione permette di impostare il valore statico delle linee digitali. Il formato del comando da inviare è il seguente:

0	1	2	3	4	5	6	7
OP	PM	BY		CS1	CS2	D1	D2
4	0-3	7		0-2	0-2	0-2	0-2

8	9	10	11	12	13	14	15
D3	D4	D5					
0-2	0-2	0-2					

Tabella 4.8: Formato configureStaticDigitalLine

- CS1, CS2 e D1-5 identificano rispettivamente CS dell'interfaccia1, CS dell'interfaccia2 e linee di commutazione Digital1-5.
  A ogni richiamo della funzione deve essere definito il valore di tutte le 7 uscite che compongono il pattern, scegliendo tra le tre opzioni disponibili:
  - 0: imposta la linea al valore logico 0;
  - 1: imposta la linea al valore logico 1;
  - 2: il valore della linea rimane invariato.

4.8.5 resetSystem

```
void resetSystem();
```

Questa funzione permette di resettare via software il microcontrollore qualora si riscontrino malfunzionamenti dell'interfaccia. Dopo un reset il programma riparte dalla funzione main, e tutte le variabili locali e globali vengo resettate.
Nel caso in cui non sia possibile inviare il comando è possibile resettare l'interfaccia tramite un pulsante (reset hardware) presente sulla board XMC_4700 (sez. 3.2.3). Il formato del comando di reset è il seguente:

0	1	2	3	n
OP	PM	BY		
5	0-3	0		

Tabella 4.9: Formato resetSystem

4.8.6 readPatternData

```
void readPatternData(uint8_t patternMode);
```

Nel caso di un pattern di comunicazione SPI questa funzione permette di leggere i comandi SPI ricevuti durate la comunicazione (MISO). La lettura dei comandi SPI può avvenire solo alla fine del pattern. Il formato di lettura è il seguente:

0	1	2	3	n
OP	PM	BY		
6	0-3	0		

Tabella 4.10: Formato lettura readPatternData

I comandi SPI ricevuti durante il pattern vengono inviati sulla porta virtuale di comunicazione e si suppone che la richiesta di lettura (MISO) venga dallo stesso dispositivo driver utilizzato per la configurazione e generazione dei comandi e del pattern. Il driver è quindi a conoscenza dei valori contenuti nella struttura `Pattern_setting`. In particolare, la conoscenza dei valori di`numWordCMD` e `numCMD` permette al driver di conoscere il numero esatto di parole che devono essere lette dopo l'invio di una richiesta di lettura dei comandi. Ogni parola che compone i comandi (massimo 8-bit) è definita da un singolo byte di comunicazione, quindi il driver è in grado di suddividere autonomamente i byte ricevuti per ricostruire i comandi MOSI che possono essere inviati uno dopo l'altro senza ulteriori informazioni. Il formato è il seguente:

0	1	2					n
W0	W1	W2					Wn
0-255	0-255	0-255					0-255

Tabella 4.11: config PWM

4.8.7 pwmStart-Stop

```
void pwmStart_Stop();
```

Questa funzione permette di impostare, avviare e stoppare due segnali PWM indipendenti dal pattern. Come richiesto dalle specifiche di progetto (sez. 2.2), l'interfaccia permette la generazione di due segnali PWM a frequenza, duty cycle e livello di stato passivo programmabile dall'utente. Inoltre, è possibile sincronizzare l'avvio delle due PWM, e agendo sul livello di stato passivo è possibile ottenere segnali PWM complementari.
Queste PWM sono implementate utilizzando il modulo CCU4, molto simile al modulo CCU8 utilizzato per il pattern digitale. Il formato del comando di gestione pattern è il seguente:

0	1	2	3	4	5	6	7
OP	PM	BY		ACT	DP1	FP1	
7	0-3	13		0-7	0-100	2.000-40.000.000	

8	9	10	11	12	13	14	15
		PSL1	DP2	FP2			
		0-1	0-100	2.000-40.000.000			

16	
PSL2	
0-1	

Tabella 4.12: Formato pwmStart_stop

- ACT, definisce come agire sulle PWM;
  - 0: start PWM1, stop PWM2;
  - 1: start PWM2, stop PWM1;
  - 2: stop PWM2;
  - 3: stop PWM1;
  - 4: start PWM2;
  - 5: start PWM1;
  - 6: start PWM1, start PWM2 sincronizzato;
  - 7: stop PWM1, stopPWM2;

- DP1(0-100): duty cycle della PWM1 in percentuale;
- FP1(2000-40.000.000); frequenza PWM1 espressa in Hz;
- PSL1: passive data level PWM1;
- DP2(0-100): duty cycle della PWM2 in percentuale;
- FP2(2000-40.000.000): frequenza PWM2 espressa in Hz;
- PSL2: passive data level PWM2.

4.8.8 readStaticDigitalLine

```
void readStaticDigitalLine();
```

Questa funzione permette di leggere il valore logico di tutte le 5 linee di commutazione del pattern al momento della richiesta. Il formato del comando di lettura è il seguente:

0	1	2	3	n
OP	PM	BY		
8	0-3	0		

Tabella 4.13: Formato lettura readStaticDigitalLine

Il valore logico delle linee è contenuto nella struttura dati `staticDigitalValue` e viene restituito al richiedente attraverso l'invio di un comando sulla porta seriale. Il formato di questo comando è:

0	1	2	3	4	
LV1	LV2	LV3	LV4	LV5	
0-1	0-1	0-1	0-1	0-1	

Tabella 4.14: Formato scrittura readStaticDigitalLine

- LV1-5 identificano rispettivamente il valore delle linee digitali Digital1-5:
  - 0: valore logico ;
  - 1: valore logico 1.

4.9 Mappatura dei pin

La configurazione base delle periferiche e degli interrupt è stata sviluppata con l'ausilio delle APP fornite da DAVE^{TM}. Queste APP, assieme a un'apposita interfaccia di mappatura dei pin, permettono di scegliere e impostare i pin di I/O dei segnali prodotti dai moduli, ovvero i segnali:

- SCLK, MOSI, MISO in uscita dalle interfacce SPI del modulo USIC;
- DX2S in ingresso ad ognuna delle due interfacce USIC;
- CS1 ,CS2, Digital_(1-5) in uscita dai moduli CCU8 PWM per la generazione del pattern digitale;
- PWM_1 e PWM_2 in uscita dai moduli CCU4 PWM per la generazione delle PWM programmabili;
- trigger-in in ingresso per il PIN_interrupt di avvio pattern;

La board XMC41_RELAX_V1 (sez. 3.2.3) permette di accedere agevolmente a 40 pin del micronotrollore figuraaa. Ogni pin di I/O è riservato ad alcune specifiche funzioni e non può essere connesso a tutti i moduli. Ciò limita il numero di segnali digitali, ma soprattutto il numero di interfacce seriali disponibili sull'interfaccia.
Per come è sviluppato il firmware, un pattern può essere inviato su ognuno dei 5 canali USIC disponibili, ma in un pattern è consentito l'uso di una sola interfaccia alla volta, ovvero non è consentito l'uso in parallelo di due o più interfacce. Per cambiare interfaccia di uscita si deve attendere la fine del pattern e configurare un nuovo pattern cambiando il parametro di configurazione SPI (EN).
Purtroppo, con i 40 pin accessibili sulla board è possibile implementare solamente due interfacce SPI seriali ma, progettando un'apposita board con accesso a tutti i pin del microcontrollore, e con una semplice modifica al firmware è possibile implementare fino a cinque interfacce seriali. Lo stesso discorso vale per il modulo PWM, con i pin accessibili dalla board sono generati sette segnali (CS1_1, CS2_2, Digital_(1-5)) che possono aumentare con lo sviluppo di una board dedicata.

Come visto in sez. 4.7.2, per garantire la possibilità di personalizzare la polarità del CS è necessario utilizzare entrambe le uscite del timer impostato come generatore del segnale CS. La prima uscita (CS_x_1) con PSL=1 è collegata direttamente al segnale di validazione della trasmissione DX2S_x, mentre la seconda uscita (CS_x_2) è utilizzata come vero e proprio segnale di uscita CS dell'interfaccia SPI.
In caso di comunicazione SCI valgono le stesse considerazioni fatte per SPI, infatti il segnale dati è rappresentato dal MOSI della comunicazione SPI, mentre il segnale SYNC è rappresentato dalla linea Digital_1 per l'interfaccia1 e Digital_2 per l'interfaccia2.

La comunicazione con il driver (PC) avviene per mezzo di comunicazioni seriali attraverso la connessione USB.
La board XMC47_RELAX_V1 è provvista di due porte USB ma solamente la porta microUSB X100(Figura 4.18) è abilitata per le comunicazioni. La connessione con il PC deve quindi avvenire su questa porta e all prima connessione di una nuovo dispositivo è

necessaria l'installazione di un apposito driver per la porta USB fornito con il firmware. Il dispositivo sarà poi gestito dal PC come una generica porta COM.

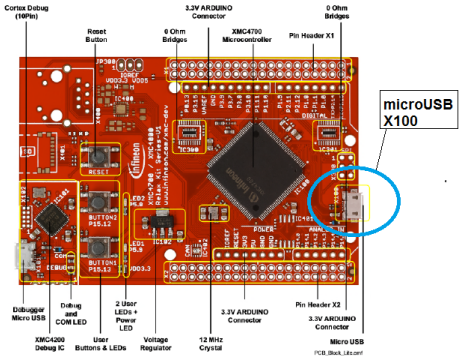


Figura 4.18: MicroUSB x100.

- Interfaccia1 comunicazioni seriali:
  - SCLK_1: P0.10;
  - MOSI_1/dati_1: P0.1;
  - MISO_1: P0.0;
  - CS_1_1: P5.11;
  - CS_1_2: P0.2;
  - DX2S_1: P0.9[9].
- Interfaccia2 comunicazioni seriali:
  - SCLK_2: P6.2;
  - MOSI_2/dati_2: P6.4;
  - MISO_2: P3.13;
  - CS_2_1: P6.1;
  - CS_2_2: P6.0;
  - DX2S_2: P3.0[10].
- Linee digitali:
  - Digital_1/SYNC_1: P0.6;
  - Digital_2/SYNC_2: P0.3;
  - Digital_3: P5.10;
  - Digital_4: P1.14;
  - Digital_5: P1.15.
- Pin interrupt:
  - trigger-in: P0.8;
- PWM configurabili:
  - PWM_1: P3.11;
  - PWM_2: P3.12.

[9]CS_1_1 (P5.11) e DX2S_1 (P0.9) devono essere collegati assieme.
[10]CS_2_1 (P6.1) e DX2S_2 (P3.0) devono essere collegati assieme.

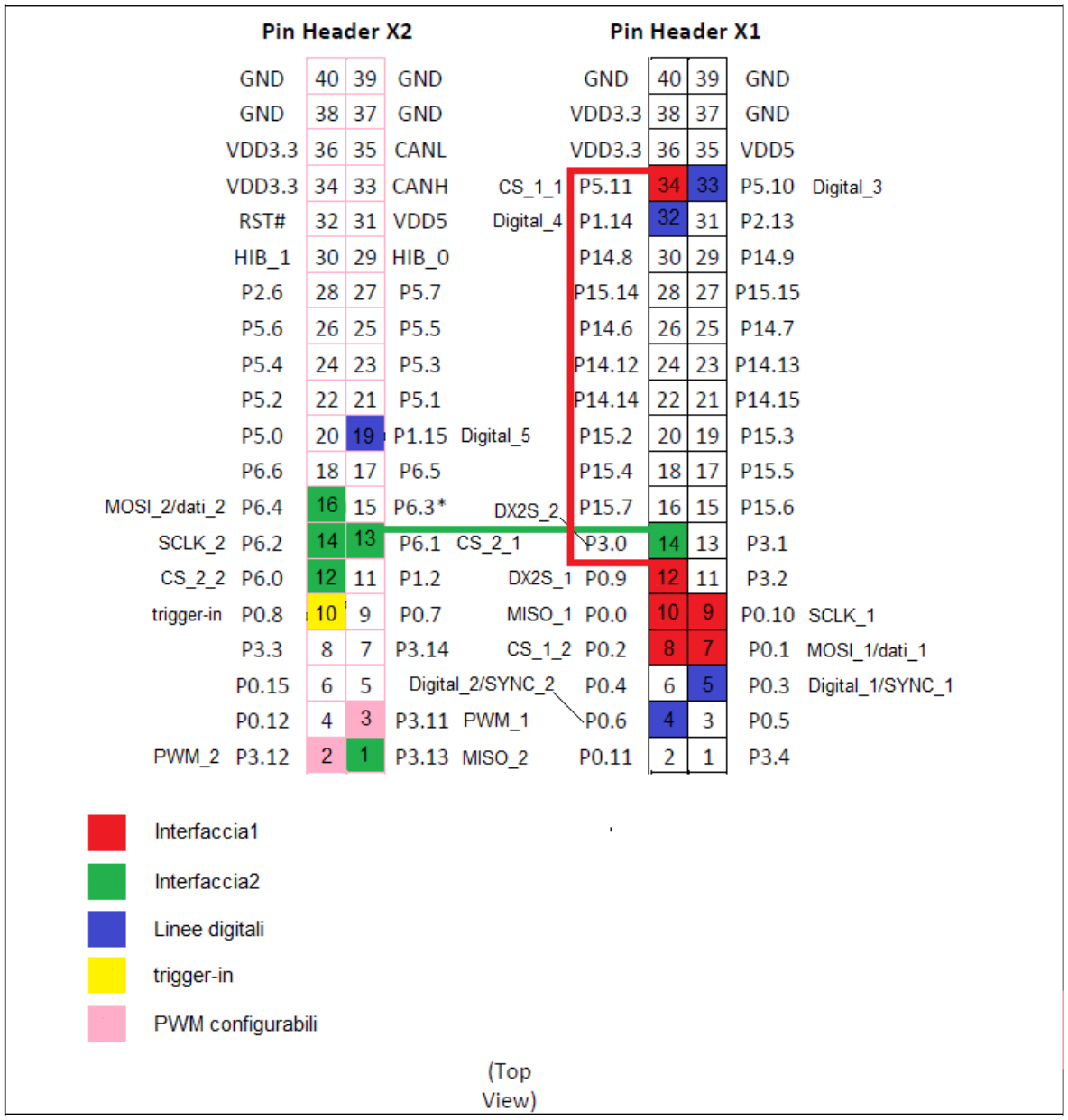


Figura 4.19: Mappatura pin I/O.

Capitolo 5

Tabella di descrizione pattern

Le specifiche di progetto (sez. 2.2.4) richiedono di poter definire il pattern da inviare attraverso una tabella, in cui ogni riga rappresenta un istante temporale, le commutazioni digitali e le comunicazioni seriali a esso associate. La struttura del firmware (sez. 4.7) impone all'utente regole e vincoli da rispettare per generare correttamente il pattern.

5.1 Formato della tabella

Come vedremo nel prossimo, il driver sviluppato per la generazione del pattern si occupa di tradurre i dati che devono essere forniti dall'utente in un formato tabellare, come richiesto dalle specifiche di progetto.
Per poter estrarre correttamente i dati dalla tabella si è definito un formato standard, che associa valore e posizione di un elemento a determinate funzioni. Non è importante come o con quale strumento sia generata la tabella, ma è fondamentale collocare i valori di configurazione esattamente come descritto in tab. 5.1, rispettando spazi vuoti ed esatta posizione dei valori (riga e colonna).

	0	1	2	3	4	5	6	7	8	9	10
0	User text										
1											
2	CSpol	0-3									
3	CLKpol	0-3									
4	TIDres	0-1									
5	PDIV	4-2048									
6	TID	1-128									
7	CMDnBit	1-420									
8	CODMsg	0-1									
9	COMProt	0-1									
10	CSDelay	0-100									
11	SelSPI	1-2									
12											
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0,000-max	HEX o BIN	2-200	0,100-max	0-1	0-1	0-1	0-1	0-1	x10..AND,OR	x10..
15	...	...	...	...	...	...	...	...	...	...	...

Tabella 5.1: Formato tabella ingresso dati

In colonna 1, da riga 2 a 11, sono definiti i parametri di configurazione dei comandi SPI in base a valori numerici che rappresentano:

- CSpol, polarità del segnale CS:
  - 0: attivo basso;
  - 1: attivo alto;
  - 2: sempre alto;
  - 3: sempre basso.
- CLKpol, polarità del segnale CLK:
  - 0: CPOL=0, CPHA=1, CLK inattivo basso, scrittura sui fronti di salita e lettura sui fronti di discesa di CLK;
  - 1: CPOL=0, CPHA=1, CLK inattivo basso, scrittura sui fronti di salita e lettura sui fronti di discesa di CLK;
  - 2: CPOL=1, CPHA=0, CLK inattivo alto, scrittura sui fronti di salita e lettura sui fronti di discesa di CLK;
  - 3: CPOL=1, CPHA=1, CLK inattivo alto, scrittura sui fronti di discesa e lettura sui fronti di salita di CLK.
- TIDres, risoluzione temporale (tq) dei ritardi $tid = tnf$:
  - 0: $tq = \frac{1}{f_{CLK}}$;
  - 1: $tq = 25ns$.
- PDIV(4-2048), frequenza del segnale CLK: $f_{CLK} = \frac{40MHz}{PDIV}$;
- TID(1-128), lunghezza dei ritardi $tid = tnf$ se, TID>32 solo multipli di 2, 3 o 4:
  - se TIDres=0: $tid = tnf = \frac{1}{f_{CLK}} \times TID$;
  - se TIDres=1: $tid = tnf = 25ns \times TID$.
- CMDnBit(1-420), numero di bit dei comandi, se CMDnBit>64 solo multipli di 8;
- CODMsg, codifica con la quale saranno definiti i comandi nella colonna CMD:
  - 0: binaria (BIN);
  - 1: esadecimale (HEX) solo se CMDnBit è multiplo di 4.
- COMProt, protocollo di comunicazione:
  - 0: SPI;
  - 1: SCI[1].
- CSDelay(0-100), parametro di regolazione per compensare il ritardo introdotto dallo schema di validazione[2] (default: 50) espresso in ns;

[1]Nel caso in cui il protocollo sia SCI, le linee digitali Digital_1 e Digital_2 sono utilizzate come segnale di SYNC rispettivamente dell'interfaccia1 e interfaccia2 e sono inutilizzabili per commutazioni digitali. Possono quindi essere utilizzate solamente le linee Digital_(3-5).

[2]Ad alte frequenze ci possono essere problemi con la sincronizzazione del segnale SYNC (SCI). Si consiglia di agire su CSDelay per allineare SYNC con il fronte del segnale dati.

- SelSPI, selezione dell'interfaccia SPI/SCI utilizzata:
  - 1: Interfaccia1;
  - 2: Interfaccia2.

A partire dalla riga 14 sono definiti i parametri per la generazione del pattern, ovvero: gli istanti temporali di avvio delle comunicazioni seriali, gli istanti in cui avvengono le commutazioni delle linee digitali, i valori dei comandi da inviare e l'eventuale configurazione dei comandi in loop.
Ogni colonna della tabella assume il seguente significato:

- TIME(μs): definisce l'istante temporale a cui sono riferite le azione descritte nelle successive colonne. È espresso in μs, ma la risoluzione temporale è nell'ordine dei ns (per esempio 10,265 μs).
  Per definire un pattern è necessaria la presenza dell'istante temporale 0, dove saranno definite quali sono le linee digitali coinvolte nel pattern. Non è invece importante l'ordine di definizione degli istanti temporali.

- CMD: definisce il comando da inviare all'istante temporale indicato sulla stessa riga ma nella colonna TIME. Il comando può essere codificato in binario (0-1) o esadecimale (0-F), in base al valore di CODmsg e al numero di bit del comando CMDnBit.

- REPEAT e TIC: si riferiscono al comando indicato nella stessa riga ma nella colonna CMD, e definiscono rispettivamente quante volte deve essere ripetuto il comando e la distanza tra fine e inizio del comando successivo (*tic*). Il minimo valore accettato per TIC è 0,100μs e dipende dalla modalità di generazione del pattern.

- Digital_(1-5): si riferiscono all'istante temporale indicato sulla stessa riga ma nella colonna TIME, definiscono le commutazioni delle linee digitali(1-5) e possono assumere tre valori:
  - 0: commutazione della linea high to low;
  - 1: commutazione della linea low to high;
  - ""[3]: nessuna commutazione.

  È importante definire all'istante iniziale (TIME=0) quali sono le linee coinvolte nel pattern indicando lo stato iniziale della linea (1 o 0). Se la cella rimane vuota la rispettiva linea digitale non è ritenuta attiva e il suo stato non varia durante tutto il pattern.

- StopCMD: si riferisce al comando indicato nella stessa riga ma nella colonna CMD e ha validità solamente se REPEAT>1. Definisce come deve essere il comando di comparazione (sez. 4.7.3) che pone fine al loop, deve essere espresso sempre in codifica binaria e può contenere 3 tipi di carattere:
  - 0: valore bit=0;
  - 1: valore bit=1;

[3]Il simbolo "" indica cella vuota.

  - x: non rilevante.

  Deve quindi essere specificato solo il valore dei bit del comando che devono essere comparati con il comando ricevuto (0-1), mentre se un bit non partecipa alla comparazione deve essere indicato con il valore x. Alla fine del comando deve essere inserita una delle due parole chiave AND o OR, che definiscono il tipo di comparazione tra i due messaggi:

  - AND: il loop termina quando tutti i bit diversi da x del comando specificato in StopCMD sono uguali al comando ricevuto (MOSI);
  - OR: il loop termina quando anche uno solo dei bit diversi da x del comando specificato in StopCMD sono uguali al comando ricevuto (MOSI).

  L'impostazione di StopCMD è interpretata come la volontà da parte dell'utilizzatore di generare un pattern in modalità loop pattern.

- StopLV: si riferisce al comando indicato nella stessa riga ma nella colonna CMD, e ha validità solamente se REPEAT>1 e se StopCMD è diverso da "". Definisce il valore che devono assumere le linee digitali qualora il loop termini a causa di un riscontro positivo delle funzioni di comparazione. Deve essere definito il valore di tutte e cinque le linee digitali, scegliendo fra:

  - 0: valore logico della linea digitale (0) al termine del loop;
  - 1: valore logico della linea digitale (1) al termine del loop;
  - x: il valore logico della linea rimane invariato .

Per generare un pattern in modalità Loop mode è necessario compilare entrambi i campi StopCMD e StopLV. La posizione dei caratteri (0, 1, x) all'interno della stringa rappresenta la posizione del bit nel comando di comparazione in StopCMD e il numero di linea digitale in StopLV.

5.2 Regole di definizione del pattern

Ci sono regole e limitazioni che devono essere rispettate in fase di definizione del pattern dettate dalla struttura del firmware.

5.2.1 Vincoli

I vincoli e limitazioni per la generazione del pattern sono rappresentati da due parametri che variano a seconda della modalità di generazione[4]:

- SPI_limit(2μs o 5μs): incide sulla minima distanza tra elementi consecutivi del pattern (comunicazioni seriali e commutazioni digitali). Rappresenta il tempo minimo necessario per evitare la sovrapposione delle istanze di interrupt generate da eventi di period match (sez. 4.7);

[4] I valori dei parametri sono definiti in sez. 5.2.2.

- ticLimit(da 0,100μs a 30μs): incide sulla minima distanza tra la fine di un comando SPI/SCI e l'inizio del successivo. Rappresenta il tempo minimo necessario per garantire al modulo USIC di inviare il comando (sez. 4.7.3).

Per definire un pattern correttamente è necessario che siano verificate le seguenti condizioni:

1. la distanza tra commutazioni digitali, anche di linee digitali diverse, che avvengono prima o dopo l'invio dei comandi SPI/SCI, deve essere maggiore di SPI_limit. L'unica eccezione è rappresentata dall'ultima commutazione delle linee digitali che avviene prima del primo comando SPI/SCI, la quale può essere collocata in qualsiasi istante;

2. La distanza tra la penultima commutazione delle linee digitali che avviene prima del primo comando SPI/SCI e l'inizio del primo comando stesso deve essere maggiore di SPI_limit-CMDLenght, dove CMDLenght rappresenta il tempo in cui il CS è attivo (Figura 5.1). Qualora siano definite una o nessuna commutazione prima del primo comando, la distanza tra l'istante 0 e l'istante di invio del comando deve essere maggiore di SPI_limit-CMDLenght;

3. la distanza tra l'inizio di due comandi deve essere sempre maggiore di SPI_limit e maggiore della distanza CMDLenght+ticLimit;

4. all'interno del lasso temporale che va dalla fine di un comando alla fine del successivo[5] non esistono limitazioni temporali per definire commutazioni digitali (sez. 4.7.1). Tali commutazioni possono avvenire in qualsiasi istante, ma il loro numero è limitato. Per ogni linea digitale possono infatti avvenire solamente le seguenti commutazioni:

   - 2 commutazioni: low to high seguito da high to low;
   - 1 commutazione: low to high;
   - 1 commutazione: high to low.

   Per il primo comando questo lasso di tempo va dalla penultima commutazione digitale che avviene prima del primo comando fino alla fine del comando stesso.

- CMDLenght=tid+tfd+tcmd (Figura 5.1): rappresenta il tempo tra inizio e fine di un comando (CS attivo), non varia durante il pattern ed è definito dai parametri di configurazione dei comandi SPI TIDres, PDIV, TID, CMDnBit e CSDelay, che devono essere impostati prima di definire il pattern. CMDLenght può essere calcolato in due modi:

  – se TIDres=0:
  CMDLenght = $[(\frac{PDIV}{40MHz} \times CMDnBit) + (2 \times \frac{PDIV}{40MHz} \times TID) + (2 \times CSDealy)]$;
  – se TIDres=1:
  CMDLenght = $[(\frac{PDIV}{40MHz} \times CMDnBit) + (2 \times 25ns \times TID) + (2 \times CSDealy)]$.

[5]La fine di un comando può essere calcolata aggiungendo il valore di CMDLenght all'istante di inzio comando definito dalla colonna TIME.

- tic (Figura 5.1): rappresenta il tempo tra fine di un comando e inizio del successivo (CS inattivo), può variare durante il pattern e può essere definito per mezzo dell'interfaccia di inserimento dei dati in due modi:
  - nel caso di comandi singoli, è possibile definire l'avvio di ogni comando nel campi CMD della tabella (sez. 5.1), specificandone l'inizio nella relativo campo TIME. Considerando che la lunghezza dei comandi CMDLenght è fissa e conosciuta, basta calcolare gli istanti di avvio di ogni comando per ottenere specifici tic;
  - in caso di ripetizione (REPEAT) dello stesso comando, tic può essere definito nell'apposta campo (TIC) e sarà il driver a occuparsi di inserire i comandi nella corretta posizione per garantire il valore di tic desiderato. Utilizzando la colonna REPEAT, per inserire nuovi comandi l'utente deve tener conto della presenza dei comandi ripetuti, anche se non sono visualizzati nella tabella, per evitare di sovrapporre i comandi.

  Esiste una soglia sotto la quale il quale tic non può scendere, ovvero ticLimit. Questa soglia varia da 0,100μs a 30μs a seconda della modalità di pattern utilizzato.

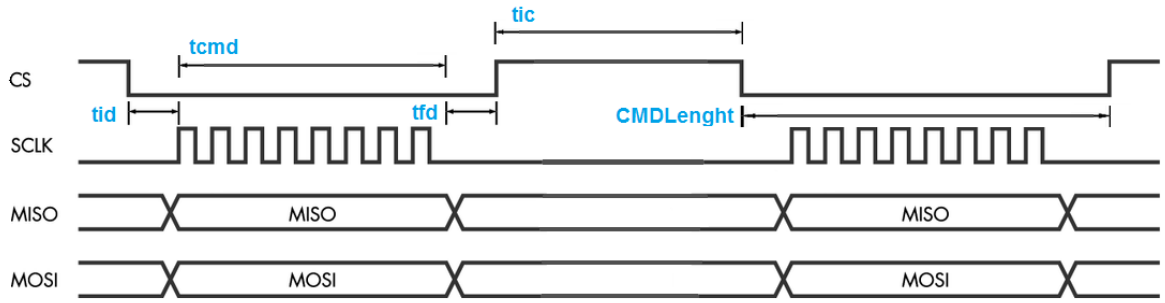


Figura 5.1: Parametri temporali della comunicazione SPI.

5.2.2 Vincoli delle modalità di generazione del pattern

Le modalità di generazione del pattern si differenziano per numero e tipo di comandi e numero di linee di commutazione digitale disponibile. Queste caratteristiche determinano i limiti del pattern definiti da: SPI_limit e ticLimit.
Sono disponibili quattro diverse modalità di generazione del pattern:

Fast mode

- comandi seriali con lunghezza minore di 64-bit;
- numero massimo di bit dell'intera comunicazione 512-bit;
- pattern con 1 linea digitale (Digital_1[6]) sincronizzata con il CS della comunicazione;
- SPI_limit=2μs;

[6]In Fast mode deve esser utilizzata una sola linea digitale, che deve obbligatoriamente essere Digital_1.

- ticLimit:
  - se CMDLenght<SPI_limit allora ticLimit=SPI_limit-CMDLenght;
  - se CMDLenght>SPI_limit allora ticLimit=0,100μs.
- solo protocollo SPI.

Normal mode

- comandi seriali con lunghezza minore di 64-bit;
- numero masssimo di bit dell'intera comunicazione 512-bit;
- pattern con 2-5 linee digitali sincronizzate con il CS della comunicazione;
- SPI_limit=5μs;
- ticLimit:
  - se CMDLenght<SPI_limit allora ticLimit=SPI_limit-CMDLenght;
  - se CMDLenght>SPI_limit allora ticLimit=0,100μs.
- solo protocollo SPI.

Long mode

- comandi seriali con lunghezza maggiore di 64-bit (multipli di 8-bit) oppure numero di bit dell'intera comunicazione maggiore di 512-bit;
- pattern con 1-5 linee digitali sincronizzate con il CS della comunicazione.
- SPI_limit=5μs;
- ticLimit=31μs;
- protocollo SPI e protocollo SCI. Nel caso in cui il protocollo sia SCI, le linee digitali Digital_1 e Digital_2 sono utilizzate come segnale di SYNC rispettivamente dell'interfaccia1 e interfaccia2 e sono inutilizzabili per commutazioni digitali. Possono quindi essere utilizzate solamente le linee Digital_(3-5).

Loop mode

- comandi seriali con qualsiasi lunghezza;
- pattern con 1-5 linee digitali sincronizzate con il CS della comunicazione;
- SPI_limit=5μs;
- ticLimit=31μs;
- protocollo SPI e protocollo SCI. Nel caso in cui il protocollo sia SCI, le linee digitali Digital_1 e Digital_2 sono utilizzate come segnale di SYNC rispettivamente dell'interfaccia 1 e interfaccia 2 e sono inutilizzabili per commutazioni digitali. Possono quindi essere utilizzate solamente le linee Digital_(3-5).
- devono essere definiti obbligatoriamente i campi StopCM e StopLV in corrispondenza di un istante temporale con un comando e con REPEAT>2.

5.2.3 Esempi di tabelle

Per capire meglio come gestire le limitazioni procediamo con qualche esempio.

	0	1	2	3	4	5	6	7	8	9	10
0	User text										
1											
2	CSpol	0									
3	CLKpol	0									
4	TIDres	1									
5	PDIV	8									
6	TID	2									
7	CMDnBit	16									
8	CODMsg	1									
9	COMProt	0									
10	CSDelay	50									
11	SelSPI	1									
12											
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0				1	1					
15	6					0					
16	8.2				0	1					
17	8.5	FC				0					
18	13.8	AA			1						

Tabella 5.2: Esempio 1.

In questo caso (tab. 5.2), abbiamo due comandi con 16-bit ciascuno e due linee linee digitali che partecipano al pattern.
Siamo quindi in Normal mode e SPI_limit=5μs.
CMDLenght = $[(\frac{PDIV}{40MHz} \times CMDnBit) + (2 \times 25ns \times TID) + (2 \times CSDealy)]$ e quindi,
CMDLenght = $[(\frac{20}{40MHz} \times 8) + (2 \times \ 25ns \times 2) + (2 \times 50ns)]$=3400ns=3,4$\mu$s.
CMDLenght<SPI_limit e quindi ticLimit=SPI_limit-CMDLenght=1.6μs

Tutte e quattro le condizioni sono verificate. Mantenendo le stesse configurazioni dell'SPI possiamo vedere altri esempi variando la definizione del pattern:

- esempio 2 (tab. 5.3): non è rispettata la prima condizione. Infatti, le due commutazioni digitali che avvengono prima dell'inizio delle comunicazioni SPI (r14 - r15) non rispettano la distanza SPI_limit=5μs;
- esempio 3 (tab. 5.4): non è rispettata la quarta condizione. Infatti, nel lasso di tempo che va dalla penultima commutazione digitale prima del primo comando fino alla fine del comando stesso (r15 - [r17+CMDLenght]) avvengono due commutazione, high to low seguita da low to high;
- esempio 4 (tab. 5.5): non è rispettata la terza condizione. Infatti, la distanza tra l'inizio dei due comandi SPI (r17 - r18) è inferiore alle distanze SPI_limt e CMDLenght+ticLimit;

- esempio 5 (tab. 5.6): tutte le condizioni sono rispettate. In particolare è mostrato un esempio di utilizzo delle colonne REPEAT e TIC. Sono definiti 5 comandi (FCBB) che si ripetono con distanza fissa 2.070μs<ticLimit a partire dall'istante temporale 8,5 (r16);
- esempio 6 (tab. 5.7): tutte le condizioni sono rispettate. In particolare è mostrato un esempio di Loop Pattern. In questo caso ticLimit=31. A partire dall'istante 100(r18) è ripetuto al massimo per 20 volte il comando 0001. Il loop termina con la variazione della linea Digital_2 qualora MISO[10]=0 oppure MISO[11]=1.

	0	1	2	3	4	5	6	7	8	9	10
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0				1	1					
15	3					0					
16	8.2				0	1					
17	8.5	FCBB				0					
18	13.8	AACC			1						

Tabella 5.3: Esempio 2.

	0	1	2	3	4	5	6	7	8	9	10
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0				1	1					
15	6					0					
16	8.2				0	1					
17	8.5	FCBB			1	0					
18	13.8	AACC									

Tabella 5.4: Esempio 3.

	0	1	2	3	4	5	6	7	8	9	10
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0				1	1					
15	6					0					
16	8.2				0	1					
17	8.5	FCBB			1	0					
18	11	AACC									

Tabella 5.5: Esempio 4.

	0	1	2	3	4	5	6	7	8	9	10
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0				1	1					
15	6					0					
16	8.5	FCBB	5	2,070							
17	50				0	1					
18	100	AACC									

Tabella 5.6: Esempio 5.

	0	1	2	3	4	5	6	7	8	9	10
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0				1	1					
15	6										
16	50	AAC0			0						
18	100	0001	20	35						xxxx10xxxxxxxxx OR	x0xxx

Tabella 5.7: Esempio 6.

Capitolo 6

Driver

Il driver si occupa di:

- tradurre i dati di definizione del pattern dal formato descritto in sez. 5.1 ai messaggi di configurazione e gestione visti sez. 4.8.1 e sez. 4.8.2;
- comunicare con l'interfaccia attraverso l'invio dei comandi di configurazione prodotti dalla traduzione della tabella, dei comandi di lettura dati, configurazione statica delle linee, start del pattern ecc. sez. 4.8.

Il driver è stato implementato in Labview al fine di integrare l'interfaccia in un ampio sistema di test. All'utente sono forniti quindi appositi file.vi, che permettono di svolgere le attività di comunicazione e di generazione dei dati.
In questo caso, a differenza di quanto fatto per il firmware del microcontrollore, non è possibile scendere in dettaglio nella descrizione del codice dei file.vi (VI[1]) che costituiscono il driver. La natura grafica dell'ambiente di sviluppo Labview e il numero di file prodotti (circa 50 elementi[2]) rende difficile documentare dettagliatamente ogni singolo file.
Nelle seguenti sezioni, quindi ci concentreremo sulla descrizione della funzione svolta dai singoli VI e di come devono essere utilizzati. Per maggiori informazioni sulla struttura interna e sulle soluzioni tecniche adottate, si invita l'utente a visionare direttamente i file.vi, all'interno dei quali si possono trovare numerosi commenti al codice.

6.1 Tree

I VI forniti all'utente per la gestione dell'interfaccia e la configurazione dei pattern da inviare sono riassunti in Figura 6.1, e possono essere suddivisi in tre categorie:

- VI che si occupano di gestire l'interfaccia (sez. 6.2):
  - main.vi: è una GUI che permette la completa gestione di tutte le funzioni messe a disposizione dell'interfaccia. Fornisce anche appositi tool che aiutano l'utente nella compilazione della tabella di descrizione del pattern;

[1] Un programma o sottoprogramma Labview è denominato VI.
[2] Nel proseguo della trattazione, in alcuni casi, si è utilizzato il termine elemento per indicare un VI.

  - sendStandardMsg.vi: permette di generare pattern standard ovvero pattern che non richiedono particolari caratteristiche. È consentito l'invio di molteplici comandi SPI tutti con lo stesso tic e solamente una commutazione di linee digitali. Il punto di forza di questo vi è la semplicità di utilizzo[3].

- VI che si occupano di controllare i dati della tabella di ingresso e generare i comandi di configurazione (sez. 6.3):

  - patternFromTableGenerator.vi (sez. 6.4): è il cuore del driver, il più complesso dei VI, e si occupa di tradurre la tabella di descrizione del pattern (sez. 5.1) generando i comandi di configurazione SPI e configurazione del pattern. Nelle operazioni di traduzione è implementato un sistema di controllo del pattern che verifica il rispetto delle regole di definizione.

- VI che si occupano di comunicare con l'interfaccia, inviando e ricevendo comandi rispettando il formato dati visto in sez. 4.8.

  - sendConfigureSPI.vi: invia i comandi di configurazione SPI generati da patternFromTableGenerator.vi;
  - sendConfigurePattern.vi: invia i comandi di configurazione pattern generati da patternFromTableGenerator.vi;
  - sendStartPattern.vi: invia i comandi di avvio del pattern;
  - sendConfigureStaticDigitalLine.vi: invia i comandi di configurazione statica delle linee digitali;
  - sendSystemReset.vi: invia il comando di reset;
  - readPatternData.vi: invia il comando di lettura dei datii ricevuti dall'interfaccia durante la comunicazione SPI;
  - sendConfigureAndStartPWM.vi: invia il comando di configurazione dei segnali PWM;
  - sendReadStaticDigitalLine.vi: invia il comando di lettura del valore delle linee digitali.

La comunicazione con l'interfaccia avviene grazie ai driver di comunicazione seriale VISA.

[3]Il 90% dei pattern generati per caratterizzare i dispositivi sono semplici pattern di qualche comando che non richiedono particolari caratteristiche. È quindi essenziale poter generare pattern in modo semplice e veloce.

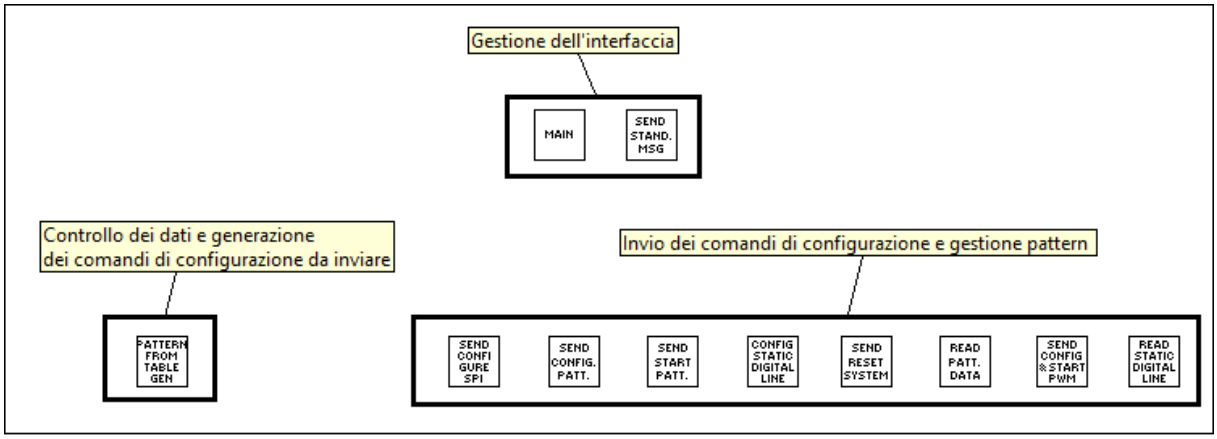
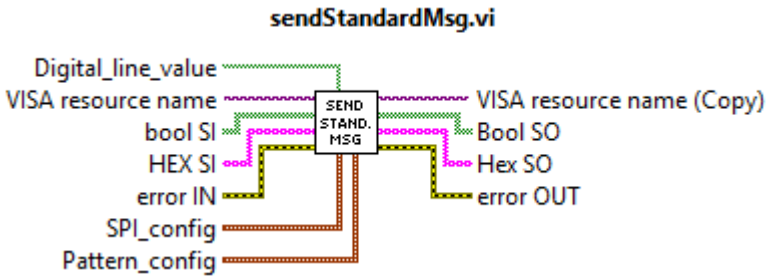


Figura 6.1: Tree.

6.2 Gestione dell'interfaccia

Tali VI riuniscono gli elementi che si occupano di controllo e generazione del pattern e gli elementi che si occupano di inviare i comandi di configurazione, fornendo all'utente un modo semplice e veloce per gestire l'interfaccia.

6.2.1 sendStandardMsg.vi



Questo elemento permette all'utente di generare semplici pattern di comunicazioni con impostazioni standard:

- distanza tra i comandi (tic) uguale per tutti i comandi;
- un solo evento di commutazione delle linee digitali, prima o dopo l'avvio della comunicazione.

La caratteristica principale di questo elemento è la semplicità di utilizzo. Per inviare un semplice pattern con caratteristiche standard è sufficiente definire un array bidimensionale (bool SI o HEX SI) contenente i messaggi da inviare e indicare la porta COM (VISA resource name) sulla quale inviare i comandi di configurazione. È comunque possibile personalizzare il pattern attraverso ingressi opzionali.
I comandi possono essere definiti tramite un array di elementi bool se codificati in binario (BIN), oppure string se codificati in esadecimale (HEX, solo comandi multipli di 4-bit). Nel caso di codifica HEX ogni stringa dell'array rappresenta un comando. Per la codifica

binaria invece, si utilizza un array bidimensionale bool, dove ogni riga rappresenta un comando.
In entrambi i casi l'ordine di invio dei comandi nel pattern rispetta l'ordine in cui sono definiti e il primo comando definisce la lunghezza di tutti i comandi del pattern. Non ci sono limitazioni sul numero di bit dei comandi (se maggiore di 64-bit, solo multipli di 8-bit)
Le commutazioni delle linee digitali sono opzionali e sono gestite attraverso un array bool (Digital_line_value) che definisce il valore della linea, 1 o 0. La posizione dell'elemento nell'array definisce a quale segnale è associato (tab. 6.1). Nel caso di commutazioni SCI le linee Digital_1 e Digital_2 devono comunque essere definite anche se poi verranno ignorate, in quanto utilizzate come segnale di SYNC dal protocollo.

0	1	2	3	4
Digital-1	Digital-2	Digital-3	Digital-4	Digital-5

Tabella 6.1:

I comandi SPI ricevuti durante il pattern (MISO) sono disponibili, a fine pattern, negli array boolSO e HEX SO.
Esistono due modalità per inviare i comandi: standard e personalizzata.

Standard

Per inviare un comando in modalità standard è sufficiente definire uno dei due array di ingresso HEX SI o bool SI, e opzionalmente definire i valori delle commutazioni digitali. Si ottiene quindi un comando con le seguenti caratteristiche:

- comandi SPI;
- tic=31μs;
- $f_{CLK} = 1MHz$;
- Interfaccia1 (sez. 4.9);
- CS attivo basso;
- CPOL=0, CPHA=1, CLK inattivo basso, scrittura sui fronti di salita e lettura su fronti di discesa di CLK;
- $tid = tnf = \frac{1}{f_{CLK}} \times 2$;
- commutazione delle linee digitali alla fine del pattern;
- distanza tra la fine dell'ultimo comando e le commutazioni digitali 10μs.

Personalizzazione

È possibile personalizzare i comandi e il pattern attraverso due cluster. SPI_config permette la personalizzazione dei comandi attraverso i paramteri di ingresso:

- frequency (4-2048): $f_{CLK} = \frac{40MHz}{(4-2048)}$;
- Interface selection:
  - 1: interfaccia1;

  - 2: interfaccia2.

- CS polarity:
  - 0: attivo basso;
  - 1: attivo alto;
  - 2: sempre alto;
  - 3: sempre basso.
- CLK polarity:
  - 0: CPOL=0, CPHA=1, CLK inattivo basso, scrittura sui fronti di salita e lettura sui fronti di discesa di CLK;
  - 1: CPOL=0, CPHA=1, CLK inattivo basso, scrittura sui fronti di salita e lettura sui fronti di discesa di CLK;
  - 2: CPOL=1, CPHA=0, CLK inattivo alto, scrittura sui fronti di salita e lettura sui fronti di discesa di CLK;
  - 3: CPOL=1, CPHA=1, CLK inattivo alto, scrittura sui fronti di discesa e lettura sui fronti di salita di CLK.
- TID resolution:
  - 0: $tq = \frac{1}{f_{CLK}}$;
  - 1: $tq = 25ns$.
- TID number of cycles (TID):
  - se TID resolution=0: $tid = tnf = \frac{1}{f_{CLK}} \times TID$;
  - se TID resolution=1: $tid = tnf = 25ns \times TID$.
- Communication protocol:
  - 0: SPI;
  - 1: SCI.

Pattern_config permette di personalizzare il pattern modificando i parametri:

- TIC(μs)>31: definisce la distanza tra i comandi del pattern;
- Istante comm.digitali:
  - 0: le commutazioni definite in Digital_line_value avvengono a fine pattern;
  - 1: le commutazioni definite in Digital_line_value avvengono a inizio pattern.
- delay(μs)>10:
  - se Istante comm.digitali=0: la distanza tra fine dell'ultimo comando e commutazione linee digitali;
  - se Istante comm.digitali=1: la distanza tra commutazione linee digitali e inizio del primo comando.

  tabella in out

Il cuore di sendStandardMsg.vi (Figura 6.2) è standardCommTableGen.vi. Questo elemento si occupa di generare una tabella rispettando il formato di (sez. 5.1), a partire dai dati di ingresso forniti dall'utente.
La tabella è poi data in ingresso all'elemento di controllo e generazione dei comandi di

configurazione (patternFromTableGenerator.vi) che sono inviati al pattern con gli appositi elementi di invio dei comandi di configurazione.

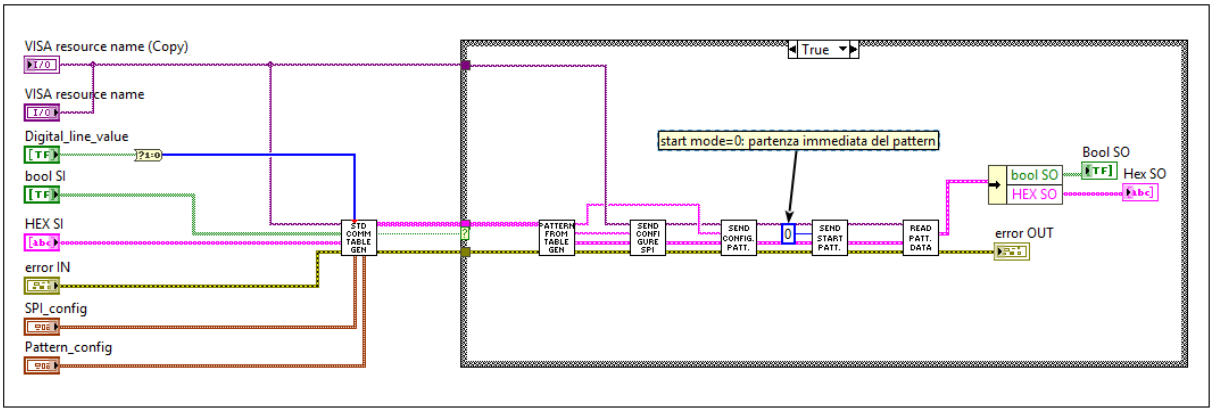


Figura 6.2: sendStandardMsg.vi.

6.2.2 main.vi

Le regole per definire correttamente il pattern (sez. 5.2.1) sono abbastanza complesse e richiedono una buona conoscenza del struttura della firmware.
Per questo, è stata creata un'apposita GUI (main.vi) che aiuta l'utente a gestire la definizione del pattern fornendo informazioni sulle lunghezze dei tempi che caratterizzano i comandi e su eventuali errori commessi (Figura 6.3).
Attraverso un apposito calcolatore si può individuare l'istante temporale esatto in cui definire l'invio di un comando per ottenere la distanze tic desiderata.
Anche in questo caso, la tabella generata è data in ingresso all'elemento di controllo e generazione dei comandi di configurazione (patternFromTableGenerator.vi). L'invio dei comandi non è però immediato come in sendStandardMsg.vi, ma è gestito dall'utente attraverso semplici pulsanti.
In questo modo l'interfaccia può essere usata dall'utente con il solo scopo di generare correttamente un pattern senza doverlo inviare[4]. Il pattern può essere salvato in appositi file.txt per essere riutilizzato successivamente, anche in altri porgetti. Saranno messi a disposizione dell'utente dei modelli standard di pattern che potranno essere modificati, controllati ed eventualmente salvati.
L'interfaccia grafica di main.vi è quella in Figura 6.3:

- VISA resurce name: permette di selezionare la porta COM del dispositivo con il quale si intende comunicare (sez. 4.9);
- file path: permette di importare una tabella di configurazione pre-compilata, per esempio con fogli di calcolo (Excel)[5], che sarà poi visualizzata nella tabella pattern. È assolutamente necessario che la tabella rispetti il formato visto in sez. 5.1 con l'esatto posizionamento dei valori nelle rispettive righe e colonne, e il mantenimento degli spazi vuoti.

[4]È consigliato utilizzare l'interfaccia main.vi solo per la fase di definizione della tabella. Per l'invio dei comandi è consigliato utilizzare le apposite funzioni di comunicazione con l'interfaccia (sez. 6.3).
[5]In Excel è necessario salvare nel formato: "Testo (delimitato da tabulazione)".

- pattern: permette di compilare la tabella, anche in questo caso nel rispetto delle regole (sez. 5.2.1). Può essere generata una tabella partendo dal foglio vuoto oppure è possibile modificare il file importato attraverso file path[6].A ogni modifica la tabella viene controllata e eventuali violazioni delle condizioni (sez. 5.2.1) o errori di compilazione sono descritti nell'apposito box (error Pattern);

- Save pattern: permette di salvare il pattern in modo da poter essere riutilizzato o nuovamente importato per ulteriori modifiche;

- CMD proprieties: contiene informazioni utili all'utente per agevolare la corretta definizione del pattern (sez. 5.2.1)):

  - Pattern: indica il tipo di pattern configurato in tabella:
    - 0: Fast mode;
    - 1: Normal mode;
    - 2: Long mode;
    - 3: Loop mode.
  - SPI_limit;
  - ticLimit: indica la minima distanza tra fine di un comando e inizio del successivo;
  - min distance SPI: indica la minima distanza tra l'inzio di un comando e l'inzio del successivo;
  - CMDlenght(μs): indica la lunghezza totale del periodo in cui CS è attivo, CMDlenght=tid+tcmd+tid;
  - tcmd(μs): indica la lunghezza del periodo in cui CLK è attivo;
  - tid(μs): indica la distanza tra la commutazione di attivazione CS e l'inizio della trasmissione dei dati, che è uguale alla distanza tra la fine della trasmissione dei dati e la commutazione di disattivazione CS.

- tic desiderato: permette di calcolare l'istante temporale (visualizzato in nexSPI-StartTime) in cui deve essere definito un comando, garantendo l'esatta distanza tic dall'ultimo comando definito;

- ConfigureSPI: permette, se il pattern è corretto, di inviare il comando di configurazione SPI (sez. 4.8.1) al dispositivo selezionato attraverso VISA resurce name.

- StartPattern: permette, se il pattern è corretto, di inviare il comandi di configurazione Pattern (sez. 4.8.2), seguito dal comando di start (sez. 4.8.3), seguito dal comando di lettura dei dati ricevuti dall'interfaccia durante il pattern (MOSI);

- startMode: permette di definire quando deve essere avviato il pattern:

  - 0: immediatamente, la procedura di avvio è subito avviata alla ricezione del comando,
  - 1: in attesa di un fronte di salita del segnale di trigger-in;

[6]Si consiglia di importare e modificare modelli standard.

  - 2: in attesa di un fronte di discesa del segnale di trigger-in;
  - 3: in attesa di un fronte di salita o discesa del segnale di trigger-in.

  Il segnale di trigger-in deve essere collegato al pin di ingresso P0.8 (sez. 4.9).

- readData: permette di visualizzare i dati ricevuti dall'interfaccia durante il pattern (MOSI) in formato binario ed esadecimale. L'ordine dei comandi negli array è concorde all'ordine di invio dei comandi;

- PWM and Static_digital_Line: sono implementati due cluster, Static_Digital_Line_Values e PWM_configurator che permettono di impostare il valore statico delle linee e di configurare i segnali PWM. Il loro utilizzo è descritto in dettaglio rispettivamente in sez. 6.3.4 e sez. 6.3.7.

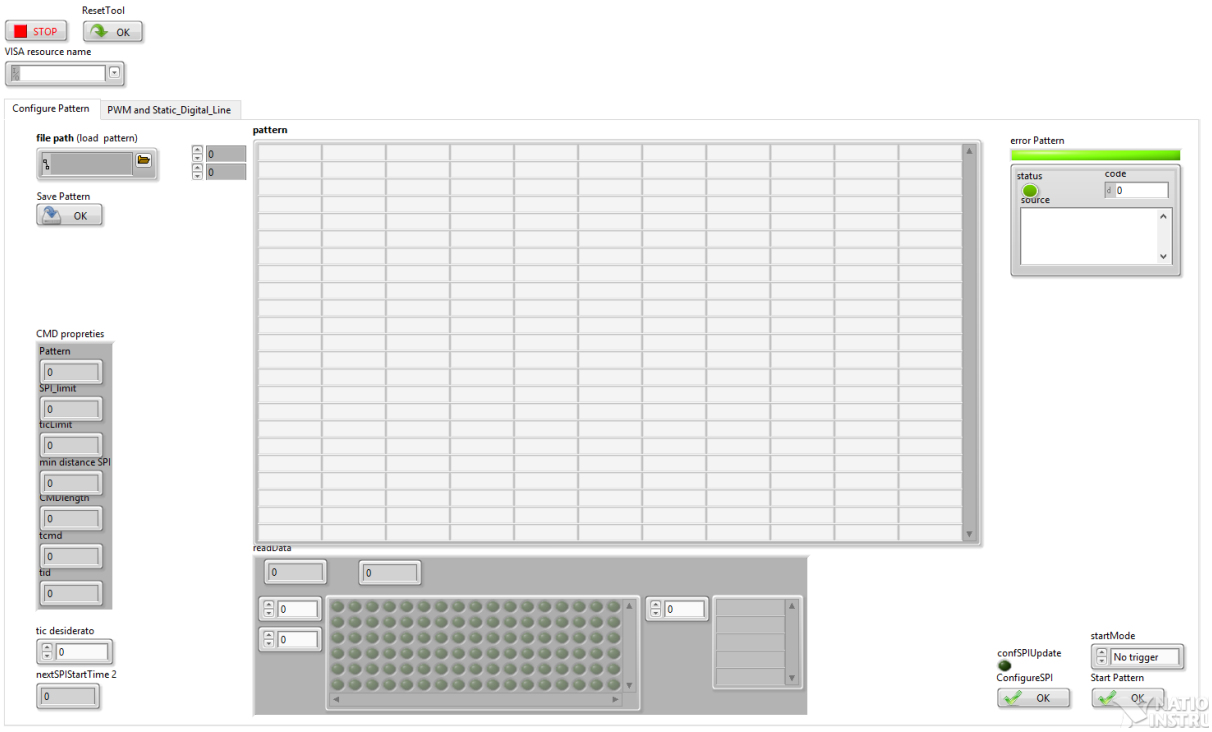
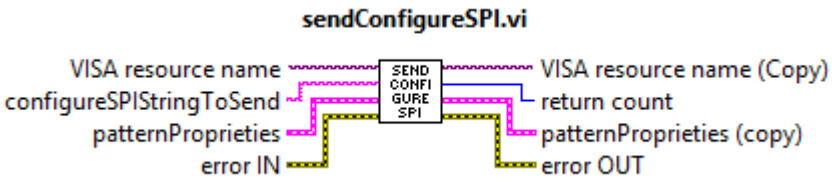


Figura 6.3: main.VI.

6.3 Comunicazione con l'interfaccia

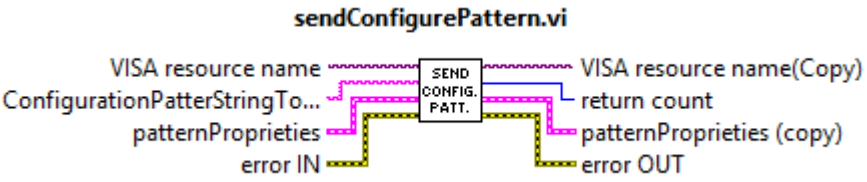
Le comunicazioni con l'interfaccia avvengono attraverso appositi file.vi, che permetto di comunicare attraverso comandi seriali, seguendo il protocollo descritto in sez. 4.8. Tutti questi elementi richiedono in ingresso VISA resource name, che definisce su quale porta COM è connessa l'interfaccia.

6.3.1 sendConfigureSPI.vi



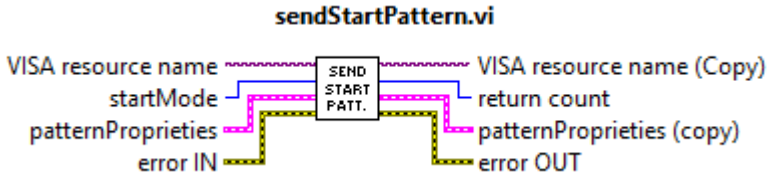
Si occupa di inviare il comando per la configurazione dell'interfaccia SPI/SCI, seguendo il formato descritto in (sez. 4.8.1)
Il parametro in ingresso configureSPIStringToSend è una stringa in uscita dall'elemento di controllo dei dati e generazione dei comandi di configurazione (patternFromTableGenerator.vi).
Anche il parametro di ingresso patternProprieties è in uscita dall'elemento patternFromTableGenerator.vi, ed è necessario per indicare il tipo di pattern inviato.
Per generare un pattern è fondamentale che il comando di configurazione SPI sia inviato prima del comando di configurazione pattern.

6.3.2 sendConfigurePattern.vi



Si occupa di inviare il comando per la configurazione del pattern seguendo il formato descritto in (sez. 4.8.2)
Il parametro in ingresso configurationPatternStringToSend è una stringa in uscita dall'elemento di controllo dei dati e generazione dei comandi di configurazione (patternFromTableGenerator.vi).
Anche il parametro di ingresso patternProprieties è in uscita dall'elemento patternFromTableGenerator.vi, ed è necessario per indicare il tipo di pattern inviato.
Per generare un pattern è fondamentale che il comando di configurazione pattern sia inviato dopo il comando di configurazione SPI.

6.3.3 sendStartPattern.vi

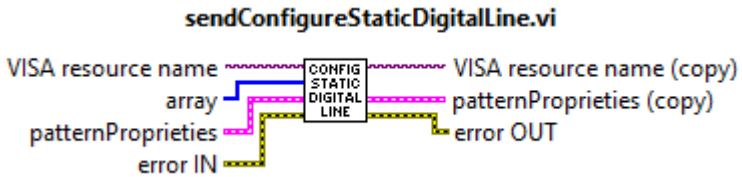


Si occupa di generare e inviare il comando di start del pattern, seguendo il formato descritto in (sez. 4.8.3).
L'ingresso startMode è un intero fornito dall'utente che deve esser compreso tra 0-3 e permette di decidere quando deve essere inviato il pattern:

- 0: immediatamente, la procedura di avvio è subito avviata alla ricezione del comando;
- 1: in attesa di un fronte di salita del segnale di trigger-in;
- 2: in attesa di un fronte di discesa del segnale di trigger-in;
- 3: in attesa di un fronte di salita o discesa del segnale di trigger-in.

Il segnale di trigger-in deve essere collegato al pin di ingresso P0.8 (sez. 4.9).

6.3.4 sendConfigureStaticDigitalLine.vi



Si occupa di generare e inviare il comando per la configurazione statica del valore logico delle linee digitali e CS del pattern, seguendo il formato descritto in (sez. 4.8.4)
Prima o dopo l'invio di un pattern è necessario poter impostare staticamente le linee a un valore logico definito dall'utente. Il parametro in ingresso array definito dall'utente, è un array di interi con sette elementi, ognuno dei quali associato a uno specifico segnale del pattern. La posizione dell'elemento nell'array definisce a quale segnale è associato (tab. 6.2) .

0	1	2	3	4	5	6
CS-1	CS-2	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5

Tabella 6.2: Elementi array in ingresso a sendConfigureStaticDigitalLine.vi.

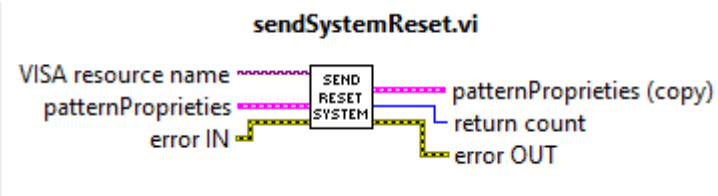
Ogni elemento deve assumere una valore compreso tra 0-2, a seconda del livello che si vuole impostare su ogni linea:

- 0: livello logico 0;
- 1: livello logico 1;

- 2: il livello logico rimane invariato.

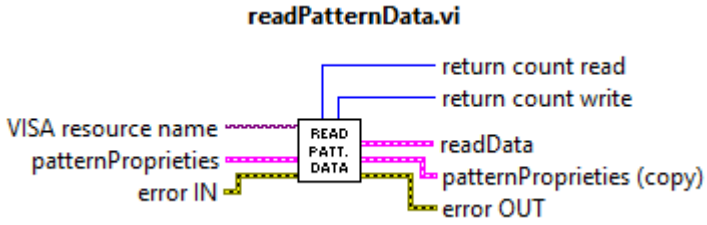
Ogni volta che viene inviato un comando di configurazione statica delle linee digitali è necessario specificare tutti e sette gli elementi del vettore.

6.3.5 sendSystemReset.vi



Si occupa di generare e inviare il comando di reset del microcotrollore seguendo il formato descritto in (sez. 4.8.5).

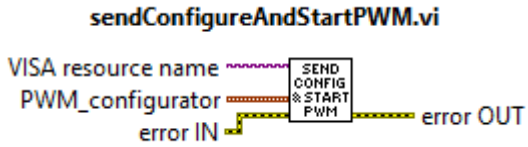
6.3.6 readPatternData.vi



Si occupa di generare e inviare il comando di lettura dei comandi ricevuti dall'interfaccia durante il pattern (MOSI) e di leggere la risposta dell'interfaccia (sez. 4.8.6). Questo elemento calcola automaticamente il numero di byte che saranno ricevuti in base alle proprietà del pattern (ingresso patterProprieties). Grazie all'elemento readDataString-ToNumeriConversion.vi, i dati ricevuti in formato seriale vengono suddivisi e inserti in appositi array con differenti codifiche. L'ordine degli elementi negli array è concorde con l'ordine d'invio dei comandi nel pattern; il primo elemento dell'array rappresenta la risposta al primo comando inviato ecc.
L'uscita readData di readPattern.vi è un clauster che contiene:

- bool SO: un array bool che contiene i comandi di risposta MOSI in formato bianrio;
- HEX SO: un array string che contiente i comandi di risposta MOSI in formato esadecimale. Ha senso solamente quando il numero di bit è multiplo di 4;
- numOfCMD: numero intero che rappresenta il numero totale di comandi di risposta MOSI ricevuti;
- numOfCMDbit: numero intero che rappresenta il numero di bit dei comandi.

6.3.7 sendConfigureAndStartPWM.vi

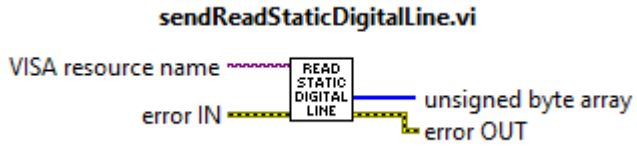


Si occupa di generare e inviare il comando per la configurazione e lo start/stop dei segnali PWM (pwm1 e pwm2), seguendo il formato descritto in (sez. 4.8.7)
L'ingresso PWM_configurator è un cluster fornito dall'utente che contiene i dati di configurazione delle PWM ed è così composto:

- Duty_pwm_1(%): valore intero (0-100) che rappresenta il duty cycle espresso in percentuale del segnale pwm1;
- freq_pwm_1 (kHz): valore intero (2-40.000) che rappresenta la frequenza espressa in kHz del segnale pwm1;
- Passive level pwm_1: valore intero (0-1) che rappresenta il valore del livello passivo della linea della pwm1;
- Action pwm_1: valore intero (0-2) che rappresenta l'azione da svolgere nei confronti del segnale pwm1:
  - 0: stop del segnale pwm1;
  - 1: start del segnale pwm1;
  - 2: nessuna operazione per pwm1.
- Duty_pwm_2(%): valore intero (0-100) che rappresenta il duty cycle espresso in percentuale del segnale pwm2;
- freq_pwm_2 (kHz): valore intero (2-40.000) che rappresenta la frequenza espressa in kHz del segnale pwm2;
- Passive level pwm_2: valore intero (0-1) che rappresenta il valore del livello passivo della linea della pwm2;
- Action pwm_2: valore intero (0-2) che rappresenta l'azione da svolgere nei confronti del segnale pwm2:
  - 0: stop del segnale pwm2;
  - 1: start del segnale pwm2;
  - 2: nessuna operazione per pwm2.

Ogni volta che viene inviato un comando di configurazione è necessario specificare tutti i parametri del cluster.

6.3.8 sendReadStaticDigitalLine.vi



Si occupa di generare e inviare il comando di lettura del valore delle linee digitali e di leggere la risposta dell'interfaccia, seguendo il formato descritto in (sez. 4.8.8).
L'uscita unsigned byte array è un array che rappresenta il valore logico attuale delle linee di commutazione digitale attraverso valori numerici 0-1. La posizione dell'elemento nell'array definisce a quale segnale è associato (tab. 6.3).

0	1	2	3	4
Digital-1	Digital-2	Digital-3	Digital-4	Digital-5

Tabella 6.3:

6.3.9 Comunicazione VISA

La comunicazione tra il driver e l'interfaccia avviene grazie al driver di comunicazione VISA e alle funzioni messe a disposizione dell'utente (Figura 6.4). Entrambi gli elementi VisaSendMsg.vi e VisaSend_ReciveMsg utilizzano le funzioni offerte da VISA per aprire e chiudere una classica connessione seriale con pacchetti a 8-bit e baudrate 9600.

- VisaSendMsg.vi: permette di inviare comandi seriali definiti in una stringa di ingresso: write buffer. È utilizzato per inviare comandi che non prevedono comunicazioni di risposta;
- VisaSend_ReciveMsg: permette di inviare comandi seriali definiti nella stringa di ingresso write buffer, ma una volta inviato l'intero comando si mette in attesa di ricevere un numero di byte definito dal parametro di ingresso nOfByteToRead, che potranno essere letti nella stringa di uscita read buffer. È utilizzato per inviare comandi che prevedono comunicazioni di risposta.

Impostando la lunghezza dei pacchetti a 8-bit, ogni carattere della stringa in ingresso al write buffer rappresenta un byte ed è codificado secondo lo standard ASCII. Il microcontrollore in lettura riceve il byte che viene però decodificato con la classica codifica binaria. Per inviare un valore intero, quindi, non è sufficiente convertire il valore numerico in stringa, ma è necessario inviare il carattere corrispondente alla codifica binaria del numero.
Ad esempio, se nel write buffer è presente il carattere 1, sarà inviato sulla porta di comunicazione il byte (00110001). In ricezione questo valore viene inserito in una variabile unsigned e indica il valore intero 49.
L'elemento uintToASCIIString.vi si occupa di tradurre i numeri interi nelle sequenze di caratteri che rappresentano il numero in codifica binaria. Questo elemento permette di

codificare interi di 8-bit, 16-bit e 32-bit. Un elemento di 16-bit, per esempio, sarà identificato da 2 caratteri (8-bit) contigui, che in lettura saranno decodificati dal micontrollore in un'unica variabile a 16-bit.

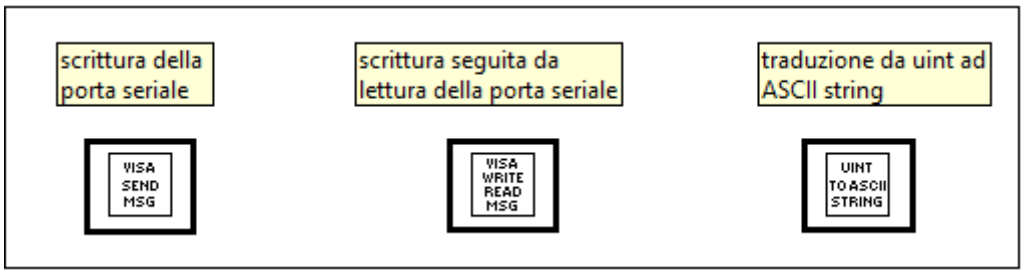
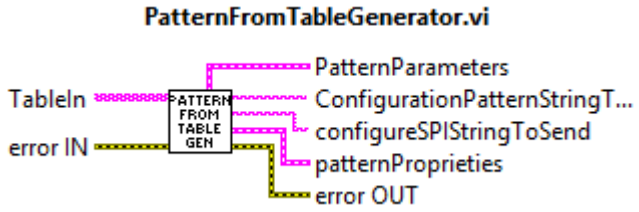


Figura 6.4: VI per la comunicazione VISA.

6.4 Controllo dei dati e generazione dei comandi di configurazione

Il cuore del driver è rappresentato da patternFromTableGenerator.vi che si occupa di controllare e tradurre la tabella di definizione del pattern nei comandi di configurazione e gestione del pattern.

6.4.1 patternFromTableGenerator.vi



In ingresso è richiesta solamente la tabella di definizione pattern, che deve essere di tipo string e deve rispettare rigorosamente il formato visto in sez. 5.1. La tabella viene controllata, assicurandosi che siano rispettate le regole definite in sez. 5.2.1. Se tutto è corretto viene tradotta in due stringe di configurazione del pattern e configurazione SPI.

- La stringa di uscita configureSPIStringToSend rispetta il formato definito in (sez. 4.8.1) e contiene i dati di configurazione dei comandi SPI/SCI.

- La stringa di uscita configurationPatternStringToSend rispetta il formato definito in (sez. 4.8.2) e contiene i dati di configurazione del pattern, ovvero i valori dei registri period e compare che definiscono le commutazioni digitali di CS e linee digitali, i comandi da inviare durante il pattern ed eventualmente i parametri di gestione dei comandi in loop.

Questo elemento è il più importante del driver, e sicuramente anche il più complesso. La sua struttura è mostrata in Figura 6.5 e Figura 6.6.[7]

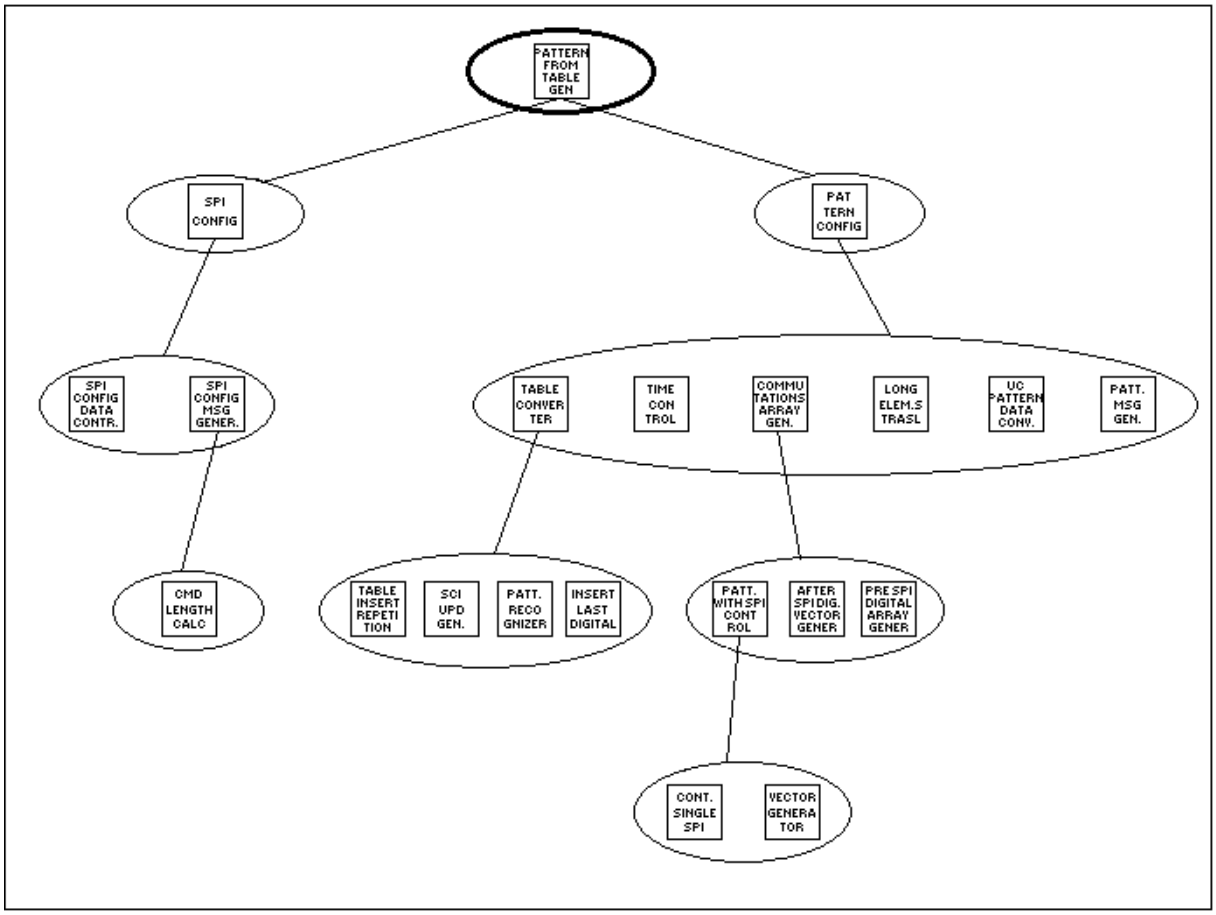
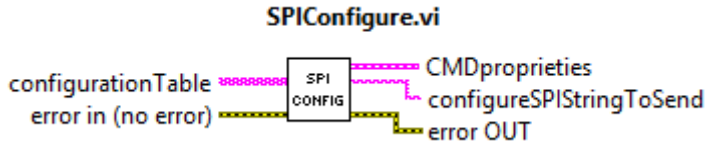
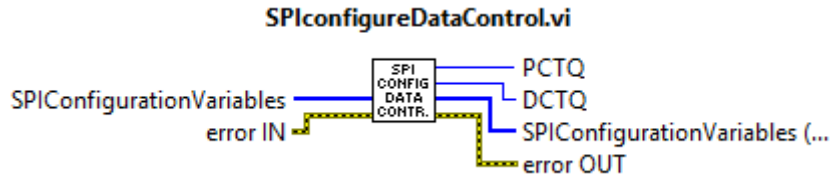


Figura 6.5: Mappa concettuale di patternFromTableGenerator.vi.

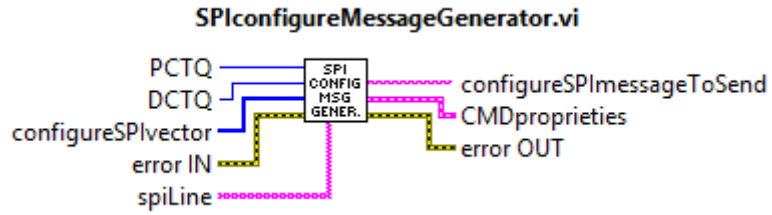


SPIconfigure.vi: a partire dalla tabella di ingresso controlla i dati e genera il messaggio di configurazione dell'interfaccia SPI/SCI da inviare al microcontrollore, e un cluster contenente i parametri dei CMD del pattern.

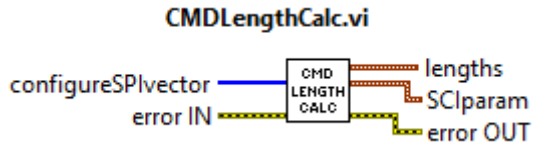
[7]Sono riportati solamente i VI più importanti.



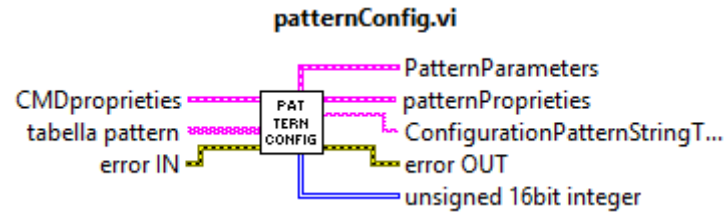
SPIconfigureDataControl.vi: controlla che i dati inseriti dall'utente per la configurazione dell'interfaccia SPI/SCI siano corretti.



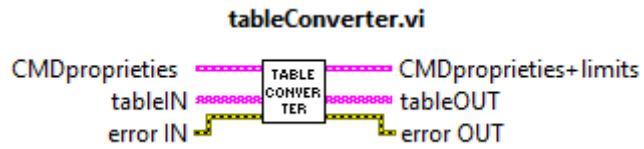
SPIconfigureMessageGenerator.vi: genera la stringa contenente i dati di configurazione dell'interfaccia SPI/SCI seguendo il formato di sez. 4.8.1



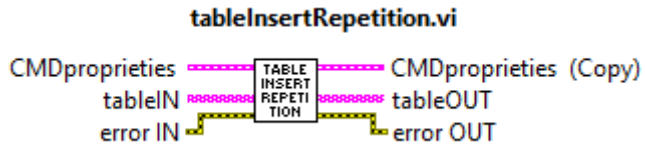
CMDLenghtCalc.vi: calcola le lunghezze che caratterizzano i comandi SPI e SCI: tcmd, tid, CMDlength=tcmd+2*tid e gli istanti di commutazione della linea SYNC (SCI) e l'inizio del comando.



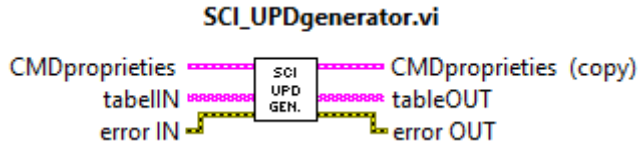
patternConfig.vi: a partire dalla tabella di ingresso, controlla i dati e genera il messaggio di configurazione Pattern da inviare al microcontrollore e un cluster contenente i parametri pattern (Figura. 6.6).



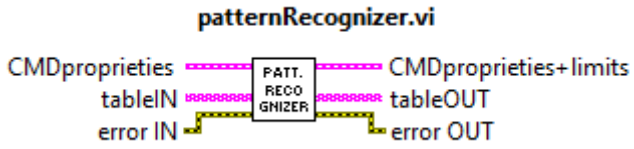
tableConverter.vi: converte la tabella di ingresso generata dall'utente in una tabella con formato adatto a essere controllato e trasformato dai successivi VI.



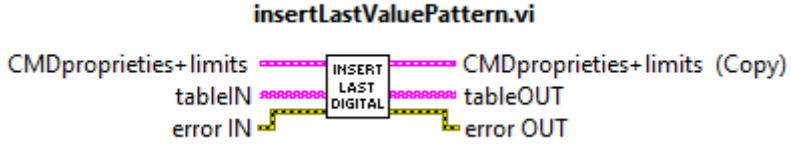
tableInsertRepetition.vi: a partire dalle informazioni contenute nelle colonne REPEAT e TIC, genera nuove righe nella tabella copiando il valore del relativo CMD in corrispondenza degli istanti che soddisfano la distanza tic definita da TIC (sez.. 5.2.1).



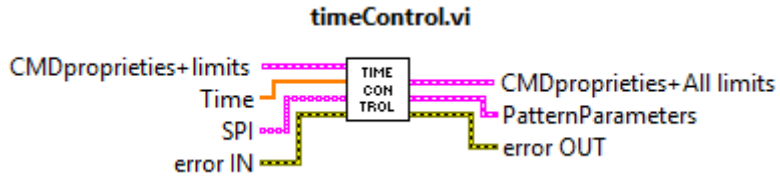
SCI_SYNCgenerator.vi: se il protocollo di comunicazione è SCI, inserisce nuove righe nella tabella con commutazioni digitali che rappresentano il segnale SYNC. Nel caso dell'interfaccia1, SYNC è rappresentato dalla linea digitale Digital_1, mentre per l'interfaccia2 abbiamo Digital_2.



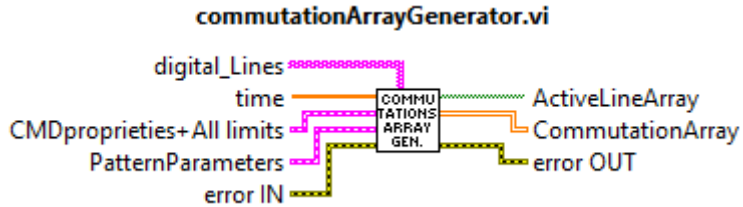
patternRecognizer.vi: in base al protocollo usato, al numero di bit dei comandi e al numero di linee digitali utilizzate riconosce la modalità di generazione del pattern e imposta il valore dei limiti temporali (ticLimit, SPI_limit ecc.).



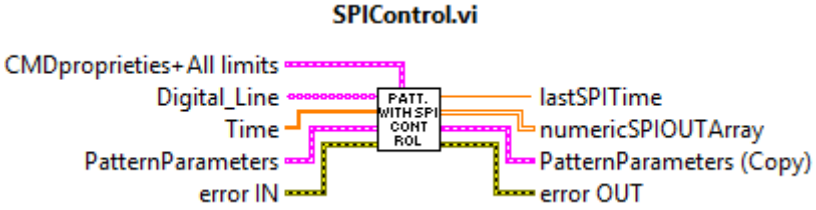
insertLastValuePattern.vi: inserisce al termine della tabella un'ultima riga per garantire che il valore dell'ultimo periodo (registri period e compare) sia sufficientemente grande per dare la possibilità al microcontrollore di arrestare il pattern correttamente.



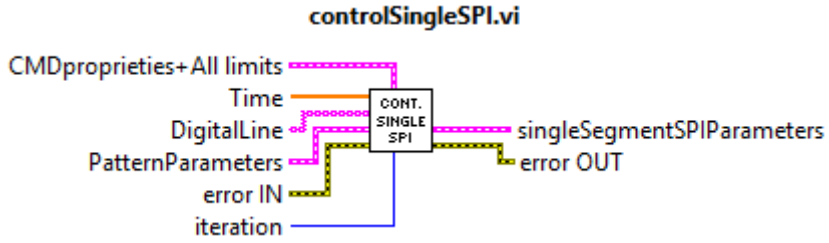
timeControl.vi: controlla che il pattern rispetti le varie regole e limitazioni temporali (SPI_limit e ticLimit). Vengono quindi controllate tutte le distanze fra gli elementi per assicurarsi il rispetto delle regole viste in sez. 5.2.1.



commutationArrayGenerator.vi: controlla che il pattern rispetti le varie regole e limitazioni temporali (SPI_limit e ticLimit). Controlla che il tipo e il numero di commutazioni delle linee digitali rispettino i vincoli visti in sez. 5.2.1 e trasforma i dati contenuti in tabella in un array bidimensionale numerico che rappresenta gli istanti temporali di commutazione delle linee digitali e dei CS.



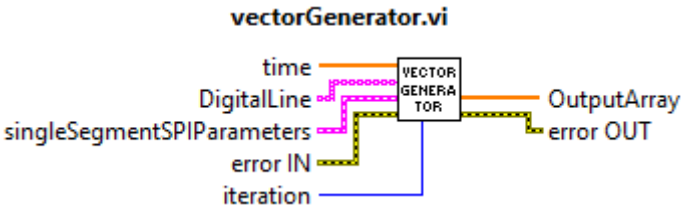
SPIcontrol.vi: controlla che il numero e il tipo di commutazioni delle linee digitali che avvengono all'interno dell'intera comunicazione SPI/SCI rispettino le regole definite in sez. 5.2.1. Se corretti, trasforma i dati contenuti in tabella in un array bidimensionale numerico che rappresenta gli istanti temporali delle commutazioni digitali nel segmento di pattern contenente la comunicazione SPI/SCI.



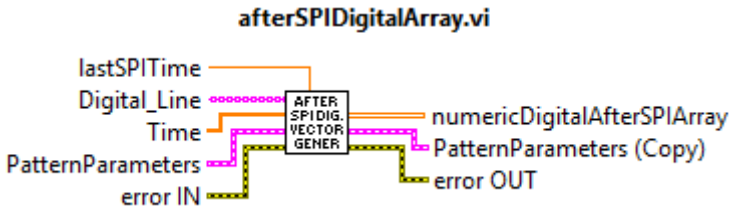
controlSingleSPI.vi: controlla che un singolo segmento di SPI (segmento temporale che va dalla fine di un comando alla fine del comando successivo) rispetti il numero e tipo di commutazione. Un segmento corretto può contenere le seguenti commutazioni (sez. 5.2.1):

- low to high seguito da high to low;
- high to low;
- low to hig.

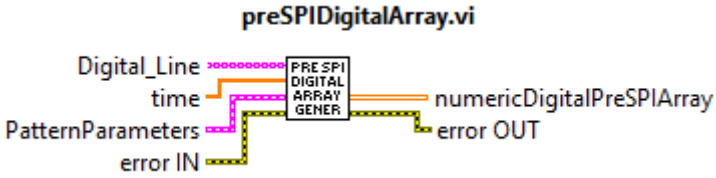
Genera in uscita un cluster contenente i parametri di un singolo segmento di SPI.



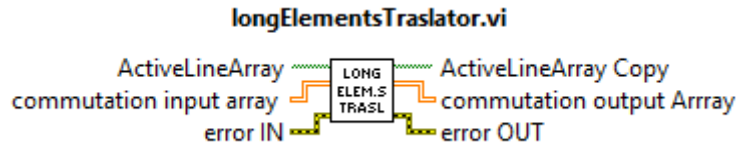
vectorGenerator.vi: trasforma il cluster contenente i parametri di un singolo segmento SPI in un array numerico monodimensionale che rappresenta gli istanti temporali di commutazione di un singolo segmento SPI. In pratica, è generato un array che contiene la durata del periodo (lunghezza del segmento) e gli istanti in cui avvengono le eventuali commutazioni di ogni linea digitale. Ogni linea può avere al massimo due commutazioni: una commutazione low to high e una commutazione high to low (sez. 5.2.1).



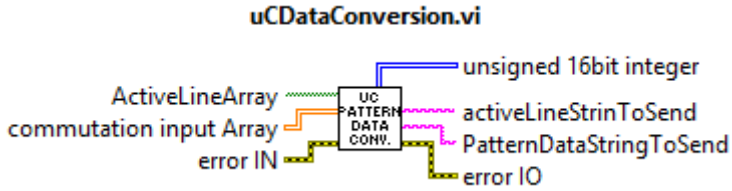
afterSPIDigitalArray.vi: controlla che le commutazioni delle linee digitali che avvengono dopo la comunicazione SPI rispettino le regole del pattern (sez. 5.2.1). Trasforma i dati contenuti in tabella in un array bidimensionale numerico che rappresenta gli istanti temporali di commutazione del segmento di pattern digitale dopo la comunicazione SPI.



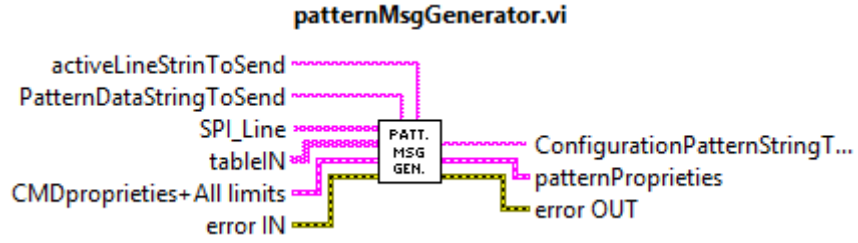
preSPIDigitalArray.vi: controlla che le commutazioni delle linee digitali che avvengono prima della comunicazione SPI rispettino le regole del pattern (sez. 5.2.1). Trasforma i dati contenuti in tabella in un array bidimensionale numerico che rappresenta gli istanti temporali di commutazione del segmento di pattern digitale dopo la comunicazione SPI.



longElementsTraslator.vi: per quanto visto in sez. 4.4.4, non possono essere gestiti segmenti temporali troppo grandi. Ovvero, se le distanze tra gli elementi (comandi SPI o commutazioni digitali) sono maggiori di 420μs è necessario suddivide questi elementi in una somma di elementi più piccoli, così da rispettare la massima lunghezza del periodo delle PWM.



ucDataConversion.vi: converte l'array numerico bidimensionale delle commutazioni generato da commutationArrayGenerator.vi in un array bidimensionale contenente i valori dei registri compare e period delle PWM.



patternMsgGenerator.vi: genera il comando string ConfigurePatternStringToSend rispettando il formato visto in sez. 4.8.2.

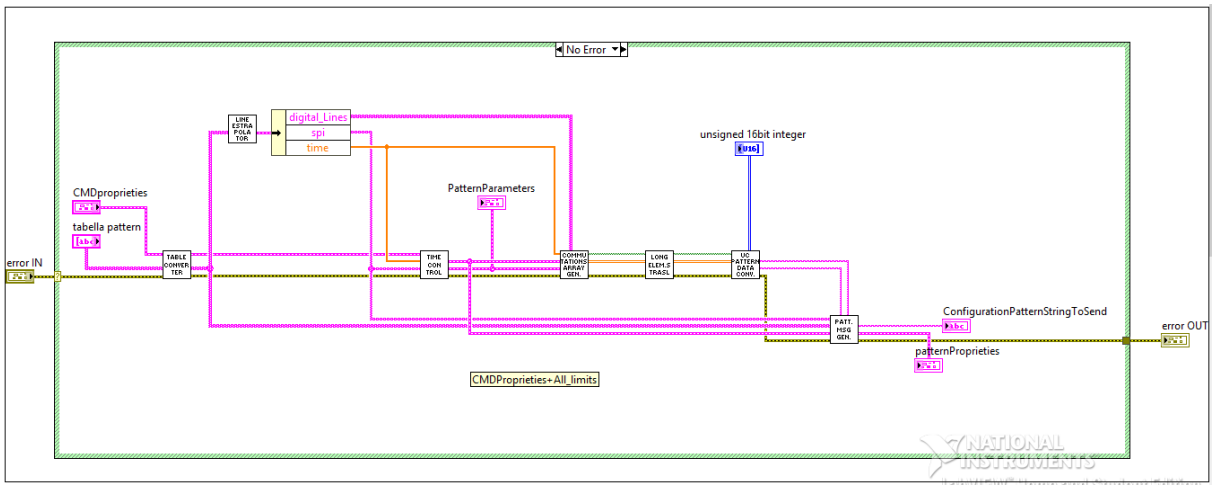


Figura 6.6: Struttura di patternConfig.vi.

Capitolo 7

Esempi di utilizzo dell'interfaccia

In questo capitolo verranno mostrati due esempi di utilizzo dell'interfaccia che ne mostrano alcune funzionalità. Il primo esempio sfrutta le proprietà di sincronizzazione del pattern, ovvero la possibilità di gestire la distanza tra commutazioni digitali e comunicazioni serali. Il secondo esempio, più complesso, sfrutta l'invio di comandi in loop con lettura del MISO e lo start del pattern associato a eventi esterni (trigger-in).

7.1 Active to Limp Home (TLE75008_EMD)

In sez. 1.1.2 abbiamo visto che in fase di caratterizzazione del chip TLE75008_EMD[1] è necessario misurare il tempo impiegato dal chip per cambiare modalità operativa in modo da fornire all'utente finale, tramite datasheet, informazioni certe sui tempi di lavoro del dispositivo come mostrato in Figura 1.1. Per esempio, per misurare $t_{ACTIVE2LH}$, ovvero il tempo necessario per passare da Active mode a Limp Home mode, bisogna eseguire sequenzialmente le seguenti operazioni:

- set IDLE pin: passare da Sleep mode ad Idle mode mediante commutazione low to high della linea digitale Idle;
- write Idle to Active: passare da Idle mode ad Active mode aggiornando l'opportuno registro con comunicazione SPI;
- set IN1 pin: attivare un'uscita mediante commutazione low to high della linea digitale IN1;
- reset IDLE pin: passare da Active mode a Limp Home mode mediante commutazione high to low della linea digitale Idle;
- read Standard Diagnosis: leggere mediante comunicazione SPI il registro Standard Diagnosis per verificare l'avvenuto cambiamento di modalità operativa.

Possiamo collegare IDLE pin all'uscita dell'interfaccia Digital-1, IN1 pin all'uscita Digital-2 e gli ingressi SPI (CS, CLK, MOSI, MIS0) all'interfaccia1 di comunicazione seriale.
In questo caso, il pattern deve essere così composto:

- set IDLE pin: set Digital-1;
- write Idle to Active: MOSI=8C80;
- set IN1 pin: set Digital-2;
- reset IDLE pin: rest Digital-1;

- read Standard Diagnosis: MOSI=0001.

Il cambio di modalità operativa è indicato dalla risposta del chip al comando di Standard Diagnosis (MOSI=0001). In particolare, la risposta MISO=8682 indica il passaggio da Active mode a Limp Home mode. In questo caso, per misurare $t_{ACTIVE2LH}$ è necessario poter variare la dista tra Reset IDLE pin e l'invio del comando di lettura Standard Diagnosis. Partendo da una misura inferiore al tempo stimato, è sufficiente ripetere la misura con piccole variazioni della distanza fino a quando il comando di risposta non è pari a 8682.
Il valore tipico di $t_{ACTIVE2LH}$ è 50μs, e quindi possiamo impostare 20μs come distanza iniziale tra reset di IDLE pin e l'invio del comando 0001.
Le distanze tra gli altri eventi non influenzano la misura e sono scelti arbitrariamente; inoltre, è stato aggiunta una commutazione di reset IN1 pin per preparare il chip a ripetere la misura con le stesse condizioni iniziali.
La frequenza di clock della comunicazione è 1MHz, i registri del chip sono a 16-bit, la polarità di CS e CLK è rispettivamente attivo basso e CPOL=0, CPHA=1. Gli altri parametri dei comandi SPI non influenzano la misura. Abbiamo quindi un pattern così composto tab. 7.1:

	0	1	2	3	4	5	6	7	8	9	10
0	User text										
1											
2	CSpol	0									
3	CLKpol	0									
4	TIDres	0									
5	PDIV	40									
6	TID	2									
7	CMDnBit	16									
8	CODMsg	1									
9	COMProt	0									
10	CSDelay	50									
11	SelSPI	1									
12											
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0				1	0					
15	30	8C80									
16	100					1					
17	300				0						
18	320	0001									
19	400					0					

Tabella 7.1: Pattern: Active to Limp Home.

Per inviare questo pattern è necessario utilizzare gli elementi di generazione dati (patternFromTableGenerator.vi) e di comunicazione con l'interfaccia, come mostrato in (Figura 7.1):

- il file.txt è convertito in una matrice di tipo string, ed è dato in ingresso a patternFromTableGenerator.vi, che si occupa di controllarlo e, nel caso sia corretto, di generare i comandi di configurazione SPI e configurazione del pattern;
- sendConfigureSPI.vi invia il comando di configurazione SPI[1];
- sendConfigurePattern.vi invia il comando di configurazione del pattern;
- sendStartPattern.vi con ingresso startMode=0 dà immediato avvio al pattern;
- readPatternData.vi permette di leggere i comandi ricevuti dall'interfaccia (MISO).

In questo modo il pattern è inviato solamente una volta, con distanza tra reset IDLE pin (r 18) e invio del comando MOSI=0001 pari a 20μs.

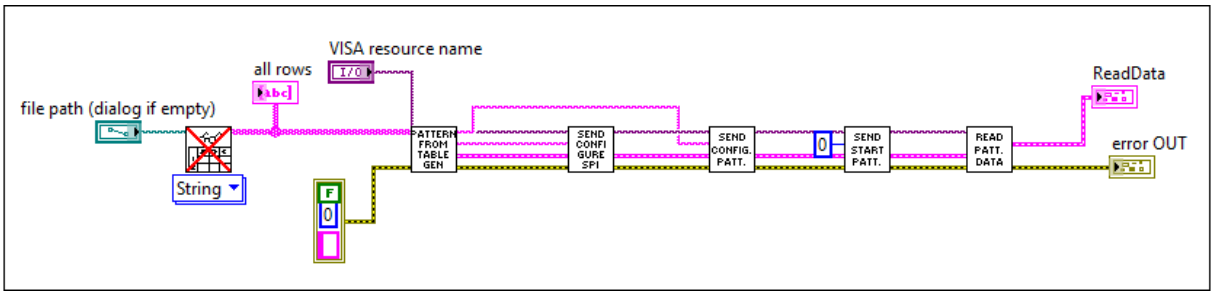


Figura 7.1: VI invio singolo pattern.

Per poter misurare $t_{ACTIVE2LH}$ è necessario variare dinamicamente la distanza. Per fare ciò possiamo agire come in figura: la struttura di invio pattern descritta in precedenza è inserita in un ciclo for di 10 ripetizioni (Figura 7.2). A ogni istanza del ciclo, l'elemento della tabella di ingresso di riga=18 e colonna=0 è incrementato di 1μs e viene inviato un nuovo pattern. I messaggi letti vengono confrontati con il comando di risposta MISO=8682 per dare all'utilizzatore un riscontro visivo della misura.

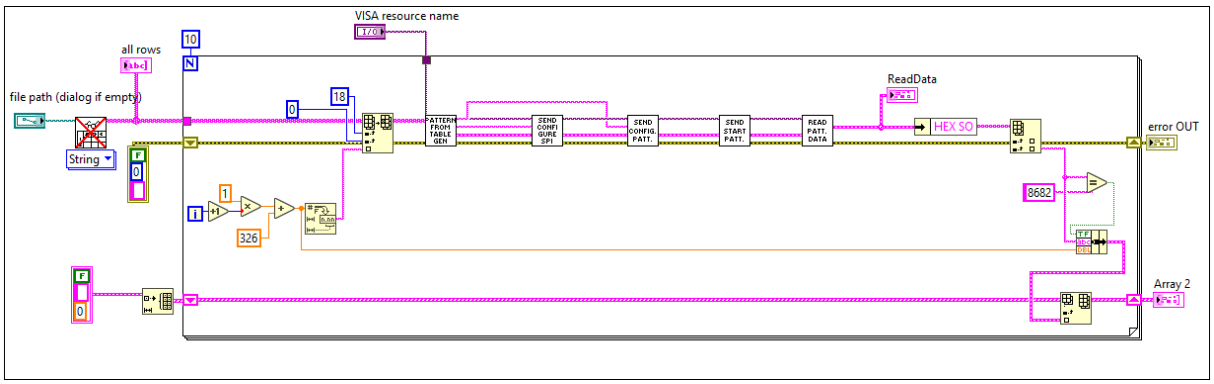
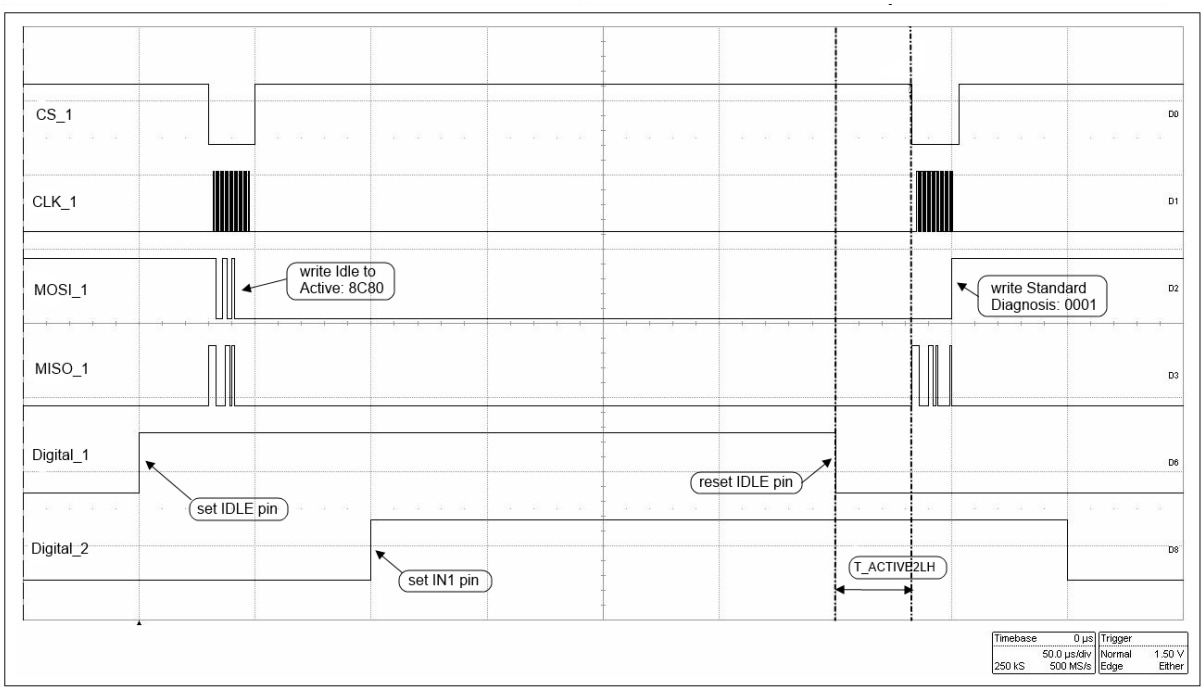


Figura 7.2: VI invio multiplo pattern.

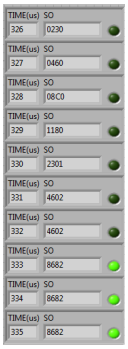
Così facendo, si ottengono i risultati di Figura 7.3(b), dove si nota che l'effettivo cambiamento di modalità operativa (MISO=8682) avviene quando il comando è inviato dopo

[1]Il comando di configurazione SPI deve essere inviato rigorosamente prima del comando di configurazione del pattern.

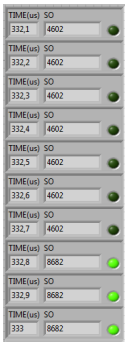
333μs, ovvero abbiamo una misura di $t_{ACTIVE2LH} = (333 - 300) = 33\mu s$. È interessante notare che, riducendo l'incremento a 0,100μs (Figura 7.3(c)), si possono ottenere misure con elevata risoluzione. Il limite di risoluzione raggiungibile dall'interfaccia è 7ns.
In (Figura 7.3(a)) sono mostrate le forme d'onda del pattern relative alla misura di $t_{ACTIVE2LH} = 33\mu s$, che sono state raccolte per mezzo di un oscilloscopio con canali di ingresso digitali con scala temporale di 50μs/div e trigger impostato sul primo fronte di salita della linea Digital_1.



(a) Forme d'onda $t_{ACTIVE2LH}$.



(b) Ris. 1μs.



(c) Ris. 100ns.

Figura 7.3: Misure di $t_{ACTIVE2LH}$.

7.2 TLD5541-1QV MSF topology

TLD5541-1QV è un H-Bridge DC/DC controllabile via comandi SPI, pensato per il pilotaggio di LED e per questo utilizzabile come regolatore di corrente o tensione costante.
In modalità regolatore di corrente possono essere pilotati più LED in serie. Se alcuni di questi LED vengono disconnessi, è possibile avere degli spike di corrente dovuti alle capacità di uscita che possono danneggiare i dispositivi. TLD5541-1QV prevede una routine di controllo che permette al dispositivo di passare in modalità di regolazione della tensione e scaricare le capacità di uscita. La regolazione della tensione è automatica partendo dalle informazioni inviate dall'utente sul numero di LED collegati prima e dopo la disconnessione. Quando la routine è finita, la tensione d'uscita raggiunge il corretto valore ed è abilitato un bit di flag (EOFM) presente nella Standard Diagonsis che segnala la fine di MSF. A partire da questo momento, il dispositivo rimane in modalità di regolazione della tensione per un tempo programmabile tprep per permettere la disconnessione dei LED.
Lo scopo di questo esempio è verificare il corretto funzionamento di questa routine.
L'idea è di pilotare inizialmente 10 LED, impostare e avviare la routine MSF e, una volta conclusa, all'interno del tempo tprep, disconnettere 4 LED e passare quindi a pilotare 6 LED.
Per fare ciò ci serviamo di una board sulla quale sono montati 32 LED, che sono pilotati attraverso 4 Low-Side Relay Switch (TLE7420) controllabili via SPI (16-bit) e connessi in modalità daisy chain (64-bit).
Sebbene ogni interfaccia di generazione del pattern disponga di due interfacce SPI, in questo caso abbiamo bisogno di due distinte interfacce di generazione del pattern: INTF1 e INTF2, connesse rispettivamente al dispositivo TLD5541-1QV e alla catena di relay switch. Il pin di ingresso di TLD5541-1QV per attivare l'uscita PWMI è connesso all'uscita Digital-1 di INTF1, mentre l'uscita Digital-2 di INTF1 è connessa al pin di trigger-in di INTF2.
La sequenza delle operazioni da svolgere è la seguente:

- INTF2: invio immediato del comando di abilitazione di 10 LED (tab. 7.2);
  - write MOSI_2=AEAAAAAAAAAFFFFF.
- INTF2: invio del comando di start in attesa di trigger-in (rising edge) del comando di abilitazione di 6 LED (tab. 7.3);
  - write (in attesa di tigger-in) MOSI_2=AEAAAAAAAAAAFFF.
- INTF1: invio immediato del seguente pattern al dispositivo TLD5541-1QV (tab. 7.4);
  1. write MOSI_1=7880[2] impostazione della modalità di utilizzo;
  2. write MOSI_1=8001: impostazione della modalità si utilizzo ;
  3. set Digital-1: set PWMI accensione dell'uscita;
  4. write MOSI_1=880A: impostazione del numero di LED collegati (10);
  5. write MOSI_1=8AFF: impostazione del numero di LED collegati dopo la disconnessione (6) e impostazione di tprep;
  6. write MOSI_1=8024: avvio della routine MSF;

[2]I comandi di configurazione e abilitazione della routine MSF sono puramente indicativi.

7. write loop MOSI_1=0001: invio in loop della Standard Diagnosis controllando il bit MSF. Il loop termina quando il nono bit della risposta MISO_1[9]=1. Al termine del loop è impostata la commutazione della linea Digital-2=1.

Riassumendo, tramite l'interfaccia INTF2 impostiamo la board con l'attivazione di 10 LED poi, sempre grazie a INTF2, viene inviato il comando di avvio condizionato al trigger-in (rising edge) del comando di attivazione di 6 LED. Il segnale di trigger-in (INTF2) è connesso al segnale Digital-2 di INTF1 che avrà una commutazione low to high (rising edge) solamente quando sarà letto il comando MISO_1[9]=1, ovvero quando è terminata la routine MSF. Tendendo conto dei vari ritardi dovuti alla lettura del MISO_1 e al tempo necessario per avviare un comando da trigger-in (30μs), è preferibile impostare tprep al massimo valore (300μs).

In questo caso le distanze tra i vari comandi sono arbitrarie, a eccezione del loop della Standard Diagnosis che deve iniziare immediatamente dopo l'avvio della routine MSF. La frequenza di clock di entrambe le comunicazioni è 2MHz, i comandi sono a 16-bit per INTF1 e 64-bit per INTF2, la polarità di CS e CLK è rispettivamente attivo basso e CPOL=0, CPHA=1. Gli altri parametri dei comandi SPI non influenzano la misura.

In Figura 7.4 è visualizzato il pattern di avvio MSF (CS_1, CLK_1, MOSI_1, MISO_1), il comando di abilitazione 6 LED[3](CS_1, CLK_1, MOSI_1, MISO_1) e i segnali caratteristici dell'H-Bridge DC/DC (SWN ,VFB, VOUT[4], IL).
Le forme d'onda sono state raccolte per mezzo di un oscilloscopio con canali di ingresso digitali con scala temporale di 200μs/div e trigger impostato sul primo fronte di discesa del CS_1.
Possiamo distinguere 4 fasi:

- fase 1: sono abilitati 10 LED e l'uscita PWMI dell'H-bridge è disabilitata (Digital_1=0). Sono inviati i due comandi di impostazione della modalità di utilizzo;

- fase 2: è abilitata l'uscita e dopo una fase di assestamento la tensione VOUT è costante. Sono inviati i comandi di impostazione e avvio MSF;

- fase 3: ha inizio la routine MSF con la diminuzione della tensione di uscita VOUT fino al valore finale (circa 6/10 dell'inziale). Ha inizio anche il loop di lettura della Standard Diagnosis;

- fase 4: raggiunto il valore finale di VOUT si attiva il bit EOMSF che segnala la fine di MSF. Il loop termina con la lettura di MISO=MISO_1[9]=1 e la linea Digital_2 commuta. Digital_2 è collegata all'ingresso trigger-in di INTF2, sulla quale è stato pre-caricato il comando di abilitazione di 6 LED. La commutazione di Digital-2=trigger-in dà quindi avvio al comando.

[3]Il comando di abilitazione è già stato inviato.
[4]Non è la tensione di uscita ai LED, ma una tensione di riferimento interna al chip.

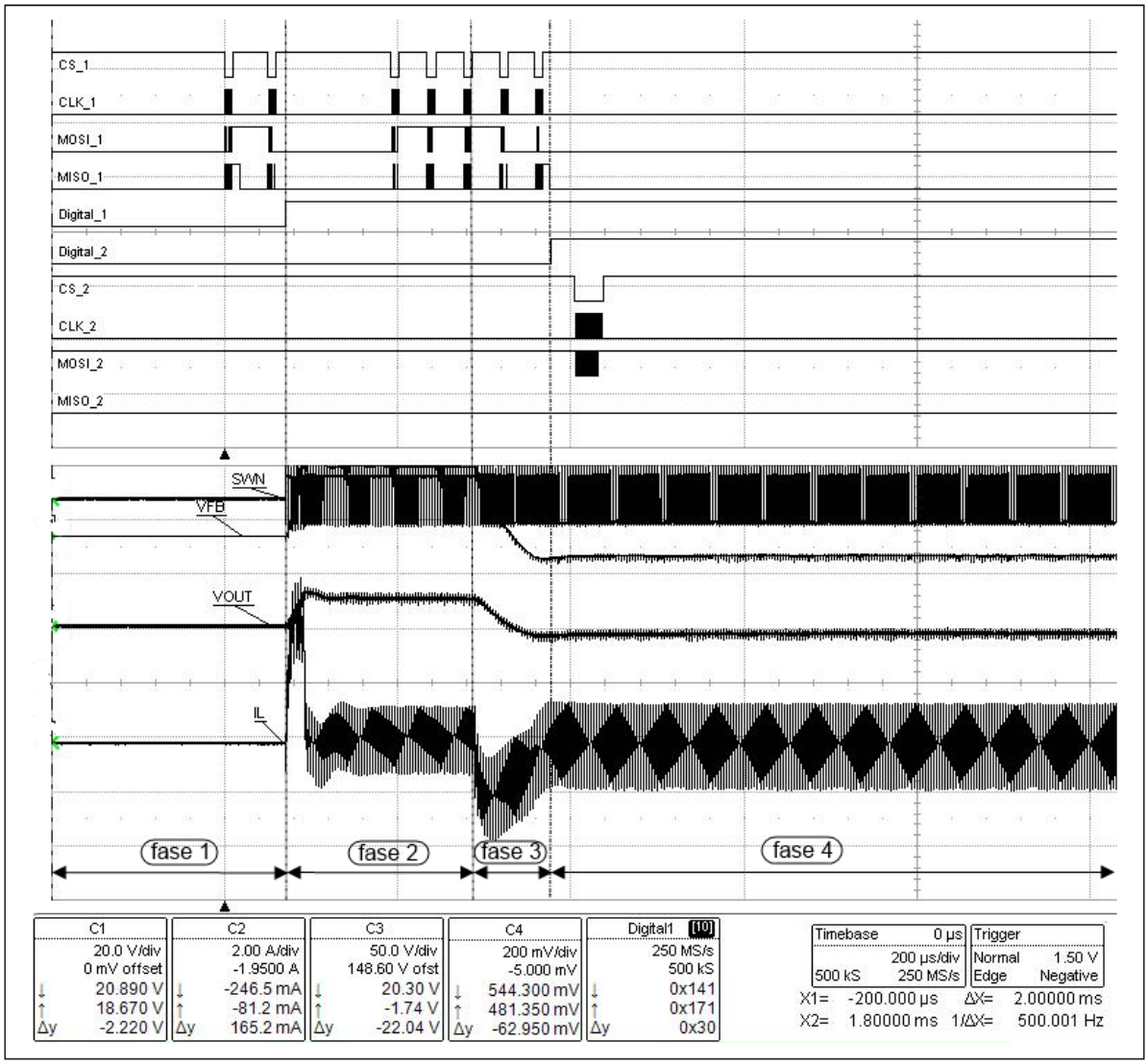


Figura 7.4: Forme d'onda routine MSF.

	0	1	2	3	4	5	6	7	8	9	10
0	User text										
1											
2	CSpol	0									
3	CLKpol	0									
4	TIDres	0									
5	PDIV	20									
6	TID	2									
7	CMDnBit	64									
8	CODMsg	1									
9	COMProt	0									
10	CSDelay	50									
11	SelSPI	1									
12											
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0	AEAAAAAAAAAFFFFF									

Tabella 7.2: Pattern: abilitazione di 10 LED (INTF2).

	0	1	2	3	4	5	6	7	8	9	10
0	User text										
1											
2	CSpol	0									
3	CLKpol	0									
4	TIDres	0									
5	PDIV	20									
6	TID	2									
7	CMDnBit	64									
8	CODMsg	1									
9	COMProt	0									
10	CSDelay	50									
11	SelSPI	1									
12											
13	TIME(μs)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0	AEAAAAAAAAAAFFF									

Tabella 7.3: Pattern: abilitazione di 6 LED (INTF2).

	0	1	2	3	4	5	6	7	8	9	10
0	User text										
1											
2	CSpol	0									
3	CLKpol	0									
4	TIDres	0									
5	PDIV	20									
6	TID	2									
7	CMDnBit	16									
8	CODMsg	1									
9	COMProt	0									
10	CSDelay	50									
11	SelSPI	1									
12											
13	TIME(us)	CMD	REPEAT	TIC	Digital-1	Digital-2	Digital-3	Digital-4	Digital-5	StopCMD	StopLV
14	0	7880			0	0					
15	50	8001									
16	70				1						
17	192	880A									
18	234	84FF									
19	276	8024									
20	318	0001	10	31						xxxxxx1xxxxxxxxx	x1xxx

Tabella 7.4: Pattern: MSF routine (INTF1).

Capitolo 8

Conclusioni

Per caratterizzare un dispositivo, occorre verificarne il funzionamento e ricavare informazioni sulle performance e sulle tempistiche di lavoro definite in fase di design. Per fare ciò, è necessario forzare e monitorare gli ingressi e le uscite del dispositivo nelle diverse condizioni di lavoro. A tale scopo, nel corso di questo progetto è stata creata un'interfaccia per la generazione di comunicazioni seriali programmabili e sincronizzabili con segnali digitali che possa essere integrata in un sistema di test.
In questo specifico caso, i dispositivi da testare permettono all'utente di interfacciarsi tramite pin digitali e comunicazioni seriali basate sui protocolli SPI e SCI. L'interfaccia, dunque, deve permettere la generazione di pattern digitali, ovvero un insieme di comunicazioni seriali temporizzate e sincronizzate con segnali digitali. È fondamentale poter personalizzare i parametri tipici di una comunicazione seriale (frequenza, numero di bit, ritardi) e le distanze tra gli elementi del pattern.
Nel nostro caso, la caratterizzazione avviene per mezzo di sistemi di test automatizzati che comprendono vari strumenti (oscilloscopi, multimetri, generatori di forme d'onda, ecc.) e che sono implementati con l'ambiente di sviluppo Labview. L'interfaccia deve quindi presentare un driver Labview che ne permetta l'integrazione nei sistemi di test, e una semplice e intuitiva GUI di gestione.

Dopo aver scelto l'hardware da utilizzare, cioè il microcontrollore XMC4700 (Infineon Technologies), è stato programmato il firmware. La personalizzazione delle comunicazioni seriali è stata ottenuta sfruttando le caratteristiche del modulo Universal Serial Interface Channel (USIC), presente nel microcontrollore. La sincronizzazione delle linee digitali, invece, è stata realizzata tramite l'interazione tra il modulo USIC, il modulo per la generazione di segnali PWM (CCU8) e gli eventi di interrupt.Per ottenere elevate prestazioni, si è cercato di minimizzare l'utilizzo della CPU, affidando nei limiti del possibile le operazioni da svolgere alle periferiche.Inoltre, sono state implementate funzioni di gestione e configurazione dell'interfaccia, che permettono la comunicazione con il driver attraverso un protocollo di scambio dati progettato appositamente.
La struttura principale del firmware è stata realizzata tramite appositi tool forniti dall'ambiente di sviluppo DAVE™, mentre nei casi in cui erano richieste performance elevate, è risultato necessario sfruttare al massimo le potenzialità dei moduli, modificando direttamente i registri di configurazione e di gestione. Per permettere all'utente di gestire i pattern da inviare, è stata creata una tabella di definizione dei dati avente un formato

standard, che associa valore e posizione di un elemento a determinate funzioni dell'interfaccia. Entrando maggiormente nel dettaglio, ogni riga della tabella rappresenta un istante temporale, di cui vengono definite nelle varie colonne le commutazioni digitali e le comunicazioni seriali. Inoltre, sono stati descritti i vincoli e le limitazioni che devono essere rispettati nella definizione della tabella.
Infine, è stato sviluppato il driver, il cui compito è integrare l'interfaccia nel sistema di test realizzato in Labview. Nello specifico, si occupa di controllare e tradurre le informazioni contenute nella tabella nei messaggi di configurazione e di gestione dell'interfaccia. Il driver gestisce anche la comunicazione di tali messaggi all'interfaccia attraverso il protocollo di scambio dati.

8.1 Risultati ottenuti

I requisiti di progetto (tab. 8.1), in particolare le richieste di personalizzazione delle comunicazioni seriali e di sincronizzazione con linee digitali, sono stati in buona parte soddisfatti. Lo strumento progettato dispone di due interfacce SPI/SCI programmabili che permettono di inviare comandi con numero di bit che va da 1-bit fino a un massimo di 420-bit ciascuno. Qualora il numero di bit sia maggiore di 64-bit, sono accettati solamente valori multipli di 8. É possibile anche inviare sequenze di comandi in loop che possono essere interrotte con la lettura di uno specifico comando definito dall'utente. La frequenza del segnale di sincronizzazione CLK può essere scelta tra 2048 diversi valori compresi tra 20 kHz e 10MHz, e i ritardi caratteristici delle comunicazioni SPI (tic, tfd) possono essere configurati dall'utente con risoluzione pari a 25ns. Anche le polarità dei segnali CS e CLK sono completamente programmabili. Per quanto riguarda CS, è possibile scegliere tra quattro soluzioni: attivo basso, attivo alto, sempre altro e sempre basso. Invece, per quanto riguarda CLK, è possibile scegliere il valore dello stato inattivo e i fronti di sincronizzazione per la lettura (MISO) e la scrittura (MOSI) dei dati.
Le comunicazioni seriali possono essere sincronizzate con cinque segnali digitali. In particolare, è possibile gestire le distanze tra commutazioni dei segnali digitali e comandi seriali, e la distanza tra la fine di un comando e l'inizio del comando successivo (tic), entrambe con risoluzione pari a 7ns. Tuttavia, ci sono alcuni limiti: il numero e il tipo di commutazioni dei segnali digitali, la minima distanza tra due commutazioni e la minima distanza tra due comandi sono imposti dalla struttura del firmware e dipendono dal tipo di comandi inviati.
L'interfaccia dispone di un apposito ingresso di trigger-in, che può essere utilizzato per associare l'invio di un pattern di comunicazioni seriali e commutazioni digitali a eventi esterni, ovvero ai fronti di salita e discesa di un segnale di trigger. La sequenza di comandi in loop può essere letta e confrontata con comandi di fine loop definiti dall'utente. Qualora il confronto sia positivo, è possibile terminare il loop con eventuali commutazioni di segnali digitali. Sono disponibili anche due uscite digitali per la generazione di segnali PWM con frequenza e duty cycle personalizzabili.
La variazione delle tensioni di input/output e l'implementazione di una comunicazione LVDS sono permesse dall'utilizzo di componenti esterni al microcontrollore, rispettivamente voltage level shifter e LVDS driver/receiver.
Infine, è possibile gestire l'interfaccia attraverso un apposito driver Labview che ne permette l'integrazione in un sistema di test. All'utente sono fornite intuitive GUI che con-

sentono di sfruttare tutte le funzionalità appena descritte e definire i pattern da inviare attraverso un formato tabellare.
Non è stato possibile soddisfare tutte le richieste: alcuni risultati non sono stati ottenuti per il poco tempo a disposizione (driver Matlab), altri, invece, a causa delle soluzioni hardware/software adottate. La lettura del TER-bit non è prevista nel modulo USIC integrato nel microcontrollore, mentre la disconnessione meccanica delle linee e la protezione da cortocircuiti e sovratensioni richiedono lo sviluppo di una board dedicata. Il numero di segnali digitali e il numero e tipo di commutazioni all'intero del pattern è soggetto a limiti imposti dalla struttura software, ma è comunque sufficiente per la maggior parte delle applicazioni.

8.2 Sviluppi futuri

L'interfaccia è stata sviluppata su un evaluation board, ovvero un circuito stampato dove, oltre al microcontrollore, sono presenti appositi moduli di comunicazione (USB), connettori di alimentazione, debugger (USB) e connettori di accesso ad alcuni pin del microcontrollore. In questo modo, lo sviluppatore si deve occupare solamente della programmazione del firmware. Questa soluzione facilita la progettazione, ma limita il numero di interfacce di comunicazione e di segnali digitali disponibili. Inoltre, per poter collegare lo strumento con il dispositivo da testare, occorrono ulteriori connessioni che, soprattutto a elevate frequenze, disturbano i segnali.
Un possibile sviluppo di questo progetto è sviluppare una specifica board con accesso a tutti i pin del microcontrollore, per aumentare il numero di segnali digitali, ma soprattutto il numero di interfacce SPI/SCI. In alternativa, è possibile prevedere il diretto inserimento del microcontrollore sulle board di caratterizzazione dei dispositivi, in modo da minimizzare la distanza tra ricevitore e trasmettitore per ridurre i disturbi ad alte frequenze. Queste soluzioni, che prevedono la progettazione di una nuova board, possono includere i voltage level shifter, le interfacce LVDS e dispositivi per la disconnessione fisica e per la protezione da cortocircuiti e sovratensioni.
Il modulo USIC implementa svariati tipi di protocolli di comunicazione seriale: UART, LIN, I2C e IIS, che condividono lo stesso schema di validazione della trasmissione dei dati, già utilizzato per SPI/SCI e che permette la sincronizzazione con i segnali digitali. In particolare, I2C è un protocollo molto diffuso per comunicazioni a basse frequenze. Un ulteriore sviluppo è quindi l'inserimento di nuovi protocolli nell'interfaccia, per poterla utilizzare nella caratterizzazione di altri tipi di dispositivi.

Programmable digital lines interface module

Requisiti di progetto	Risultati ottenuti
tensione di input/output variabile da 3 a 5,5V (opzionale: inferiore a 2V)	OK, voltage level shifter
opzionale: protezione da cortocircuiti e sovratensioni	OK, voltage level shifter
opzionale: livello di tensione dell'interfaccia SPI/SCI e delle linee digitali configurabile individualmente	OK, voltage level shifter
opzionale: disconnessione meccanica delle connessioni	non soddisfatto
opzionale: interfaccia di comunicazione LVDS	OK, LVDS driver/receiver
interfacce comunicazione seriale: 1 SPI, 1 SCI (linee CS/SI/CLK/SO) (opzionale più di 1)	OK, 2 SCI e 2 SPI
lettura del TER bit	non soddisfatto
numero di bit da 1 a 1024	fino a 420-bit (per più di 64-bit solo valori multipli di 8)
frequenza CLK fino a 10MHz (opzionale fino a 50MHz)	OK
distanza tra comandi SPI programmabile (sequenze di comandi SPI fino a 100 elementi)	OK, risoluzione 7ns
possibilità di inviare sequenze di comandi SPI in loop	OK
ritardi (tid, tfd, tnf, ...) programmabili	OK, risoluzione 25ns
possibilità di scegliere polarità de segnali SPI (CS, CLK edge, ...)	OK
8 segnali digitali programmabili e sincronizzabili con l'interfaccia SPI/SCI	5 segnali digitali
2 canali di generazione segnali PWM (freq/duty cycle programmabili)	OK
1 linea digitali di input(opzionale 2): un segnale di tigger in ingresso che permetta di attivare una comunicazione SPI o commutazioni di linee digitali	OK
1 linea digitale di output: un segnale di trigger in uscita attivato da eventi (fine di comunicazione SPI o lettura programmabile dei comandi SPI ricevuti)	OK
comunicazione con PC attraverso l'interfaccia USB	OK
creazione di una GUI	OK, Labview
creazione di driver per Labview e Matlab	OK, Labview

Tabella 8.1: Requisiti di progetto e risultati.

Bibliografia

[1] Infineon Technologies (2015). *TLE75008-EMD*, Monaco: Infineon Technologies AG. http://www.infineon.com/dgdl/Infineon-TLE75008-EMD-DS-v01_01-EN.pdf?fileId=5546d4624fb7fef2014fef502f6c3395 [ultimo accesso: 17/01/2017].

[2] Motorola Inc. (2003). *SPI Block Guide V03.06*.

[3] Anand, N., Joseph, G., Oommen, S.S., Dhanabal, R. (2014). "Design and implementation of a high speed Serial Peripheral Interface". *2014 International Conference on Advances in Electrical Engineering (ICAEE)*, IEEE.

[4] Leens, F. (2009). "An introduction to I2C and SPI protocols". *IEEE Instrumentation & Measurement Magazine*, 12/1, pp. 8-13.

[5] Kugelstadt, T. (2009). "Signal Chain Basics (Part 31): Digital interfaces (con't) – The SPI Bus". *EETimes*. http://www.eetimes.com/document.asp?doc_id=1272534 [ultimo accesso: 14/01/2017].

[6] Freescale Semiconductor, Inc. (2004). *AN991/D, Rev.1, 1/2002: Using the Serial Peripheral Interface to Communicate Betweeen Multiple Microcomputers*.

[7] Maxim Integrated (2006). *Application Note 3947: Daisy-Chaining SPI Devices*. https://www.maximintegrated.com/en/app-notes/index.mvp/id/3947 [ultimo accesso 15/01/201].

[8] Infineon Technologies (2016). *Board User Manual XMC4700 XMC4800 Relax Kit Series*, Monaco: Infineon Technologies AG. http://www.infineon.com/dgdl/Infineon-Board_User_Manual_XMC4700_XMC4800_Relax_Kit_Series-UM-v01_02-EN.pdf?fileId=5546d46250cc1fdf01513f8e052d07fc [ultimo accesso 1/02/2017].

[9] Infineon Technologies (2016). *XMC4700/XMC4800 Data Sheet*, Monaco: Infineon Technologies AG. http://www.infineon.com/dgdl/Infineon-XMC4700-XMC4800-DS-v01_00-EN.pdf?fileId=5546d462518ffd850151908ea8db00b3 [ultimo accesso 1/02/2017].

[10] Infineon Technologies, (2016). *XMC4700 XMC4800 Reference Manual*, Monaco: Infineon Technologies AG. http://www.infineon.com/dgdl/Infineon-ReferenceManual_XMC4700_XMC4800-UM-v01_03-EN.pdf?fileId=5546d462518ffd850151904eb90c0044 [ultimo accesso 1/02/2017].

[11] Gosatwar, P., Ghodeswar, U. (2016). "Design of voltage level shifter for multi-supply voltage design". *2016 International Conference on Communication and Signal Processing (ICCSP)*, IEEE.

[12] Texas Instruments (2014). *SN74LVC1T45 Single-Bit Dual-Supply Bus Transceiver With Configurable Voltage Translation and 3-State Outputs*. `http://www.ti.com/lit/ds/symlink/sn74lvc1t45.pdf` [ultimo accesso 1/02/2017].

[13] Texas Instruments (2008). *LVDS Owner's Manual, Including High-Speed CML and Signal Conditioning*. `http://www.ti.com/lit/ml/snla187/snla187.pdf` [ultimo accesso 1/02/2017].

[14] ON Semiconductor (2001). *FIN1019 3.3V LVDS High Speed Differential Driver/Receive*, Aurora, Colorado, USA: Literature Distribution Centre for ON Semiconductor. `https://www.fairchildsemi.com/datasheets/FI/FIN1019.pdf` [ultimo accesso 1/02/2017].

Ringraziamenti

Desidero ringraziare la professoressa Giada Giorgi per la disponibilità e la cortesia dimostratemi, e per il prezioso aiuto fornito durante la stesura della tesi.

Un sentito ringraziamento va a tutta la divisione body power di Infineon Technologies Italia Srl, e in particolare all'ing. Enrico Mottin per l'occasione offerta e per avermi seguito costantemente con entusiasmo e professionalità durante il progetto.

Infine desidero ringraziare tutta la mia famiglia, in particolare mamma Ivana e mia sorella Stefania per il supporto incondizionato durante tutto il percorso di studi e di vita. Un ringraziamento a Giovanna, per essermi sempre vicina, e a tutti i miei amici.