SVILUPPO DI INTERFACCE DI COMUNICAZIONE DIGITALE CON LABVIEW FPGA

Vediamo come si può utilizzare il modulo LabVIEW FPGA per implementare un'ampia gamma di protocolli di comunicazione su una scheda d'interfaccia custom

a cura di Scott Savage

Gli ingegneri che sviluppano applicazioni di test usando protocolli di comunicazione digitale custom o non supportati possono utilizzare il modulo NI LabVIEW FPGA per implementare o prototipare rapidamente interfacce di comunicazione differenti sull'hardware di I/O riconfigurabile Serie R basato su Fpga. A differenza di quando si progetta e si costruisce hardware custom come un Asic o si scrive il proprio codice Vhdl da eseguire su un Fpga, con il modulo LabVIEW FPGA è possibile sviluppare, testare e mettere a punto facilmente nuova funzionalità senza la necessità di tool di sviluppo specializzati.

LABVIEW FPGA

Un'esigenza comune quando si sviluppano sistemi di test nei settori automotive, aeronautico e in molti altri campi industriali è quella di sviluppare o implementare interfacce per la comunicazione digitale fra il sistema di test e altri dispositivi. Tali dispositivi includono altri sistemi di test o computer, strumenti, DUT, componenti di sistema di basso livello come le ECU, e così via.

Idealmente, quando sviluppate un nuovo sistema di test, dovreste avere un'interfaccia già pronta per il vostro protocollo di comunicazione digitale che include un driver di semplice uso per il vostro tool di sviluppo delle applicazioni. In molti casi, tuttavia, la realtà è diversa. Potreste infatti trovarvi di fronte la necessità di sviluppare una scheda o altra interfaccia per comunicare con il vostro dispositivo esterno. Potete risolvere il poblema sviluppando il vostro hardware custom fino al limite di progettare il vostro Asic, o almeno assemblando una nuova scheda con componenti di serie. Con l'hardware di I/O riconfigurabile ed il modulo NI LabVIEW FPGA potete progettare il vostro hardware custom usando solo LabVIEW ed i suoi tool di programmazione grafica. Il progetto del vostro hardware custom è caricato nel chip Fpga dell'hardware di I/O riconfigurabile per creare la scheda d'interfaccia custom specifica, ritagliata sulla base delle vostre esigenze.

Descriveremo come potete utilizzare il modulo LabVIEW FPGA per implementare un'ampia gamma di protocolli di comunicazione sulla vostra scheda d'interfaccia custom. Queste tecniche sono utilizzabili sia per le schede di I/O riconfigurabile plug-in PCI e PXI (Serie R), sia per la piattaforma NI CompactRIO, in grado di offrire un sistema robusto per uso industriale.

PROTOCOLLI DI COMUNICAZIONE DIGITALE

Nei sistemi di test viene utilizzata un'ampia gamma di protocolli di comunicazione, che spaziano da interfacce estremamente comuni, come l'RS-232 e la Gpib, a protocolli custom implementati dai singoli fornitori e sviluppatori di applicazioni. Le interfacce più comuni, benché non siano largamente note, includono l'SPI (Serial Peripheral Interface) e l'I2C (Inter-Integrated Circuit), usate per la comunicazione all'interno e fra dispositivi e sistemi elettronici. Benché siano comuni nelle applicazioni, questi protocolli potrebbero non essere sempre supportati dalle interfacce di comunicazione disponibili in un'ampia gamma di piattaforme ed ambienti software.

Per queste esigenze, NI LabVIEW FPGA e la piattaforma di I/O riconfigurabile sono un tool ideale per ottenere una soluzione conveniente.

I protocolli di comunicazione digitale possono essere raggruppati per aree applicative o settori, nonché sulla base della natura e delle specifiche tecniche del protocollo stesso. Nei paragrafi seguenti discuteremo in modo approfondito i dettagli tecnici e la natura di vari protocolli ed illustreremo come utilizzare LabVIEW per implementarli. Per quanto riguarda le applicazioni ed i settori, qui sotto è riportato un insieme fondamentale di categorie e protocolli esemplificativi che includono interfacce di comunicazione risolte utilizzando LabVIEW FPGA

Comunicazione componenti/IC
Progettazione elettronica: SPI, I2C, Jtag, PS/2, …
Comunicazione di sistema
Aerospaziale: MIL-STD-1553, ARINC-429, …
Automotive: CAN, Most, KWP, 1939, …
Telecomunicazioni
Comunicazioni satellitari ed aerospaziali: PCM/Telemetria

Elettronica di consumo
Audio digitale: S/Pdif, I2S
Custom
Implementazioni specifiche per il dispositivo

CLASSIFICAZIONE DEI PROTOCOLLI

I protocolli di comunicazione digitale possono essere classificati sulla base della loro definizione tecnica e dei loro requisiti. Questo modo di considerare un protocollo, indipendentemente dall'applicazione, ci aiuta a focalizzarci sui dettagli che sono importanti perselezionare l'implementazione ottimale usando i tool di sviluppo disponibili. Il seguito di questo articolo si concentrerà su questi criteri di classificazione e sull'implementazione dei diversi gruppi di protocolli su Fpga.

La maggior parte delle interfacce di comunicazione digitale rientra in due categorie molto generali, seriale e parallela, in base al numero di segnali di dati fra il trasmettitore ed il ricevitore. Se i dati sono trasferiti su una singola linea dati si ha una linea dati seriale, dove tutti i bit sono trasmessi sequenzialmente o in serie. Una linea dati parallela ha più di una linea dati, spesso un multiplo di otto linee dati. I dati sono trasferiti in parallelo, normalmente un byte o parola per volta. Tradizionalmente, sia interfacce seriali (per es. RS-232) sia interfacce parallele (per es. Gpib) sono state utilizzate con linee di comunicazione parallela per ottenere velocità di trasferimento dati più elevate alla stessa frequenza di clock. Quando le velocità in bit possibili su una singola linea dati hanno raggiunto l'ordine dei Megabit e dei Gigabit, le linee seriali sono diventate molto più comuni e dominanti sul mercato, perché offrono una soluzione più conveniente, riducendo il costo dei componenti dell'interfaccia (per es. dei driver di linea) ed anche il costo del cablaggio. Nelle interfacce di comunicazione seriali e parallele vi possono essere linee di segnale addizionali usate per il clock e per il trigger, nonché per la comunicazione di controlli e comandi.
Una seconda classificazione fondamentale dell'interfaccia fisica di comunicazione è costituita dai livelli di tensione e dal riferimento di terra utilizzati per le linee di segnale digitali. I bit di dati ed altre informazioni sono trasmesse dai livelli di tensione variabili delle linee di segnale, ma vi può essere una differenza significativa nei livelli di tensione usati nei vari tipi di interfacce. In un segnale digitale single-ended, il livello di tensione della linea di segnale viene misurato rispetto ad un riferimento di terra comune. Un segnale digitale differenziale consiste di due segnali non referenziati ed il livello di tensione fra le due linee di segnale rappresenta il valore del segnale. I segnali differenziali sono più immuni al rumore captato dalle linee di segnale e possono essere normalmente utilizzati per trasferire segnali su distanze maggiori. Ai livelli di tensione specifici usati nella comunicazione digitale fanno riferimento standard come TTL, Cmos e Lvds e non vengono presi in considerazione nell'implementazione del protocollo sull'Fpga. Gli ingressi e le uscite digitali sull'hardware di I/O riconfigurabili usano livelli di tensione TTL/Cmos compatibili. Se sono richiesti altri livelli, è necessario inserire un traslatore/convertitore di segnale fra l'hardware dell'interfaccia e la linea di comunicazione.

SEGNALI E BIT-BANGING

Quando analizziamo un protocollo e consideriamo la sua implementazione in NI LabVIEW FPGA, dobbiamo identificare le varie linee di segnale utilizzate dal protocollo e lo scopo di ciascun segnale. Il numero di linee di segnale utilizzate dal protocollo determinerà quante risorse hardware dobbiamo configurare nel progetto LabVIEW e quante linee digitali utilizzeremo nel diagramma di LabVIEW. Il diagramma di temporizzazione riportato nella figura 1, relativo al protocollo SPI (Serial Peripheral Interface), illustra l'uso delle tre linee di segnale che dobbiamo gestire nell'implementazione di questo protocollo.

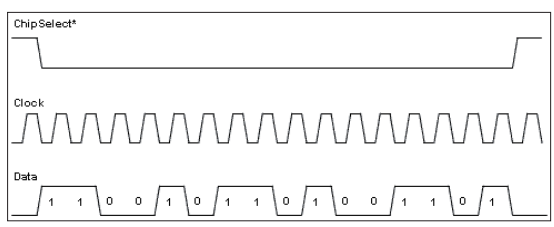

Figura 1 - Semplice diagramma di temporizzazione del protocollo SPI con una linea dati

Quando iniziamo l'implementazione ci rendiamo conto che per convertire il diagramma di temporizzazione nel codice NI LabVIEW FPGA corrispondente, dobbiamo risolvere due problemi – commutare on e off le linee digitali e attendere la quantità di tempo appropriata fra l'impostazione dello stato delle diverse linee digitali. Nelle applicazioni in cui leggiamo un protocollo digitale dobbiamo leggere lo stato delle linee digitali e notare la quantità di tempo fra le transizioni su ciascuna linea di segnale.

CLOCK

Uno dei prossimi criteri che utilizzeremo per classificare un protocollo è la sorgente di clock utilizzata dal protocollo. Le due categorie principali sono quelle dei protocolli sincroni e asincroni. I protocolli sincroni includono uno specifico segnale di temporizzazione nel protocollo, mentre i protocolli asincroni sono temporizzati utilizzando un bit rate definito.
Un tipico esempio di protocollo sincrono è il protocollo illustrato nel diagramma di temporizzazione della figura 1.
Il protocollo include un segnale di clock dedicato che permette a qualsiasi ricevitore di utilizzare la linea di clock come base tempi per leggere o scrivere la linea dati. Il protocollo asincrono probabilmente più noto è il bus seriale (RS-232) utilizzato su molti PC. Quando usiamo il bus seria-

le dobbiamo specificare il baud rate (bit al secondo) del dispositivo con il quale stiamo comunicando. Il baud rate è utilizzato dal trasmettitore e dal ricevitore per aggiornare o leggere il segnale dati alla stessa velocità. Poiché la sorgente temporale non può essere esattamente uguale alle due estremità della comunicazione, i pacchetti di dati della comunicazione asincrona hanno una lunghezza limitata per evitare l'errato allineamento dei bit. La comunicazione asincrona deve risincronizzarsi frequentemente per tenere conto di leggere differenze di temporizzazione. I protocolli di comunicazione sincrona, d'altra parte, possono comunicare continuamente perché sono sincronizzati su ogni bit di dati. Quando implementiamo un segnale di clock nell'ambito di un protocollo sincrono, utilizzeremo il clock dell'Fpga per determinare la base tempi del protocollo e aggiornare di conseguenza il segnale di clock.

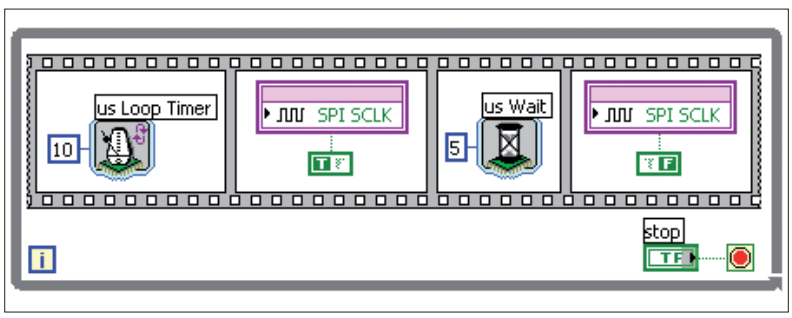

Figura 2 - Schema NI LabVIEW FPGA per generare un segnale di clock di 10 us (100 kHz)

Per un protocollo asincrono non viene generato un segnale di clock separato, ma si utilizzano il clock dell'Fpga ed un baud rate definito per determinare quando aggiornare o leggere il/i segnale/i dati.

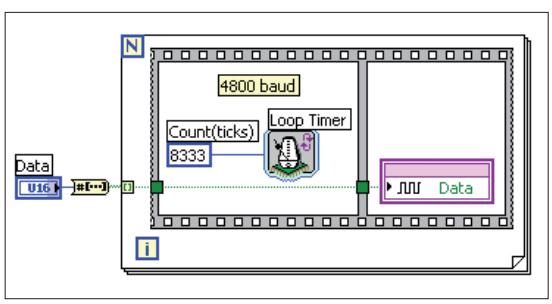

Figura 3 - Schema NI LabVIEW FPGA per aggiornare la linea dati a 4800 baud (8333 cicli di clock Fpga = 208,3 us = 1/4800 Hz)

CODIFICA/DECODIFICA – MODULAZIONE CODIFICATA DI IMPULSI

In base alla velocità specificata dal segnale di clock o dalla specifica di temporizzazione, i dati sono codificati sul segnale dati o letti dal segnale dati e decodificati. La codifica dei dati può essere eseguita in una varietà di formati differenti. Il metodo più comune è quello di rappresentare il valore di un bit di dati con uno o più stati del segnale dati. Questa classe generale di metodi di codifica è chiamata modulazione codificata di impulsi (Pulse Code Modulation - PCM).
Il sottoinsieme di metodi PCM in cui il bit di dati è rappresentato con un singolo stato del segnale dati è chiamatoNon-Return to Zero (NRZ). Per esempio, un segnale dati alto rappresenta un bit '1', mentre un segnale dati basso rappresenta un bit '0'. Questo metodo è chiamato NRZ-L perché il livello del segnale dati rappresenta il bit di dati. In alcuni protocolli, come l'RS-232, questa logica è invertita. Quindi, lo stato alto rappresenta '0' e lo stato basso rappresenta '1'. Questo è chiamato NRZ-I (inverso). Altri sottotipi dell'NRZ sono l'NRZ-M (mark) e l'NRZ-S (space), dove il bit di dati '1' è rappresentato da un cambiamento

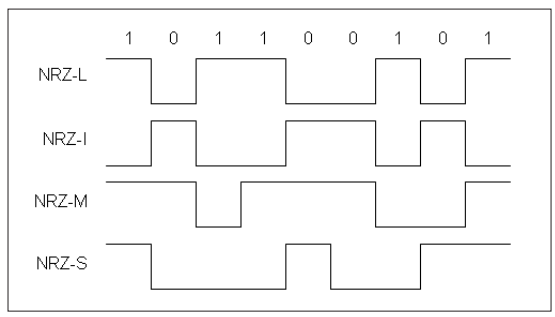

Figura 4 - Metodi di codifica NRZ (non-return to zero)

del segnale dati (NRZ-M) o uno '0' è rappresentato da un cambiamento del segnale dati (NRZ-S).
La modulazione codificata di impulsi include un altro insieme di schemi di codifica che combina il segnale di clock ed il segnale dati in una linea dati; ciascun bit è rappresentato da stati multipli del segnale dati. Questa codifica è chiamata bifase ed il comune schema di codifica Manchester è un tipo di codifica bifase.
Altri schemi di codifica più avanzati includono diverse forme di codifica a larghezza d'impulso. Lo schema di modulazione a larghezza d'impulso (PWM) comunemente utilizzato converte direttamente un valore analogico nella larghezza d'impulso variabile di un treno d'impulsi a frequenza costante. Altre forme di codifica a larghezza d'impulso usano due larghezze d'impulso differenti per rappresentare un bit '0' e un '1' in una sequenza di bit tradizionale.

IMPLEMENTAZIONE IN LABVIEW FPGA

Ora che abbiamo una nomenclatura base per descrivere i protocolli comuni possiamo esaminare come implementare protocolli in NI LabVIEW FPGA. Per implementare un protocollo in LabVIEW FPGA, inizieremo tipicamente scorrendo il diagramma di temporizzazione e convertendo i cambiamen-

ti di stato delle diverse linee di segnale e la temporizzazione fra tali cambiamenti nelle corrispondenti funzioni e strutture LabVIEW. Ogni cambiamento di stato di una linea digitale è implementato usando il nodo di I/O dell'Fpga e la temporizzazione è implementata usando le funzioni di temporizzazione di LabVIEW FPGA (Loop Timer e Wait). I passi nel diagramma di temporizzazione che si ripetono più volte sono implementati usando il For Loop o il While Loop. Per protocolli articolati, gruppi di funzioni e strutture possono essere incapsulati in subVI per consentire il riutilizzo del codice e rendere il codice più modulare e gestibile.

USCITA SPI

Come primo esempio implementeremo il diagramma di temporizzazione SPI illustrato nella precedente figura 1. La comunicazione con il protocollo SPI consiste in pacchetti di dati trasmessi fra due dispositivi. Nella comunicazione SPI c'è un dispositivo master che controlla il segnale ChipSelect ed il segnale Clock. Possono quindi esserci uno o più dispositivi slave con una linea di segnale ChipSelect dedicata dal master a ciascuno slave ed una linea dati comune per tutti i dispositivi. Se l'applicazione include una comunicazione in entrambe le direzioni fra il master e lo/gli slave, normalmente vengono utilizzate due linee dati, chiamate Master Out Slave In (Mosi) e Master In Slave Out (Miso). Nel nostro esempio prenderemo in esame una sola linea dati.

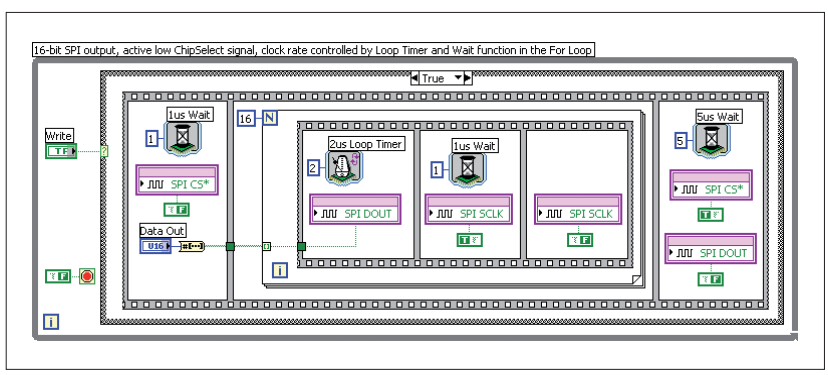

Figura 6 - Schema NI LabVIEW FPGA di una semplice implementazione d'uscita SPI

I trasferimenti di pacchetti sono sempre iniziati dal dispositivo master, che attiva la linea ChipSelect del dispositivo slave indirizzato. Normalmente, la linea ChipSelect è attiva bassa, quindi rimane in uno stato alto quando il sistema è inattivo e viene abbassata dal master per iniziare una trasmissione. Dopo l'attivazione del segnale ChipSelect, il master aggiorna la linea dati e commuta quindi la linea di clock per trasferire ciascun bit di dati allo slave. Il contenuto del pacchetto dati è specifico per l'applicazione ed è lasciato alla definizione dello sviluppatore e del progettista di dispositivi.

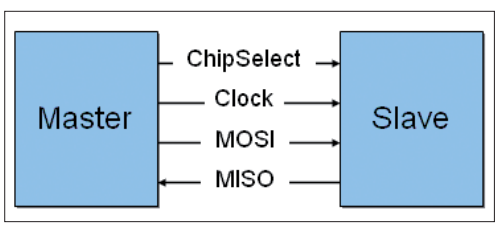

Figura 5 - Schema di cablaggio per la comunicazione SPI fra due dispositivi

Nello schema NI LabVIEW FPGA corrispondente (figura 6) ogni trasferimento di dati è iniziato settando il controllo booleano 'Write'. Ciò provoca l'esecuzione del caso True. Inizialmente, la linea ChipSelect (SPI CS*) viene attivata forzando bassa l'uscita digitale. Nello stesso tempo, il valore del dato 'Data Out' è convertito nella corrispondente sequenza di bit (array booleano). La funzione di attesa di un microsecondo (1us) lascia al dispositivo slave il tempo di prepararsi per i bit di dati che seguono. Viene quindi ripetuta una sequenza di aggiornamento della linea dati e di commutazione della linea di clock per 16 volte nel loop For. Ogni bit dell'array booleano viene posto in uscita sulla linea dati (SPI Dout), seguito dal settaggio della linea di clock (SPI Sclk) prima allo stato alto e poi allo stato basso. La funzione Loop Timer permette l'esecuzione del loop ad intervalli di 2 us, mentre la funzione Wait controlla la durata della fase alta del segnale di clock in modo che sia di 1 us, generando un segnale di clock di 500 kHz con un duty cycle del 50%. Dopo la generazione di tutti i 16 bit, le linee Chip Select e dati vengono riportate nel loro stato di inattività e inserita la funzione Wait per un tempo di inattività minimo di 5us. A questo punto l'Fpga è pronto per il prossimo comando Write.

In base alla specifica dei dispositivi SPI utilizzati e all'applicazione, qualsiasi parametro della comunicazione (per es. numero di bit di dati da trasferire, frequenza del segnale di clock, ecc.) può essere regolato sullo schema o controllato dinamicamente usando un controllo sul pannello frontale del VI LabVIEW FPGA.

Se è necessario, è possibile impostare programmaticamente la direzione di ciascuna delle linee digitali utilizzate nell'applicazione usando il Set Output Enable Fpga I/O Method Node (figura 7).

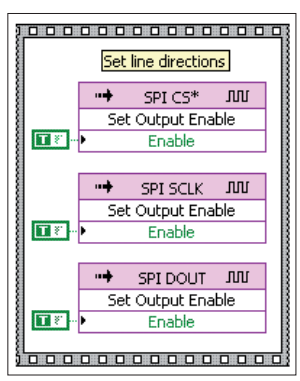

Figura 7 - Inizializzazione NI LabVIEW FPGA di linee digitali

INGRESSO SPI

L'implementazione della parte di input di un protocollo può presentare alcuni problemi unici, perché il codice deve essere più flessibile nel rilevamento e nell'elaborazione del protocollo. Anziché aggiornare le linee d'uscita digitali ed inserire i ritardi appropriati, il codice deve monitorare lo stato di diverse linee digitali e, se necessario, misurare il tempo fra specifiche transizioni sulle linee di segnale.

L'esempio seguente illustra un'implementazione fondamentale di un ingresso SPI, corrispondente al precedente esempio d'uscita SPI. Mentre è nello stato Idle, l'Fpga monitora la linea ChipSelect e rileva eventuali fronti di discesa. In risposta ad un fronte di discesa, esso inizia a monitorare la linea di clock. Per ciascuno dei 16 fronti di salita del segnale di clock, l'Fpga legge il segnale dati e memorizza il valore del bit in un array booleano preallocato. Al termine del pacchetto dati, dopo 16 cicli di clock, l'array booleano è convertito in un valore dato intero e reso disponibile sul pannello frontale del VI.

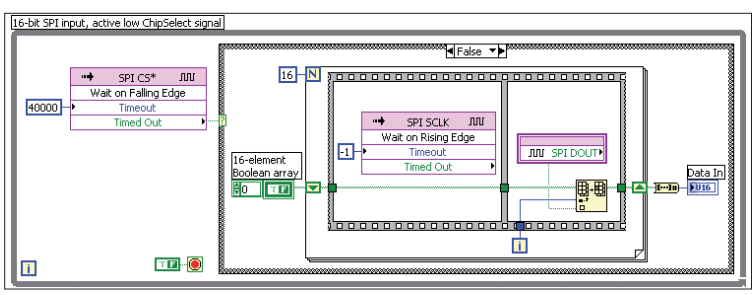

Figura 8 - Schema NI LabVIEW FPGA di una semplice implementazione d'ingresso SPI

Data la natura del protocollo SPI, questo esempio non richiede l'esecuzione di alcuna misura temporale, perché tutta la temporizzazione è controllata direttamente dai fronti nel protocollo.

CONSIDERAZIONI SULL'IMPLEMENTAZIONE

In questo paragrafo affronteremo una serie di argomenti addizionali da considere quando si sviluppa un protocollo di comunicazione digitale usando NI LabVIEW FPGA.

LINEE DI SEGNALE OPEN-COLLECTOR/OPEN-DRAIN

Nei precedenti esempi SPI, ogni linea di segnale è pilotata solo da un singolo dispositivo. In molti protocolli, tuttavia, le linee di segnale possono essere pilotate o controllate da più di un dispositivo, in funzione dello stato del bus o della comunicazione. Ciò permette a più di un dispositivo di iniziare la trasmissione sul bus o di utilizzare la stessa linea dati per trasmettere e ricevere dati. Tipicamente, ciò si ottiene utuilizzando un circuito open-collector/open-drain. In questa configurazione, un dispositivo può soltanto pilotare una linea di segnale nello stato basso, lasciando fluttuare la linea di segnale se vuole settare la linea nello stato alto o se non vuole pilotare la linea stessa. Su una linea di segnale, oltre a tutti i dispositivi collegati, c'è anche un resistore di pull-up ad una tensione prefissata per stabilire la tensione alta della linea se nessuno dei dispositivi collegati forza il segnale nello stato basso. Usando questa configurazione, qualsiasi dispositivo può forzare la linea nello stato basso senza creare una contesa di tensione fra dispositivi differenti. Questi segnali sono tipicamente definiti come attivi-bassi: ciò significa che la linea è in uno stato alto quando il bus è inattivo e che un dispositivo attiva il segnale forzando bassa la linea. I segnali attivi-bassi sono spesso indicati con una barra attraverso il nome del segnale o un asterisco dopo il nome del segnale, come nel caso del segnale ChipSelect* (SPI CS*) utilizzato negli esempi precedenti.

Per implementare un segnale open-collector in NI LabVIEW FPGA utilizziamo la capacità di controllare la direzione della linea digitale in modo da distinguere fra il pilotaggio della linea nello stato basso e la sua fluttuazione. In LabVIEW FPGA, si usa un I/O Method Node per abilitare o disabilitare una linea d'uscita digitale. Mentre una linea è disabilitata, l'Fpga non pilota la linea digitale e le permette di fluttuare nello stato alto. Per forzare bassa la linea quando è abilitata, impostiamo Output Data su False. Output Data è un registro software che mantiene il suo valore indipendentemente dal fatto che la linea sia impostata per essere pilotata o lasciata fluttuare.

Le figure seguenti illustrano un esempio di configurazione open-collector in LabVIEW FPGA per il protocollo I2C (Inter-Integrated Circuit). L'I2C è utilizzato in applicazioni simili a quelle dell'SPI per comunicare con diversi tipi di circuiti integrati, come Eeprom, ADC, DAC, ecc. Il bus I2C ha solo due linee di segnale, clock e dati, ciascuna delle quali è una linea open-collector. Per selezionare il corretto ricevitore per una comunicazione, il trasmettitore trasmette prima un indirizzo di dispositivo unico che specifica il dispositivo ricevente.

Per inizializzare il VI per una comunicazione open-collector, le due linee di segnale (SCL e SDA) sono disabilitate per portare il bus nello stato inattivo. L'Output Data di ciascun segnale è impostato su False. Da questo punto in poi, lo stato di ciascuna linea è controllato usando il metodo Set Output Enable per abilitare la linea e forzarla bassa o per disabilitarla e lasciarla fluttuare alta.

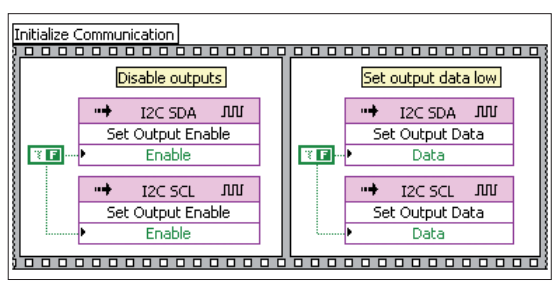

Figura 9 - Configurazione di due linee digitali per la comunicazione open-collector

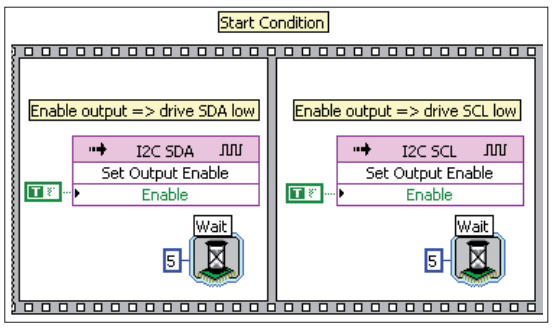

Figura 10 - Implementazione della condizione di Start I2C sul bus

Per iniziare una trasmissione sul bus I2C, il trasmettitore trasmette una condizione di start forzando bassa la linea dati (SDA) e forzando successivamente bassa la linea di clock (SCL). Per trasmettere un bit di dati sul bus I2C (figura 11), il trasmettitore aggiorna la linea dati (1° frame) e commuta quindi la linea di clock da alta (2° frame) a bassa (4° frame).
Nel protocollo I2C, dopo il trasferimento di ogni byte di dati (8 bit), viene inserito un ciclo di clock addizionale nella comunicazione per consentire al dispositivo ricevente di riconoscere l'avvenuta ricezione del byte precedente. A tale scopo, il ricevitore forza bassa la linea dati durante il 9° ciclo di clock. Sul lato trasmettitore, nel nostro esempio, ciò è implementato trasmettendo un bit di dati alto e controllando lo stato attuale della linea dati (3° frame) durante il ciclo di clock. Benché il trasmettitore stia lasciando fluttuare alta la linea dati, l'effettivo valore del dato dovrebbe essere basso, perché il ricevitore sta forzando bassa la linea dati per riconoscere l'ultimo byte di dati.

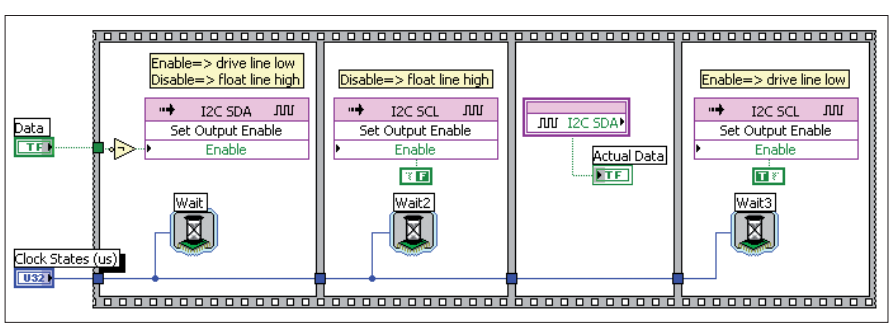

Figura 11 - Trasmissione di un bit di dati sul bus I2C

IL MODELLO DI RIFERIMENTO OSI

Il modello di riferimento OSI (Open Systems Interconnection) è una rappresentazione dei diversi strati logici di un protocollo di comunicazione, inclusa l'applicazione che sta usando il protocollo di comunicazione. Esso è utilizzato per definire e comprendere meglio i differenti aspetti di un protocollo e di una rete.
Lo strato 1 è lo strato fisico, riguardante i dettagli elettrici e meccanici che offrono la possibilità di inviare dati su una portante. I semplici pacchetti asincroni operano a questo livello. Lo strato 2 fornisce la definizione base dei dati a livello di bit e byte, nonché della modulazione, al di là di NRZ e sincronizzazione.
Il successivo insieme di strati riguarda l'indirizzamento, la definizione dei pacchetti di dati, il controllo degli errori, ecc. Come possiamo vedere dai nostri esempi, l'Fpga opera allo Strato Fisico L1, interfacciandosi direttamente con ciascuna delle linee digitali d'ingresso e d'uscita al livello elettrico. Ciò significa che, come sviluppatori di un protocollo di comunicazione digitale in NI LabVIEW FPGA, siamo responsabili di tutti gli strati del modello di riferimento per quanto riguarda il protocollo e l'applicazione. Per molti protocolli più semplici ciò richiede soltanto un po' più di programmazione, rispetto a quanto abbiamo fatto finora, per aggiungere un'interfaccia in cima allo strato del protocollo per consentire all'applicazione di interagire con l'Fpga e usare il protocollo. Tuttavia, i protocolli più avanzati potrebbero richiedere una programmazione significativamente maggiore per ottenere questi strati intermedi del protocollo, come il rilevamento e la gestione degli errori a livello del bus, la costruzione e distribuzione dei pacchetti, la gestione di diversi tipi di pacchetti, ecc.

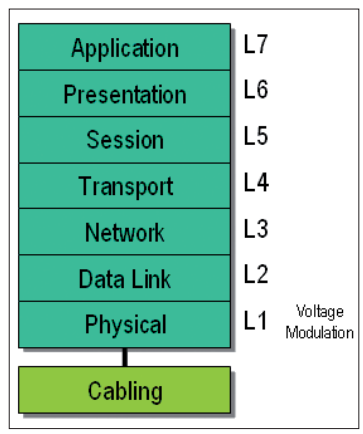

Figura 12 - Il modello di riferimento OSI

TEMPORIZZAZIONE E RISOLUZIONE

Come abbiamo visto, la temporizzazione gioca un ruolo molto importante nello sviluppo dell'implementazione di un protocollo. Le linee dati e di clock devono essere aggiornate ad intervalli precisi. Quando si decodifica il protocollo è importante misurare con precisione questi stessi intervalli. E' importante tenere presente il comportamento della temporizzazione dell'Fpga quando si sviluppano protocolli di comunicazione, per fare in modo che la propria implementazione risponda ai requisiti ed alle specifiche del protocollo.
L'Fpga opera su una frequenza di clock base e tutte le funzioni di temporizzazione si basano sulla stessa frequenza di clock. Per NI LabVIEW FPGA, la frequenza di clock di default dell'Fpga è di 40 MHz, quindi ogni ciclo di clock ed unità di tempo è pari a 25 nanosecondi (ns). In LabVIEW FPGA pos-

siamo specificare intervalli di tempo per le funzioni di temporizzazione in unità di millisecondi, microsecondi e tick. Ogni tick corrisponde ad un ciclo di clock o 25 ns. Ciò significa che la massima risoluzione di cui disponiamo per specificare qualsiasi ritardo o tempo di iterazione di un loop è di 25 ns. Benché questa precisione sembri molto elevata, quando procediamo al contrario e determiniamo le possibili frequenze che possiamo generare, vediamo l'effetto che si ottiene alle frequenze più alte.
Per generare una frequenza di aggiornamento di 1 MHz per una linea digitale possiamo usare una funzione Loop Timer impostata per 40 tick.
1 MHz => 1 us = 1000 ns
1000 ns / 25 ns per tick = 40 tick
E se volessimo una frequenza di aggiornamento di 1,25 MHz?
1.25 MHz =>0,8 us = 800 ns
800 ns / 25 ns per tick = 32 tick
Per 1,25 MHz usiamo un ritardo di 32 tick. E se volessimo una frequenza di aggiornamento di 1,1 MHz?
1,1 MHz => 0,9091 us = 909,1 ns
909,1 ns / 25 ns per tick = 36,36 tick
Poiché non possiamo scegliere specifici cicli di clock parziali, dovremmo scegliere 36 cicli di clock, pari ad un ritardo di 900 ns e corrispondenti ad una frequenza di 1,111 MHz. La frequenza inferiore più vicina che potremmo usare è pari a 37 tick o 1,081 MHz. Quindi, possiamo vedere che usando il clock di 40 MHz dell'Fpga possiamo aggiornare le linee digitali a 1,081 MHz o 1,111 MHz, ma non a frequenze comprese fra questi due valori.
Usando le proprietà dell'Fpga e le risorse di clock nel progetto LabVIEW, abbiamo la possibilità di cambiare la frequenza clock base dell'Fpga a 80 o 120 MHz. Questo farà funzionare l'Fpga ad una frequenza più elevata e migliorerà la risoluzione di temporizzazione delle funzioni di temporizzazione. Tuttavia, la maggiore frequenza di clock avrà anche l'effetto di ridurre il livello di complessità utilizzabile nello schema del VI Fpga.
Quando si ha intenzione di implementare un protocollo temporizzato è necessario determinare se la risoluzione di temporizzazione dell'Fpga è adeguata per le esigenze del protocollo e dell'applicazione. Ciò è particolarmente importante per la temporizzazione richiesta per generare un protocollo, come i calcoli precedenti hanno dimostrato. Per la lettura e la decodifica di protocolli spesso è sufficiente essere in grado di leggere le linee di segnale ad una frequenza maggiore della massima frequenza di aggionamento del protocollo. Per sicurezza, dovremmo riuscire a campionare i segnali ad una frequenza almeno doppia di quella di aggiornamento.
Se dobbiamo eseguire misure di tempo su un segnale d'ingresso per decodificare i dati contenuti nel protocollo, dobbiamo eseguire calcoli addizionali per determinare la risoluzione di temporizzazione richiesta in modo da decodificare con precisione il protocollo.

MACCHINE A STATI

Le macchine a stati possono essere una tecnica utile per implementare la codifica o decodifica di un protocollo di comunicazione digitale. In LabVIEW, le macchine a stati possono essere facilmente implementate usando il ciclo While ed una struttura Case per rappresentare i diversi stati.

La macchina a stati aiuta nello sviluppo di protocolli di comunicazione perché suddivide in modo naturale il diagramma di temporizzazione in passi separati, ciascuno dei quali può essere convertito in uno stato separato nello schema NI LabVIEW FPGA. Questo isola ogni passo di programma e riduce la complessità della programmazione in ogni parte dell'implementazione. Inoltre, semplifica operazioni generiche come la gestione degli errori e delle eccezioni, permettendoci di saltare da qualsiasi punto nell'esecuzione del protocollo a speciali stati di gestione degli errori. Qualche volta, le specifiche del protocollo sono scritte in termini di una macchina a stati, che potete tradurre direttamente in un diagramma a stati LabVIEW.
Per lo schema di temporizzazione SPI nella figura 1, sono possibili i passi seguenti per suddividere il diagramma di temporizzazione in una macchina a stati.

- Set ChipSelect low
- Set Data (0)
- Set Clock high
- Set Clock low
- Set Data (1)
- Set Clock high
- Set Clock low
  (ripetizione di Data e Clock per i bit 2-15)
- Set ChipSelect high

Osserviamo che ci sono cinque passi distinti, nonostante alcuni di essi si ripetano per ciascun bit di dati. In origine abbiamo implementato questi passi ripetitivi in un ciclo For. Nella macchina a stati, creiamo uno stato unico per ciascuno di questi cinque passi e li eseguiamo quindi ciclicamente. Per i tre passi ripetitivi che aggiornano la linea dati e commutano la linea di clock, configuriamo la macchina a stati in modo che ripeta questa sequenza 16 volte prima di procedere all'ultimo passo. Nella macchina a stati LabVIEW, ciò viene implementato usando un contatore nel registro a scorrimento del ciclo While.

USO DEL SINGLE CYCLE TIMED LOOP

Un altro vantaggio dell'architettura della macchina a stati è che ci permette di realizzare l'implementazione di un protocollo all'interno di un Single Cycle Timed Loop (Sctl) NI LabVIEW FPGA. L'Sctl assicura un'esecuzione più veloce dello schema LV Fpga, permettendo di eseguire ogni ciclo del loop in un ciclo di clock. Questo ci consente di aggiornare una linea di segnale alla frequenza di clock base dell'Fpga.
L'Sctl ottimizza inoltre la generazione di codice, quindi il

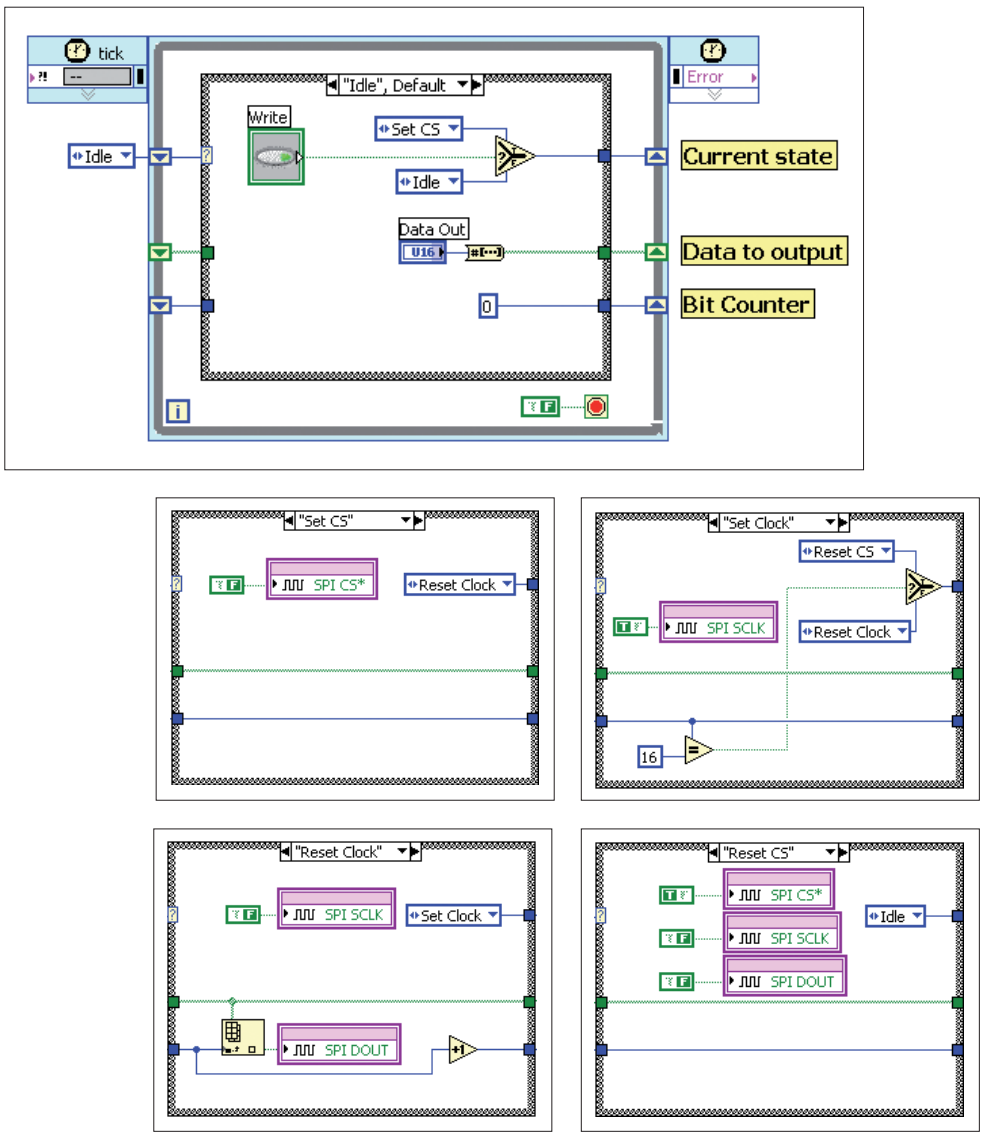

Figura 13 - Implementazione dell'uscita SPI in un Single Cycle Timed Loop

codice sull'Fpga è più efficiente ed utilizza meno risorse fisiche dell'Fpga. Tuttavia, vi sono varie limitazioni sul codice implementato all'interno di un Sctl. Per esempio, possiamo accedere a ciascuna delle linee di segnale una sola volta per iterazione dell'Sctl. Pertanto, dobbiamo definire gli stati della nostra macchina a stati in modo da leggere o aggiornare ciascuna linea di segnale una sola volta per stato.

Gli schemi seguenti (figura 13) illustrano l'implementazione del protocollo d'uscita SPI usando una macchina a stati all'interno di un Sctl. Lo stato **Idle** attende che il comando Write inizi l'emissione del prossimo pacchetto di dati. Esso converte inoltre il valore del dato in un array booleano per l'operazione d'uscita. Il passo successivo **Set CS** attiva la linea ChipSelect. Inizia quindi l'emissione dei bit di dati. In **Reset Clock** aggiorniamo la linea dati con il valore del prossimo bit e resettiamo il segnale di clock. In **Set Clock** settiamo il segnale di clock che fornisce un trigger al ricevitore per leggere il segnale dati. In questa coppia di stati incrementiamo un contatore in un registro a scorrimento, che seleziona il corretto valore dati dall'array booleano, per inviarlo sulla linea dati. Dopo l'emissione del 16° bit, transitiamo nello stato **Reset CS** che resetta tutte le linee di segnale nello stato di inattività del bus. Da qui torniamo nello stato Idle ed attendiamo il prossimo comando Write.

Note sull'autore

Scott Savage è il Product Marketing Manager della linea di prodotti di I/O digitale ad alta velocità in National Instruments