Implementazione di un modulo CRC mittente/destinatario in VHDL

Esame di Digital Systems Design

David Costa, Giuliano Peraz

14 gennaio 2013

Cos'è il CRC

CRC (Cyclic Redundancy Code/Check) è un algoritmo di rilevazione degli errori utilizzato spesso nelle trasmissioni dati quali USB, ATM, GSM.

Il funzionamento si basa sull'invio di un messaggio di M bit ai quali seguono F bit di ridondanza che contengono il codice CRC.

Il messaggio binario è considerato come un polinomio di ordine M-1 in cui ogni bit rappresenta un coefficiente della somma di potenze, ad esempio:

$1\,0000\,1010 = 1 \cdot x^8 + 0 \cdot x^7 + 0 \cdot x^6 + \cdots + 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0 = x^8 + x^3 + x$

L'algoritmo prevede la definizione di un polinomio di ordine F che sarà usato per calcolare il codice CRC da spedire, oppure per verificare l'integrità del dato ricevuto, per tale ragione il polinomio divisore deve essere noto sia al mittente che al destinatario. L'efficacia nella rivelazione degli errori dipende fortemente dal polimonio divisore (che deve possedere alcune caratteristiche matematiche particolari) per questa ragione ne esistono già alcuni standard di comprovata efficacia. Una lista esaustiva di tali codici si può reperire sul sito su Wikipedia inglese alla voce Cyclic Redundancy Check.

Come si calcola il CRC

Il calcolo del CRC non è altro che una divisione polinomiale tra il messaggio e polinomio divisore. Al termine della divisione si avranno risultato e resto, si scarta il risultato e si salva il resto della divisione negli F bit del codice. Gli F bit saranno poi inseriti in coda agli M bit del messaggio per ottenere l'intera stringa da inviare.

[FIGURA: schema ricevitore_trasmettitore]

Al termine della trasmissione della stringa di bit lunga M+F bit il destinatario potrà verificare la correttezza dividendo ciò che ha ricevuto per lo stesso polinomio divisore del mittente, se il resto è nullo allora il messaggio trasmesso è stato ricevuto correttamente.

Implementazione

Noi abbiamo implementato un modulo CRC mittente/destinatario sincrono, cioè un componente hardware capace di funzionare sia come mittente di una stringa di bit formata da messaggio + CRC sia come destinatario in grado di verificare se la stringa è giunta a destinazione senza alcun errore. Le specifiche fornite impongono M=56 bit e F=8 bit per un totale T=64 bit di messaggio da trasmettere o da ricevere.

Il comportamento del modulo è il seguente:

- mittente: riceve in input una sequenza di 56 bit che inizia a trasmettere dopo 8 cicli di clock e durante la ricezione del messaggio è calcolato il CRC da inserire in coda al messaggio di output;
- destinatario: riceve in ingresso un messaggio di 64 bit del quale deve calcolare il CRC. In output è replicato il messaggio ricevuto in ingresso ma gli ultimi 8 bit segnalano l'effettiva correttezza del messaggio ricevuto.

Noi abbiamo impostato che gli ultimi 8 bit siano uguali a 0 se il messaggio ricevuto è corretto, questo per poter sfruttare la capacità di rivelazione dell'errore da parte dell'algoritmo CRC senza dover aggiungere ulteriori moduli che potessero complicare il design.

Il chip ha la seguente interfaccia:

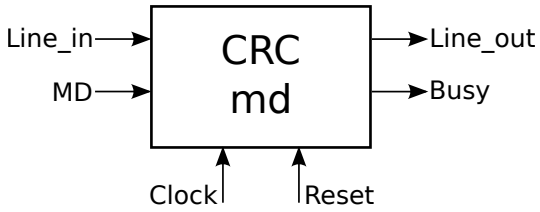


Figura 1: Interfaccia CRC_Module

Descrizione dei segnali:

- md : indica al chip se funzionare come mittente o come destinatario (0 = mittente, 1 = destinatario);
- line_in : linea seriale in ingresso, da questa vengono trasmessi/ricevuti i bit;
- clock : temporizzatore per sincronizzazione del circuito e della logica interna;
- reset : per ripristinare il circuito e portarlo in uno stato conosciuto;
- line_out : output seriale dal quale escono i 64 bit (56 messaggio + 8 CRC se mittente, 56 + 8 di verifica se destinatario);
- busy : indica quando il circuito non accetta input.

Descrizione dei blocchi

- SHIFT_REGISTER: serve principalmente per sincronizzare l'uscita e per ritardare di 8 cicli di clock l'invio del messaggio in modo da calcolare il codice CRC e spedire tutti i 56 + 8 bit contiguamente, sia che il modulo sia impostato come mittente sia come destinatario.
- CRC_LOGIC: è uno shift register con la possibilità di abilitare la funzione di calcolo del CRC mediante il segnale enable_crc;
- CRC_CONTROLLER: abilita il calcolo del codice CRC e controlla quale siano i bit da spedire su line_out;
- MULTIPLEXER: multiplexa l'uscita tra lo SHIFT_REGISTER e CRC_LOGIC in modo da inviare prima tutti i 56 bit del messaggio ricevuto e poi gli 8 bit di CRC/codice di rilevazione errori contenuti nel CRC_LOGIC;

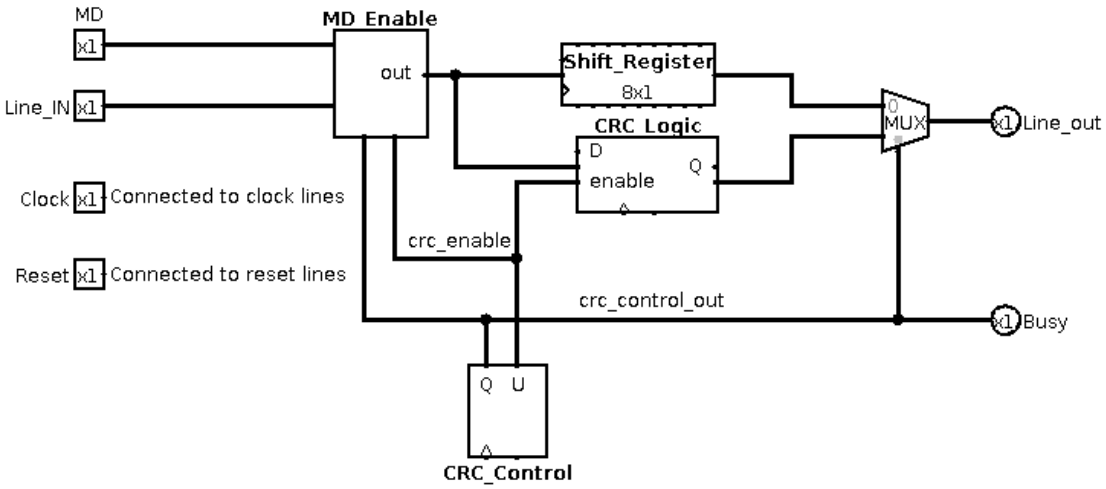


Figura 2: Schema a blocchi di CRC_Module

- MD_ENABLE: imposta il comportamento della logica come mittente o come destinatario, disabilita la propagazione del segnale di CRC_CONTROLLER alla CRC_LOGIC per continuare a calcolare il CRC nel caso il modulo fosse impostato come destinatario.

I blocchi SHIFT_REGISTER, CRC_LOGIC e CRC_CONTROLLER hanno parametri generici in modo da essere riutilizzabili per calcolare diversi polinomi o per essere utilizzati in progetti successivi.

Criteri utilizzati nella progettazione

Il design scelto è stato implementato tenendo in considerazione le seguenti metriche per la realizzazione del modulo hardware:

- prestazioni: invece di salvare tutto il messaggio in una memoria, calcolare il CRC, salvare quest'ultimo e poi inviare tutti i T bit, il nostro modulo impiega un numero pari a F cicli di clock prima di inviare l'intero messaggio, cioè quelli strettamente necessari a CRC_LOGIC per il calcolo del codice senza perdere tempo;

- robustezza: ogni modulo è stato realizzato pensando e testando i casi in cui si potrebbe avere un input indesiderato o che possa portare a output non definiti, a tal proposito è stato sviluppato una semplice procedura di controllo tra output fornito dal nostro modulo e output atteso;

- occupazione di area: gli elementi sono stati pensati e implementati avendo in mente la minore occupazione di area possibile per il chip anche grazie a un design che permette il riutilizzo dei moduli per fungere sia da logica di calcolo, sia da celle di memoria e scegliendo componenti che occupano il minore spazio possibile ma che permettano di far funzionare correttamente il modulo;

- Riusabilità del codice: sono stati creati moduli generici quali il CRC_LOGIC e il CRC_CONTROLLER che possono essere usati per altri progetti, il polinomio divisore e la sua lunghezza sono configurabili;
- manutenzione: il codice VHDL è stato scritto per essere il più chiaro possibile in modo che chiunque possa essere in grado di leggerlo senza difficoltà.

Implementazione dei blocchi

SHIFT_REGISTER

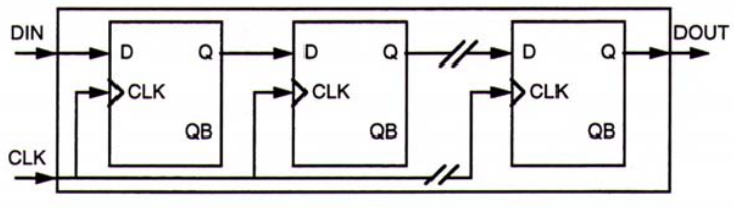


Figura 3: Struttura di uno shift-register

È stato scelto uno shift register perché permette di temporizzare l'uscita con il clock e perché è stato necessario inserire un ritardo di F cicli di clock per il calcolo del CRC. In generale il modulo deve aspettare un numero di cicli di clock pari al numero di flip flop che compongono il CRC_LOGIC.

CRC_LOGIC

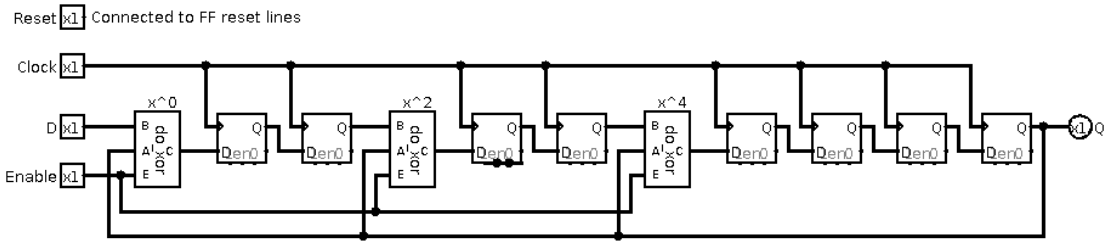
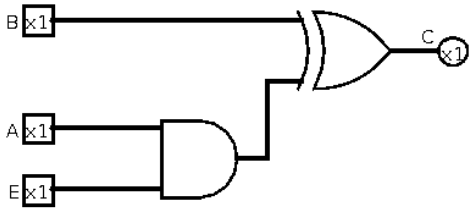


Figura 4: Struttura di CRC_Logic

Il modulo che implementa la logica non è che uno shift register con la possibilità di inserire le celle DO_XOR tra un flip flop e l'altro in modo da poter configurare il modulo perché possa calcolare polinomi differenti da quello fornito nelle specifiche.

È possibile abilitare le celle DO_XOR tramite il segnale enable_crc, in questo modo anziché utilizzare due shift register, uno adibito al calcolo del CRC e uno al suo salvatag-



A	B	E	C
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0
-	0	0	0
-	1	0	1

Figura 5: Funzione logica di DO_XOR

gio, si ottimizza il componente utilizzando un solo shift register per svolgere entrambe le funzioni.

Nel nostro caso il modulo implementa la divisione polinomiale tra il messaggio in input e il polinomio dato dalle specifiche ($x^8+x^4+x^2+1$). All'inizio della ricezione del messaggio il modulo è resettato in modo da eliminare elementi di disturbo che potrebbero essere presenti nelle celle di memoria. Questo comportamento è stato introdotto per motivi di robustezza.

CRC_CONTROLLER

È un contatore generico che conta i cicli di clock e fornisce in output due uscite che rimangono attive per un numero di cicli di clock configurabile.

Per questo modulo abbiamo provveduto all'implementazione di due architetture per avere due comportamenti diversi a seconda delle necessità dell'utente:

1. Architettura NO_PRELOAD:

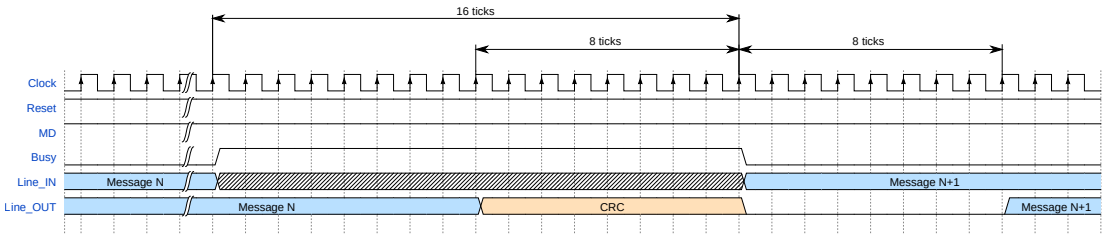


Figura 6: Trasmissione con architettura NO_PRELOAD

Tra due invii di messaggio + CRC è presente un byte nullo in modo che un ricevente possa riconoscere quando iniziano i messaggi. Questo è stato pensato per l'utilizzo di un ricevitore che non si aspetta una trasmissione continua dei messaggi.
Nel funzionamento da destinatario il modulo invia comunque il byte nullo per separare i messaggi, questo implica che se il messaggio è corretto si avranno un totale di 16 bit posti a zero (8 a zero per segnalare il corretto ricevimento del messaggio e 8 bit di separazione tra i messaggi).

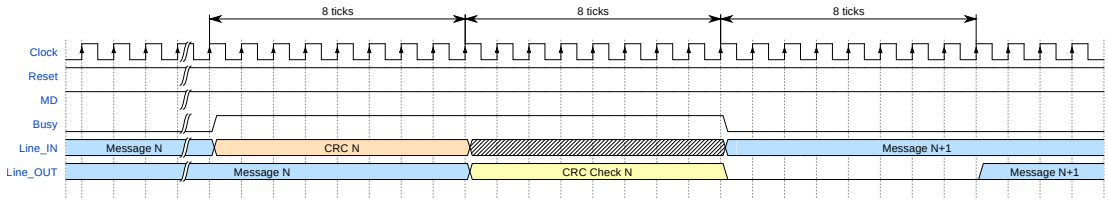


Figura 7: Ricezione con architettura NO_PRELOAD

In questa architettura l'uscita Q rimane attiva per un numero di cicli di clock doppio rispetto a U per aggiungere il byte nullo al termine dell'invio del messaggio.
Con questa architettura inoltre i messaggi non si sovrappongono mai nel tempo, i bit in uscita sono sempre riferiti all'ultimo dato posto all'ingresso; pertanto l'utilizzatore deve avere memoria sufficiente per lavorare su un solo messaggio.

2. Architettura PRELOAD:
   Non ci sono separazioni tra i messaggi inviati in modo da velocizzare la spedizione

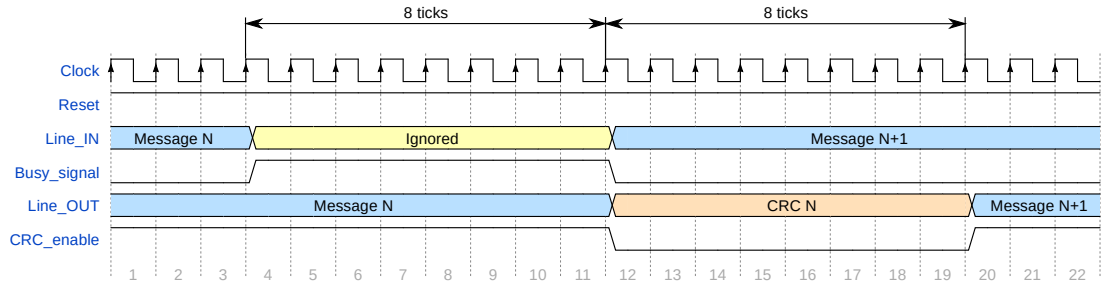


Figura 8: Trasmissione con architettura PRELOAD

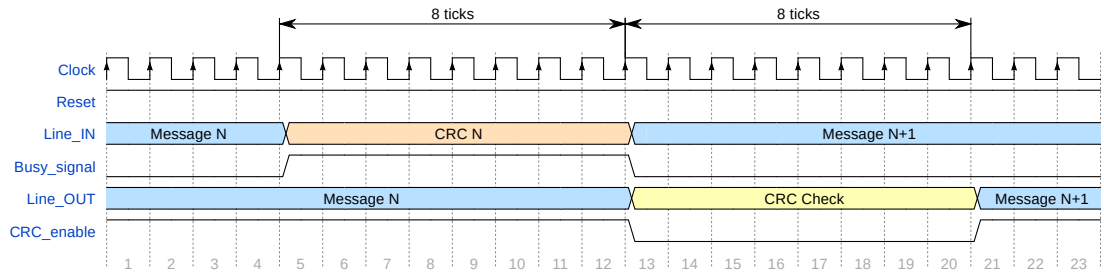


Figura 9: Ricezione con architettura PRELOAD

creando un bit stream continuo.
Le uscite si attivano una dopo l'altra, prima Q poi U per il numero di clock impostato durante la compilazione del modulo.

Questa architettura è stata pensata per l'utilizzo del modulo in dispositivi con ricevitore avente clock sincrono rispetto al trasmettitore.

I segnali di uscita del modulo determinano:

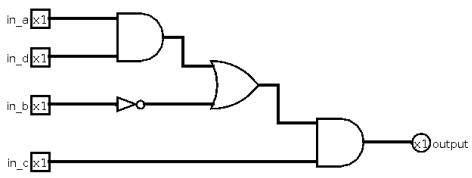
- quando si commuta l'uscita tra input e uscita del CRC_LOGIC;
- quando si ignora l'input perché è in corso la trasmissione degli F bit del CRC;
- quando si disabilita la logica DO_XOR di CRC_LOGIC per inviare i bit del codice.

Nel nostro modulo sono stati utilizzati questi segnali per poter temporizzare l'invio del messaggio, il calcolo del CRC e l'invio del CRC. In pratica i moduli DO_XOR del CRC Logic devono essere spenti per 8 cicli in modo da inviare il CRC contiguamente al messaggio in uscita.

MULTIPLEXER

Scopo del multiplexer è mandare all'output la copia dei bit del messaggio oppure l'esito del calcolo del CRC al tempo oppurtuno, per questo il suo segnale di controllo è crc_enable negato.

MD_ENABLE



CD\AB	00	01	11	10
00				
01	1			1
11	1		1	1
10				

Figura 10: Funzione logica di DO_XOR

MD_ENABLE ha lo scopo di ignorare l'input del modulo quando questo può corrompere il calcolo del CRC. Questa scelta non obbliga così l'utente a inserire necessariamente degli zeri quando il modulo ha la segnalazione "busy" attiva, rendendolo così più robusto. La sintesi è stata ottenuta minimizzando la funzione logica tramite una mappa di Karnaugh a 4 variabili.

Testing del modulo

Il testing del modulo è affidato a un testbench scritto anch'esso in VHDL. Tale testbench contiene quattro architetture per testare il modulo sia in modo Mittente sia Destinatario, con architettura preload o no_preload. Per ogni tipo di test è stato predisposto un file di configurazione in modo che vengano istanziati i componenti con le corrette architetture.

Tutte le architetture del testbench seguono la seguente logica:

1. istanziano un CRC_module;
2. leggono dal file di input una riga;
3. presentano nell'ordine corretto i bit al modulo (un bit per colpo di clock);
4. attendono il tempo necessario affinché il modulo restituisca l'output;
5. lo confrontano con quanto trovano nel file di output precedentemente generato;

La differenza tra il segnale atteso dal modulo e quello effettivo si può controllare tramite un segnale del testbench chiamato "error".

I dati di input per il test (e il relativo output con calcolato il CRC) sono pseudo-casuali, in modo da non favorire nessuna particolare sequenza. Poiché la costruzione di questi file sarebbe stata manualmente lenta e certamente non priva di errori, si è preferito scrivere un programma in Python per la sua automazione.

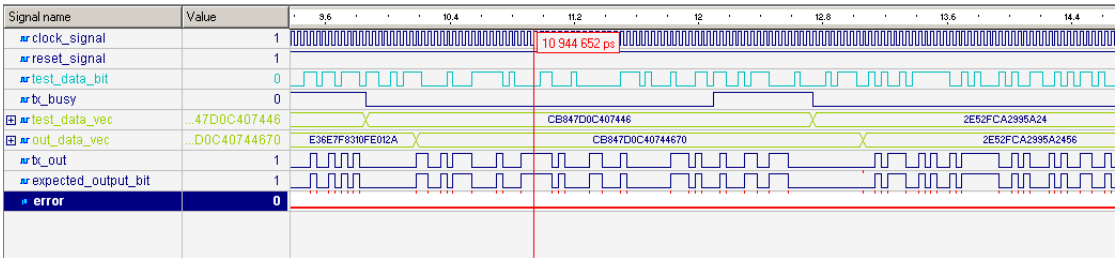


Figura 11: Testbench in esecuzione su Active-HDL® con architettura no_preload

Implementazione dell'algoritmo CRC

Progettare un circuito digitale che implementi l'algoritmo di Cyclic Redundancy Check (CRC), sia per il *mittent*e che per il *destinatario*, secondo le specifiche sotto indicate. Tale algoritmo è un potente metodo di controllo che sfrutta l'idea di ridondanza; ad una sequenza dati di M bit viene aggiunta (dal mittente) una sequenza di F bit ridondanti (Frame Control Sequence FCS) in modo che il messaggio trasmesso, su M+F bit, risulti divisibile per un divisore prefissato, detto polinomio CRC. Il destinatario, tramite una divisione per lo stesso polinomio CRC utilizzato dal mittente, può riconoscere la correttezza dei dati ricevuti.

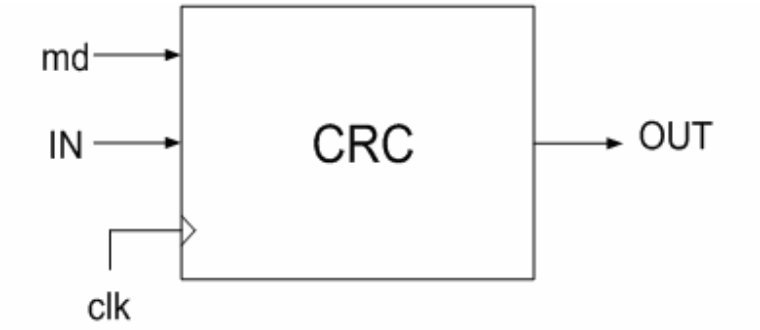
Messaggio M=56 bit

Frame Control Sequence F=8 bit

Polinomio CRC di ordine ordine 9: x^8+x^4+x^2+1

(La rappresentazione binaria del polinomio sarà quindi 100010101)

Messaggio trasmesso M+F = 64 bit



Tramite il segnale md si imposta se si vuole utilizzare il circuito come mittente o come destinatario. Nel funzionamento come destinatario gli ultimi F valori di OUT segnalano la correttezza del messaggio ricevuto

La relazione finale del progetto deve contenere:

- Introduzione (descrizione algoritmo, possibili applicazioni, possibili architetture, etc.)
- Descrizione dell'architettura selezionata per la realizzazione (diagramma a blocchi, ingressi/uscite, etc.)
- Codice VHDL (con commenti dettagliati)
- Testbench per la verifica
- Conclusioni