Politecnico di Torino

DIPARTIMENTO DI ELETTRONICA E TELECOMUNICAZIONI
Corso di Laurea Magistrale in Ingegneria Elettronica

Sistemi Digitali Integrati
LABORATORIO 3

Progettazione di un calcolatore CRC

Prof. Massimo Ruo Roch

Laboratorio LED3

Author:
Bricco Letizia (s328719)

A.A. 2023/2024

Indice

Sommario

L'obiettivo del progetto è implementare in hardware e testare un blocco in grado di calcolare il CRC (*Cyclic Redundancy Check*) di uno stream di dati organizzato in parole di 16 bit.

Il circuito è stato realizzato rispettando rigorose specifiche di progetto, che impongono di avvalersi dello standard CRC-16-CCITT e di processare i 16 bit di ciascuna parola a partire dall'MSB, ricercando negli archivi online la documentazione necessaria.

Le connessioni I/O del blocco comunicano con un'interfaccia a registri che, a sua volta, è collegata allo slave SPI precedentemente progettato. Opportune operazioni di scrittura e lettura nei registri consentono di inviare al *processing element* le parole su cui effettuare il calcolo del CRC e di ricevere il risultato generato.

La *control unit* della IP è stata progettata come una macchina a stati di Moore: in tal modo, il timing dei controlli risulta deterministico in quanto dipende unicamente dal clock di sistema.

La descrizione dell'architettura è stata implementata in VHDL in maniera gerarchica e il corretto comportamento è stato verificato sia mediante *testbench* sia su piattaforma fisica.

La simulazione è stata condotta utilizzando l'ambiente di sviluppo basato su Quartus-ModelSim e, per testare il blocco in maniera il più possibile completa, è stata automatizzata mediante uno script in linguaggio C++.

Per il test fisico, invece, ci si è avvalsi della scheda VirtLAB e, in particolare, il micro-controllore master STM32L496 è stato utilizzato come master SPI per la trasmissione e la ricezione dei dati.

1 Introduzione

Il **controllo di ridondanza ciclico** o ***Cyclic Redundancy Check*** (CRC) è un metodo per la generazione di una stringa di controllo a partire da uno stream di dati di lunghezza arbitraria, utilizzato per individuare eventuali errori nella trasmissione di un messaggio su una linea. È di particolare utilità quando si ha a che fare con linee particolarmente rumorose poiché aiuta a comprendere se l'informazione è stata corrotta.

La sua implementazione si basa sui principi dell'aritmetica modulare; in virtù delle proprietà matematiche delle divisioni con resto, il cui risultato si ripete ciclicamente con una ciclicità pari al valore del divisore, non è affidabile per verificare la completa correttezza dei dati trasmessi.

1.1 Richiami matematici sull'implementazione del CRC

Si consideri una parola B codificata su n bit, $n \in \mathbb{N}\backslash\{0\}$; secondo l'algebra dei campi finiti, essa può essere associata ad un polinomio di grado $n-1$, che denotiamo con $x \mapsto b(x)$, appartenente al campo di Galois[1] $\mathcal{G}(2^n)$.

Si definisca, inoltre, un polinomio binario detto **polinomio generatore** $x \mapsto g(x) \in \mathcal{G}\left(2^{m+1}\right)$, $m \in \mathbb{N} \setminus \{0\}$; secondo lo standard CRC-16-CCITT, ad esempio, tale polinomio è

$$g(x) = x^{16} + x^{12} + x^5 + 1 \equiv (1000100000100001)_2 = (11021)_{16} \in \mathcal{G}\left(2^{17}\right). \tag{1.1}$$

Il codice CRC del numero B è dato dal resto della divisione

$$q(x) = \frac{x^m b(x)}{g(x)}, \tag{1.2}$$

[1] In algebra, si definisce campo di Galois, o campo binario, di ordine 2^n, e lo si denota con $\mathcal{G}(2^n)$, l'insieme dei polinomi binari di grado $(n-1)$. Ogni polinomio $p \in \mathcal{G}(2^n)$ è equivalente a una stringa di n bit in cui l'i-esimo bit rappresenta il coefficiente del polinomio nella stessa posizione.

In altre parole, si tratta del polinomio $x \mapsto c(x) \in \mathcal{G}(2^m)$ tale che

$$x^m b(x) = g(x)q(x) + c(x). \tag{1.3}$$

L'utilità dell'algoritmo risiede nella possibilità di utilizzare il CRC calcolato per controllare la correttezza del dato inviato su una linea. Supponiamo di trasmettere il messaggio

$$m(x) = x^m b(x) - c(x) \equiv g(x)q(x), \tag{1.4}$$

dove l'uguaglianza dell'ultimo passaggio è una conseguenza immediata dell'Equazione (1.3).

Osservando l'equazione appena scritta, è semplice notare che condizione necessaria (ma non sufficiente) affinché il messaggio sia corretto è che il resto della divisione dello stesso per il polinomio generatore non produca resto. È, tuttavia, possibile che errori di trasmissione nulli generino un messaggio errato che ha comunque resto nullo. La probabilità di tale evento è tanto più bassa quanto più è alto il grado del polinomio.

L'aritmetica utilizzata per la computazione del CRC prevede di effettuare calcoli con numeri binari senza tenere conto dei riporti: questo implica che l'addizione e la sottrazione modulo 2 sono entrambe equivalenti ad un *bitwise xor*:

A	B	$A+B$	$A-B$	$(A+B) \mod 2$	$(A-B) \mod 2$	$A \oplus B$
0	0	0	0	0	0	0
0	1	1	-1	1	1	1
1	0	1	1	1	1	1
1	1	0	0	0	0	0

Analogamente, moltiplicazioni e divisioni possono essere implementate dapprima con uno *shift left* di un numero di bit pari alla lunghezza del divisore, e poi mettendo il risultato dello shift in *bitwise xor* con il divisore stesso.

Con queste premesse, è possibile scrivere lo pseudocodice dell'algoritmo utilizzando come riferimento lo standard CRC-16-CCITT adottato nel progetto:

```
g = 10001000000100001 (17 bit)
b = xxxxxxxxxxxxxxxx (16 bit)
p = b << 16 = xxxxxxxxxxxxxxxx0000000000000000 (32 bit)
q = p / g
crc = p % g
m = p - crc = (q * g) + crc - crc = q * g
check = m % g
if (check != 0) then
    "messaggio errato"
else
    "messaggio corretto, a meno di errori multipli"
```

2 Individuazione dello standard corretto

Parte integrante del progetto è rappresentata dall'individuazione delle informazioni sullo standard da utilizzare per il calcolo del CRC. In particolare, le specifiche impongono di

1. utilizzare l'algoritmo **CRC-16-CCITT**;
2. utilizzare i bit del messaggio a partire dal **MSB**.

La sigla CCITT è l'acronimo di ***Consultative Committee International on Telephones and Telegraphy***: si tratta di un'organizzazione fondata a Parigi nel 1865 per la gestione delle reti di telecomunicazioni che, proprio in quegli anni, stavano vedendo la luce.

Dal 1947, la CCITT è diventata l'agenzia specializzata dell'ONU per le tegnologie ICT con il nome di ITU (*International Telecommunication Union*) [2]; ad oggi, l'organizzazione conta 193 Stati membri e si occupa, tra le altre cose, di mettere a punto gli standard tecnici per l'utilizzo e lo sviluppo di tecnologie nel mondo delle telecomunicazioni.

Nonostante l'esistenza di questo comitato regolatore, sul sito [5] in letteratura e nel mondo dell'industria esistono moltissime versioni dell'algoritmo per il calcolo del CRC, come testimonia il catalogo visualizzabile al link [3]. I vari metodi, differiscono, principalmente, per:

- il grado e il valore del polinomio generatore;
- il valore di inizializzazione del calcolatore di CRC;
- l'ordine di utilizzo dei bit del messaggio (*MSB first* o *LSB first*);
- la riflessione dell'uscita del calcolatore di CRC (*reflected* o *unreflected implementations*);
- la riflessione del polinomio generatore (*reversed poly*).

Sulla base della documentazione letta e riportata in fondo a questo documento, l'implementazione scelta è **CRC-16-CCITT/XMODEM**, caratterizzata dei seguenti parametri:

1. *Polinomio generatore*: $poly = 0x1021$;
2. *Ampiezza polinomio generatore* (troncato): 16 bit;
3. *Valore di inizializzazione*: init $= 0x0000$;
4. *Riflessione del messaggio prima dell'implementazione*: refin = false (algoritmo *MSB-first*);
5. *Riflessione del risultato*: refin = false (*unreflected implementation*).

La scelta effettuata risulta coerente con le specifiche di progetto in virtù di quanto letto nella descrizione degli algoritmi elencati nel catalogo [3]. In particolare:

- Alla voce dell'algoritmo CRC-16/IBM-3740, anche noto come CRC-16-CCITT/FALSE, viene sottolineato che tale algoritmo è spesso *erroneamente* identificato con lo standard CRC-16-CCITT;
- Al contrario, la sigla CRC-16-CCITT si riferisce, tipicamente, alla versione LSB-first della ITU-T Recommendation V.41 [4], implementata dall'algoritmo CRC-16-CCITT/KERMIT; l'algoritmo CRC-16-CCITT/XMODEM, invece, è la sua "*MSB counterpart*", i.e., si tratta dello stesso algoritmo implementato senza riflessione del messaggio di ingresso.

3 Descrizione generale e funzionale

3.1 Specifiche e protocollo

Le specifiche di progetto imposte per la trasmissione dei dati al processing element e la ricezione dei risultati prodotti dallo stesso sono le seguenti:

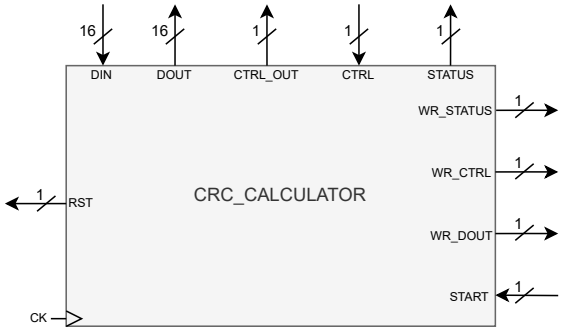


Figura 3.1: CRC calculator

- Le connessioni I/O del calcolatore di CRC devono essere collegate ad un'opportuna interfaccia a registri, in grado di interagire sia con il PE sia con lo slave SPI precedentemente progettato;
- L'indirizzo 0 dell'interfaccia a registri identifica il **Data In Register**: lo slave SPI scrive al suo interno le parole di 16 bit da utilizzare per il calcolo del CRC, mentre il processing element preleva i dati in esso contenuti per poi procedere alla loro elaborazione;
- L'indirizzo 1 dell'interfaccia a registri identifica il **CRC Out Register**: il calcolatore di CRC scrive al suo interno il valore corrente del CRC, mentre lo slave SPI legge il suo contenuto per inviare il risultato al master;
- L'indirizzo 2 dell'interfaccia a registri identifica il **Control Register**: se lo slave SPI scrive un '1' nel LSB, il calcolatore di CRC viene resettato, i.e., il valore del CRC viene riportato al valore di inizializzazione (0);
- L'indirizzo 3 dell'interfaccia a registri identifica lo **Status register**: se il calcolatore di CRC scrive 1 nel LSB, significa che ha terminato il *processing* di un dato ed è pronto a ricevere una nuova parola di 16 bit (stato *free*); viceversa, se scrive uno '0', significa che è impegnata nello svolgimento del calcolo (stato *busy*).

Il protocollo adottato impone importanti vincoli sulla struttura dell'interfaccia a registri e sulle operazioni effettuabili su di essa. Per una descrizione più dettagliata, si rimanda al paragrafo 4.

3.2 Connessioni I/O

Lo schema delle porte di ingresso e di uscita della IP progettata, con i relativi parallelismi, è riportato in Figura 3.1. I segnali utilizzati sono i seguenti:

- **CK** (porta di ingresso, parallelismo 1 bit): clock di sistema; gli ingressi di tutti gli elementi sequenziali presenti nel circuito sono *positive edge triggered*, i.e., sono sensibili al fronte di salita di CK;
- **RST** (porta di ingresso, parallelismo 1 bit): segnale di reset asincrono che effettua il reset completo del calcolatore di CRC, riportando ai valori di inizializzzazione il contenuto dei registri agli indirizzi 1, 2 e 3 dell'interfaccia;
- **START** (porta di ingresso, parallelismo 1 bit): segnale utilizzato per far partire il processing element; viene automaticamente generato nell'interfaccia a registri ogni volta che lo slave SPI completa una transazione di scrittura all'indirizzo 0;

- **DIN** (porta di ingresso, parallelismo 16 bit): bus dati utilizzato per la ricezione delle parole su cui calcolare il CRC dall'indirizzo 0 dell'interfaccia a registri;
- **DOUT** (porta di uscita, parallelismo 16 bit): bus dati utilizzato per inviare all'indirizzo 1 dell'interfaccia a registri il risultato corrente del calcolo del CRC;
- **CTRL** (porta di ingresso, parallelismo 1 bit): linea utilizzata per la ricezione del segnale di reset del calcolatore di CRC dall'indirizzo 2 dell'interfaccia a registri;
- **CTRL_OUT** (porta di uscita, parallelismo 1 bit): linea utilizzata per scrivere uno '0' nel LSB dell'indirizzo 2 dell'interfaccia a registri;
- **STATUS** (porta di uscita, parallelismo 1 bit): linea utilizzata per inviare all'indirizzo 3 dell'interfaccia a registri lo stato del calcolatore di CRC ('1' se *free*, '0' se *busy*);
- **WR_DOUT** (porta di uscita, parallelismo 1 bit): segnale che abilita la scrittura nel CRC Out Register (indirizzo 1 dell'interfaccia a registri);
- **WR_CTRL** (porta di uscita, parallelismo 1 bit): segnale che abilita la scrittura nel Control Register (indirizzo 2 dell'interfaccia a registri);
- **WR_STATUS** (porta di uscita, parallelismo 1 bit): segnale che abilita la scrittura nello Status Register (indirizzo 3 dell'interfaccia a registri).

3.3 Progettazione di Execution Unit e Control Unit

L'architettura del calcolatore di CRC progettato è mostrata in Figura A.4 (Appendice A), mentre la struttura della control unit può essere osservata nelle Figure B.1 e B.2 (Appendice B).

Come è possibile osservare, la topologia del datapath è piuttosto semplice ed è stata derivata tenendo conto del fatto che le specifiche richiedono di progettare un blocco in grado di calcolare il CRC di una *sequenza arbitraria* di parole di 16 bit.

Sappiamo che, per calcolare il CRC correttamente, è necessario appendere in coda al messaggio un numero di zeri pari al grado del generatore, in questo caso 16; quando il calcolatore riceve un dato dal Data In Register, tuttavia, non può sapere se la parola in esame è o meno l'ultima del messaggio da codificare.

Per ogni dato processato, pertanto, è necessario prevedere la possibilità di mantenere in memoria due valori:

- Il **CRC parziale**, ottenuto senza appendere gli zeri in coda all'ultima parola ricevuta, da utilizzare nel caso in cui venga scritta una nuova parola nel Data In Register;
- il **CRC finale**, ottenuto appendendo gli zeri in coda all'ultima parola ricevuta, da inviare al CRC Out Register affinché lo slave SPI possa leggerlo e trasmetterlo al master.

3.3.1 Execution Unit

Sulla base delle considerazioni del paragrafo precedente, è ora possibile analizzare nel dettaglio l'architettura. Il nucleo del calcolatore CRC è un particolare registro a scorrimento che prende il nome di **Linear Feedback Shift Registers** (LFSR). Si tratta di un modulo costituito da un numero di D-flip-flop pari al grado del polinomio generatore, con porte XOR poste davanti ai flip-flop corrispondenti ai coefficienti non nulli del polinomio stesso (0, 5, 12 nel caso del CRC-16-CCITT).

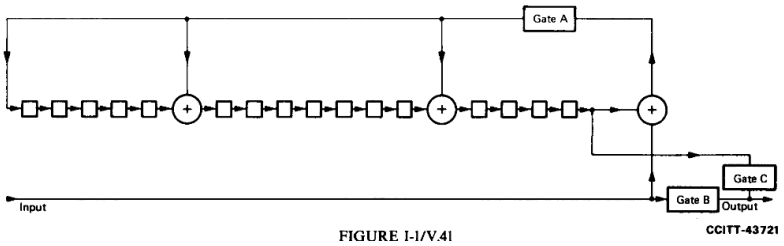


FIGURE I-1/V.41

Figura 3.2: LFSR per il calcolo del CRC-16-CCITT secondo la ITU-T Recommendation V.41 ([4], Figura I-1/V.41, pag. 9).

I dati vengono inviati serialmente da sinistra, i.e., il MSB corrisponde al D-flip-flop situato più a destra; tutti i flip-flop sono inoltre inizializzati a zero, i.e., il valore del CRC subito dopo il reset è zero.

L'implementazione messa a punto dalla ITU-T Recommendation V.41 è osservabile in Figura 3.2, mentre la rappresentazione del blocco utilizzato all'interno della IP progettata è riportata in Figura A.3 (Appendice A).

Per quanto riguarda gli altri blocchi del datapath, è importante sottolineare quanto segue:

- L'ingresso del LFSR è seriale, ma il dato arriva sottoforma di una stringa di 16 bit; pertanto, prima di calcolarne il CRC è necessario serializzarlo mediante il PISO PISO_DIN;
- Il PISO non è collegato direttamente a DIN, bensì all'uscita di un multiplexer a due vie, il cui segnale di selezione è s_DIN (2 bit):
  1. Se s_DIN = 0, la parola inviata a PISO_DIN è il CRC parziale CRC_PRTL calcolato con i dati precedenti, senza appendere gli zeri in coda;
  2. Se s_DIN = 1, la parola inviata a PISO_DIN è il dato DIN ricevuto dal Data In Register dell'interfaccia a registri;
  3. Se s_DIN = 2, la parola inviata a PISO_DIN è la stringa di 16 zeri da appendere in coda al CRC parziale per ottenere il valore corretto, da inviare al CRC Out Register.
- L'uscita del LFSR è collegata all'ingresso di due registri, REG_PARTIAL e REG_FINAL, in cui vengono caricati, rispettivamente, il CRC parziale (CRC_PRTL, no zeri in coda) e il CRC finale (CRC_FNL, zeri in coda): il primo verrà usato per il processing di un eventuale nuovo dato, mentre il secondo viene inviato al CRC Out Register;
- L'uscita di REG_FINAL è uno dei due ingressi di un multiplexer a due vie denotato con MUX_DOUT, con segnale di selezione s_DOUT, che sceglie il valore da inviare al CRC Out Register:
  1. Se s_DIN = 0, il dato inviato sulla linea DOUT è una stringa di 16 zeri; questo avviene solo in fase di reset della macchina;
  2. Se s_DIN = 1, il dato inviato sulla linea DOUT è CRC_FNL;
  3. L'architettura comprende anche un contatore a 4 bit, COUNT_16, che viene incrementato di 1 ad ogni colpo di CK per effettuare il numero di shift corretti nel LFSR.

3.3.2 Control Unit

Per quanto riguarda la struttura dell'unità di controllo, è importante sottolineare quanto segue:

- È previsto uno stato di **reset complessivo** della macchina, durante il quale vengono resettati
  1. il CRC Out Register (DOUT = 0);
  2. il Control Register, grazie all'invio del segnale CTRL_OUT = '0' da parte del calcolatore di CRC;
  3. lo Status Register (STATUS = '1');
  4. il LFSR, che viene riportato al valore di inizializzazione previsto dallo standard adottato (CRC = 0);
  5. i registri REG_PARTIAL e REG_FINAL;
  6. il contatore.

  Dopo lo stato di reset, il calcolatore entra in uno stato di IDLE, in cui i valori di tutti i segnali sono uguali a quelli di default.

- Esistono altri due stati in cui il calcolatore viene parzialmente resettato:
  1. EXTERNAL_RESET: si entra in tale stato quando lo slave SPI scrive un '1' nel Control Register (CTRL = '1'); prevede di resettare il CRC Out Register, il LFSR, REG_PARTIAL e REG_FINAL; successivamente si entra nello stato di IDLE_RESET, durante il quale il Control Register viene resettato per evitare che la macchina si resetti una seconda volta;
  2. INTERNAL_RESET: è uno stato di reset interno preliminare allo svolgimento dei calcoli, previsto tutte le volte che il calcolatore riceve un nuovo START e CTRL = '0'; prevede di resettare soltanto il LFSR, che viene riportato al valore di inizializzazione; il motivo della presenza di tale stato sarà più chiaro dopo la descrizione del resto della macchina a stati.

- La macchina è sensibile al reset esterno solo quando si trova nello stato di IDLE; se l'SPI scrive un '1' nel Control Register quando il calcolatore di CRC si trova in un altro stato, il segnale non verrà rilevato fino a quando la macchina non tornerà nello stato di IDLE;

- Subito dopo la ricezione dello START, si entra in un gruppo di stati, colorati in giallo nella ASM chart, durante i quali il CRC parziale del ciclo precedente viene caricato in PISO_DIN; si noti che tale operazione è inutile se la macchina è appena stata resettata perché CRC_PRTL = 0; tuttavia, siccome $f_{sck} \ll f_{ck}$ e l'operazione richiede, nel complesso, circa 50 colpi di CK, il costo in termini di tempo risulta del tutto irrisorio; in questa fase, lo stato del calcolatore di CRC è ancora *free* (STATUS = '1');

- Dopo il caricamento di CRC_PRTL, sono presenti due gruppi di stati, colorati in verde e azzurro nell'ASM chart, in cui avviene il caricamento dapprima di DIN, e poi dei 16 zeri, nel LFSR. Nello stato LOAD_DIN lo stato del calcolatore diventa *busy*, mentre in LOAD_FINALE ritorna *free*; l'uscita del LFSR viene caricata
  1. in REG_PRTL al termine del caricamento di DIN (stato LOAD_CRC_DIN);
  2. in REG_FNL al termine dell'*append* degli zeri (stato LOAD_CRC_FINALE).

- Infine, è previsto uno stato di DONE, durante il quale il CRC Out Register viene aggiornato con il valore corrente del CRC.
- L'inizio e la fine delle operazioni sono gestite mediante un semplice protocollo di **handshaking** tra i segnali di START e STATUS: infatti, una volta terminato il processing del dato, si resta nello stato di DONE finché START $= 0$.

4 Test del funzionamento

La descrizione in VHDL dell'IP e dei *component* necessari a realizzarla è riportata nell'appendice D.

Per testare il funzionamento del blocco, è stata creata una top entity (si veda l'appendice D.4.4) all'interno della quale le porte dello slave SPI e del calcolatore di CRC vengono opportunamente collegate a un'interfaccia a registri.

Tale blocco, denotato con il label REGISTER_INTERFACE, è dotato delle seguenti connessioni I/O:

- **D** (porta di ingresso, parallelismo 16 bit): bus dati usato durante le operazioni di scrittura per la memorizzazione di un dato inviato dallo slave SPI; va collegato alla porta DIN dell'SPI;
- **Q** (porta di uscita, parallelismo 16 bit): bus dati usato durante le operazioni di lettura per inviare il dato richiesto dallo slave SPI; va collegato alla porta DOUT dell'SPI;
- **ADDR** (porta di ingresso, numero intero codificabile su 8 bit): indirizzo della cella da scrivere o leggere; la conversione tra `integer` e `std_logic_vector` va effettuata nella top level mediante la definizione di un opportuno segnale;
- **WR** (porta di ingresso, parallelismo 1 bit): linea seriale su cui lo slave SPI invia l'impulso di scrittura; va collegato alla porta WR dell'SPI;
- **RD** (porta di ingresso, parallelismo 1 bit): linea seriale su cui lo slave SPI invia l'impulso di lettura; va collegato alla porta RD dell'SPI.
- **START** (porta di uscita, parallelismo 1 bit): impulso di un colpo di clock generato quando l'SPI scrive un dato all'indirizzo 0 dell'interfaccia; va collegato alla porta START del calcolatore di CRC;
- **Q_DIN** (porta di uscita, parallelismo 16 bit): bus dati utilizzato per inviare alla porta DIN del calcolatore di CRC il dato da processare, contenuto nel Data In Register;
- **D_DOUT** (porta di ingresso, parallelismo 16 bit): bus dati utilizzato dal calcolatore per inviare al CRC Out register il valore corrente del CRC mediante la porta DOUT;
- **EN_DOUT** (porta di ingresso, parallelismo 1 bit): linea seriale che, quando asserita, abilita la scrittura nel CRC Out Register da parte del calcolatore;
- **D_CTRL** (porta di ingresso, parallelismo 16 bit): bus dati utilizzato dal calcolatore per inviare un dato al Control Register durante lo stato di reset (per maggiori dettagli su questo punto, si vedano le descrizioni della Execution Unit e della Control Unit);
- **EN_CTRL** (porta di ingresso, parallelismo 1 bit): linea seriale che, quando asserita, abilita la scrittura nel Control Register da parte del calcolatore;

- **Q_CTRL** (porta di uscita, parallelismo 16 bit): bus dati utilizzato per inviare alla porta CTRL del calcolatore di CRC il segnale di reset esterno, contenuto nel Control Register;
- **D_STATUS** (porta di ingresso, parallelismo 16 bit): bus dati utilizzato dal calcolatore per inviare allo Status Register lo stato della macchina (*busy* o *free*);
- **EN_STATUS** (porta di ingresso, parallelismo 1 bit): linea seriale che, quando asserita, abilita la scrittura nello Status Register da parte del calcolatore.

4.1 Simulazione ModelSim automatizzata con C++

Le simulazioni sono state condotte in due step successivi mediante il software ModelSim:

1. Preliminarmente, è stata scritta una semplice testbench a scopo di *debug* (Appendice E.1); in tal modo, è possibile verificare "ad occhio" che il comportamento della IP sia corretto;
2. Una volta conclusa la prima fase della simulazione, il processo di simulazione è stato automatizzato mediante uno script in C++ basato sui principi della programmazione ad oggetti e una testbench con I/O da file (Appendici E.2 e E.3).

Non essendo presenti particolari limitazioni sulla dinamica dei dati da scrivere/leggere, ad eccezione del parallelismo di 16 bit, la simulazione automatizzata consiste nella generazione casuale di 100 numeri compresi tra 0 e 65535, ognuno dei quali viene scritto nel Data In Register e inviato all'IP progettata per il calcolo del CRC.

Per testare le funzionalità della macchina in maniera il più possibile completa, la simulazione è suddivisa in due parti:

Prima parte. Dopo ogni transazione di scrittura nel Data In Register, lo slave SPI scrive un '1' nel Control Register per resettare il calcolatore. In questo modo, i valori calcolati corrispondono ai CRC delle *singole* parole lette dal Data In Register;

Seconda parte. Lo slave SPI non richiede il reset della macchina tra una transazione e l'altra; i valori calcolati, quindi, corrispondono al CRC ottenuto appendendo l'ultimo dato ricevuto a tutti quelli ricevuti precedentemente.

Per automatizzare la simulazione sono state scritte tre classi: **Tools**, per l'impostazione del polinomio generatore e il calcolo del CRC, **Converter**, per la conversione dei numeri tra decimale e binario, e **Simulation**, per eseguire la simulazione e controllare la correttezza dei risultati.

La classe Simulation è dotata dei seguenti metodi pubblici:

1. `Simulation::generateCommands`, che genera un file all'interno del quale vengono scritti i comandi di scrittura per lo slave SPI;
2. `Simulation::generateReference_CRCsingle`, per calcolare il CRC delle singole parole contenute nel file dei comandi generato con `generateCommands`;
3. `Simulation::generateReference_CRClong`, per calcolare il CRC delle parole contenute nel file dei comandi, assumendo di appendere ogni nuovo dato a quelli già processati nelle iterazioni precedenti;
4. `Simulation::run`, per avviare la simulazione ModelSim da linea di comando mediante una chiamata del comando `system`, contenuto nella libreria `cstdlib`;

5. `Simulation::report`, per confrontare il file di output prodotto da ModelSim con quello di riferimento e stampare a video un feedback circa la correttezza o meno dei risultati; in caso di errori, vengono stampati i numeri delle righe del file dei comandi che hanno creato problemi.

L'esecuzione del programma di collaudo, in cui vengono richiamati tutti i metodi sopracitati, ha permesso di verificare la correttezza dei valori calcolati sia in modalità "single" (reset tra una parola e l'altra) sia in modalità "long".

4.2 Test su piattaforma fisica

Una volta constatata la correttezza dei risultati prodotti dalla testbench, il funzionamento dell'IP è stato verificato mediante la scheda VirtLAB.

Innanzitutto, sul MCU user è stato caricato il file eseguibile `virtlab-user-spi-tester.elf`, già usato per il test dello slave SPI, che configura la porta USB utente come seriale virtuale.

A questo punto, dopo aver caricato sulla FPGA il file di configurazione prodotto dal sintetizzatore (`fpga-user.rbf`), è possibile effettuare delle transazioni di scrittura e lettura inviando opportuni comandi al MCU tramite seriale.

Come si può vedere dall'ouput dell'emulatore di terminale seriale, il corretto funzionamento è verificato: infatti, il CRC calcolato dal blocco hardware corrisponde a quello atteso in ognuno dei casi considerati.

```
******************************
*   VirtLAB SPI tester v1.0   *
******************************

>w001206
Writing 1206 to register 00
>r01
Reading from register 01: 05d7
>w002203
Writing 2203 to register 00
>r01
Reading from register 01: 1488
>w000303
Writing 0303 to register 00
>r01
Reading from register 01: ba07
>w003601
Writing 3601 to register 00
>r01
Reading from register 01: 3e33
>w020001
Writing 0001 to register 02
>r02
Reading from register 02: 0000
>r03
Reading from register 03: 0001
>r01
Reading from register 01: 0000
>w00aaaa
Writing aaaa to register 00
>r01
Reading from register 01: e615
>
```

A Execution Unit

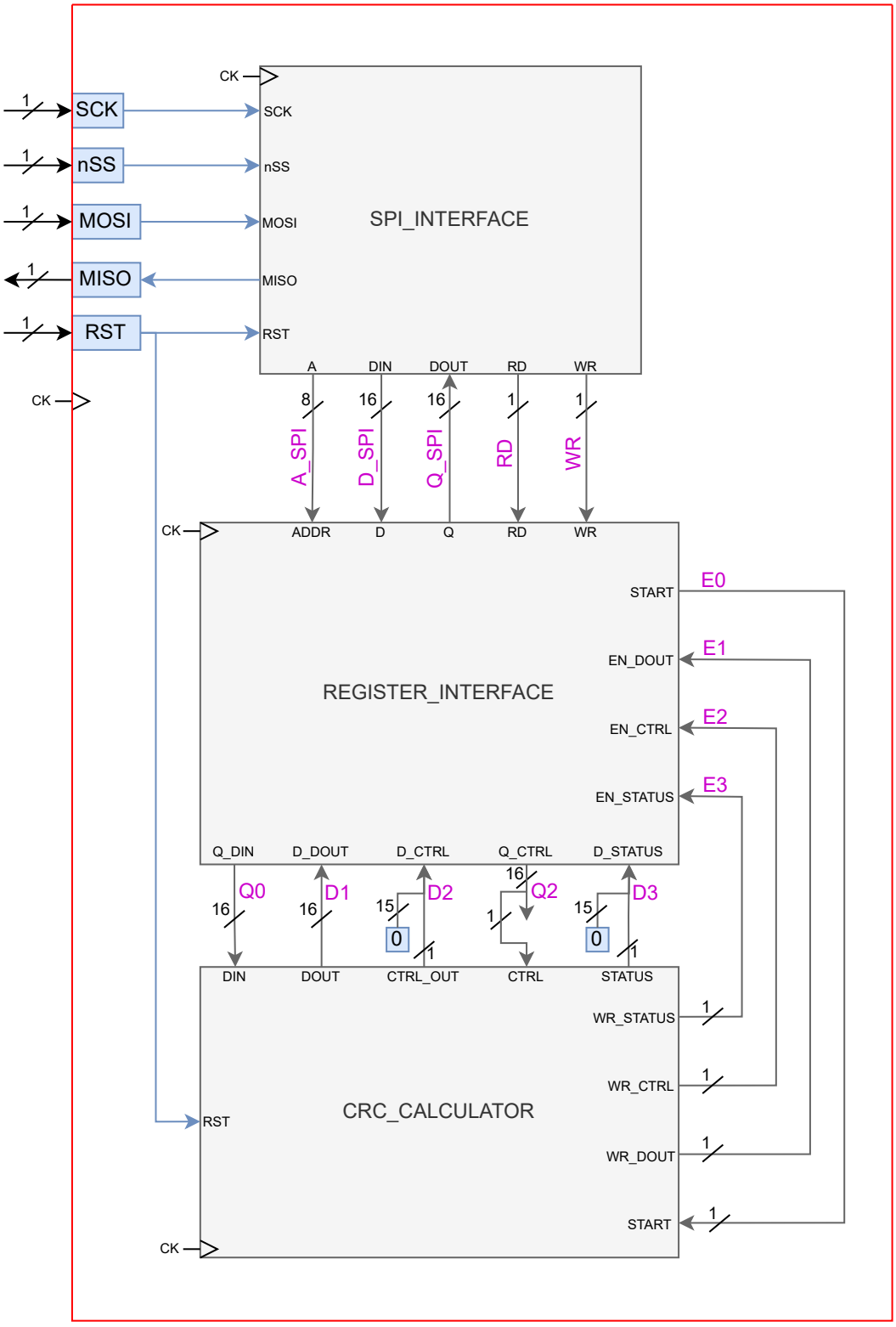


Figura A.1: Top level

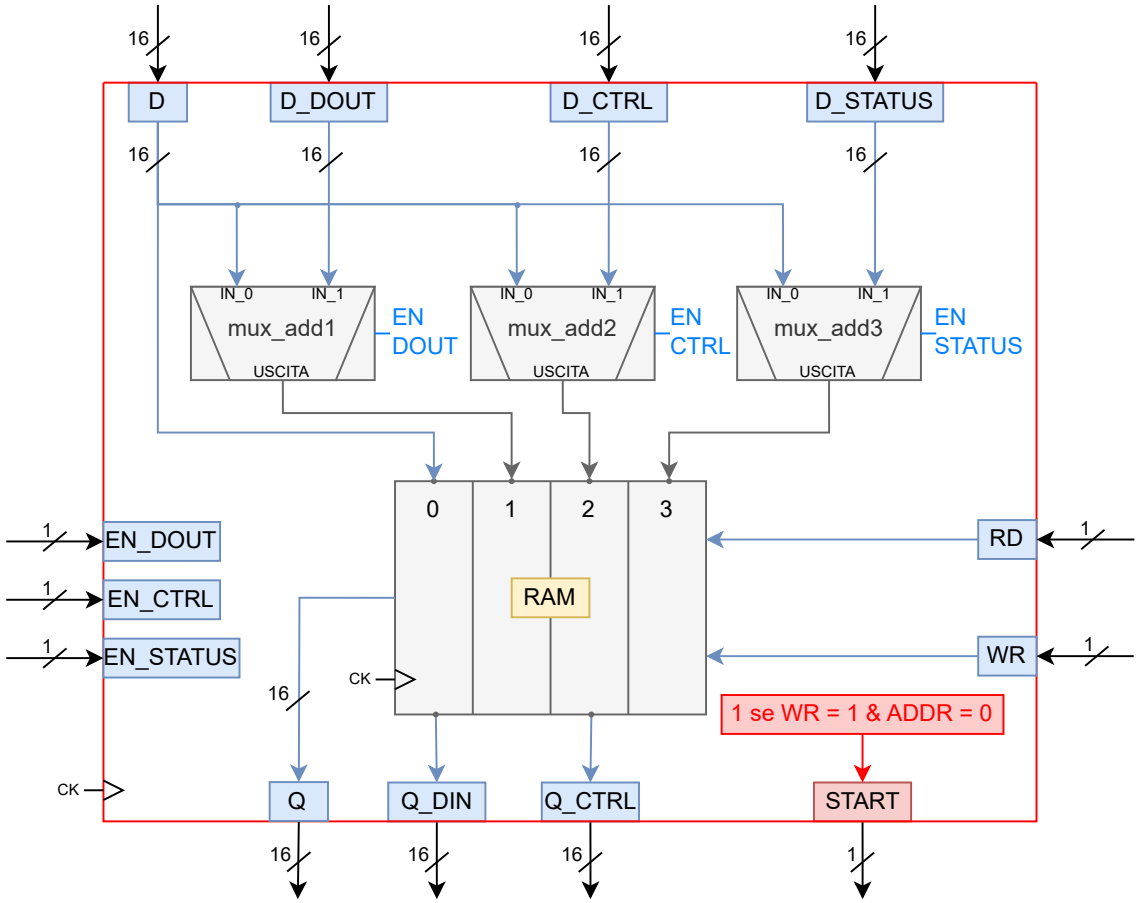


Figura A.2: Interfaccia a registri

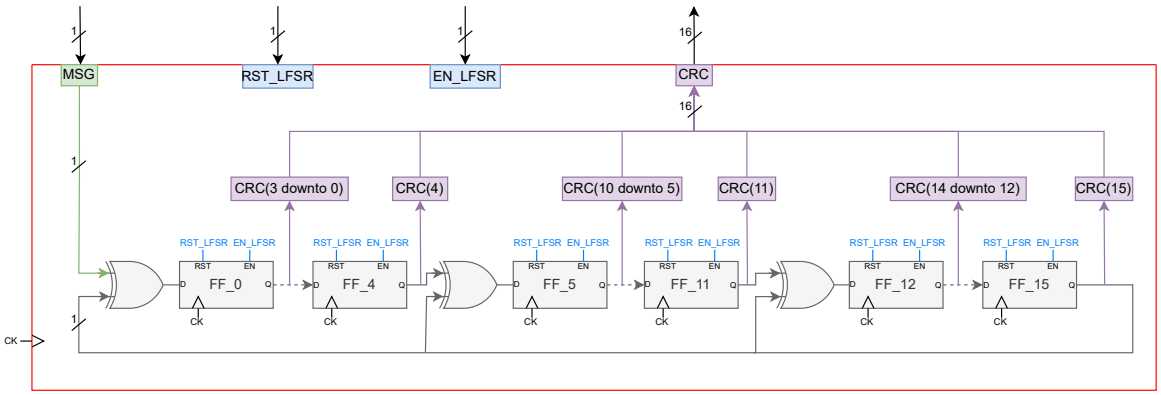


Figura A.3: Linear Feedback Shift Register

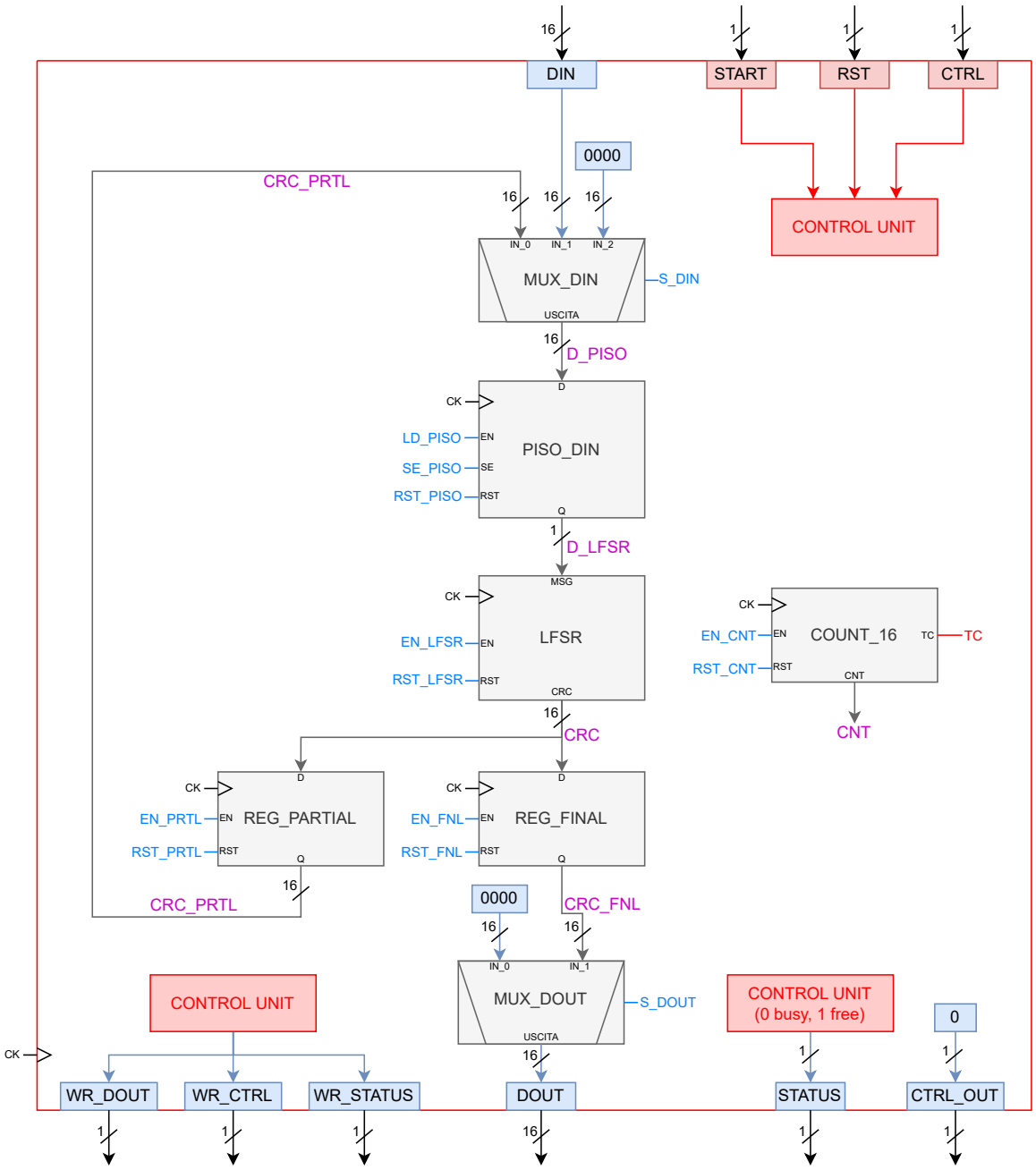


Figura A.4: Calcolatore di CRC

B Control Unit

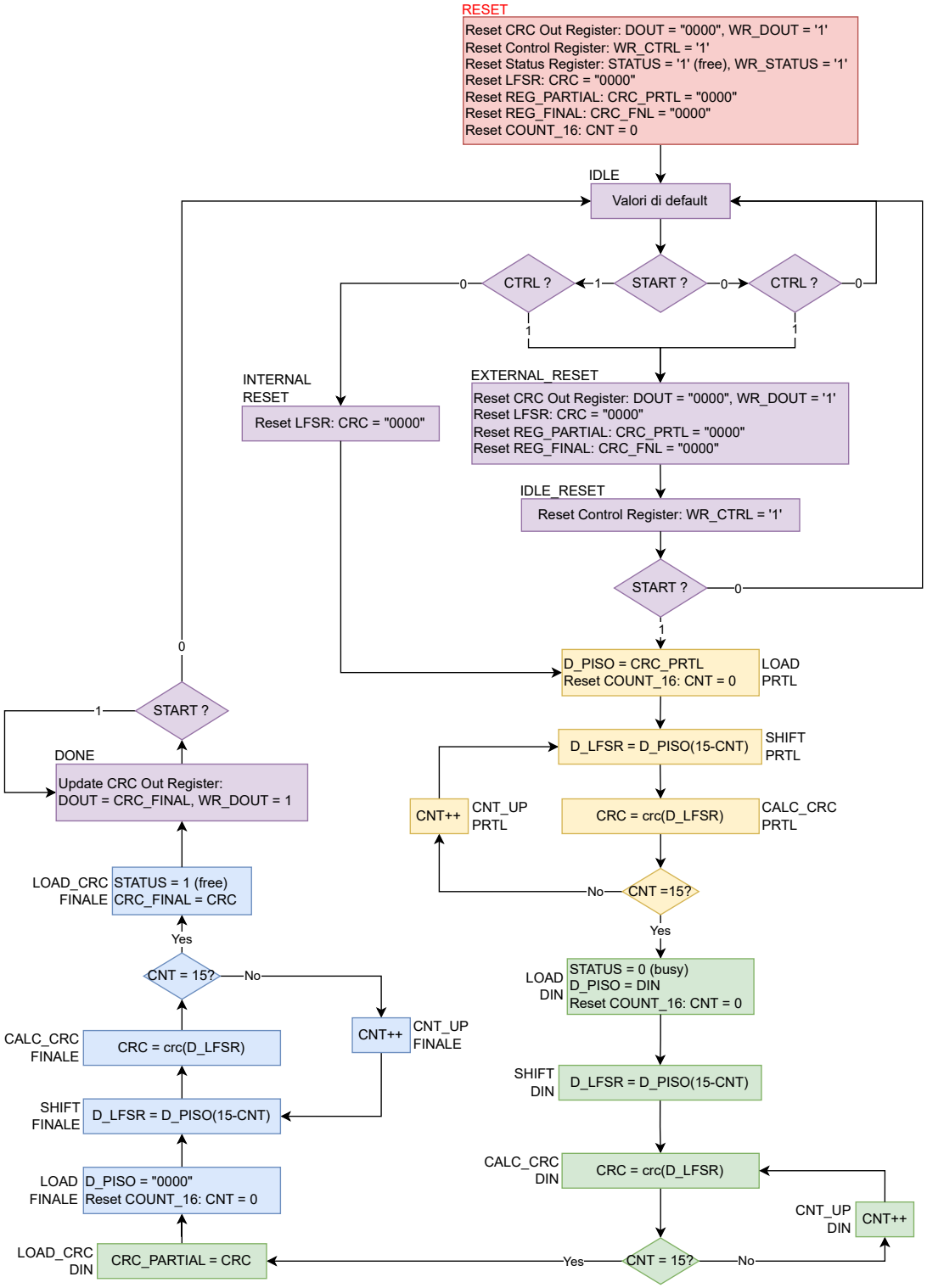


Figura B.1: ASM chart

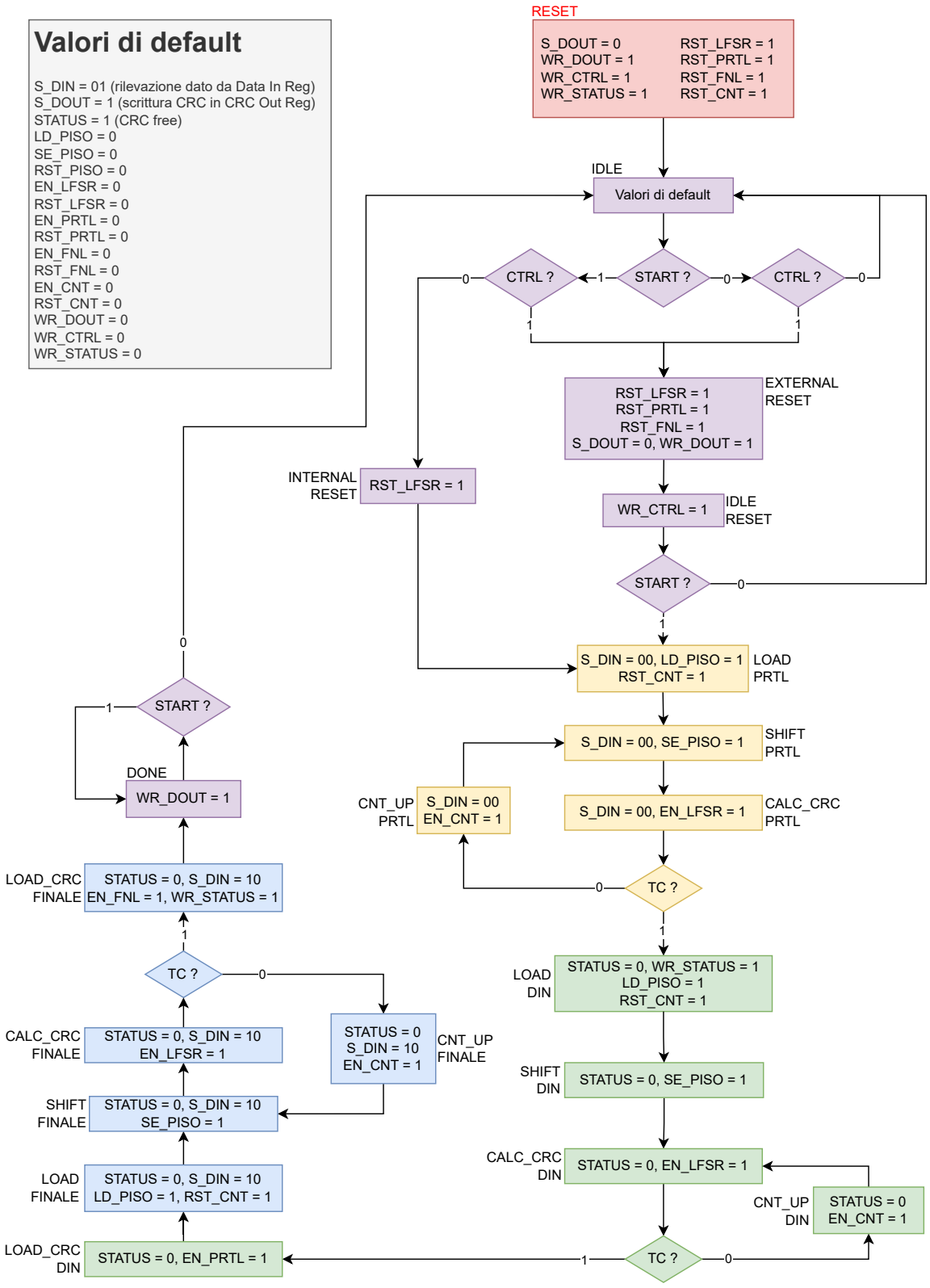
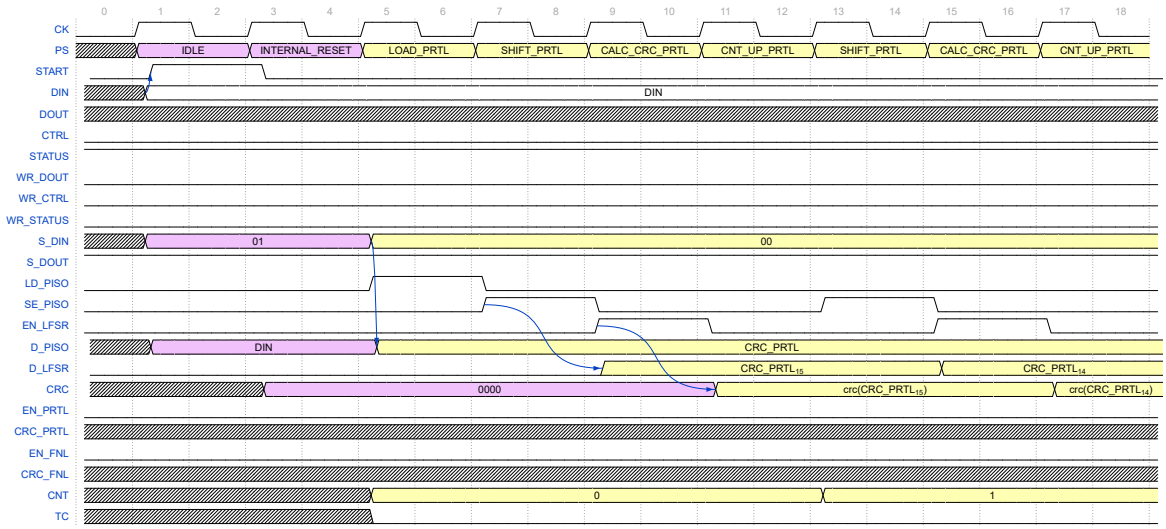
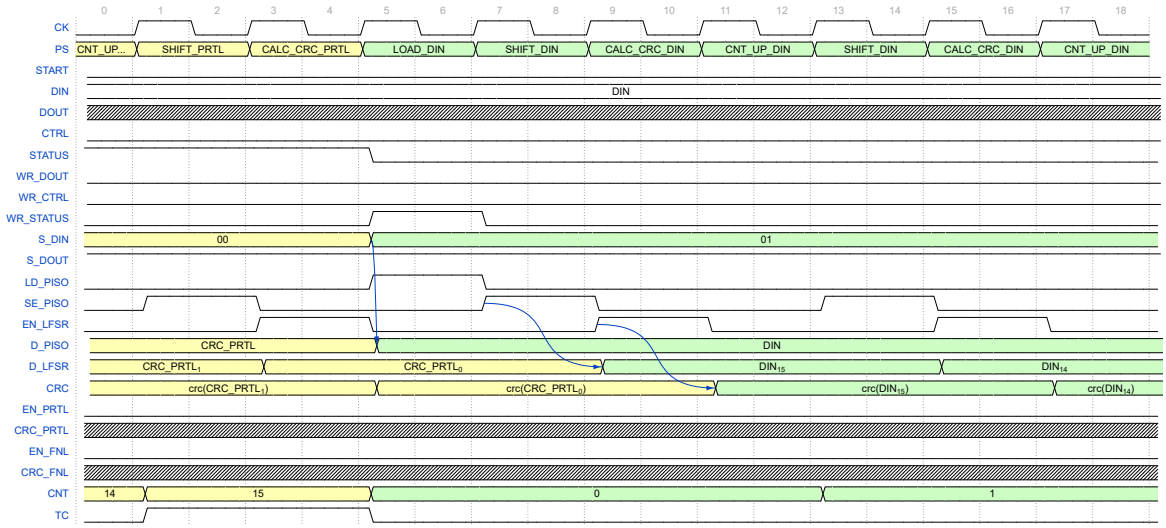


Figura B.2: Control ASM chart: per ogni stato, sono riportati i segnali di controllo il cui valore differisce dal valore di default.

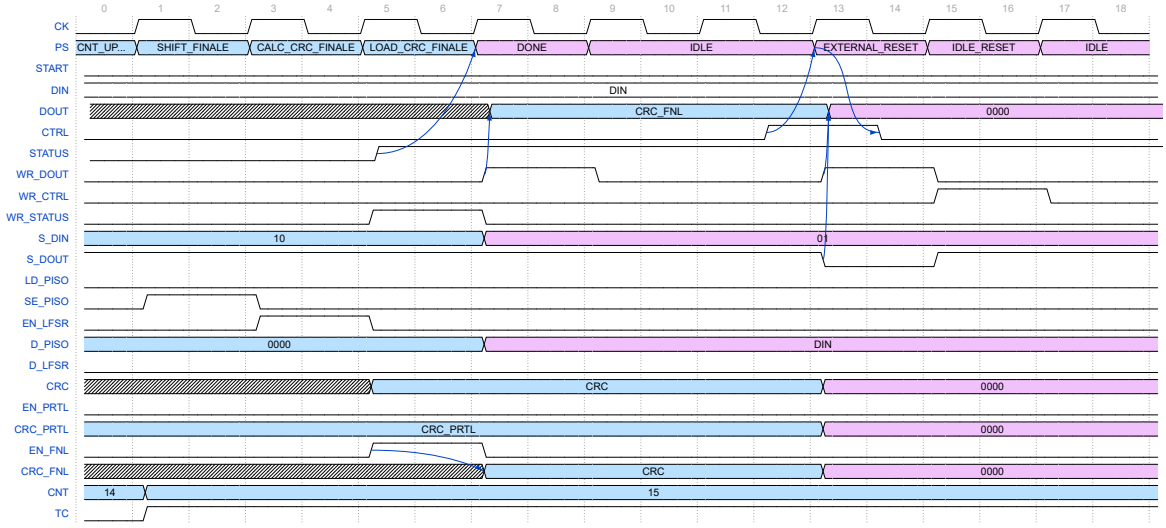
C Timing diagram



(a) Caricamento del CRC parziale dal registro REG_PARTIAL



(b) Caricamento del dato da processare dal Data In Register



(c) Caricamento del CRC nel Data Out Register e del reset indotto dall'esterno dal comando CTRL

D Descrizione dell'hardware

D.1 Component *slave* SPI

D.1.1 Rilevatore dei fronti di SCK

```
--*******************************************************************************
--* Rilevatore dei fronti di SCK con sovracampionamento
--* Fronte di discesa = 1100
--* Fronte di salite = 0011
--*******************************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity clock_edge is
  generic (N : integer := 4);
  port (
    sck          : in std_logic;
    clk, en, rst : in std_logic;
    sck_lox      : out std_logic;
    sck_hix      : out std_logic
  );
end clock_edge;

architecture structure of clock_edge is

  component SIPO is
    generic (N : integer);
    port (
      clk : in std_logic;
      rst : in std_logic;
      en  : in std_logic;
      d   : in std_logic;
      q   : out std_logic_vector(N - 1 downto 0)
    );
  end component;

  signal edge : std_logic_vector(3 downto 0);

begin
  REG_SCK : SIPO
  generic map(N => N)
  port map(clk => clk, rst => rst, en => en, d => sck, q => edge);

  SCK_LOx <= (edge(3) and edge(2)) and (not(edge(1)) and not(edge(0)));
  SCK_HIx <= (not(edge(3)) and not(edge(2))) and (edge(1) and edge(0));

end structure;
```

D.1.2 Rilevatore del comando *read*/*write*

```
--*******************************************************************************
--* Rilevatore del comando di scrittura o lettura
--* Scrittura = 00100000
```

```
--* Lettura = 00100001
--*********************************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity command is
  port (
    cmd  : in std_logic_vector(7 downto 0);
    w_en : out std_logic;
    r_en : out std_logic
  );
end command;

architecture structure of command is

begin
  w_en <= (not(cmd(7)) and not(cmd(6)) and cmd(5) and not(cmd(4)) and not(cmd(3)) and
  not(cmd(2)) and not(cmd(1))) and not(cmd(0));
  r_en <= (not(cmd(7)) and not(cmd(6)) and cmd(5) and not(cmd(4)) and not(cmd(3)) and
  not(cmd(2)) and not(cmd(1))) and cmd(0);

end structure;
```

D.1.3 Contatore a 5 bit

```
--*********************************************************************************
--* Contatore che solleva un flag quando arriva a 7 (TC8), 15 (TC16) e 31 (TC32)
--*********************************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
  port (
    en, rst, clk    : in std_logic;
    tc8, tc16, tc32 : out std_logic
  );
end counter;

architecture structure of counter is

  signal Q : unsigned(4 downto 0);

begin
  process (clk, en, rst)
  begin
    if (rst = '1') then -- reset attivo alto
      Q <= (others => '0');
    elsif (clk'event and clk = '1') then -- fronte di salita del clock
      if (en = '1') then
        Q <= Q + 1;
      end if;
    end if;
  end process;
```

```
31
32    tc8  <= Q(0) and Q(1) and Q(2) and not(Q(3)) and not(Q(4)); --7=00111
33    tc16 <= Q(0) and Q(1) and Q(2) and Q(3) and not(Q(4));      --15=01111
34    tc32 <= Q(0) and Q(1) and Q(2) and Q(3) and Q(4);           --31=11111
35
36  end structure;
```

D.1.4 Multiplexer a due vie per l'alta impedenza

```
1  --*********************************************************************************
2  --* Multiplexer a due vie con ingressi e uscita su 1 bit
3  --* s=0: l'uscita va in alta impedenza
4  --* s=1: trasmettiamo in uscita il valore presente in ingresso
5  --*********************************************************************************
6
7  library ieee;
8  use ieee.std_logic_1164.all;
9  use ieee.numeric_std.all;
10
11 entity mux_z is
12   port (
13     ingresso : in std_logic; --input
14     s        : in std_logic; --selettore
15     uscita   : out std_logic --output
16   );
17 end mux_z;
18
19 architecture structure of mux_z is
20
21 begin
22   uscita <= 'Z' when s = '0' --alta impedenza (s=0)
23     else
24     ingresso; --trasmissione dato (s=1)
25
26 end structure;
```

D.2 Component calcolatore di CRC

D.2.1 D flip-flop

```
1  --*********************************************************************************
2  --* Flip-flop di tipo D con parallelismo unitario sia in ingresso sia in uscita
3  --*********************************************************************************
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use ieee.numeric_std.all;
8
9  entity dflipflop is
10     port (
11         clk, rst, en : in std_logic;
12         d            : in std_logic;
13         q            : out std_logic
14     );
15 end dflipflop;
16
```

```
17  architecture structure of dflipflop is
18  begin
19
20      process (clk, rst)
21      begin
22          -- reset attivo --> inizializzo Q a 0:
23          if (rst = '1') then
24              q <= '0';
25          -- enable attivo sul fronte di salita del clock:
26          elsif (clk'event and clk = '1') then
27              if (en = '1') then
28                  q <= d;
29              end if;
30          end if;
31      end process;
32
33  end structure;
```

D.2.2 Linear Feedback Shift Register

```
1   --*******************************************************************************
2   --* Linear Feedback Shift Register
3   --* Utilizzato per calcolare il CRC con lo standard CRC-16-CCITT XMODEM
4   --*******************************************************************************
5
6   library ieee;
7   use ieee.std_logic_1164.all;
8   use ieee.numeric_std.all;
9
10  entity lfsr_crc16ccitt is
11      port (
12          clk, rst, en : in std_logic;
13          msg          : in std_logic;
14          crc          : buffer std_logic_vector(15 downto 0)
15      );
16  end lfsr_crc16ccitt;
17
18  architecture structure of lfsr_crc16ccitt is
19
20      signal xor0, xor5, xor12 : std_logic;
21      signal q15, q4, q11      : std_logic;
22
23      component dflipflop is
24          port (
25              clk, rst, en : in std_logic;
26              d            : in std_logic;
27              q            : out std_logic
28          );
29      end component;
30
31  begin
32
33      xor0  <= msg xor crc(15);
34      xor5  <= crc(4) xor crc(15);
35      xor12 <= crc(11) xor crc(15);
36
37      FF_0 : dflipflop
```

```
38      port map(clk => clk, rst => rst, en => en, d => xor0, q => crc(0));
39
40      FF_1_4 : for ii in 1 to 4 generate
41          FF_ii : dflipflop
42          port map(clk => clk, rst => rst, en => en, d => crc(ii - 1), q => crc(ii));
43      end generate;
44
45      FF_5 : dflipflop
46      port map(clk => clk, rst => rst, en => en, d => xor5, q => crc(5));
47
48      FF_6_11 : for ii in 6 to 11 generate
49          FF_ii : dflipflop
50          port map(clk => clk, rst => rst, en => en, d => crc(ii - 1), q => crc(ii));
51      end generate;
52
53      FF_12 : dflipflop
54      port map(clk => clk, rst => rst, en => en, d => xor12, q => crc(12));
55
56      FF_13_15 : for ii in 13 to 15 generate
57          FF_ii : dflipflop
58          port map(clk => clk, rst => rst, en => en, d => crc(ii - 1), q => crc(ii));
59      end generate;
60
61  end structure;
```

D.2.3 Contatore *generic*

```
1   --*******************************************************************************
2   --* Contatore 0 to N con count enable e terminal count
3   --*******************************************************************************
4
5   library ieee;
6   use ieee.std_logic_1164.all;
7   use ieee.numeric_std.all;
8
9   entity contatore is
10    generic (N : integer := 16);
11    port (
12      clock   : in std_logic;
13      rst, en : in std_logic;
14      TC      : out std_logic;
15      cnt     : buffer integer range 0 to N + 1
16    );
17  end contatore;
18
19  architecture structure of contatore is
20  begin
21
22    count_process : process (clock, rst)
23    begin
24      if (rst = '1') then -- rst asincrono
25        cnt <= 0;
26        TC  <= '0';
27      elsif (clock'event and clock = '1') then -- fronte di salita del clock
28        if (en = '1') then
29          TC  <= '0';
30          cnt <= cnt + 1;
```

```
        if cnt = N then -- fine ciclo conta, riporto cnt a zero
          cnt <= 0;
        elsif cnt = N - 1 then -- alzo flag di terminal count
          TC <= '1';
        end if;
      end if;
    end if;
  end process count_process;

end structure;
```

D.2.4 Multiplexer a due vie

```
--*******************************************************************************
--* Multiplexer a due vie con ingressi e uscita su N bit
--* s=0: out_mux = IN_0
--* s=1: out_mux = IN_1
--*******************************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_n_bits2to1 is
  generic (N : integer := 16);
  port (
    IN_0, IN_1 : in std_logic_vector(N - 1 downto 0); -- input a N bit
    s          : in std_logic;                        -- selettore a 1 bit
    uscita     : out std_logic_vector(N - 1 downto 0) -- output a N bit
  );
end mux_n_bits2to1;

architecture structure of mux_n_bits2to1 is
begin
  uscita <= IN_0 when s = '0' else
    IN_1;
end structure;
```

D.2.5 Multiplexer a quattro vie

```
--*******************************************************************************
--* Multiplexer a quattro vie con ingressi e uscita a N bit
--* s=00: out_mux = IN_0 (0)
--* s=01: out_mux = IN_1 (1)
--* s=10: out_mux = IN_2 (2)
--* s=11: out_mux = IN_3 (3)
--*******************************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_n_bits4to1 is
  generic (N : integer := 16);
  port (
    IN_0, IN_1, IN_2, IN_3 : in std_logic_vector(N - 1 downto 0); -- input a N bit
```

```
    s                     : in std_logic_vector(1 downto 0);     -- selettore a 2 bit
    uscita                : out std_logic_vector(N - 1 downto 0) -- output a N bit
  );
end mux_n_bits4to1;

architecture structure of mux_n_bits4to1 is
begin
  uscita <=
    IN_0 when s = "00" else --0
    IN_1 when s = "01" else --1
    IN_2 when s = "10" else --2
    IN_3;                   --3
end structure;
```

D.3 Component condivisi da tutti i blocchi

D.3.1 Registro con reset ed enable

```
--*******************************************************************************
--* Registro con parallelismo di ingresso e uscita pari a N bit (generic)
--*******************************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity reg is
  generic (N : integer);
  port (
    d            : in std_logic_vector(N - 1 downto 0);
    clk, rst, en : in std_logic;
    q            : out std_logic_vector(N - 1 downto 0)
  );
end reg;

architecture structure of reg is
begin

  process (clk, rst)
  begin
    if (rst = '1') then --reset asincrono
      q <= (others => '0');
    elsif (clk'event and clk = '1') then --fronte di salita del clock
      if (en = '1') then
        q <= d;
      end if;
    end if;
  end process;

end structure;
```

D.3.2 Registro SIPO

```
--*******************************************************************************
--* SIPO con parallelismo di uscita pari a N bit (generic)
--*******************************************************************************
```

```
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use ieee.numeric_std.all;
8
9  entity SIPO is
10   generic (N : integer);
11   port (
12     clk : in std_logic;
13     rst : in std_logic;
14     en  : in std_logic;
15     d   : in std_logic;
16     q   : out std_logic_vector(N - 1 downto 0)
17   );
18 end SIPO;
19
20 architecture structure of SIPO is
21
22   signal data : std_logic_vector(N - 1 downto 0);
23
24 begin
25
26   process (clk, rst)
27   begin
28     if (rst = '1') then -- reset attivo alto
29       data <= (others => '0');
30     elsif (clk'event and clk = '1') then -- fronte di salita del clock
31       if en = '1' then
32         data <= data(N - 2 downto 0) & d;
33       end if;
34     end if;
35   end process;
36
37   q <= data;
38
39 end structure;
```

D.3.3 Registro PISO

```
1  --*****************************************************************************
2  --* PISO con parallelismo di ingresso pari a N bit (generic)
3  --*****************************************************************************
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use ieee.numeric_std.all;
8
9  entity PISO is
10   generic (N : integer := 16);
11   port (
12     clk : in std_logic;
13     se  : in std_logic; -- shift enable
14     rst : in std_logic;
15     en  : in std_logic; -- load enable
16     d   : in std_logic_vector(N - 1 downto 0);
17     q   : out std_logic
18   );
```

```
19  end PISO;
20
21  architecture structure of PISO is
22
23    signal data : std_logic_vector(N - 1 downto 0);
24
25  begin
26    process (CLK, RST)
27    begin
28      if (RST = '1') then                          -- reset attivo alto
29        data <= (others => '0');                 -- reset
30        q    <= 'Z';                             -- uscita in alta impedenza
31      elsif (clk'event and clk = '1') then        -- fronte di salita del clock
32        if (EN = '1') then                             -- load
33          data <= d;
34        elsif (EN = '0' and SE = '1') then             -- shift
35          q                    <= data(15);           -- mando fuori il MSB
36          data(N - 1 downto 1) <= data(N - 2 downto 0); -- shifto di 1 bit verso sx
37          data(0)              <= '0';                -- appendo uno zero a dx (LSB)
38        end if;
39      end if;
40    end process;
41
42  end structure;
```

D.4 Progetto completo

D.4.1 Slave SPI

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity spi is
6     port (
7       CK, SCK : in std_logic;                     -- main clock e clock di sistema
8       nSS     : in std_logic;                     -- slave select (attivo basso)
9       RST     : in std_logic;                     -- reset (attivo alto)
10      MOSI    : in std_logic;                     -- Master Out Slave In
11      MISO    : out std_logic;                    -- Master In Slave Out
12      RD, WR  : out std_logic;                    -- segnali di controllo per memoria
13      A       : out std_logic_vector(7 downto 0);  -- indirizzo di memoria
14      DIN     : out std_logic_vector(15 downto 0); -- ingresso memoria (uscita per spi)
15      DOUT    : in std_logic_vector(15 downto 0)   -- uscita memoria (ingresso per spi)
16    );
17  end spi;
18
19  architecture structure of spi is
20
21    --*************************************************************************
22    --* Elenco degli stati
23    --*************************************************************************
24
25    type state_type is (
26      RESET, WAIT_nSS,
27      -- trasmissione CMD su MOSI ------------------------------------------------
```

```
28      CMD_WAIT_HI, CMD_SCK_HI, CMD_SCK_LOx, CMD_SCK_LOy, CMD_SCK_LOz, CMD_CNT_UP, UP_CMD,
        UP_CMD_LOz, CMD_CNT_UP_TC,
29      -- trasmissione ADD su MOSI ------------------------------------------------
30      ADD_SCK_HI, ADD_SCK_LOx, ADD_SCK_LOy, ADD_SCK_LOz, ADD_CNT_UP,
31      -- trasmissione DIN su MOSI -------------------------------------------------
32      DIN_WAIT_HI, DIN_CNT_UP_TC, DIN_SCK_HI, DIN_SCK_LOx, DIN_SCK_LOy, DIN_SCK_LOz,
        DIN_CNT_UP,
33      -- scrittura DIN in memoria --------------------------------------------------
34      UP_MEMx, UP_MEMy,
35      -- trasmissione DOUT su MISO -------------------------------------------------
36      DOUT_SCK_HIx, DOUT_SCK_HIy, DOUT_SHIFT, DOUT_MUX, DOUT_LSB, DOUT_MSB, DONE
37    );
38    signal PS, NS : state_type; -- present state (PS) e next state (NS)
39
40    --*****************************************************************************
41    --* Definizione segnali interni (N.B. I SEGNALI DI CONTROLLO SONO TUTTI ATTIVI ALTI)
42    --*****************************************************************************
43
44    signal SE_CMD, RST_CMD_SR, RST_CMD, CMD_EN : std_logic; -- comando
45    signal SE_ADD, RST_ADD_SR, RST_ADD, ADD_EN : std_logic; -- indirizzo
46    signal SE_DIN, RST_DIN_SR, RST_DIN, DIN_EN : std_logic; -- dato in scrittura
47    signal SE_DOUT, LD_DOUT, RST_DOUT_SR       : std_logic; -- dato in lettura
48    signal RST_CNT, CNT_EN, TC8, TC16, TC32    : std_logic; -- contatore
49    signal SCK_LOx, SCK_HIx, EN_SCK_EDGE       : std_logic; -- rilevatori dei fronti di
      SCK
50    signal S_MISO                              : std_logic; -- selettore mux di uscita
51
52    signal CMD_SR_OUT : std_logic_vector(7 downto 0);
53    signal CMD_OUT    : std_logic_vector(7 downto 0);
54    signal R_EN, W_EN : std_logic;
55    signal ADD_SR_OUT : std_logic_vector(7 downto 0);
56    signal DIN_SR_OUT : std_logic_vector(15 downto 0);
57    signal OUT_MUX    : std_logic;
58
59    --*****************************************************************************
60    --* Dichiarazione component
61    --*****************************************************************************
62
63    -- registro con ingressi e uscite su N bit
64    component reg is
65      generic (N : integer);
66      port (
67        d            : in std_logic_vector(N - 1 downto 0);
68        clk, rst, en : in std_logic;
69        q            : out std_logic_vector(N - 1 downto 0)
70      );
71    end component;
72
73    -- shift register SIPO con uscite su N bit
74    component SIPO is
75      generic (N : integer);
76      port (
77        clk : in std_logic;
78        rst : in std_logic;
79        en  : in std_logic;
80        d   : in std_logic;
81        q   : out std_logic_vector(N - 1 downto 0)
82      );
83    end component;
```

```
84
85    -- registro parallel in serial out con ingressi su N bit
86    component PISO is
87      generic (N : integer := 16);
88      port (
89        clk : in std_logic;
90        se  : in std_logic;
91        rst : in std_logic;
92        en  : in std_logic;
93        d   : in std_logic_vector(N - 1 downto 0);
94        q   : out std_logic
95      );
96    end component;
97
98    -- contatore a 5 bit con rilevatore di 7, 15, 31
99    component counter is
100     port (
101       en, rst, clk    : in std_logic;
102       tc8, tc16, tc32 : out std_logic
103     );
104   end component;
105
106   -- rilevatore del comando di scrittura o lettura
107   component command is
108     port (
109       cmd  : in std_logic_vector(7 downto 0);
110       w_en : out std_logic;
111       r_en : out std_logic
112     );
113   end component;
114
115   -- rilevatore dei fronti di SCK
116   component clock_edge is
117     generic (N : integer := 4);
118     port (
119       sck          : in std_logic;
120       clk, en, rst : in std_logic;
121       sck_lox      : out std_logic;
122       sck_hix      : out std_logic
123     );
124   end component;
125
126   -- multiplexer a due vie che collega l'ingresso all'uscita oppure la mette in Z
127   component mux_z is
128     port (
129       ingresso : in std_logic; -- input
130       s        : in std_logic; -- selettore
131       uscita   : out std_logic -- output
132     );
133   end component;
134
135   --*****************************************************************************
136   --* Architecture
137   --*****************************************************************************
138
139 begin
140
141   --* STEP 1: ASM dei controlli ***********************************************
142   controlASM : process (PS, nSS, SCK_LOx, SCK_HIx, TC8, TC16, TC32, W_EN, R_EN)
```

```
143   begin
144
145     ---------------------------------------------------------------------------
146     -- Valori di default ------------------------------------------------------
147     SE_CMD     <= '0';
148     RST_CMD_SR <= '0';
149     RST_CMD    <= '0';
150     CMD_EN     <= '0';
151     --
152     SE_ADD     <= '0';
153     RST_ADD_SR <= '0';
154     RST_ADD    <= '0';
155     ADD_EN     <= '0';
156     --
157     SE_DIN     <= '0';
158     RST_DIN_SR <= '0';
159     RST_DIN    <= '0';
160     DIN_EN     <= '0';
161     --
162     SE_DOUT     <= '0';
163     LD_DOUT     <= '0';
164     RST_DOUT_SR <= '0';
165     S_MISO      <= '0';
166     --
167     WR <= '0';
168     RD <= '0';
169     --
170     RST_CNT <= '0';
171     CNT_EN  <= '0';
172     --
173     EN_SCK_EDGE <= '1';
174     ---------------------------------------------------------------------------
175
176     case PS is
177
178       when RESET => -- resetto la macchina
179         RST_CMD_SR  <= '1';
180         RST_CMD     <= '1';
181         RST_ADD_SR  <= '1';
182         RST_ADD     <= '1';
183         RST_DIN_SR  <= '1';
184         RST_DIN     <= '1';
185         RST_DOUT_SR <= '1';
186         RST_CNT     <= '1';
187         -------------------
188         NS <= WAIT_nSS;
189
190         ---- ASM_CMD ----------------------------------------------------------
191         -- Il master invia gli 8 bit di CMD sul MOSI
192       when WAIT_nSS => -- reset contatore (TC8, TC16, TC32 = 0), aspetto asserimento di
          nSS
193         RST_CNT <= '1';
194         -------------------
195         if (nSS = '0') then
196           NS <= CMD_WAIT_HI;
197         else
198           NS <= WAIT_nSS;
199         end if;
200
```

```
201         when CMD_WAIT_HI => -- valori di default, aspetto fronte di salita SCK
202           if (SCK_HIx = '1') then
203             NS <= CMD_SCK_HI;
204           else
205             NS <= CMD_WAIT_HI;
206           end if;
207
208         when CMD_SCK_HI => -- valori di default, aspetto fronte di discesa SCK per
            campionare il MOSI
209           if (SCK_LOx = '1') then
210             NS <= CMD_SCK_LOx;
211           else
212             NS <= CMD_SCK_HI;
213           end if;
214
215         when CMD_SCK_LOx => -- CMD_SR campiona MOSI = CMD(7-CNT)
216           SE_CMD <= '1';
217           -------------------
218           NS <= CMD_SCK_LOy;
219
220         when CMD_SCK_LOy => -- stato di attesa, valori di default
221           if (TC8 = '1') then
222             NS <= UP_CMD;
223           else
224             NS <= CMD_SCK_LOz;
225           end if;
226
227         when CMD_SCK_LOz => -- valori di default, aspetto fronte di salita SCK per
            incrementare il contatore
228           if (SCK_HIx = '1') then
229             NS <= CMD_CNT_UP;
230           else
231             NS <= CMD_SCK_LOz;
232           end if;
233
234         when CMD_CNT_UP => -- incremento contatore con TC8=0
235           CNT_EN <= '1';
236           -------------------
237           NS <= CMD_SCK_HI;
238
239         when UP_CMD => -- memorizzo CMD in CMD_REG
240           CMD_EN <= '1';
241           -------------------
242           NS <= UP_CMD_LOz;
243
244         when UP_CMD_LOz => -- valori di default, aspetto fronte di salita SCK per
            incrementare il contatore
245           if (SCK_HIx = '1') then
246             NS <= CMD_CNT_UP_TC;
247           else
248             NS <= UP_CMD_LOz;
249           end if;
250
251         when CMD_CNT_UP_TC => -- incremento contatore con TC8=1
252           CNT_EN <= '1';
253           -------------------
254           NS <= ADD_SCK_HI;
255
256           ---- ASM_ADD ----------------------------------------------------------------
```

```
257           -- Il master invia gli 8 bit di indirizzo sul MOSI
258         when ADD_SCK_HI => -- valori di default, aspetto fronte di discesa SCK per
            campionare il MOSI
259           if (SCK_LOx = '1') then
260             NS <= ADD_SCK_LOx;
261           else
262             NS <= ADD_SCK_HI;
263           end if;
264
265         when ADD_SCK_LOx => -- ADD_SR campiona MOSI = ADD(15-CNT)
266           SE_ADD <= '1';
267           -------------------
268           NS <= ADD_SCK_LOy;
269
270         when ADD_SCK_LOy => -- stato di attesa, valori di default
271           if (TC16 = '1') then
272             if (W_EN = '1') then -- scrittura
273               NS <= DIN_WAIT_HI;
274             elsif (R_EN = '1') then -- lettura
275               NS <= DOUT_SCK_HIx;
276             else
277               NS <= RESET;
278             end if;
279           else
280             NS <= ADD_SCK_LOz;
281           end if;
282
283         when ADD_SCK_LOz => -- valori di default, aspetto fronte di salita SCK per
            incrementare il contatore
284           if (SCK_HIx = '1') then
285             NS <= ADD_CNT_UP;
286           else
287             NS <= ADD_SCK_LOz;
288           end if;
289
290         when ADD_CNT_UP => -- incremento contatore con TC16=0
291           CNT_EN <= '1';
292           -------------------
293           NS <= ADD_SCK_HI;
294
295           ---- ASM_DIN ------------------------------------------------------------
296           -- Il master invia i 16 bit di DIN sul MOSI
297         when DIN_WAIT_HI => -- valori di default, aspetto fronte di salita SCK per
            incrementare il contatore
298           if (SCK_HIx = '1') then
299             NS <= DIN_CNT_UP_TC;
300           else
301             NS <= DIN_WAIT_HI;
302           end if;
303
304         when DIN_CNT_UP_TC => -- incremento il contatore con TC16=1
305           CNT_EN <= '1';
306           -------------------
307           NS <= DIN_SCK_HI;
308
309         when DIN_SCK_HI => -- valori di default, aspetto fronte di discesa SCK per
            campionare il MOSI
310           if (SCK_LOx = '1') then
311             NS <= DIN_SCK_LOx;
```

```
312           else
313             NS <= DIN_SCK_HI;
314           end if;
315
316         when DIN_SCK_LOx => -- DIN_SR campiona MOSI = DIN(31-CNT)
317           SE_DIN <= '1';
318           -------------------
319           NS <= DIN_SCK_LOy;
320
321         when DIN_SCK_LOy => -- stato di attesa, valori di default
322           if (TC32 = '0') then
323             NS <= DIN_SCK_LOz;
324           else
325             NS <= UP_MEMx;
326           end if;
327
328         when DIN_SCK_LOz => -- valori di default, aspetto fronte di salita SCK per
            incrementare il contatore
329           if (SCK_HIx = '1') then
330             NS <= DIN_CNT_UP;
331           else
332             NS <= DIN_SCK_LOz;
333           end if;
334
335         when DIN_CNT_UP => -- incremento il contatore con TC32=0
336           CNT_EN <= '1';
337           -------------------
338           NS <= DIN_SCK_HI;
339
340         when UP_MEMx => -- memorizzo A in ADD_REG e DIN in DIN_REG
341           DIN_EN <= '1';
342           ADD_EN <= '1';
343           -------------------
344           NS <= UP_MEMy;
345
346         when UP_MEMy => -- invio il segnale di scrittura
347           WR <= '1';
348           -------------------
349           NS <= DONE;
350
351           ---- ASM_DOUT -----------------------------------------------------------------
352           -- Lo slave invia i 16 bit di DOUT sul MISO
353         when DOUT_SCK_HIx => -- memorizzo A in ADD_REG
354           ADD_EN <= '1';
355           -------------------
356           NS <= DOUT_SCK_HIy;
357
358         when DOUT_SCK_HIy => -- invio il segnale di lettura, carico DOUT nel PISO, aspetto
            fronte di salita SCK per iniziare a mandare i dati sul MISO
359           RD      <= '1';
360           LD_DOUT <= '1';
361           -------------------
362           if (SCK_HIx = '1') then
363             NS <= DOUT_MSB;
364           else
365             NS <= DOUT_SCK_HIy;
366           end if;
367
```

```
368         when DOUT_MSB => -- invio MSB di DOUT sull'ingresso 0 del mux, ma lascio ancora il
            MISO in Z
369           SE_DOUT <= '1';  -- OUT_MUX = DOUT(15)
370           CNT_EN  <= '1';  -- CNT++
371           S_MISO  <= '0';  -- MISO in Z
372           -------------------
373           NS <= DOUT_MUX;
374
375         when DOUT_SHIFT =>
376           SE_DOUT <= '1'; -- OUT_MUX = DOUT(31-CNT)
377           CNT_EN  <= '1'; -- CNT++
378           S_MISO  <= '1'; -- dati su MISO
379           -------------------
380           NS <= DOUT_MUX;
381
382         when DOUT_MUX => -- invio dati su MISO; aspetto fronte di salita di SCK per fare
            un nuovo shift o TC32 per terminare la transazione
383           S_MISO <= '1';
384           -------------------
385           if (TC32 = '0' and SCK_HIx = '1') then
386             NS <= DOUT_SHIFT;
387           elsif (TC32 = '0' and SCK_HIx = '0') then
388             NS <= DOUT_MUX;
389           else
390             NS <= DOUT_LSB;
391           end if;
392
393         when DOUT_LSB => -- aspetto fronte di salita di SCK o deasserimento nSS per andare
            in DONE
394           S_MISO <= '1';
395           -------------------
396           if (nSS = '1' or SCK_HIx = '1') then
397             NS <= DONE;
398           else
399             NS <= DOUT_LSB;
400           end if;
401
402         when DONE => -- stato di done (aspetto il deasserimento di nSS)
403           if (nSS = '1') then
404             NS <= WAIT_nSS;
405           else
406             NS <= DONE;
407           end if;
408
409         when others =>
410           NS <= RESET;
411
412       end case;
413     end process controlASM;
414
415     --* STEP 2: transizioni di stato ************************************************
416     transitionsFSM : process (CK, RST)
417     begin
418       if (RST = '1') then -- reset asincrono attivo alto
419         PS <= RESET;
420       elsif (CK'event and CK = '1') then -- fronte di salita del CK
421         PS <= NS;
422       end if;
423     end process transitionsFSM;
```

```
424
425     --* STEP 3: datapath ***************************************************************
426
427     CMD_SR : SIPO
428     generic map(N => 8)
429     port map(clk => CK, en => SE_CMD, rst => RST_CMD_SR, d => MOSI, q => CMD_SR_OUT);
430
431     CMD_REG : reg
432     generic map(N => 8)
433     port map(clk => CK, en => CMD_EN, rst => RST_CMD, d => CMD_SR_OUT, q => CMD_OUT);
434
435     ADD_SR : SIPO
436     generic map(N => 8)
437     port map(clk => CK, en => SE_ADD, rst => RST_ADD_SR, d => MOSI, q => ADD_SR_OUT);
438
439     ADD_REG : reg
440     generic map(N => 8)
441     port map(clk => CK, en => ADD_EN, rst => RST_ADD, d => ADD_SR_OUT, q => A);
442
443     DIN_SR : SIPO
444     generic map(N => 16)
445     port map(clk => CK, en => SE_DIN, rst => RST_DIN_SR, d => MOSI, q => DIN_SR_OUT);
446
447     DIN_REG : reg
448     generic map(N => 16)
449     port map(clk => CK, en => DIN_EN, rst => RST_DIN, d => DIN_SR_OUT, q => DIN);
450
451     DOUT_SR : PISO
452     generic map(N => 16)
453     port map(clk => CK, en => LD_DOUT, se => SE_DOUT, rst => RST_DOUT_SR, d => DOUT, q =>
        OUT_MUX);
454
455     COUNT : counter
456     port map(clk => CK, en => CNT_EN, rst => RST_CNT, tc8 => TC8, tc16 => TC16, tc32 =>
        TC32);
457
458     CMD_BLOCK : command
459     port map(cmd => CMD_OUT, w_en => W_EN, r_en => R_EN);
460
461     EDGE : clock_edge
462     generic map(N => 4)
463     port map(clk => CK, sck => SCK, rst => RST, en => EN_SCK_EDGE, sck_lox => SCK_LOx,
        sck_hix => SCK_HIx);
464
465     EXIT_MUX : mux_z
466     port map(ingresso => OUT_MUX, S => S_MISO, uscita => MISO);
467
468 end structure;
```

D.4.2 Interfaccia a registri

```
1 --*************************************************************************************
2 --* Register file con doppia interfaccia (SPI e CRC)
3 --* Il parallelismo dei dati è pari a N bit (generic)
4 --* ADDR = 0 Data In Register
5 --* ADDR = 1 CRC Out Register
6 --* ADDR = 2 Control Register
```

```
--* ADDR = 3 Status Register
--*****************************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity register_interface is
  generic (N : integer := 16);
  port (
    --* porte per interfaccia SPI/memoria ************************************
    CK    : in std_logic;
    WR    : in std_logic;            -- write mem (attivo alto)
    RD    : in std_logic;            -- read mem (attivo alto)
    ADDR  : in integer range 0 to 3; -- indirizzo mem
    START : out std_logic;           -- start CRC
    D     : in std_logic_vector(N - 1 downto 0);
    Q     : out std_logic_vector(N - 1 downto 0);
    --* porte per interfaccia CRC/memoria ************************************
    Q_DIN : out std_logic_vector(N - 1 downto 0);
    ------------------------------------------------------------------------
    D_DOUT  : in std_logic_vector(N - 1 downto 0);
    EN_DOUT : in std_logic; -- enable scrittura in CRC Out Register
    ------------------------------------------------------------------------
    D_CTRL  : in std_logic_vector(N - 1 downto 0);
    EN_CTRL : in std_logic; -- enable scrittura in Control Register
    Q_CTRL  : out std_logic_vector(N - 1 downto 0);
    ------------------------------------------------------------------------
    D_STATUS  : in std_logic_vector(N - 1 downto 0);
    EN_STATUS : in std_logic -- enable scrittura in Status Register
  );
end register_interface;

architecture structure of register_interface is

  type ram_array is array (0 to 3) of std_logic_vector (N - 1 downto 0);
  signal ram : ram_array;

begin

  crc_spi_interface : process (CK)
  begin

    if (CK'event and CK = '1') then
      START <= '0';
      if (RD = '1') then
        Q <= ram(ADDR);
      end if;

      if (WR = '1') then -- SPI scrive
        ram(ADDR) <= D;    -- scrivo in ram
        if (ADDR = 0) then -- start CRC
          START <= '1';
        end if;
      else -- CRC scrive
        if (EN_DOUT = '1') then
          ram(1) <= D_DOUT; -- scrivo in ram
        end if;

```

```
66          if (EN_CTRL = '1') then
67            ram(2) <= D_CTRL; -- scrivo in ram
68          end if;
69
70          if (EN_STATUS = '1') then
71            ram(3) <= D_STATUS; -- scrivo in ram_crc
72          end if;
73
74        end if;
75      end if;
76    end process crc_spi_interface;
77
78    Q_DIN  <= ram(0); -- CRC legge il Data In Register (sempre)
79    Q_CTRL <= ram(2); -- CRC legge il Control Register (sempre)
80
81  end structure;
```

D.4.3 Calcolatore di CRC

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity crc is
6       port (
7           CK         : in std_logic;
8           RST        : in std_logic;
9           START      : in std_logic;
10          DIN        : in std_logic_vector(15 downto 0);
11          DOUT       : out std_logic_vector(15 downto 0);
12          CTRL       : in std_logic;  -- reset calcolatore di CRC
13          CTRL_OUT   : out std_logic; -- dato da scrivere nel Control Register
14          STATUS     : out std_logic; -- stato calcolatore di CRC (busy/free)
15          WR_DOUT    : out std_logic; -- enable scrittura in CRC Out Register
16          WR_CTRL    : out std_logic; -- enable scrittura in Control Register
17          WR_STATUS : out std_logic  -- enable scrittura in Status Register
18      );
19  end crc;
20
21  architecture structure of crc is
22
23      --*****************************************************************************
24      --* Elenco degli stati
25      --*****************************************************************************
26
27      type state_type is (
28          RESET, IDLE, IDLE_RESET, EXTERNAL_RESET, INTERNAL_RESET,
29          LOAD_PRTL, SHIFT_PRTL, CALC_CRC_PRTL, CNT_UP_PRTL,
30          LOAD_DIN, SHIFT_DIN, CALC_CRC_DIN, CNT_UP_DIN, LOAD_CRC_DIN,
31          LOAD_FINALE, SHIFT_FINALE, CALC_CRC_FINALE, CNT_UP_FINALE, LOAD_CRC_FINALE,
32          DONE
33      );
34      signal PS, NS : state_type; -- present state (PS) e next state (NS)
35
36      --*****************************************************************************
37      --* Definizione segnali interni (N.B. I SEGNALI DI CONTROLLO SONO TUTTI ATTIVI ALTI)
38      --*****************************************************************************
```

```
39
40      signal S_DIN             : std_logic_vector(1 downto 0);  -- selettore MUX_DIN (2
        bit)
41      signal S_DOUT            : std_logic;                     -- selettore MUX_DOUT (1
        bit)
42      signal LD_PISO, SE_PISO  : std_logic;                     -- controlli PISO_DIN
43      signal RST_PISO          : std_logic;                     -- reset PISO_DIN
44      signal EN_LFSR, RST_LFSR : std_logic;                     -- controlli LFSR
45      signal EN_PRTL, RST_PRTL : std_logic;                     -- controlli REG_PARTIAL
46      signal EN_FNL, RST_FNL   : std_logic;                     -- controlli REG_FINAL
47      signal EN_CNT, RST_CNT   : std_logic;                     -- controlli contatore
48      signal TC                : std_logic;                     -- terminal count
49      signal D_PISO            : std_logic_vector(15 downto 0); -- ingresso PISO_DIN
50      signal D_LFSR            : std_logic;                     -- ingresso LFSR
51      signal CRC               : std_logic_vector(15 downto 0); -- uscita LFSR
52      signal CRC_PRTL, CRC_FNL : std_logic_vector(15 downto 0); -- uscite REG_PARTIAL e
        REG_FINAL
53      signal zeros             : std_logic_vector(15 downto 0); -- stringa di 16 zeri (per
        il reset)
54      signal CNT               : integer;                       -- uscita contatore
55
56      --*************************************************************************************
57      --* Dichiarazione component
58      --*************************************************************************************
59
60      -- linear feedback shift register (LFSR)
61      component lfsr_crc16ccitt is
62          port (
63              clk, rst, en : in std_logic;
64              msg          : in std_logic;
65              crc          : buffer std_logic_vector(15 downto 0)
66          );
67      end component;
68
69      -- registro con ingressi e uscite su N bit
70      component reg is
71          generic (N : integer);
72          port (
73              d            : in std_logic_vector(N - 1 downto 0);
74              clk, rst, en : in std_logic;
75              q            : out std_logic_vector(N - 1 downto 0)
76          );
77      end component;
78
79      -- registro parallel in serial out con ingressi su N bit
80      component PISO is
81          generic (N : integer := 16);
82          port (
83              clk : in std_logic;
84              se  : in std_logic;
85              rst : in std_logic;
86              en  : in std_logic;
87              d   : in std_logic_vector(N - 1 downto 0);
88              q   : out std_logic
89          );
90      end component;
91
92      component contatore is
93          generic (N : integer := 16);
```

```
94          port (
95              clock   : in std_logic;
96              rst, en : in std_logic;
97              TC      : out std_logic;
98              cnt     : buffer integer range 0 to N + 1
99          );
100     end component;
101
102     component mux_1_bit2to1 is
103         port (
104             IN_0, IN_1 : in std_logic; -- input a 1 bit
105             s          : in std_logic; -- selettore a 1 bit
106             uscita     : out std_logic -- output a 1 bit
107         );
108     end component;
109
110     -- multiplexer a due vie (N bit)
111     component mux_n_bits2to1 is
112         generic (N : integer := 16);
113         port (
114             IN_0, IN_1 : in std_logic_vector(N - 1 downto 0); -- input a N bit
115             s          : in std_logic;                        -- selettore a 1 bit
116             uscita     : out std_logic_vector(N - 1 downto 0) -- output a N bit
117         );
118     end component;
119
120     -- multiplexer a 4 vie
121     component mux_n_bits4to1 is
122         generic (N : integer := 16);
123         port (
124             IN_0, IN_1, IN_2, IN_3 : in std_logic_vector(N - 1 downto 0); -- input a N
                bit
125             s                      : in std_logic_vector(1 downto 0);     -- selettore a
                2 bit
126             uscita                 : out std_logic_vector(N - 1 downto 0) -- output a N
                bit
127         );
128     end component;
129
130     --*************************************************************************
131     --* Architecture
132     --*************************************************************************
133
134 begin
135
136     --* STEP 1: ASM dei controlli *********************************************
137     controlASM : process (PS, START, CTRL, TC)
138     begin
139
140         ---------------------------------------------------------------------------
141         -- Valori di default ------------------------------------------------------
142         S_DIN  <= "01"; -- rilevazione dati da register file
143         S_DOUT <= '1';  -- caricamento risultato nel CRC Out Register
144         --
145         LD_PISO  <= '0';
146         SE_PISO  <= '0';
147         RST_PISO <= '0';
148         --
149         EN_LFSR  <= '0';
```

```
150         RST_LFSR <= '0';
151         --
152         EN_PRTL  <= '0';
153         RST_PRTL <= '0';
154         --
155         EN_FNL  <= '0';
156         RST_FNL <= '0';
157         --
158         EN_CNT  <= '0';
159         RST_CNT <= '0';
160         --
161         WR_DOUT   <= '0';
162         WR_CTRL   <= '0';
163         WR_STATUS <= '0';
164         STATUS    <= '1'; -- CRC "free"
165         ---------------------------------------------------------------------------
166
167         case PS is
168
169             when RESET => -- resetto la macchina
170                 S_DOUT    <= '0';
171                 WR_DOUT   <= '1';
172                 WR_CTRL   <= '1';
173                 STATUS    <= '1';
174                 WR_STATUS <= '1';
175                 RST_LFSR  <= '1';
176                 RST_PRTL  <= '1';
177                 RST_FNL   <= '1';
178                 RST_CNT   <= '1';
179                 -------------------
180                 NS <= IDLE;
181
182             when IDLE => -- valori di default
183                 if (START = '1') then
184                     if (CTRL = '1') then
185                         NS <= EXTERNAL_RESET;
186                     else
187                         NS <= INTERNAL_RESET;
188                     end if;
189                 else -- START=0
190                     if (CTRL = '1') then
191                         NS <= EXTERNAL_RESET;
192                     else
193                         NS <= IDLE;
194                     end if;
195                 end if;
196
197             when EXTERNAL_RESET => -- reset innescato dall'esterno
198                 RST_LFSR <= '1';
199                 RST_PRTL <= '1';
200                 RST_FNL  <= '1';
201                 S_DOUT   <= '0'; -- giro il mux di uscita
202                 WR_DOUT  <= '1'; -- scrivo 16 zeri nel CRC Out Register
203                 -------------------
204                 NS <= IDLE_RESET;
205
206             when IDLE_RESET => -- attesa dello START dopo il reset esterno
207                 WR_CTRL <= '1';    -- reset Control Register
208                 -------------------
```

```
209                 if (START = '1') then
210                     NS <= LOAD_PRTL;
211                 else
212                     NS <= IDLE;
213                 end if;
214
215             when INTERNAL_RESET => -- reset interno, preliminare a qualsiasi calcolo
216                 RST_LFSR <= '1';
217                 -------------------
218                 NS <= LOAD_PRTL;
219
220             when LOAD_PRTL => -- carico il CRC parziale del ciclo precedente nel PISO
                (N.B. inutile al primo giro perché la macchina è stata appena resettata
                quindi carico tutti zeri, però lo faccio lo stesso perché l'SPI è molto
                lenta e non se ne accorge nemmeno)
221                 S_DIN   <= "00";  -- giro il mux di ingresso su REG_PARTIAL
222                 LD_PISO <= '1';   -- carico il PISO
223                 RST_CNT <= '1';   -- resetto il contatore
224                 -------------------
225                 NS <= SHIFT_PRTL;
226
227             when SHIFT_PRTL => -- shifto di 1 posizione il parziale memorizzato nel PISO
                e mando il LSB al calcolatore di CRC
228                 S_DIN   <= "00";   -- giro il mux di ingresso su REG_PARTIAL
229                 SE_PISO <= '1';    -- shift PISO
230                 -------------------
231                 NS <= CALC_CRC_PRTL;
232
233             when CALC_CRC_PRTL => -- calcolo il CRC appendendo in coda il bit inviato
                dal PISO
234                 S_DIN   <= "00";      -- giro il mux di ingresso su REG_PARTIAL
235                 EN_LFSR <= '1';       -- abilito il calcolatore di CRC
236                 -------------------
237                 if (TC = '1') then -- CNT=15
238                     NS <= LOAD_DIN;
239                 else -- CNT<15
240                     NS <= CNT_UP_PRTL;
241                 end if;
242
243             when CNT_UP_PRTL => -- incremento di 1 il contatore
244                 S_DIN  <= "00";     -- giro il mux di ingresso su REG_PARTIAL
245                 EN_CNT <= '1';      -- cnt++
246                 -------------------
247                 NS <= SHIFT_PRTL;
248
249             when LOAD_DIN =>  -- carico il contenuto del Data In Register nel PISO +
                dico all'SPI che il CRC è busy
250                 STATUS    <= '0'; -- crc "busy"
251                 WR_STATUS <= '1'; -- scrivo "busy" nello Status Register
252                 LD_PISO   <= '1'; -- carico il PISO
253                 RST_CNT   <= '1'; -- resetto il contatore
254                 -------------------
255                 NS <= SHIFT_DIN;
256
257             when SHIFT_DIN => -- shifto di 1 posizione il dato memorizzato nel PISO e
                mando il LSB al calcolatore di CRC
258                 STATUS  <= '0';   -- crc "busy"
259                 SE_PISO <= '1';   -- shift PISO
260                 -------------------
```

```
261                 NS <= CALC_CRC_DIN;
262
263             when CALC_CRC_DIN => -- calcolo il CRC appendendo in coda il bit inviato dal
                PISO
264                 STATUS  <= '0';      -- crc "busy"
265                 EN_LFSR <= '1';      -- abilito il calcolatore di CRC
266                 -------------------
267                 if (TC = '1') then -- CNT=15
268                     NS <= LOAD_CRC_DIN;
269                 else -- CNT<15
270                     NS <= CNT_UP_DIN;
271                 end if;
272
273             when CNT_UP_DIN => -- incremento di 1 il contatore
274                 STATUS <= '0';     -- crc "busy"
275                 EN_CNT <= '1';     -- cnt++
276                 -------------------
277                 NS <= SHIFT_DIN;
278
279             when LOAD_CRC_DIN => -- carico il nuovo CRC parziale in REG_PARTIAL
280                 STATUS  <= '0';      -- crc "busy"
281                 EN_PRTL <= '1';      -- carico il risultato in REG_PARTIAL
282                 -------------------
283                 NS <= LOAD_FINALE;
284
285             when LOAD_FINALE => -- carico i 16 zeri finali nel PISO
286                 STATUS  <= '0';     -- crc "busy"
287                 S_DIN   <= "10";    -- giro il mux sui 16 zeri (append finale)
288                 LD_PISO <= '1';     -- carico il PISO
289                 RST_CNT <= '1';     -- resetto il contatore
290                 -------------------
291                 NS <= SHIFT_FINALE;
292
293             when SHIFT_FINALE => -- shifto di 1 posizione il dato memorizzato nel PISO e
                mando il LSB al calcolatore di CRC
294                 STATUS  <= '0';      -- crc "busy"
295                 S_DIN   <= "10";     -- giro il mux sui 16 zeri (append finale)
296                 SE_PISO <= '1';      -- shift PISO
297                 -------------------
298                 NS <= CALC_CRC_FINALE;
299
300             when CALC_CRC_FINALE => -- calcolo il CRC appendendo in coda il bit inviato
                dal PISO
301                 STATUS  <= '0';         -- crc "busy"
302                 S_DIN   <= "10";        -- giro il mux sui 16 zeri (append finale)
303                 EN_LFSR <= '1';         -- abilito il calcolatore di CRC
304                 -------------------
305                 if (TC = '1') then -- CNT=15
306                     NS <= LOAD_CRC_FINALE;
307                 else -- CNT<15
308                     NS <= CNT_UP_FINALE;
309                 end if;
310
311             when CNT_UP_FINALE => -- incremento di 1 il contatore
312                 STATUS <= '0';        -- crc "busy"
313                 S_DIN  <= "10";       -- giro il mux sui 16 zeri (append finale)
314                 EN_CNT <= '1';        -- cnt++
315                 -------------------
316                 NS <= SHIFT_FINALE;
```

```
317
318             when LOAD_CRC_FINALE => -- carico il CRC definitivo nel registro REG_FINAL +
                "libero" il CRC
319                 STATUS    <= '1';       -- crc "free"
320                 S_DIN     <= "10";      -- giro il mux sui 16 zeri (append finale)
321                 EN_FNL    <= '1';       -- carico il risultato in REG_FINAL
322                 WR_STATUS <= '1';       -- scrivo "free" nello Status Register
323                 -------------------
324                 NS <= DONE;
325
326             when DONE =>    -- scrivo il risultato in memoria
327                 WR_DOUT <= '1'; -- scrivo nel CRC Out Register
328                 -------------------
329                 if (START = '1') then
330                     NS <= DONE;
331                 else
332                     NS <= IDLE;
333                 end if;
334
335             when others =>
336                 NS <= RESET;
337
338         end case;
339
340     end process controlASM;
341
342     --* STEP 2: transizioni di stato ************************************************
343     transitionsFSM : process (CK, RST)
344     begin
345         if (RST = '1') then -- reset asincrono attivo alto
346             PS <= RESET;
347         elsif (CK'event and CK = '1') then -- fronte di salita del CK
348             PS <= NS;
349         end if;
350     end process transitionsFSM;
351
352     --* STEP 3: datapath ************************************************************
353     zeros    <= (others => '0');
354     CTRL_OUT <= '0';
355
356     MUX_DIN : mux_n_bits4to1
357     generic map(N => 16)
358     port map(IN_0 => CRC_PRTL, IN_1 => DIN, IN_2 => zeros, IN_3 => zeros, s => S_DIN,
        uscita => D_PISO);
359
360     PISO_DIN : PISO
361     generic map(N => 16)
362     port map(clk => CK, se => SE_PISO, rst => RST_PISO, en => LD_PISO, d => D_PISO, q =>
        D_LFSR);
363
364     LFSR : lfsr_crc16ccitt
365     port map(clk => CK, rst => RST_LFSR, en => EN_LFSR, msg => D_LFSR, crc => CRC);
366
367     REG_PARTIAL : reg
368     generic map(N => 16)
369     port map(d => CRC, clk => CK, rst => RST_PRTL, en => EN_PRTL, q => CRC_PRTL);
370
371     REG_FINAL : reg
372     generic map(N => 16)
```

```
373     port map(d => CRC, clk => CK, rst => RST_FNL, en => EN_FNL, q => CRC_FNL);
374
375     MUX_DOUT : mux_n_bits2to1
376     generic map(N => 16)
377     port map(IN_0 => zeros, IN_1 => CRC_FNL, s => S_DOUT, uscita => DOUT);
378
379     COUNT_16 : contatore
380     generic map(N => 15)
381     port map(clock => CK, rst => RST_CNT, en => EN_CNT, TC => TC, cnt => CNT);
382
383 end structure;
```

D.4.4 Top level

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity top_level is
6      port (
7          CK   : in std_logic;
8          SCK  : in std_logic;
9          nSS  : in std_logic;
10         RST  : in std_logic;
11         MOSI : in std_logic;
12         MISO : out std_logic
13     );
14 end entity;
15
16 architecture structure of top_level is
17
18     --*****************************************************************************
19     --* Definizione segnali interni
20     --*****************************************************************************
21
22     signal RD, WR     : std_logic;                     -- controlli reg file lato SPI
23     signal A_SPI      : std_logic_vector(7 downto 0);  -- indirizzo reg file (vector)
24     signal A_SPI_INT  : integer range 0 to 3;          -- indirizzo reg file (integer)
25     signal D_SPI      : std_logic_vector(15 downto 0); -- ingresso reg file lato SPI
26     signal Q_SPI      : std_logic_vector(15 downto 0); -- uscita reg file lato SPI
27     signal E0         : std_logic;                     -- start crc
28     signal E1, E2, E3 : std_logic;                     -- controlli reg file lato CRC
29     signal D1, D2, D3 : std_logic_vector(15 downto 0); -- ingressi reg file lato CRC
30     signal Q0, Q2     : std_logic_vector(15 downto 0); -- uscite reg file lato CRC
31     signal CTRL_CRC   : std_logic;                     -- segnale per Control Register
32     signal STATUS_CRC : std_logic;                     -- stato CRC (busy/free)
33
34     --*****************************************************************************
35     --* Dichiarazione component
36     --*****************************************************************************
37
38     component spi is
39         port (
40             CK, SCK : in std_logic;
41             nSS     : in std_logic;
42             RST     : in std_logic;
43             MOSI    : in std_logic;
```

```
            MISO    : out std_logic;
            RD, WR  : out std_logic;
            A       : out std_logic_vector(7 downto 0);
            DIN     : out std_logic_vector(15 downto 0);
            DOUT    : in std_logic_vector(15 downto 0)
        );
    end component;

    component register_interface is
        generic (N : integer := 16);
        port (
            --* porte per interfaccia SPI/memoria ************************************
            CK    : in std_logic;
            WR    : in std_logic;
            RD    : in std_logic;
            ADDR  : in integer range 0 to 3;
            START : out std_logic;
            D     : in std_logic_vector(N - 1 downto 0);
            Q     : out std_logic_vector(N - 1 downto 0);
            --* porte per interfaccia CRC/memoria ************************************
            Q_DIN : out std_logic_vector(N - 1 downto 0);
            ------------------------------------------------------------------------
            D_DOUT  : in std_logic_vector(N - 1 downto 0);
            EN_DOUT : in std_logic;
            ------------------------------------------------------------------------
            D_CTRL  : in std_logic_vector(N - 1 downto 0);
            EN_CTRL : in std_logic;
            Q_CTRL  : out std_logic_vector(N - 1 downto 0);
            ------------------------------------------------------------------------
            D_STATUS  : in std_logic_vector(N - 1 downto 0);
            EN_STATUS : in std_logic
        );
    end component;

    component crc is
        port (
            CK        : in std_logic;
            RST       : in std_logic;
            START     : in std_logic;
            DIN       : in std_logic_vector(15 downto 0);
            DOUT      : out std_logic_vector(15 downto 0);
            CTRL      : in std_logic;
            CTRL_OUT  : out std_logic;
            STATUS    : out std_logic;
            WR_DOUT   : out std_logic;
            WR_CTRL   : out std_logic;
            WR_STATUS : out std_logic
        );
    end component;

begin

    A_SPI_INT <= to_integer(unsigned(A_SPI));

    D2 <= (0 => CTRL_CRC, others => '0');
    D3 <= (0 => STATUS_CRC, others => '0');

    SPI_INTERFACE : spi
    port map(
```

```
103         CK   => CK,
104         SCK  => SCK,
105         nSS  => nSS,
106         RST  => RST,
107         A    => A_SPI,
108         DIN  => D_SPI,
109         DOUT => Q_SPI,
110         RD   => RD,
111         WR   => WR,
112         MOSI => MOSI,
113         MISO => MISO
114     );
115
116     REGISTER_FILE : register_interface
117     generic map(N => 16)
118     port map(
119         CK   => CK,
120         WR   => WR,
121         RD   => RD,
122         ADDR => A_SPI_INT,
123         D    => D_SPI,
124         Q    => Q_SPI,
125         ----------------------------------------------------
126         Q_DIN => Q0,
127         START => E0,
128         ----------------------------------------------------
129         D_DOUT  => D1,
130         EN_DOUT => E1,
131         ----------------------------------------------------
132         D_CTRL  => D2,
133         EN_CTRL => E2,
134         Q_CTRL  => Q2,
135         ----------------------------------------------------
136         D_STATUS  => D3,
137         EN_STATUS => E3
138     );
139
140     CRC_CALCULATOR : crc
141     port map(
142         CK         => CK,
143         RST        => RST,
144         START      => E0,
145         DIN        => Q0,
146         DOUT       => D1,
147         CTRL       => Q2(0),
148         CTRL_OUT   => CTRL_CRC,
149         STATUS     => STATUS_CRC,
150         WR_DOUT    => E1,
151         WR_CTRL    => E2,
152         WR_STATUS  => E3
153     );
154
155 end structure;
```

E Test

E.1 Testbench a scopo di *debug*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_crc_simple is
end tb_crc_simple;

architecture behavioral of tb_crc_simple is

    -- definizione segnali interni
    signal clock   : std_logic := '0';              -- main clock
    signal s_clock : std_logic := '0';              -- serial clock
    signal nSS_spi : std_logic := '1';              -- slave select
    signal reset   : std_logic := '0';              -- reset esterno
    signal s_mosi  : std_logic := '1';              -- MOSI seriale
    signal s_miso  : std_logic := 'Z';              -- MISO seriale
    signal MOSI_wr : std_logic_vector(31 downto 0); -- vettore dati per WR
    signal MOSI_rd : std_logic_vector(15 downto 0); -- vettore dati per RD
    signal MISO_rd : std_logic_vector(15 downto 0); -- vettore restituito su MISO

    -- dichiarazione UUT
    component top_level is
        port (
            CK, SCK : in std_logic;
            nSS     : in std_logic;
            RST     : in std_logic;
            MOSI    : in std_logic;
            MISO    : out std_logic
        );
    end component;

begin

    -- istanza UUT
    TB_CRC_CALCULATOR : top_level
    port map(
        CK => clock, SCK => s_clock,
        RST => reset, nSS => nSS_spi,
        MOSI => s_mosi, MISO => s_miso
    );

    -- Process CK (clock FPGA, periodo 100 ns, f=10 MHz)
    CK_process : process
    begin
        wait for 50 ns;
        clock <= not clock;
    end process CK_process;

    -- Process di lettura e scrittura
    -- Effettuo alcune transazioni
    RD_WR_process : process
        variable cnt_bit : integer := 15;
    begin
        wait for 100 ns;
```

```
55          reset <= '1';
56          wait for 100 ns;
57          reset <= '0';
58
59          -- SCRITTURA NEL DATA IN REGISTER: w001206
60          -- CMD 00100000 '20' (w)
61          -- ADD 00000000 '00'
62          -- DIN 0001001000000110 '1206'
63          MOSI_wr <= "00100000" & "00000000" & "0001001000000110";
64          nSS_spi <= '0';
65          wait for 100 ns;
66
67          for i in 0 to 31 loop --invio dati su mosi
68              s_mosi  <= MOSI_wr(31 - i);
69              s_clock <= '1';
70              wait for 500 ns;
71              s_clock <= '0';
72              wait for 500 ns;
73          end loop;
74
75          wait for 200 ns;
76          nSS_spi <= '1';
77          wait for 5 us;
78
79          -- LETTURA DEL CRC OUT REGISTER: r01
80          -- CMD 00100001 '21' (r)
81          -- ADD 00000001 '01'
82          MOSI_rd <= "00100001" & "00000001";
83          nSS_spi <= '0';
84          wait for 1 us;
85
86          for i in 0 to 15 loop --invio dati su mosi
87              s_mosi  <= MOSI_rd(15 - i);
88              s_clock <= '1';
89              wait for 500 ns;
90              s_clock <= '0';
91              wait for 500 ns;
92          end loop;
93
94          --cnt_bit <= 15;
95          for i in 0 to 18 loop -- aspetto il dato sul MISO: CRC(5555) = FB1A
96              s_clock <= '1';
97              if (s_miso /= 'Z') then
98                  MISO_rd(cnt_bit) <= s_miso;
99                  cnt_bit := cnt_bit - 1;
100             end if;
101             wait for 500 ns;
102             s_clock <= '0';
103             wait for 500 ns;
104         end loop;
105
106         cnt_bit := 15;
107         nSS_spi <= '1';
108         wait for 5 us;
109
110         -- SCRITTURA NEL DATA IN REGISTER: w002203
111         -- CMD 00100000 '20' (w)
112         -- ADD 00000000 '00'
113         -- DIN 0010001000000011 '2203'
```

```
114         MOSI_wr <= "00100000" & "00000000" & "0010001000000011";
115         nSS_spi <= '0';
116         wait for 100 ns;
117 
118         for i in 0 to 31 loop --invio dati su mosi
119             s_mosi  <= MOSI_wr(31 - i);
120             s_clock <= '1';
121             wait for 500 ns;
122             s_clock <= '0';
123             wait for 500 ns;
124         end loop;
125 
126         wait for 200 ns;
127         nSS_spi <= '1';
128         wait for 5 us;
129 
130         -- LETTURA DEL CRC OUT REGISTER: r01
131         -- CMD 00100001 '21' (r)
132         -- ADD 00000001 '01'
133         MOSI_rd <= "00100001" & "00000001";
134         nSS_spi <= '0';
135         wait for 1 us;
136 
137         for i in 0 to 15 loop --invio dati su mosi
138             s_mosi  <= MOSI_rd(15 - i);
139             s_clock <= '1';
140             wait for 500 ns;
141             s_clock <= '0';
142             wait for 500 ns;
143         end loop;
144 
145         for i in 0 to 18 loop -- aspetto il dato sul MISO: CRC(5555AAAA) = 9A55
146             s_clock <= '1';
147             if (s_miso /= 'Z') then
148                 MISO_rd(cnt_bit) <= s_miso;
149                 cnt_bit := cnt_bit - 1;
150             end if;
151             wait for 500 ns;
152             s_clock <= '0';
153             wait for 500 ns;
154         end loop;
155 
156         cnt_bit := 15;
157         nSS_spi <= '1';
158         wait for 5 us;
159 
160         -- SCRITTURA NEL DATA IN REGISTER: w000306
161         -- CMD 00100000 '20' (w)
162         -- ADD 00000000 '00'
163         -- DIN 0000001100000110 '0306'
164         MOSI_wr <= "00100000" & "00000000" & "0000001100000110";
165         nSS_spi <= '0';
166         wait for 100 ns;
167 
168         for i in 0 to 31 loop --invio dati su mosi
169             s_mosi  <= MOSI_wr(31 - i);
170             s_clock <= '1';
171             wait for 500 ns;
172             s_clock <= '0';
```

```
173             wait for 500 ns;
174         end loop;
175 
176         wait for 200 ns;
177         nSS_spi <= '1';
178         wait for 5 us;
179 
180         -- SCRITTURA NEL CONTROL REGISTER: w020001
181         -- CMD 00100000 '20' (w)
182         -- ADD 00000010 '02'
183         -- DIN 0000000000000001 '0001'
184         MOSI_wr <= "00100000" & "00000010" & "0000000000000001";
185         nSS_spi <= '0';
186         wait for 100 ns;
187 
188         for i in 0 to 31 loop --invio dati su mosi
189             s_mosi  <= MOSI_wr(31 - i);
190             s_clock <= '1';
191             wait for 500 ns;
192             s_clock <= '0';
193             wait for 500 ns;
194         end loop;
195 
196         wait for 200 ns;
197         nSS_spi <= '1';
198         wait for 5 us; -- tempo affinche il dato venga mandato in memoria
199 
200         -- SCRITTURA NEL DATA IN REGISTER: w003f1b
201         -- CMD 00100000 '20' (w)
202         -- ADD 00000000 '00'
203         -- DIN 0011111100011011 '3f1b'
204         MOSI_wr <= "00100000" & "00000000" & "0011111100011011";
205         nSS_spi <= '0';
206         wait for 100 ns;
207 
208         for i in 0 to 31 loop --invio dati su mosi
209             s_mosi  <= MOSI_wr(31 - i);
210             s_clock <= '1';
211             wait for 500 ns;
212             s_clock <= '0';
213             wait for 500 ns;
214         end loop;
215 
216         wait for 200 ns;
217         nSS_spi <= '1';
218         wait for 5 us;
219 
220         -- LETTURA DEL CRC OUT REGISTER: r01
221         -- CMD 00100001 '21' (r)
222         -- ADD 00000001 '01'
223         MOSI_rd <= "00100001" & "00000001";
224         nSS_spi <= '0';
225         wait for 1 us;
226 
227         for i in 0 to 15 loop --invio dati su mosi
228             s_mosi  <= MOSI_rd(15 - i);
229             s_clock <= '1';
230             wait for 500 ns;
231             s_clock <= '0';
```

```
232             wait for 500 ns;
233         end loop;
234 
235         for i in 0 to 18 loop -- aspetto il dato sul MISO: CRC(3F1B) = B6F1
236             s_clock <= '1';
237             if (s_miso /= 'Z') then
238                 MISO_rd(cnt_bit) <= s_miso;
239                 cnt_bit := cnt_bit - 1;
240             end if;
241             wait for 500 ns;
242             s_clock <= '0';
243             wait for 500 ns;
244         end loop;
245 
246         cnt_bit := 15;
247         nSS_spi <= '1';
248         wait for 5 us;
249 
250         -- SCRITTURA NEL CONTROL REGISTER: w020001
251         -- CMD 00100000 '20' (w)
252         -- ADD 00000010 '02'
253         -- DIN 0000000000000001 '0001'
254         MOSI_wr <= "00100000" & "00000010" & "0000000000000001";
255         nSS_spi <= '0';
256         wait for 100 ns;
257 
258         for i in 0 to 31 loop --invio dati su mosi
259             s_mosi  <= MOSI_wr(31 - i);
260             s_clock <= '1';
261             wait for 500 ns;
262             s_clock <= '0';
263             wait for 500 ns;
264         end loop;
265 
266         wait for 200 ns;
267         nSS_spi <= '1';
268         wait for 5 us; -- tempo affinche il dato venga mandato in memoria
269 
270         -- LETTURA DELLO STATUS REGISTER: r03
271         -- CMD 00100001 '21' (r)
272         -- ADD 00000011 '03'
273         MOSI_rd <= "00100001" & "00000011";
274         nSS_spi <= '0';
275         wait for 1 us;
276 
277         for i in 0 to 15 loop --invio dati su mosi
278             s_mosi  <= MOSI_rd(15 - i);
279             s_clock <= '1';
280             wait for 500 ns;
281             s_clock <= '0';
282             wait for 500 ns;
283         end loop;
284 
285         for i in 0 to 18 loop -- aspetto il dato sul MISO
286             s_clock <= '1';
287             if (s_miso /= 'Z') then
288                 MISO_rd(cnt_bit) <= s_miso;
289                 cnt_bit := cnt_bit - 1;
290             end if;
```

```
            wait for 500 ns;
            s_clock <= '0';
            wait for 500 ns;
        end loop;

        cnt_bit := 15;
        nSS_spi <= '1';
        wait for 5 us;

    end process RD_WR_process;

end architecture;
```

E.2 Testbench con I/O da file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;

entity tb_crc_complete is
end tb_crc_complete;

architecture behavioral of tb_crc_complete is

    -- definizione segnali interni
    signal clock   : std_logic := '0';              -- main clock
    signal s_clock : std_logic := '0';              -- system clock
    signal nSS_spi : std_logic := '1';              -- slave select
    signal reset   : std_logic := '0';              -- reset esterno
    signal s_mosi  : std_logic := '1';              -- MOSI seriale
    signal s_miso  : std_logic := 'Z';              -- MISO seriale
    signal MOSI_wr : std_logic_vector(31 downto 0); -- vettore dati per WR
    signal MOSI_rd : std_logic_vector(15 downto 0); -- vettore dati per RD
    signal MISO_rd : std_logic_vector(15 downto 0); -- vettore restituito su MISO

    -- definizione file di I/O
    file file_INPUT_SINGLE  : text;
    file file_INPUT_LONG    : text;
    file file_OUTPUT_SINGLE : text;
    file file_OUTPUT_LONG   : text;

    -- dichiarazione UUT
    component top_level is
        port (
            CK, SCK : in std_logic;
            nSS     : in std_logic;
            RST     : in std_logic;
            MOSI    : in std_logic;
            MISO    : out std_logic
        );
    end component;

begin

    -- istanza UUT
```

```
    TB_CRC_CALCULATOR : top_level
    port map(
        CK => clock, SCK => s_clock,
        RST => reset, nSS => nSS_spi,
        MOSI => s_mosi, MISO => s_miso
    );

    -- Process CK (clock FPGA, periodo 100 ns, f=10 MHz)
    CK_process : process
    begin
        wait for 50 ns;
        clock <= not clock;
    end process CK_process;

    -- Process di lettura e scrittura con I/O su file
    RD_WR_process : process
        variable v_ILINE  : line; -- riga file di input
        variable v_OLINE  : line; -- riga file di output
        variable v_CMD_WR : std_logic_vector(31 downto 0);
        variable v_CMD_RD : std_logic_vector(15 downto 0);
        variable cnt_bit  : integer := 15;

    begin

        wait for 100 ns;
        reset <= '1';
        wait for 100 ns;
        reset <= '0';

        --*****************************************************************************
        --* Calcolo di CRC di singole parole di 16 bit
        --*****************************************************************************

        -- Apro file di I/O in modalità di lettura/scrittura
        file_open(file_INPUT_SINGLE, "input_commands_single.txt", read_mode);
        file_open(file_OUTPUT_SINGLE, "output_results_single.txt", write_mode);

        -- Leggo da input_commands_single.txt
        while not endfile(file_INPUT_SINGLE) loop

            -- scrittura
            readline(file_INPUT_SINGLE, v_ILINE);
            read(v_ILINE, v_CMD_WR);
            MOSI_wr <= v_CMD_WR;
            nSS_spi <= '0';
            wait for 100 ns;

            for i in 0 to 31 loop
                s_mosi  <= MOSI_wr(31 - i);
                s_clock <= '1';
                wait for 500 ns;
                s_clock <= '0';
                wait for 500 ns;
            end loop;

            wait for 200 ns;
            nSS_spi <= '1';
            wait for 5 us;

```

```
102             -- lettura
103             MOSI_rd <= "00100001" & "00000001";
104             nSS_spi <= '0';
105             wait for 1 us;
106 
107             for i in 0 to 15 loop
108                 s_mosi  <= MOSI_rd(15 - i);
109                 s_clock <= '1';
110                 wait for 500 ns;
111                 s_clock <= '0';
112                 wait for 500 ns;
113             end loop;
114 
115             for i in 0 to 18 loop
116                 s_clock <= '1';
117                 if (s_miso /= 'Z') then
118                     MISO_rd(cnt_bit) <= s_miso;
119                     cnt_bit := cnt_bit - 1;
120                 end if;
121                 wait for 500 ns;
122                 s_clock <= '0';
123                 wait for 500 ns;
124             end loop;
125 
126             cnt_bit := 15;
127             nSS_spi <= '1';
128             wait for 2 us;
129 
130             -- Scrivo in output_results.txt
131             write(v_OLINE, MISO_rd, right, 16);
132             writeline(file_OUTPUT_SINGLE, v_OLINE);
133 
134             -- reset
135             MOSI_wr <= "00100000" & "00000010" & "0000000000000001";
136             nSS_spi <= '0';
137             wait for 100 ns;
138 
139             for i in 0 to 31 loop
140                 s_mosi  <= MOSI_wr(31 - i);
141                 s_clock <= '1';
142                 wait for 500 ns;
143                 s_clock <= '0';
144                 wait for 500 ns;
145             end loop;
146 
147             wait for 200 ns;
148             nSS_spi <= '1';
149             wait for 5 us;
150 
151         end loop;
152 
153         -- Chiudo i file di I/O
154         file_close(file_INPUT_SINGLE);
155         file_close(file_OUTPUT_SINGLE);
156 
157         --*************************************************************************
158         --* Calcolo di CRC di messaggi lunghi
159         --*************************************************************************
160 
```

```
        -- Apro file di I/O in modalità di lettura/scrittura
        file_open(file_INPUT_LONG, "input_commands_long.txt", read_mode);
        file_open(file_OUTPUT_LONG, "output_results_long.txt", write_mode);

        -- Leggo da input_commands_long.txt
        while not endfile(file_INPUT_LONG) loop

            -- scrittura
            readline(file_INPUT_LONG, v_ILINE);
            read(v_ILINE, v_CMD_WR);
            MOSI_wr <= v_CMD_WR;
            nSS_spi <= '0';
            wait for 100 ns;

            for i in 0 to 31 loop
                s_mosi  <= MOSI_wr(31 - i);
                s_clock <= '1';
                wait for 500 ns;
                s_clock <= '0';
                wait for 500 ns;
            end loop;

            wait for 200 ns;
            nSS_spi <= '1';
            wait for 5 us;

            -- lettura
            MOSI_rd <= "00100001" & "00000001";
            nSS_spi <= '0';
            wait for 1 us;

            for i in 0 to 15 loop
                s_mosi  <= MOSI_rd(15 - i);
                s_clock <= '1';
                wait for 500 ns;
                s_clock <= '0';
                wait for 500 ns;
            end loop;

            for i in 0 to 18 loop
                s_clock <= '1';
                if (s_miso /= 'Z') then
                    MISO_rd(cnt_bit) <= s_miso;
                    cnt_bit := cnt_bit - 1;
                end if;
                wait for 500 ns;
                s_clock <= '0';
                wait for 500 ns;
            end loop;

            cnt_bit := 15;
            nSS_spi <= '1';
            wait for 2 us;

            -- Scrivo in output_results.txt
            write(v_OLINE, MISO_rd, right, 16);
            writeline(file_OUTPUT_LONG, v_OLINE);

        end loop;
```

```
220
221         -- Chiudo i file di I/O
222         file_close(file_INPUT_LONG);
223         file_close(file_OUTPUT_LONG);
224
225         wait;
226     end process;
227
228 end architecture;
```

E.3 Automatizzazione della simulazione con C++

E.3.1 Classe Tools

```
1  #include <iostream>
2  #include <fstream>
3  #include <cstring>
4  #include <string>
5  #include <vector>
6  #include <algorithm>
7
8  using namespace std;
9
10 #include "Tools.hpp"
11 #include "Converter.hpp"
12
13 // Costruttore
14 Tools::Tools(string p, int d, string c)
15     : poly{p}, deg{d}, crc{c}
16 {
17     vector_message.clear();
18     vector_crc.clear();
19 }
20
21 // Distruttore
22 Tools::~Tools() {}
23
24 // Imposta il polinomio generatore di CRC
25 // @param generator: polinomio generatore (11021 CRC-16-CCITT)
26 // @param degree: grado polinomio generatore (16 per  CRC-16-CCITT)
27 // @param init_value: valore di inizializzazione del LFSR (0 per XMODEM)
28 void Tools::set_crc(string generator, int degree, int init_value)
29 {
30     poly = generator;
31     deg = degree;
32     init = init_value;
33
34     // individuo i coefficienti del polinomio generatore per capire quando usare l'xor
35     coefficients.clear();
36     for (int i = 0; i < generator.length(); i++)
37     {
38         if (generator.substr(i, 1) == "1" && i != 0 && i != generator.length() - 1)
39             coefficients.push_back(i - 1);
40     }
41 }
42
43 // Calcola il CRC di una parola di 16 bit e la scrive su file
```

```
// @param message: messaggio di cui calcolare il CRC in formato binario
void Tools::calc_crc(string message, ofstream &outFile)
{
    Converter C;
    vector_message.clear();
    vector_crc.clear();
    vector<int> old_crc;

    // inserisco il crc nel vettore
    for (int i = 0; i < deg; i++)
    {
        vector_crc.push_back(init);
    }

    message += "0000000000000000";

    // inserisco il messaggio nel vettore
    for (int i = 0; i < message.length(); i++)
    {
        vector_message.push_back(stoi(message.substr(i, 1)));
    }

    // calcolo il CRC usando la stessa tecnica implementata a livello circuitale
    for (int i = 0; i < message.length(); i++)
    {
        old_crc = vector_crc;
        int msg = vector_message.front();             // ingresso LFSR = MSB messaggio
        vector_message.erase(vector_message.begin()); // rimuovo MSB dal messaggio

        for (int j = deg - 1; j >= 0; j--)
        {
            if (j == deg - 1)
                vector_crc[j] = msg ^ old_crc[0];
            else if (find(coefficients.begin(), coefficients.end(), j) !=
            ↪  coefficients.end())
                vector_crc[j] = old_crc[j + 1] ^ old_crc[0];
            else
                vector_crc[j] = old_crc[j + 1];
        }
    }

    for (int i = 0; i < deg; i++)
    {
        outFile << vector_crc[i];
    }
    outFile << endl;
}
```

E.3.2 Classe Converter

```
#include <iostream>
#include <fstream> // per il file processing
#include <string>  // per creazione e manipolazione stringhe
#include <cstring> // per manipolazione di stringhe C-like
#include <cmath>   // per l'elevazione a potenza con pow(a,b)

#include "Converter.hpp"
```

```

using namespace std;

// Costruttore
Converter::Converter(int n, int r)
    : number{n}, result{} {}

// Distruttore
Converter::~Converter() {}

// Converte un numero da intero a binario (con parallelismo a n bit) e scrive il risultato in un file
void Converter::intToBin(string conv_number, unsigned int n, ofstream &outFile)
{
    int digit;                  // singola cifra del numero da convertire
    number = stoi(conv_number); // numero (intero) da convertire

    // Sfruttiamo l'overloading dell'operatore >> (shift right)
    // Ad ogni iterazione shiftiamo number a dx di i posizioni (ovvero calcoliamo number % 2^i) e mettiamo il risultato in bitwise and con 1
    for (int i = n - 1; i >= 0; i--)
    {
        digit = (number >> (i)) & 1;
        outFile << digit;
    }
}

// Converte un numero da binario a intero
int Converter::binToInt(string conv_number)
{
    result = 0;
    int bit_number = conv_number.length();
    for (int i = 0; i < bit_number; i++)
    {
        string bit = conv_number.substr(i, 1);          // estraggo l'i-esimo bit
        int bit_int = stoi(bit);                        // trasformo il bit in un numero intero
        result += pow(2, bit_number - 1 - i) * bit_int; // incremento il risultato
    }
    return result;
}
```

E.3.3 Classe Simulation

```
#include <iostream>
#include <fstream>    // per file processing
#include <sstream>    // per trattare le stringhe come stream di dati
#include <filesystem> // per verificare l'esistenza dei file
#include <cstdlib>    // per usare comandi shell
#include <string>     // per manipolazione stringhe
#include <cstring>    // per manipolazione stringhe C-like
#include <vector>     // per manipolazione vettori
#include <cmath>      // per funzioni matematiche
#include <iomanip>    // per formattazione I/O

#include "Tools.hpp"
#include "Converter.hpp"
```

```cpp
#include "Simulation.hpp"

using namespace std;

// Costruttore
Simulation::Simulation(unsigned int c)
    : correct{c} {}

// Distruttore
Simulation::~Simulation() {}

// Genero file con i comandi per lo slave SPI
// @param iFName = nome file contenente comandi di interazione con l'interfaccia a registri
// @param ref_FName = nome file contenente i dati generati (per confronto con i risultati di Modelsim)
void Simulation::generateCommands(string iFName)
{
    Converter C;

    // check esistenza file (se non esistono li creo con una chiamata a system)
    if (!filesystem::exists(iFName))
        system(("touch " + iFName).c_str());

    // apro il file di input
    ofstream input_oF(iFName);

    // scrivo i comandi di scrittura e lettura generando casualmente 100 parole di 16 bit di cui calcolare il CRC
    if (input_oF) // se l'apertura è andata a buon fine
    {
        for (int word = 0; word < 100; word++)
        {
            int din = rand() % 65536; // dato da scrivere

            // scrivo comando di scrittura di DIN nell'indirizzo 0 (Data In Register)
            C.intToBin(to_string(32), 8, input_oF);   // comando di scrittura
            C.intToBin(to_string(0), 8, input_oF);    // indirizzo
            C.intToBin(to_string(din), 16, input_oF); // dato
            input_oF << endl;
        }

        // chiudo il file di input
        input_oF.close();
    }
}

// Calcolo il CRC di singole parole
void Simulation::generateReference_CRCsingle(string iFName, string ref_FName)
{
    // apro il file di input in lettura e il file dei CRC di riferimento in scrittura
    ifstream input_iF(iFName);
    ofstream ref_F(ref_FName);
    string line;

    Tools CRC_calculator;

    // imposto parametri per calcolo CRC-16-CCITT XMODEM
    // generatore = x^16 + x^12 + x^5 + 1, LFSR inizializzato a 0
```

```
70      CRC_calculator.set_crc("10001000000100001", 16, 0);
71
72      if (input_iF && ref_F)
73      {
74          while (getline(input_iF, line))
75          {
76              string message = line.substr(16, 16); // nuovo messaggio
77              CRC_calculator.calc_crc(message, ref_F);
78          }
79      }
80
81      // chiudo i file
82      input_iF.close();
83      ref_F.close();
84  }
85
86  void Simulation::generateReference_CRClong(string iFName, string ref_FName)
87  {
88      // apro il file di input in lettura e il file dei CRC di riferimento in scrittura
89      ifstream input_iF(iFName);
90      ofstream ref_F(ref_FName);
91      string line;
92
93      Tools CRC_calculator;
94
95      // imposto parametri per calcolo CRC-16-CCITT XMODEM
96      // generatore = x^16 + x^12 + x^5 + 1, LFSR inizializzato a 0
97      CRC_calculator.set_crc("10001000000100001", 16, 0);
98      string message;
99      if (input_iF && ref_F)
100     {
101         while (getline(input_iF, line))
102         {
103             message += line.substr(16, 16); // aggiungo nuova parola
104             CRC_calculator.calc_crc(message, ref_F);
105         }
106     }
107
108     // chiudo i file
109     input_iF.close();
110     ref_F.close();
111 }
112
113 // Esecuzione simulazione mediante chiamata a system
114 void Simulation::run(string fileCompilazione)
115 {
116     system(("vsim -c -do " + fileCompilazione).c_str()); // lancio la simulazione
117 }
118
119 // Controlla la correttezza dei risultati
120 unsigned int Simulation::report(string risultati_tb, string risultati_ref)
121 {
122     string line_tb, line_ref; // righe dei due file
123     int cnt_lines_tb = 0;     // contatore di riga del file generato dalla tb
124     int cnt_lines_ref = 0;    // contatore di riga del file di riferimento
125     int tot_correct = 0;      // numero totale di righe corrette all'interno del file
    ↪  generato dalla tb
126
127     // apro i file in lettura
```

```
128     ifstream tbF(risultati_tb);
129     ifstream ref_F(risultati_ref);
130 
131     while (tbF.good() && ref_F.good())
132     {
133         // estraggo una riga da ognuno dei due file
134         if (getline(tbF, line_tb) && getline(ref_F, line_ref))
135         {
136             // se i risultati della tb e del file di riferimento sono uguali incremento
                ↪  il contatore di righe corrette
137             if (line_tb == line_ref)
138             {
139                 tot_correct++;
140             }
141             // se i risultati sono diversi, esco dal ciclo (non ho più bisogno di
                ↪  controllare le righe restanti)
142             else
143             {
144                 cout << "Errore con il messaggio inviato alla riga " << cnt_lines_tb <<
                    ↪  endl;
145                 break;
146             }
147 
148             // incremento i contatori di riga
149             cnt_lines_tb++;
150             cnt_lines_ref++;
151         }
152     }
153 
154     // chiudo i file
155     tbF.close();
156     ref_F.close();
157 
158     // stabilisco il valore del flag che mi dice se la simulazione è andata a buon fine
159     if ((cnt_lines_tb == cnt_lines_ref) && (tot_correct == cnt_lines_ref))
160         correct = 1;
161     else
162         correct = 0;
163 
164     return correct;
165 }
```

E.3.4 Main

```
1  #include <iostream>
2  #include <fstream>    // per file processing
3  #include <sstream>    // per trattare le stringhe come stream di dati
4  #include <filesystem> // per verificare l'esistenza dei file
5  #include <cstdlib>    // per usare comandi shell
6  #include <string>     // per manipolazione stringhe
7  #include <cstring>    // per manipolazione stringhe C-like
8  #include <vector>     // per manipolazione vettori
9  #include <cmath>      // per funzioni matematiche
10 #include <iomanip>    // per formattazione I/O
11 
12 #include "Tools.hpp"
13 #include "Converter.hpp"
```

```cpp
#include "Simulation.hpp"

using namespace std;

int main(int argc, char **argv)
{

    int ret = 0;          // variabile per il return
    Simulation Simulator; // oggetto della classe Simulation per l'automatizzazione della simulazione
    Tools CRC_calculator;

    /*****************************************************************************/
    /*        Inizializzazione degli oggetti necessari alla gestione dei file    */
    /*****************************************************************************/

    string tbFileName = "tb_crc_complete.vhd"; // testbench
    string compileFileName = "compile.do";     // file con le info per la simulazione
    string input_singleFileName = "input_commands_single.txt";
    string input_longFileName = "input_commands_long.txt";
    string ref_singleFileName = "ref_single.txt";
    string ref_longFileName = "ref_long.txt";
    string tb_singleFileName = "output_results_single.txt";
    string tb_longFileName = "output_results_long.txt";

    // check esistenza testbench
    if (!filesystem::exists(tbFileName))
    {
        cerr << "Errore! La testbench " << tbFileName << " non esiste." << endl;
        ret = 1;
    }

    // check esistenza file per la compilazione
    if (!filesystem::exists(compileFileName))
    {
        cerr << "Errore! Il file per la compilazione " << compileFileName << " non esiste." << endl;
        ret = 1;
    }

    /*****************************************************************************/
    /*                  Calcolo del CRC di singole parole da 16 bit              */
    /*****************************************************************************/

    // generazione file di scrittura
    Simulator.generateCommands(input_singleFileName);
    // generazione file di riferimento
    Simulator.generateReference_CRCsingle(input_singleFileName, ref_singleFileName);

    /*****************************************************************************/
    /*                      Calcolo del CRC di messaggi lunghi                   */
    /*****************************************************************************/

    // generazione file di scrittura
    Simulator.generateCommands(input_longFileName);
    // generazione file di riferimento
    Simulator.generateReference_CRClong(input_longFileName, ref_longFileName);

    // simulazione automatizzata
```

```
71      cout << endl
72           << "**********************************************************************"
73           << endl;
74      cout << "Inizio Simulazione Modelsim" << endl;
75      Simulator.run(compileFileName);
76      cout << endl
77           << "Fine Simulazione Modelsim" << endl;
78      cout << "**********************************************************************"
79           << endl
80           << endl;
81
82      // controllo risultati
83      cout << "Calcolo CRC di singole parole di 16 bit:" << endl;
84      int single_OK = Simulator.report(ref_singleFileName, tb_singleFileName);
85      if (single_OK == 1)
86      {
87          cout << "I risultati della simulazione corrispondono a quelli generati con lo
            ↪  script C++." << endl
88               << endl;
89      }
90      else
91      {
92          cout << endl
93               << "Non tutti i risultati della simulazione corrispondono a quelli generati
                 ↪  con lo script C++." << endl
94               << endl;
95      }
96
97      cout << endl
98           << "Calcolo CRC di messaggi arbitrarimente lunghi (fino a 100 parole):" <<
             ↪  endl;
99      int long_OK = Simulator.report(ref_longFileName, tb_longFileName);
100     if (long_OK == 1)
101     {
102         cout << "I risultati della simulazione corrispondono a quelli generati con lo
            ↪  script C++." << endl
103              << endl;
104     }
105     else
106     {
107         cout << endl
108              << "Non tutti i risultati della simulazione corrispondono a quelli generati
                 ↪  con lo script C++." << endl
109              << endl;
110     }
111
112     if (single_OK && long_OK)
113     {
114         cout << "Il calcolatore di CRC-16-CCITT/XMODEM funziona correttamente! :)" <<
            ↪  endl;
115     }
116     else
117     {
118         cout << "Il calcolatore di CRC-16-CCITT/XMODEM non funziona correttamente. :("
            ↪  << endl;
119     }
120     cout << endl;
121     return ret; // punto di uscita dal programma
122 }
```

E.4 Test su VirtLab

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library lpm;
use lpm.lpm_components.all;
library altera_mf;
use altera_mf.altera_mf_components.all;

entity user is
  port (
    -- Main clock inputs
    mainClk : in std_logic;
    slowClk : in std_logic;
    -- Main reset input
    reset : in std_logic;
    -- MCU interface (UART, I2C)
    mcuUartTx : in std_logic;
    mcuUartRx : out std_logic;
    mcuI2cScl : in std_logic;
    mcuI2cSda : inout std_logic;
    -- Logic state analyzer/stimulator
    lsasBus : inout std_logic_vector(31 downto 0);
    -- Dip switches
    switches : in std_logic_vector(7 downto 0);
    -- LEDs
    leds : out std_logic_vector(3 downto 0)
  );
end user;

architecture behavioural of user is

  signal clk     : std_logic;
  signal pllLock : std_logic;

  signal lsasBusIn  : std_logic_vector(31 downto 0);
  signal lsasBusOut : std_logic_vector(31 downto 0);
  signal lsasBusEn  : std_logic_vector(31 downto 0) := (others => '0');

  signal mcuI2cDIn  : std_logic;
  signal mcuI2CDOut : std_logic;
  signal mcuI2cEn   : std_logic := '0';

  component myAltPll
    port (
      areset : in std_logic := '0';
      inclk0 : in std_logic := '0';
      c0     : out std_logic;
      locked : out std_logic
    );
  end component;

  component top_level is
    port (
      CK   : in std_logic;
      SCK  : in std_logic;
      nSS  : in std_logic;
```

```
57        RST  : in std_logic;
58        MOSI : in std_logic;
59        MISO : out std_logic
60      );
61    end component;
62
63  begin
64
65    --*************************************************************************************
66    --* Main clock PLL
67    --*************************************************************************************
68
69    myAltPll_inst : myAltPll port map(
70      areset => reset,
71      inclk0 => mainClk,
72      c0     => clk,
73      locked => pllLock
74    );
75
76    --*************************************************************************************
77    --* LEDs
78    --*************************************************************************************
79
80    leds(2 downto 0) <= switches(2 downto 0);
81    lsasbusEn (14)   <= '1';
82
83    --*************************************************************************************
84    --*    lsasBus  : inout std_logic_vector( 31 downto 0 )
85    --*************************************************************************************
86
87    lsasBusIn <= lsasBus;
88
89    lsasBus_tristate :
90    process (lsasBusEn, lsasBusOut) is
91    begin
92      for index in 0 to 31 loop
93        if lsasBusEn(index) = '1' then
94          lsasBus(index) <= lsasBusOut (index);
95        else
96          lsasBus(index) <= 'Z';
97        end if;
98      end loop;
99    end process;
100
101   leds(3) <= '1';
102
103   crc : top_level
104   port map(
105     CK   => mainClk,
106     SCK  => lsasbus(13),
107     nSS  => lsasbus(12),
108     RST  => switches(0),
109     MOSI => lsasbus(15),
110     MISO => lsasbusOut(14)
111   );
112
113 end behavioural;
```

Riferimenti bibliografici

[1] Algebra dei campi finiti:
`https://www.doc.ic.ac.uk/~mrh/330tutor/ch04s04.html`

[2] International Telecommunication Union (ITU) website:
`https://www.itu.int/en/Pages/default.aspx`

[3] Catalogue of parametrised CRC algorithms with 16 bits:
`https://reveng.sourceforge.io/crc-catalogue/16.htm`

[4] ITU-T Recommendation V.41 (11/88):
`https://www.itu.int/rec/T-REC-V.41/en`

[5] Williams, R. N. (Copyright), *A painless guide to CRC error detection algorithms* (1993), version 3:
`http://www.ross.net/crc/download/crc_v3.txt`

[6] Dizon, R., *CpE 405 Final Project: CRC-16-CCITT Hardware Implementation*:
`http://www.ee.unlv.edu/~regent/NVSG/CRC16.pdf`

[7] On-line CRC calculation and free library by Lammert Bies:
`https://www.lammertbies.nl/comm/info/crc-calculation`