CAPITOLO 5

La comunicazione su *backplane*

Questo capitolo descrive il sistema di trasmissione dei dati sul *backplane* tra ROD e le schede che ricevono i dati del trigger e del rivelatore, ossia Sector Logic e concentratore ottico. In particolare il sistema è basato sull'impiego di dispositivi che trasmettono informazioni su *backplane* in modo pseudo-seriale. Si discutono le motivazioni che hanno portato alla definizione dell'architettura, il funzionamento di tali dispositivi e i problemi che possono presentarsi nella trasmissione dei dati. Inoltre viene descritto il progetto, la realizzazione in laboratorio del sistema per il collaudo di tali dispositivi e i risultati ottenuti.

Il collaudo è stato eseguito presso i Laboratori di ricerca del Dipartimento di Scienze Fisiche dell'Università *'Federico II'* di Napoli e della Sezione di Napoli dell'INFN. In particolare, è stato verificato che l'intero sistema di trasmissione rispettasse i requisiti imposti dal trigger di livello 1 dello spettrometro per muoni per l'apparato ATLAS.

La struttura del crate

Come già descritto nei capitoli precedenti, per il sistema di acquisizione dati dello spettrometro per muoni dell'apparato ATLAS è stata ideata una struttura modulare. Essa consiste di due schede ricevitrici RX (o concentratori ottici) e due schede SECTOR LOGIC, alloggiate in un crate VME: queste quattro schede comunicano con un ROD tramite due *backplane* dedicati, come mostrato in figura 5.1. Ciascun ROD è responsabile della gestione di uno dei 32 settori in cui è suddiviso lo spettrometro.

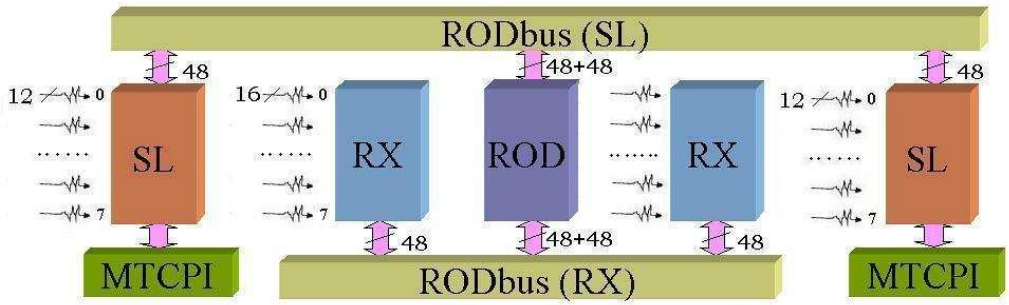


Figura 5.1 - Il ROD comunica con le schede RX e le Sector Logic

Organizzare questa sezione della struttura di acquisizione dati in più schede, piuttosto che in una sola, è una scelta dettata innanzitutto da limitazioni geometriche, dovute alle dimensioni dei ricevitori del link ottico. Una tale architettura consente anche di parallelizzare la logica che gestisce i dati e di diminuire la complessità delle singole schede riducendo la potenza dissipata.

Il problema più evidente che però si presenta in questa situazione è quello di trasferire i dati via *backplane* tra le schede ed il ROD. E' necessario, infatti, trasferire 96 bit dati di ciascuna delle due schede RX, con un numero di connessioni richieste di gran lunga superiore a quelle disponibili sul

backplane VME. Si rende necessario progettare un *backplane* dedicato che garantisca le prestazioni richieste.

La soluzione da noi adottata consiste nell'utilizzare dispositivi che *multiplexano* 48 bit dati su più canali seriali, ad una frequenza di trasmissione multipla dei 40 MHz del clock del sistema di acquisizione di trigger e dati dello spettrometro per muoni.

Tali dispositivi, prodotti dalla National, sono il *DS90CR483* e il *DS90CR484*, rispettivamente trasmettitore (o serializzatore) e ricevitore (o deserializzatore) e trattano i dati come illustrato nello schema in figura 5.2 e come descritto nel paragrafo seguente [7].

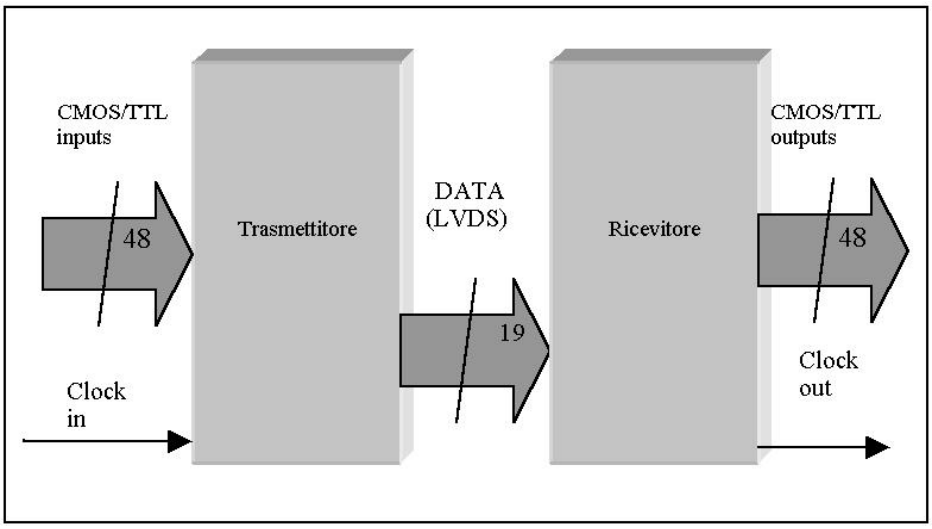


Figura 5.2 - Il percorso dei dati da serializzatore a deserializzatore

I serializzatori

Il trasmettitore accetta in ingresso parole dati a 48 bit, che vengono *multiplexate* su 8 coppie di linee, secondo lo schema mostrato in figura 5.3, a cui va aggiunta un'altra coppia per la trasmissione del segnale di sincronizzazione (*clock*). Questi 9 segnali vengono trasmessi secondo lo standard LVDS su coppie differenziali e con una differenza di tensione

tipica tra i due livelli logici di ~ 350 mV. Una più accurata descrizione dello standard LVDS è fatta nel paragrafo successivo.

L'idea di base e' quella di trasmettere i dati su connessioni seriali multiple. In ciascun periodo di clock esterno, su ogni connessione vengono inviati sei bit della parola dati ed un bit di *DC Balance*, il cui scopo è descritto nel paragrafo successivo. La frequenza del clock di trasmissione dovrà di conseguenza essere 7 volte maggiore del clock esterno, e in particolare, per il clock di 40 MHz, richiesto per il sistema di acquisizione dati per l'esperimento ATLAS, la frequenza di trasmissione sarà di 280 Mbit/s. I dispositivi utilizzati acquisiscono dati in parallelo, li trasferiscono nella forma "*pseudo-seriale*" descritta, solamente su 9 coppie, e li restituiscono in parallelo. Si riduce così il numero totale di interconnessioni e gli effetti di interferenza sono limitati dalla trasmissione differenziale.

Il ricevitore accetta in ingresso i nove canali seriali LVDS e ricostruisce la parola originale a 48 bit, sincronizzandola con il segnale di clock del trasmettitore.

Canale	DATI [47:0]						
# 0	Dato 0	Dato 1	Dato 2	Dato 3	Dato 4	Dato 5	DCBAL
# 1	Dato 8	Dato 9	Dato 10	Dato 11	Dato 12	Dato 13	DCBAL
# 2	Dato 16	Dato 17	Dato 18	Dato 19	Dato 20	Dato 21	DCBAL
# 3	Dato 6	Dato 7	Dato 14	Dato 15	Dato 22	Dato 23	DCBAL
# 4	Dato 24	Dato 25	Dato 26	Dato 27	Dato 28	Dato 29	DCBAL
# 5	Dato 32	Dato 33	Dato 34	Dato 35	Dato 36	Dato 37	DCBAL
# 6	Dato 40	Dato 41	Dato 42	Dato 43	Dato 44	Dato 45	DCBAL
# 7	Dato 30	Dato 31	Dato 38	Dato 39	Dato 46	Dato 47	DCBAL

Figura 5.3 - L'assegnazione dei 48 bit dati ai canali di trasmissione dei serializzatori

Le caratteristiche elettriche

Come già accennato in precedenza, nello standard LVDS (*Low Voltage Differential Signaling*) il segnale logico viene trasmesso in differenziale su due linee e con una differenza di tensione tipica di poche centinaia di mV. Da un lato, questo garantisce tempi di commutazione e dissipazione di potenza inferiori agli altri standard logici, dall'altro la dinamica limitata ha come potenziale svantaggio quello di avere una bassa immunità al rumore.

Lo stadio di ingresso di un ricevitore LVDS è costituito da un amplificatore differenziale, che ha la funzione di amplificare la differenza tra due segnali.

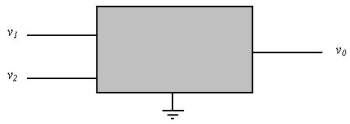


Figura 5.4 Un generico ricevitore differenziale

La figura 5.4 rappresenta un dispositivo lineare attivo con due tensioni di ingresso, v_1 e v_2, e la tensione di uscita v_0, ciascuna misurata rispetto alla massa. In un amplificatore differenziale ideale la tensione di uscita v_0 è espressa come:

$$v_0 = A_d\,(v_1 - v_2)$$

dove A_d è l'amplificazione dell'amplificatore differenziale. Si può così notare come tutti i segnali comuni ai due ingressi non abbiano effetti sulla tensione di uscita. In un amplificatore differenziale reale, però, la tensione di uscita non è solo funzione di v_d , segnale differenza tra le due tensioni di ingresso, ma anche del loro valore medio v_c (chiamato *segnale di modo comune*), dove

$$v_d = v_1 - v_2$$

$$v_c = \tfrac{1}{2}\,(v_1 + v_2)$$

Esprimiamo le tensioni di uscita come combinazione lineare delle tensioni di ingresso $v_0 = A_1 v_1 + A_2 v_2$, dove A_1 (A_2) è l'amplificazione di tensione dell'ingresso 1(2) nell'ipotesi che l'ingresso 2(1) sia connesso a massa. Poichè $v_1 = v_c + \frac{1}{2} v_d$ e $v_2 = v_c - \frac{1}{2} v_d$ otteniamo

$$v_0 = A_d v_d + A_c v_c, \qquad \text{(eq. 5-1)}$$

con $A_d \equiv \frac{1}{2}(A_1 - A_2)$ ed $A_c \equiv A_1 + A_2$.

Evidentemente sarà necessario che sia A_d elevato, mentre A_c sia piccolo, ed idealmente uguale a zero. Un buon fattore di merito per l'amplificatore differenziale è, allora, il *rapporto di reiezione di modo comune*, definito come

$$\rho \equiv \left| \frac{A_d}{A_c} \right|$$

Poichè dall'equazione 5.1 deriva $v_0 = A_d v_d \left(1 + v_c / \rho \cdot v_d \right)$, quanto più è grande ρ rispetto a v_c/v_d, tanto migliore sarà l'amplificatore differenziale.

La tecnologia LVDS presenta il notevole vantaggio di essere poco sensibile rispetto ad un "rumore" di modo comune. Infatti, a causa della trasmissione differenziale dei dati, la presenza di un disturbo può essere individuata e rimossa dallo stadio ricevitore dello standard LVDS, come mostrato in figura 5.5. L'immunità al rumore di modo comune per lo standard LVDS risulta essere di ~ ± 1 V.

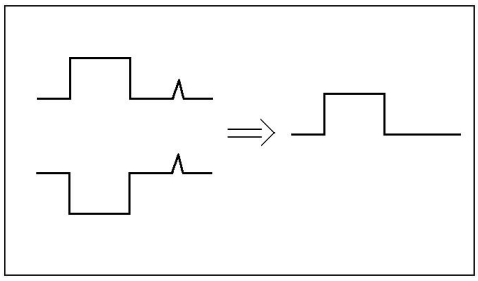


Figura 5.5 - Eliminazione di un disturbo sulle due linee dello standard LVDS

Una successione arbitraria di "0" ed "1" ha un valore medio che cambia in funzione del tempo. Inoltre, quando il segnale viaggia su una linea reale, i parametri tipici della linea, quali resistenza e capacità, influenzano la forma del segnale trasmesso in funzione della frequenza. A causa di ciò il segnale ricostruito dal ricevitore può essere diverso da quello inviato, introducendo errori di trasmissione.

Per chiarire la situazione, schematizziamo la linea di trasmissione reale come un circuito RC serie, utilizzando un modello semplificato a costanti concentrate, come mostrato nella figura 5.6. Questo modello semplificato permette di studiare il comportamento della linea di trasmissione limitatamente al problema in questione. Tuttavia non permette di analizzare ulteriori problematiche quali fenomeni di induzione o riflessioni, che richiedono modelli a costanti distribuite.

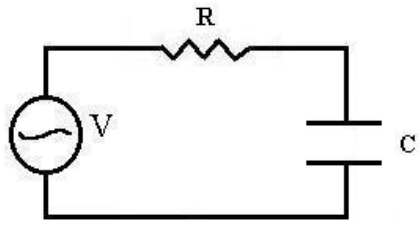


Figura 5.6 Circuito equivalente semplificato di una linea
di trasmissione a costanti concentrate

Com'è noto, la tensione prelevata ai capi del condensatore C, nel caso di un gradino di tensione in ingresso di ampiezza V_0, risulta avere un andamento nel tempo descritto dall'equazione $V(t)=V_0(1-e^{-t/RC})$.

Nel caso di un impulso in ingresso, invece, si distinguono due casi:

per tempi inferiori a t_p (durata dell'impulso) $V(t) = V_0(1- e^{-t/RC})$;

per tempi successivi a t_p $V(t) = V_T e^{-(t-tp)/RC}$ dove $V_T =V_0(1- e^{-tp/RC})$;

Analizzando il caso dell'onda quadra simmetrica, ossia $T_1=T_2$ in figura 5.7, si può ricavare l'espressione per le uscite durante i livelli di ingresso alto e basso nel primo periodo dell'onda quadra, rispettivamente:

$$V_{o1}= V' - (V'-V_1)e^{-t/RC} \qquad \text{(eq. 5-2)}$$

$$V_{o2}= V'' - (V''-V_2)e^{-(t-T1)/RC} \qquad \text{(eq. 5-3)}$$

dove V' ed V'' sono i due livelli di tensione in ingresso, mentre V_1 ed V_2 indicano i livelli di tensione raggiunti dall'uscita negli intervalli di tempo T_1 e T_2.

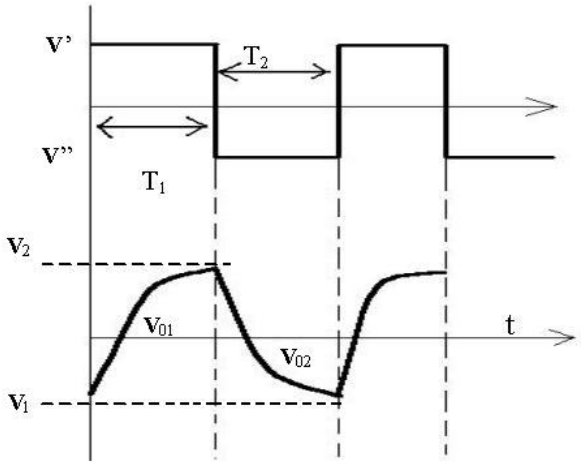


Figura 5.7 Risposta di un circuito RC all'onda quadra

Se l'onda quadra ha un valore medio non nullo, occorrerà sommare questo valore di continua alla tensione in uscita, intorno al quale, a regime, sarà centrato il segnale.

Nel caso in cui l'ingresso è descritto da una sequenza arbitraria di "0" e di "1", la situazione che si presenta sarà complessa. E' possibile, tra le diverse eventualità, la situazione portata come esempio in figura 5.8, in cui il condensatore di uscita non riesce sempre a ritornare al valore iniziale, poichè sopraggiunge un nuovo segnale positivo a caricarlo.

Se, infatti, la costante di tempo τ=RC del circuito è grande rispetto a t_p, un singolo impulso basso non sarà sufficiente a far scaricare abbastanza il condensatore. Spostando il discorso su una linea di trasmissione reale, tale situazione si traduce nel considerare il rapporto tra la durata di ciascun impulso, pari al periodo di clock del sistema di trasmissione e che indichiamo ancora con t_p, e la grandezza RC, caratteristica della linea di trasmissione e del relativo carico.

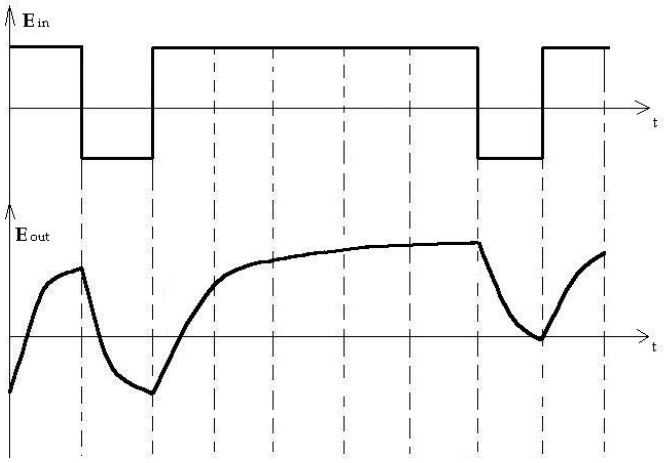


Figura 5.8 In una sequenza arbitraria di bit il valore medio del segnale di uscita cambia in funzione del tempo (*baseline wander*)

Il valore medio del segnale trasmesso dipende dai dati (s*ymbol*) che si sono susseguiti sulla linea e, quindi, dalla "storia" della linea stessa. Una situazione in cui gli errori di un sistema di trasmissione dipendono dal *pattern* trasmesso va sotto il nome di *ISI* (*Inter Symbol Interference*). Il bit di *DC balance* viene aggiunto ai dati da trasmettere per limitare le variazioni del valore medio del segnale trasmesso entro un intervallo compatibile con le specifiche del ricevitore *LVDS*.

Per simulare il comportamento del modello a costanti concentrate della linea di trasmissione, in funzione del rapporto t_p/RC, ho sviluppato un programma in linguaggio C che utilizza le formule (5-2) e (5-3) per

simulare la risposta del circuito RC serie al treno di impulsi riportato in figura 5.9. A sequenze in cui c'è una successione di "0" e di "1", si alternano sequenze di 5 bit uguali consecutivi. Le figure 5.10 e 5.11 mostrano i due esempi di risposta del circuito RC.

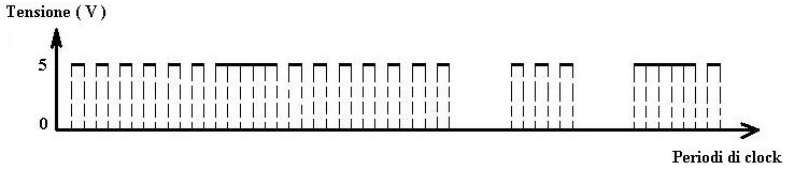


Figura 5.9 Il treno di impulsi utilizzato per simulare un circuito RC

Nel caso in figura 5.10, caratterizzato da un rapporto $t_p/RC = 2$, osserviamo come la presenza di sequenze di 5 impulsi alti o bassi non crei problemi, in quanto i valori di tensione assunti dal segnale sono sempre correttamente interpretati come un valore logico legale.

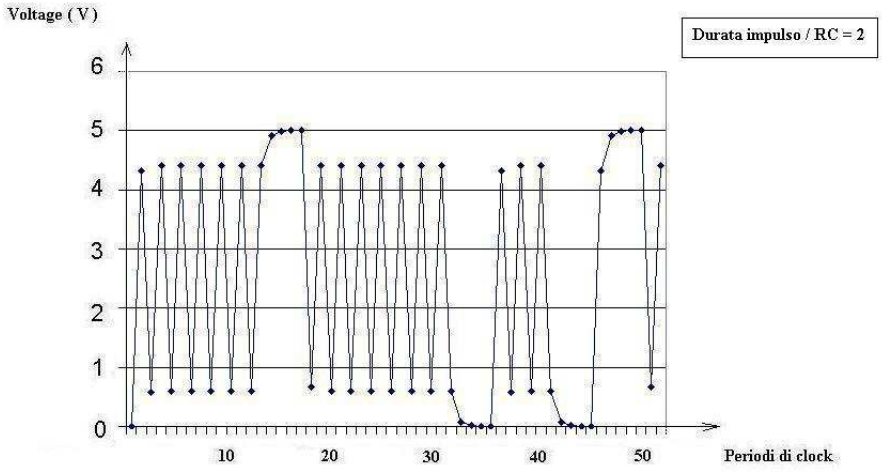


Figura 5.10 Risposta di un circuito RC al treno di impulsi della figura 5.9, nel caso $t_p/RC = 2$

Nella figura 5.11 è riportato il caso in cui $t_p/RC = 1{,}65$. Come si vede, questo rapporto è troppo piccolo per permettere un buon recupero del

corretto livello logico di tensione, corrispondente cioè al valore logico di ingresso. In particolare, dopo i periodi di clock evidenziati in figura si presentano situazioni per cui la tensione di uscita non è riconoscibile come un corretto livello logico. Tuttavia, questi sono gli unici punti di "sofferenza" del circuito, in quanto non si verificano ulteriori errori durante il ciclo di funzionamento in cui in ingresso c'è una alternanza di "0" e di "1".

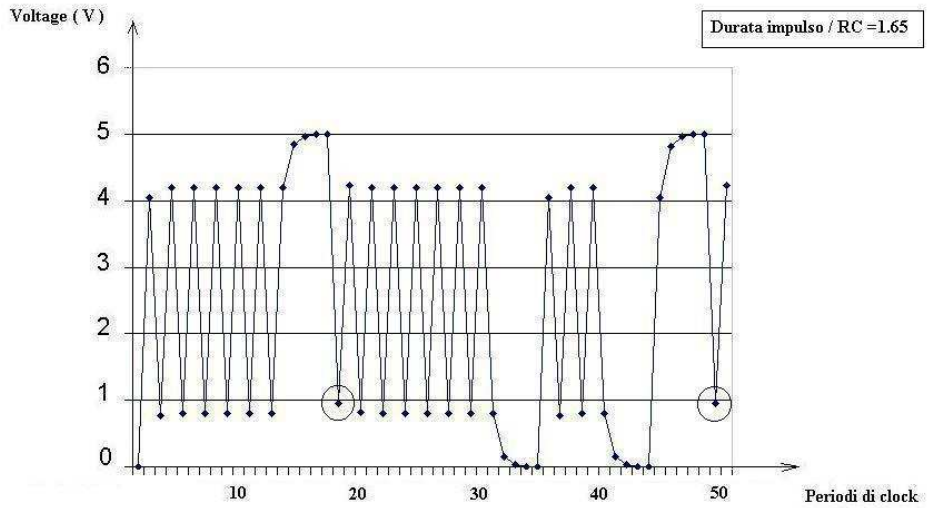


Figura 5.11 Risposta di un circuito RC al treno di impulsi della figura 5.9, nel caso t_p/ RC = 1,65

E' però chiaro che, anche se un circuito col rapporto t_p/RC = 2 non ha problemi per un treno di 5 impulsi uguali consecutivi in ingresso, può non funzionare con sequenze più lunghe. E' per questo motivo che vengono studiati algoritmi di codifica (*scrambling*) in grado di ripristinare un corretto livello di tensione continua.

I serializzatori utilizzano uno di questi algoritmi, generando un bit di *DC balance* ogni 6 trasmessi, su ciascuna linea, secondo il protocollo descritto in seguito. In tal caso, inserendo tale bit aggiuntivo ed eventualmente trasferendo i dati invertiti, il livello di tensione continua viene mantenuto ragionevolmente costante.

Il valore del bit di *DC Balance* (*DCBAL*) è calcolato per ciascuna parola di 6 bit che viene trasmessa su ogni canale [7]. Il bit *DCBAL* si calcola a partire da due caratteristiche della parola, la *current data disparity* e la *running word disparity.*

La *current data disparity* (*CDD*) è ottenuta sottraendo il numero di bit pari a zero dal numero di bit uguali ad uno, nella parola da trasmettere.

Per il calcolo della *running word disparity* (*RWD*) bisogna definire la *modified data disparity* (*MDD*). Essa è pari a (*CDD* -1) se i dati sono inviati non invertiti, altrimenti è pari a (1-*CDD*). La *running word disparity* sarà ottenuta sommando il valore di *MDD* al valore di *RWD* calcolato per la parola precedente. All'inizio la *RWD* può assumere qualsiasi valore compreso tra +7 e –6 e si mantiene sempre in questo intervallo.

Per stabilire se un dato deve essere invertito o meno si usano i valori di *RWD* e *CDD*, secondo la tabella 5.1.

Il valore del bit di *DC Balance* sarà 1 se la parola viene inviata invertita, altrimenti sarà pari a zero.

Running word disparity	Current data disparity	Parola (6 bit)	DC Balance
> 0	> 0	Invertita	1
> 0	≤ 0	Non Invertita	0
< 0	> 0	Non Invertita	0
< 0	≤ 0	Invertita	1
= 0	-----	Invertita	1

Tab 5.1 La tabella per il calcolo del bit di *DCBAL*

Il sistema di collaudo

Allo scopo di collaudare i due serializzatori, è stato realizzato un sistema costituito, come mostrato in figura 5.12, da due schede, una, TX, su cui è alloggiato il trasmettitore e una, RX, su cui è il ricevitore.

Viene utilizzata una *FPGA* con un generatore di sequenze sulla scheda TX, ed una *FPGA* sulla scheda RX capace di riconoscere le sequenze trasmesse e di asserire un segnale di errore, nell' eventualità di una cattiva trasmissione dei dati. La realizzazione delle due schede è stata fatta avvalendosi di un *CAD* elettronico, che consente la progettazione e la simulazione delle funzioni delle schede.

Due versioni basate esclusivamente sull'uso del linguaggio di programmazione *VHDL* (*Very high speed integrated circuit Hardware Design Language*) [8] sono state realizzate e collaudate, allo scopo di poter adattare questo sistema di test anche ad altri componenti.

Il *VHDL* sta sostituendo i più tradizionali sistemi di disegno e progettazione, e quindi le schede nella versione *VHDL* potranno essere usate per il test dei componenti di ultima generazione.

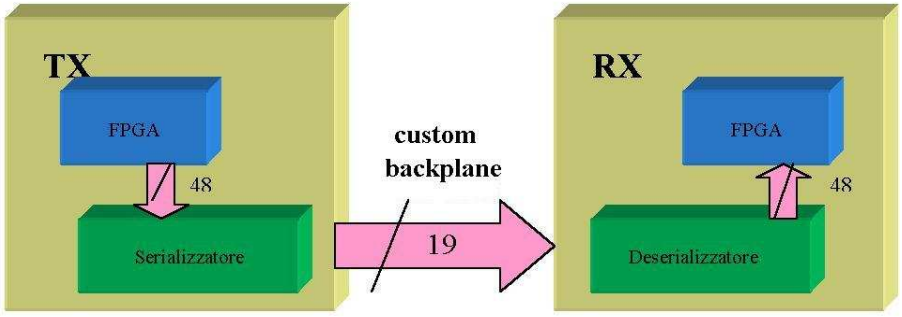


Figura 5.12 Il sistema TX – RX

Sono state seguite due filosofie di progetto: un progetto che ha seguito un approccio totalmente strutturale ed uno in cui la *FPGA* contiene un'unica

entità *VHDL*. Le caratteristiche dei due progetti in *VHDL* e i file utilizzati sono riportati nel paragrafo successivo.

Anche tutte le macchine a stati sono state sintetizzate a partire da codice *VHDL*. La codifica degli stati per le logiche di gestione della trasmissione e della ricezione è di tipo *ONE HOT*. Questa tecnica di codifica utilizza un *flip-flop* per stato ed è particolarmente indicata per dispositivi come le *FPGA*, poiché consente di ridurre notevolmente i livelli di logica tra i *flip-flop*, aumentando così la massima frequenza di clock.

La scelta di avvalersi di *FPGA* consente di realizzare un dispositivo che lavori alla frequenza desiderata con l'affidabilità di una logica cablata. Inoltre, grazie alla programmabilità del componente, si ha la possibilità di modificare la logica per adattarla ad eventuali nuove esigenze. La scelta di utilizzare un generatore di sequenze è stata dettata dal voler sottoporre ciascuno degli elementi del sistema serializzatore-deserializzatore ad uno stress costituito da un flusso di dati pressochè *random*, ma di cui è possibile studiare alcuni parametri caratteristici.

Per simulare lo stress a cui è sottoposto un sistema di trasmissione esistono sequenze, dette *pseudorandom*, che permettono una analisi del traffico dati in base a particolari esigenze di test. Per il collaudo dei serializzatori, è stato scelto un *Linear Feedback Shift Register* (*LFSR*).

Il generatore di sequenze

Il generatore di *PRBS* (*Pseudo Random Bit Sequence*) più comune e semplice da realizzare è il *Feedback Shift Register*. Esso è costituito da un registro a scorrimento della lunghezza di *m* bit. Una porta logica che è una certa funzione logica tra un qualsiasi bit, *n,* e solitamente l'ultimo bit (*m*) genera il segnale di input dello *shift register*. Uno schema è mostrato in figura 5.13.

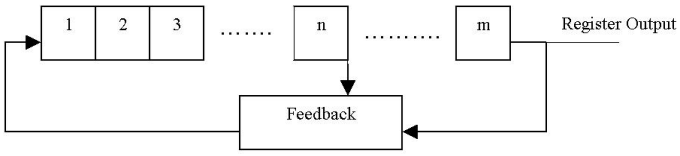


Figura 5.13 Schema a blocchi di un *Feedback Shift Register*

Se il *feedback* è fatto con una porta *OR*-Esclusivo, che in logica binaria rappresenta la somma, il *feedback* è lineare e il registro viene detto *Linear Feedback Shift Register*.

Questo circuito assumerà una serie di stati (che sono definiti dai bit del registro dopo ciascun periodo di clock), ripetendo le sue configurazioni dopo K periodi di clock, e dunque K è il periodo del registro. Il numero massimo di stati che un registro di *m* bit può assumere è $K=2^m$, cioè il numero di combinazioni binarie di *m* bit. A queste bisogna sottrarre la sequenza in cui ci sono tutti "0", perchè questa bloccherebbe, senza possibilità di uscirne, il registro nello stato di *m* zeri. La *sequenza massimale* che si può generare ha dunque una lunghezza di 2^m-1 bit. Sebbene la sequenza dia l'impressione di essere una successione arbitraria di bit 0 ed 1 (*pseudorandom)*, è in realtà completamente ricostruibile ed è particolarmente adatta al collaudo di sistemi di trasmissione dati. Una sequenza massimale si può ottenere solo scegliendo opportunamente il punto *n* del registro in cui inserire il *feedback*. Per alcuni valori di *m*, per ottenere sequenze massimali si rende necessario usare più di un *feedback*. Inoltre si possono usare valori diversi di *n* per avere gli stessi risultati e, nel caso di un solo *feedback,* è equivalente usare un *feedback* nella posizione individuata da *n* oppure da *m-n,* ossia contando la posizione del *feedback* da uno qualsiasi dei due estremi del registro (quest'ultima caratteristica viene chiamata specularità del *LFSR*).

E' possibile studiare le sequenze generate da un *LFSR*, associando ad ogni registro un polinomio, detto polinomio generatore,

$$G(X) = g_m X^m + g_{m-1} X^{m-1} + \ldots\ldots + g_1 X + g_0 ,$$

con coefficienti binari: il coefficiente della posizione in cui c'è un *feedback* sarà 1. Il polinomio viene detto primitivo se non si fattorizza e divide X^r+1, dove $r = 2^m-1$. Si utilizza un'algebra in cui l'operatore di somma è realizzato con l'OR esclusivo. A titolo di esempio, consideriamo un polinomio di grado m=3: in tal caso sarà $r = 2^m-1= 7$. I polinomi primitivi di grado 3 sono i fattori non triviali di X^7+1.

Poichè è $(X+1)(X^3+X+1)(X^3+X^2+1) = (X^4+X^2+X+X^3+X+1)(X^3+X^2+1) = X^7+1$ i due polinomi di grado 3, i cui schemi sono mostrati in figura 5.14, sono: X^3+X+1 e X^3+X^2+1, l'uno "immagine speculare" dell'altro. Inoltre dovrà essere $g_m=g_0=1$. Se il polinomio è primitivo, la sequenza prodotta sarà massimale.

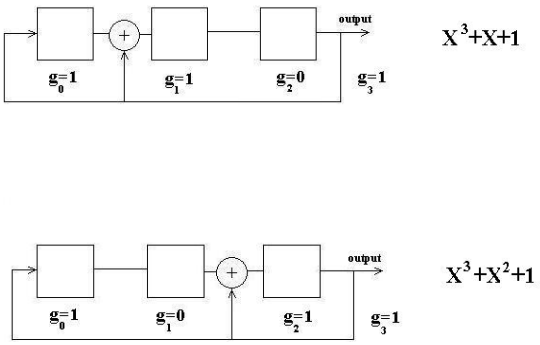


Figura 5.14 I due polinomi generatori ottenuti per m=3

Le principali proprietà di una sequenza massimale sono:

• In un intero ciclo (K periodi di clock), su un bit, il numero di "1" è più grande del numero di "0" di una unità, poichè manca la sequenza di *m* zeri. In una sequenza random, invece, la differenza tra le due somme dovrebbe tendere progressivamente a zero. Questo, per una lunga sequenza (ad es. già per *m*= 16) può essere considerato irrilevante. Tuttavia nel caso in cui si utilizzi la sequenza *PRBS* con un sistema di trasmissione, il valore medio del segnale trasmesso non è zero, ma leggermente superiore, ad ogni ciclo.

Su molti cicli, dunque, il valore medio dei dati trasmessi può discostarsi sensibilmente da zero. Si può comprendere come sia necessario, nella procedura di trasmissione sul *backplane* di una sequenza *PRBS*, l'uso di un algoritmo per il ripristino del *DC Balance* per ripristinare un corretto livello del valore medio.

• In un intero ciclo (K periodi di clock):

definendo come *run* di lunghezza r una successione di r simboli uguali, compresi tra simboli opposti (ad es. 100001 è un *run* di 4 zeri), in una sequenza massimale di m bit ci saranno $1/2^i$ dei run di lunghezza i, con 1<i<m; di questi, la metà sarà di "1" e la metà di "0". Così la metà dei *run* sarà composto da *run* lunghi un bit, un quarto sarà fatto da *run* lunghi 2 bit e così via, fino ad un singolo *run* di "1", di lunghezza m, e un *run* di "0" di lunghezza m-1. La probabilità di avere "0" oppure "1" sarà, su una sequenza sufficientemente lunga, sempre ½ e non dipenderà, in senso probabilistico, dallo stato del *flip flop* precedente.

• Sommando con un *OR* esclusivo una sequenza massimale e la stessa sequenza sfasata, si ottiene ancora la stessa sequenza, ulteriormente sfasata.

• Definendo una "finestra" di *m* bit e scorrendo tutte le 2^m-1 posizioni di una sequenza massimale, si ottengono una e una sola volta, tutti i possibili valori rappresentabili con *m* bit, a parte la sequenza di *m* zeri.

In figura 5.15 è riportato lo schema del generatore di sequenze realizzato nella *FPGA*, ovvero un registro a scorrimento in cui ci sono tre *feedback* costruiti con gli *OR* esclusivi, con il quale è possibile generare 2^{16} –1 differenti parole di 16 bit, ossia una *Pseudo Random Bit Sequence (PRBS).* La configurazione dei *feedback* è quella scelta dalla *CCITT* (*International Telegraph and Telephone Consultative Committee*) per consentire l'uniformità dei test con una sequenza *PRBS*. Allo scopo di porre il *LFSR* nella condizione di partenza, due *multiplexer* iniettano, per otto periodi di

clock, 0 o 1 su tre bit: in tal modo, la prima parola del *LFSR* sarà "0000100000000000" (con Q[15:0]).

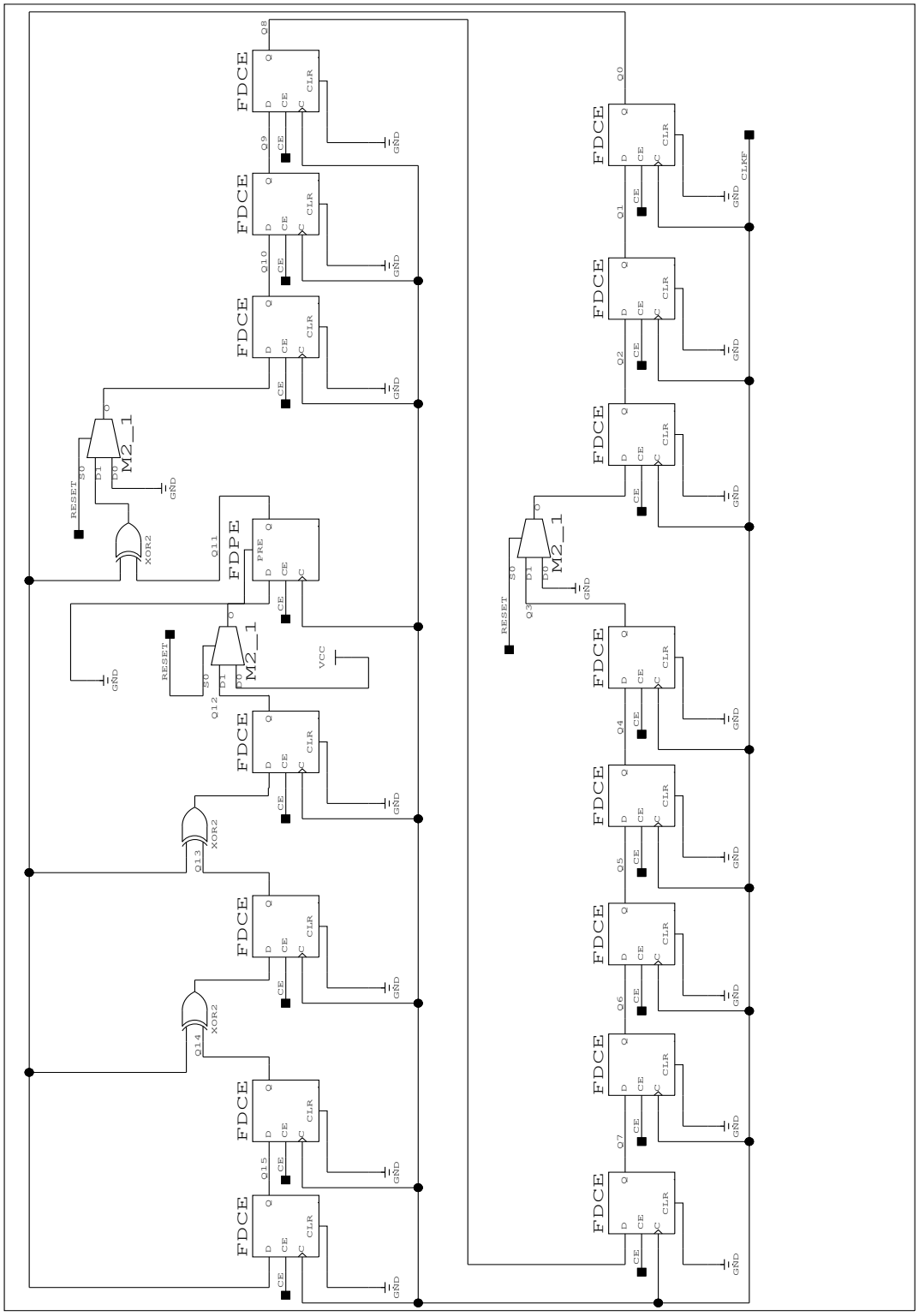


Figura 5.15 Il generatore di sequenze pseudo-random

Il *VHDL*

Come già accennato precedentemente, il *VHDL* è un linguaggio di programmazione che permette la descrizione di un sistema *hardware*. Il *VHDL* presenta differenti livelli di astrazione che l'utente può adoperare [9].

Il *gate level* permette una descrizione del sistema in termini di un *network* di porte logiche, utilizzando l'algebra Booleana.

Il *Register Transfer Level (RTL*) descrive il comportamento sincrono ed asincrono di un sistema *hardware*; permette la descrizione, tra gli altri, di macchine a stati, operatori (+, *, >...) e registri.

Il livello *Behavioural* consente di costruire modelli per simulazione, ma spesso non consente la sintesi diretta di strutture come registri, operatori o macchine a stati. Questo livello viene realizzato con moduli funzionali, connessi tra loro, che costituiscono il modello da simulare. Ciascun modulo contiene una o più funzioni e le relazioni temporali da rispettare.

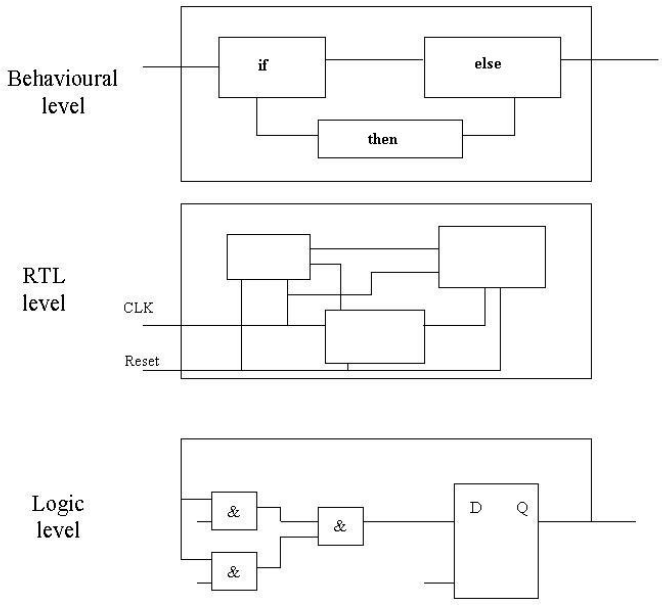


Figura 5.16 I differenti livelli di astrazione per una progettazione in *VHDL*

Un elemento centrale della progettazione in *VHDL* è la definizione dei componenti [10]. Un componente è costituito da due elementi principali: una entità *(entity)*, che contiene le dichiarazioni per gli ingressi e le uscite del componente, ed una architettura *(architecture)* che è la descrizione funzionale del componente stesso. La figura 5.17 mostra un banale esempio di realizzazione di un componente *VHDL*.

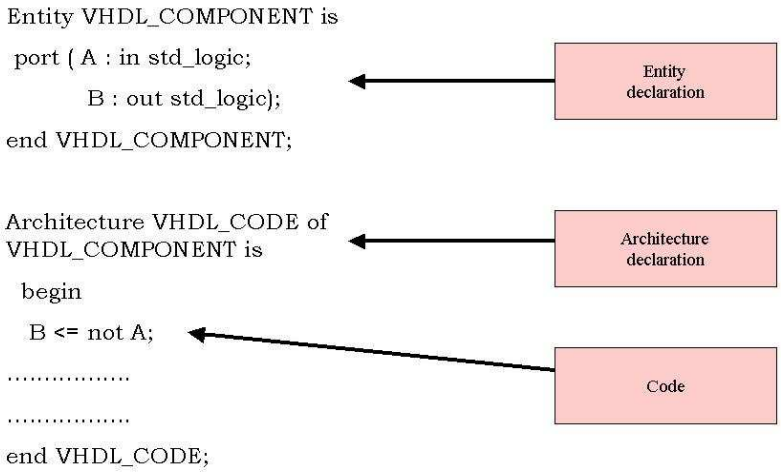


Figura 5.17 Un esempio di componente *VHDL*

All'interno di una architettura *VHDL* è possibile definire più processi, che permettono l'esecuzione delle istruzioni nel codice *VHDL*. Esistono due tipi di processi: i processi combinatori ed i processi registrati. I processi combinatori descrivono la logica combinatoria dell'elemento cui si riferiscono; i processi registrati sono sincroni con un segnale di cadenza (*clock*) e consistono di *flip flop* e una possibile logica combinatoria. Quello che è importante sottolineare è che tutti i processi in un file *VHDL* sono eseguiti "in parallelo".

Nella realizzazione di un progetto con il *VHDL* un ruolo importante è ricoperto dal *VHDL* strutturale. Il *VHDL* strutturale può essere paragonato ad una *netlist* che lega insieme differenti componenti. In particolare, il file descrive ciascuno dei componenti e le interconnessioni tra gli stessi. In tal modo viene creata una struttura gerarchica in cui un componente (detto

top-level) contiene al suo interno diversi livelli o ulteriori componenti, tenuti insieme dalla descrizione del *VHDL* strutturale. L'utilizzo di questa struttura per realizzare un progetto rende chiara l'utilità del linguaggio *VHDL*. Sebbene un *VHDL* strutturale sia più complesso da interpretare, rispetto ad un comune schematico, esso è totalmente indipendente dalla piattaforma di lavoro, dal tool di programmazione e dalla tecnologia di realizzazione del progetto.

L'esempio seguente illustra i principi del *VHDL* strutturale. Il componente descritto è un *multiplexer*, costituito, come mostrato in figura 5.18, da due porte *AND* ed una *OR*.

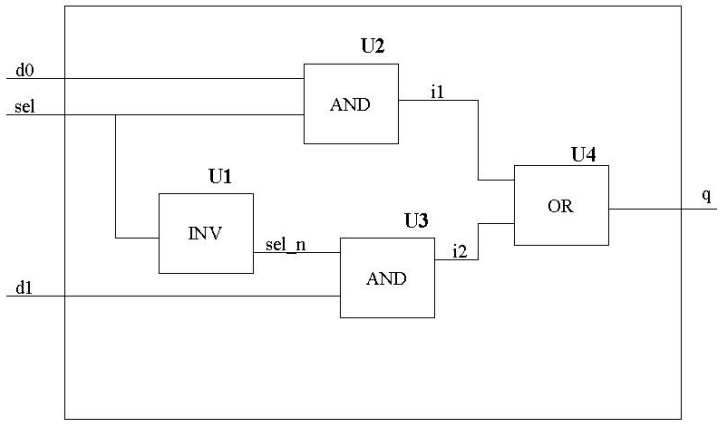


Figura 5.18 Un esempio di multiplexer realizzato con porte logiche *and* e *or*

```
Entity mux is
 Port ( d0, d1, sel: in  std_logic;
        q:           out std_logic);
end;

Architecture struct_mux of mux is

  -- Component declaration.

Component and_comp
  Port ( a, b: in std_logic;
         C: out std_logic);
End component;

Component or_comp
  Port ( a, b: in std_logic;
         C: out std_logic);
End component;
```

```
Component inv_comp
  Port ( a: in std_logic;
         b: out std_logic);
End component;

Signal i1, i2, sel_n: std_logic;

-- component specification.
For U1 : inv_comp Use Entity work.inv_comp(rtl);
For U2, U3 : and_comp Use Entity work.and_comp(rtl);
For U4 : or_comp Use Entity work.or_comp(rtl);

Begin
-- component instantiation.
U1 : inv_comp port map(sel, sel_n);
U2 : and_comp port map(d0, sel, i1);
U3 : and_comp port map(sel_n, d1, i2);
U4 : or_comp port map(i1, i2, q);

End;
```

All'interno dell'architettura del *multiplexer* sono definiti i componenti *and_comp*, *or_comp* e *inv_comp*, mentre le connessioni tra i componenti sono realizzate tramite il comando *port map*. Così, ad esempio, l'uscita *sel_n* dell'*inverter inv_comp (U1)* viene collegata ad uno dei due ingressi della porta logica *and_comp (U3)*.

Nel corso del capitolo saranno riportati i file in *VHDL* strutturale attraverso i quali sono state realizzate le funzioni logiche relative alla scheda TX ed alla scheda RX, realizzate per il collaudo del sistema di trasmissione dati su backplane.

La scheda TX

La figura 5.19 mostra lo schema a blocchi per il funzionamento della scheda. Nella progettazione si è seguita una modularità a 16 bit: esisteranno nella *FPGA* tre blocchi logici (*PRBS* 16), ognuno responsabile della gestione di un singolo *LFSR* a 16 bit e della successiva trasmissione dei dati sul *backplane*. Sulla scheda, oltre alla *FPGA*, responsabile della gestione dei dati, sono presenti tre *jumper*, J[2:0], che permettono di decidere su

quanti bit lavorare. In particolare ciascun *jumper* abilita alla trasmissione un campo di 16 bit. Inoltre sulla scheda si trova un pulsante, che permette di avviare e fermare il generatore, e quattro *led:* un *led* segnala, quando acceso, che la generazione del *PRBS* è in corso, mentre ciascuno degli altri tre *led* evidenzia il corrispondente campo di 16 bit che viene trasmesso.

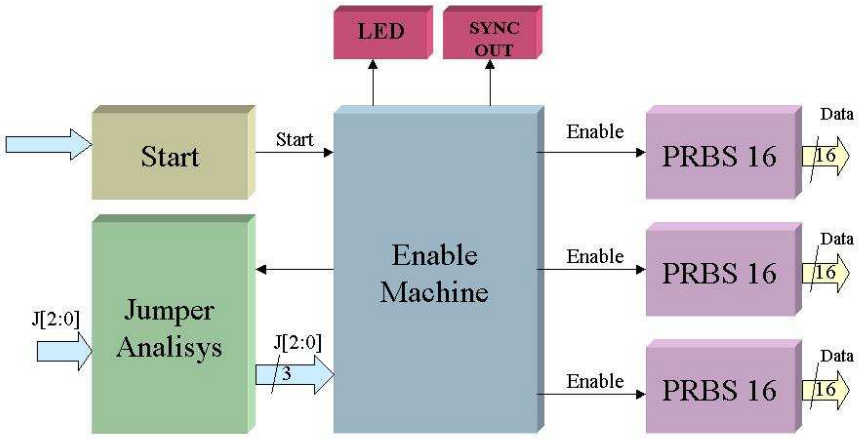


Figura 5.19 Lo schema a blocchi della scheda TX

La figura 5.20 mostra come viene elaborata l'uscita del *LFSR*. Prendendo uno dei 16 bit di ciascuna parola del *LFSR*, in particolare il meno significativo, e riproducendolo 16 volte tramite il blocco logico *FAN OUT 1:4*, sono costruite le parole da 16 bit da trasmettere. Combinando fino a tre di queste sequenze, è possibile costruire le sequenze di 16, 32 o 48 bit da fornire in ingresso al serializzatore. La scelta di un qualsiasi altro bit sarebbe assolutamente identica, purchè coerente per tutte le 2^{16}-1 parole.

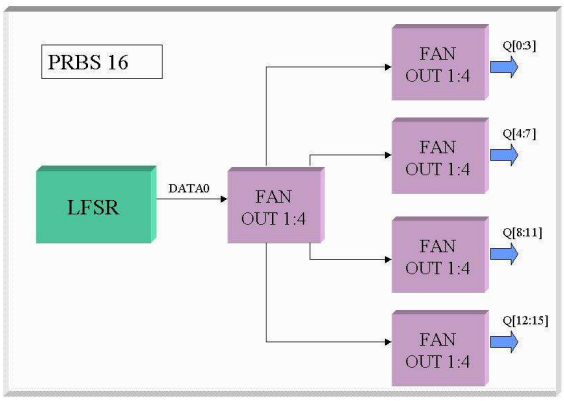


Figura 5.20 Lo schema a blocchi del *PRBS 16*

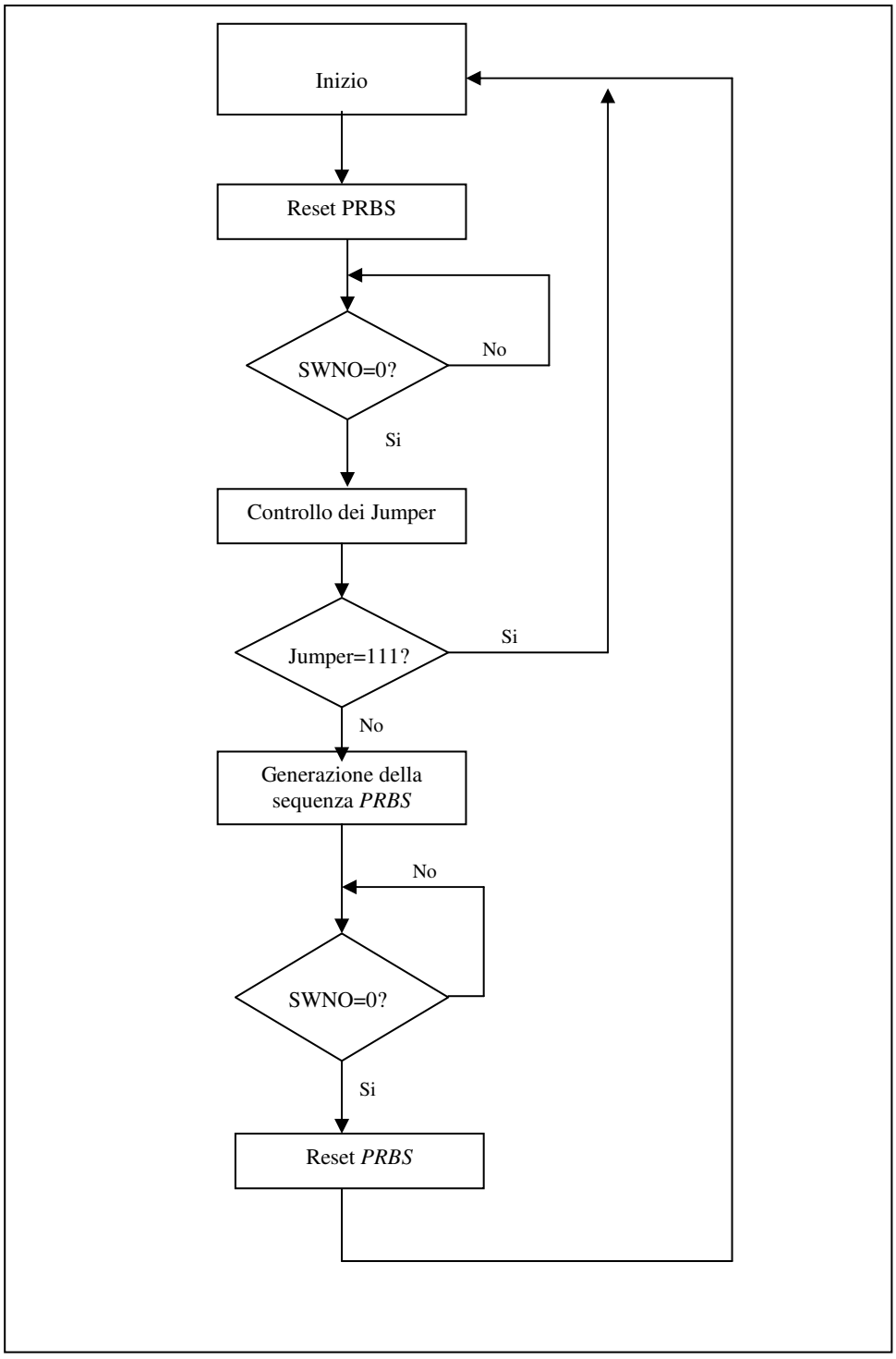


Figura 5.21 Il diagramma di flusso della scheda TX

La figura 5.21 mostra il diagramma di flusso che riguarda la modalità di funzionamento della scheda. Alla pressione del pulsante, SWNO=0, viene eseguito il reset dei tre generatori di sequenze, che dura otto periodi di

clock, durante i quali sono iniettati, su tre opportuni bit di ciascun *LFSR*, i valori "0" oppure "1". Viene inoltre controllata la disposizione dei *jumper*.

Se nessuno dei *jumper* è stato selezionato, nessun generatore da 16 bit viene attivato, e la scheda si riporta nella situazione iniziale. Se, invece, almeno un *jumper* è stato inserito correttamente nel suo alloggiamento, il modulo da 16 bit corrispondente a quel *jumper* genera la sequenza *pseudorandom*.

Ad una successiva pressione del pulsante, viene interrotta la produzione di ulteriori sequenze, viene eseguito il reset di ogni *LFSR* e la scheda si predispone per iniziare un nuovo ciclo di trasmissione. La scheda TX genera anche un bit di *sync_out*, che individua in ciascun periodo uno specifico elemento della sequenza *PRBS* (in particolare "1111111111111111"). Un'unità di conteggio esterna può essere così utilizzata per registrare il numero totale di parole trasmesse ($N*2^{16}$).

La macchina a stati *Switch machine*

In figura 5.22 è mostrato il diagramma a bolle della macchina a stati *switch_machine* utilizzata per la gestione della logica della scheda TX.

La macchina, alla partenza, si porta nello stato S1 che provvede al reset dei *LFSR*: tale reset viene eseguito, come già accennato, iniettando per otto periodi di clock, all'ingresso di determinati *flip-flop* del *LFSR*, degli "0" oppure degli "1". Per fare ciò si rende necessario l'uso di contatori a tre bit, attivati dalla macchina *switch_machine*. Esiste un contatore a tre bit in ciascun modulo che gestisce 16 bit. La macchina a stati *switch_machine* gestirà le abilitazioni dei tre contatori per il reset del *LFSR*, indicate con *counter0_start*, *counter1_start, counter2_start.*

Una volta trascorsi otto periodi di clock, a reset concluso, il bit *counter* va alto e la macchina aspetta nello stato S2 la pressione del pulsante di *start* per iniziare la generazione della sequenza.

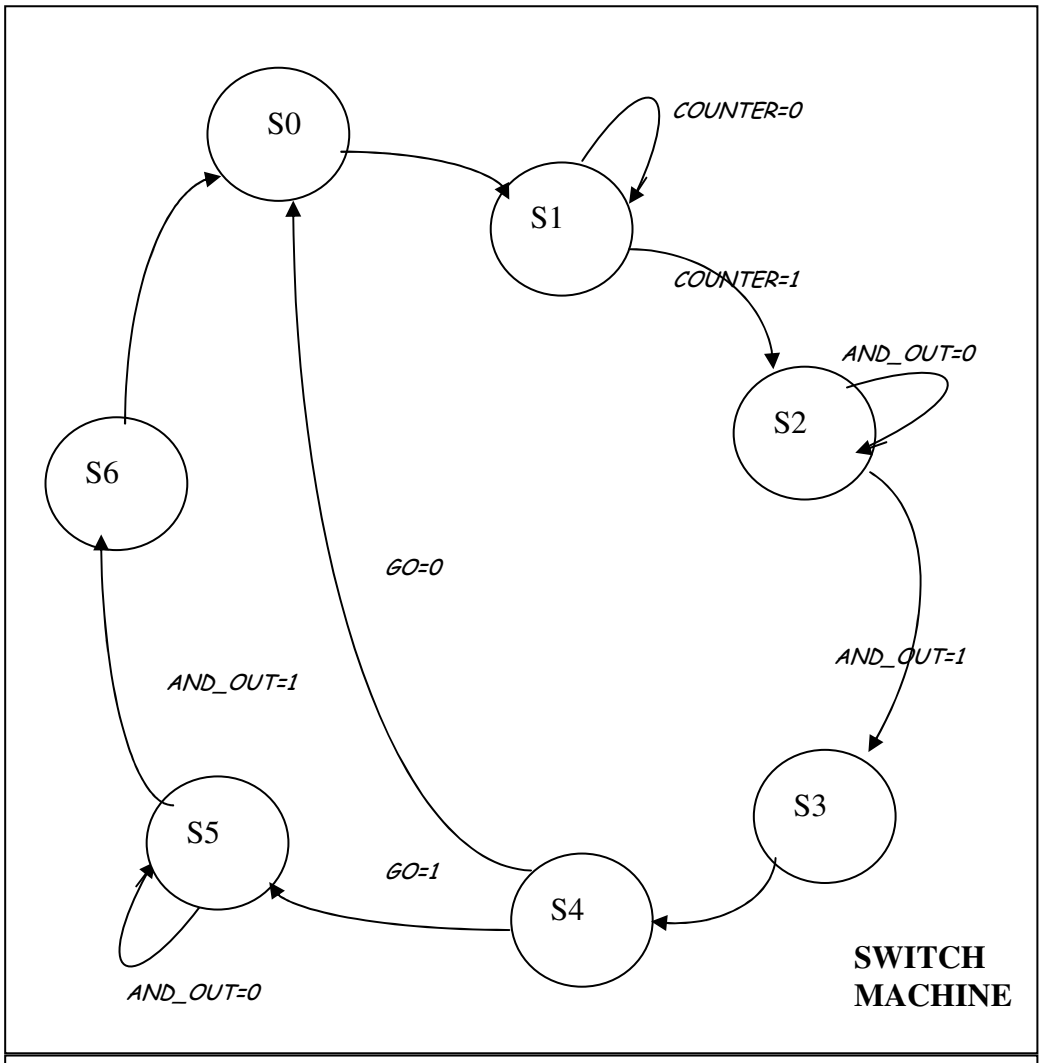


Figura 5.22 Il diagramma a bolle della macchina a stati *switch_machine*

A seguito della pressione del pulsante di *start*, il bit *and_out* va alto per un periodo di clock. In conseguenza di ciò, la macchina passa nello stato S3 ed abilita, col bit *j_start*, la macchina chiamata *jumper_machine,* responsabile del controllo della presenza dei *jumper*. Successivamente, la *switch machine* aspetta in S4 il segnale di *go*=1 dalla *jumper_machine* per dare l'abilitazione ai LFSR selezionati, tramite i bit *PRBS_ENABLE0*, *PRBS_ENABLE1, PRBS_ENABLE2*. Se nessuno dei *jumper* è stato selezionato (*go*=0) la *switch machine* ritorna allo stato di partenza. Per fornire l'abilitazioni al primo modulo a 16 bit, il segnale *PRBS_ENABLE0*viene messo in *and* logico con il bit *out(0)* dell'uscita *out[2:0]* della macchina *jumper machine*, che indica quali dei tre moduli devono essere attivati. Analogamente viene realizzato un *and* logico tra il segnale *PRBS_ENABLE1* e il bit *out (1)* e tra il segnale *PRBS_ENABLE2* e il bit *out (2).*

Una successiva pressione del pulsante di *start*, corrispondente ad un nuovo valore di *and_out*=1, interrompe la generazione dei dati del *LFSR* e fa ritornare la macchina *switch_machine* allo stato di partenza, pronta per un nuovo ciclo. Inoltre, nello stato S6, il bit *j_start* va alto per un periodo di clock, riportando in tal modo la macchina *jumper_machine* in uno stato di attesa.

In Appendice B è riportato il file *VHDL* che descrive la macchina a stati *switch_machine*.

La macchina a stati *Jumper Machine*

La macchina a stati che gestisce i *jumper* è la *j_machine*, il cui diagramma a stati è nella figura 5.23, ed è codificata in forma binaria, come è possibile vedere dal file sorgente riportato in Appendice B.

Al segnale di *j_start* della *switch_machine* vengono esaminate le posizioni dei *jumper* sulla scheda e la macchina si porta in uno degli 8 stati possibili, ciascuno relativo a una delle configurazioni dei *jumper* (a parte quella ovvia "111"), asserendo un segnale di *go*=1. Un successivo valore di *j_start* =1, corrispondente ad una nuova pressione del pulsante, fa ritornare la macchina nella condizione iniziale, pronta per una nuova analisi dei *jumper*.

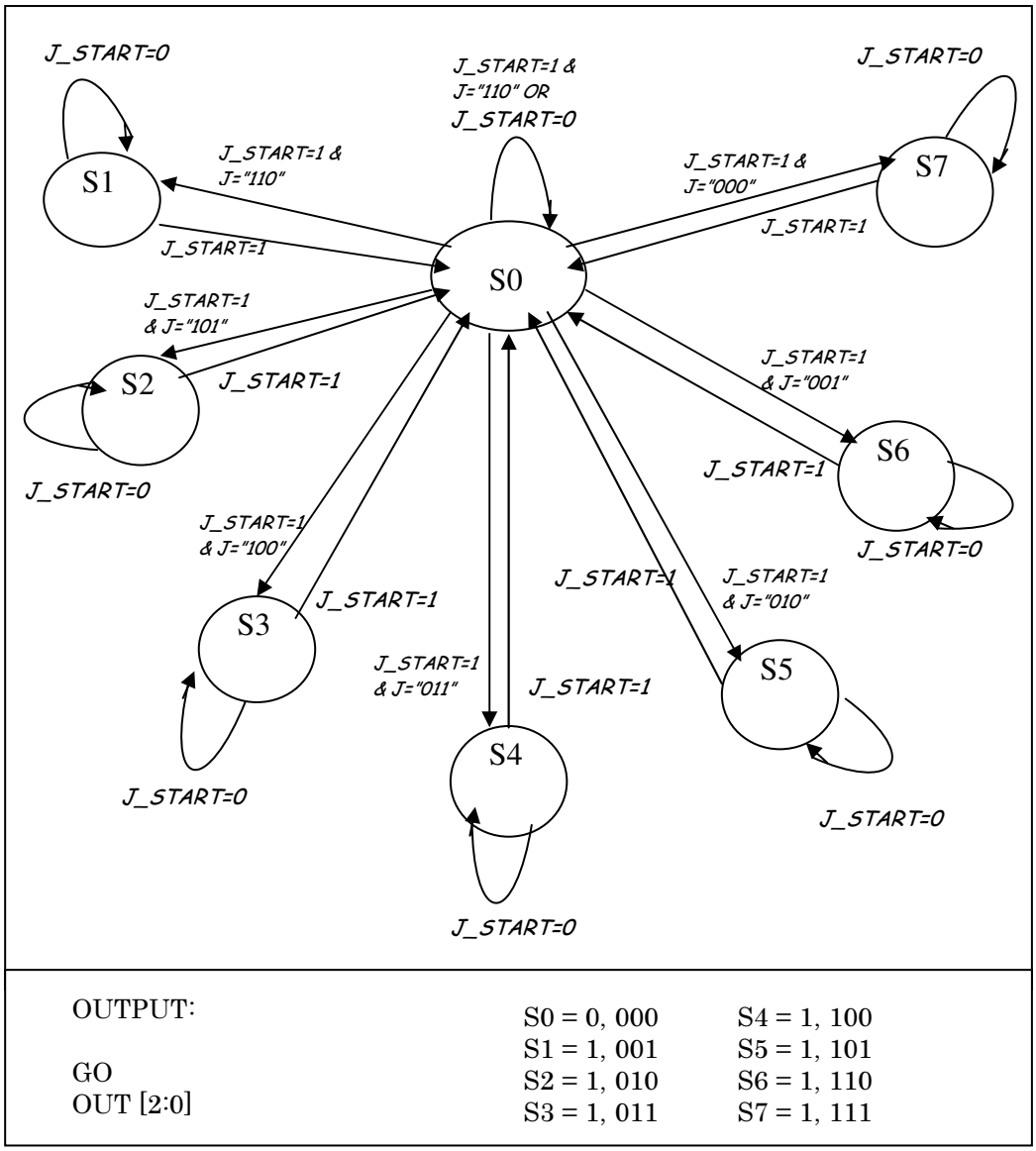


Figura 5.23 Il diagramma a bolle della macchina a stati *j_machine*

I due *file* utilizzati per la realizzazione di tutte le funzioni logiche della scheda TX sono un file totalmente strutturale, *struct_tx_fpga.vhd*, ed un file, *big_entity.vhd*, in cui c'è un'unica entità *VHDL*, il cui comportamento è descritto da una successione di processi. In questi *file* sono uniti insieme i diversi blocchi logici che costituiscono la scheda. Tali *file* non sono riportati nel presente lavoro di tesi a causa delle loro eccessive dimensioni (~ 2000 righe di codice).

La scheda RX

Anche nella progettazione della scheda RX si è seguita una modularità a 16 bit. Come si vede in figura 5.24, che mostra lo schema a blocchi della scheda RX, esisteranno nella *FPGA* tre blocchi logici (*PRBS 16 RX*) ognuno responsabile del controllo di 16 bit in ingresso.

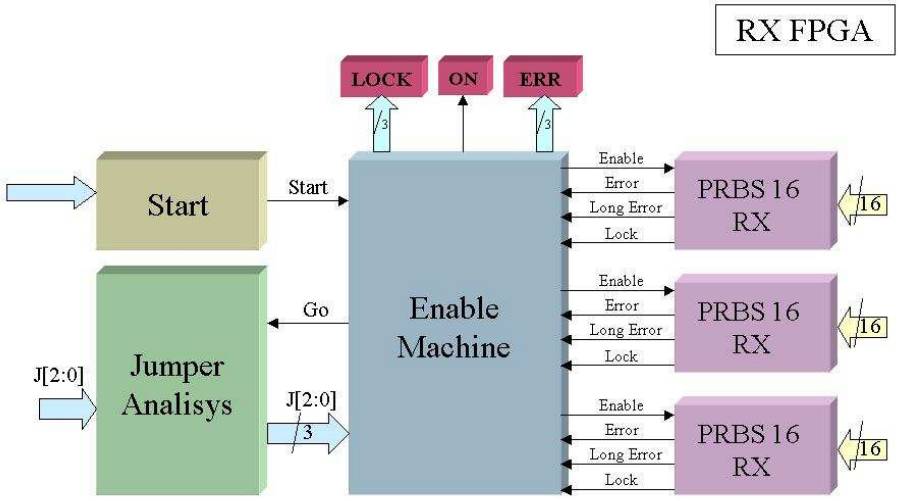


Figura 5.24 Lo schema a blocchi della scheda RX

Sulla scheda, oltre alla *FPGA*, responsabile della gestione dei dati, sono presenti ancora tre *jumper*, con le stesse funzioni ricoperte sulla scheda TX. I *jumper* permettono di decidere su quanti bit lavorare: in particolare ciascun *jumper* rappresenta un campo di 16 bit che si possono ricevere e

controllare. C'è un pulsante (*START*) che permette la partenza e lo stop del generatore, e, stavolta, sette *led*: uno (*ON*) che segnala, quando acceso, che la scheda è in funzione; tre (*LOCK*) che comunicano, uno per ciascun modulo da 16 bit, che la scheda è in uno stato di *LOCK*, il cui significato verrà discusso tra breve, ed altri tre *led* (*ERR*), uno per ciascun modulo da 16 bit, che testimoniano che si è verificato almeno un errore di trasmissione. La gestione della scheda, stavolta, è affidata a due livelli di macchine a stati: c'è una macchina a stati, la *enable_machine*, che controlla i diversi moduli da 16 bit, e, per ciascun modulo, una macchina a stati locale, la *local_machine*, gestisce tutte le funzioni. Nelle figure 5.25 e 5.26 sono rappresentati il diagramma di flusso e lo schema a bolle della macchina a stati *enable_machine.*

Il *lock*

L'idea che sta alla base del sistema di collaudo dei serializzatori è che la scheda ricevitrice riesca ad eseguire un confronto tra i dati che genera al suo interno e quelli che giungono dal *backplane*. Per far sì che tale confronto avvenga correttamente a ciascun periodo di clock è necessario un vero e proprio meccanismo di "aggancio" (*LOCK*) tra le due sequenze di dati. Bisogna cioè che la scheda RX inizi a generare i dati al suo interno solo dopo aver riconosciuto in maniera univoca una certa parola (*Signature*) in ingresso. L'univocità è garantita da una delle proprietà di una sequenza pseudorandom massimale descritte nei paragrafi precedenti: definendo una "finestra" di *m* bit e scorrendo tutte le 2^m-1 posizioni di una sequenza massimale, si ottengono una e una sola volta, tutti i possibili valori rappresentabili con *m* bit, a parte la sequenza di *m* zeri. Identificata una di queste parole (la *signature* in particolare è "111111111111111"), è possibile allineare anche tutte le altre parole che arriveranno alla scheda RX con il generatore locale.

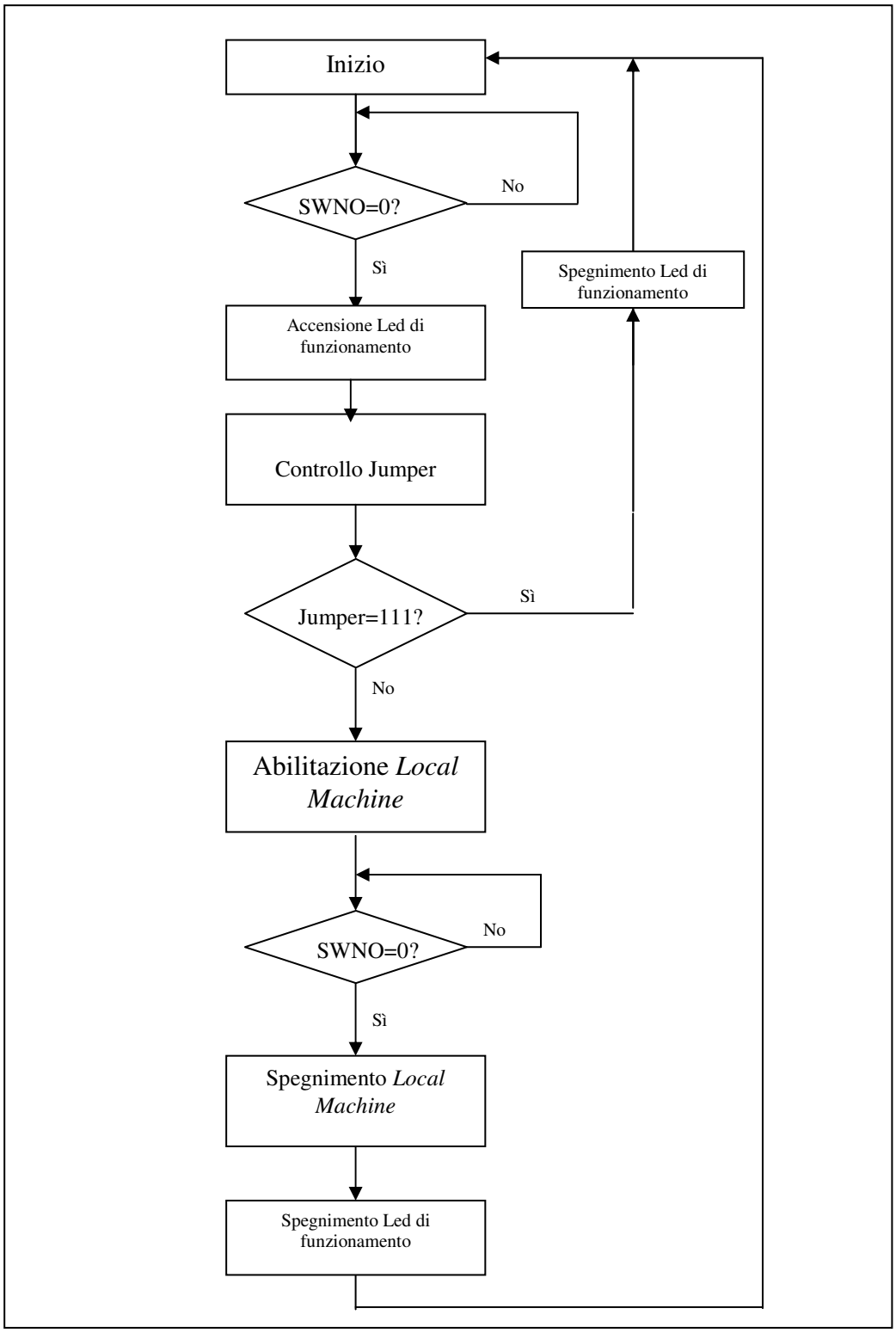


Figura 5.25 Il diagramma di flusso della macchina a stati *enable_machine*

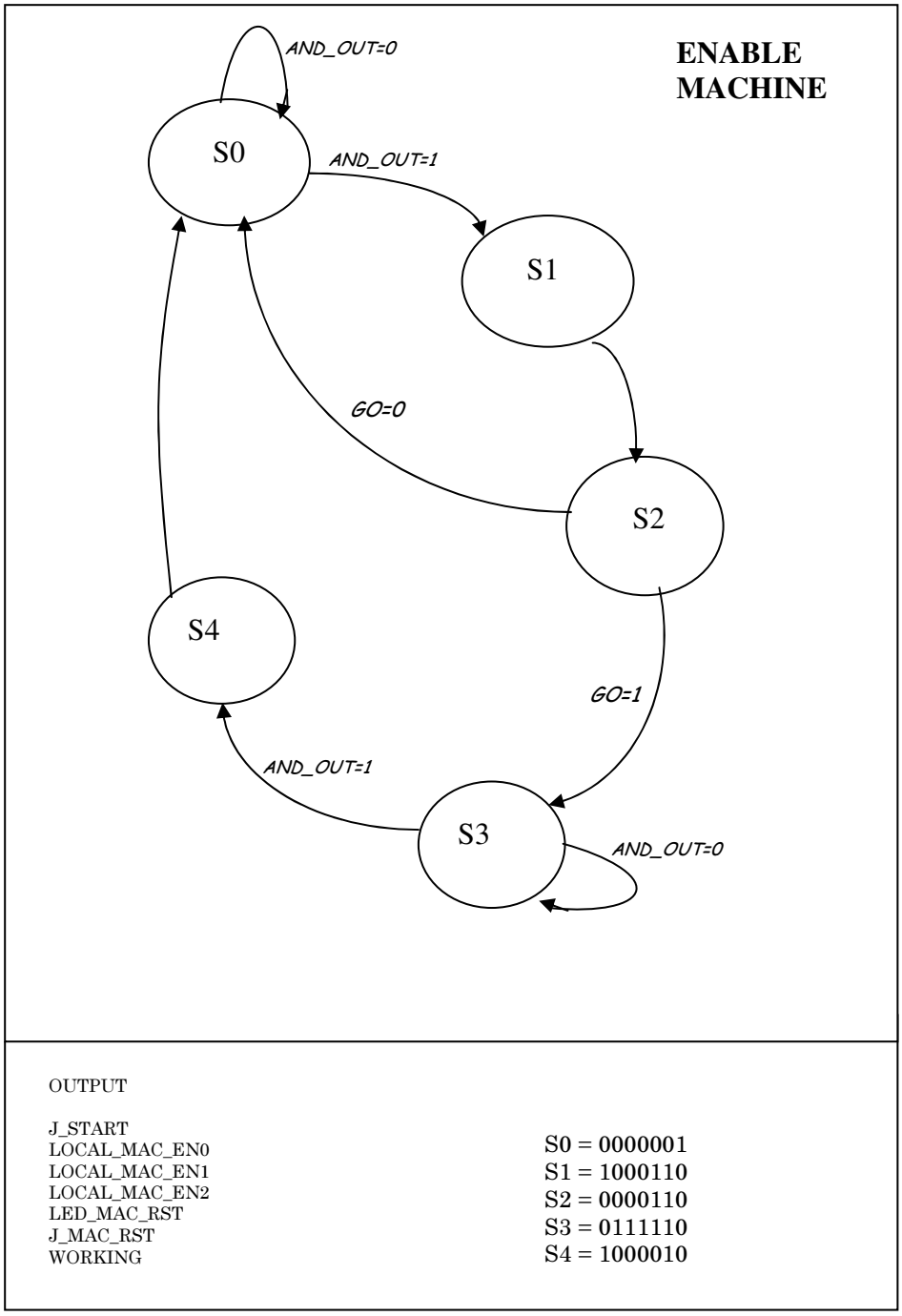


Figura 5.26 Il diagramma a bolle della macchina a stati *enable_machine*

E' sufficiente quindi sincronizzare il riconoscimento della *signature* con la partenza del *LFSR* della scheda ricevitrice da una certa parola iniziale per garantire un accordo tra le due sequenze a ciascun periodo di clock. In questo caso si può essere certi che, in assenza di errori di trasmissione, la sequenza trasmessa e quella generata sulla scheda sono uguali. Lo scopo del

led di *lock* è comunicare che la *signature* è stata riconosciuta, che è in corso la generazione della sequenza *pseudorandom* da parte del *LFSR* della scheda e che ha senso effettuare un confronto tra i dati in arrivo e quelli prodotti dalla scheda RX.

La macchina a stati *enable_machine*

All'accensione, la macchina attende la pressione del pulsante di *start* per controllare la posizione dei tre *jumper* della scheda. Alla pressione del pulsante di *start* segue anche l'accensione del led *working*, che testimonia il funzionamento della scheda. Se nessuno dei *jumper* è stato selezionato (*j=111*), nessun modulo da 16 bit viene attivato, e la scheda si riporta nella situazione iniziale. Se, invece, almeno un *jumper* è stato inserito correttamente nel suo alloggiamento, il modulo da 16 bit corrispondente a quel *jumper* viene attivato, abilitando la macchina locale *local_machine*, nel blocco *PRBS 16 RX* corrispondente. In tal caso inizia il controllo della correttezza dei dati in ingresso a ciascuno dei tre moduli da 16 bit e la ricerca della *signature*. Ricordiamo che i dati in ingresso devono essere tutti uguali tra loro, per come sono stati prodotti sulla scheda TX: sono, infatti, tutti la replica di un unico bit. Ad una successiva pressione del pulsante di *start*, le *local_machine* attivate vengono disabilitate, il led *working* di funzionamento della scheda viene spento, la macchina *jumper_machine* ritorna nello stato di partenza e la macchina *enable_machine* si riporta nella situazione iniziale in cui aspetta una pressione dello *switch* per ricominciare con un nuovo ciclo di funzionamento.

In figura 5.26 è mostrato il diagramma a bolle della macchina a stati *enable_machine*. La macchina resta nello stato S0 di attesa fino a quando la pressione del pulsante di *start* fa sì che il bit *and_out* resti alto per un periodo di clock. In questo caso la macchina passa nello stato S1 in cui viene abilitata, con il bit *j_start*, la macchina che gestisce i jumper, il cui

funzionamento è analogo a quello sulla scheda TX. Se il segnale *go* prodotto dalla *jumper machine* è basso, vuol dire che nessun jumper è stato selezionato e la macchina *enable_machine* torna nello stato S0, pronta per un nuovo ciclo di funzionamento. Se il segnale *go* della *jumper_machine* è alto, la macchina *enable_machine* passa nello stato S3, in cui vengono abilitate le macchine locali con i bit *Local_mac_en0*, *Local_mac_en1, Local_mac_en2.* Per fornire l'abilitazioni al primo modulo a 16 bit, il segnale *Local_mac_en0* viene messo in *and* logico con il bit *out(0)* dell'uscita *out[2:0]* della macchina *jumper machine*, che indica quali dei tre moduli devono essere attivati. Analogamente viene realizzato un *and* logico tra il segnale *Local_mac_en1* e il bit *out (1)* e tra il segnale *Local_mac_en2* e il bit *out (2).*

Una successiva pressione del pulsante di *start*, corrispondente ad un nuovo valore di *and_out*=1, interrompe il controllo dei dati in ingresso, disabilita le macchine locali e fa ritornare la macchina *enable_machine* allo stato di partenza S0, pronta per un nuovo ciclo. Inoltre, nello stato S4, il bit *j_start* va alto per un periodo di clock, riportando in tal modo la macchina *jumper_machine* in uno stato di attesa.

La macchina a stati *Local_machine*

La figura 5.27 mostra lo schema a blocchi della *local_machine* La macchina si occupa della gestione di un modulo da 16 bit di dati. In particolare ci sono quattro moduli logici che la macchina deve tenere sotto controllo: il *PRBS* della scheda, o *Rx_PRBS*, uno *shift_register*, il blocco *data_check* e la *lock_machine*. Tre sono i blocchi logici responsabili della identificazione della *signature*: *shift_register*, *data_check* e *lock_machine*. La *signature* è, come già accennato, la parola in corrispondenza della quale viene iniziata la generazione dei dati dal *Rx_PRBS* e viene fatto il confronto tra i dati in ingresso e quelli generati dal *Rx_PRBS*.

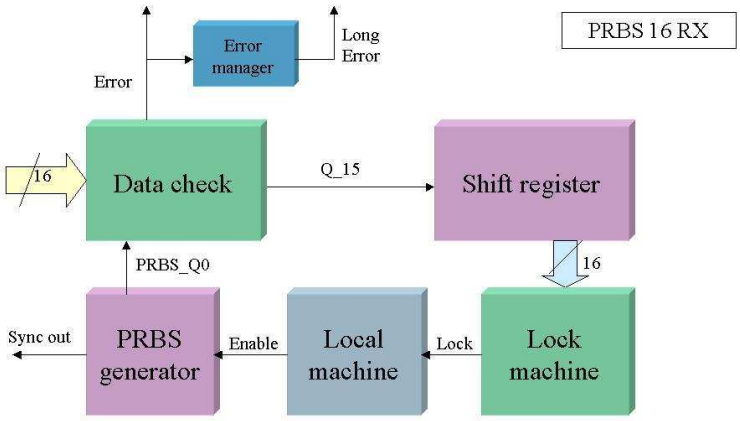


Figura 5.27 Lo schema a blocchi della macchina a stati *Local_machine*

Il blocco logico *data_check* esegue un controllo su tutti i 16 bit che entrano in parallelo nella scheda (e che devono essere uguali tra loro) e ne confronta uno con quello generato dal *Rx_PRBS*. Per eseguire questo confronto è stato scelto il bit più significativo dei 16 in ingresso, ma essendo tutti i 16 bit uguali qualsiasi altra scelta è irrilevante. Il generatore della sequenza (*Rx_PRBS*) è un *LFSR* di cui viene promosso in uscita il bit meno significativo, che viene inviato nel blocco logico *data_check*. Dal *Rx_PRBS* viene prodotto anche un bit di *sync_out*, in corrispondenza di una determinata parola del *LFSR*. La parola scelta è "1011011111110101" : in questo caso i bit di *sync_out* di ciascuna scheda sono prodotti in corrispondenza del medesimo fronte di salita del clock. Lo *shift_register* è il registro che effettua una conversione serie-parallelo accettando al suo ingresso il bit più significativo dei 16 provenienti dalla scheda TX. Le uscite di ciascuno dei *flip flop* che costituiscono il registro forniranno ad ogni periodo di clock una parola di 16 bit, che viene confrontata con la *signature* scelta. La *lock_machine* riconosce, tra tutte le parole di 16 bit in uscita dallo *shift_register* una particolare parola (la *Signature,* che è "1111111111111111") e dà l'abilitazione al *Rx_PRBS* per iniziare la generazione dei dati. Questi tre blocchi logici sono descritti nel corso di questo capitolo.

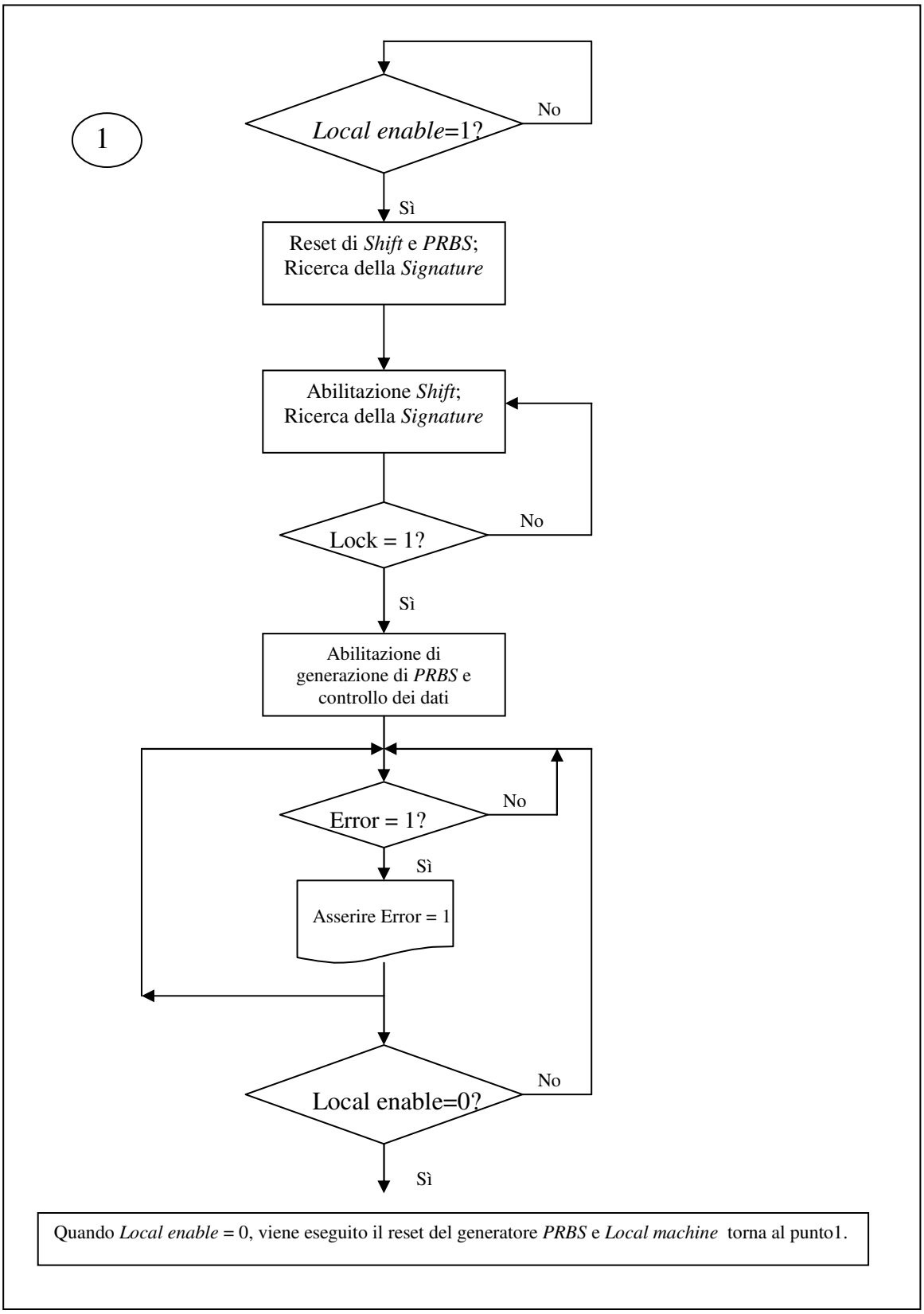


Figura 5.28 Il diagramma di flusso della macchina a stati *Local_machine*.

Una volta abilitata dalla *enable_machine* con il segnale di *local_enable*, la macchina *local_machine* opera il reset del *Rx_PRBS*, dello *shift_register* e dei blocchi logici *data_check* e *lock_machine*, che sono responsabili della ricerca della *signature* (*Signature_search*). Vengono successivamente abilitati lo *shift_register* e il blocco di *signature_search*, fin quando non è stata individuata la *signature*. A questo punto, dalla *lock_machine*, giunge il segnale di *lock* e la *local machine* abilita la generazione dei dati dal *Rx_PRBS* e la comparazione, nel blocco *data_check*, tra il dato generato e il più significativo dei 16, tutti uguali, ricevuti.

Verrà asserito un segnale di errore se c'è disuguaglianza tra dato interno e ricevuto o se non tutti i 16 bit in ingresso sono uguali tra loro. In caso di errore la macchina *local_machine* asserisce un segnale di errore, ma continua la comparazione, senza perdere il sincronismo tra i dati in ingresso e quello interno, anche se c'è una lunga sequenza di errori. Il segnale di *lock* viene asserito dalla *lock_machine* ogni volta che viene riconosciuta la *signature*, che si ripresenta in uscita dallo *shift_register* ogni 2^{16}-1=65535 cicli di clock. Una volta che è in atto la generazione dei dati da parte del *Rx_PRBS*, a ciascun ulteriore segnale di *lock*, se si è verificato almeno un errore, il *Rx_PRBS* viene resettato e fatto ripartire, ancora in sincronismo con i dati in ingresso. Se, invece, non si è verificato alcun errore il nuovo segnale di *lock* viene ignorato. La pressione del pulsante di *start*, in qualsiasi istante, fa sì che la *enable_machine* disabiliti la *local_machine* asserendo il bit *local_enable* =0. In questo caso la *local_machine* è riportata al punto di partenza, operando il reset del *Rx_PRBS*. Nel caso di una successione prolungata di errori, viene dato, in uscita dall'*FPGA*, un impulso di errore (*Long Error,* in figura 5.27) per ogni periodo di clock, gestito dal blocco logico *Error manager*.

Le figure 5.28 e 5.29 mostrano il diagramma di flusso e il diagramma a bolle della macchina *local_machine*. Nello stato S0 la macchina esegue il reset dei blocchi logici che controlla: *Rx_PRBS* (*prbs_rst*), *lock_machine*

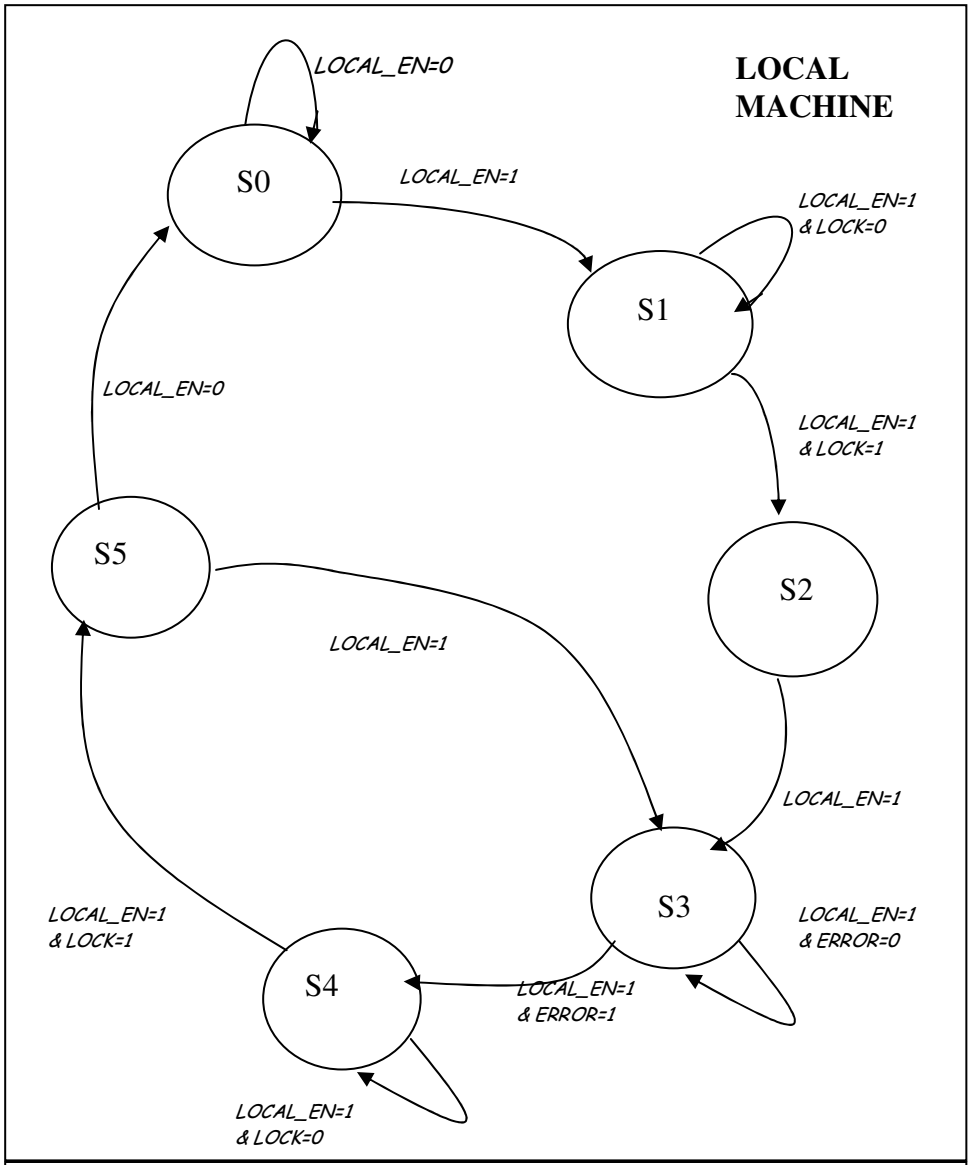


Figura 5.29 Il diagramma a bolle della macchina a stati *Local_machine*

(lock_mac_rst), data_check (*data_check_rst*) e *shift_register* (*shift_rst*). Resta inoltre, in attesa del segnale di abilitazione *local_en*, proveniente

dalla macchina *enable_machine*, che fa passare la macchina nello stato S1. In S1 vengono abilitati i blocchi logici *data_check* e *shift_register*, i cui segnali di reset non sono più asseriti. Inoltre tramite il segnale *lock_mac_en* viene abilitato il blocco *lock_machine*.

Nello stato S1 la macchina resta in attesa del segnale di *lock* generato dalla macchina *lock_machine* quando viene individuata la *signature*. Dopo essere passata per un ciclo di clock nello stato S2, la macchina transita nello stato S3, in cui viene abilitato anche il *Rx_PRBS*. La macchina resta nello stato S3 a meno che non venga riconosciuto un errore nei dati. In tal caso, giunge in ingresso il segnale *error* dalla macchina *data_check* e la macchina *local_machine* passa nello stato S4, pur continuando a svolgere le stesse funzioni dello stato S3, in attesa di un successivo segnale di *lock*. Una volta che la macchina *lock_machine* ha riconosciuto ancora la *signature* e di conseguenza ha asserito il segnale di *lock*, la macchina *local_machine* si porta nello stato S5, per eseguire un nuovo reset del *Rx_PRBS* e passare, al ciclo di clock successivo, ancora in S3. La generazione dei dati del *Rx_PRBS* ed il confronto con i dati in ingresso possono essere interrotti solo togliendo l'abilitazione *local_en* alla macchina *local_machine*. Se il bit *local_en* è uguale a "0", la macchina *local_machine*, in qualsiasi stato si trovi, ritorna allo stato S0, pronta per un nuovo ciclo di funzionamento.

Lo *Shift_register*

La funzione dello *shift_register* è quella semplice di un registro a scorrimento, cioè di un convertitore serie-parallelo: acquisisce un dato per volta in ingresso per riproporlo in uscita in una parola da 16 bit, che dovrà essere acquisita e riconosciuta dalla *lock_machine*. I dati che vanno in ingresso sono stati preventivamente filtrati dalla macchina *data_check*, che ha tra i suoi compiti quello di verificare che tutti i 16 bit in ingresso siano uguali tra loro. Visto l'obiettivo di riconoscere una parola di 16 bit, è necessario che tale parola si presenti, univocamente, solo una volta su 2^{16}-1,

in uscita dallo *shift_register*. Tale condizione è soddisfatta soltanto se la *signature* è "111111111111111". In tutti gli altri casi potrebbero verificarsi riconoscimenti errati a causa della presenza nello *shift_register* di "0" indesiderati. Questi "0" possono provenire dall'operazione di reset del registro, oppure dalla trasmissione di "0" sul *backplane* da parte dei serializzatori, quando il trasmettitore è spento.

La macchina *Data_check*

In figura 5.30, è illustrato lo schema della macchina *Data_check*. La macchina *Data_check* è contenuta in ciascuno dei tre moduli *PRBS 16 RX* , che trattano, all'interno della *FPGA*, i dati in ingresso alla scheda. Nella macchina *data_check* confluiscono e vengono analizzati 16 bit in ingresso. I dati vengono comparati tra loro, tramite il blocco *CMP* mostrato in figura 5.30, in blocchi di quattro bit. Se sono tutti uguali fra loro, da ciascun blocco di quattro viene preso un bit e confrontato con altri tre provenienti ciascuno da un altro blocco. In questo modo, solo se tutti i 16 bit sono uguali fra loro, ne viene estratto uno, in particolare il più significativo, e viene trasmesso allo *shift_register*.

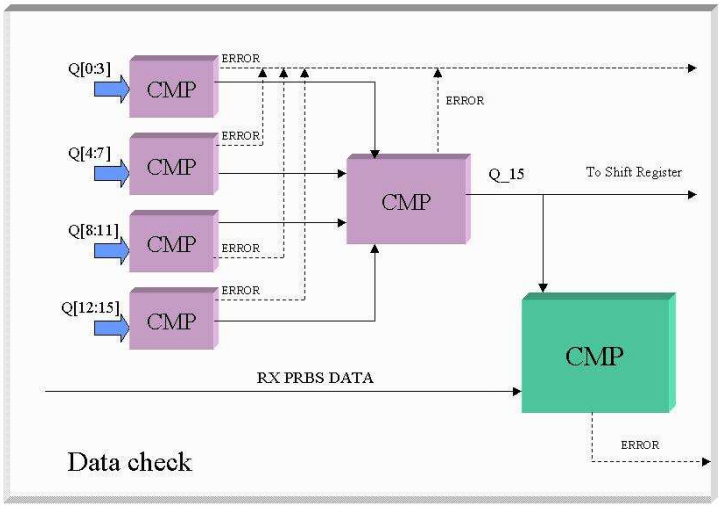


Figura 5.30 Schema della macchina *Data_check*

La macchina *Data_check* esegue anche una comparazione tra i dati in arrivo e quelli prodotti dal *Rx_PRBS* all'interno del modulo da 16 bit. In

particolare viene eseguito il confronto tra il bit più significativo dei dati in ingresso e quello più significativo di quelli generati dal *Rx_PRBS*. La macchina *Data_check* è responsabile della gestione degli errori. Un errore può essere di trasmissione, se i 16 bit che arrivano alla scheda non sono tutti uguali, oppure un errore di disaccordo tra la sequenza proveniente dalla scheda TX e la sequenza prodotta dal generatore locale. Il bit di errore dovuto alla comparazione tra dato trasmesso e ricevuto viene asserito solamente se nel blocco *PRBS 16 RX* che contiene la macchina *Data_check* è stata riconosciuta almeno una volta la *signature*. Il bit di errore dovuto alla disuguaglianza dei dati che giungono sulla scheda viene asserito a patto che il blocco *PRBS 16 RX* sia abilitato. In caso di errore viene acceso un *led* sulla scheda che resta acceso anche se il blocco *PRBS 16 RX* corrispondente riconosce una nuova *signature* ed asserisce il bit di *lock*.

La macchina *Lock_machine*

La macchina *lock_machine* legge le parole che ad ogni periodo di clock sono presenti all'uscita dello *shift_register* e le confronta con quella da riconoscere, la *signature*. Una volta riconosciuta la *signature*, viene dato il segnale di *lock*. Per la struttura dei dati, la *signature* viene riconosciuta ogni 2^{16}-1 periodi di clock, per cui il segnale di *lock* si presenterà con questa frequenza. Esso sarà ignorato dalla *local_machine*, se non c'è stato nessun errore, oppure servirà alla macchina *local_machine* per riprendere il sincronismo con i dati, come già descritto precedentemente. Inoltre, la macchina *lock_machine* gestisce il led di *lock* della scheda. Tale led viene comandato dalla macchina di accensione dei led di *lock*, *lock_led_machine*, che riceve dalla *lock_machine* il segnale di *lock_state*. In figura 5.31 è riportato il diagramma a bolle della macchina *lock_machine* per la generazione del segnale di *lock_state*. Quando viene generato il segnale di *lock*, la macchina passa dallo stato S0 allo stato S1 e il segnale *lock_state*

diventa uguale a “1”. La macchina ritorna in S0 quando viene eseguito il reset, con il segnale *lock_rst* generato dalla *local_machine*.

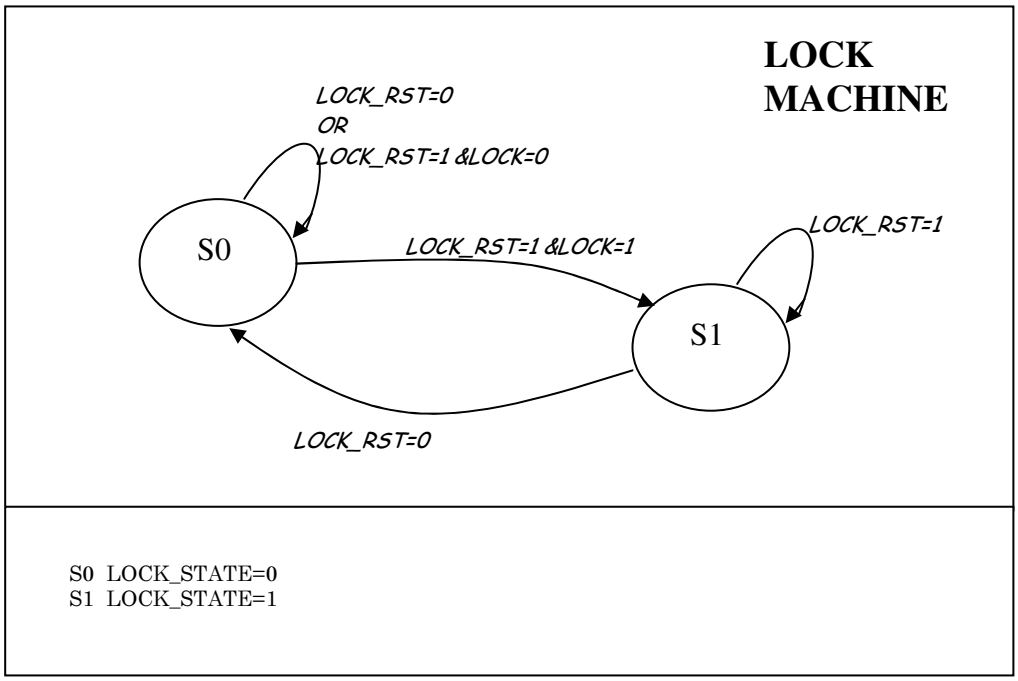


Figura 5.31 Il diagramma a bolle della macchina a stati *lock_machine*

La macchina che gestisce il led di *lock*, *lock_led_machine*, ha una struttura molto semplice. Essa è costituita da un multiplexer 2 a 1, che ha ai suoi due ingressi il livello logico alto e il valore invertito di *lock_state.* La selezione tra questi due ingressi del multiplexer viene effettuata tramite il bit *led_sync_rst*, generato dalla macchina *enable_machine*. Se il bit *led_sync_rst* è =0, viene eseguito il reset della macchina *lock_led_machine* e l’uscita del multiplexer è “1”; se invece il bit *led_sync_rst* è =1, l’uscita del multiplexer è il valore invertito del bit *lock_state*. Tale valore di uscita del multiplexer va in ingresso ad un *flip-flop* e, al successivo ciclo di clock, è connessa al led di *lock* del corrispondente modulo da 16 bit. Il valore invertito del bit *lock_state*, dunque, è quello che comanda l’effettiva accensione del led di *lock*, quando necessario.

La macchina a stati e*rror_led_machine*

I tre led di errore presenti sulla scheda RX, uno per ciascun modulo da 16 bit, sono gestiti alla macchina *error_led_machine*. Il diagramma a bolle della macchina *error_led_machine* è riportato in figura 5.32.

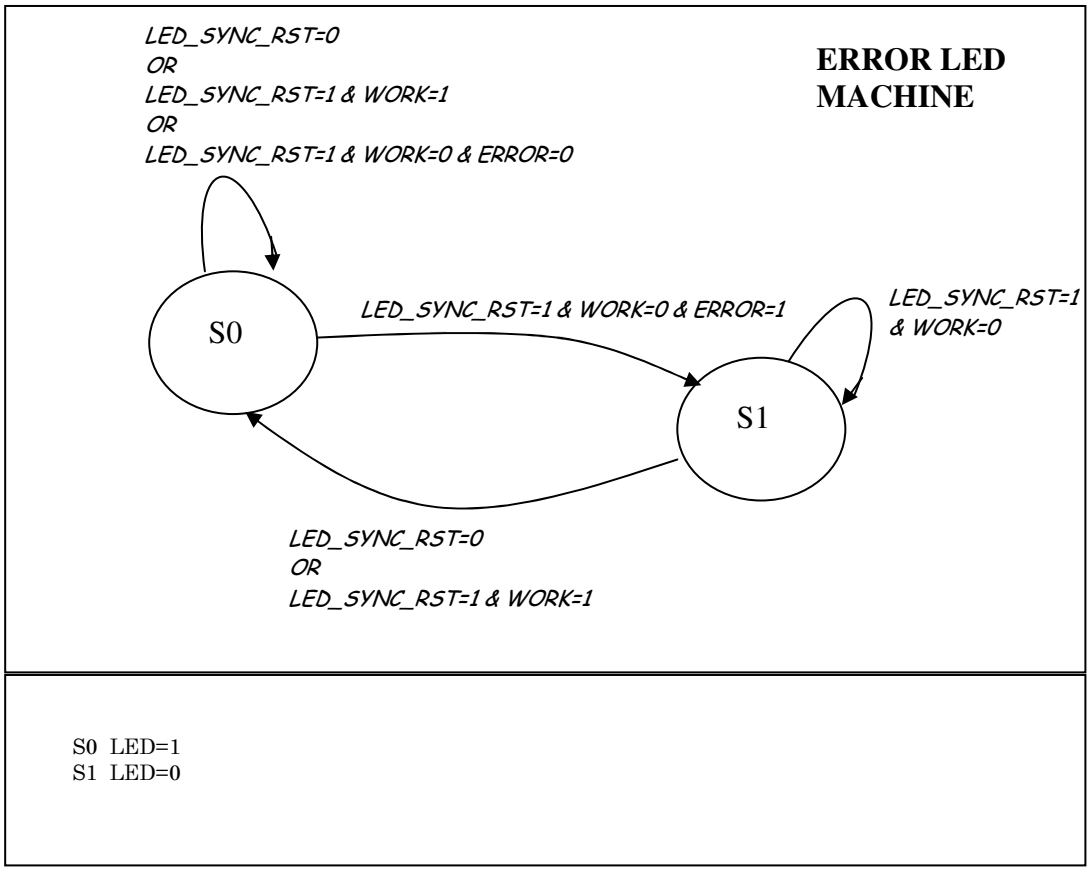
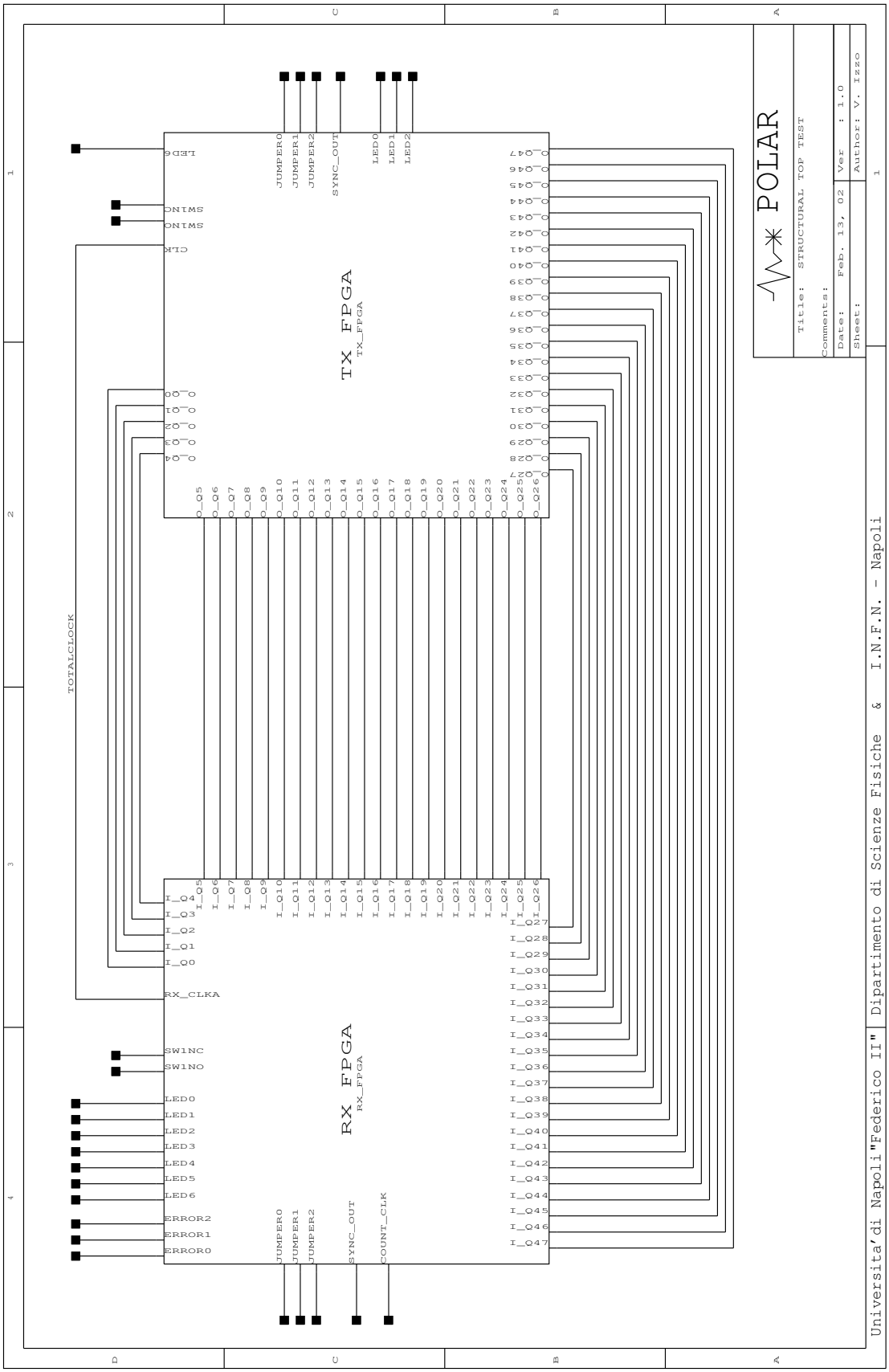


Figura 5.32 Il diagramma a bolle della macchina e*rror_led_machine*

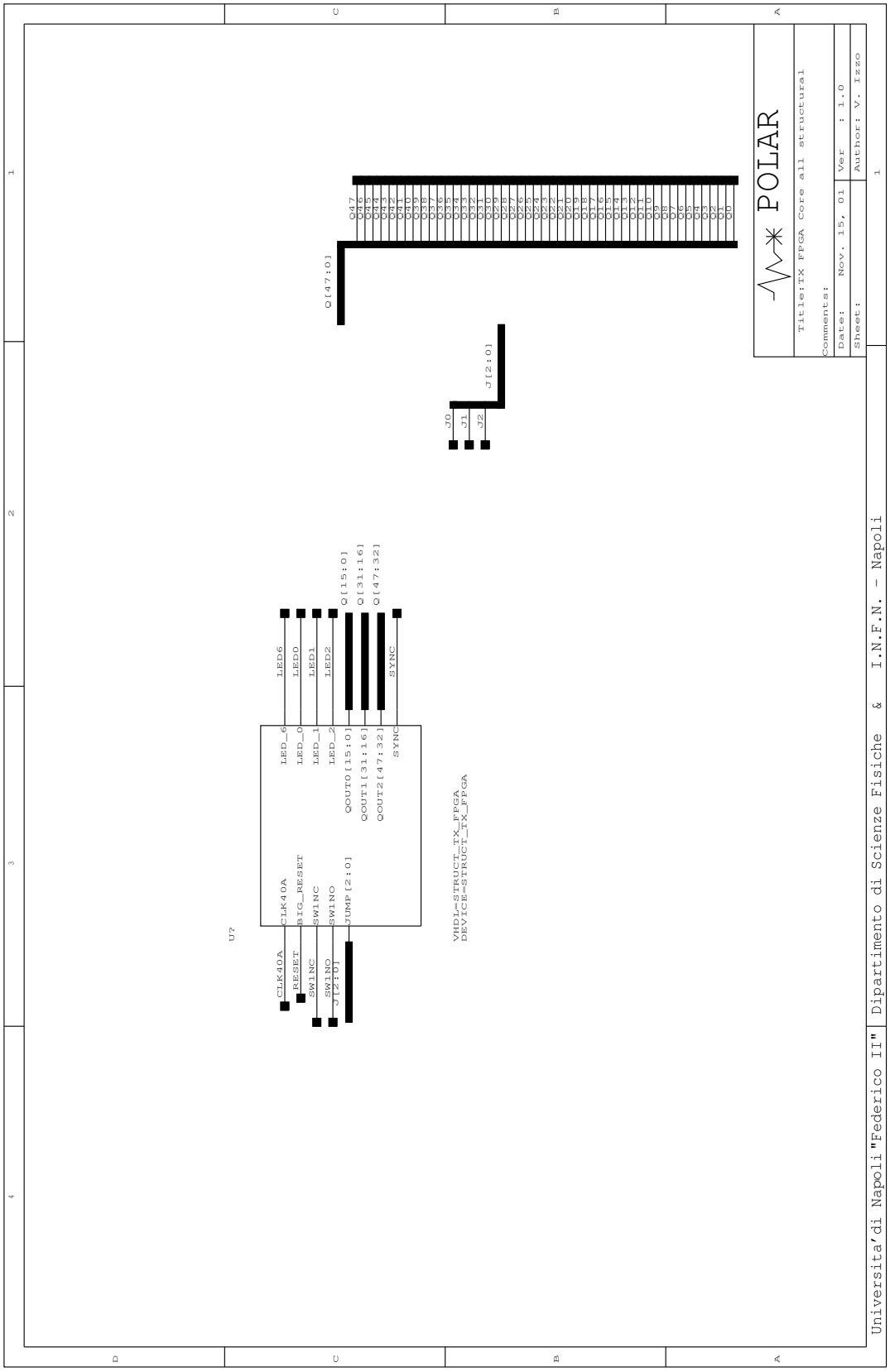
La macchina, all'accensione, si trova nello stato S0 e l'uscita *led*, che è connessa fisicamente al *led* di errore del corrispondente modulo da 16 bit, è alta, mantenendo così spento il *led* di errore sulla scheda. La macchina resta nello stato S0 se il bit *led_sync_rst*, che viene generato dalla macchina *enable_machine*, è al livello logico basso, oppure se il bit *led_sync_rst* è al livello logico alto ma il bit *work*, che testimonia dell'effettivo funzionamento della scheda ed è generato dalla *enable_machine*, è al livello

logico alto. La macchina resta nello stato S0 anche se il bit *led_sync_rst* è alto, il bit *work* è basso ed il bit *error*, generato dalla macchina *data_check*, è basso. Se, invece, *led_sync_rst*="1", *work*="0" ed *error* ="1", la macchina passa nello stato S1, dove il bit di uscita *led* è al livello logico basso e dunque il corrispondente *led* della scheda viene acceso. Dallo stato S1 la macchina può ritornare nello stato S0 se *led_sync_rst*="0" (che implica che la *local_machine* corrispondente è stata disabilitata) oppure se *led_sync_rst*="1" e *work*="1" (il che implica che la scheda non è in funzione).

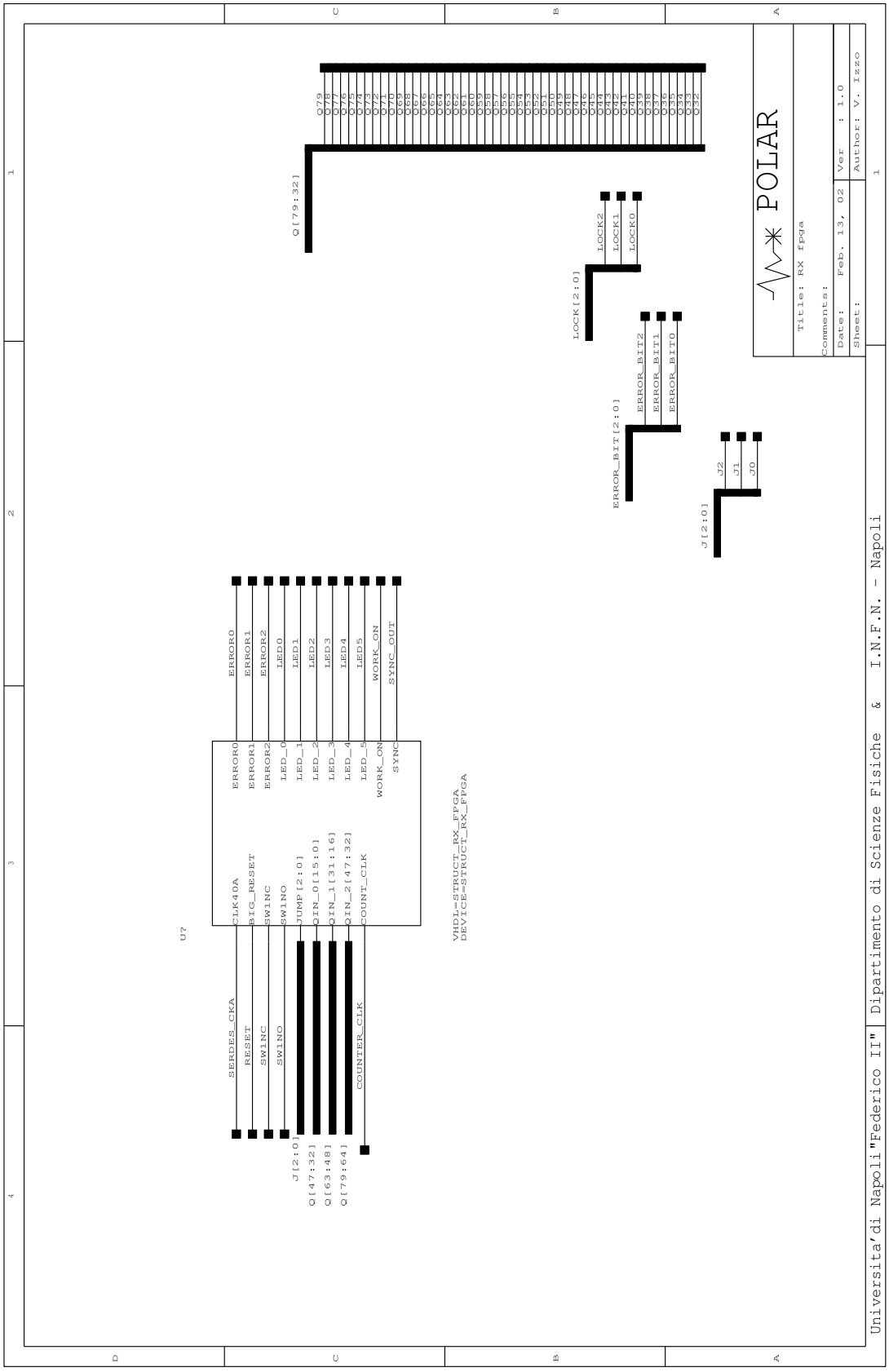
Le tavole 5.1 – 5.8 mostrano gli schematici e le simulazioni, realizzati tramite *CAD* elettronico e simulatore *software*, riguardanti il progetto del sistema TX – RX e il suo funzionamento. La simulazione consente la verifica delle funzioni logiche del progetto e permette di investigare le diverse situazioni che si possono presentare in fase di collaudo, al fine di valutare il comportamento del sistema realizzato su *FPGA*. Nelle tavole sono riportate alcune delle situazioni interessanti per il corretto funzionamento del sistema, nonchè il rilevamento di errori di trasmissione. Viene dunque mostrata l'abilitazione della *local machine* (Tav. 5.4) susseguente alla pressione del pulsante di avvio, il riconoscimento della *signature* e il segnale di *lock* asserito (Tav. 5.5) ed i due segnali di *sync_out*, generati dal trasmettitore e dal ricevitore (Tav. 5.6). La tav. 5.7 mostra l'individuazione di un errore di trasmissione, mentre nella Tav. 5.8 è riportato il comportamento del ricevitore in seguito ad una successione di errori.



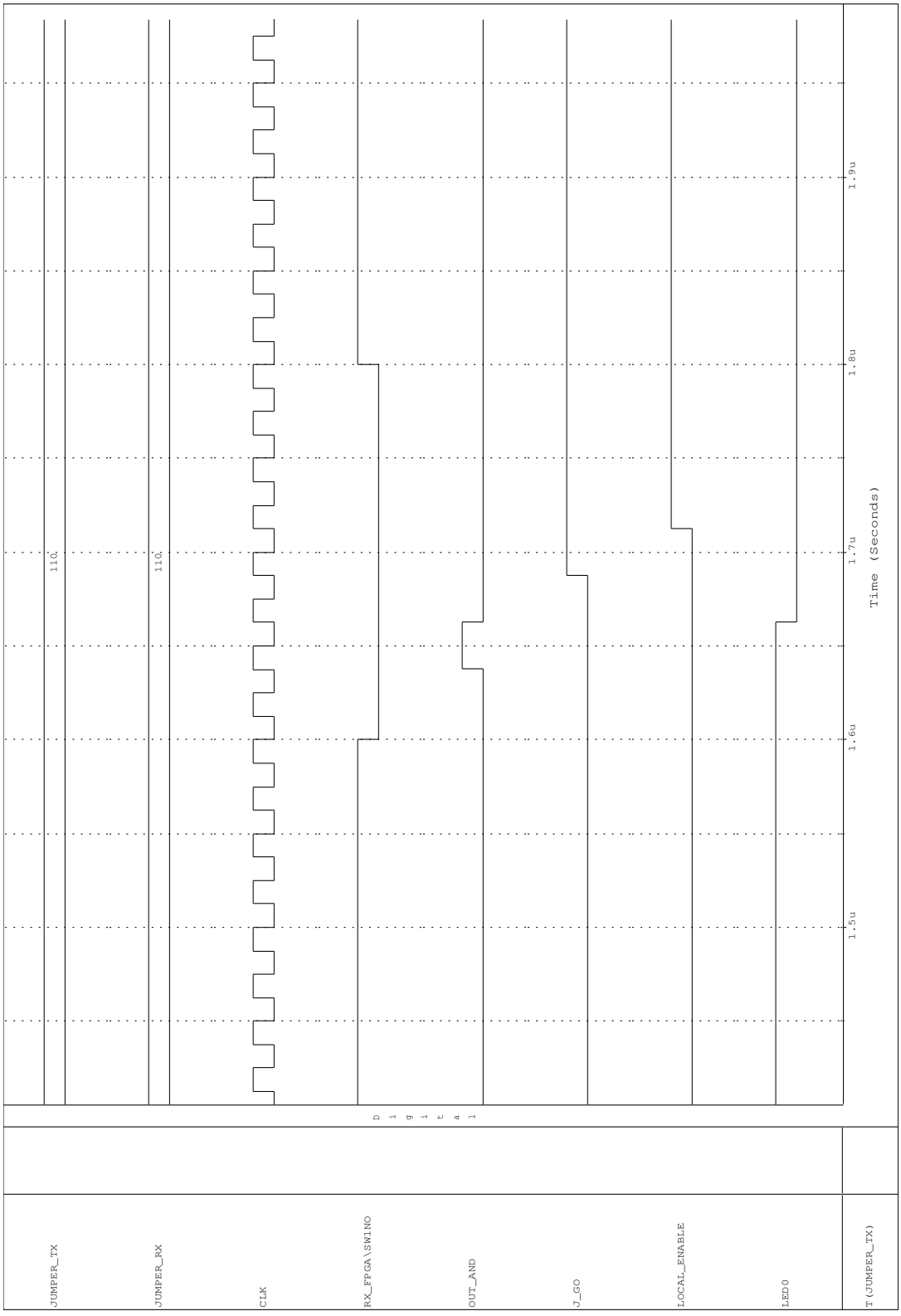
Tav. 5.1 Il sistema TX-RX



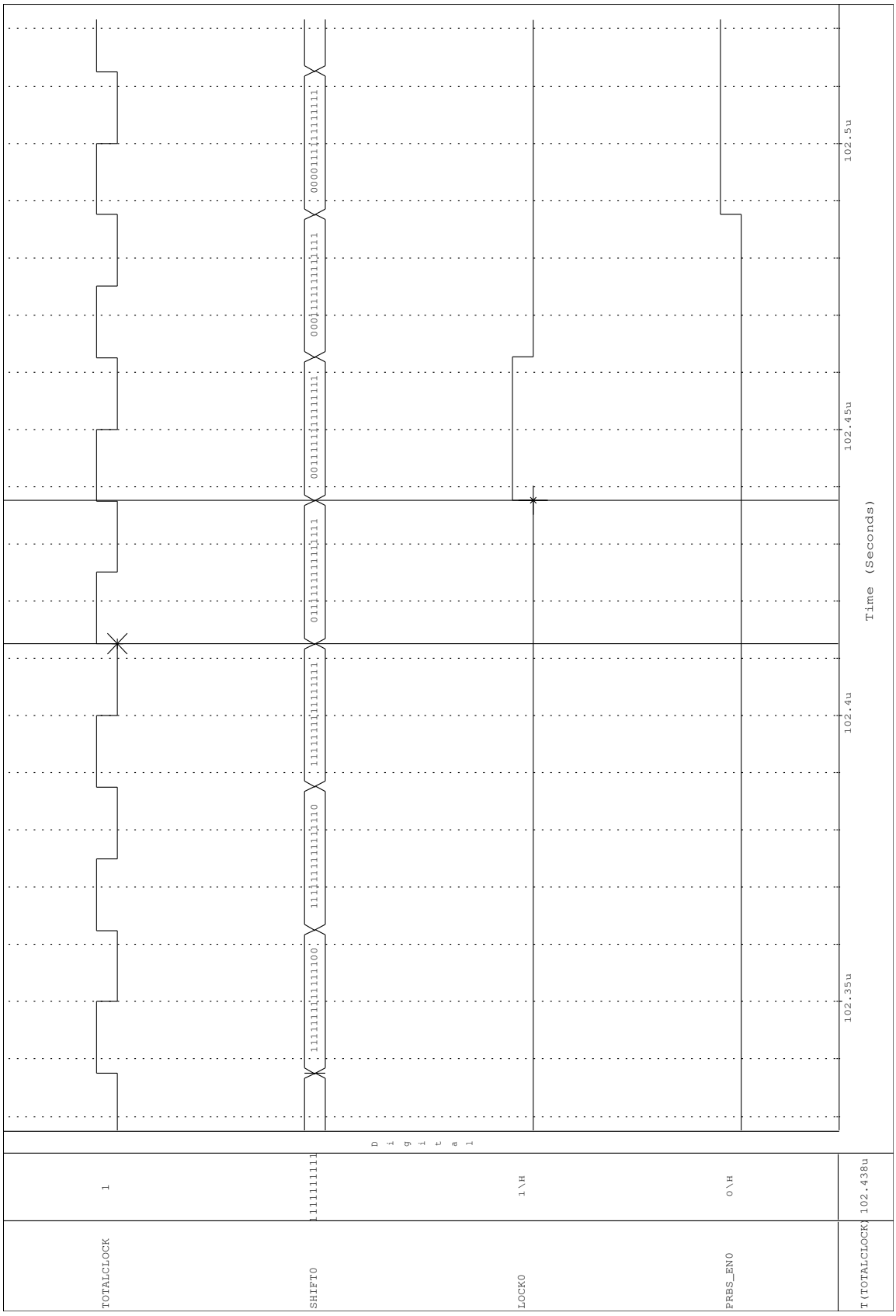
Tav. 5.2 Lo schematico del trasmettitore



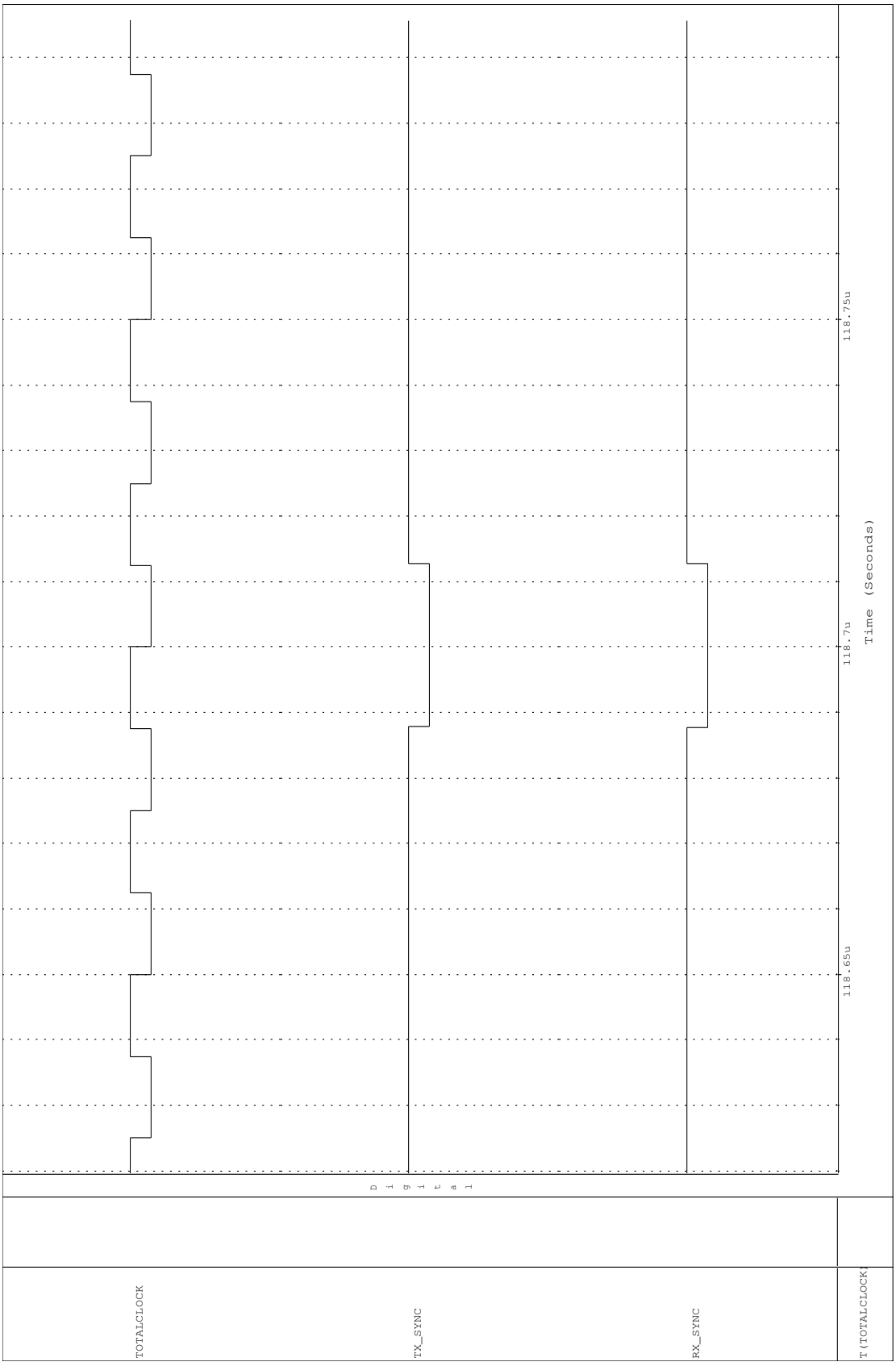
Tav. 5.3 Lo schematico del ricevitore



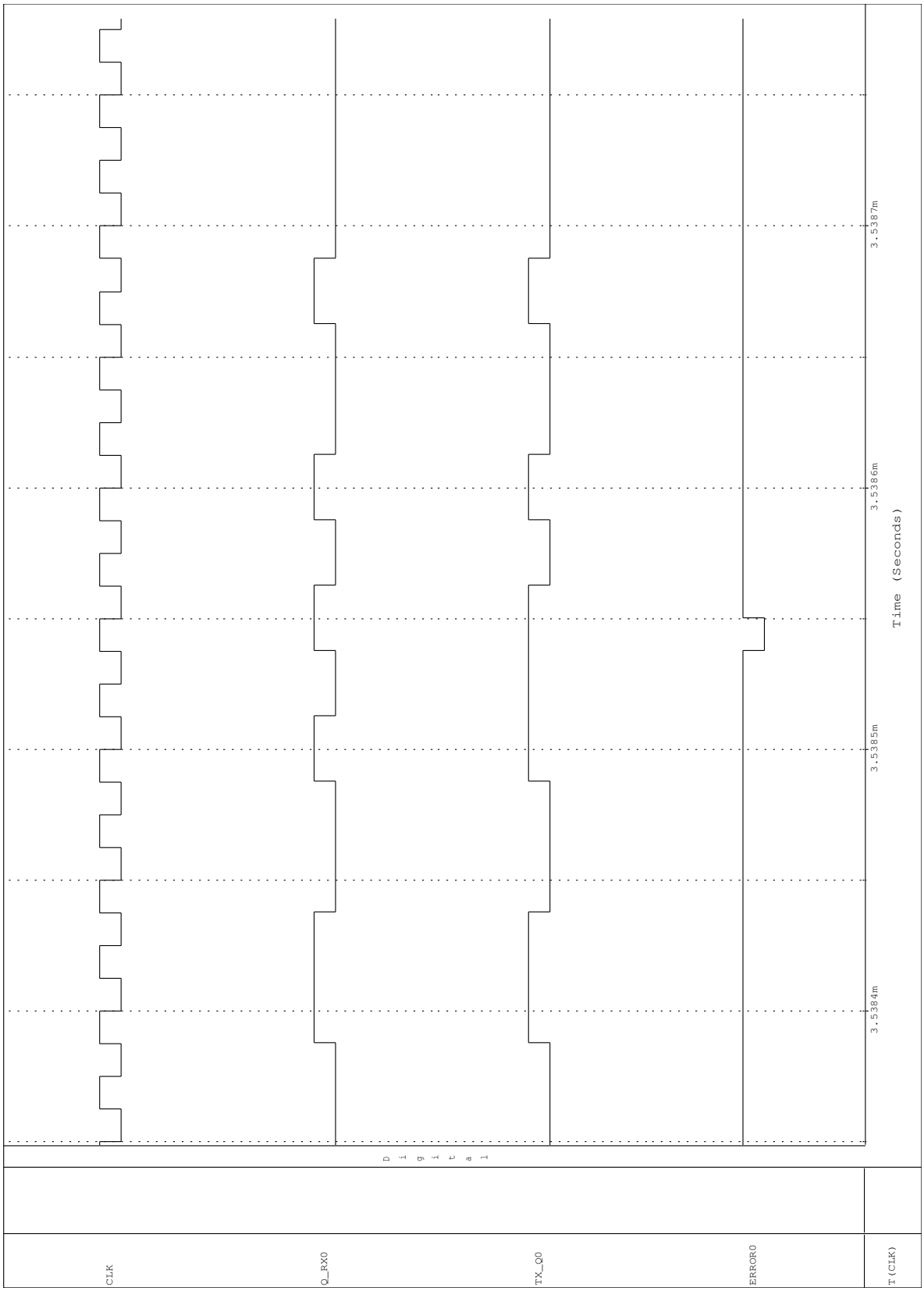
Tav. 5.4 Il segnale di *local enable* a seguito della pressione dello switch



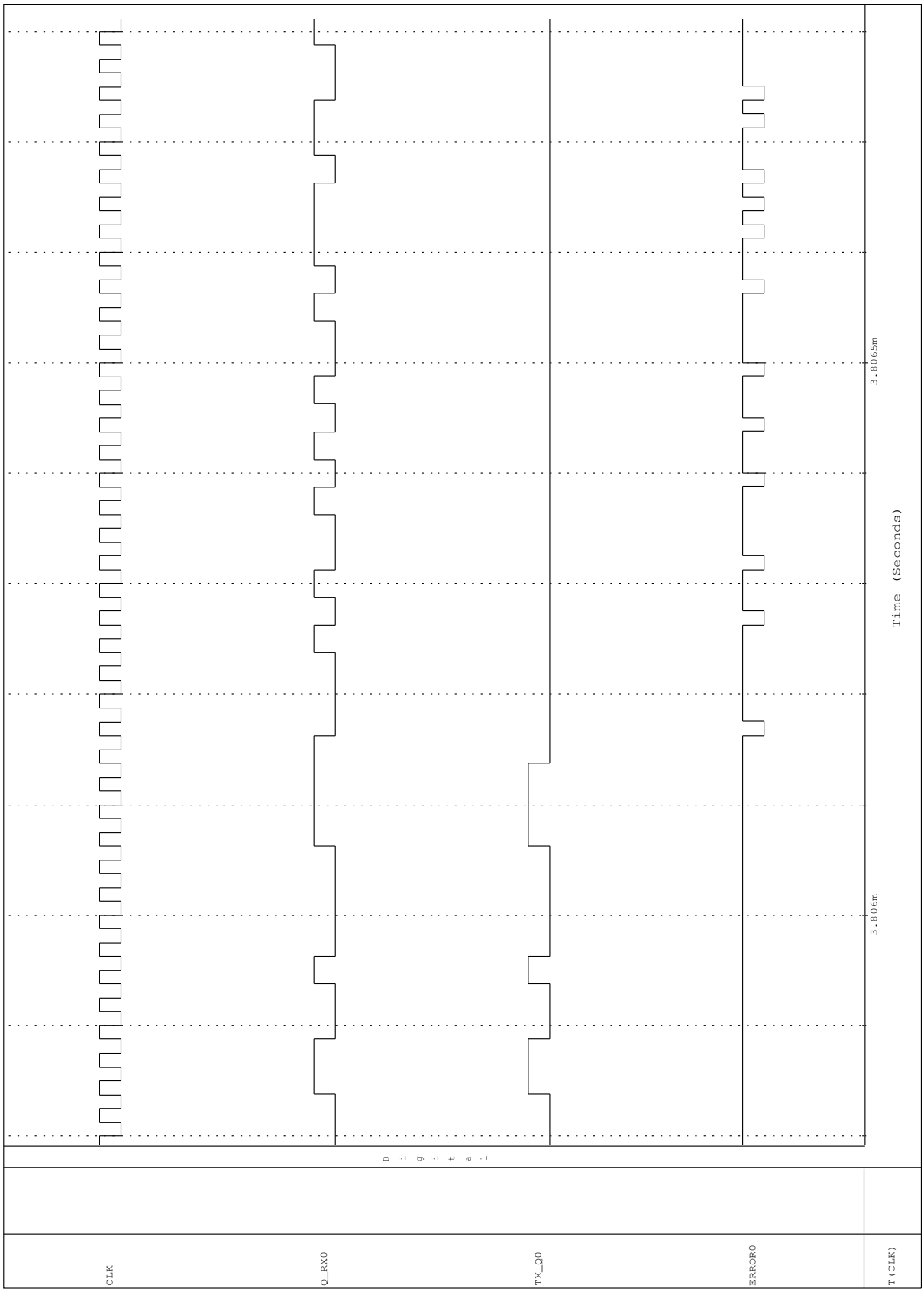
Tav. 5.5 Il segnale di lock a seguito del riconoscimento della signature



Tav. 5.6 I segnali di *sync_out* asseriti dal trasmettitore (*TX sync_out*)
e dal ricevitore (*Rx sync_out*)



Tav. 5.7 Il bit di errore asserito se non c'è corrispondenza tra i dati



Tav. 5.8 La serie di impulsi dovuti ad una successione di errori

La realizzazione

Una adeguata collocazione delle risorse logiche, utilizzate per l'implementazione del progetto nel dispositivo *FPGA*, indubbiamente rende migliore l'utilizzo di tali risorse. Un buon posizionamento consente di ottimizzare le prestazioni in termini di frequenza e di occupazione dell'area. Le prestazioni in frequenza sono espresse dal massimo ritardo interno di propagazione tra due *flip-flop*, compresi i tempi di *clock to output* e dei tempi caratteristici dei segnali di ingresso e uscita. L' occupazione di un dispositivo programmabile si misura attraverso la percentuale di risorse impiegate rispetto a quelle disponibili.

I dati relativi all'utilizzazione delle risorse logiche del dispositivo *FPGA* sono forniti automaticamente dal *tool* di sintesi. In figura 5.33 è mostrato un resoconto sommario di tali dati.

Device utilization summary	TX FPGA		RX FPGA	
Number of bonded IOB_s	59 out of 192	30%	66 out of 192	34%
Number of CLB_s	90 out of 576	15%	159 out of 576	27%
Number of clock IOB pads	1 out of 8	12%	1 out of 8	12%

Figura 5.33 L'utilizzazione delle risorse logiche all'interno dei due dispositivi *FPGA TX* e *FPGA RX*

Le figure 5.34 e 5.35 mostrano il vero e proprio *layout* del dispositivo fisico, con le risorse utilizzate, sia logiche che di interconnessione. Analizzando i *layout* si può notare come le restrizioni imposte sul posizionamento dei piedini di ingresso/uscita e della logica interna abbiano portato il *tool* di sintesi a posizionare i diversi elementi presenti nella logica in modo regolare.

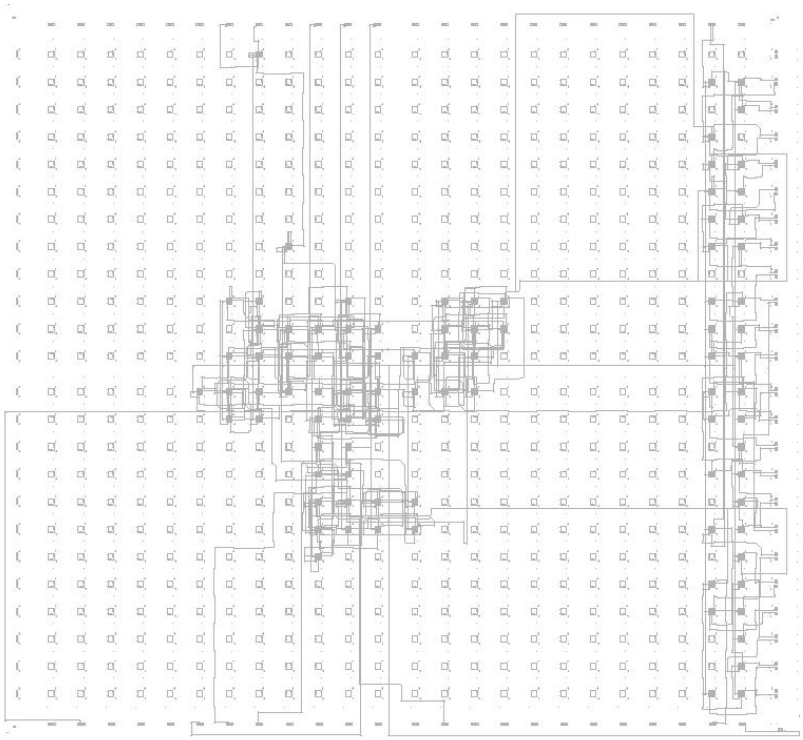


Figura 5.34 La disposizione delle risorse nella *FPGA TX*

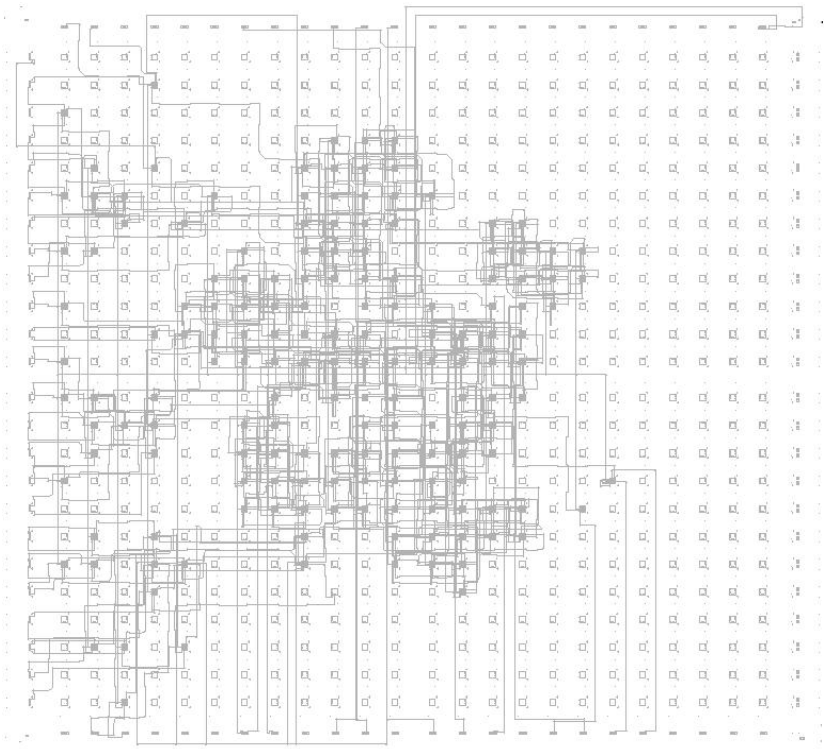


Figura 5.35 La disposizione delle risorse nella *FPGA RX*

La valutazione delle prestazioni in frequenza dell' *FPGA* viene effettuata attraverso l'analisi dei tempi di propagazione dei segnali interni e dei segnali di ingresso\uscita del dispositivo. Le informazioni sulle prestazioni in frequenza all'interno dell' *FPGA* sono fornite dal *tool* di sintesi. Poichè tutti i segnali di ingresso\uscita utilizzano un *flip flop* dell'*IOB*, i tempi di commutazione sono quelli riportati dal *databook* del dispositivo [11].

Descrizione	Min	Max	Unità
Ritardo di propagazione Uscite registrate Scheda TX		5.5	ns
Tempo di Setup Scheda RX	4.5		ns
Tempo di Hold Scheda RX	0		ns

Figura 5.36 Le prestazioni in frequenza delle due *FPGA*

Il collaudo

In figura 5.37 è mostrata l'immagine delle due schede, TX ed RX, utilizzate per il collaudo dei serializzatori e del *backplane*. Il collaudo è stato effettuato nei laboratori di ricerca del Dipartimento di Scienze Fisiche dell'Università *"Federico II"* di Napoli e della Sezione di Napoli dell'*INFN*.

Le schede sono realizzate in FR4 e la logica è gestita da componenti *FPGA* della famiglia *SPARTANXL* (in particolare *XCS30XLPQ240*) prodotte dalla *XILINX*, alimentate a 3.3V [11]. I dati di configurazione per le due *FPGA* sono acquisiti con protocollo seriale da una *PROM* (*Programmable Read Only Memory*) alloggiata nella parte inferiore di ciascuna scheda.

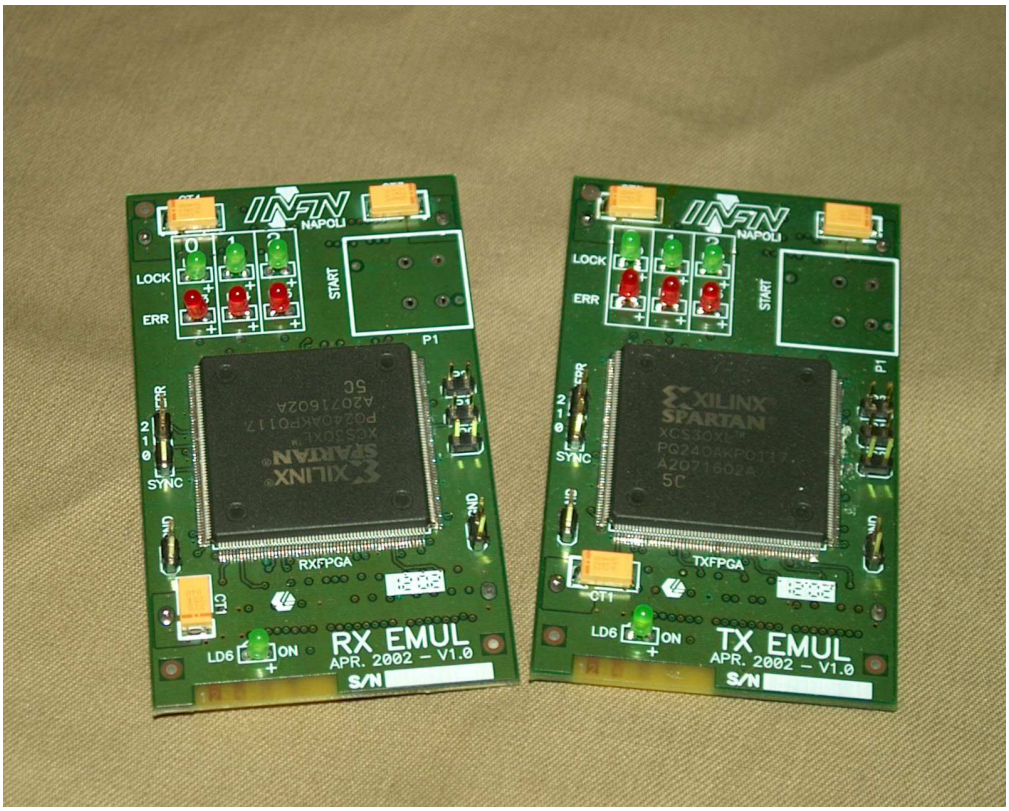


Figura 5.37 Le schede TX ed RX utilizzate per il collaudo dei serializzatori

In figura 5.38 è riportata la misura all'oscilloscopio della latenza di trasmissione dei serializzatori. Tale misura è fatta prendendo come riferimento il segnale di *sync_out* di ciascuna scheda.

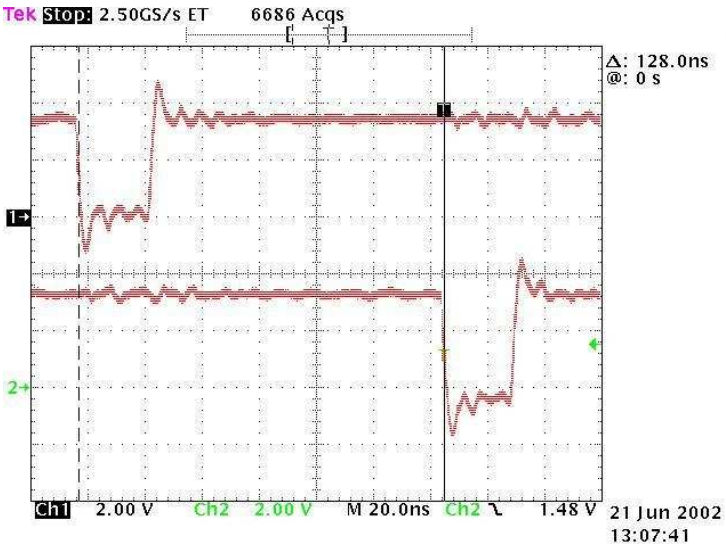


Figura 5.38 La misura della latenza di trasmissione dei serializzatori

Come si vede dalla tav. 5.6 i due segnali sono prodotti dalla scheda TX e dalla scheda RX in corrispondenza del medesimo fronte di salita del clock. Misurando all'oscilloscopio la differenza temporale tra i due segnali si può calcolare la latenza di trasmissione dei serializzatori. Il valore misurato risulta essere di 128±4 ns, in accordo col valore di 5 periodi di clock riportati nelle specifiche dei dispositivi ai quali bisogna aggiungere un *offset* di alcuni nanosecondi, come riportato dal *databook* dei serializzatori.

Durante il collaudo sono stati trasferiti sul *backplane* ~ 1.2×10^{12} bit. Non sono stati riscontrati errori di trasmissione dei serializzatori che, pertanto, possono essere considerati affidabili.