

INCONTRO INFORMATIVO SUI LINGUAGGI DI DESCRIZIONE DELL'HARDWARE (HDL)

RELATORI:

GIULIANO CARDINALI

EMANUEL CONTI

A CURA DI:

<https://www.pidrsm.sm>

<https://www.facebook.com/professionistiperinnovazionedigitale>

PROFESSIONISTI PER L'INNOVAZIONE DIGITALE

— ASSOCIAZIONE —



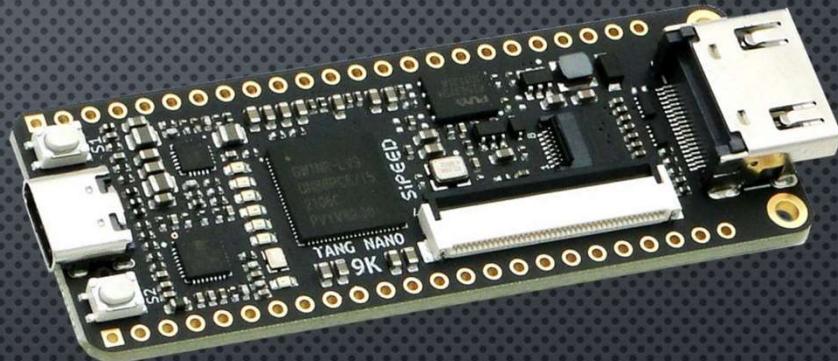
HDL IN PRATICA: PROGETTO, SINTESI E SIMULAZIONE

PRESENTAZIONE DEL KIT DI SVILUPPO

Per questa presentazione si è scelto di utilizzare un kit di sviluppo facilmente reperibile e a basso costo.

Nello specifico si userà il kit di SiPeed modello Tang Nano 9K.

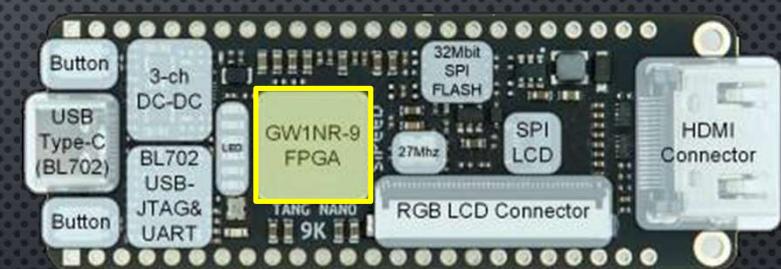
Tang Nano 9K utilizza una FPGA non volatile basata su Flash, di marca GOWIN e appartenente alla famiglia LittleBee®, dotata di RAM integrata (GW1NR). Dispone di poco meno di 9000 LUT/CLB a 4 ingressi, funzioni DSP e diversi moduli tra cui due PLL.



TANG NANO 9K DI SIPEED (FPGA GOWIN GW1NR)

Caratteristiche tecniche

Logic Units (LUT4)	8640
Crystal oscillator	27 MHz
SPI FLASH (bits)	32 M SPI flash (Onboard)
Registers (Flip Flops)	6480
Shadow SRAM SSRAM (bits)	17280
Block SRAM B-SRAM (bits)	468 K (26 x 18K)
Number of B-SRAM	26
User flash (bits)	608 K
SDR SDRAM (bits)	64 M
18 x 18 Multiplier	20
Number of PLL	2
Display interface	HDMI interface, SPI screen interface and RGB screen interface
Debugger	Onboard BL702 chip with USB-JTAG and USB-UART
I/O	4-24 mA, Bus Keeper, pull-up/pull-down resistors, Open Drain output options for each I/O
Connector	TF card slot, 2x24 P 2.54 mm Header pads
Button	2 programmable buttons for users
LED	Onboard 6 programmable LEDs



SISTEMI DI SVILUPPO (IDE)

Esistono due sistemi di sviluppo (toolchain) utilizzabili con TangNano 9K:

- IDE/Toolchain del produttore: GoWin FPGA Designer. Supporta tutte le famiglie di FPGA di GoWin e offre una ottima integrazione di tutti i moduli (hard/soft IP) disponibili. È privo di un simulatore integrato ma può sfruttare un simulatore esterno o una risorsa Cloud accessibile dall'IDE.
- VS Code + plugin Lushay Code di Lushay Labs:

<https://learn.lushaylabs.com/vscode-extension/>

Lushay Code sfrutta la toolchain OSS-Cad-Suite (Yosys, Apicula-nextpnr, openFPGALoader, iverilog, etc.).

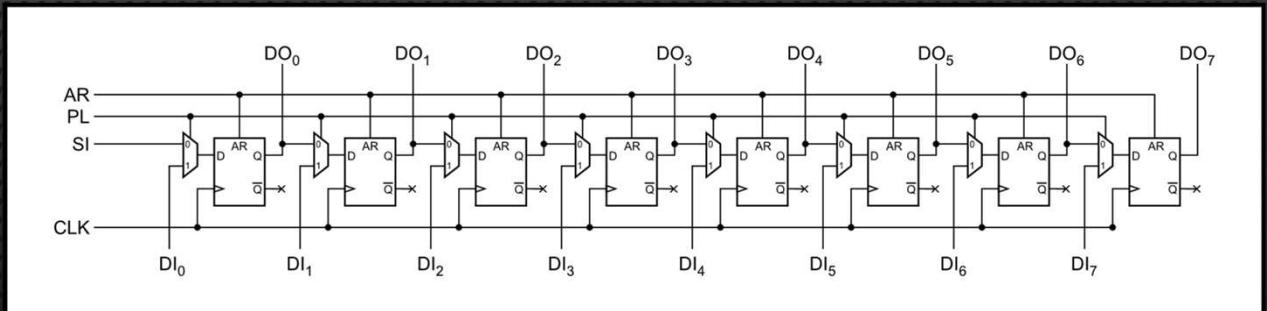
<https://github.com/YosysHQ/oss-cad-suite-build>

- Non copre tutte le funzionalità (hard IP) specifiche della tecnologia di GoWin e permette l'utilizzo solo di un numero limitato di dispositivi.

HDL IN PRATICA: PROGETTO

REGISTRI A SCORRIMENTO (SHIFT REGISTERS)

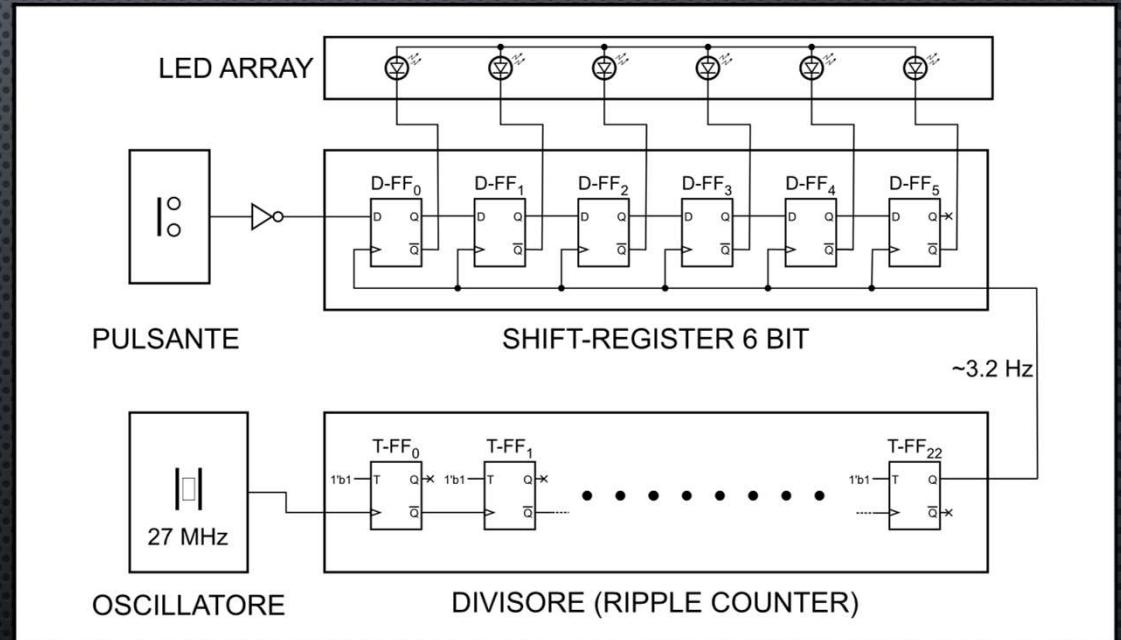
- I registri a scorrimento (Shift Register) possono essere usati per la conversione di dati da parallelo a seriale (serializzatore) o da seriale a parallelo (deserializzatore). Sono costituiti da Flip-Flop di tipo D in cascata (l'uscita di un Flip-Flop entra nel successivo). Gli ingressi di clock dei Flip-Flop sono collegati tutti assieme, così come un eventuale reset sincrono o asincrono.
- Se gli stati dei Flip-Flop possono essere assegnati simultaneamente in un singolo ciclo di clock (Parallel Load), uno Shift Register può essere usato come serializzatore (PISO, Parallel In Serial Out) prelevando l'uscita dell'ultimo Flip-Flop.
- Uno Shift Register che espone tutte le uscite dei propri Flip-Flop può essere usato come deserializzatore (SIPD, Serial In Parallel Out).



UN ESEMPIO PRATICO COL KIT DI SVILUPPO (1)

Al fine di mostrare la meccanica di utilizzo dell'IDE, sfruttando le risorse presenti sul Kit di sviluppo, mostreremo un piccolo progetto che comprende una rete sequenziale sincrona.

Si tratterà di un registro a scorrimento (shift register) a 6 bit il cui ingresso seriale è pilotato da uno dei due pulsanti della scheda di sviluppo. L'uscita parallela dello Shift Register verrà riportata sui led della Tang Nano 9K.

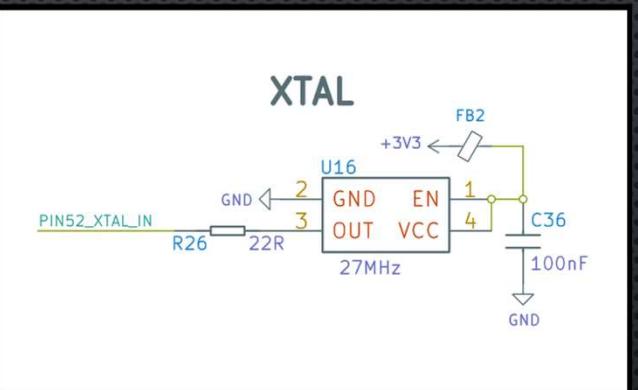
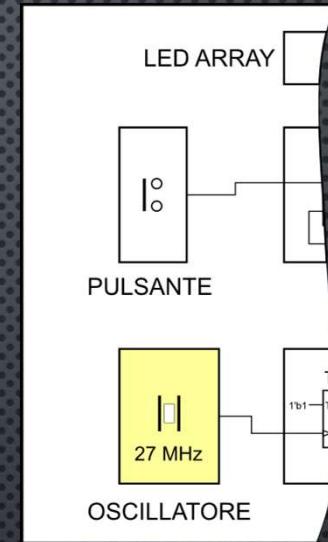
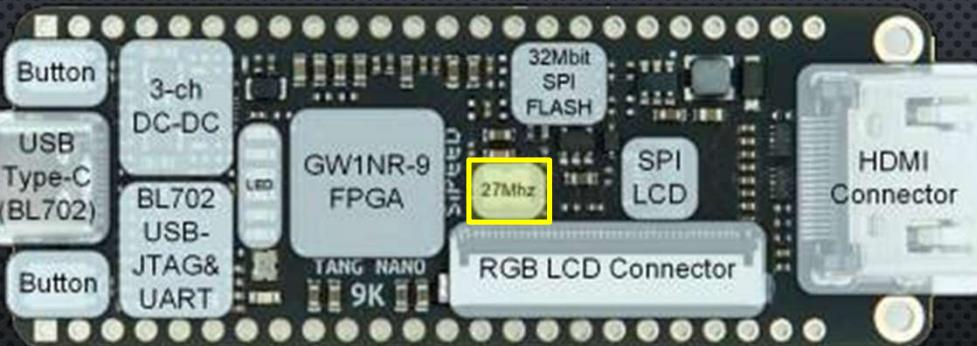


UN ESEMPIO PRATICO COL KIT DI SVILUPPO (2)

TangNano 9K dispone di un **XO (Cristal Oscillator)** che ha una buona stabilità in frequenza, ed è in grado di generare un segnale a 27 MHz.

Per XO si un modulo oscillatore controllato da un cristallo di quarzo. Tale dispositivo ottiene la sua stabilità in frequenza dalla quella intrinseca del cristallo che è modestamente sensibile alle variazioni di temperatura. Questa caratteristica è generalmente specificata in decine di parti per milione (ppm). La precisione iniziale a temperatura ambiente (+25 °C) è data principalmente dalla calibrazione del cristallo da parte del produttore del modulo.

Useremo questo riferimento di tempo, opportunamente trattato, per fornire una cadenza (un clock) alla rete logica che definiremo.



UN ESEMPIO PRATICO COL KIT DI SVILUPPO (3)

L'uscita a 27 MHz ha una velocità eccessiva ai fini del nostro esperimento, in quanto vorremmo poter apprezzare l'effetto dello scorrimento dello Shift Register direttamente sui Led presenti sulla Tang Nano 9K.

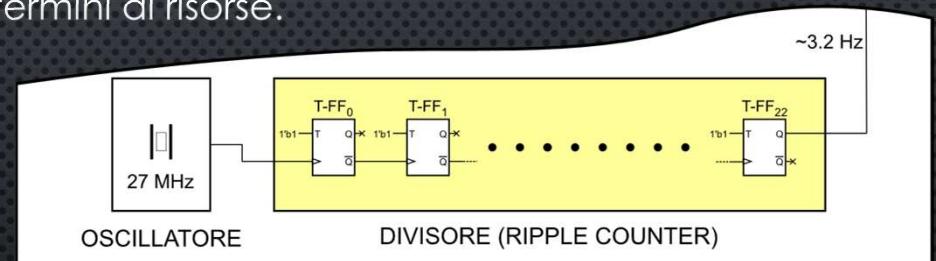
Per portare la velocità ad un valore adeguato (circa 3.2 Hz) dobbiamo «dividere» la frequenza dell'XO per 8.388.608: non si tratta di un valore casuale, bensì corrisponde al numero ottenuto calcolando 2^{23} .

Realizzare un contatore sincrono con 8.388.608 di stati per dividere la frequenza, richiede che la rete combinatoria per il calcolo del valore successivo sia di complessità non trascurabile, nonché di dimensioni ragguardevoli. Inoltre parliamo di una rete logica (un addizionatore) con molti livelli (fino a 23).

Siccome non interessa lo stato intero del contatore ma solo lo stato del bit più significativo e che in questa applicazione abbiamo un solo clock, si può optare per un cosiddetto «Ripple Counter». Oltre ad essere molto veloce è anche poco costoso in termini di risorse.

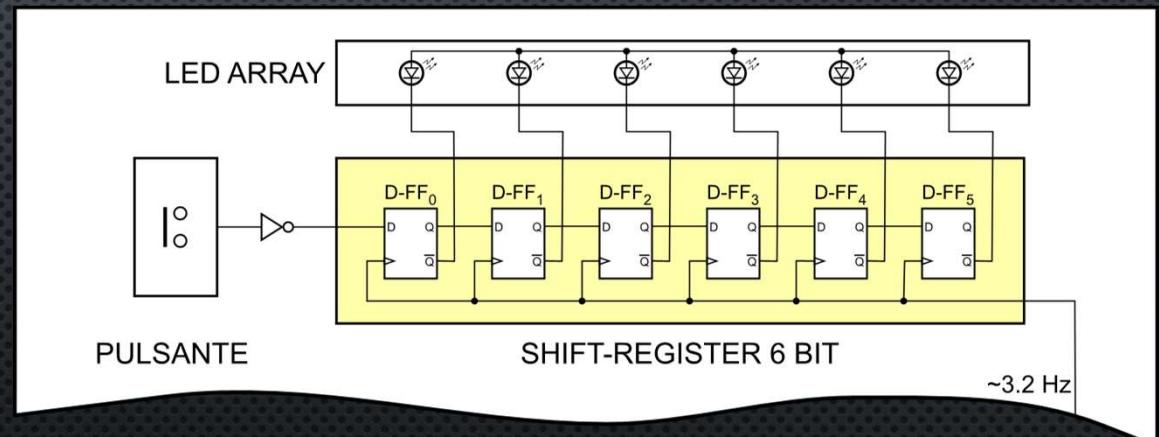
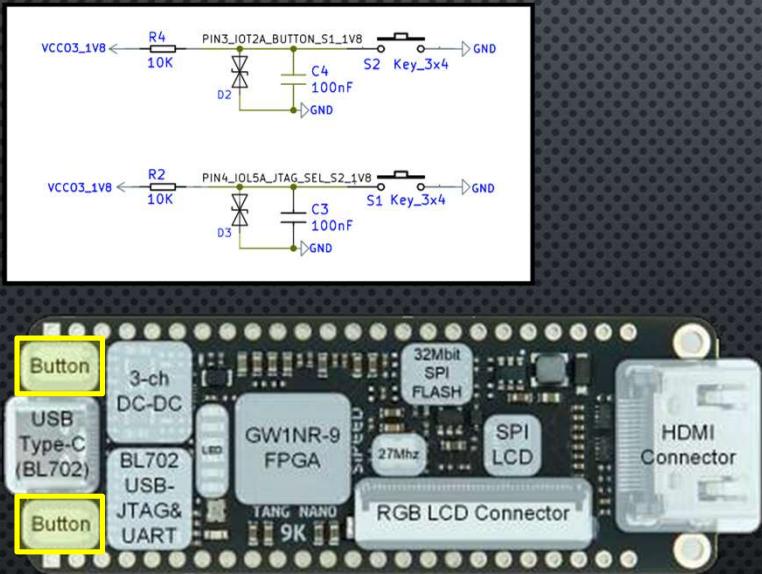
Di contro c'ha che le uscite non commutano in modo sincrono fra loro (non simultaneamente), ma per la nostra applicazione sappiamo che questo non è un problema.

La struttura classica prevede che l'uscita negata di un Flip-Flop di tipo T (T sta per «toggle», cioè che inverte le uscite quando l'ingresso T è ad 1) funga da sorgente di clock per il flip flop successivo. Dopo diverse e successive divisioni per due (fino ad arrivare alla cifra di 8.388.608) della frequenza iniziale di 27 MHz, otteniamo infine una cadenza di 3,218... Hz.



UN ESEMPIO PRATICO COL KIT DI SVILUPPO (4/1)

Il clock che esce dal Ripple Counter è collegato all'ingresso del clock di uno **Shift Register a sei bit**. Per semplicità nella figura è stato omesso il reset asincrono collegato in parallelo a tutti i Flip-Flop. Sudetto reset proviene da uno dei **due pulsanti** della TangNano 9K, opportunamente invertito perché è normalmente attivo basso (da zero quando premuto).

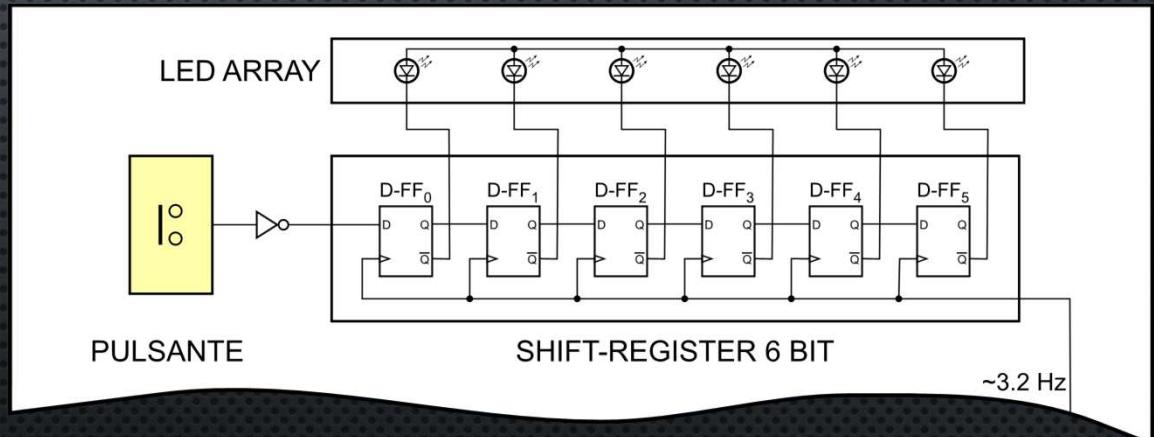
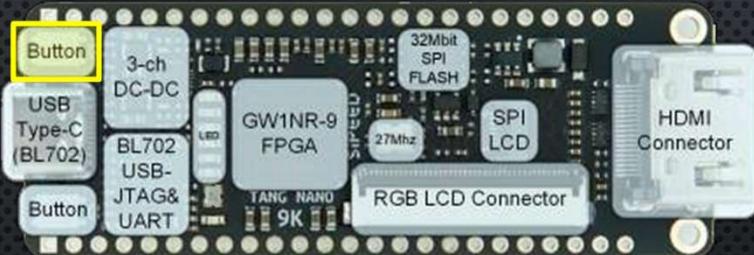


UN ESEMPIO PRATICO COL KIT DI SVILUPPO (4/2)

Il clock che esce dal Ripple Counter è collegato all'ingresso del clock di uno Shift Register a sei bit. Per semplicità nella figura è stato omesso il reset asincrono collegato in parallelo a tutti i Flip-Flop. Sudetto reset proviene da uno dei due pulsanti della TangNano 9K, opportunamente invertito perché è normalmente attivo basso (da zero quando premuto).

L'ingresso seriale del registro a scorrimento è collegato al **secondo pulsante**, presente sulla scheda, anch'esso invertito.

Ad ogni fronte di salita del clock,
lo Shift Register copia lo stato
del pulsante collegato al suo
ingresso e lo trasferisce alle uscite.

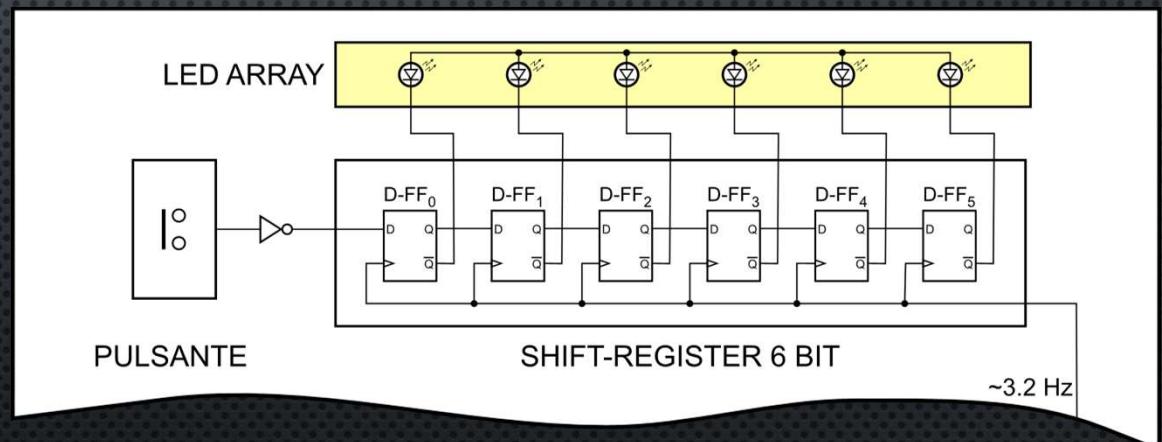
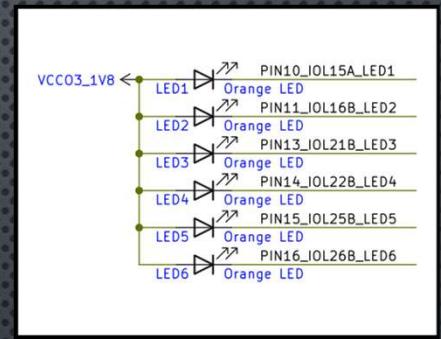
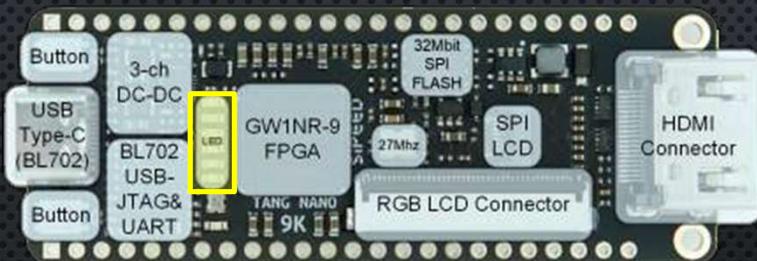


UN ESEMPIO PRATICO COL KIT DI SVILUPPO (5)

Alle uscite dello Shift Register a sei bit sono collegati i led presenti sulla scheda.

Si può notare che i segnali per accendere i led sono prelevati dalle uscite negate dei Flip-Flop dello Shift Register. Ciò è reso necessario per il fatto che, sulla scheda Tang Nano 9K, i led sono tutti «ancorati» al positivo dal lato dell'anodo (i led incorporano anche una resistenza di limitazione della corrente).

Ciò significa che per accendere un led bisogna portare a massa il rispettivo catodo, il che equivale a emettere uno livello zero sull'uscita.



HDL IN PRATICA: UN ESEMPIO DI TRASMISSIONE DATI

LA TRASMISSIONE DEI DATI

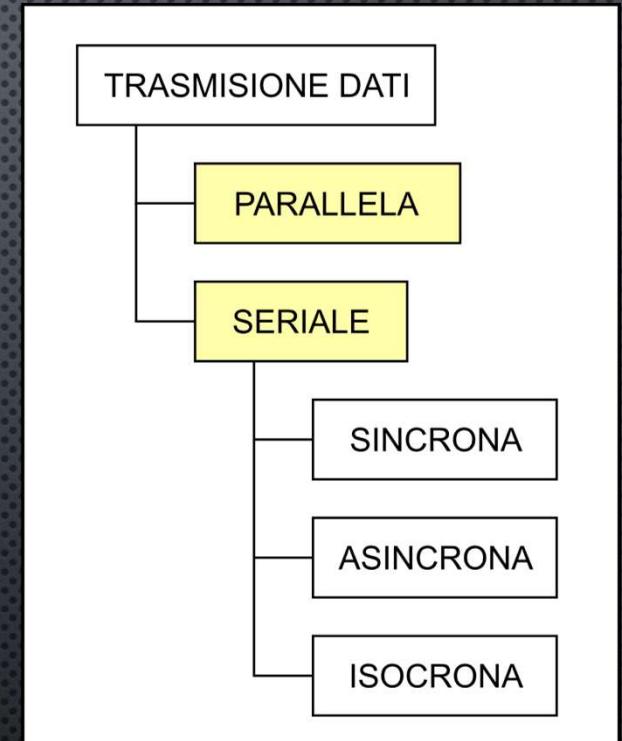
Trasmissione dati Parallela:

In generale è sincrona con la cadenza dettata da un clock. Usa n fili per inviare i bit di una parola tutti quanti contemporaneamente. Ogni bit ha il proprio filo e tutti i bit sono trasmessi in un singolo colpo clock. Ulteriori collegamenti potrebbero essere presenti per gestire il flusso (cioè il cosiddetto handshake) fra trasmettitore e ricevitore.

Trasmissione dati Seriale:

Nella trasmissione seriale un bit segue temporalmente l'altro. Quindi serve un solo filo per trasmettere i dati. Altri collegamenti aggiuntivi possono essere richiesti per gestire il flusso (handshake), ma non sempre presenti.

La trasmissione seriale dei dati può essere suddivisa in tre macro-categorie: sincrona, asincrona e isocrona.



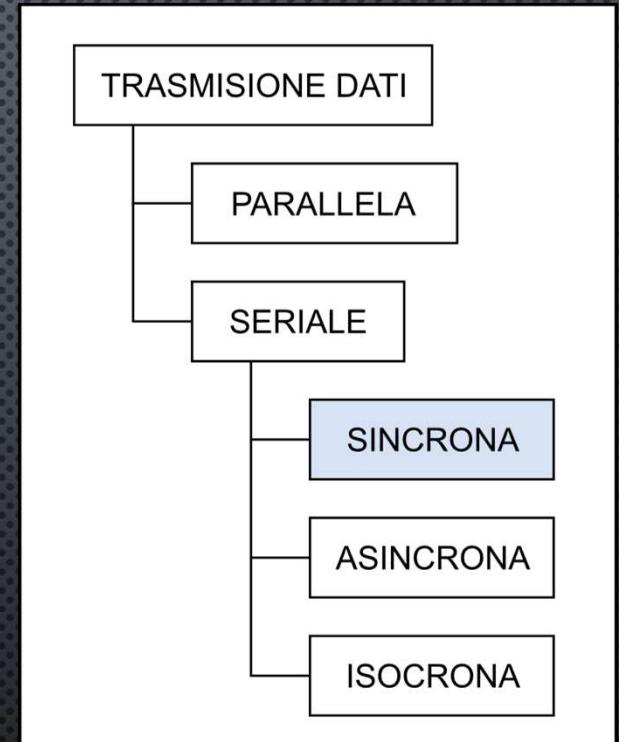
MODI DI TRASMISSIONE DATI SERIALI (1)

Trasmissione sincrona:

Nella trasmissione sincrona, il flusso di bit viene combinato in strutture più lunghe che possono contenere più parole (ad esempio parole da otto bit, cioè i byte). Ogni parola è inviata senza intervalli tra una e la successiva. Il ricevitore dovrà decodificare il flusso di bit e riconoscere e l'inizio e la fine delle parole. Vale a dire che, dal momento che i dati sono trasmessi come una sequenza ininterrotta di 1 e di 0, sarà il ricevitore a ricomporre le informazioni partendo da quella stringa di bit.

Si chiama trasmissione sincrona perché le informazioni sono sempre accompagnate da una cadenza precisa (clock), che può arrivare con un collegamento aggiuntivo o che è incorporata nel flusso dei dati con l'aiuto di particolari codifiche. Le interfacce hardware quali SPI o I2C hanno un clock dedicato su un filo separato. Protocolli come il Manchester, richiedendo però il doppio della banda necessaria, possono incorporare la cadenza direttamente nel segnale, richiedendo così solo un unico collegamento.

La trasmissione seriale sincrona permette di ottenere velocità di trasferimento solitamente anche molto elevate.



MODI DI TRASMISSIONE DATI SERIALI (2)

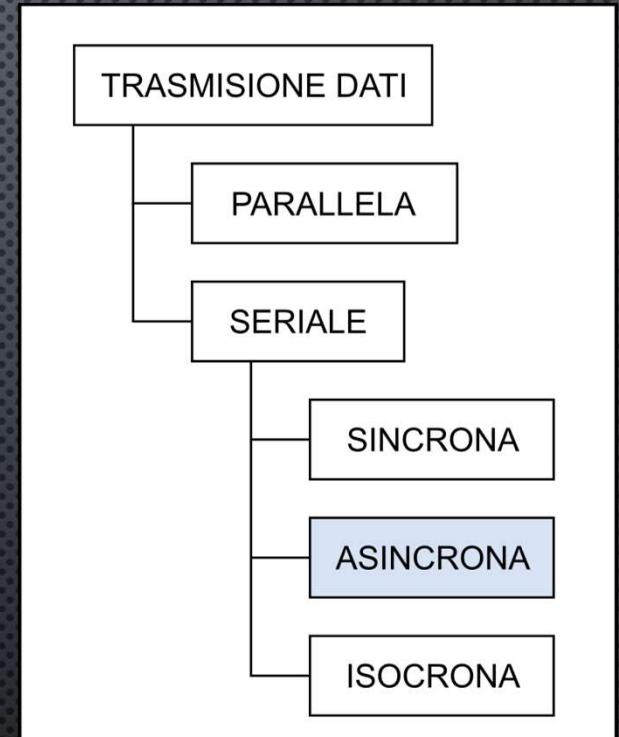
Trasmissione asincrona:

La trasmissione asincrona è così chiamata perché le informazioni sono trasmesse secondo un protocollo concordato senza un preciso riferimento di tempo (clock).

Di solito i bit in sequenza sono raggruppati per formare le parole (byte). Ogni parola è inviata come gruppo di bit a lunghezza fissa e ogni gruppo è gestito in modo indipendente ed inviato solo quando è completamente disponibile. Senza una sincronizzazione, il ricevente non può basarsi su istanti di tempo precisi per prevedere quando arriverà il gruppo successivo.

Per avvisare il ricevente dell'arrivo di un nuovo gruppo, viene aggiunto un bit in più all'inizio di ogni parola (byte). Questo bit, solitamente uno 0, è chiamato bit di start. Per far sapere al ricevitore che la parola è finita, sono aggiunti uno o più bit posti ad 1 alla fine della parola stessa. Questi bit sono chiamati bit di stop. Con questo metodo, la dimensione di ciascuna parola viene aumentata di almeno due bit, cioè da quello di start e dai bit di stop. Inoltre, la trasmissione di una parola può essere seguita da un intervallo di durata variabile (gap).

La trasmissione asincrona, per sua natura, limita la velocità a qualche centinaio di KB al secondo.



MODI DI TRASMISSIONE DATI SERIALI (3)

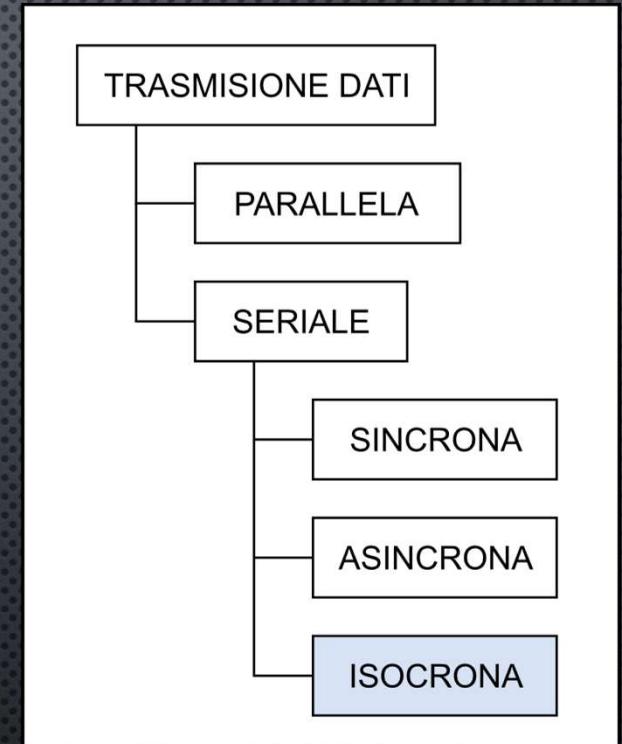
Trasmissione isocrona:

Nella trasmissione isocrona è importante che le informazioni siano ricevute poco tempo dopo che queste sono state trasmesse (bassa latenza), e che il trasporto delle informazioni avvenga su canali che supportino velocità di base medio-alte; in genere la velocità del canale deve essere diverse volte più alta dell'informazione effettiva da trasmettere.

Le applicazioni tipiche della trasmissione isocrona sono le comunicazioni in tempo reale, quali audio, video, dati provenienti da sensori che li generano con cadenza regolare e precisa o per il trasporto dei dati che provengono da edge-computing.

Come accennato, la banda del canale di trasporto deve essere n volte (di solito da 4 volte in su) maggiore della banda richiesta dalle informazioni vere e proprie. Le informazioni possono essere anche accumulate temporaneamente (buffering) purché questo non comporti eccessiva latenza. I dati sono poi trasferiti alla velocità più alta possibile al ricevitore.

Dopo la ricezione e la decodifica, la cadenza della informazioni può essere ricostruita com'era in origine se necessario. Infatti, nelle applicazioni multimediali, per questioni di intellegibilità o di interpretazione, è importante che il flusso esca dal ricevitore alla stessa velocità di com'è entrato nel trasmettitore, seppur con un po' di accettabile ritardo.



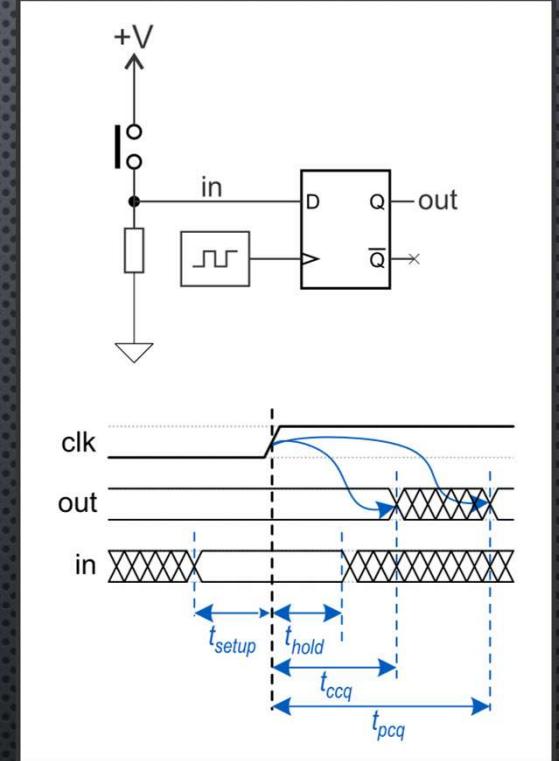
HDL IN PRATICA: AFFIDABILITÀ DELLE RETI LOGICHE

EVENTI ESTERNI ASINCRONI (1)

Consideriamo il disegno a fianco: un circuito con un ingresso proveniente da un pulsante collegato all'ingresso D di un Flip-Flop. Quando il pulsante non è premuto, il flip-flop riceve un valore stabile in ingresso. Se invece si preme il tasto, l'ingresso del flip-flop cambia casualmente rispetto al clock.

Qualcuno potrebbe premere il pulsante proprio mentre il clock sale. Ciò può portare a un fenomeno chiamato «metastabilità», in cui il flip-flop cattura in ingresso un valore a metà strada tra 0 e 1 che può richiedere una quantità illimitata di tempo per risolversi in un valore logico stabile in uscita.

Quando il clock sale e D è stabile, l'uscita può iniziare a cambiare dopo il ritardo (detto di contaminazione) clock-to-Q, t_{ccq} , e deve definitivamente assestarsi sul valore finale entro il ritardo di propagazione clock-to-Q, t_{pcq} .

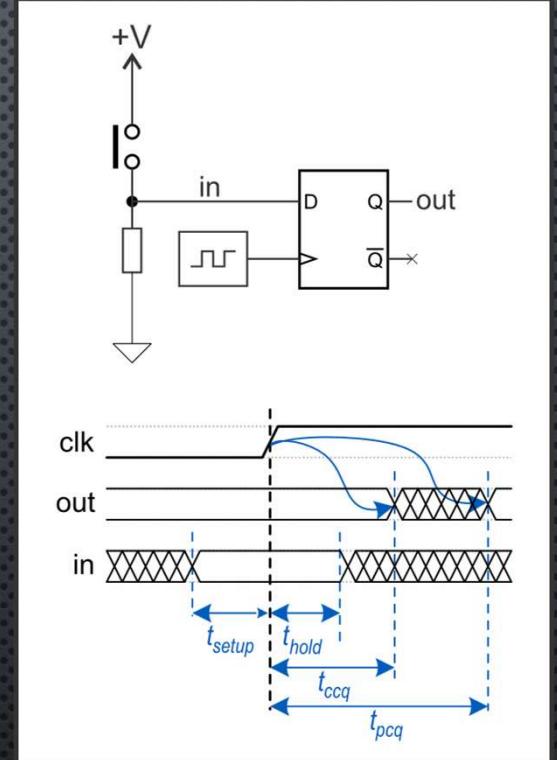


EVENTI ESTERNI ASINCRONI (2)

Il flip-flop ha due parametri fondamentali, t_{setup} e t_{hold} : sono due tempi per i quali è richiesta stabilità dell'ingresso rispetto al fronte del clock su cui lavora il flip-flop. La somma dei tempi di t_{setup} e t_{hold} è chiamata tempo di apertura.

La violazione di uno di questi due tempi porta alla condizione metastabile sull'uscita del flip-flop. Quando questo diventa metastabile non v'è modo di sapere con ragionevole precisione entro quanto tempo uscirà da quello stato e che valore finale prenderà Q.

Questa condizione di asincronismo dei dati rispetto al clock la si trova ogni qualvolta non c'è correlazione fra D e il clock: gli ingressi esterni che provengono da contatti, sensori o segnali di altre apparecchiature, sono generalmente asincroni e quindi potenzialmente causa di metastabilità.

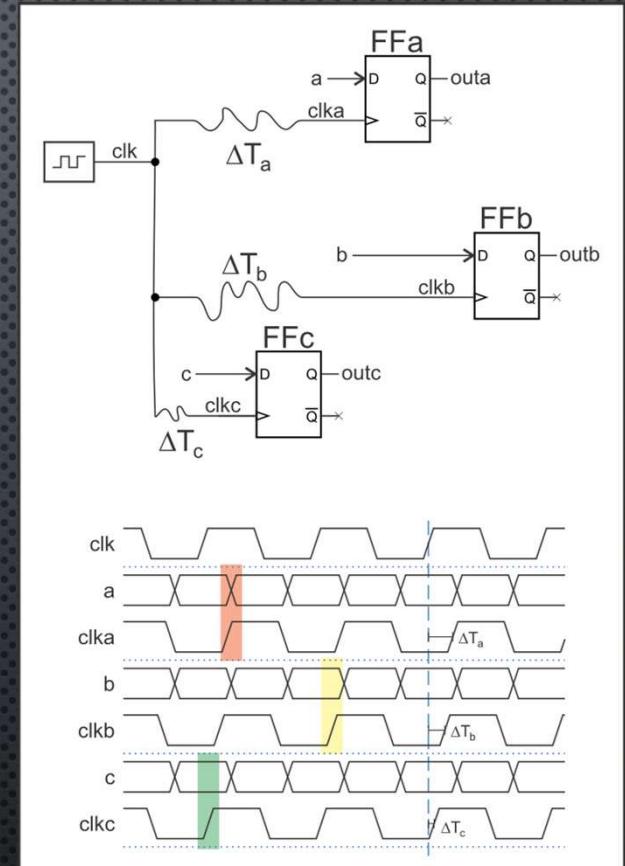


CLOCK SKEW

Nei sistemi reali, il clock potrebbe non raggiungere tutti flip-flop contemporaneamente. Questa variazione nel tempo è chiamata «clock skew». I fili che dalla sorgente di clock arrivano ai registri, o il rumore sul clock stesso, possono generare ritardi differenti per ogni filo. Anche i componenti inseriti nel percorso di clock (per ottenere il cosiddetto clock gating), possono provocare clock skew.

Anche se sulla carta ci siamo assicurati che un dominio di clock sia sincronizzato con tutti i segnali ad esso correlati, la differenza dei tempi di propagazione del clock agli ingressi dei diversi flip-flop di una rete sincrona, possono creare condizioni di metastabilità. Questo vale anche per il ritardo dei dati rispetto al clock.

Nei dispositivi logici programmabili moderni, esistono percorsi prioritari per i segnali di clock per cui la distribuzione ai vari flip-flop e dispositivi è altamente uniforme. Questi segnali hanno una struttura tale da fargli esibire anche un alto fan-out. Il fan-out di una uscita digitale condiziona il numero di ingressi che possono essere collegati ad esso (il carico) senza che il segnale degradi eccessivamente.



CONDIZIONE METASTABILE

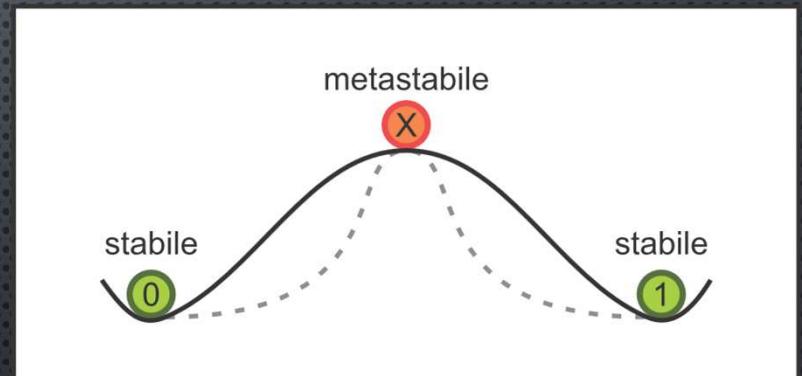
In situazioni reali, quando un flip-flop campiona un ingresso che cambia durante il suo tempo di apertura, l'uscita Q può momentaneamente assumere una tensione compresa tra 0 e VDD (1 logico) che si trova nella «zona proibita». Questa è la cosiddetta condizione metastabile. Alla fine, il flip-flop risolverà l'output in uno stato stabile (0 o 1). Tuttavia, il tempo di risoluzione necessario per raggiungere lo stato stabile può essere infinito o comunque non determinato.

Lo stato metastabile di un flip-flop è analogo a quello di una palla sulla sommità di una collina tra due vallate, come mostrato a lato.

Le due vallate sono gli stati stabili, perché una palla nella valle rimarrà lì finché non viene sollecitata. La cima della collina è chiamata zona metastabile perché la palla potrebbe rimanere lì se sottoposta ad un perfetto bilanciamento di forze. Ma prima o poi, magari a causa di una lieve brezza, la palla rotolerà da una parte o dall'altra.

Il tempo necessario affinché si verifichi questo cambiamento dipende da quanto era il bilanciamento di forze, da quanto è stretta la collina e da fattori disturbanti (rumore).

Ogni dispositivo bistabile ha sempre uno stato metastabile tra i due stati stabili.

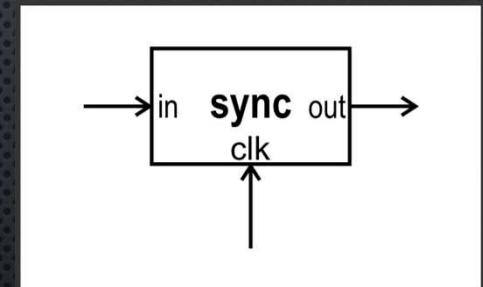


SINCRONIZZATORI (1)

Si consideri un circuito con un ingresso proveniente una sorgente asincrona che si muove, talvolta violando i tempi t_{setup} e t_{hold} e creando quindi, se gestiti con noncuranza, situazioni di metastabilità, cioè causando guasti irregolari del sistema che sono estremamente difficili da rintracciare e correggere. L'obiettivo di un progettista di sistemi digitali dovrebbe essere quello di garantire che, dati gli ingressi asincroni, la probabilità di incontrare una condizione metastabile sia sufficientemente piccola.

Il concetto di probabilità sufficientemente piccola, dipende dal contesto dell'applicazione: per un personal computer forse un errore ogni 10 anni è accettabile, perché basta riavviare il PC se questi si blocca. Per un dispositivo medico o il controllo di un missile balistico, un fallimento al massimo in tutta la vita dell'Universo (10^{10} anni) è meglio.

La soluzione a tali input asincroni consiste nel farli passare attraverso i sincronizzatori, che mettono in correlazione il segnale asincrono in ingresso col clock del sistema digitale e hanno una probabilità molto piccola (ma sempre maggiore di zero) di produrre un valore logico non valido.



SINCRONIZZATORI (2)

Un sincronizzatore, mostrato a lato, è un dispositivo che riceve un input asincrono (in) e un clock (clk). Produce un output (out) che, entro un periodo di tempo limitato, deve avere il livello logico giusto e valido con una probabilità estremamente elevata.

Diciamo che un sincronizzatore fallisce se Q, diventa metastabile. Solitamente un sincronizzatore è realizzato con un flip-flop tipo D di ottima fattura.

Se l'ingresso D è stabile durante l'apertura, Q dovrebbe assumere lo stesso valore di D.
Se D cambia durante l'apertura, Q può assumere il valore 1 o 0, ma non dovrebbe essere frequentemente metastabile, o se lo diventa non dovrebbe rimanere a lungo in quella condizione.

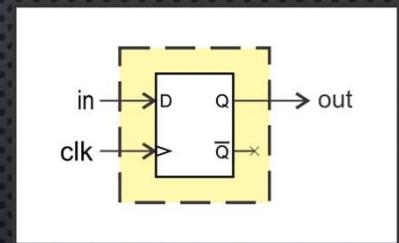
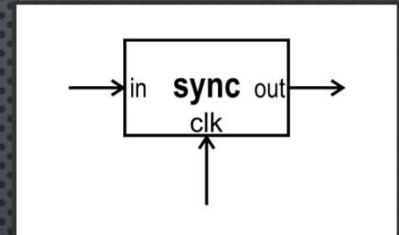
L'affidabilità del sistema viene solitamente misurata in MTBF (Mean Time Between Failures) che è il tempo medio che intercorre tra i guasti del sistema stesso. È calcolato come il reciproco della probabilità che il sistema fallisca in un dato secondo:

$$MTBF = \frac{1}{P(\text{Fallimenti})/\text{sec}}$$

Una tipica equazione per calcolare l'MTBF è:

$$MTBF = \frac{e^{S/\tau}}{T_w F_c F_D}$$

Dove T_w è la somma di t_{setup} e t_{hold} , F_c è la frequenza del clock, F_D è il numero di transizioni al secondo all'ingresso D, S è il tempo in cui è stimato che si risolva l'evento metastabile (di solito un ciclo di clock) e τ è un parametro legato alla tecnologia del dispositivo (tipicamente un gate delay).



HDL IN PRATICA: LA CODIFICA «MANCHESTER»

LA CODIFICA MANCHESTER

La codifica Manchester (dal nome dell'università dove è stata sviluppata) è una codifica binaria (o se vogliamo una modulazione) a scostamento di fase (2-BPSK).

Tale codifica garantisce frequenti transizioni di potenziale sulla linea di trasmissione, che sono direttamente proporzionali alla frequenza del clock, agevolando quindi il recupero (sincronizzazione) di quest'ultimo sul ricevitore. Il codice Manchester ha sempre una transizione a metà di ciascun bit.

Soltanente il segnale viene privato della componente continua che, al fine della interpretazione dei dati, non è rilevante. Normalmente, dal punto di vista elettrico, si opta per una trasmissione differenziale.

La linea di trasmissione necessita di una banda almeno doppia rispetto a quella delle informazioni.

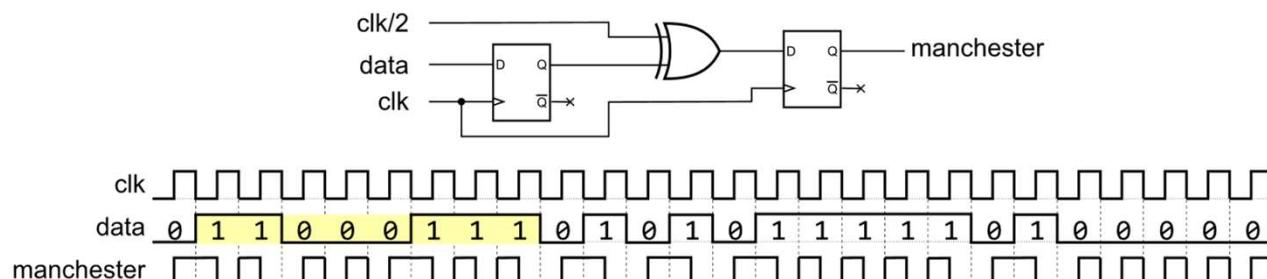
Portando anche l'informazione di cadenza (clock) assieme ai dati, è considerata una trasmissione sincrona, anche se non utilizza una linea separata per il clock come fanno altri tipi di interfacce sincrone.

CODIFICATORE MANCHESTER (1)

Per realizzare un Codificatore Manchester nella sua forma più semplice, basterebbe operare uno XOR fra il clock (clk) e i dati (convenzione di Stallings, standard IEEE 802.x).

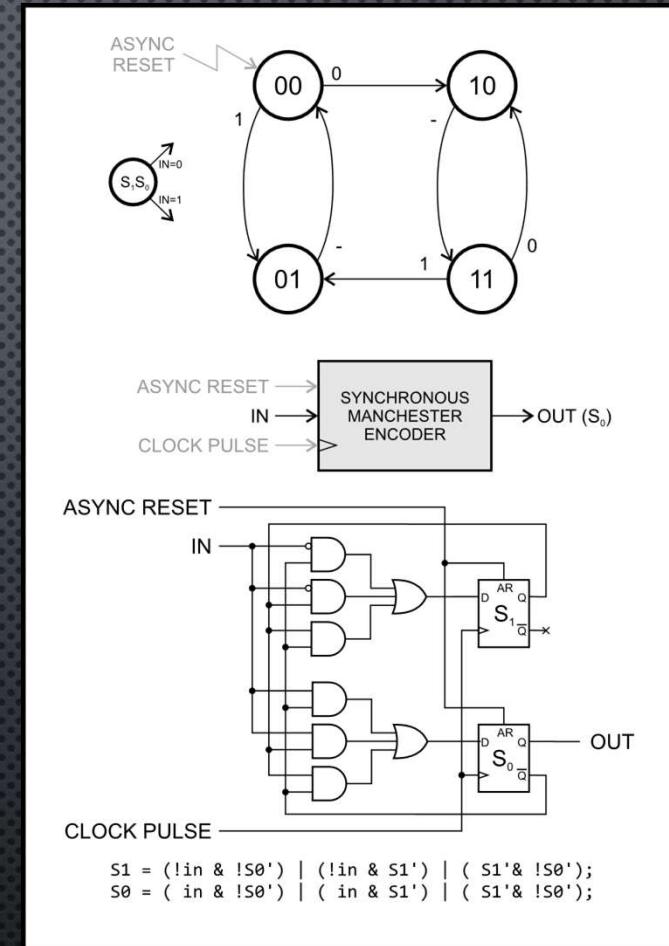
Per migliorare l'affidabilità dell'encoder è stato aggiunto un sincronizzatore (flip-flop) all'ingresso dei dati.

Sebbene aggiunga del ritardo al segnale d'uscita dell'encoder, un secondo flip-flop è stato aggiunto allo scopo di eliminare eventuali alee statiche generate dalla porta XOR. Infatti, a seconda della tecnologia e dell'architettura relativa all'implementazione della porta XOR, dal momento che la funzione è costituita da due minterm disgiunti, esiste la probabilità che si possano generare alee statiche.



CODIFICATORE MANCHESTER (2)

- Una struttura alternativa per il codificatore Manchester vede l'utilizzo di una macchina a stati per la generazione del segnale.
- Il vantaggio portato da questa architettura risiede nel fatto che è sufficiente solo un clock per il suo funzionamento e che è del tutto annullata la probabilità di avere alee statiche in uscita.
- Di contro utilizza più parti per realizzare la funzione.



DECODIFICATORE MANCHESTER

- La decodifica di un segnale Manchester, prevede il recupero della cadenza (clock) assieme all'informazione vera e propria (i dati).
- Nella pratica, per i segnali provenienti da canali ad altissima velocità, il recupero del clock è fatto mediante particolari circuiti (PLL, Phase Locked Loop, cioè «circuito ad anello ad aggancio di fase») in grado di sincronizzarsi su una frequenza o un multiplo di essa. Tali dispositivi, dopo un esiguo numero di cicli del segnale, generano un clock perfettamente sincronizzato con quello in esso contenuto. Questo clock può essere utilizzato per recuperare le informazioni originali dal segnale stesso.
- Se la comunicazione è costituita da un segnale non particolarmente veloce, si possono usare tecniche alternative che non richiedono l'utilizzo di parti analogiche come i PLL. Le architetture di questo tipo si basano sulla rilevazione dei cambiamenti del segnale accompagnate a campionamenti ritardati, entrambi ottenibili con circuiti logici. In questa presentazione utilizzeremo questo secondo approccio.

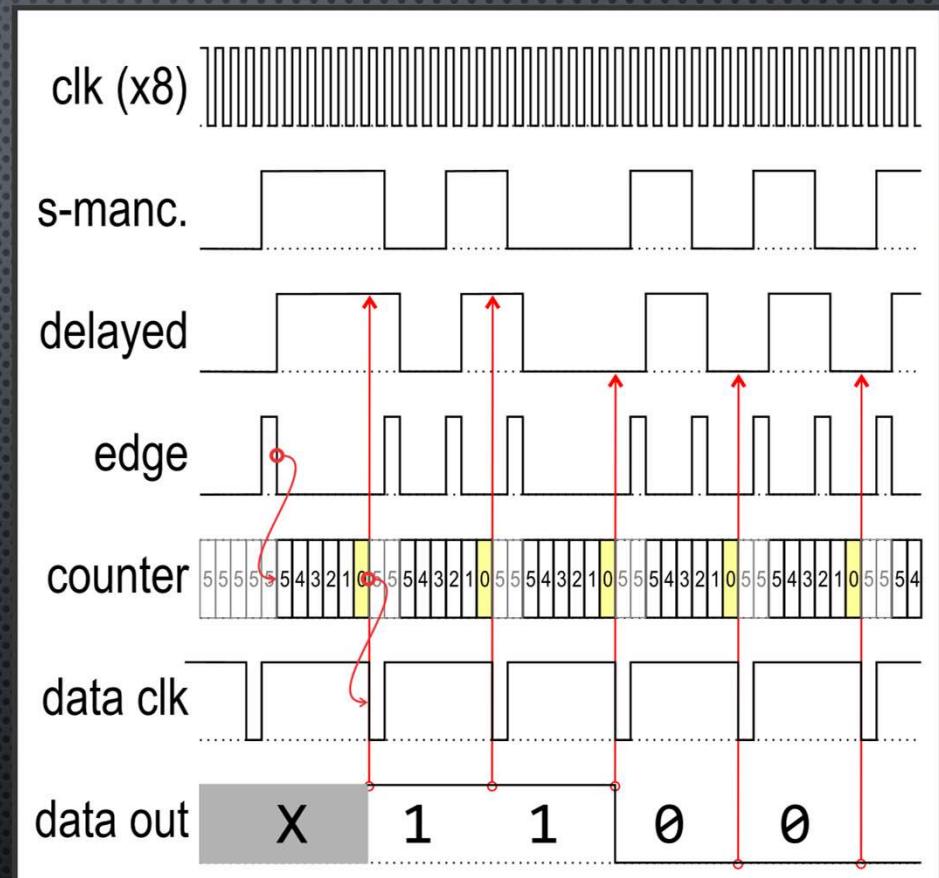
DECODIFICATORE MANCHESTER

Il decodificatore estraе il clock (data clock) e i dati (data out) dal segnale (s-manc).

Facendo lo XOR fra il segnale (s-manc) e una sua copia ritardata (delayed) otteniamo il segnale edge.

Il segnale edge alimenta un ingresso di una macchina a stati (counter/sequencer). Se quest'ultima è in attesa di uno stimolo e se edge è a uno, la macchina a stati, nel nostro caso partendo da 5, innesca un conteggio a ritroso che, arrivato a zero, determina il momento nel quale il decoder campiona il valore del dato contenuto nel segnale di ingresso (s-manc).

Durante il conteggio, l'ingresso edge è ignorato.



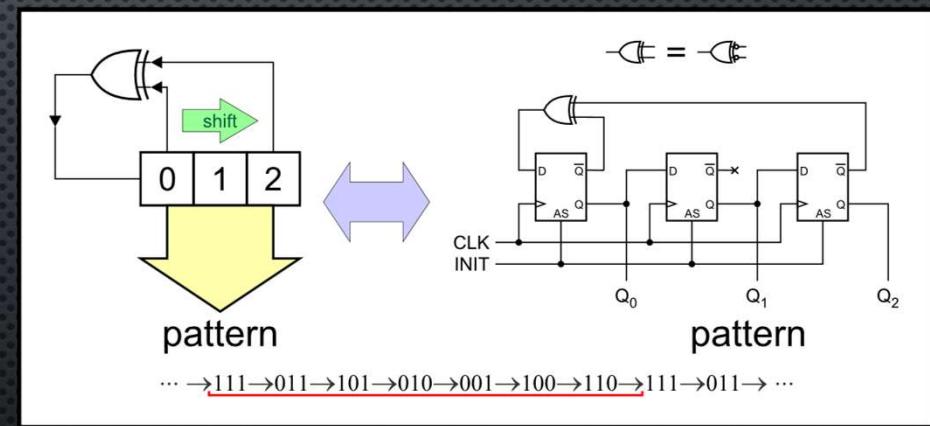
HDL IN PRATICA: (UTILE INTERMEZZO) GLI LFSR

LFSR (LINEAR FEEDBACK SHIFT REGISTER)

LFSR: è una sorta di *Uroboro* (il serpente che si mangia la coda) dell'elettronica digitale, in cui l'output di uno Shift Register è manipolato e reimmesso nel suo ingresso in modo tale da far sì che la funzione cicli all'infinito attraverso una sequenza di numeri tutti diversi (pattern previsto).

Gli LFSR sono molto veloci, semplici da realizzare e sono molto utili per un'ampia varietà di applicazioni.

Una delle forme più comuni di LFSR è formata da un semplice Shift Register con feedback che partono da due o più punti (detti *tap*) presi nella catena dei registri. A lato un esempio di LFSR a tre bit con due tap.



LFSR TESTBENCH

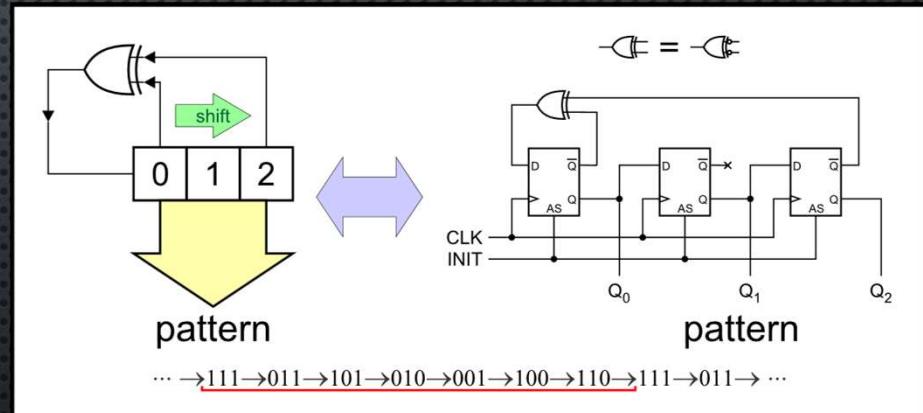
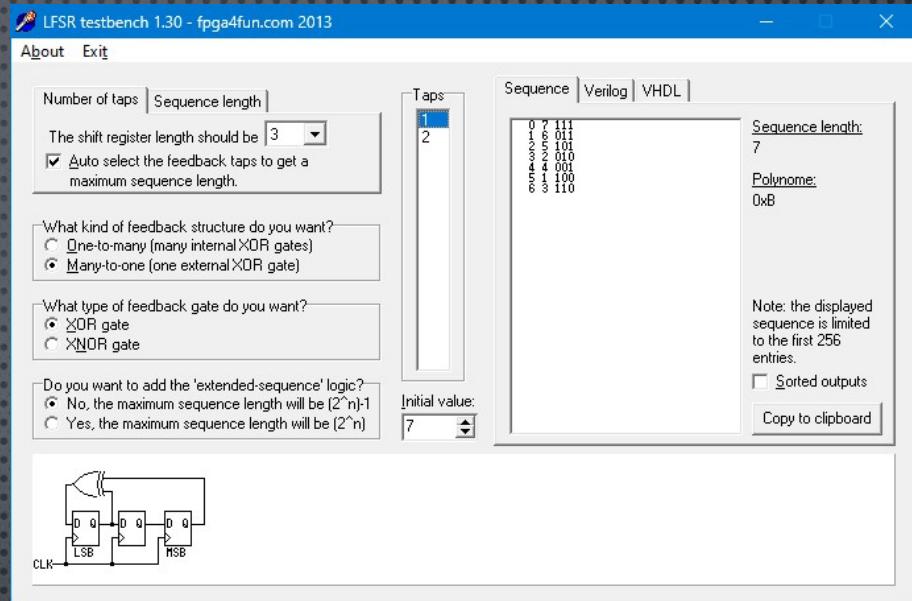
Per sintetizzare un LFSR si può usare uno strumento open source chiamato LFSR testbench, scaricabile a questo URL:

<https://www.fpga4fun.com/files/LFSRTestbench.zip>

Normalmente, nella configurazione più semplice, il numero dei codici prodotti (lunghezza della sequenza) è minore di 2^n dove n è la cosiddetta «lunghezza» in termini di bit dello LFSR.

Aggiungendo alcune porte è possibile avere generatori LFSR con lunghezza della sequenza pari a 2^n .

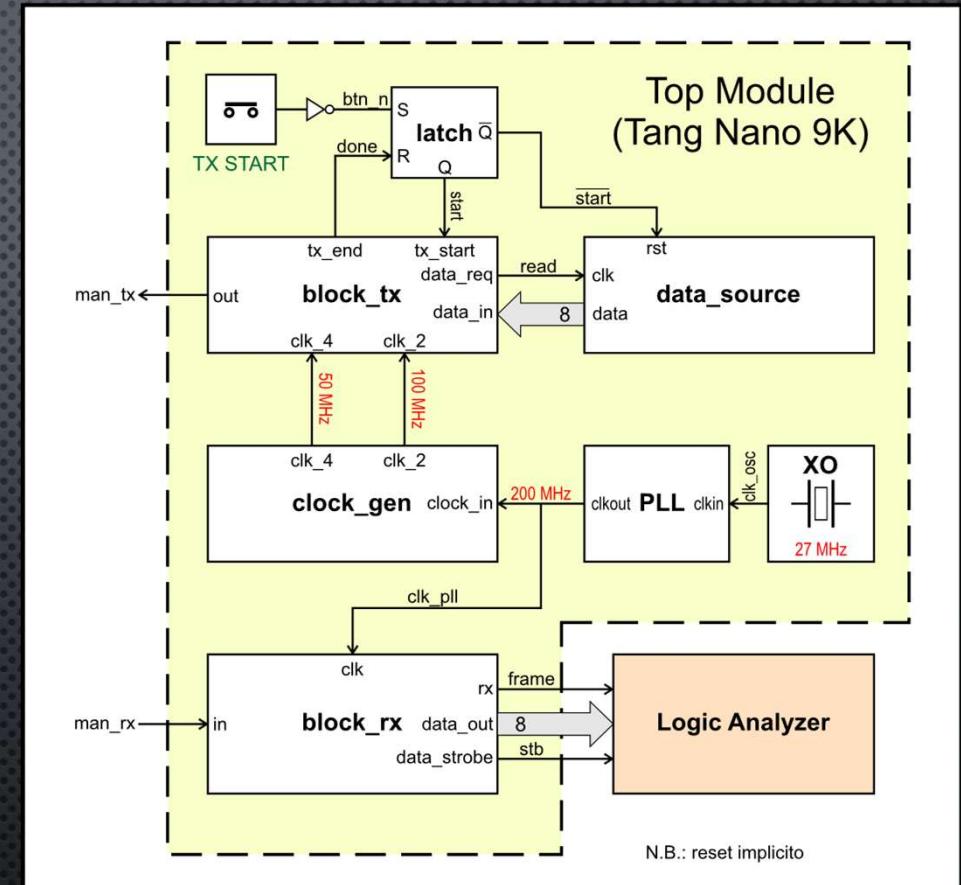
Tale software permette di specificare diversi parametri dello LFSR e fornisce il risultato sotto forma di schema elettrico, sequenza prodotta, codice Verilog o codice VHDL.



HDL IN PRATICA: ESPERIMENTO CON LA CODIFICA MANCHESTER

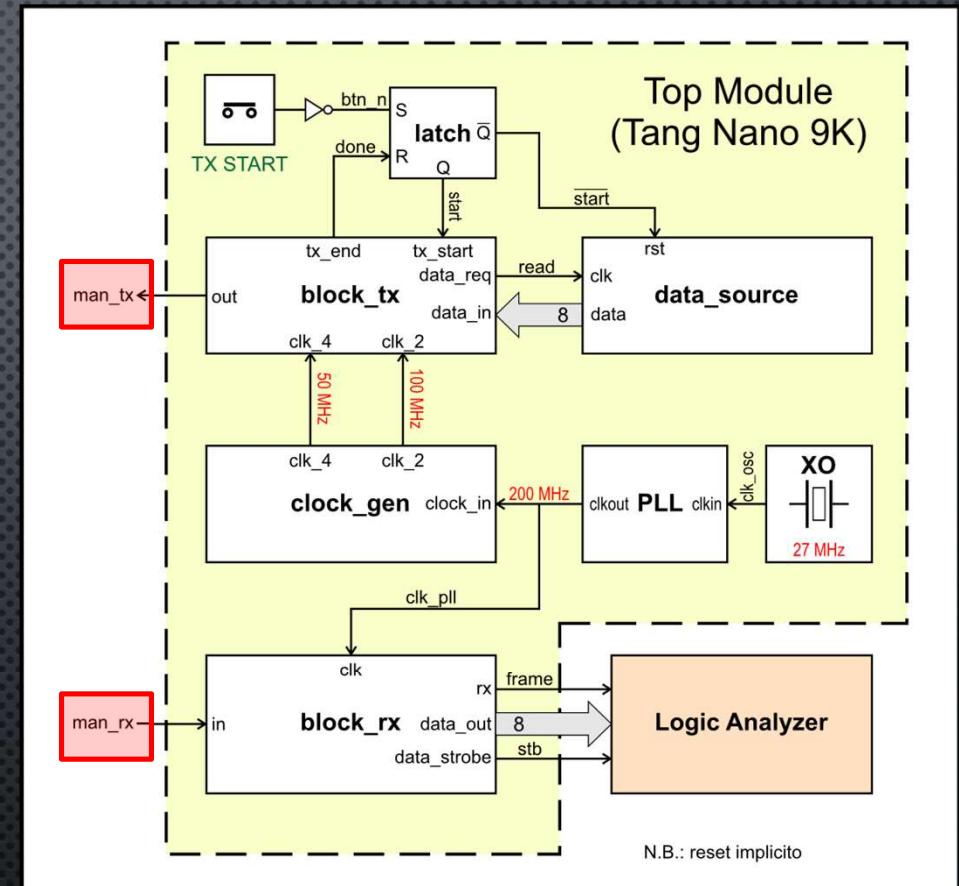
ESEMPIO: UN TRANSCEIVER MANCHESTER (1)

- Il nostro esperimento, la cui struttura è mostrata a lato, è interamente realizzato su una Tang Nano 9K di SiPeed.



ESEMPIO: UN TRANSCEIVER MANCHESTER (2)

- Il nostro esperimento, la cui struttura è mostrata a lato, è interamente realizzato su una Tang Nano 9K di SiPeed.
- Possiamo osservare ingresso ed uscita dei segnali manchester (rispettivamente man_rx e man_tx).



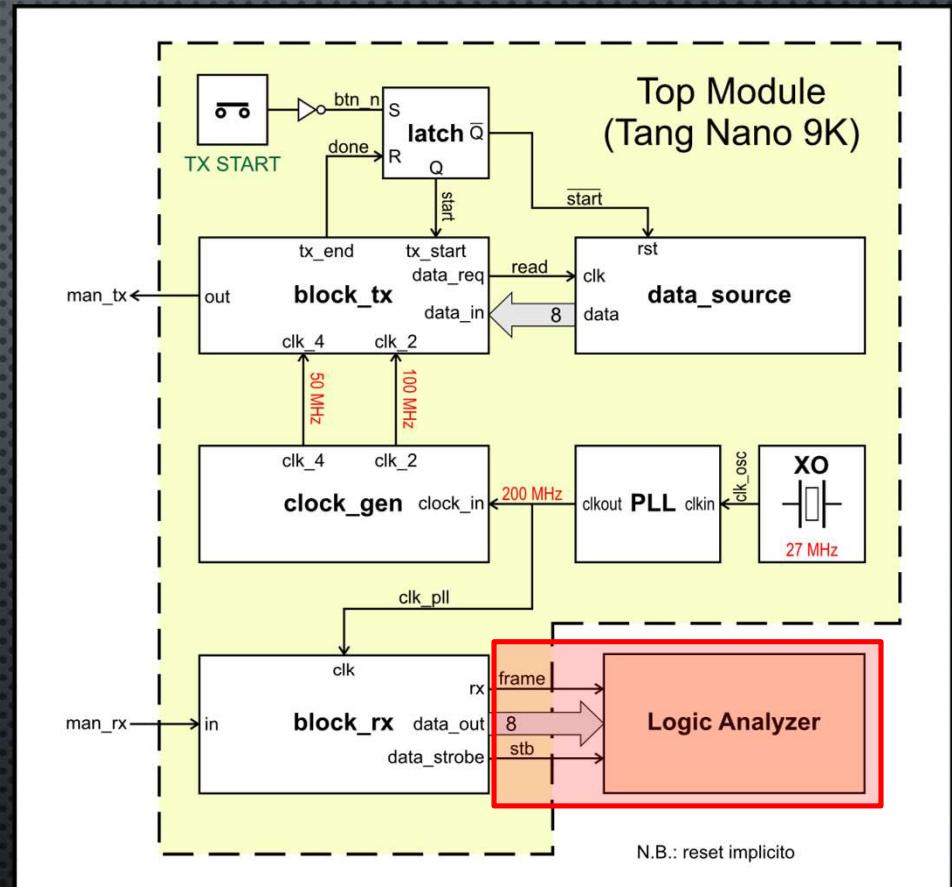
ESEMPIO: UN TRANSCEIVER MANCHESTER (3)

Il nostro esperimento, la cui struttura del Top Module è mostrata a lato, è interamente realizzato su una Tang Nano 9K di SiPeed.

Possiamo osservare ingresso ed uscita dei segnali manchester (rispettivamente man_rx e man_tx).

L'uscita del ricevitore è presentata come gruppo di bit paralleli (data_out), assieme a due segnali di sincronismo (frame e stb).

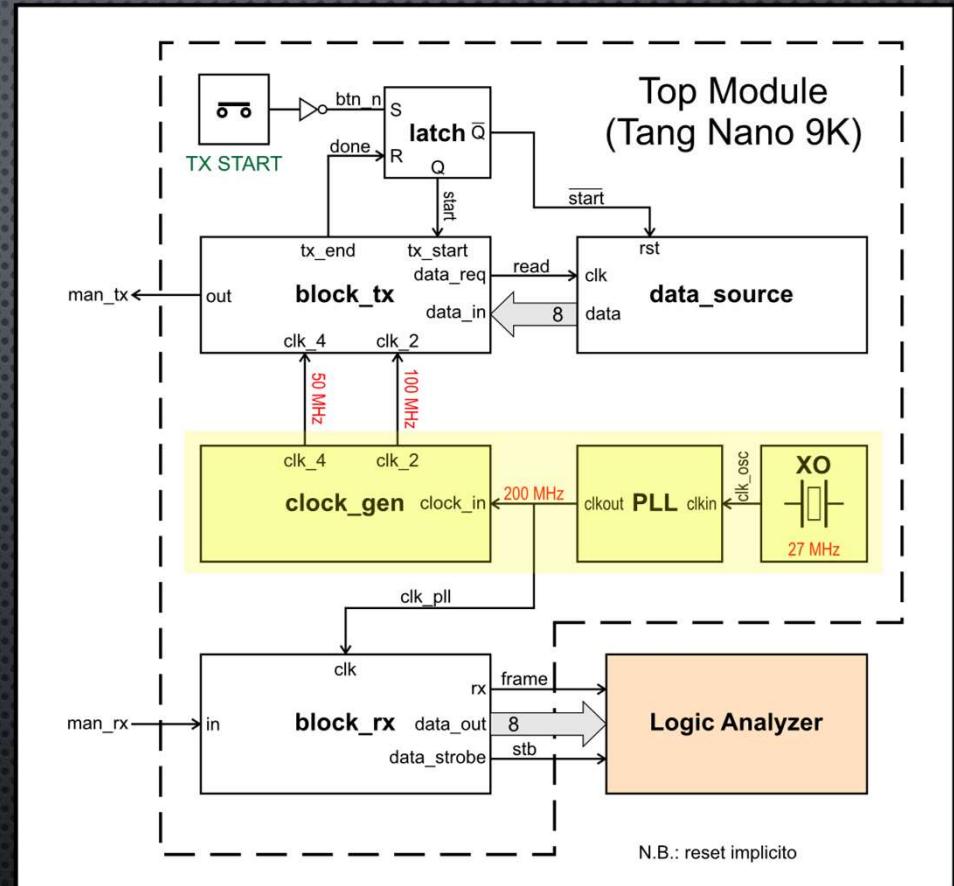
Per riuscire ad apprezzare il funzionamento del ricevitore, i segnali in uscita saranno registrati da un Analizzatore Logico che ci permetterà così di osservare il risultato.



ESEMPIO: UN TRANSCEIVER MANCHESTER (4)

All'interno del *top module* abbiamo tre sezioni distinte: generazione del clock, sezione di trasmissione e sezione di ricezione.

La prima è relativa al clock da fornire alle restanti due sezioni, ed è ricavata dall'XO a 27 MHz presente sulla Tang Nano 9K. L'uscita dell'XO entra nella FPGA e attraversa un PLL: è un Hard Block (o Hard IP) che moltiplica la frequenza dell'XO per un fattore frazionario e lo porta a circa 200 MHz. Il clock a 200 MHz entra in un secondo Hard Block che lo divide per due e poi per quattro, così da ottenere due ulteriori cadenze, rispettivamente da 100 MHz e da 50 MHz e che serviranno alle sezioni di ricezione e trasmissione che fra poco vedremo.



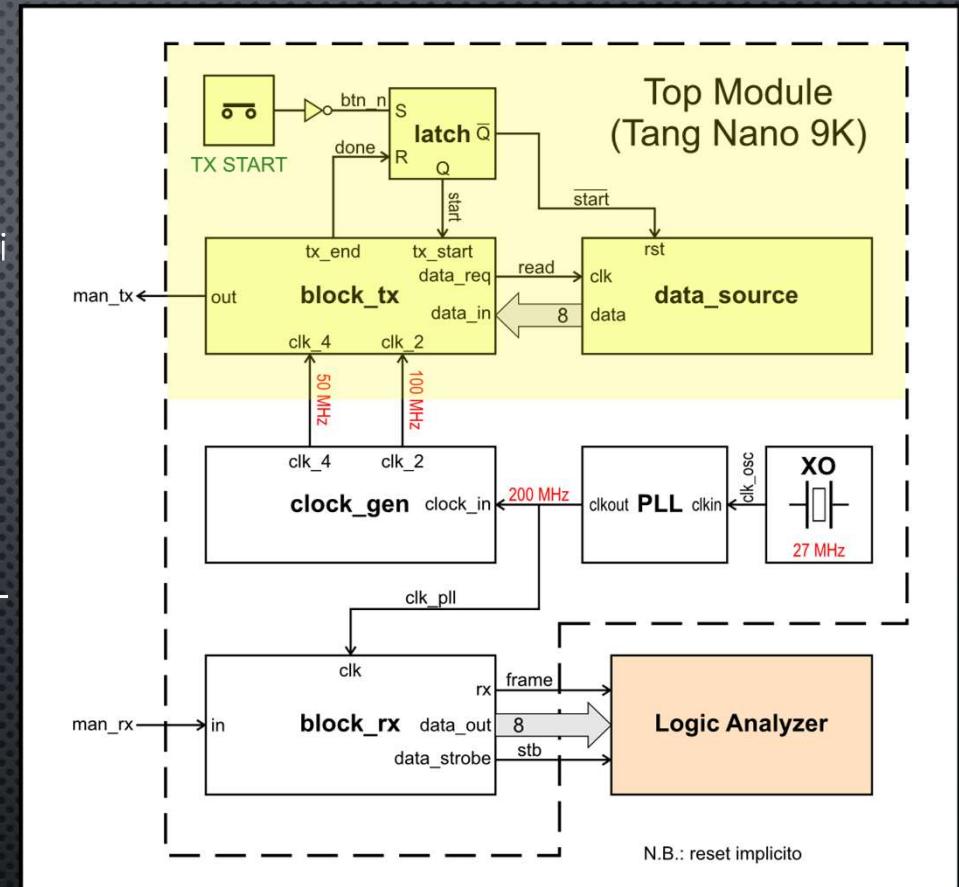
ESEMPIO: UN TRANSCEIVER MANCHESTER (5)

La **sezione di trasmissione** è composta da quattro parti. Il blocco di trasmissione (block_tx) contiene il modulatore Manchester così come visto precedentemente. Genera il segnale d'uscita (man_tx) ed usa due clock (clk_4 e clk_2).

Una sorgente (ottenuta da uno **LFSR**) produce i dati paralleli (data) che block_tx trasmetterà serialmente. Lo LFSR produrrà un nuovo codice ogni qualvolta block_tx lo richiederà con un fronte positivo attraverso la net *read* che collega i due moduli.

Uno dei due pulsanti presenti sulla Tang Nano 9K è collegato ad un latch attraverso il quale si innesca una richiesta di trasmissione (tx_start) di un pacchetto di 255 byte (generato dallo LFSR). Il pacchetto dati (payload) è preceduto da un preambolo di due byte (0xAA55) che servirà a sincronizzare il ricevitore. In tutto, la trama sarà lunga 257 byte.

Quando block_tx ha terminato di inviare i dati, genera un impulso (tx_end) che azzerà il latch e resetta data_source attraverso la net *!start*.

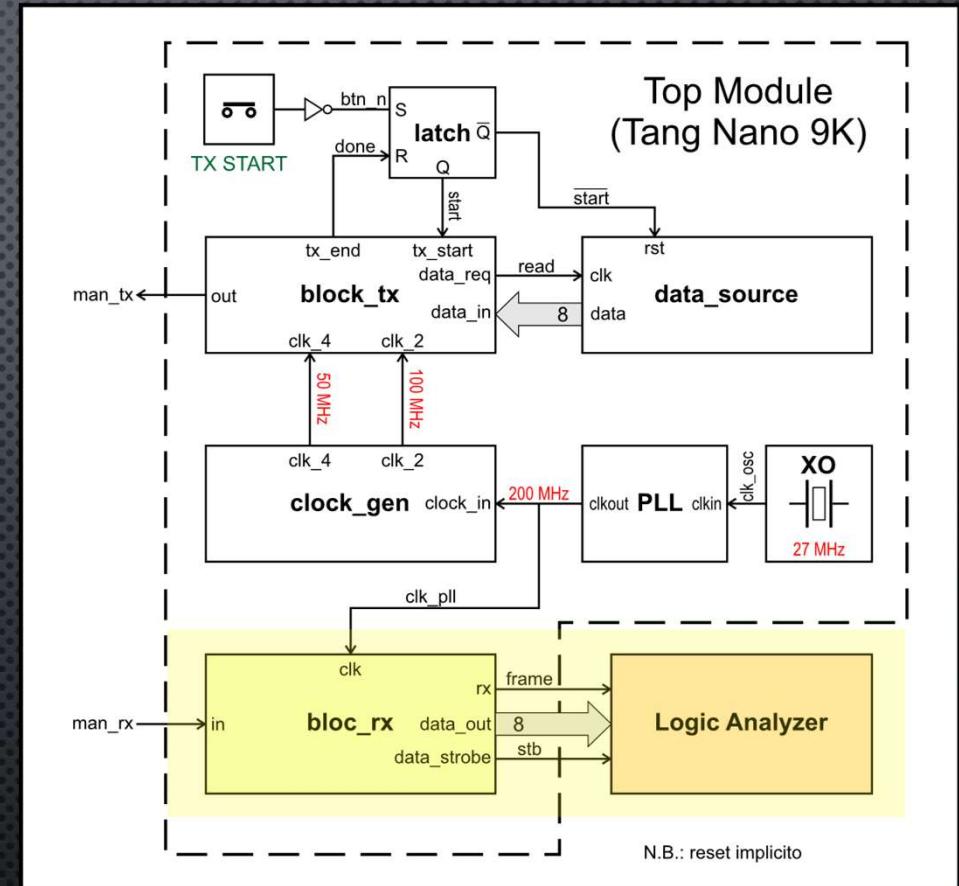


ESEMPIO: UN TRANSCEIVER MANCHESTER (6)

Il segnale in ingresso (man_rx) raggiunge block_rx: è il modulo preposto alla decodifica del protocollo Manchester. Questi riceve il clock più veloce (clk_pll) e produce due segnali scalari (frame e stb) ed un vettore ad otto bit (data_out).

Il compito di block_tx è di sincronizzarsi col trasmettitore e di estrarre i dati effettivi (il payload) e di presentarli consecutivamente sotto forma di byte in uscita.

All'uscita di block_rx è connesso un Analizzatore di Stati Logici che permette di vedere quanto ricevuto.



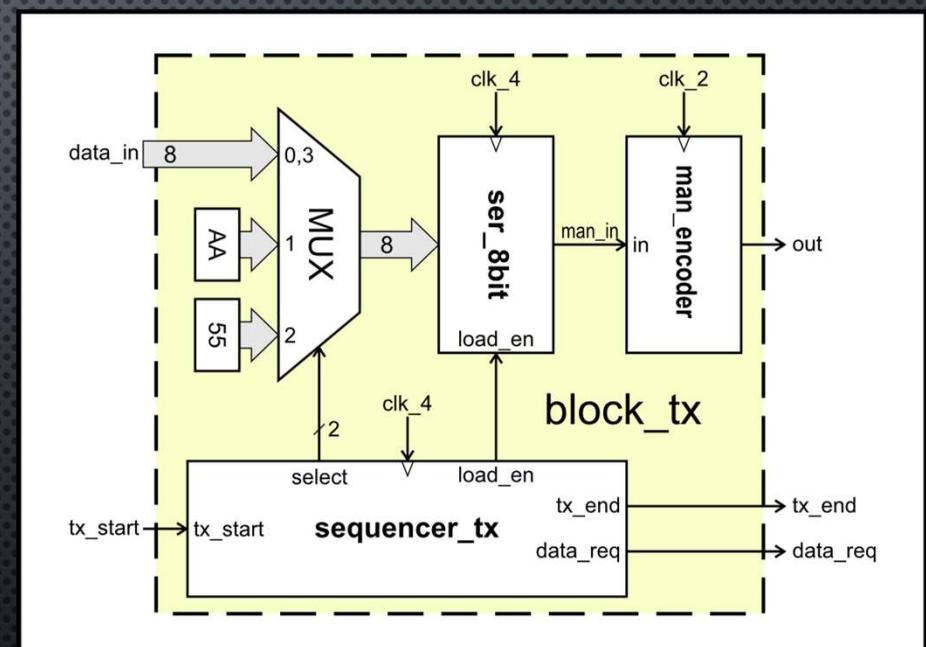
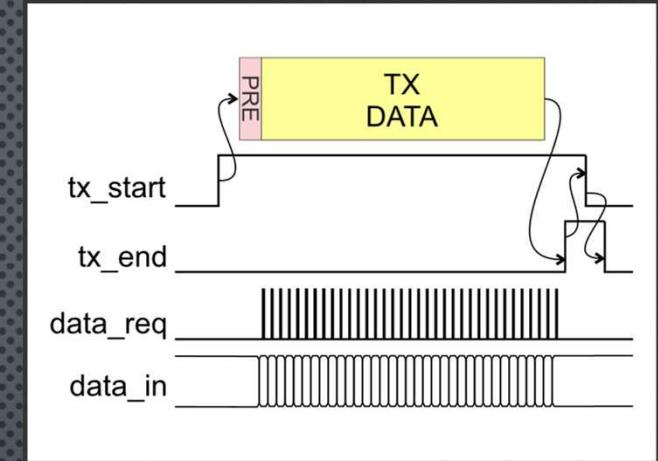
MODULO BLOCK_TX

Il modulo block_tx sfrutta tre sottomoduli: ser_8bit, man_encode e sequencer_tx. Il modulo ser8_bit è un serializzatore (PISO) come già visto in precedenza. Il modulo man_encode, già spiegato in addietro, genera il segnale Manchester.

Il modulo block_tx riceve i dati (il payload) in formato parallelo da un vettore di otto bit (data_in). I dati entrano in un multiplexer (MUX) codificato direttamente nel modulo. Suddetto multiplexer seleziona, uno alla volta, anche i due byte utilizzati per la sequenza del preambolo di sincronizzazione.

Il modulo sequencer_tx è preposto alla generazione dei segnali che orchestrano il funzionamento dell'intero modulo block_tx. Quando il modulo sequencer_tx riceve un livello alto sull'ingresso tx_start, inizialmente istruisce il multiplexer per selezionare il primo byte del preambolo di sincronizzazione (0xAA). Contemporaneamente abilita il serializzatore (ser_8bit) che invia i bit uno ad uno al codificatore manchester (man_encoder) il quale genera il segnale in uscita. Successivamente, sequencer_tx seleziona il secondo byte di sincronizzazione (0x55) e lo invia. Dopo, inizia l'invio di tutto il payload che è richiesto, byte a byte, generando fronti utili tramite il segnale data_req.

Infine, sequencer_tx provvederà con tx_end a segnalare la fine della trasmissione del payload. A questo punto sequencer_tx attenderà che l'ingresso start_tx torni a zero. Quando start_tx è deasserrito, anche tx_end tornerà a zero.



MODULO BLOCK_RX

Il modulo block_rx contiene 5 sottomoduli. Il segnale d'ingresso (in) incontra un primo blocco di sincronizzazione (sync_meta) che produce un segnale sincronizzato (in_d) col clock (clk).

Il segnale in_d arriva al modulo man_decoder il quale è responsabile dell'estrazione dei bit di informazione (man_out) e del relativo clock ad essi correlato (man_clk).

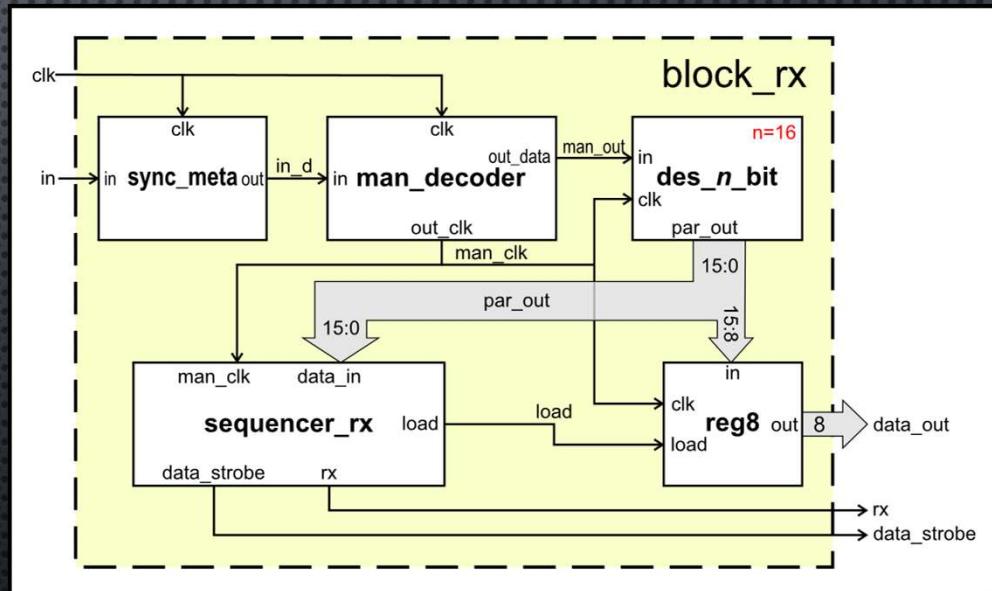
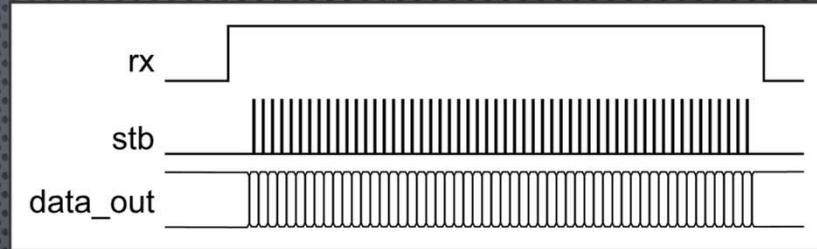
Il segnale man_out è fornito al modulo des_n_bit il quale costituisce un deserializzatore (SIPO) a 16 bit. I bit deserializzati sono propagati nel modulo attraverso un bus a 16 bit (par_out) che raggiunge sequencer_rx, cioè la macchina stati che controlla il funzionamento dell'intero modulo.

Quando sequencer_tx legge il pattern 0xAA55 (il premabolo), riconosce i bit successivi come payload. Il segnale rx è asserito ad indicare che sta sopraggiungendo il blocco di bit col payload.

Ogni 8 bit di payload, sequencer_tx produce un impulso che istruisce reg8 (un registro ad 8 bit) a campionare gli otto bit più significativi di par_out (par_out[15:8]).

Successivamente, sequencer_tx genera l'impulso data_strobe che avvisa il destinatario dei dati, nel nostro caso il Logic Analyzer, che un byte del payload è pronto per essere letto.

Al termine del payload, sequencer_rx deasserisce il segnale rx, indicando che l'intero frame è terminato.



Q&A

- Relatori:
Emanuel Conti (e.conti@embedded.sm)
Giuliano Cardinali (giulicard@gmail.com)

Source code download:

<https://tinyurl.com/HDLParte2Code>



RIFERIMENTI BIBLIOGRAFICI

- Digital Design and Computer Architecture 2nd Edition
David Harris, Sarah Harris - ISBN978-0123944245
- Reti logiche e calcolatori
F. Luccio, L. Pagli – ISBN: 978-8833954870
- FPGAs: Instant Access
Clive Maxfield - ISBN: 978-0750689748
- Bebop to the Boolean Boogie: An Unconventional Guide to Electronics
Clive Maxfield - ISBN: 978-1856175074
- Logic Circuit Design: Selected Topics and Methods, Second Ed.
Shimon P. Vingron - ISBN: 978-3031406720
- The Verilog® Hardware Description Language, 5th Ed.
Donald E. Thomas, Philip R. Moorby - ISBN: 978-1402070896
- Digital design : with an introduction to the verilog hdl
M. Morris Mano, Michael D. Ciletti.—5th ed. - ISBN-13: 978-0132774208
- Digital Design Principles and Practices
John F. Wakerly - ISBN: 978-0134460093