# Introduzione alle FPGA

Laboratorio Acquisizione e Controllo Dati
a.a. 2025 - 2026

## Contents

# 1 Introduzione alle FPGA

FPGA sta per Field-Programmable Gate Array. Le FPGA sono un tipo di **circuito integrato** che può essere configurato per svolgere funzioni specifiche dopo la produzione, ovvero "sul campo", anziché avere una funzione predefinita come i circuiti integrati tradizionali (e.g. CPU, Microcontroller, ASIC).

Le FPGA possono essere configurate per implementare vari tipi di **circuiti digitali**. Le FPGA infatti contengono una rete di **blocchi logici e interconnessioni** che possono essere programmati per eseguire operazioni logiche.

Le FPGA sono programmate utilizzando linguaggi di tipo **Hardware Description Language (HDL)**, come VHDL o Verilog, che descrivono il comportamento hardware desiderato.

Le caratteristiche principali delle FPGA sono

- **Parallelismo:** A differenza delle CPU, che eseguono le istruzioni in modo sequenziale, le FPGA eseguono molte operazioni in parallelo.
- **Riconfigurabilità:** Una FPGA può essere riprogrammata, permettendo di modificarne la funzionalità senza dover sostituire l'hardware.
- **Alta velocità di prestazione:** Poiché le FPGA operano a livello hardware, possono raggiungere velocità di elaborazione molto superiori rispetto ai software che girano su un processore generico.

In generale, le FPGA trovano applicazione in qualsiasi problema di elettronica digitale in cui siano necessarie prestazioni elevate, bassa latenza e flessibilità in tempo reale. Le FPGA sono utilizzate in vari settori quali telecomunicazioni (e.g. gestione del traffico internet in torri cellulari), finanza (e.g. algoritmi per trading ad alta frequenza), difesa (e.g. elaborazione del segnale digitale radar, e di immagini da telecamere a infrarossi), AI (acceleratori hardware, implementazioni di algoritmi di machine learning e reti neurali), automazione industriale, elaborazione video, sviluppo di prototipi hardware.

## 1.1 Esempio di uso di FPGA in fisica delle particelle

Negli esperimenti di fisica delle particelle, come ad esempio ATLAS o CMS al Large Hadron Collider (LHC) al CERN, la selezione in tempo reale degli

eventi è fatta dal sistema di trigger. Il segnale di trigger è sviluppato per selezionare le interazioni rilevanti per gli studi dei diversi canali di fisica. La quantità di dati prodotti nelle collsioni all'LHC è tale per cui non è possibile registrare e analizzare ogni singolo evento. Il sistema di trigger permette di ridurre il numero di eventi da registrare, mantenendo solo gli eventi interessanti e eliminando i segnali di fondo.

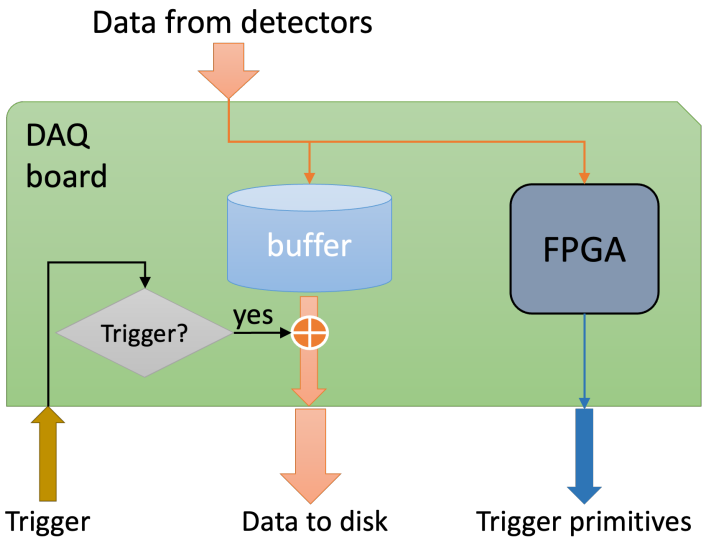



Figure 1: Schema semplificato della logica di trigger per un esperimento di fisica delle particelle.

Il sistema di trigger acquisisce dati da diverse parti del rivelatore per cercare le cosiddette firme di eventi fisici interessanti. Ad esempio, il rilascio di energia del calorimetro permette di identificare particelle specifiche quali elettroni, fotoni, tau. I dati per generare il segnale di trigger sono elaborati in tempo reale tramite algoritmi complessi implementati su FPGA, e il segnale di trigger viene generato in alcuni micro secondi. I dati acquisiti dai vari rivelatori vengono salvati in memorie locali (buffer) in attesa del trigger. I dati selezionati dal sistema di trigger vengono estratti dai rivelatori e salvati su server per essere analizzati.

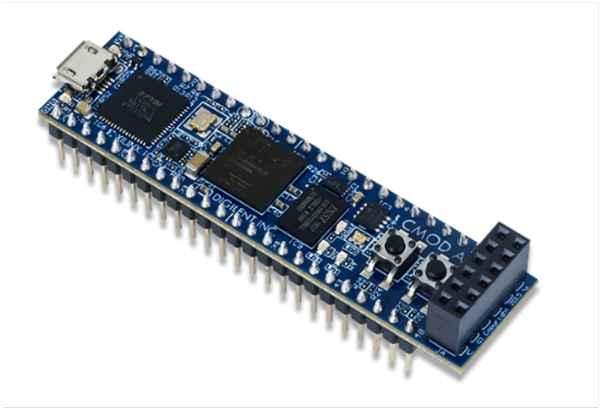
## 1.2 Evoluzione delle FPGA

La prima FPGA è stata creata da Xilink nel 1985 (XC2064) e conteneva meno di mille gates. Le prime FPGA eseguivano funzioni logiche molto semplici, ma un singolo dispositivo sostituiva l'utilizzo di molti componenti discreti, riducendo i costi, risparmiando spazio sulle scheda di circuito, e consentendo la riprogrammazione del disegno man mano che i requisiti del progetto cambiavano.

Le capacità degli FPGA sono aumentate drasticamente nel corso degli anni. Le FPGA odierne contengono millioni di gates e implementano circuiti e algoritmi complessi. Oggi le FPGA contengono anche componenti specializzati, dedicati all'esecuzione di un compito specifico, detti **hard IP (Intellectual Property) blocks**. I blocchi di circuito IP nelle FPGA moderne consentono al dispositivo di interfacciarsi direttamente con dispositivi USB, memorie e altri componenti esterni. Alcune di queste capacità (come un'interfaccia USB-C) semplicemente non sarebbero possibili senza un hard IP dedicato per svolgere il lavoro. In alcuni modelli di FPGA sono inseriti processori dedicati (chiamati processori hard) in modo da poter eseguire il normale codice C all'interno dell'FPGA stesso.

I principali produttori di FPGA sono **ADM (ex-Xilink), Intel (ex-Altera), Microchip e Lattice Semiconductor**. Tipicamente, i produttori forniscono anche il software necessario per la progettazione delle FPGA, quali, ad esempio, **Vivado (ADM), Quartus (Intel), Diamond (Lattice), Libero (Microchip)**.

Tipicamente le FPGA sono integrate su schede elettroniche che contengono connettori per interfacciarsi con altri dispositivi e per la configurazione e l'alimentazione delle FPGA, risorse per la generazione di segnali di clock, e altri componenti, ad esempio, interrutori, LED, display. La figura 2 mostra la scheda Digilent Cmod A7 basata su una FPGA Xilinx Artix-7 che useremo in questo laboratorio. La scheda include anche un circuito di programmazione USB-JTAG, un bridge USB-UART, una sorgente di clock, un connettore host Pmod, memoria SRAM, memoria Flash Quad SPI e dispositivi di I/O di base.



**Cmod A7**

Breadboardable Artix-7 FPGA Module

| Features | |
|---|---|
| • Programmable over JTAG and Quad-SPI Flash<br>• 1 MSPS on-chip ADC | |
| **Key Specifications** | |
| **LUTs** | 10,400 (15T)<br>20,800 (35T) |
| **Flip-flops** | 20,800 (15T)<br>41,800 (35T) |
| **Block RAM** | 112.5KB (15T)<br>225KB (35T) |
| **SRAM** | 512KB with an 8-bit bus and 8ns access time |
| **Quad-SPI Flash** | 4MB |
| **Connectivity and Onboard I/O** | |
| **Pmod Connectors** | 1 with 8 Digital I/O |
| **Connectors** | 48-pin DIP with 44 Digital I/O<br>and 2 Analog inputs (0-3.3V) |
| **Buttons** | 2 |
| **LEDs** | 2 |
| **RGB LED** | 1 |
| **Electrical** | |
| **Power** | USB or external 3.3-5.5V supply |
| **Physical** | |
| **Width** | 0.7 in |
| **Length** | 2.75 in |

Figure 2: Esempio di scheda elettronica con FPGA.

# 2 Progettazione delle FPGA

## 2.1 Flusso di progetto

La progettazione delle FPGA si può suddividere in quattro fasi (figura 3): Design, Synthesis, Place & Route (Implementation), Programming[1].

- **Design**. Scrivere codice HDL che descrive le funzionalità da implementare su FPGA.
- **Synthesis**. Tradurre il codice HDL in componenti di basso livello (i.e. in liste fisiche di porte logiche e connessioni).
- **Place & Route (Implementation)**. Mappare il progetto sintetizzato sul layout fisico dell'FPGA (place) e cablare la connessione tra i componenti (route).
- **Programming**. L'output del passaggio P&R (i.e. il bitfile) viene caricato sull'FPGA fisico.

La descrizione HDL, nella fase di disegno o progettazione funzionale, costituisce una descrizione cosiddetta di alto livello del circuito che si vuole implementare. Il modo in cui si descrive il funzionamento di un circuito digitale tramite linguaggio HDL è detto comportamentale o **Register Transfer Level (RTL)**, a seconda che contenga o meno costrutti non sintentizzabili. Questi concetti sono discussi nel capitolo 2.3

Il tool di sintesi converte la descrizione RTL nella descrizione detta a **gate level**, che serve per realizzare il bitstream. La descrizione a gate level contiene una netlist, cioè l'elenco completo delle connessioni, porte logiche, blocchi IP necessari a implementare correttamente il comportamento funzionale e temporale del circuito.

Il risultato della sintesi è quindi usato dagli applicativi di P&R per creare la disposizione fisica del circuito integrato sulla FPGA che si intende usare.

Come detto in precedenza, la maggior parte della logica in una FPGA è logica sequenziale sincrona. Il disegno deve quindi funzionare correttamente alla frequenza di clock specificata. Uno degli aspetti più importanti nel ciclo di progettazione di una FPGA è quindi specificare e convalidare correttamente i **vincoli temporali (timing constraints)** del progetto per

[1] In italiano: progettazione, sintesi, implementazione, configurazione.

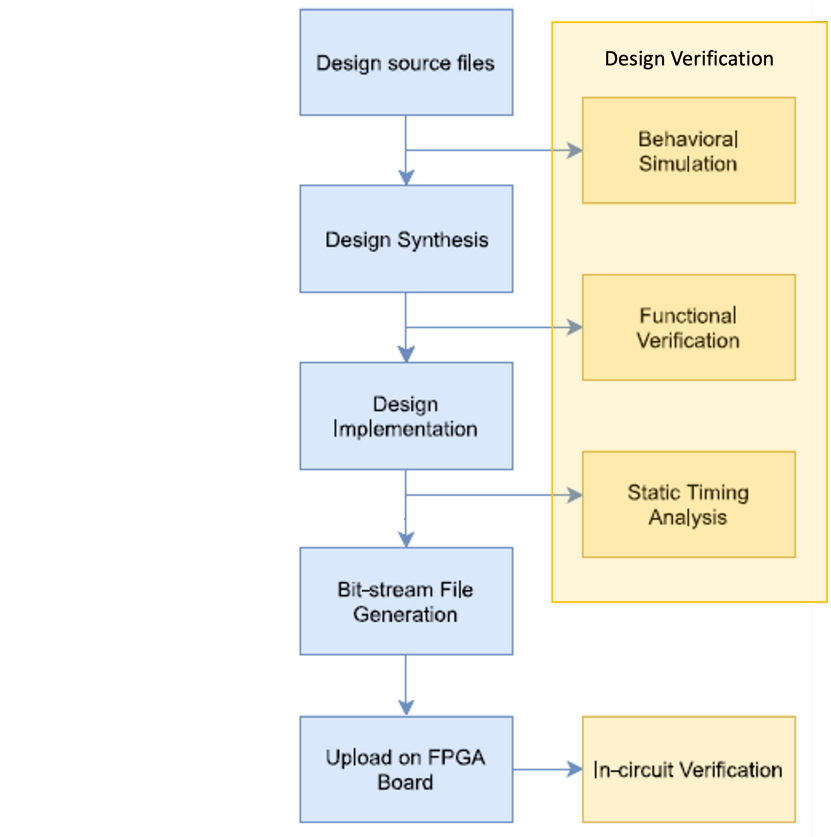



Figure 3: Flusso di progetto di una FPGA, incluse la fasi di verifica del disegno.

garantire che vengano soddisfatti i requisiti di temporizzazione e quindi che vengano raggiunte le prestazioni desiderate. I vincoli temporali permettono di definire la frequenza di clock, verificare i tempi di setup e hold (appendice B), individuare percorsi critici, guidare l'ottimizzazione di placement e routing. I vincoli temporali garantiscono che il circuito funzioni alla frequenza desiderata.

Oltre ai vincoli temporali vanno specificati anche i **vincoli di posizionamento (physical constraints)**. I segnali logici del progetto devono essere mappati sui pin fisici dell'FPGA. Ciò garantisce il corretto routing del segnale all'interno della FPGA e una corretta interfaccia con il mondo esterno. I vincoli di posizionamento permettono inoltre di impostare lo standard elettrico corretto per gli I/O, controllare il posizionamento interno di specifici blocchi in determinate aree (raro nei progetti base, comune in progetti complessi), gestire risorse specifiche (e.g. PLL, transceiver, BRAM). I vincoli di posizionamento rendono il progetto compatibile con la FPGA e con la scheda su cui è montata.

In Xilinx/AMD Vivado vincoli di temporizzazione e posizionamento convivono nello stesso file (con estensione .xdc).

## 2.2 Simulazioni HDL e Static Timing Analysis

La simulazione HDL è un processo di verifica della funzionalità dei circuiti digitali descritti in HDL. I principali strumenti di simulazione digitale disponibili sul mercato sono: **QuestSim, ModelSim, Riviera-Pro, XCelium, XSim, GHDL**. Per eseguire una simulazione HDL si scrive un codice di testbench. Il testbench è una descrizione HDL che applica segnali in ingresso al circuito da simulare e ne analizza le uscite.

Il processo di simulazione è di fondamentale importanza nel disegno digitale per verificare la corretta funzionalità del disegno hardware prima dell' implementazione. Una simulazione ben fatta consente di testare tutti i possibili scenari di input per garantire che il disegno risponda correttamente in tutti i casi. Qualora la risposta alla simulazione evidenziasse un comportamento diverso da quello atteso, questo aiuterebbe il digital designer a individuare il problema (che può essere ad esempio di logica o timing) e risolverlo prima dell'implementazione del bitfile e della configurazione della FPGA. Eseguire il debugging in simulazione è molto più semplice e veloce che su hardware.

Ci sono due tipi di simulazione HDL. La **simulazione comportamen-**

**tale e temporale** testa la funzionalità del progetto, comprese le informazioni di temporizzazione, per convalidare che il disegno funzioni correttamente, alla frequenza di clock specificata. La **simulazione post-sintesi (functional verification)** convalida la progettazione dopo la sintesi per garantire che corrisponda alla logica prevista.

Successivamente al P&R si esegue una **Static Timing Analysis**. Questo tipo di analisi viene fatta sul disegno fisico, ovvero sul disegno mappato sulle risorse della FPGA scelta per il progetto. Per questo tipo di analisi si estraggono i tempi di propagazione dei segnali, e i loro ritardi, direttamente sul disegno fisico in modo da controllare eventuali violazioni di timing sul circuito stesso, anziché tramite una simulazione del disegno. Se il progetto è troppo lento per funzionare alla frequenza di clock desiderata, si verificheranno errori di temporizzazione e l'implementazione del progetto probabilmente non funzionerà correttamente. Concetti di temporarizzazione delle FPGA sono discussi nell'appendice B.

## 2.3 Hardware Description Language

I linguaggi HDL sono utilizzati per descrivere il comportamento dei circuiti digitali da implementare su FPGA (o ASIC), e per verificare i progetti digitali con simulazioni. I principali linguaggi disponibili sono VHDL e Verilog.

Il modello computazionale dei linguaggi di descrizione hardware è profondamente diverso dai linguaggi di programmazione tradizionali. Nei linguaggi di programmazione gli statement del linguaggio definiscono istruzioni, che vengono eseguite sequenzialmente da una infrastruttura (e.g. dalla CPU nel caso di C). Nei linguaggi HDL gli statement del linguaggio definiscono blocchi di hardware. Non c'è nessuna esecuzione sequenziale, nessun run-time, nessuna infrastruttura sottostante. L'esecuzione di codice HDL è parallela e immediata, e l'infrastruttura sottostante è definita dal codice e implementata dagli applicativi di sintesi e P&R.

L'esecuzione parallela invece che sequenziale è probabilmente la differenza più fondamentale tra i linguaggi di programmazione hardware e software. Ciò che si intende per codice seriale è che le linee di codice vengono eseguite una alla volta. I linguaggi di programmazione hardware sono noti come linguaggi di logica parallela e tutte le linee di codice possono essere eseguite e verranno eseguite contemporaneamente (concurrency).

Un altro punto da tenere a mente quando si usa un linguaggio di programmazione HDL è che solo un sottoinsieme del linguaggio può essere effettiva-

mente sintetizzato. I linguaggi HDL forniscono costrutti non sintetizzabili che però possono essere molto utili per la descrizione e la simulazione comportamentale del disegno. Quando si scrive un testbench per la simulazione, usare costrutti di codice non sintetizzabili, spesso semplifica e migliora il testbench. Il codice non sintetizzabile più usato è quello per generare un ritardo. Le FPGA non hanno alcun concetto di tempo. Se è necessario inserire un ritardo definito nella propagazione di un segnale, questo viene fatto in hardware con l'uso di flip-flops.

# 3 Architettura delle FPGA

## 3.1 Nozioni generali di circuiti digitali

Le FPGA sono circuiti digitali, ovvero elaborano segnali rappresentati in forma digitale. La figura 4 mostra un esempio di segnale digitale. Un segnale digitale può assumere solo valori discreti; in un dato momento può assumere solo uno di un numero finito di valori. Tipicamente i segnali digitali sono rappresentati usando il **sistema binario**, quindi i segnali digitali binari possono assumere solo due valori, 0 e 1 (o HIGH/LOW, true/false), che corrispondono a due livelli di tensione. Ogni valore rappresenta un bit. Un gruppo di bit che rappresenta una certa unità di informazione si chiama digital word. Un digital word fatto di otto bits è un byte.

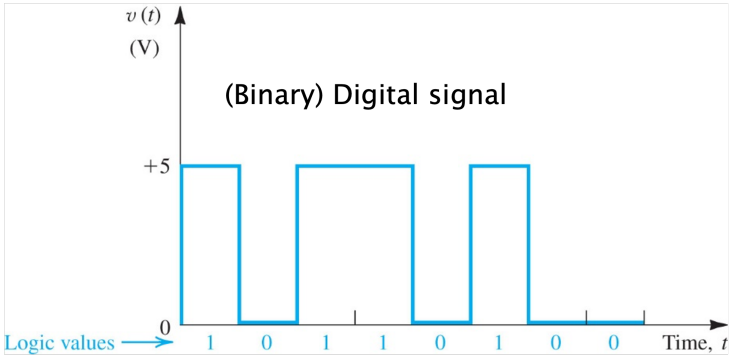



Figure 4: Esempio di segnale digitale.

I blocchi di base dell'elettronica digitale possono essere riassunti come mostrato in figura 5: porte logiche, registri, connessioni e interruttori.

I segnali digitali sono elaborati usando le regole dell'algebra booleana. L'algebra booleana descrive operazioni in cui gli input e gli output assumono i valori 1 o 0. Con le regole stabilite dall'algebra booleana, è possibile eseguire tutte le operazioni logiche di base. Le funzioni logiche sono rappresentate nei sistemi digitali con porte logiche (figura 6). Esse permettono di effettuare le operazioni logiche di base quali AND, OR, NOT e combinazioni di queste come NOR, NAND, XOR e XNOR. Le porte logiche si possono rappresentare in modo equivalente tramite il simbolo circuitale, la tabella di verità (truth table, TT), o l'espressione booleana. Una tabella di verità è una tabella che descrive un'operazione logica in base agli input e agli output. Una TT

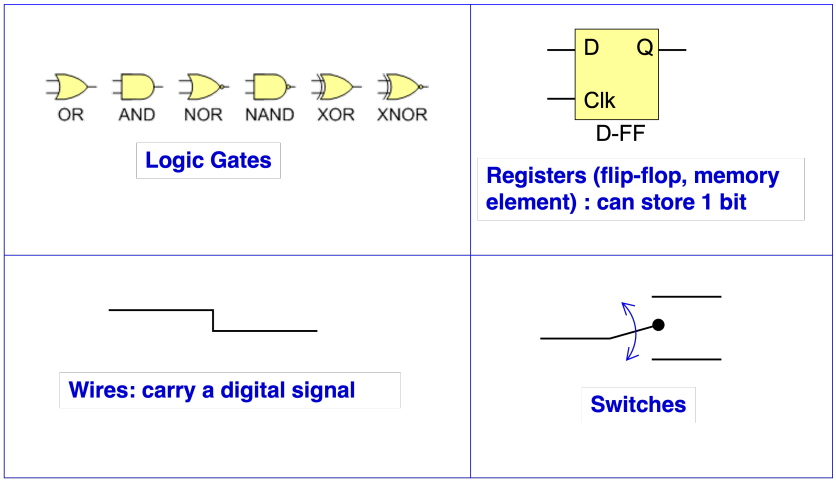



Figure 5: Blocchi di base della logica digitale.

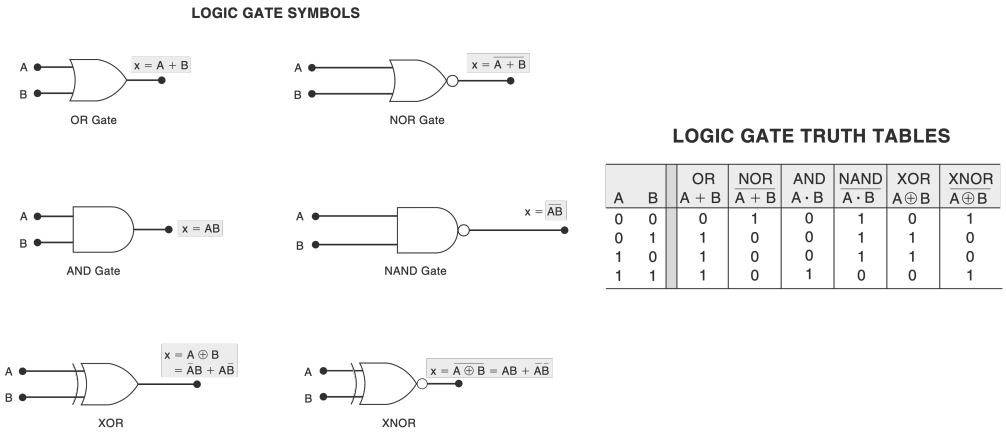
elenca tutte le possibili combinazioni di input e output per un'equazione algebrica booleana. La porta NOT che implementa l'operazione di inversione è mostrata in figura 7. Nei simboli circuitali, l'inversione è rappresentata da una bolla. Nelle espressioni booleane, l'inversione è rappresentata da una barra.

Combinando tra loro più porte logiche si realizzano circuiti logici più complessi, quali per esempio i flip-flop, circuiti in grado di memorizzare informazioni elementari, e in generale reti logiche variamente complesse.

I circuiti digitali possono implementare **logica combinatoria** o **logica sequenziale**. La logica combinatoria esegue tutte le funzioni logiche nel circuito e in genere è costituita da porte logiche. In ogni istante, nella logica combinatoria, l'uscita di un circuito è una funzione solo dei valori presenti all'ingresso.

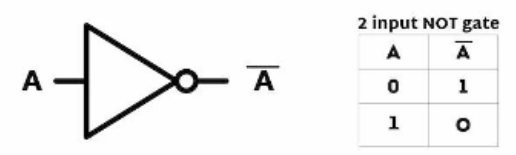
In un circuito logico sequenziale, l'output dipende non solo dal valore attuale dei suoi segnali di input, ma anche dagli stati precedenti. La logica sequenziale ha memoria. Come mostra la figura 8, un circuito sequenziale si ottiene aggiungendo memoria a un circuito combinatorio. Un flip-flop è un circuito sequenziale con proprietà di memoria.

In un circuito logico sequenziale le uscite seguono una sequenza predeterminata di stati con un nuovo stato che si verifica a ogni ciclo di clock. Questo tipo di logica è detto logica sincrona. Nei **sistemi sincroni**, tutti i segnali sono sincronizzati da una forma d'onda di temporizzazione di base,




| A | B | OR A + B | NOR $\overline{A+B}$ | AND A · B | NAND $\overline{A \cdot B}$ | XOR A⊕B | XNOR $\overline{A \oplus B}$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

Figure 6: Porte logiche.




| A | $\overline{A}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

Figure 7: NOT gate.

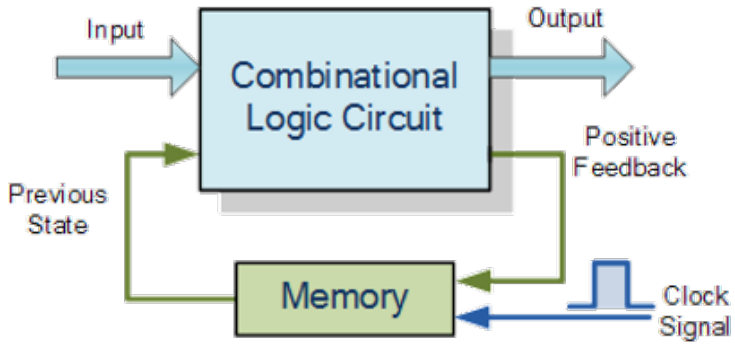



Figure 8: Logica sequenziale.

il **clock**. Tipicamente, il periodo del clock è il tempo occupato da un bit. Nell'esempio di figura 9, un nuovo bit è disponibile a ogni fronte di salita del clock. La transizione da un bit al successivo può avvenire anche sul fronte di discesa del clock, o su entrambi i fronti.

**La maggior parte della logica implementata nelle FPGA è realizzata con logica sequenziale sincrona.**

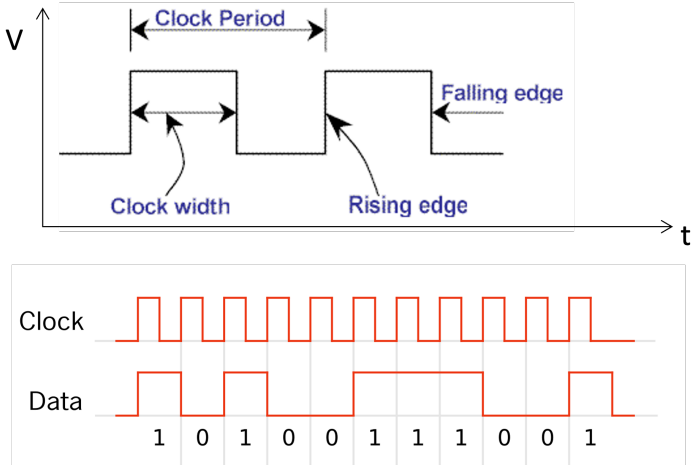



Figure 9: Segnale di clock.

## 3.2 Elementi costitutivi delle FPGA

L'architettura di una FPGA è mostrata in modo schematico in figura 10.

Gli elementi principali che costituiscono una FPGA sono:

- **Configurable Logic Block, CLB**: I blocchi logici configurabili sono le unità fondamentali in una FPGA. Sono configurabili per eseguire una varietà di funzioni logiche. Comprendono Look-Up Tables (LUT), flip-flop (FF) e multiplexer.
- **Interconnessioni:** Rete di cablaggio che collega tra loro i CLB. Le interconnessioni sono fatte da segmenti di filo uniti da interruttori elettricamente programmabili.
- **Blocchi input/output (I/O):** L'interfaccia tra una FPGA e altri dispositivi esterni è abilitata da blocchi I/O. Gli I/O sono risorse di

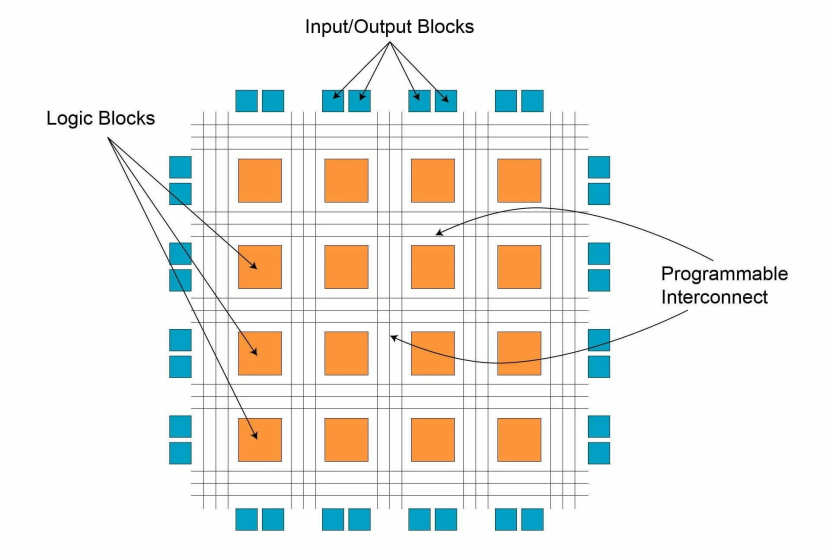



Figure 10: Architettura delle FPGA.

input e output programmabili, configurate per adattarsi ai protocolli di qualsiasi dispositivo esterno a cui si connette la FPGA. Tutti i segnali che entrano o escono dalla FPGA lo fanno tramite i pin del dispositivo e I/O associati.

- **Risorse di clock:** Risorse dedicate al clock per sincronizzare le operazioni su tutto il chip (e.g. Phase-Locked Loop (PLL), reti di distribuzione del clock).
- **Block RAM**: Blocchi di memoria dedicati, utilizzati per archiviare dati e istruzioni.
- **Digital Signal Processing blocks (DSP)**: Blocchi specializzati per l'esecuzione di calcoli matematici complessi.
- **Memoria di configurazione**: Memorizza i dati di configurazione che definiscono il comportamento dell'FPGA.

## 3.3 CLB

La figura 11 mostra lo schema di un CLB. I CLB sono costituti da tre elementi: LTU, FF, multiplexer. Le LUT implementano funzioni logiche. Il FF è usato come elemento di memoria. I multiplexer sono utilizzati per il routing all'interno del CLB.

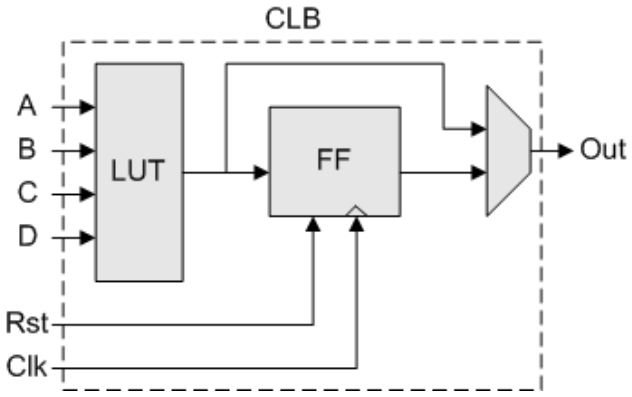


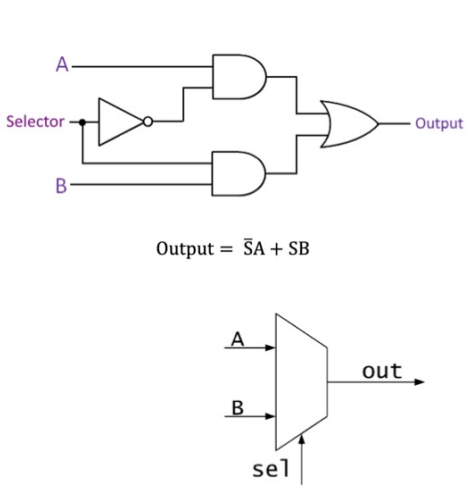

Figure 11: Configurable Logic Block.

### 3.3.1 Multiplexer

Il **multiplexer** è un circuito combinatorio capace di selezionare uno tra vari ingressi possibili e di trasferire il dato in esso presente in uscita. E' sempre dotato di uno o più ingressi di selezione: $m$ ingressi di selezione servono per pilotare $n = 2^m$ ingressi dati. La figura 12 mostra un esempio di MUX con due ingressi.

### 3.3.2 Look-Up Tables

Le porte logiche discrete (AND, OR, NOT, ...) non esistono all'interno di una FPGA. Le FPGA utilizzano le **Look-Up Table (LUT)** per eseguire operazioni di algebra booleana. La LUT è programmata dal digital designer per eseguire un'equazione di algebra booleana. Le LUT utilizzano lo stesso concetto di tabella di verità dei gates per correlare gli output agli input. Una LUT può essere vista come una tabella di verità programmabile che implementa qualunque funzione logica combinatoria su un numero limitato di ingressi. Invece di costruire la logica con porte AND/OR/NOT, la FPGA




| A | B | S | Output |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

Figure 12: Multiplexer: circuito, funzione logica, tabella di verità.

memorizza tutti i possibili output in una tabella. Gli ingressi selezionano quale output leggere dalla tabella.

La struttura di base di una LUT ha: ingressi, di solito 4 o 6 bit (LUT4, LUT6); tabella di valori con $2^N$ bit per bit per N ingressi (es. LUT6 $\rightarrow$ 64 bit); multiplexer interno che seleziona l'output corrispondente all'ingresso. Fisicamente una LUT è implementata con una memoria SRAM interna (per LUT6, SRAM a 64 bit) e multiplexer a cascata per selezionare l'uscita corretta. L'esempio in figura 13 mostra una LUT a 4 bit usata per costruire una funzione booleana con quattro variabili di ingresso (A, B, C e D), in cui l'uscita è 1 quando due qualsiasi delle variabili di ingresso sono uguali a 1. La funzione logica è mostrata nella tabella di verità. A, B, C e D sono gli ingressi della LUT. I valori della variabile di uscita per ciascuna loro combinazione sono memorizzati nelle celle SRAM. La linea rossa tratteggiata in figura 13 mostra come viene selezionato l'output corrispondente a ABCD = 0101.

### 3.3.3 Flip-flop

I **flip-flop** sono circuiti logici sequenziali sincroni, e fungono da celle di memoria elementari. I flip-flop sono il componente principale di una FPGA. Tutta la logica sequenziale nelle FPGA è costituita da FF.

**Truth Table**

| Inputs | | | | Output |
|---|---|---|---|---|
| A | B | C | D | Y |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

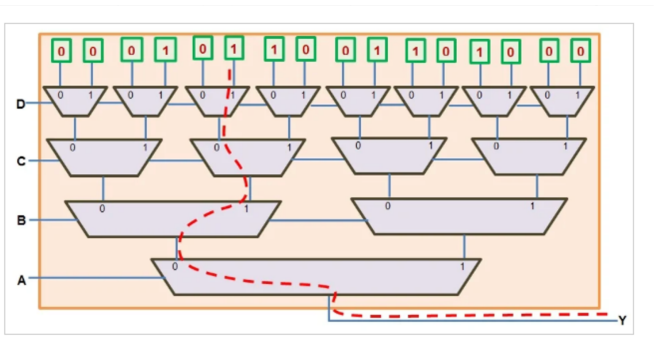



Figure 13: LUT a 4 inputs.

I flip-flop hanno due uscite che sono l'una l'inversa dell'altra, denominate $Q$ (uscita normale) e $\overline{Q}$ (uscita invertita). Quando $Q = 1$ e $\overline{Q} = 0$ si dice che l'uscita del FF è HIGH, o che il FF è SET. Viceversa, quando $Q = 0$ e $\overline{Q} = 1$ si dice che l'uscita del FF è LOW, o che il FF è RESET. I flip-flop possono avere uno o più ingressi. Quando gli ingressi fanno andare il flip-flop allo stato HIGH, il FF è settato (SET). Quando gli ingressi fanno andare il flip-flop allo stato LOW, il FF è resettato (RESET). Per fare un SET o RESET del flip-flop, è necessario un impulso sugli ingressi in coincidenza con il segnale di clock, applicato ad un ingresso dedicato. I FF possono funzionare sul fronte di salita o discesa del clock. Dopo che un impulso mette i flip-flop in un certo stato di uscita (SET o RESET), il FF rimane in quello stato anche dopo che l'impulso è terminato (caratteristica di memoria dei flip-flop). Una schema generico di FF è mostrato in figura 14.

Esistono vari tipi di FF: SET/RESET; JK; toogle (T); D. Nel contesto delle FPGA è bene studiare il FF D (figura 15). Questo FF ha un ingresso per i dati, un ingresso di clock, e le uscite $Q$ e $\overline{Q}$. L'output segue l'input all'arrivo del fronte del clock. Il FF D viene utilizzato per memorizzare dati in determinati momenti, i.e. $Q = D$ solo quando il clock attiva il flip-flop.

L'applicazione principale dei flip-flop è quella di memorizzare dati. I dati sono normalmente memorizzati in gruppi di flip-flop chiamati registri. Nelle FPGA, i FF sono usati per formare **registri a scorrimento (shift regis-**

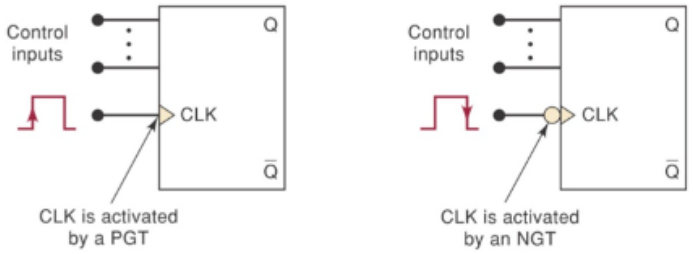



Figure 14: Simbolo generico di FF con un certo numero di inputs, gli outputs $Q$ e $\overline{Q}$, e l'ingresso per il clock. Il FF può essere attivato dal fronte di salita (positive going transistion, PGT) o di discesa (negative going transistion, NGT) del clock.

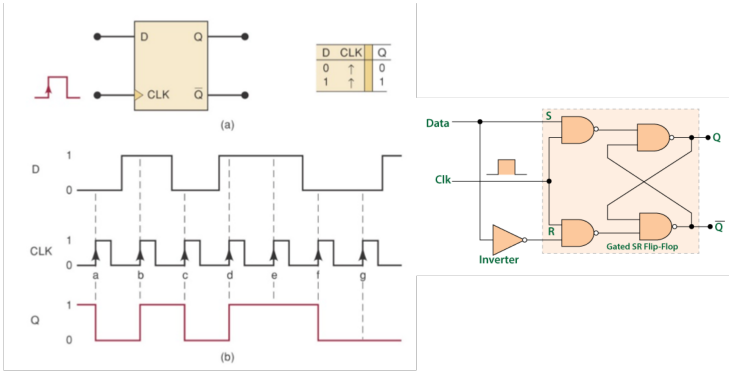



Figure 15: D flip-flop.

**ters)**. Un registro a scorrimento è un blocco fondamentale nell'elettronica digitale, utilizzato per memorizzare e spostare bit, ad esempio per convertire dati da seriale a parallelo e viceversa. Un registro a scorrimento è formato da una cascata di flip-flop, che condividono lo stesso clock. L'uscita di ogni flip-flop è collegata all'ingresso dati del flip-flop successivo nella catena.

La figura 16 mostra uno shift register formato da quattro FF. Ad ogni ciclo di clock, lo shift register carica nel primo FF un bit nuovo del segnale in ingresso, fa scorre i bits contenuti nei suoi FF di un posto verso destra, e mette in uscita il bit contenuto nell'ultimo FF. Questo tipo di shift register è un registro a scorrimento serial-in serial-out (SISO): i dati entrano serialmente in un flip-flop alla volta; l'uscita è anch'essa seriale; ad ogni impulso di clock, il bit memorizzato si sposta di una posizione verso l'uscita. Uno shift register SISO è utile per memorizzare temporaneamente dati o ritardare segnali. Ogni flip-flop può memorizzare un bit. Un registro formato da quattro FF come nell'esempio di figura 16, può memorizzare digital words lunghi quattro bits. Quando un registro SISO è usato per ritardare i dati in ingresso, la quantità di ritardo temporale è controllata dal numero di FF nel registro o variando l'applicazione degli impulsi di clock. Nel caso dell'esempio di figura 16, il primo bit in ingresso al registro, appare in uscita dallo shift register dopo quattro cicli di clock.

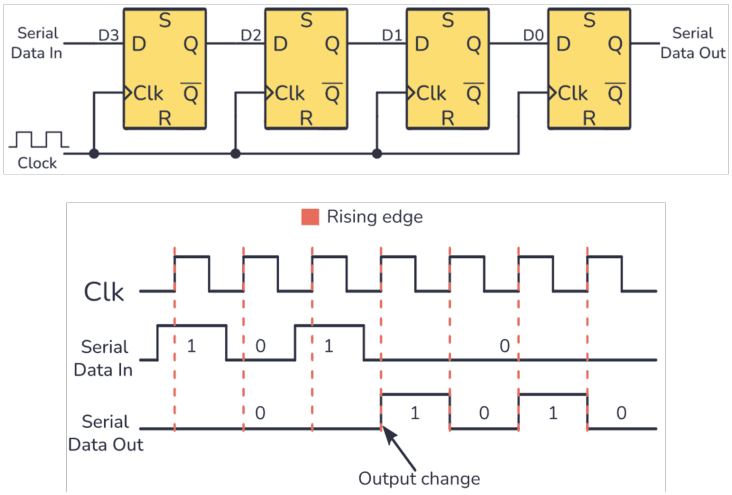



Figure 16: Esempio di shift register serial-in serial-out (SISO).

I registri a scorrimento possono avere ingressi e uscite sia paralleli che seriali. In un registro a scorrimento serial-in parallel-out (SIPO) i dati en-

trano serialmente, ma possono essere letti tutti insieme in parallelo. Un registro SIPO permette di convertire dati seriali in paralleli, ed è utile ad esempio per collegare dispositivi seriali a bus paralleli (figura 17). In un registro a scorrimento parallel-in serial-out (PISO) i dati vengono caricati in parallelo nei flip-flop. L'uscita è seriale, quindi i bit vengono letti uno alla volta. Questo tipo di registro si può usare per trasmettere dati paralleli su un canale seriale. In un registro a scorrimento parallel-in parallel-out (PIPO) i dati vengono caricati in tutti i flip-flop contemporaneamente e l'uscita è parallela, quindi i bit vengono letti tutti insieme. Un registro PIPO è usato quando serve semplicemente memorizzare un blocco di dati senza trasformazioni seriale/parallelo.

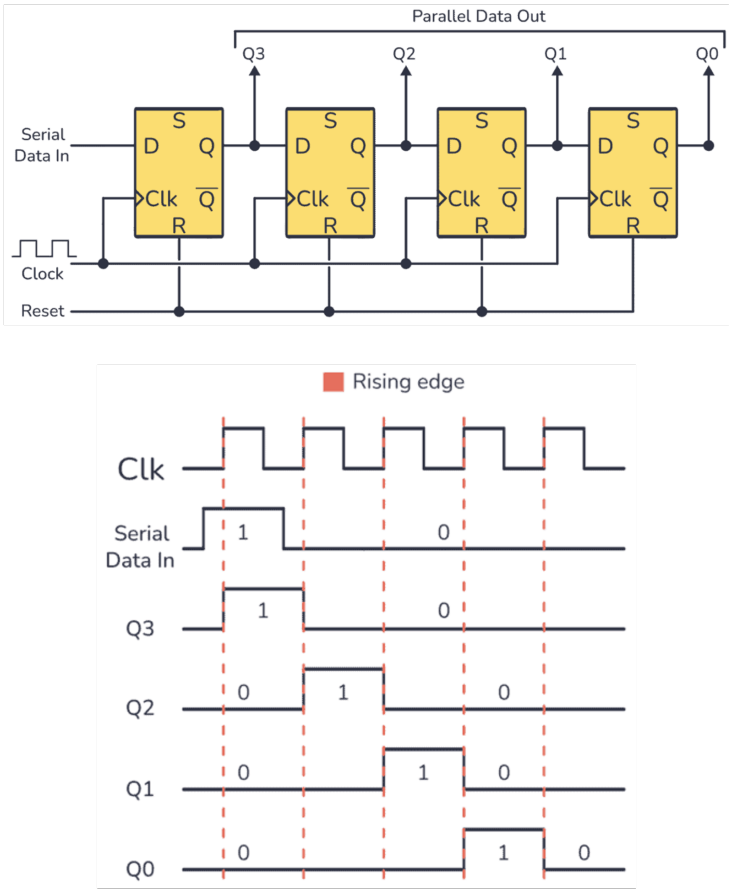



Figure 17: Esempio di shift register serial-in parallel-out.

## 3.4 Blocchi input/output (I/O)

Le FPGA hanno un numero elevato di pin per connettere segnali interni e esterni. Si hanno due tipologie di pin: i pin I/O e i pin dedicati. I pin I/O possono essere programmati dal digital desginer per essere input, output o bidirezionali. I pin dedicati sono codificati per una funzione specifica, e sono usati per fornire l'alimentazione alla FPGA, configurare la FPGA, e per l'ingresso dei segnali di clock.

## 3.5 Risorse di clock

I segnali di clock all'interno di una FPGA vengono usati da tutta la logica sequenziale. Tipicamente nelle FPGA si possono implementare clock con frequenze diverse, quando diverse parti della logica devono funzionare a diverse velocità. I clock vengono generati esternamente all'FPGA, ad esempio da circuiti oscillatori, e forniti in ingresso alla FPGA tramite pin dedicati. Le FPGA contengono anche logica di routing dedicata per ridurre il ritardo nella distribuzione del segnale di clock nelle varie parti del circuito.

# A Circuiti integrati

I **circuiti integrati (Integrated Circuits, IC)** sono circuiti elettronici i cui componenti sono realizzati in un piccolo chip di silicio. I circuiti integrati contengono, in un area di pochi $mm^2$ - $cm^2$, miliardi di componenti miniaturizzati (transistori, diodi, resistenze, capacità), connessi a formare circuiti digitali di varia complessità (porte logiche, registri, memorie, etc.).

Sono circuiti integrati digitali i seguenti dispositivi:

- **Microprocessore**: Circuito integrato progettato per eseguire funzioni aritmetiche, logiche e di controllo (e.g. CPU).
- **Microcontrollore**: CPU integrata con periferiche.
- **ASIC**: Application Specific Integrated Circuit. Circuito integrato progettato per un uso specifico.
- **FPGA**: Circuito integrato che può essere programmato sul campo per eseguire la funzione richiesta dall'utente.

# B Temporizzazione nelle FPGA

Come abbiamo già detto, la maggior parte della logica implementata in una FPGA è sincrona e deve quindi funzionare correttamente alla velocità definita dalla frequenza del clock.

Questo potrebbe non succedere per via di **ritardi nella propagazione del segnale (propagation delay)**. Il ritardo di propagazione è la quantità di tempo impiegata dai segnali per passare tra due flip-flop. Supponiamo che l'uscita di un FF passi attraverso logica combinatoria e interconnessioni prima di arrivare all'ingresso di un altro FF, e che ci metta 10 ns. Se i FF funzionano a una frequnza di 50 MHz, quindi con un periodo di 20 ns, il segnale riuscirà ad arrivare in tempo al secondo FF. Se la frequenza del clock fosse 200 MHz (periodo 5 ns), il tempo di propagazione del segnale sarebbe troppo lungo. In questo caso, si avrebbe un errore di temporizzazione (timing) e il circuito non funzionerebbe correttamente.

Per ovviare a questo problema ci sono due strategie. La più semplice, ma non sempre possibile, è di ridurre la frequenza del clock, in modo da avere un periodo di clock superiore al tempo di ritardo più alto. Un'altra possibilità è di svolgere meno operazioni in un ciclo di clock. Considerando l'esempio

appena fatto, si potrebbe dividere la logica combinatoria tra i due FF in più blocchi come meno funzionalità, riducendo quindi la quantità di operazioni da fare in un ciclo di clock. Dopo ciascuno di questi blocchi si inserisce un FF per salvare il dato intermedio dell'operazione. In questo modo si possono soddisfare i requisiti di timing a frequenze di clock più elevate.

Un altro concetto importante quando si considera la temporizzazione di una FPGA è quello di **tempi di setup e hold (setup and hold time)** (figura 18). L'input di un FF deve essere stabile per un breve lasso di tempo prima dell'arrivo del fronte del clock. Questo lasso di tempo è chiamato setup time. Il tempo di hold è simile al tempo di setup, ma si occupa degli eventi successivi al verificarsi di un fronte di clock. Il tempo di hold è il tempo minimo necessario affinché l'input di un FF sia stabile dopo un fronte di clock. Se la Static Timing Analysis indica che il progetto presenta violazioni del tempo di setup o hold, l'output dei FF potrebbe non essere stabile. Potrebbe essere zero, uno, o assumere un valore intermedio. Non è noto quale sia il valore all'uscita del FF. Questa condizione è nota come **metastabilità**.

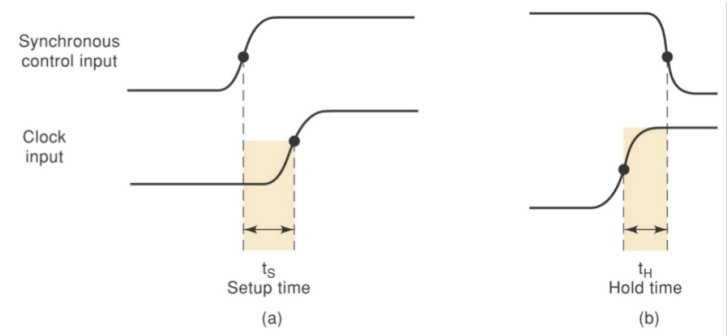



Figure 18: Illustrazione dei tempi di setup e hold per un segnale sincrono in ingresso ad un FF.

I tempi di setup ($t_{su}$) e hold ($t_h$), e il ritardo di propagazione ($t_p$) definisco il periodo del clock ($T$) come

$$T = t_{su} + t_h + t_p.$$

Nelle FPGA, i tempi di setup ($t_{su}$) e hold ($t_h$) per i FF sono fissi. Il digital designer ha controllo solo sul ritardo di propagazione.