Università degli Studi di Perugia


Dipartimento di Matematica e Informatica

# Report di Laboratorio
# Sistemi Elettronici e Sensori per l'Informatica

**Studenti**

Dedi Vittorio
Palladino Danilo

Anno Accademico 2024-2025

# Sommario

# Introduzione

Lo scopo di questo report di laboratorio è descrivere la nostra prima esperienza con le FPGA, con particolare riferimento alla scheda Basys3 e all'ambiente di sviluppo Vivado. Inizieremo con una breve introduzione alle FPGA, questi sono dei dispositivi composti da circuiti integrati che consentono di definire il comportamento dei circuiti elettronici attraverso l'uso di linguaggi di descrizione hardware, come Verilog o VHDL.

Per poter utilizzare una di queste schede è necessario disporre di un ambiente di sviluppo e di un linguaggio che permetta di comunicare con il dispositivo. In questo caso, abbiamo adottato Vivado, una piattaforma offerta da Xilinx (ora facente parte di AMD), che rappresenta uno degli strumenti più diffusi per la progettazione e l'implementazione di progetti FPGA. Vivado ci consente di gestire l'intero flusso di lavoro, dalla definizione delle specifiche dei circuiti, passando per la simulazione e la verifica, fino alla generazione del file bitstream necessario per programmare fisicamente il dispositivo FPGA.

Con questo elaborato esploreremo i principi delle FPGA attraverso l'analisi e la spiegazione degli esercizi pratici svolti in laboratorio. Fornendo un'introduzione all'uso di Vivado come ambiente di sviluppo per i progetti FPGA. L'obiettivo finale è comprendere come queste tecnologie siano utili e come possano essere impiegate per realizzare progetti rendendo così la programmazione hardware più accessibile e flessibile.

L'elaborato è organizzato in due capitoli, ciascuno suddiviso in sotto capitoli dedicati agli esercizi svolti. In ogni sottocapitolo analizzeremo nel dettaglio i codici e i risultati ottenuti durante le attività di laboratorio. La scelta di questa suddivisione è stata presa per evidenziare la progressione graduale della complessità degli esercizi: i primi quattro, contenuti nel Capitolo 1, rappresentano un'introduzione pratica al linguaggio di programmazione e all'interfaccia software, mentre i successivi tre, contenuti nel Capitolo 2, presentano un livello di difficoltà più elevato, applicando le conoscenze acquisite nel primo capitolo.

# Capitolo 1: Fondamenti e introduzione

In questo capitolo vedremo le nostre personali implementazioni dei primi 4 codici scritti per programmare le FPGA. Le esercitazioni si sono svolte all'interno del laboratorio del dipartimento di fisica e la scheda che abbiamo usato è stata la Basys3 che monta il chip Xilinx Artix-7. Per poterci interfacciare alla scheda abbiamo utilizzato un Raspberry Pi sul quale abbiamo avviato Vivado.

Per iniziare, abbiamo creato un nuovo progetto tramite l'interfaccia iniziale del software. Dopo aver assegnato un nome al progetto e scelto la cartella di destinazione in cui verrà creato, è stato necessario specificare il modello della scheda utilizzata.
Successivamente, nella sezione "Add Sources", abbiamo aggiunto i design sources. In questa fase, è stato importato un file da internet che conteneva una definizione precisa dei nomi degli input e output della scheda FPGA. Questo file ci ha permesso di configurare automaticamente i segnali di ingresso e uscita del progetto.

Dopo aver completato la configurazione delle sorgenti e della struttura del progetto, siamo passati alle seguenti fasi operative: Run Synthesis, Run Implementation e Generate Bitstream

## 1.1 Primo Esercizio

Questo modulo permette di accendere o spegnere il LED 0 alla pressione del pulsante centrale della pulsantiera.
Di seguito è riportato il codice realizzato:

```
module led_accensione_bottone(
   input clk, // clock del sistema
   input btnC, // bottone centrale
   output reg [15:0] led // array contenente i 16 LED della scheda
);

    reg oldbtnC = 0; // Variabile ausiliaria

    always @ (posedge clk) begin
        oldbtnC <= btnC;
        if (oldbtnC == 0 && btnC == 1) begin
            led[0] <= !led[0];
```

```
        end
    end
endmodule
```

Nella parte iniziale vengono definiti gli input, che in questo caso sono il pulsante centrale (btnC) e il clock di sistema (clk). Successivamente, vengono definiti gli output, rappresentati dai 16 LED presenti sulla scheda.
La linea di codice **`reg oldbtnC = 0`** inizializza a 0 una variabile che sarà utile per verificare se il pulsante centrale è stato effettivamente premuto. Senza questa variabile, ci sarebbe stato un problema: il sistema avrebbe continuato a invertire lo stato del LED 0 durante ogni ciclo di clock in cui il pulsante risultava premuto. Questo avrebbe causato un comportamento instabile, poiché la pressione del pulsante avrebbe potuto generare più cambi di stato del LED 0 a causa dell'elevata frequenza del clock.
Il blocco always è sensibile al fronte di salita del clock **`posedge clk`** e aggiorna lo stato della variabile **`oldbtnC`** con il valore corrente di **`btnC.`** Se **`oldbtnC`** è 0 e **`btnC`** è 1, significa che c'è stato un fronte di salita sul pulsante centrale, quindi il LED 0 invertirà il suo stato (da acceso a spento o viceversa) tramite l'operazione **`!led[0]`** .

## 1.2 Secondo Esercizio

La richiesta era quella di generare un codice che permettesse di far lampeggiare un LED con una frequenza di 2 Hz.
Di seguito è riportato il codice realizzato:

```
module led_lampeggiante_2HZ(
   input clk, // clock del sistema
   output reg [15:0] led // array contenente i 16 led della scheda
);

    reg [26:0] counter = 0; // contatore

    always @ (posedge clk) begin
       if (counter == 50000000) begin
          counter <= 0; // Reset del contatore
          led[0] <= !led[0]; // Inversione dello stato del LED
       end
       else begin
          counter <= counter + 1; // Incremento del contatore
```

```
        end
    end
endmodule
```

Nella parte iniziale definiamo gli input, che in questo sono: il clock di sistema **`clk`** e l'output ovvero i 16 LED presenti sulla scheda. La variabile counter viene inizializzata a 0 e verrà usata per dividere la frequenza del clock.
Nel blocco always, sensibile al fronte di salita del clock **`posedge clk`**, il contatore viene incrementato a ogni ciclo di clock, quando il valore del contatore raggiunge 50.000.000, il contatore viene resettato a 0 e lo stato del LED viene invertito con l'operazione **`led[0] <= !led[0]`**. Questo processo crea un segnale che fa lampeggiare il LED a una frequenza di 2 Hz. Senza l'utilizzo del contatore per dividere la frequenza del clock, il LED lampeggerebbe a una velocità estremamente elevata, rendendo il cambiamento di stato impercettibile all'occhio umano.

## 1.3 Terzo Esercizio

La richiesta era quella di far lampeggiare un LED con una frequenza di 2Hz solo quando il primo switch è attivo.
Di seguito è riportato il codice realizzato:

```
module led_lampeggiante_2HZ_switch(
   input clk, // clock del sistema
   input sw[15:0], // array contenente i 16 switch della scheda
   output reg [15:0] led // array contenente i 16 led della scheda
);

    reg [26:0] counter = 0; // contatore

    always @ (posedge clk) begin
       if (sw[0]== 1) begin // check stato dello switch
          if (counter == 50000000) begin
             counter <= 0; // reset del contatore
             led[0] <= !led[0]; // cambio di stato del LED
          end
          else begin
             counter <= counter + 1; // incremento del contatore
          end
```

```
        end
        else begin
           counter <= 0; // reset del contatore se lo switch è OFF
           led[0] <= 0; // spegnimento del led
        end
    end
endmodule
```

Il codice di questo esercizio è molto simile a quello dell'esercizio precedente ma rispetto a quest'ultimo il comportamento è stato modificato.
Abbiamo un'istruzione di controllo if sullo switch **`sw[0]`**, che opera nel modo seguente: Quando lo switch è attivo quindi quando: **`sw[0] == 1`**, il contatore si comporta come nell'esercizio precedente, generando un lampeggio del LED 0 con una frequenza di 2 Hz, mentre quando lo switch è spento **`sw[0] == 0`**, il contatore viene resettato e il LED rimane spento.

## 1.4 Quarto Esercizio

La richiesta per questo esercizio era di far lampeggiare un LED con una frequenza di 2 Hz quando il primo switch è disattivo, e con una frequenza doppia (4 Hz) quando lo switch è attivo.
Di seguito è riportato il codice realizzato:

```
module led_lampeggiante_2HZ_4HZ(
    input clk, // clock del sistema
    input sw[15:0], // array contenente i 16 switch della scheda
    output reg [15:0] led // array contenente i 16 led della scheda
);

    reg [26:0] counter = 0; // contatore

    always @ (posedge clk) begin
        if (sw[0] == 1) begin // check stato dello switch
            if (counter == 25000000) begin
                counter <= 0; // reset del contatore
                led[0] <= !led[0]; // cambio di stato del LED
            end
```

```
            else begin
               counter <= counter + 1; // incremento del contatore
            end
         end
         else begin // se switch spento
            if (counter == 50000000) begin
               counter <= 0; // reset del contatore
               led[0] <= !led[0]; // cambio di stato del LED
            end
            else begin
               counter <= counter + 1; // incremento del contatore
            end
         end
   end
endmodule
```

Il codice appena visto è molto simile a quello del precedente esercizio, ma il comportamento del sistema varia a seconda  dello stato dello switch. In particolare: Quando lo switch **`sw[0]`** è attivo, ovvero nel caso in cui **`sw[0] == 1`**, il contatore genera un lampeggio del LED 0 con una frequenza di 4 Hz, ciò avviene grazie alla riduzione del numero massimo del contatore a 25000000.
D'altro canto quando lo switch è disattivo: **`sw[0] == 0`**, il contatore permette al LED 0 di lampeggiare con una frequenza di 2 Hz, mantenendo il limite massimo del contatore a 50000000.

# Capitolo 2: Prosecuzione e sviluppo dei codici

## 2.1 Sesto Codice

La richiesta per questo esercizio era quella di accendere inizialmente un solo LED e fare in modo che la luce si spostasse a sinistra a ogni pressione del pulsante sinistro della board, fermandosi quando si raggiunge il LED più a sinistra (LED[15]).
Di seguito è riportato il codice realizzato:

```
module spostamento_sinistra_led(
   input clk, // clock del sistema
   input btnL, // bottone sinistro
   output reg [15:0] led // array contenente i 16 LED della scheda
);

   reg [3:0] i = 0; // indice per rappresentare la posizione del
                       LED attivo
   reg Old_btnL = 0; // stato precedente del pulsante

   always @ (posedge clk) begin
      if (btnL && !Old_btnL) begin // Controlla il
                                      fronte di salita
                                      del pulsante
         if (i < 15) begin
            i <= i + 1; // Incrementa l'indice per spostare il
                           LED a sinistra
         end
      end

      Old_btnL <= btnL; // Memorizza lo stato corrente
                           del pulsante
       // Aggiorna lo stato dei LED in base all'indice
       led <= 16'b0; // Spegne tutti i LED
       led[i] <= 1; // Accende il LED corrispondente all'indice
   end
endmodule
```

Nella prima parte del codice vengono inizializzati gli ingressi e le uscite. Tra questi troviamo il clock di sistema, il pulsante sinistro **`btnL`**, che servirà per spostare il LED attivo lungo l'array di LED, e due variabili: **`i`** e **`Old_btnL`**. La variabile **`i`** viene utilizzata come indice per selezionare il LED da accendere, mentre **`Old_btnL`** serve a prevenire movimenti indesiderati del LED, evitando che lo scorrimento avvenga più volte per una singola pressione del pulsante. Quando il pulsante sinistro **`btnL`** viene premuto, l'indice **`i`** viene incrementato, spostando il LED attivo verso sinistra. Il movimento si ferma automaticamente quando l'indice raggiunge il valore massimo di 15, corrispondente al sedicesimo LED (LED[15]). Ad ogni iterazione, tutti i LED vengono inizialmente spenti assegnando **`16'b0`** al registro led. Successivamente, il LED corrispondente all'indice attuale (**`led[i]`**) viene acceso, visualizzando così il nuovo stato. All'avvio, il LED[0] è acceso. Ogni pressione del pulsante sinistro sposta il LED attivo di una posizione verso sinistra. Quando si raggiunge LED[15], il movimento si arresta e il LED rimane acceso sull'ultima posizione disponibile.

## 2.2 Settimo Codice

La richiesta per questo esercizio era quella di accendere inizialmente un solo LED e fare in modo che la luce si spostasse a destra a ogni pressione del pulsante destro della board, fermandosi quando si raggiunge il LED più a destra (LED[0]). Possiamo notare come ciò sia tutto molto simile all'esercizio precedente, talvolta anche la spiegazione sarà quasi uguale, con le modifiche necessarie per adattarla a questo caso.
Di seguito è riportato il codice realizzato:

```
module spostamento_destra_led(
   input clk, // clock del sistema
   input btnR, // bottone sinistro
   output reg [15:0] led // array contenente i 16 LED della scheda
);

   reg [3:0] i = 15; // indice per rappresentare la posizione del
                        LED attivo
   reg Old_btnR = 0; // stato precedente del pulsante

   always @ (posedge clk) begin
      if (btnR && !Old_btnR) begin // Controlla il
                                      fronte di salita
                                       del pulsante
         if (i > 0) begin
            i <= i - 1; // Incrementa l'indice per spostare il
```

```
                        LED a sinistra
            end
        end

        Old_btnR <= btnR; // Memorizza lo stato corrente
                          del pulsante
        led <= 16'b0; // Spegne tutti i LED
        led[i] <= 1; // Accende il LED corrispondente all'indice
    end
endmodule
```

Nella prima parte del codice vengono inizializzati gli ingressi e le uscite. Tra questi troviamo il clock di sistema, il pulsante destro **`btnR`**, che servirà per spostare il LED attivo lungo l'array di LED, e due variabili: **`i`** e **`Old_btnR`**. La variabile **`i`** viene utilizzata come indice per selezionare il LED da accendere, mentre **`Old_btnR`** serve a prevenire movimenti indesiderati del LED, evitando che lo scorrimento avvenga più volte per una singola pressione del pulsante. Quando il pulsante destro del **`btnR`** viene premuto, l'indice **`i`** viene decrementato, spostando il LED attivo verso destra. Il movimento si ferma automaticamente quando l'indice raggiunge il valore minimo di 0, corrispondente al primo LED (`LED[0]`). Ad ogni iterazione, tutti i LED vengono inizialmente spenti assegnando **`16'b0`** al registro led. Successivamente, il LED corrispondente all'indice attuale (**`led[i]`**) viene acceso, visualizzando così il nuovo stato. All'avvio, il LED[15] è acceso. Ogni pressione del pulsante destro sposta il LED attivo di una posizione verso destra. Quando si raggiunge LED[0], il movimento si arresta e il LED rimane acceso sull'ultima posizione disponibile.

# Conclusioni

Il laboratorio svolto ha rappresentato un'importante opportunità per entrare in contatto con l'interessante mondo delle FPGA. In particolar modo, l'utilizzo della scheda Basys3 ci ha permesso di vivere un'esperienza formativa importante. Durante il percorso, abbiamo acquisito una comprensione di come lavorare con le FPGA e del loro utilizzo in contesti reali, sebbene in una forma contenuta. I vari esercizi ci hanno consentito di applicare concretamente i concetti teorici inizialmente spiegati, rafforzando le nostre competenze. Inoltre, l'approccio pratico al design digitale ci ha permesso di comprendere meglio come il software e l'hardware possano essere integrati in un unico sistema, aprendo la strada alla progettazione di soluzioni avanzate che utilizzano circuiti digitali programmabili.

# Bibliografia

Digilent Inc. **Basys 3 Programming Guide.** Preso da: https://digilent.com/reference/learn/programmable-logic/tutorials/basys-3-programming-guide/start (2024)