

# Esperienza 4: Programmazione open-hardware mediante HDL Verilog di FPGA su evaluation board Basys3

Laura Francesca Iacob, Sara Pieri, Sara Schippa  
Università degli Studi di Perugia, CdL in Fisica

Anno Accademico 2024/2025

**Estratto:** Negli ultimi anni lo sviluppo dell'elettronica si è indirizzato verso la programmazione open-hardware. Strumenti come l'FPGA permettono di avere a che fare con architetture programmabili secondo i propri interessi con un'ampia flessibilità rispetto ad elementi elettronici progettati per svolgere un'unica operazione.

Se ne verifica il funzionamento mediante l'esecuzione di semplici istruzioni, scritte in linguaggio Verilog, riferite all'evaluation board su cui viene montato.

**Keywords:** FPGA, hardware programming, Verilog, Vivado

## 1 Introduzione teorica

Il Field Programmable Gate Array (FPGA) è un dispositivo logico programmabile. Storicamente, venne inizialmente impiegato nell'area del *prototyping*, successivamente, con l'incremento del numero di celle, l'uso di frequenze maggiori e una crescente economicità, la sua diffusione iniziò a riguardare l'applicazione diretta in open-hardware.

### 1.1 Proprietà elettroniche dell'FPGA

In relazione a computing devices come microprocessori, microcontrollori o Application-Specific Integrated Circuits (ASIC), l'FPGA presenta delle caratteristiche in comune: sono tutte componenti elettroniche digitali e programmabili. La loro base di operazione sono gli *0* ed *1 logici*, corrispondenti a due valori di tensione standard, ed il loro compito è quello di leggere in sequenza le istruzioni del programma e di eseguirle.

La specificità dell'FPGA risiede nella fase di programmazione. Se negli altri dispositivi l'esecuzione del programma avviene a livello di software e non modifica il chip, l'FPGA è una componente a logica programmabile, ovvero il programma agisce sull'hardware e modifica il circuito stesso.

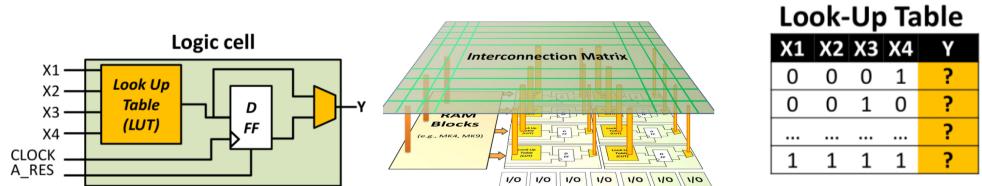
L'FPGA non possiede un layout predefinito perchè il design delle porte è riprogrammabile, quindi l'hardware è rinconfigurabile. L'elevata flessibilità del dispositivo fa sì che ci sia un minor divario tra hardware e software per un elemento elettronico che è intrinsecamente concorrente ed eterogeneo. A differenza della Central Processing Unit (CPU) e della Graphic Processing Unit (GPU), l'FPGA ha un'architettura adattabile, un set di istruzioni malleabile ed un livello di parallelismo eccellente.

Un'ulteriore confronto può essere fatto con l'ASIC che necessita di più tempo per la progettazione, ha un design permanente, lavora in un ampio range di frequenza ed è dotato di porte non riprogrammabili. In un primo momento il costo dell'investimento è alto, ma diventa più accessibile una volta avvenuta la specializzazione. Al contrario, l'FPGA non richiede particolari investimenti iniziali, ma la sua riprogrammabilità aumenta i costi del prodotto finale.

D'altronde, l'unica difficoltà nell'utilizzo di un elemento di questo tipo è la curva di apprendimento, dato che la programmazione a basso livello richiede la conoscenza di linguaggi che agiscono a livello hardware.

## 1.2 Struttura dell'FPGA

La componente fondamentale dell'FPGA è la cella logica (Figure 1 sinistra). Questa è dotata di un numero di ingressi arbitrario ed una uscita determinata dalla tabella di verità<sup>1</sup> ad essa associata in base all'operazione che deve compiere secondo il programma in esecuzione. In alternativa, l'output può essere definito da un *flip flop*, ovvero una unità di memoria. A decidere quale dei due sistemi viene messo in atto è il multiplexer (Mux), che funge da selettore.

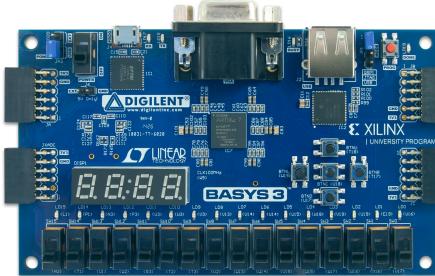


**Figure 1:** Cella elementare (sinistra), matrice di interconnessione (centro) ed esempio di tabella di verità (destra).

Un FPGA è composto da milioni di celle elementari, connesse tra loro da una matrice di interconnessione (Figure 1 centro). L'attivazione delle celle opportune è regolata dal programma in esecuzione, che interviene sulla configurazione circuitale adottata dal dispositivo per farle svolgere l'operazione richiesta.

La suddetta astrazione di celle e memorie permette di creare qualsiasi circuito si voglia. Una volta redatto il programma desiderato in un linguaggio opportuno, un file di dati procederà in direzione dell'FPGA per trasformarlo.

L'evaluation board (Figure 2) è una piattaforma elettronica provvista di pin, connettori, pulsanti e LED in connessione con l'FPGA, il quale sfrutta questi elementi nell'esecuzione del programma. Il riconoscimento della board da parte del dispositivo è messa in atto attraverso un file contenente i *constraints* specifici del modello in uso.



**Figure 2:** Evaluation board Basys3.

## 1.3 FPGA programming

La programmazione dell'FPGA può avvenire mediante disegni diretti di porte logiche, una metodologia immediata che però non ha degli standard fissi e diventa difficoltosa per progetti di ampia complessità.

Ciò che si usa più spesso sono le Hardware Description Languages (HDLs) che prevedono la scrittura del programma mediante un codice standardizzato e riutilizzabile, facilmente impiegabile per la realizzazione di progetti più complessi.

La programmazione open-hardware è ben diversa da quella a livello del software, dato che le istruzioni vengono eseguite tutte contemporaneamente su una scala dei tempi scandita dal clock del dispositivo, ad una frequenza specifica (somiglia maggiormente alla programmazione in parallelo).

<sup>1</sup>La tabella di verità è una tabella a ruolo combinatorio che definisce l'output di qualsiasi porta logica riassumendo tutti i risultati che questa può dare in base ai valori in input secondo l'algebra booleana.

Il codice HDL viene impiegato per descrivere un circuito e può descrivere due tipi di flussi differenti.

L'impiego più diffuso non prevede l'esecuzione del programma in forma diretta, perché questa richiede molto tempo, soprattutto se i progetti sono di lunghezza considerevole. Per questo si isolano i moduli HDL e si verifica il loro funzionamento per mezzo di **test benches** specifiche. Le simulazioni permettono di studiare la funzionalità del codice per ciascuna istruzione separatamente ed individuare in tempi ragionevoli le eventuali correzioni necessarie.

L'alternativa più vicina alla programmazione software consiste nell'esecuzione diretta del codice tramite **simulazioni** per poter usufruirne immediatamente sulla board in uso. Dopo aver realizzato l'HDL code, facendo uso di un Integrated Development Environment (IDE) come Vivado, avviene un processo di *synthesis*: viene creato il circuito logico a livello astratto, tanto da poter visualizzare lo schematics e la tabella di verità associata (Figure 6).

La seconda fase è quella dell'*implementation*, che prevede il collegamento diretto con l'hardware: viene svolta una mappatura delle porte logiche astratte con relativa tabella di verità sull'evaluation board, individuando i pin corrispondenti agli elementi circuitali coinvolti e connettendoli tra di loro al fine di assegnare a ciascun elemento hardware il suo scopo (programmazione a vincoli: *constraints*, Paragraph 1.4).

Ora che le celle sono state mappate correttamente, si procede con la generazione del *bitstream* o *firmware*: un file di bit che viene inviato all'hardware e che trasporta le informazioni che vanno a caratterizzare il circuito, al fine di svolgere i comandi prestabiliti. L'invio del suddetto determina l'effettiva creazione del circuito, perché ne permette il flusso di dati e quindi il funzionamento.

## 1.4 Constraints file

Dal momento che l'FPGA non sa come è collegata agli elementi elettronici dell'evaluation board, come i LED, gli switch o il clock, nella stesura di un codice che ne fa uso è necessario specificare la posizione di tali elementi sulla board, e quindi i pin che li connettono all'FPGA.

Questo è possibile grazie al file di constraints, fornito dalla casa di produzione della evaluation board, e che contiene i "vincoli", cioè i pin, della board in dotazione per l'FPGA, ed elenca gli elementi circuitali corrispondenti.

In questo modo, l'IDE può sapere di un determinato elemento della board quando se ne fa uso nel codice, cioè nel circuito di output, può riconoscere la sua posizione e, al momento dell'implementazione, svolgere il corretto collegamento all'hardware.

L'uso degli elementi elencati nel constraints file è svolto tramite la dichiarazione degli input o degli output, a seconda dello scopo che si vuole dare loro, nel top level module del file sorgente.

```

6: # Clock signal
7: set_property PACKAGE_PIN W5 [get_ports clk]
8:   set_property IOSTANDARD LVCMOS33 [get_ports clk]
9:   create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
10:
11: # Switches
12: set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
13:   set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
14: set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
15:   set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
16: #set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
17:   #set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
```

**Figure 3:** Porzione del file di *constraints* dell'evaluation board Basys3: clock e primi switch.

## 2 Materiali e strumentazione

E' possibile consultare i codici usati per l'elaborazione e l'analisi dei dati nella repository GitHub. Ulteriori materiali inerenti sono disponibili presso la cartella *Drive* in collegamento ipertestuale.

In laboratorio è stata adoperata la strumentazione seguente:

- evaluation board Basys 3;
- FPGA artix-7;
- HDL Verilog;
- IDE Vivado.

## 3 Esercitazione in laboratorio

L'FPGA in dotazione è la artix-7, dotata di circa 33000 celle, può arrivare ad operare ad una frequenza del clock di 450 MHz partendo da un valore di costruzione della board di 100 MHz.

L'FPGA è inizialmente vuota, sono presenti solo i connettori collegati ai pin dei vari elementi utilizzabili, senza che questi siano ancora associati a operazioni da compiere. Per costruire il circuito funzionante che colleghi tra loro i vari elementi circuitali, si procede con la scrittura dell'HDL code e lo si sottopone a sintesi ed implementazione, per finire con la generazione del bitstream finale.

A partire dal terminale, si predispone l'IDE Vivado per usufruire dell'HDL Verilog.

### 3.1 Verilog

In Verilog i blocchi di codice prendono il nome di moduli. Un programma tipo è costituito da un *top level module* che svolge il ruolo di main, delimitato dai comandi “*module nome\_modulo()*” ed “*endmodule*”.

I moduli possono richiedere l'inserimento di parametri, da dichiarare tra parentesi tonde come *input* e *output*, corrispondenti a livello hardware a fili (bus se in forma vettoriale) di collegamento.

Il blocco di istruzioni di un modulo viene delimitato dai termini “*begin*” ed “*end*”, esattamente come nel linguaggio di programmazione software C++ si usano le parentesi graffe. In Verilog la fine dell'istruzione è identificata dal punto e virgola, infatti l'indentazione non è necessaria ma preferibile per garantire una maggior leggibilità del codice.

Lo scheletro del top level module è composto da variabili e da ulteriori blocchi. Questi ultimi possono essere di due tipologie, ovvero

- *initial*: eseguiti alla partenza della simulazione, specifica i valori iniziali assunti dalle variabili in gioco;
- *always*: eseguiti ogni volta che varia la condizione specificata, solitamente si impiega la scansione temporale di un clock.

Le variabili, in un primo momento, devono essere dichiarate, poi inizializzate nel modulo initial e successivamente impiegate a piacimento. Queste possono essere di tipo *wire*, ad indicare i “fili” di collegamento che trasportano l'informazione da un punto ad un altro del circuito venendo letti e assegnati, o di tipo *reg*, ovvero contenitori di signoli bit che memorizzano il valore dell'ultimo assegnamento operato.

L'assegnazione delle variabili avviene in maniera continua non bloccante, contrassegnata dalla combinazione di simboli “*<=*”. Essendo le istruzioni eseguite contemporaneamente, infatti, questo tipo di assegnazione viene resa effettiva al ciclo di clock successivo. Si analizzino gli esempi seguenti per comprendere la dinamica del sistema.

*Esempio 0*    $A <= 5;$    Al battito di clock successivo A assume il valore 5,  
 $B <= A;$    B prende il valore che A aveva in precedenza.

*Esempio 1*  $A \leq 5$ ; Al battito di clock successivo A assume il valore 5,  
 $A \leq 6$ ; contemporaneamente assime il valore 6: si genera un errore.

*Esempio 2*  $A \leq B$ ; Al battito di clock successivo A assume il valore di B,  
 $B \leq A$ ; B prende il valore che A aveva prima: le variabili si scambiano.

In caso si vogliano generare variabili in forma vettoriale, nella dichiarazione accanto al tipo si specificano l'indice dell'ultimo e del primo bit a partire da 0. Ciascun elemento può essere poi richiamato mediante le parentesi quadre con l'indice corrispondente, identicamente ai linguaggi di programmazione software come python.

### 3.2 Codici dei programmi implementati

#### 3.2.1 Porte logiche

I primi esercizi consistono nella codifica (Figure 4) di porte logiche e nella loro sintetizzazione, senza passare all'implementazione del circuito effettivo. Alla fine del processo, da Vivado viene generato un circuito visualizzabile tramite lo schematics e la tabella di verità cirrispondente (Figure 6).

```

module notMod (
    input wire A,
    output wire B
);
    assign B=!A;
endmodule

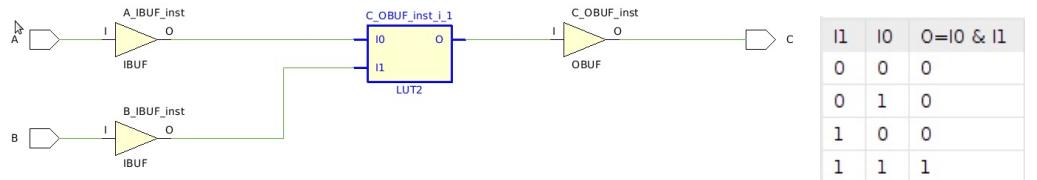
module andMod (
    input wire A,
    input wire B,
    output wire C
);
    assign C=A&B;
endmodule

module logicDoor (
    input wire A,
    input wire B,
    input wire C,
    output wire D,
    output wire E
);
    assign D=A&(B|C);
    assign E=B|C;
endmodule

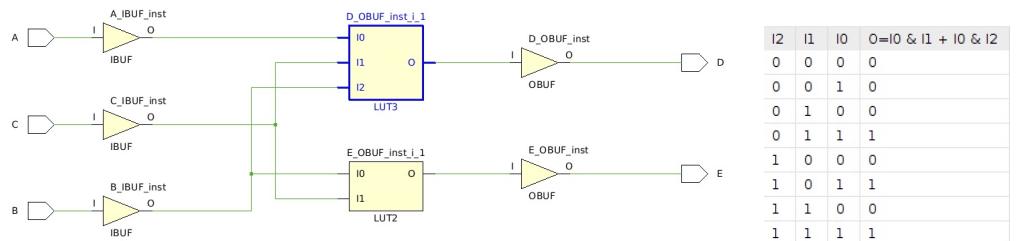
```

**Figure 4:** Programmi di implementazione di porte logiche elementari: not (sinistra), and (centro). A destra una porta logica composta a tre entrate e due uscite.

I programmi delle porte logiche proposte sono costituite da un unico modulo i cui parametri sono i valori ricevuti in input e quelli uscenti in output. A questi ultimi vengono assegnati i risultati delle operazioni logiche descritte, dove and, or e not sono rispettivamente contrassegnati dai simboli “&”, “|” e “!”.



**Figure 5:** Porta logica and: schematics (sinistra) e truth table (destra).



**Figure 6:** Porta logica composta: schematics (sinistra) e truth table (destra).

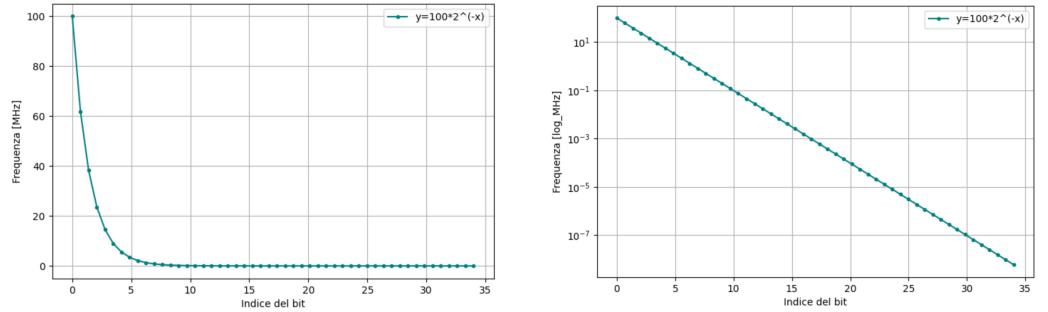
### 3.2.2 LED blinking

Si realizzano dei programmi che, una volta implementati ed inviato il bitstream generato, modificano lo stato acceso-spento dei LED in dotazione dell'evaluation board in uso. A questi, come anche al clock, si accede vettorialmente utilizzando i nominativi corretti indicati nel constraints file, corrispondenti ai pin di collegamento hardware.

Il primo codice permette al primo LED della board di accendersi e di spegnersi a ritmo della variazione della variabile “counter” ad un bit (Figure 8). Questa, ad ogni rintocco del clock, viene aumentata di una unità binaria grazie all'operazione “+1”: ad una frequenza di 100MHz il suo valore rimbalza tra lo 0 e l'1 logico.

Il LED si comporta di conseguenza, anche se, non potendo l'uomo distinguere eventi distanti tra loro nell'ordine dei nanosecondi, sembra costantemente acceso.

In un secondo tentativo, viene dichiarato un counter a 33 bit. Infatti, l'operazione binaria che accresce di una unità il counter fa sì che, per motivi legati all'algebra booleana, ogni bit cambi valore ad una frequenza che è pari a metà di quello che lo precede. Per cui, se il bit meno significativo è governato da una frequenza di 100MHz, il successivo da una di 50MHz, quello dopo ancora a 25MHz e così via, secondo un andamento esponenziale in base 2 (Figure 7).



**Figure 7:** Andamento della frequenza di variazione in funzione dell'indice i-esimo del bit: in scala lineare (sinistra) e logaritmica (destra).

Impostando la modifica del LED in base al 24-esimo bit, la frequenza di accensione e spegnimento si aggira intorno ai 2Hz, mentre al 32-esimo bit lo stato si modifica all'incirca ogni 20 secondi.

```

module blink (
    input clk,
    output reg [7:0] LED
);
reg counter;
initial;
    counter=0;
    always @ (posedge clk) begin
        LED[0] <= counter;
        counter <= counter +1;
    end
endmodule

module blink (
    input clk,
    output reg [7:0] LED
);
reg [32:0] counter;
initial;
    counter=0;
    always @ (posedge clk) begin
        LED[0] <= counter[23];
        counter <= counter +1;
    end
endmodule

module blink (
    input clk,
    output reg [7:0] LED
);
reg [32:0] counter;
initial;
    counter=0;
    always @ (posedge clk) begin
        LED[0] <= counter[31];
        counter <= counter +1;
    end
endmodule

```

**Figure 8:** Programmi che permettono il lampeggiamento del primo LED della board con una frequenza da 100MHz (sinistra), 2Hz (centro) e 0,05Hz.

Un'ultima manipolazione del LED è permessa dell'interruttore corrispondente (Figure 9). Si delinea un programma in cui il lampeggiamento della prima luce avviene solo nella condizione in cui lo switch risulti acceso.

```

module LED_sw (
    input clk,
    input [1:0] sw,
    output reg [7:0] LED
);
reg [32:0] counter;
initial
    counter=0;
always @ (posedge clk) begin
    if (sw[0]==1'b1) begin
        LED[0] <= counter[25];
        counter <= counter +1;
    end
end
endmodule

```

**Figure 9:** Programma che prevede il lampeggiamento del primo LED solo nel caso in cui l'interruttore corrispondente sia acceso.

### 3.2.3 Counter binario

Il counter definito nei passaggi precedenti può essere visualizzato nella sua evoluzione temporale mediante l'accensione dei LED opportuni (Figure 10). Usufruendo dei 16 LED a disposizione della board, si realizza un *ciclo for* che ne provoca il lampeggiamento in sequenza in base ai valori assunti da ciascun bit del counter. Ancora a causa dei limiti della percezione dell'occhio umano, si selezionano gli elementi dell'array del counter dal 23-esimo in poi.

```

module counter_binario (
    input clk,
    output reg [15:0] LED
);
reg [50:0] counter;
integer i;
initial
    counter=0;
always @ (posedge clk) begin
    for (i=0; i<16; i=i+1)
        LED[i] <= counter[23+i];
        counter <= counter +1;
    end
endmodule

```

**Figure 10:** Programma che permette alla board di fungere da counter binario mediante l'accensione e lo spegnimento dei LED.

### 3.2.4 Joystick

L'evaluation board è dotata di un joystick a cinque pulsanti: uno centrale e gli altri ad indicare ciascuno un punto cardinale. A queste nei moduli si ha accesso sotto forma di input mediante il nominativo indicato sul constraints file.

Si realizza un programma che permette di accendere e spegnere un LED prescelto, usando il tasto centrale del joystick (*btnC*) come interruttore (Figure 11). L'operazione da compiere, seppur semplice, richiede degli accorgimenti nella scrittura del codice. Essendo i tempi dell'FPGA scanditi dal clock ad un'alta frequenza, la pressione esercitata dal dito sul tasto del joystick viene interpretato come una serie di *1 logici* ed il suo rilascio fa tornare i bit a *0*. È necessario isolare il tocco del tasto, perchè, se non venisse fatto, il LED sarebbe acceso solo tenendo premuto il pulsante e spento in caso di rilascio.

```

module joystick_c (
    input clk,
    input btnC
    output reg [7:0] LED
);
reg old;
initial
    LED[0]=1'b0;
always @ (posedge clk) begin
    old<=btnC;
    if(btnC!=old & btnC==1)
        if(LED[0]==1'b0)
            LED[0]<=1'b1;
        else if (LED[0]==1'b1)
            LED[0]<=1'b0;
    end
endmodule

```

**Figure 11:** Programma che usa il tasto centrale del joystick come interruttore del primo LED.

A questo proposito, si sfrutta la peculiarità della programmazione open-hardware, in cui le istruzioni vengono eseguite contemporaneamente al clock successivo. Immagazzinando in una variabile reg “old”, si possono confrontare i valori binari assunti dal joystick a due istanti di clock successivi. Questo permette di individuare i punti di variazione del bit nel momento di pressione e di rilascio del pulsante. Perciò, la condizione richiesta è che le due variabili siano diverse tra loro ed il tasto centrale del joystick ad 1; in caso di LED acceso se ne richiede lo spegnimento e viceversa per mezzo di una ulteriore condizione ipotetica indicata dall’*if*.

Si implementa un secondo programma (Figure 12) che, partendo dalla condizione in cui il primo LED è brillante, utilizza i tasti destro e sinistro del joystick per accendere rispettivamente il LED successivo e precedente a quello attuale.

```

module kitt (
    input clk,
    input btnL,
    input btnR,
    output reg [15:0] LED
);
reg oldR;
reg oldL;
reg [4:0] i;
initial begin
    i=0;
    LED[0]=1'b1;
end
always @ (posedge clk) begin
    oldR<=btnR;
    oldL<=btnL;
    if (oldR!=btnR & btnR==1 & i>0) begin
        LED[i]<=1'b0;
        LED[i-1]<=1'b1;
        i<=i-1;
    end
    else if (oldL!=btnL & btnL==1 & i<15) begin
        LED[i]<=1'b0;
        LED[i+1]<=1'b1;
        i<=i+1;
    end
end
endmodule

```

**Figure 12:** Programma che, partendo dal primo LED, permette di accendere il LED a destra o a sinistra di quello corrente mediante i corrispondenti tasti del joystick.

---

La stesura del codice ha richiesto lo stesso meccanismo di isolamento del tocco del tasto del programma precedente, riferito sia al pulsante destro che a quello sinistro. La possibilità di spostare il LED di riferimento è fornita da una variabile “*i*”, che si modifica ad ogni click ad indicare l’indice della luce in uso.

Questo numero viene impiegato come ulteriore condizione dell’*if*, per impedire all’utente di superare i limiti destro e sinistro di numerazione dei LED della board.

## 4 Conclusioni

In definitiva si può affermare che l’implementazione dei programmi proposti è stata svolta correttamente, come pure testimoniano i video lasciati in collegamento ipertestuale: gli obiettivi prefissati per questa esperienza, quali la comprensione, limitata agli strumenti forniti, della programmazione open hardware e dell’FPGA, sono stati dunque raggiunti con successo.

### Riferimenti

1. Duranti, M.; Laboratorio di elettronica e Tecniche di acquisizione dati. Università degli Studi di Perugia, 2024. Dispense online: [https://www.fisgeo.unipg.it/~duranti/laboratoriode/laboratorioele\\_24-25.html](https://www.fisgeo.unipg.it/~duranti/laboratoriode/laboratorioele_24-25.html).
2. Mariotti, M.; Introduction to FPGA Programming. Università degli Studi di Perugia, 2024. Dispense online: [https://www.mirkomariotti.it/courses\\_20180529-alghero/](https://www.mirkomariotti.it/courses_20180529-alghero/).