Laboratorio di elettronica e tecniche d'acquisizione dati

Esperienza III: FPGA

Gentili Irene, Pace Lorenzo, Tintori Filippo

11 novembre 2024

Indice

1 Introduzione

Lo scopo di questa relazione è descrivere i metodi adottati e i risultati ottenuti nell'ambito della programmazione di una FPGA mediante l'utilizzo di una scheda dedicata. Verranno analizzati i programmi sviluppati in linguaggio Verilog per affrontare le consegne assegnate, discutendo nel dettaglio il comportamento della scheda utilizzata e l'efficacia delle soluzioni implementate.

2 Aspetti teorici

Una *Field Programmable Gate Array* (FPGA) è un dispositivo costituito da un circuito integrato le cui funzionalità logiche ed elaborative sono programmabili e modificabili attraverso opportuni linguaggi di descrizione dell'hardware (*HDL*, *Hardware Description Language*). La FPGA è composta da un insieme di *Configurable Logic Blocks* (CLB), a loro volta costituiti da più *Logic Cells*. Generalmente, una singola cella include una *look-up table* (LUT) che consente di mappare qualsiasi funzione binaria con 4 input e 1 output, un *flip-flop* per la memorizzazione e, eventualmente, un multiplexer (*mux*).

La differenza principale tra una FPGA e altri chip progettati per operazioni specifiche, come gli *ASIC* (*Application-Specific Integrated Circuits*), risiede nella loro flessibilità: mentre gli *ASIC* sono progettati e ottimizzati per un'applicazione specifica e non possono essere modificati una volta prodotti, una FPGA può essere configurata e riconfigurata per molteplici applicazioni. Gli elementi di circuito interni di una FPGA sono disposti in una struttura fissa (CLB), ma le interconnessioni tra di essi sono completamente riconfigurabili.

Questa riconfigurazione viene realizzata mediante un linguaggio di descrizione dell'hardware, generalmente a basso livello. A differenza dei linguaggi di programmazione tradizionali, dove gli assegnamenti definiscono istruzioni eseguite sequenzialmente da un processore, negli *HDL* si definiscono blocchi hardware, fili e circuiti. Non esiste un'esecuzione sequenziale né un tempo di esecuzione (*run-time*): tutte le operazioni sono eseguite simultaneamente, e la sequenzialità deve essere esplicitamente progettata.

L'esecuzione di un programma *HDL*, una volta completato il controllo sintattico, segue alcune fasi fondamentali: nella fase di sintesi, il software genera una rete di porte logiche elementari; questa rete viene successivamente tradotta in porte fisiche sulla FPGA attraverso il processo di implementazione. Infine, la fase di generazione del *bitstream* e la programmazione del dispositivo consentono di utilizzare la scheda per l'applicazione specificata.

3 Strumentazione

Di seguito sono elencati gli strumenti e i materiali utilizzati per lo svolgimento dell'esperienza:

- il software Vivado, con supporto al linguaggio di descrizione dell'hardware Verilog;
- una *Evaluation Board* modello Basys3.

La *Evaluation Board* utilizzata presenta una frequenza operativa di 450 MHz, dispone di 16 interruttori e 16 LED e contiene un totale di 33.280 celle logiche. Il file di *constraint* associato al progetto è "`Basys3 master.xdc`".

In allegato alla presente relazione è incluso il codice sorgente implementato durante l'esperienza.

4 Programmi

In questa sezione saranno analizzati, commentati e motivati i codici scritti per configurare la **FPGA**, al fine di soddisfare le consegne assegnate.

4.1 Porte logiche

Una porta logica è un circuito elementare che implementa le operazioni fondamentali dell'algebra booleana (operazioni $\mathbb{B} \to \mathbb{B}$, dove $\mathbb{B} = \{0, 1\}$) o una loro combinazione.

Le porte logiche sono i blocchi costitutivi fondamentali nei circuiti digitali. Esse manipolano segnali binari di ingresso in base a una funzione logica predefinita e sono utilizzate per progettare sistemi complessi come processori, memorie e unità di controllo. Le principali porte logiche includono:

- **AND**: restituisce 1 solo se tutti gli ingressi sono 1;
- **OR**: restituisce 1 se almeno un ingresso è 1;
- **NOT**: restituisce l'inverso del valore di ingresso;
- **NAND**, **NOR**, **XOR**, **XNOR**: varianti avanzate utilizzate per combinazioni logiche specifiche.

In questa sottosezione saranno presentati i codici sviluppati per l'implementazione delle porte logiche con le rispettive tabella di verità generate da Vivado, verificando che corrispondano a quelle attese.

Le porte logiche sono fondamentali per rappresentare e manipolare dati digitali. Ad esempio, la porta **AND** è usata in circuiti per funzioni di mascheramento, mentre la **OR** è impiegata per unire segnali. La comprensione e l'implementazione di tali circuiti rappresentano il primo passo verso la progettazione di sistemi digitali più complessi.

4.1.1 Porta NOT

La porta **NOT** è una delle porte logiche fondamentali e realizza l'operazione di negazione logica. Prende un singolo ingresso e restituisce l'inverso del suo valore: se l'ingresso è 1, l'uscita sarà 0, e viceversa.

```
1 module notm (
2     input wire A ,
3     output wire B
4     );
5
6     assign B = ~ A ;
7
8 endmodule
```

$B = \sim A$	
A	B
0	1
1	0

4.1.2 Porta AND

La porta **AND** è una porta logica fondamentale che implementa l'operazione di congiunzione logica. Restituisce 1 in uscita solo se tutti gli ingressi sono 1; in caso contrario, l'uscita sarà 0.

```
1 module notm (
2     input wire A, B,
3     output wire C
4     );
5
6     assign C = A & B ;
7 endmodule
```

$C = A \wedge B$		
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

4.1.3 Porta OR

La porta **OR** è una porta logica fondamentale che implementa l'operazione di disgiunzione logica. Restituisce 1 in uscita se almeno uno degli ingressi è 1; l'uscita sarà 0 solo quando tutti gli ingressi sono 0.

```
module orm (
    input wire A, B,
    output wire C
    );

    assign C = A | B ;

endmodule
```

$C = A \vee B$		
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

4.1.4 Combinazione di porte logiche

Come ultimo esercizio relativo alle porte logiche implementato all'interno dell'FPGA, viene rappresentata un'espressione che combina le operazioni booleane `AND` e `OR`. In particolare, sono definiti tre ingressi (A, B, C) e due uscite (D, E). L'uscita D assume il valore $A \wedge (B \vee C)$, mentre E è definito come $B \vee C$.

```
module complex (
    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
```

$E = B \vee C$ $D = A \wedge E$				
A	B	C	D	E
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

4.2 Lampeggio del LED[0] a 100 MHz

```
module blink (
    input CLK100MHZ,
    output reg [15:0] LED
);
reg counter;
initial
    counter = 0;

always @(posedge CLK100MHZ) begin
    LED[0] <= counter;
    counter <= counter + 1;
    end
endmodule
```

In questo primo programma implementato nell'FPGA, si fa lampeggiare il primo LED alla frequenza di 100 MHz, che corrisponde alla frequenza tipica del clock definito come ingresso. L'output è l'array che rappresenta i 16 LED presenti nell'*evaluation board*. Con `reg counter` si implementa un contatore come registro a 1 bit, che quindi può assumere solo i valori 0 e 1. Il contatore viene inizializzato a zero. Ad ogni ciclo di clock, il contatore viene incrementato e il valore del contatore viene assegnato al `LED[0]`. Questo fa sì che il LED lampeggi alla frequenza del clock. Ovviamente, il lampeggio non è percepibile ad occhio nudo, e infatti si osserva l'accensione persistente.

4.3 Lampeggio del LED[0] a 2 Hz

```
module blink2Hz (
    input CLK100MHZ,
    output reg [15:0] LED;
);
    reg [32:0] counter;
    initial
        counter = 0;
always @(posedge CLK100MHZ) begin
    LED[0] <= counter[23];
    counter <= counter + 1;
    end
endmodule
```

Al fine di osservare un effettivo lampeggiamento, si riduce notevolmente la frequenza. In questo caso, si mantiene come input il clock con frequenza di 100 MHz e come output il registro `reg [15:0] LED`, che rappresenta i LED dell'*evaluation board*. Successivamente, viene dichiarato un contatore a 33 bit, inizializzato a 0.

Il valore del `LED[0]` viene impostato sul ventiquattresimo bit del contatore tramite `LED[0] <= counter[23]`. Poiché il ventiquattresimo bit del contatore cambia valore ogni 16.777.216 cicli di clock, il LED lampeggerà a una frequenza di 2 Hz.

4.4 Lampeggio del LED[0] azionato da switch

```
module blinkIFswitch (
    input CLK100MHZ,
    input sw,
    output reg [15:0] LED
);

reg [32:0] counter;

initial
    counter = 0, LED = 0;

always @(posedge CLK100MHZ) begin
    if (sw) begin
        LED[0] <= counter[23];
        counter <= counter + 1;
    end else begin
        LED[0] = 0;
        counter = 1;
        end
    end
endmodule
```

In questo caso, si dichiara come input il clock e l'interruttore `sw`, mentre come output si definisce un registro a 16 bit per il controllo dei LED. Successivamente, viene dichiarato il registro `reg [32:0] counter`, un contatore a 33 bit utilizzato per contare i cicli del clock, e inizializzato a `0`, insieme al primo LED. Ad ogni ciclo, se il primo switch è alto, il contatore viene incrementato. Il bit `counter[23]` viene utilizzato per accendere e spegnere `LED[0]` in base alla sua alternanza tra 0 e 1, creando così un lampeggio (con la stessa frequenza di 2 Hz dell'esercizio precedente). Se invece `sw` è basso, il LED viene spento, mentre il contatore viene reimpostato a 1, senza essere più incrementato finché `sw` non torna alto.

4.5 Lampeggio del LED[0] al doppio della frequenza azionato da switch

```
module blinkIFswitch (
    input CLK100MHZ,
    input sw,
    output reg [15:0] LED
);

reg [32:0] counter;

initial begin
    counter = 0;
    LED = 0;
end

always @(posedge CLK100MHZ) begin
    if (sw) begin
        LED[0] <= counter[23];
        counter <= counter + 1
    end else begin
```

```
19          LED[0] <= counter[22]; # raddoppio se aperto
20          counter <= counter + 1
21          end
22      end
23  endmodule
```

Ad ogni ciclo di clock, quando il primo switch è attivo, il contatore viene incrementato di 1 e il valore del bit 24 del contatore viene assegnato al primo LED. Questo bit cambia ogni 16.777.216 cicli di clock, il che consente al LED di lampeggiare con una frequenza di 2 Hz. Quando invece `sw` è disattivo, il contatore continua a incrementarsi, ma al `LED[0]` viene assegnato il bit 23 del contatore. In questo caso, la frequenza di lampeggio del LED viene dimezzata rispetto alla precedente, poiché il bit `counter[22]` cambia valore tra 0 e 1 due volte più frequentemente rispetto al bit `counter[23]`.

4.6 Contatore binario

```
1   module counterBinaryLED (
2       input CLK100MHZ
3       output [15:0] LED
4   );
5
6   reg [32:0] counter;
7   integer i;
8
9   initial begin
10      counter = 0;
11      LED = 0;
12  end
13
14  always @(posedge CLK100MHZ) begin
15      for (i = 0; i<16; i = i+1) begin
16          LED[i] < counter [22-i];
17          counter <= counter + 1;
18          end
19      end
20  endmodule
```

Il codice presentato implementa un contatore binario all'interno dell'FPGA, il quale incrementa di un bit il numero visualizzato sulla board ad ogni ciclo di clock. Il clock viene dichiarato come ingresso, mentre il registro a 16 bit, utilizzato per il controllo dei LED, è definito come uscita. Vengono dichiarati, inoltre, un contatore a 33 bit e una variabile di tipo `integer` utilizzata come indice all'interno del ciclo `for`. Sia i LED che il contatore vengono inizializzati a zero. Ad ogni ciclo di clock, utilizzando il ciclo `for (i = 0; i < 16; i = i + 1)`, il valore di `LED[i]` viene impostato al corrispondente bit del contatore, con `LED[i] <= counter[22 - i]`. I 16 bit del registro `LED` vengono quindi popolati con i bit da `counter[22]` a `counter[7]`, dato che `counter` è un registro a 33 bit e l'indice `22 - i` varia da 22 a 7. In tal modo, i LED si accenderanno e spegneranno in base ai valori dei bit del contatore. Poiché il contatore incrementa ad ogni ciclo di clock, i LED cambieranno di stato in modo sequenziale, producendo un contatore binario visibile sui LED stessi.

4.7 Interruttore per il LED[0]

```
1   module LEDcomand (
2       input CLK100MHZ,
3       input btnC,
4       output reg[15:0] LED
5   );
6   reg obtnC, onoff;
```

```
7  initial begin
8      LED = 16'b0000_0000_0000_0001;
9      obtnC <= btnC;
10     onoff = 0;
11 end
12
13 always @(posedge CLK100MHZ) begin
14     if(!btnC & obtnC) begin
15         LED[0] = onoff;
16     end
17 end
18 endmodule
```

Nel presente codice vengono dichiarati come ingressi il clock `CLK100MHZ` e il pulsante centrale del Joystick `btnC`, mentre come uscita viene definito il registro a 16 bit `LED`, utilizzato per il controllo dei LED. Tra le variabili interne, `obtnC` è un registro che memorizza lo stato del pulsante `btnC` al ciclo di clock precedente, mentre `onoff` è un altro registro, inizializzato a 0, che tiene traccia dello stato del LED.

Il registro `LED` viene inizializzato con il primo bit, `LED[0]`, impostato su 1, così da accendere il primo LED all'avvio. La riga di codice `obtnC <= btnC` assicura che al registro `obtnC` venga assegnato il valore del pulsante al ciclo precedente, creando uno sfasamento di un ciclo tra il valore di `obtnC` e `btnC`.

La variabile `onoff` viene inizializzata a 0. Il blocco `if (!btnC & obtnC)` verifica se il pulsante centrale del Joystick è stato premuto, controllando il fronte di discesa (cioè quando `btnC` passa da 1 a 0), sfruttando il valore precedente memorizzato in `obtnC`. Quando questa condizione è vera, ovvero quando il pulsante viene rilasciato, il valore di `LED[0]` viene impostato su `onoff`, che in quel momento è 0, spegnendo così il primo LED.

Se il pulsante viene premuto e rilasciato nuovamente, il valore di `onoff` viene invertito a 1, accendendo il LED al prossimo ciclo di clock. Questo comportamento consente di controllare l'accensione e lo spegnimento del primo LED attraverso il pulsante centrale del Joystick.

4.8 Comando del LED tramite il joystick

```
1  module LEDmanualePACMAN (
2      input CLK100MHZ,
3      input btnL, btnR,
4      output reg [15:0] LED
5  );
6  reg [3:0] indled;
7  reg obtnL, obtnR;
8
9  initial begin
10     indled = 0;
11     LED = 16'b0000_0000_0000_0001;
12     obtnL = 0;
13     obtnR = 0;
14 end
15
16 always @(posedge CLK100MHZ) begin
17     obtnL <= btnL;
18     obtnR <= btnR;
19
20     if (btnL && !obtnL) begin
21         indled <= (indled == 15) ? 0 : indled +1;
22     end else if (btnR && !obtnR) begin
23         indled <= (indled == 0) ? 15 : indled -1;
```

```
    end
    LED <= 16'b0 << indled;
    LED[indled] <= 1
end
endmodule
```

Nel codice presentato, vengono dichiarati come ingressi il clock `CLK100MHZ`, il pulsante sinistro `btnL` e il pulsante destro `btnR` del joystick. Come uscita viene utilizzato il registro a 16 bit `LED`, che controlla lo stato dei LED. Tra le variabili interne, `indled` è un registro a 4 bit che tiene traccia dell'indice del LED acceso, mentre `obtnL` e `obtnR` sono registri che memorizzano lo stato del pulsante sinistro e destro al ciclo di clock precedente, rispettivamente.

All'inizio, `indled` viene inizializzato a 0, in modo che il primo LED (`LED[0]`) sia acceso. Inoltre, il registro `LED` viene inizializzato con il valore `16'b0000_0000_0000_0001`, che accende il primo LED. I pulsanti `btnL` e `btnR` vengono inizializzati come non premuti.

Ad ogni ciclo di clock, lo stato precedente dei pulsanti `btnL` e `btnR` viene memorizzato nei registri `obtnL` e `obtnR` tramite le righe `obtnL <= btnL` e `obtnR <= btnR`. La condizione `if (btnL && !obtnL)` verifica se il pulsante sinistro è stato premuto, facendo attenzione a non considerare il fronte di salita dello stesso. In caso affermativo, viene aggiornato il valore di `indled` tramite l'operazione `indled <= (indled == 15) ? 0 : indled + 1`, che fa sì che se `indled` è uguale a 15, l'indice venga riportato a 0, mentre in caso contrario aumenta di 1, spostando il LED verso sinistra.

In modo simile, la condizione `else if (btnR && !obtnR)` verifica se il pulsante destro è stato premuto. Se questa condizione è vera, il valore di `indled` viene aggiornato, spostando il LED verso destra.

La riga `LED <= 16'b0 << indled` azzera il registro `LED`, preparandolo per l'aggiornamento, mentre `LED[indled] <= 1` accende il LED corrispondente alla posizione indicata da `indled`. In questo modo, ogni volta che uno dei pulsanti viene premuto, il LED si sposterà sulla posizione corrispondente all'indice `indled`, permettendo di controllare il movimento del LED attraverso il joystick.

4.9 Spostamento automatico della luce sulla board

```
module LEDauto (
    input CLK100MHZ,
    input btnL,
    input btnR,
    output reg[15:0] LED
);
reg [3:0] indled;
reg dir;
reg [24:0] count;
reg sposta;

initial begin
    indled = 4'b0000;
    LED = 16'b0000_0000_0000_0001;
    dir = 0;
    count = 0;
    sposta = 0;
end

always @(posedge CLK100MHZ) begin
    count = 0;
    sposta = 1;

    if (count[23] == 1) begin
```

```
25          count = 0;
26          sposta = 1;
27      end else begin
28          sposta = 0;
29      end
30
31      if(sposta) begin
32          if (dir == 0) begin
33              if (indled < 15) begin
34                  indled = indled + 1;
35              end else begin
36                  dir = 1;
37              end
38          end else begin
39              if (indled > 0) begin
40                  indled = indled - 1;
41              end else begin
42                  dir = 0;
43              end
44          end
45      LED = (1 << indled);
46      end
47  end
48  endmodule
```

Il codice implementa un movimento automatico del LED, dove la direzione del movimento viene gestita attraverso il controllo di variabili interne. Le variabili in ingresso e in uscita sono simili a quelle del programma precedente, con l'aggiunta di un contatore per gestire il ritardo del movimento.

Le variabili interne dichiarate sono:

- `indled`, un registro a 4 bit che memorizza l'indice del LED attualmente acceso,
- `dir`, una variabile che determina la direzione del movimento del LED. Se `dir` è 0, il LED si sposterà verso destra, mentre se `dir` è 1, il LED si sposterà verso sinistra,
- `count`, un contatore a 25 bit utilizzato per generare un ritardo e regolare la velocità di spostamento del LED,
- `sposta`, una variabile che indica quando il LED deve effettivamente spostarsi.

All'inizio, `indled` viene inizializzato a 4 bit tutti a 0, in modo che il primo LED sia acceso. Il registro `LED` viene inizializzato con `16'b0000_0000_0000_0001`, accendendo il primo LED. Inoltre, la variabile `dir` è inizializzata a 0, impostando la direzione del movimento iniziale verso destra. Il contatore `count` e la variabile `sposta` sono anch'essi inizializzati a 0, indicando che il LED è fermo all'inizio.

Ad ogni ciclo di clock, il contatore viene azzerato e `sposta` viene impostato a 1, segnalando che il LED deve spostarsi. La condizione `if (count[23] == 1)` verifica se il bit 23 del contatore è 1. Se questa condizione è soddisfatta, il contatore viene azzerato e `sposta` viene reimpostato a 1, consentendo al LED di spostarsi. Se invece il bit 23 del contatore non è 1, `sposta` viene impostato a 0, impedendo lo spostamento del LED.

Il movimento del LED è gestito attraverso la variabile `indled`. Se `indled` è minore di 15, il LED si sposterà verso destra, incrementando l'indice. Se invece `indled` raggiunge il valore 15, la direzione (`dir`) viene invertita a 1, facendo muovere il LED verso sinistra. Quando `indled` è uguale a 0, la direzione viene nuovamente invertita verso destra.

Infine, la riga `LED = (1 << indled)` viene utilizzata per aggiornare il registro `LED` in modo da accendere il LED corrispondente all'indice `indled`. Questo avviene spostando il bit 1 nella posizione corrispondente all'indice del LED da accendere, realizzando così il movimento ciclico del LED sulla board.

4.10 Calcolatrice binaria

```
module calcolatrice(
    input CLK100MHZ,
    input wire [15:0] sw,
    input btnL, btnU, btnR, btnD, btnC,
    output reg[15:0] LED
);

reg [15:0] n1,n2;
reg [15:0] Cb, Lb, Rb, Ub, Db;
reg press;

initial begin
    n1=0, n2=0, cB=0, Lb=0; Rb=0, Ub=0, Db=0, press=0, LED=0;
end

always @(posedge CLK100MHZ) begin
    Lb <= btnL;
    Rb <= btnR;
    Ub <= btnU;
    Db <= btnD;
    Cb <= btnC;

    if ( (btnC == 0) && (Cb == 1) && (press == 0) ) begin
        n1 <= sw;
        LED <= sw;
        press <= 1;
    end

    if ( (btnC == 0) && (Cb == 1 ) && (press == 1)) begin
        n2 <= sw;
        LED <= sw;
        press <= 0;
    end

    if ( (btnL == 0) && (Lb == 1) ) begin
        LED <= n1 + n2;
    end

    if ( (btnR == 0) && (Lb == 1) ) begin
        LED <= n1 - n2;
    end

    if ( (btnD == 0) && (Db == 1) ) begin
        LED <= n1 * n2;
    end

    if ( (btnU == 0) && (Ub == 1) ) begin
        if (n2 != 0) begin
            LED <= n1 / n2;
        end else begin
            LED <= 16'hFFFF;
            press <=0;
        end

    end
end
```

```
57 endmodule
```

Il modulo implementa una calcolatrice binaria che esegue le quattro operazioni aritmetiche di base (somma, sottrazione, moltiplicazione e divisione) su numeri binari a 16 bit. I segnali in ingresso comprendono il clock `CLK100MHZ`, un array di 16 switch `sw` che consente di immettere un numero binario, e cinque pulsanti del Joystick: `btnL`, `btnU`, `btnR`, `btnD` e `btnC`. I primi quattro pulsanti sono utilizzati per selezionare l'operazione aritmetica da eseguire, rispettivamente somma, sottrazione, moltiplicazione e divisione. Il pulsante `btnC` serve invece per immettere i numeri. L'uscita del modulo è rappresentata da un registro a 16 bit, `LED`, che visualizza il risultato dell'operazione.

Il modulo utilizza i registri a 16 bit `n1` e `n2` per memorizzare i due numeri da operare. Le variabili `Cb`, `Lb`, `Rb`, `Ub` e `Db` sono utilizzate per memorizzare lo stato precedente dei pulsanti del Joystick, al fine di rilevare i cambiamenti nello stato di ciascun pulsante. La variabile `press` tiene traccia se il primo numero è già stato immesso o se si sta raccogliendo il secondo numero. Tutte le variabili sono inizializzate a zero.

Ad ogni ciclo di clock, il modulo aggiorna le variabili `Lb`, `Rb`, `Ub`, `Db`, `Cb` con lo stato dei rispettivi pulsanti. Quando `btnC` viene premuto per la prima volta (condizione `btnC == 0 e Cb == 1`), il numero immesso tramite gli switch `sw` viene memorizzato in `n1` e visualizzato sui LED. In questo momento, la variabile `press` viene impostata a 1 per indicare che il primo numero è stato registrato. Se `btnC` viene premuto nuovamente, il secondo numero `n2` viene immesso allo stesso modo e memorizzato in `n2`. Dopo questa operazione, la variabile `press` viene azzerata, consentendo il passaggio alla fase del calcolo.

Le operazioni aritmetiche sono eseguite in base ai pulsanti premuti. Se `btnL` è premuto, viene eseguita l'operazione di somma tra `n1` e `n2`. Se `btnR` è premuto, viene eseguita la sottrazione tra i due numeri. Se `btnD` è premuto, viene eseguita la moltiplicazione tra `n1` e `n2`. Infine, se `btnU` è premuto, viene eseguita la divisione tra `n1` e `n2`, ma solo se `n2` è diverso da zero. In caso contrario, viene visualizzato un valore di errore (`16'hFFFF`) sui LED, e la variabile `press` viene reimpostata a zero.

In questo modo, il modulo consente di eseguire operazioni aritmetiche di base su numeri binari, con il risultato visualizzato sui LED.

5 Implementazione FPGA nelle altre esperienze

Le FPGA rappresentano una soluzione estremamente versatile nell'ambito della progettazione digitale. La loro flessibilità le rende adatte per una vasta gamma di applicazioni, quali l'ottimizzazione di circuiti per aumentarne la velocità, ridurne le dimensioni, o anche per emulare comportamenti digitali specifici di un determinato sistema.

Un aspetto cruciale nell'utilizzo delle FPGA è la corretta gestione delle tensioni di ingresso sui pin della scheda. I pin delle FPGA sono generalmente progettati per tollerare una tensione massima di ingresso pari a 3.3 V. Superare tale limite può danneggiare i transistor MOS (la maggior parte delle FPGA moderne utilizzano logiche CMOS) alla base della logica programmabile. È quindi fondamentale adottare adeguate precauzioni per garantire l'integrità della scheda.

Nel contesto degli esperimenti descritti (ad esempio, nella seconda relazione, dove si lavora con circuiti esterni che potrebbero generare tensioni superiori a 3.3 V), è necessario regolare la tensione di alimentazione del circuito esterno (V_{cc}^{+}) per rientrare nei limiti tollerabili dall'FPGA. Tuttavia, una soluzione più generale e robusta consiste nell'inserire resistenze in serie ai pin di ingresso. Questo accorgimento limita la corrente in ingresso, anche in caso di superamento accidentale della soglia di tensione, proteggendo i componenti interni della scheda.

Un ulteriore livello di protezione potrebbe essere garantito dall'uso di divisori di tensione o di circuiti clamping (portano il segnale a una tensione di riferimento predefinita) per assicurare che la tensione ai pin dell'FPGA non superi mai il valore massimo consentito.

Infine, va ricordato che i pin delle FPGA possono essere configurati come ingressi, uscite o ingressi/uscite bidirezionali (*tri-state*), e in molti casi supportano standard di tensione selezionabili tramite configurazioni software.

5.1 Circuito invertente e integratore.

Nello studio dei circuiti trattati, l'FPGA può essere utilizzata per elaborare i dati in uscita dall'amplificatore operazionale. Grazie alle due linee di ingresso collegate a un sistema di conversione analogico-digitale interno, è possibile acquisire e memorizzare i valori delle tensioni in uscita. Implementando in Verilog la divisione tra le due ampiezze, si può calcolare il guadagno e visualizzarlo su uno schermo con 4 cifre significative. Inoltre, sfruttando un approccio simile a quello descritto nei punti **4.3** e **4.4**, l'FPGA può essere configurata come generatore di funzione, ad esempio per produrre un'onda quadra a una frequenza definita. Questo sistema, utile per analizzare i comportamenti qualitativi del derivatore e dell'integratore, è semplice da implementare: basta assegnare uno qualsiasi dei pin come `output`, eventualmente collegandolo a un oscilloscopio, tenendo conto dell'alta impedenza dello stesso, la quale garantisce che il flusso di corrente nel circuito sia trascurabile.

5.2 Flash ADC

Nel circuito con la rete di comparatori progettata per digitalizzare il segnale del sensore di temperatura, la board FPGA può essere impiegata per visualizzare il superamento delle soglie tramite i LED integrati. Le uscite dei comparatori vengono collegate direttamente ai pin dell'FPGA, in modo che i primi quattro LED rappresentino lo stato logico delle rispettive uscite. Inoltre, utilizzando un approccio analogo a quello descritto nella sezione **5.1**, è possibile visualizzare la temperatura rilevata, corrispondente alla tensione generata dall'LM35, sul display a sette segmenti integrato, con una precisione fino a quattro cifre significative.

5.3 Periodo del pendolo semplice

Nell'esperimento volto al calcolo della g a partire dal periodo di un pendolo semplice, si può collegare il fotodiodo direttamente all'FPGA tramite i pin connessi all'ADC. In questo modo, il segnale del sensore è digitalizzato in due livelli logici, distinguendo la presenza o assenza della massa davanti al sensore. Contando i cicli di clock tra due transizioni a `0` e conoscendo la frequenza della board, si determina il periodo in secondi e, con le formule usuali, si calcola g.

6 Conclusioni

L'esperimento condotto ha confermato il corretto funzionamento della scheda FPGA, evidenziando la notevole versatilità del dispositivo. La FPGA si è rivelata particolarmente utile in una varietà di applicazioni logiche complesse, dimostrando la sua capacità di gestire segnali e implementare circuiti digitali avanzati. La possibilità di realizzare funzioni diverse in un singolo chip hardware, senza la necessità di un processore centrale, rappresenta una delle caratteristiche distintive di questa tecnologia. Questo approccio offre vantaggi in termini di velocità e personalizzazione dei circuiti, che risulta difficile da ottenere con soluzioni tradizionali basate su software.

Nonostante la differenza tra la programmazione software e quella hardware, i risultati ottenuti sono stati soddisfacenti. L'FPGA ha dimostrato una grande affidabilità nell'eseguire le operazioni logiche richieste, interfacciandosi correttamente con gli altri componenti del sistema. L'esperimento ha permesso di comprendere meglio le potenzialità offerte dalla programmazione hardware e ha fornito una solida base di esperienza pratica, contribuendo a un'adeguata comprensione dei concetti teorici e delle tecniche applicative legate alla progettazione di circuiti digitali con FPGA.