Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Report di Laboratorio – Tecniche di Acquisizione Dati

“FPGA e «KITT»”

Studenti

Filippo Notari
Nicolò Tittarelli
Leonardo Nicoletta

Anno Accademico 202*3-2024*

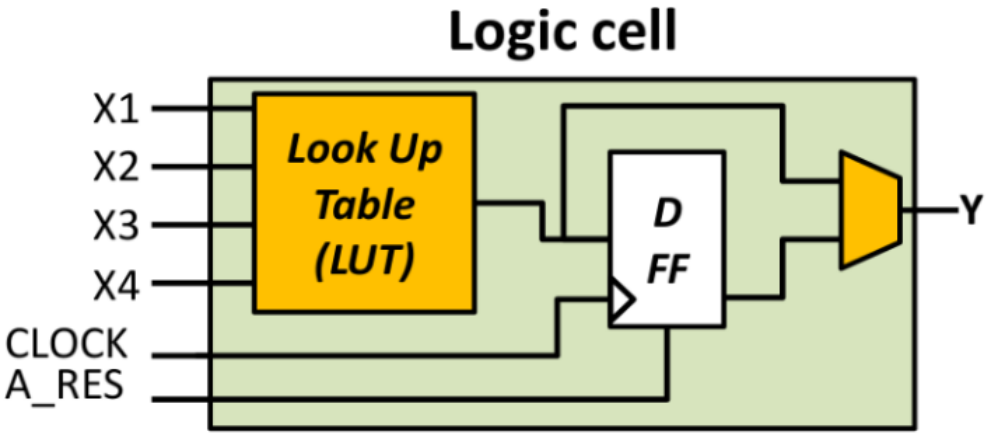
Sommario

Introduzione

Questa relazione ha l'obiettivo di esplorare alcuni utilizzi delle FPGA (Field Programmable Gate Array), analizzandone i vantaggi e gli svantaggi. Le FPGA sono dispositivi elettronici che contengono componenti logici e connessioni programmabili, permettendo la riconfigurazione dei circuiti interni tramite una stringa di bit di configurazione (bitstream). Questa caratteristica consente di implementare vari dispositivi logici, sia combinatori, come le porte logiche AND e OR, sia sequenziali, come i flip-flop e altri circuiti complessi.

Struttura di una FPGA

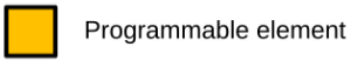
Una FPGA è costituita da un insieme di Configurable Logic Block (CLB) interconnessi tra loro. La funzione dei singoli CLB e delle loro connessioni viene determinata dal progettista tramite la programmazione. Alla base di ogni CLB c'è la rete logica, chiamata Logic Cell (LC).



Il Look Up Table (LUT) è una rete combinatoria programmabile che può essere riconfigurata per agire come uno shift register o una memoria (RAM distribuita).

Look-Up Table

X1	X2	X3	X4	Y
0	0	0	1	?
0	0	1	0	?
...	...	...	...	?
1	1	1	1	?



I valori delle tabelle di ricerca, dei mux e delle interconnessioni sono contenuti in un file chiamato bitstream, utilizzato per programmare specificamente una FPGA. La programmazione delle FPGA avviene solitamente tramite linguaggi HDL (Hardware Description Language), tra i quali i più comuni sono Verilog e VHDL. Il codice HDL serve per descrivere un circuito e può essere impiegato in due principali flussi di lavoro:

- Flusso di lavoro di programmazione: il codice viene utilizzato per programmare un dispositivo FPGA reale e renderlo operativo.
- Flusso di lavoro di simulazione e verifica: il codice viene usato per simulare il circuito tramite software, permettendo la verifica e il debug del progetto.

Clock

Le FPGA sono solitamente configurate per reti sincrone, il che richiede almeno un segnale di clock per le Reti Sequenziali Sincrone (RSS) che compongono il progetto sulla FPGA. Spesso, nei progetti, possono esserci più domini di clock, ossia diversi moduli logici all'interno della FPGA che utilizzano clock differenti.
Ogni ciclo di clock assicura che tutte le operazioni previste nel programma vengano eseguite correttamente.

Moduli

I moduli Verilog sono blocchi di codice che rappresentano circuiti e possono essere riutilizzati all'interno di altri blocchi, analogamente alle funzioni nei linguaggi di programmazione. Un blocco inizia con la parola chiave "module" e termina con "endmodule". I moduli possono avere parametri di ingresso e uscita.
Un modulo specifico, chiamato modulo principale, funge da punto di ingresso del "programma", simile al main nei linguaggi di programmazione. Solitamente, questo modulo ha ingressi e uscite collegati agli I/O fisici dell'FPGA.
Nel nostro caso, utilizzeremo il modulo "Basys 3 Master", che ci permetterà di interagire con l'FPGA che utilizzeremo. Questo modulo fornisce accesso alle funzionalità implementate nella nostra macchina, come LED, switch, pulsanti e altri componenti.

Capitolo 1. Accensione Led

1.1 Descrizione

L’obbiettivo dell’esercitazione è quello di riuscire ad accendere un led nella scheda alla pressione del tasto centrale presente nella scheda.

1.2 Svolgimento

Per poter svolgere questa esercitazione abbiamo avuto bisogno di usare il file “Basys3 master.xdc” il quale ci permette di usare le varie periferiche I/O della scheda.

Ciò di cui avremo bisogno per creare il programma sono: variabile input per il clock, variabile input per il pulsante che useremo, variabile output per il led che useremo e due registri per poter salvare lo stato del led e dell’ultima posizione del pulsante.

Quindi la prima cosa che inseriremo nel modulo sono le nostre variabili di input output:

```
      input clk,
      input btnC,
      output reg[1:0] led
```

Poi abbiamo bisogno dei due registri:

```
      reg pressed_last_cycle;
      reg led_state;
```

Ora che abbiamo tutte le variabili possiamo creare il nostro programma.

Bisogna controllare che lo stato del pulsante sia diverso dallo stato dell’ultimo clock:

```
if (btnC & (~ pressed_last_cycle)) begin
```

Quindi verifichiamo se il pulsante btnC è attualmente premuto (btnC è alto) e se non era premuto nel ciclo di clock precedente (~pressed_last_cycle è alto). L'operatore & esegue un'AND bit a bit tra btnC e il negato (~) di pressed_last_cycle.

Quando la condizione si avvera possiamo cambiare lo stato del led:

```
            led_state <= ~led_state;
            led[0] <= led_state;
```

Nella prima riga impostiamo lo stato di led_state al suo inverso; quindi, se era altro diventa basso e viceversa.

Nella seconda riga impostiamo quel valore al led.

Fuori dal controllo if bisogna modificare il registro dello stato del bottone:

```
pressed_last_cycle <= btnC;
```

1.3 Codice

```
module led_button(
    input clk,
    input btnC,
    output reg[1:0] led
    );
    reg pressed_last_cycle;
    reg led_state;

    always @ (posedge clk) begin
```

```
        if (btnC & (~ pressed_last_cycle)) begin
            led_state <= ~led_state;
            led[0] <= led_state;
        end
        pressed_last_cycle <= btnC;
    end

endmodule
```

1.4 Conclusioni

Il modulo led_button descritto in questo rapporto implementa una logica di controllo per il cambio di stato di un LED tramite la pressione di un pulsante. Utilizzando un clock (clk) come segnale di sincronizzazione, il codice Verilog rileva il bordo di salita del segnale del pulsante (btnC), garantendo che il LED cambi stato solo una volta per ogni pressione completa del pulsante.
La variabile pressed_last_cycle svolge un ruolo cruciale nella rilevazione del bordo di salita, memorizzando lo stato del pulsante del ciclo di clock precedente. Quando il pulsante viene premuto, led_state viene invertito, e il nuovo stato del LED viene aggiornato di conseguenza.
Questa implementazione dimostra un metodo semplice ma efficace per gestire l'interazione utente-dispositivo tramite un pulsante, garantendo un comportamento stabile e prevedibile del LED.

Capitolo 2. Led Lampeggiante

2.1 Led lampeggiante 2Hz

2.1.1 Descrizione

L’obbiettivo di questa esercitazione è quello di creare un codice che permette di far accendere e spengere un led in automatico ad una frequenza di 2Hz.

2.1.2 Svolgimento

Per poter svolgere questa esercitazione abbiamo avuto bisogno di usare il file “Basys3 master.xdc” il quale ci permette di usare le varie periferiche I/O della scheda.

Ciò di cui avremo bisogno per creare il programma sono: variabile input per il clock, variabile output per il led che useremo e un registro per gestire la frequenza.
Iniziamo con gli Input/Output:

```
    input clk,
    output reg [1:0] led
```

Dopodiché il registro:

```
    reg [25:0] counter;
```

Poi impostiamo che solo al primo ciclo di clock il counter va a zero:

```
    initial
        counter = 0;
```

Adesso accendiamo o spengiamo il led in base ad una frequenza di ~2Hz:

```
        led[0] <= counter[25];
```

Per fare ciò bisogna calcolare ogni quanti cicli eseguire il codice. Sapendo che la scheda gira a 50Mhz, quindi 50 milioni di cicli al secondo, noi voglia che lampeggi a ~2Hz, quindi 2 volte al secondo, dividendo 50 milioni per 2Hz abbiamo come risultato il numero di cicli da eseguire prima di far accendere o spegnere il led.

Sapendo questo se facciamo che è il numero più vicino a 25000000 che riusciamo ad ottenere, ce lo facciamo andar bene.
Quindi in questo caso il led lampeggerà più o meno ogni 0.6 secondi.
Ora non ci resta che incrementare il contatore ad ogni ciclo:

```
        counter <= counter + 1;
```

2.1.3 Codice

```
module led_blink_2hz(
    input clk,    output reg [1:0] led
    );

    reg [25:0] counter;

    initial
        counter = 0;
    always @ (posedge clk) begin
        led[0] <= counter[25];
        counter <= counter + 1;
    end
endmodule
```

2.1.4 Conclusione

In conclusione, il modulo led_blink_2hz implementa un semplice lampeggiamento di un LED utilizzando un contatore e un segnale di clock. Il contatore viene incrementato ad ogni fronte di salita del segnale di clock e il bit meno significativo del contatore controlla lo stato del LED. Questo approccio fornisce un esempio di progettazione sequenziale in Verilog, dove un comportamento desiderato viene ottenuto attraverso l'interazione di segnali e registri sincronizzati con il clock di sistema.

2.2 Led lampeggiante 2Hz con switch On

2.2.1 Descrizione

L'obbiettivo di questa esercitazione è quello di creare un codice che permette di far accendere e spengere un led in automatico ad una frequenza di 2Hz quando uno switch è su On.

2.2.2 Svolgimento

Per poter svolgere questa esercitazione abbiamo avuto bisogno di usare il file "Basys3 master.xdc" il quale ci permette di usare le varie periferiche I/O della scheda.
Ciò di cui avremo bisogno per creare il programma sono: variabile input per il clock, variabile input per lo switch, variabile output per il led che useremo e un registro per gestire la frequenza.
Iniziamo con gli Input/Output:

```
    input clk,
    input wire [1:0] sw,
    output reg [1:0] led
```

Dopodiché il registro:

```
    reg [25:0] counter;
```

Poi impostiamo che solo al primo ciclo di clock il counter va a zero:

```
    initial
        counter = 0;
```

Adesso accendiamo o spengiamo il led in base ad una frequenza di ~2Hz:

```
        led[0] <= sw[0] & counter[25];
```

Per fare ciò bisogna calcolare ogni quanti cicli eseguire il codice. Sapendo che la scheda gira a 50Mhz, quindi 50 milioni di cicli al secondo, noi voglia che lampeggi a ~2Hz, quindi 2 volte al secondo, dividendo 50 milioni per 2Hz abbiamo come risultato il numero di cicli da eseguire prima di far accendere o spegnere il led.

Sapendo questo se facciamo che è il numero più vicino a 25000000 che riusciamo ad ottenere, ce lo facciamo andar bene.

Quindi in questo caso il led lampeggerà più o meno ogni 0.6 secondi soltanto quando lo switch è On.

Ora non ci resta che incrementare il contatore ad ogni ciclo:

```
        counter <= counter + 1;
```

2.2.3 Codice

```
module led_blink_2hz_switch(
    input clk,
    input wire [1:0] sw,
    output reg [1:0] led
    );

    reg [25:0] counter;

    initial
        counter = 0;
    always @ (posedge clk) begin
        led[0] <= sw[0] & counter[25];
        counter <= counter + 1;
    end
endmodule
```

2.2.4 Conclusione

In conclusione, il modulo led_blink_2hz_switch implementa un LED lampeggiante a 2 Hz controllato da un interruttore. Il LED lampeggia a una frequenza di 2 Hz quando lo switch è attivato. Il contatore viene utilizzato per generare la frequenza desiderata e il LED si accende o si spegne in base al valore dello switch e al bit più significativo del contatore. La logica di questo modulo consente un controllo semplice ed efficace del lampeggiamento del LED in base all'input dello switch, fornendo un'implementazione pratica e funzionale.

2.3 Led lampeggiante 2Hz con switch Off, altrimenti raddoppia la frequenza

2.3.1 Descrizione

L’obbiettivo di questa esercitazione è quello di creare un codice che permette di far accendere e spengere un led in automatico ad una frequenza di 2Hz quando uno switch è su Off, mentre quando è su On radoppia la frequenza.

2.3.2 Svolgimento

Per poter svolgere questa esercitazione abbiamo avuto bisogno di usare il file “Basys3 master.xdc” il quale ci permette di usare le varie periferiche I/O della scheda.

Ciò di cui avremo bisogno per creare il programma sono: variabile input per il clock, variabile input per lo switch, variabile output per il led che useremo e un registro per gestire la frequenza.

Iniziamo con gli Input/Output:

```
    input clk,
    input wire [1:0] sw,
    output reg [1:0] led
```

Dopodiché il registro:

```
    reg [25:0] counter;
```

Poi impostiamo che solo al primo ciclo di clock il counter va a zero:

```
    initial
        counter = 0;
```

Adesso accendiamo o spengiamo il led in base ad una frequenza di ~2Hz:

```
        led[0] <= counter[25];
```

Per fare ciò bisogna calcolare ogni quanti cicli eseguire il codice. Sapendo che la scheda gira a 50Mhz, quindi 50 milioni di cicli al secondo, noi voglia che lampeggi a ~2Hz, quindi 2 volte al secondo, dividendo 50 milioni per 2Hz abbiamo come risultato il numero di cicli da eseguire prima di far accendere o spegnere il led.

Sapendo questo se facciamo che è il numero più vicino a 25000000 che riusciamo ad ottenere, ce lo facciamo andar bene.

Quindi in questo caso il led lampeggerà più o meno ogni 0.6 secondi soltanto quando lo switch è On.

Ora non ci resta che incrementare il contatore ad ogni ciclo:

```
        if(sw[0]==0)begin
            counter <= counter + 1;
        end else begin
            counter <= counter + 2;
        end
```

Il contatore aumenterà di uno se lo switch è su Off, mentre aumenterà di 2, quindi raddoppia, quando è su On.

2.3.3 Codice

```
module led_blink_2hz_switch_double(
    input clk,
    input wire [1:0] sw,
    output reg [1:0] led
    );
```

```
    reg [25:0] counter;

    initial
        counter = 0;
    always @ (posedge clk) begin
        led[0] <= counter[25];

        if(sw[0]==0)begin
            counter <= counter + 1;
        end else begin
            counter <= counter + 2;
        end
    end
endmodule
```

2.3.4 Conclusione

In conclusione, il modulo `led_blink_2hz_switch_double` implementa un sistema di controllo per un LED lampeggiante con una frequenza variabile in base allo stato di uno switch. Quando lo switch è spento, il LED lampeggia ad una frequenza di 2 Hz, mentre quando è acceso, la frequenza del lampeggio raddoppi. Ciò è ottenuto utilizzando un contatore a 26 bit che viene incrementato di uno o due ad ogni fronte di salita del clock, a seconda dello stato dello switch. Il bit più significativo del contatore viene utilizzato per controllare lo stato del LED, determinando se lampeggi a una frequenza standard o raddoppiata.

Capitolo 3. Contatore Binario

3.1 Descrizione

L'obiettivo di questa esercitazione è implementare un contatore binario per la scheda basys3 che mostri il numero attraverso i led. Come per le precedenti esercitazioni è stato utilizzato il file "Basys3 master.xdc" il quale ci permette di usare le varie periferiche I/O della scheda.

3.2 Inizializzazione variabili

Le variabili utilizzate sono:

- ‘clk’: il clock
- ‘led’: i led della scheda basys3
- ‘counter’: il registro che conterrà il numero di cicli di clock, inizializzato a 0
- ‘i’: un intero

```
input clk,
output reg[15:0] led
);
reg[38:0] counter;
integer i;
initial
counter = 0;
```

3.3 Costruzione contatore binario

Nel seguente codice ad ogni ciclo di clock viene incrementata la variabile counter, ma solo quando counter assume valore maggiore di 2^{23} avverrà l’accensione del primo led.
Scegliendo 2^{23} come valore soglia il primo led si accenderà dopo 0.1677 secondi (il tempo che impiegano 2^{23} cicli di clock a frequenza 50 Mhz).
Quando counter raggiunge i valori 2^{24} si illimina il secondo led, 2^{25} si illumina il terzo led e così via.

```
always @ (posedge clk) begin
        for(i = 0; i < 16; i = i + 1) begin
                led[i] <= counter[23 + i];
        end
        counter <= counter + 1;
End
```

3.4 Conclusioni

Il codice presentato implementa un contatore binario che sfrutta il clock della scheda Basys3 per gestire l'accensione sequenziale dei LED. Ad ogni ciclo di clock, il contatore viene incrementato. Quando il contatore supera determinate soglie (2^{23}, 2^{24}, 2^{25}, ecc.), i LED corrispondenti si accendono.

Capitolo 4. Joystick

4.1 Descrizione

Lo scopo dell'esercitazione è creare un joystick in modo che:

- La luce si sposti a sinistra alla pressione del pulsante sinistro del joystick
- La luce si sposti a destra alla pressione del pulsante destro del joystick
- In entrambi i casi, fare in modo che lo spostamento si fermi al raggiungimento del LED più esterno

Come per le precedenti esercitazioni è stato utilizzato il file "Basys3 master.xdc" il quale ci permette di usare le varie periferiche I/O della scheda.

4.2 Inizializzazione variabili

Le variabili utilizzate sono:

- 'clk': il clock
- 'btnL': il bottone sinistro della scheda FPGA
- 'btnR': il bottone destro della scheda FPGA
- 'led': i led della scheda
- 'pressed_last_cycleL': registro che tiene traccia della pressione del bottone sinistro nel ciclo precedente
- 'pressed_last_cycleR': registro che tiene traccia della pressione del bottone destro nel ciclo precedente
- 'arrayled': registro su cui avvengono le modifiche che poi verranno assegnate a registro di output 'led'

```
input clk,
input btnL,
input btnR,
output reg[15:0] led
);
reg pressed_last_cycleL;
reg pressed_last_cycleR;
reg[15:0] arrayled = 16'b0000000010000000;
initial
led<=arrayled;
```

4.3 Codice e logica

In questo if-statement viene eseguita un'operazione di left shift solo se 'btnL' in questo ciclo di clock ha valore 1 e nel ciclo precedente aveva valore 0. Inoltre, se il led acceso è l'ultimo a sinistra lo shift non avviene. Dopodiché si aggiorna il valore di 'pressed_last_cycleL' con il nuovo valore di 'btnL'.

```
if (btnL & (~ pressed_last_cycleL) & ~arrayled[15]) begin
```

```
arrayled <= arrayled << 1;
end
pressed_last_cycleL <= btnL;
```

La stessa logica viene implementata per il bottone destro.

```
if (btnR & (~ pressed_last_cycleR) & ~arrayled[0]) begin
arrayled <= arrayled >> 1;
end
pressed_last_cycleR <= btnR;
```

Infine, il registro ‘arrayled’ viene assegnato al registro di output led.

```
led<=arrayled;
```

4.4 Conclusioni

Capitolo 5. KITT

5.1 Descrizione

L’obbiettivo di questa esercitazione è quello di creare un codice che permette di far accendere e spengere una serie di led da destra verso sinistra, rimbalzando ai lati, dando l’effetto come nella macchina “KITT” dalla serie TV “Knight Rider”.

5.2 Svolgimento

Per poter svolgere questa esercitazione abbiamo avuto bisogno di usare il file “Basys3 master.xdc” il quale ci permette di usare le varie periferiche I/O della scheda.
Ciò di cui avremo bisogno per creare il programma sono: variabile input per il clock, variabile output per i led che useremo, un registro per dire se li accendiamo da destra verso sinistra o viceversa, un registro di 16 bit che

memorizza lo stato corrente delle luci LED, un contatore per rallentare l'esecuzione del programma e un registro che memorizza il valore del bit 25 del contatore nell'ultimo ciclo di clock.

Quindi la prima cosa che inseriremo nel modulo sono le nostre variabili di input output:

```
    input clk,
    output reg[15:0] led
```

Dopodiche le variabili interne:

```
    reg going_right = 1;
    reg[15:0] arrayled = 16'b1000000000000000;
    reg[31:0] counter = 0;
    reg value_last_frame = 0;
```

Ora che abbiamo le nostre variabili bisogna accendere il primo array al primo ciclo di clock:

```
    initial
        led<=arrayled;
```

Poi all'interno di always iniziamo con l'incrementare il contatore:

```
        counter <= counter + 1;
```

Controllando se il contatore è diverso dall'ultimo ciclo iniziamo ad accendere i led in una direzione:

```
        if (counter[25] != value_last_frame) begin
            if (going_right) begin
                if (arrayled[0]) begin
                    going_right <= 0;
                end else begin
                    arrayled <= arrayled >> 1;
                end
```

E poi continuiamo nell'altra direzione:

```
            end else begin
                if (arrayled[15]) begin
                    going_right <= 1;
                end else begin
                    arrayled <= arrayled << 1;
                end
            end
        end
```

Infine, aggiorniamo value_last_frame con il valore corrente del bit 25 del contatore e aggiorniamo l'accensione dei led con lo stato corrente di arrayled.

5.3 Codice

```
module kitt(
    input clk,
    output reg[15:0] led
    );
    reg going_right = 1;
    reg[15:0] arrayled = 16'b1000000000000000;
    reg[31:0] counter = 0;
    reg value_last_frame = 0;
```

```
    initial
        led<=arrayled;

    always @ (posedge clk) begin
        counter <= counter + 1;

        if (counter[25] != value_last_frame) begin
            if (going_right) begin
                if (arrayled[0]) begin
                    going_right <= 0;
                end else begin
                    arrayled <= arrayled >> 1;
                end
            end else begin
                if (arrayled[15]) begin
                    going_right <= 1;
                end else begin
                    arrayled <= arrayled << 1;
                end
            end
        end

        value_last_frame <= counter[25];
        led<=arrayled;
    end
endmodule
```

5.4 Conclusione

Il modulo “kitt” implementa effetto luminoso a scorrimento, ispirato alle luci del veicolo “KITT” dalla serie TV “Knight Rider”. Utilizzando un contatore per regolare la velocità del movimento e un semplice controllo della direzione, il codice è in grado di far oscillare un singolo bit acceso lungo una catena di 16 LED.

Capitolo 6. Calcolatrice

6.1 Descrizione

L'obbiettivo di questa esercitazione è quello di creare un codice che permette di creare uan calcotrice tramite l'uso di pulsanti, switch e led presenti sulla board. Ad ogni pulsante direzione gli viene assegnata un'operazione, tramite gli switch settiamo il primo numero in binario, premendo il pulsante centrale il numero viene salvato, impostare il secondo numero sempre tramite gli switch e infine usare un pulsante con assegnata un'operazione per visualizzare il risultato sui led, sempre in binario.

6.2 Svolgimento

Per poter svolgere questa esercitazione abbiamo avuto bisogno di usare il file "Basys3 master.xdc" il quale ci permette di usare le varie periferiche I/O della scheda.
Ciò di cui avremo bisogno per creare il programma sono: variabile input per il clock, variabile output per i led che useremo, variabili input per i pulsanti, variabile input per gli switch, variabili come contatore per cicli, registri per memorizzare lo stato precedente dei pulsanti, registro per salvare il numero corrente, registro per la frequenza e un registro che indica se c'è un errore.
Quindi la prima cosa che inseriremo nel modulo sono le nostre variabili di input output:

```
    input clk,
    input wire[15:0] sw,
    output  reg[15:0] led,
    input wire btnC,
    input wire btnR,
    input wire btnL,
    input wire btnU,
    input wire btnD
```

Poi le variabili interne:

```
    integer i=0;
    integer j=0;
    integer count=0;
    reg lastPressedC=0;
    reg lastPressedD=0;
    reg lastPressedU=0;
    reg lastPressedR=0;
    reg lastPressedL=0;
    reg[32:0] numberSave;
    reg[32:0] freq;
    reg error;
```

La prima cosa che facciamo all'interno di always è gestire l'errore, quindi se si presenta un errore questo codice fa lampeggiare tutti i led per dieci cicli di clock:

```
            if(error)begin
                if(freq[25] && count < 10)begin
                    for(j=0;j<16;j = j+1) begin
                        led[j]<=~led[j];
                    end
                    count<=count+1;
                end
                else if(count == 10)begin
                    error<=0;
                end
            end // end error
```

Se non ci sono errori iniziamo controllando se è stato premuto il pulsante centrale per salvare il numero immesso e lo salviamo:

```
            else begin
                if(lastPressedC==0 && btnC==1) begin
                    for(i=0;i<16;i = i+1) begin
                        numberSave[i]<=sw[i];
                    end

                end
                lastPressedC<= btnC;
```

Poi controlliamo la pressione dei pulsati delle operazioni, in base a quale viene premuto viene effettuata un'operazione diversa, però nel controllo del pulsato basso che fa la somma abbiamo agigunto un controll oepr vedere se la somma superava il limite di led massimo che possiamo utilizzare per visualizzare il risultato:

```
                if(lastPressedD==0 && btnD==1) begin

                    if(numberSave+sw < (2**16))begin
                        numberSave<=numberSave+sw;
                    end
                    else begin
                        count<=0;
                        error<=1;
                    end
                end
                lastPressedD<= btnD;
```

E poi tutti gli altri ma senza questo controllo:

```
                if(lastPressedU==0 && btnU==1) begin
                    numberSave<=numberSave-sw;
                end
                lastPressedU<= btnU;

                if(lastPressedL==0 && btnL==1) begin
                    numberSave<=numberSave*sw;
                end
                lastPressedL<= btnL;

                if(lastPressedR==0 && btnR==1) begin
```

```
                    numberSave<=numberSave/sw;
                end
                lastPressedR<= btnR;
```

Infine, aumentiamo la variabile freq, e poi accendiamo i led in base al risultato:

```
                freq<=freq+1;
                    for(i=0;i<16;i = i+1) begin
                        led[i]<=numberSave[i];
                    end
```

6.3 Codice

```
module calculator(
    input clk,
    input wire[15:0] sw,
    output  reg[15:0] led,
    input wire btnC,
    input wire btnR,
    input wire btnL,
    input wire btnU,
    input wire btnD
    );
    integer i=0;
    integer j=0;
    integer count=0;
    reg lastPressedC=0;
    reg lastPressedD=0;
    reg lastPressedU=0;
    reg lastPressedR=0;
    reg lastPressedL=0;
    reg[32:0] numberSave;
    reg[32:0] freq;
    reg error;

    always @ (posedge clk)
        begin

            if(error)begin
                if(freq[25] && count < 10)begin
                    for(j=0;j<16;j = j+1) begin
                        led[j]<=~led[j];
                    end
                    count<=count+1;
                end
                else if(count == 10)begin
                    error<=0;
                end
            end // end error
```

```
            else begin
                if(lastPressedC==0 && btnC==1) begin
                    for(i=0;i<16;i = i+1) begin
                        numberSave[i]<=sw[i];
                    end

                end
                lastPressedC<= btnC;

                if(lastPressedD==0 && btnD==1) begin

                    if(numberSave+sw < (2**16))begin
                        numberSave<=numberSave+sw;
                    end
                    else begin
                        count<=0;
                        error<=1;
                    end
                end
                lastPressedD<= btnD;

                if(lastPressedU==0 && btnU==1) begin
                    numberSave<=numberSave-sw;
                end
                lastPressedU<= btnU;

                if(lastPressedL==0 && btnL==1) begin
                    numberSave<=numberSave*sw;
                end
                lastPressedL<= btnL;

                if(lastPressedR==0 && btnR==1) begin
                    numberSave<=numberSave/sw;
                end
                lastPressedR<= btnR;


                freq<=freq+1;
                    for(i=0;i<16;i = i+1) begin
                        led[i]<=numberSave[i];
                    end
            end
    end
endmodule
```

6.4 Conclusione

Il modulo “calculator” rappresenta un'applicazione di calcolatrice digitale, progettata per eseguire operazioni aritmetiche di base utilizzando pulsanti e interruttori. Le operazioni supportate includono somma, sottrazione, moltiplicazione e divisione, attivate rispettivamente dai pulsanti btnD, btnU, btnL e btnR, utilizzando i valori forniti dagli interruttori (sw). Quando il risultato di un'operazione supera i limiti consentiti (16 bit), il modulo entra in uno stato di errore, segnalato dai LED che lampeggiano per un certo periodo di tempo. Questo meccanismo di segnalazione visiva aiuta a diagnosticare e correggere rapidamente gli errori operativi. Il pulsante btnC consente di salvare il valore corrente degli interruttori in un registro (numberSave), permettendo la memorizzazione temporanea dei dati per le operazioni successive. Utilizzando un segnale di clock (clk), il modulo assicura che tutte le operazioni e le transizioni di stato siano eseguite in modo sincrono, garantendo stabilità e prevedibilità nel comportamento del sistema. Gli stati e i risultati delle operazioni sono visualizzati tramite un array di 16 LED, fornendo un'interfaccia visiva immediata e chiara all'utente.

Conclusioni

In questo report abbiamo esplorato diverse tecniche di acquisizione e controllo di dati utilizzando FPGA, con particolare attenzione a esercizi pratici implementati sulla scheda Basys3. Abbiamo affrontato la progettazione e la realizzazione di vari moduli in Verilog, ognuno con un obiettivo specifico, dimostrando l'efficacia e la versatilità delle FPGA in applicazioni di controllo e gestione delle periferiche.

6.5 Sintesi dei Risultati

- Accensione LED tramite Pulsante: Abbiamo sviluppato un semplice modulo che permette l'accensione e lo spegnimento di un LED alla pressione di un pulsante.
- LED Lampeggiante a 2Hz: Il modulo successivo ha permesso di far lampeggiare un LED a una frequenza di 2Hz utilizzando un contatore sincronizzato con il clock della scheda. Questo esercizio ha mostrato come utilizzare un contatore per generare frequenze diverse da quella del clock di sistema.
- Controllo del Lampeggiamento tramite Switch: Abbiamo esteso il modulo precedente per controllare il lampeggiamento del LED tramite uno switch. Quando lo switch è su "On", la frequenza del lampeggiamento raddoppia.

- Contatore Binario: Abbiamo implementato un contatore binario che mostra il numero di cicli di clock sui LED della scheda. Questo esercizio ha evidenziato l'uso dei registri per la visualizzazione sequenziale di stati.
- Joystick: Un modulo che simula il comportamento di un joystick, spostando la luce a sinistra o a destra alla pressione dei rispettivi pulsanti, mostrando un controllo più complesso e interattivo.
- Effetto KITT: Ispirato alla famosa serie TV "Knight Rider", questo modulo ha creato un effetto luminoso a scorrimento sui LED, dimostrando un uso avanzato di contatori e controllo direzionale.
- Calcolatrice: Infine, abbiamo sviluppato una calcolatrice che utilizza pulsanti e switch per eseguire operazioni aritmetiche.

6.6 Considerazioni Finali

Questo progetto ha permesso di consolidare le competenze nella progettazione con FPGA, mettendo in pratica i concetti teorici attraverso una serie di esercizi progressivamente più complessi. L'utilizzo della scheda Basys3 e del linguaggio Verilog ha fornito un'esperienza concreta e diretta nella gestione delle periferiche e nella progettazione di sistemi digitali.

In futuro, queste competenze potranno essere estese a progetti più complessi e specifici, sfruttando le potenzialità delle FPGA in ambiti come l'elaborazione dei segnali, il controllo industriale e lo sviluppo di prototipi hardware.