Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Report di Laboratorio - Tecniche di Acquisizione Dati

FPGA

Studenti
Simone Seria
Giulia Diletta Fabbriciani

Anno Accademico 2023/2024

Introduzione

La prima esperienza di laboratorio riguarda la realizzazione di programmi per la scheda di sviluppo FPGA Basys 3 Artix-7 prodotta da Digilent.

Scopo

Le applicazioni da realizzare sono:

- Accensione di un LED alla pressione di un pulsante
- Far lampeggiare un LED con frequenza di 2Hz
- Far lampeggiare un LED con frequenza di 2Hz quando il primo switch è acceso
- Far lampeggiare un LED con frequenza di 2Hz quando il primo switch è spento
- Realizzare un contatore binario e mostrarlo sui 16 LED a disposizione della FPGA
- Fare in modo che un led si sposti a destra e sinistra in base alla pressione dei pulsanti
- Fare in modo che un led si sposti a destra e sinistra rimbalzando

Strumenti

Tutti i programmi sono stati realizzati su Vivado utilizzando Verilog. Le immagini del codice sono state generate tramite l'estensione "CodeSnap" di Visual Studio Code.

FPGA

Le FPGA (Field Programmable Gate Arrays) sono schede programmabili molto flessibili. Permettono di lavorare direttamente sulla struttura interna del circuito e quindi consentono di implementare qualunque logica digitale e a differenza dei circuiti integrati, che vengono progettati per funzioni specifiche, le FPGA possono essere riprogrammate più volte per modificare il comportamento del circuito, rendendole ideali per prototipi e sistemi evolutivi.

Realizzazione dei Programmi

Led con Pulsante

Il primo programma consiste nell'accensione e spegnimento di un LED alla pressione del pulsante centrale dell'FPGA.

Innanzitutto dichiariamo e inizializziamo i dati che ci servono.

```
module buttonCled(
    input clk,
    output reg[7:0] led,
    input wire btnC
);
reg [7:0] debounce;
reg state = 0;
reg control = 1;
```

Figura 1

All'interno delle parentesi tonde di "buttonCled()" inizializziamo solo gli input e output, che in questo caso sono il clock della scheda, l'array dei LED e il pulsante centrale.
Il resto delle operazioni riguarda tutti i dati che vanno inizializzati una volta sola all'inizio dell'esecuzione: un array per gestire il debounce e due variabili di stato e controllo.

Per il prossimo passo entriamo nella sezione "always" del codice, che viene eseguito continuamente e che associamo al fronte positivo del clock della scheda. Immaginando il clock come un'onda quadra, il fronte positivo è il momento in cui l'onda "si alza". Il clock nel nostro caso è 100MHz.

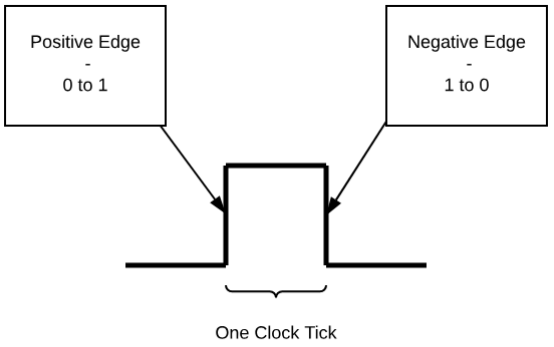

Figura 2

```
always @(posedge clk) begin
    led[0] <= state;

    if(btnC) begin
        if(debounce < 8'd10) begin
            debounce <= debounce + 1;
        end
    end else begin
        debounce <= 0;
    end

    if(debounce == 8'd10)
        if(control) begin
            state <= state + 1;
            control <= control + 1;
        end
    else begin
        if(!control)
            control <= control + 1;
        end

end
```

Figura 3

Debounce

Perché serve usare l'array di debounce? Per capirlo bisogna prima introdurre il concetto stesso di "debounce". Essendo il codice nel blocco "always" eseguito 100 milioni di volte al secondo, quando il pulsante centrale viene premuto può sembrarci una cosa semplice ma in realtà la frazione di secondo in cui rimane abbassato viene vista dal programma come una "pressione prolungata". Di conseguenza dobbiamo fare in modo che il programma capisca bene in che modo il tasto viene premuto. Per fare ciò, usiamo l'array "debounce" composto di 8 bit binari. Lo incrementiamo di volta in volta fino a quando il suo valore non è 10. In questo modo, passiamo alla sezione di codice che si occupa della gestione del LED solo dopo che il tasto sia stato rilasciato.

Una volta che il debounce è completo, verifichiamo che "control" sia "1". In caso positivo, mettiamo anche "state" a "1" (che all'inizio del codice gestisce l'accensione o lo spegnimento del LED). Infine, invertiamo control in modo da essere pronti la prossima volta che il tasto viene premuto.

Led Lampeggiante

Il secondo programma ha come scopo il far lampeggiare dei LED con condizioni differenti di volta in volta.

Anche questa volta, dichiariamo e inizializziamo i dati che ci servono.

```
module blink(
    input clk,
    output reg [7:0] led,
    input wire [15:0] sw
);
```

Figura 4

Nel primo caso, vogliamo far lampeggiare semplicemente un led a 2Hz. Oltre al clock e all'array dei led inizializziamo anche quello degli switch, visto che ci servirà in seguito.

```
reg [32:0] counter;
always @(posedge clk) begin
      led[0] <= counter[26];
      counter <= counter + 1;
end
endmodule
```

Figura 5

Il blocco always viene eseguito a ogni fronte positivo del clock. Il led viene acceso ogni volta che led[26] diventa "1". Essendo la frequenza del clock 100MHz, non possiamo semplicemente accendere e spegnere il LED in base al fronte positivo. Dobbiamo trovare un modo per "rallentare" la frequenza. Incrementando ad ogni fronte positivo il valore di "counter", quindi 200 milioni di volte al secondo, di conseguenza il 26esimo bit verrà incrementato una volta ogni 0.67 secondi, ovvero con una frequenza di 1.49Hz.

Se volessimo far lampeggiare il LED solo in base alla posizione di uno switch, basterebbe includere il blocco always in un if-else.

```
if(sw[0]) begin
        led[0] <= counter[25];
        counter <= counter + 1;
    end else
        led[0] <= 0;
```

Figura 6 - Il LED lampeggia quando lo switch è acceso

```
if(!sw[0]) begin
        led[0] <= counter[25];
        counter <= counter + 1;
    end else
        led[0] <= 0;
```

Figura 7 - Il LED lampeggia quando lo switch è spento

In entrambi i casi, nella condizione opposta il led viene settato a 0.

Possiamo fare anche in modo che in base alla posizione dello switch il LED lampeggi più o meno velocemente.

```
if(sw[0]) begin
        led[0] <= counter[25];
        counter <= counter + 1;
    end else
        led[0] <= counter[26];
        counter <= counter + 1;
```

Figura 7

In quest'ultimo caso, il LED lampeggerà a 1.49Hz quando lo switch è "acceso" e a 2.98Hz quando lo switch è "spento".

Contatore Binario

Consideriamo i 16 LED presenti sulla scheda. Vogliamo realizzare un contatore binario in cui un LED spento corrisponde ad uno “0” e un LED acceso corrisponde ad un “1”.

```
module bcount(
input clk,
output reg [15:0] led
);
reg [32:0] counter;
reg oldcounter;
```

Figura 8

Le inizializzazioni sono praticamente uguali agli altri programmi. Anche stavolta usiamo il fronte positivo del clock per il blocco always.

```
always @ (posedge clk) begin
oldcounter <= counter[25];
    if (counter[25] != oldcounter) begin
        led <= led + 1;
        end else
                counter <= counter + 1;
end
```

Figura 9

Utilizziamo come al solito “counter[25]” per “rallentare” il clock e, ogni 0.67secondi incrementiamo il valore contenuto nei 16 bit di LED. In questo modo, il contatore è direttamente associato ai LED che ne mostrano l’andamento. Inseriamo tutto all’interno di un “if” per essere sicuri dell’intervallo di tempo che deve passare per poter incrementare “counter”.

Led che “Saltella”

Adesso vogliamo realizzare un programma in cui, partendo con un solo LED acceso, questi si sposti a destra e sinistra autonomamente.

Partiamo dalle inizializzazioni:

```
module bnfled(
input clk,
output reg [15:0] led
);
initial
led[15] = 1;
reg [32:0] counter;
reg oldcounter;
```

Figura 10

Procediamo poi con il blocco always:

```
always @ (posedge clk) begin
oldcounter <= counter[25];
if (counter[25] != oldcounter) begin
    if(led[15] & !led[0])begin
        led[15] <= 0;
        led[0] <= 1;
    end else
            if(led[0] & !led[15]) begin
                led[0] <= 0;
                led[15] <= 1;
            end
    end else
            counter <= counter + 1;
end
```

Figura 11

Anche in questo caso usiamo counter[25] per rallentare il clock e inseriamo il tutto in un “if” per controllarlo. Nel successivo “if” ci accertiamo di non essere arrivato al primo o ultimo LED, in modo da poter passare al lato successivo se necessario, con il tipico effetto “pac-man”.

Led tramite Joystick

L'ultimo programma ha lo scopo di spostare un LED acceso in partenza a destra e sinistra in base alla pressione del joystick. Iniziamo anche questa volta dalle inizializzazioni:

```
module moveLight(
    input clk,
    output reg[15:0] led,
    input wire btnR,
    input wire btnL
);
initial
led[15] = 1;
reg controlR = 0;
reg controlL = 0;
```

Figura 12

L'unica differenza è "initial", ovvero un valore di default che viene impostato ogni volta che il programma viene eseguito (ci serve per partire dallo stato di "un singolo LED già acceso").

Il blocco always, anche stavolta associato al clock, collega i due pulsanti del joystick ai nostri due bit di controllo:

```
controlR <= btnR;
controlL <= btnL;
```

Figura 13

Infine controlliamo se e quale pulsante del joystick viene premuto dall'utente:

```
if(controlR == 0 & btnR == 1 & !led[0]) begin
      led <= led >> 1;
end
if(controlL == 0 & btnL == 1 & !led[15]) begin
      led <= led << 1;
end
```

Figura 14

Utilizziamo l'operatore ">>" o "<<", ovvero l'operatore di "shift". Spostiamo a destra o sinistra il bit acceso in base a quale dei due pulsanti viene premuto.

Conclusioni

Grazie a questa esperienza laboratoriale, ci siamo accorti di quanto versatile possa essere una scheda come FPGA.

Abbiamo avuto modo di entrare in contatto con meccaniche come il “Debounce”, estremamente utili alla gestione degli input utente e importanti perché il codice sia consistente e si comporti in maniera prevedibile.

Abbiamo anche avuto modo di capire in che modo il “clock” di un processore non sia un concetto aleatorio ma abbia effetti sul funzionamento diretto dei programmi.