Politecnico di Milano

Progetto di Reti Logiche

Autori:

Amina El Kharouai, matr. 982701

Jihad Founoun, matr. 990905

2023/2024

Indice

1 Introduzione **4**
- 1.1 Applicazione pratica 4
- 1.2 Scopo del progetto 4
- 1.3 Specifiche generali 5
- 1.4 Interfaccia del componente 5
  - 1.4.1 Segnali in ingresso al componente 6
  - 1.4.2 Segnali in uscita dal componente 6
- 1.5 Dati 7

2 Architettura **8**
- 2.1 Componenti 8
  - 2.1.1 Counter_Address 8
  - 2.1.2 Counter_K 9
  - 2.1.3 Selector 10
- 2.2 Finite State Machine (FSM) 11
- 2.3 Scelte progettuali 13

3 Risultati sperimentali **15**
- 3.1 Sintesi 15

4 Test Bench **17**
- 4.1 Test Bench di Esempio 17
- 4.2 Credibilità fino a zero 18
- 4.3 Primo valore della sequenza nullo 18
- 4.4 Due computazioni consecutive 18

5 Conclusioni **19**

Introduzione

Negli ultimi anni la tecnologia ha avuto un enorme virata verso l'internet of things (IoT), in cui oggetti intelligenti dotati di memoria e capacità di elaborazione comunicano tra loro. Oggigiorno, infatti, molti oggetti che ci circondano, presenti in case, automobili e luoghi pubblici, sono dotati di processori e memorie propri, guidati da software dedicati. Il progetto su cui abbiamo lavorato si cala bene su questo tipo di realtà: infatti abbiamo ideato un componente software in grado di interfacciarsi con una memoria ed eseguire opportuni aggiornamenti e stime su dati raccolti in essa presenti.

1.1 Applicazione pratica

Un esempio pratico di applicazione di questo tipo di componente è quello di un dispositivo domestico di purificazione dell'aria, dotato di sensori che rilevano l'indice di umidità o presenza di polvere e che periodicamente vanno a salvare questi dati in una memoria presente nel dispositivo, per poi presentarli successivamente all'utente; tuttavia, queste rilevazioni non avvengono in maniera continuativa, in modo da non gravare sulla principale funzione del dispositivo, ovvero quella di purificazione; oppure, per qualche malfunzionamento, i sensori potrebbero non essere riusciti a rilevare la temperatura in un certo momento; pertanto vanno eseguite delle elaborazioni e stime sui dati in grado di integrare le informazioni mancanti.

1.2 Scopo del progetto

Lo scopo del progetto, infatti, è quello di elaborare dei dati presenti in memoria e scrivere in essa il risultato di questa elaborazione, assegnando ad ogni dato un opportuno indice di credibilità. Ad ogni dato x va associato un indice di credibilità c che varia da 0 a 31. L'elaborazione va eseguita k volte,

pertanto le computazioni varieranno da i=0 a i=k-1; associamo pertanto ad ogni dato letto la sua computazione di lettura, ad ogni dato scritto la sua credibilità e la computazione di scrittura in modo da formare una tupla:

$$x_{(r\to i)}\langle x_{(w\to i)}; c_{(w\to i)}\rangle$$

Dove $x_{(r\to i)}$ è il dato letto alla computazione i; $x_{(w\to i)}$ è il dato scritto alla computazione i e $c_{(w\to i)}$ è la credibilità assegnata al dato scritto alla computazione i, scritta anch'essa assieme al dato in memoria.

1.3 Specifiche generali

La computazione del dato e l'assegnazione della credibilità, come approccio, ricordano molto un filtro bayesiano sequenziale, ovvero un filtro che assegna ad ogni nuovo dato la massima credibilità e, in assenza di nuovi dati, va a riusare il dato precedente assegnandogli un indice di credibilità minore modulato opportunamente. Come detto precedentemente il componente dovrà leggere da memoria un dato $x_{(r\to i)}$ ed elaborarlo opportunamente a seconda dei seguenti casi:

- Il dato letto è quello iniziale (alla computazione $i = 0$) ed è pari a $x_{(r\to i)} = 0$; solo in questo caso e finché non verrà letto un dato diverso da 0, il dato scritto è 0 e la credibilità assegnata è 0.

$$\langle x_{(w\to i)}; c_{(w\to i)}\rangle = \langle 0; 0\rangle$$

- Il dato letto è diverso da 0; in questo caso il dato da scrivere in memoria corrisponde al dato letto e la credibilità da assegnare è 31.

$$\langle x_{(w\to i)}; c_{(w\to i)}\rangle = \langle x_{(r\to i)}; 31\rangle$$

- Il dato letto è pari a $x_{(r\to i)} = 0$ e non ricadiamo nel caso 1); in questo caso il dato scritto è pari a quello scritto nella computazione precedente; la credibilità scritta è pari a quella scritta nella computazione precedente decrementata di 1.

$$\langle x_{(w\to i)}; c_{(w\to i)}\rangle = \langle x_{(w\to i-1)}; c_{(w\to i-1)} - 1\rangle$$

1.4 Interfaccia del componente

L'interfaccia del componente comprende segnali in ingresso e in uscita così descritti:

```
entity project_reti_logiche is
    Port (
        i_clk: in std_logic;
        i_rst: in std_logic;
        i_start: in std_logic;
        i_k: in std_logic_vector(9 downto 0);
        i_add: in std_logic_vector(15 downto 0);

        o_done: out std_logic;

        o_mem_addr: out std_logic_vector(15 downto 0);
        i_mem_data: in std_logic_vector(7 downto 0);
        o_mem_data: out std_logic_vector(7 downto 0);
        o_mem_we: out std_logic;
        o_mem_en: out std_logic
    );
end project_reti_logiche;
```

Figura 1.1: Interfaccia del componente

1.4.1 Segnali in ingresso al componente

- i_clk: segnale di clock settato nei constraints a 20 ns.
- i_rst: segnale di reset.
- i_start: segnale che indica quando è possibile cominciare con la computazione dei dati.
- i_k: segnale da 10 bit che indica la lunghezza della stringa di dati da elaborare.
- i_add: segnale da 16 bit che indica l'indirizzo della memoria da cui cominciare a leggere.
- i_mem_data: segnale da 8 bit che contiene il dato letto da memoria.

1.4.2 Segnali in uscita dal componente

- o_done: segnale che se settato a 1 va ad indicare l'avvenuta elaborazione dei dati.
- o_mem_addr: segnale da 16 bit che indica l'indirizzo della memoria da cui si vuole leggere.
- o_mem_data: segnale da 8 bit che contiene l'informazione che si vuole scrivere in memoria.

- o_mem_we: segnale per abilitare l'accesso in scrittura a memoria.
- o_mem_en: segnale per abilitare l'accesso a memoria.

1.5 Dati

Di seguito un semplice disegno esplicativo di come è strutturata l'informazione in memoria:

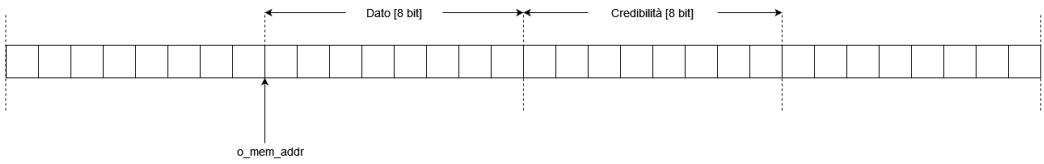

Figura 1.2: Enter Caption

Con l'indirizzo di accesso a memoria o_mem_addr, calcolato opportunamente a partire da i_add sommando di volta in volta due byte, si accede in memoria all'indirizzo contenete il dato da leggere:

$$x_{(r \to i)} = o\_mem\_addr[0:7]$$

Una volta terminata la computazione del dato andremo a sovrascrivere in memoria il nuovo dato nella posizione dove avevamo letto quello precedente e la credibilità nel byte successivo:

$$o\_mem\_addr[0:7] = x_{(w \to i)} o\_mem\_addr[8:15] = c_{(w \to i)}$$

Successivamente verrà calcolato il nuovo indirizzo per la computazione successiva, oppure se raggiunto il numero di elaborazioni richieste, verrà settato a 1 il segnale di o_done.

Architettura

L'architettura sulla quale abbiamo deciso di costruire un progetto è basata su tre componenti principali: un contatore di indirizzi, un contatore di elaborazioni ed un componente di selezione del dato e della credibilità da scrivere in memoria; tutti questi componenti fanno uso anche di sottocomponenti accessori, come registri e mutex.

2.1 Componenti

Come detto precedentemente nel progetto vi sono tre macro componenti guidati da una macchina a stati finiti al fine di elaborare adeguatamente la stringa di dati presente in memoria; tutti i componenti hanno in comune il segnale di clock (**i_clk**), il segnale di reset generale (**i_rst**) e un segnale di reset (**init**) lanciato dall'FSM per permettere di poter fare computazioni successive di stringhe.

2.1.1 Counter_Address

Il componente **Address_counter** è un contatore che gestisce l'indirizzamento della memoria in lettura e scrittura. Questo componente si interfaccia principalmente con il modulo FSM (Finite State Machine) e l'adder.

I segnali in ingresso per **Counter_Address** sono:

- **i_clk**: il segnale di clock che sincronizza le operazioni del contatore.
- **i_rst**: un segnale di reset che riporta il contatore al suo stato iniziale.
- **init**: un segnale di reset aggiuntivo utilizzato per riportare il contatore al suo stato iniziale.

- **en_counter**: un segnale di abilitazione che, quando attivo, carica l'indirizzo iniziale nel contatore.
- **on_counter**: un segnale che, quando attivo, incrementa l'indirizzo corrente del contatore di 2.
- **i_add**: un vettore di logica standard di 16 bit che rappresenta l'indirizzo iniziale da cui inizia il conteggio.

Il segnale in uscita è:

- **addr**: un vettore di logica standard di 16 bit che rappresenta l'indirizzo corrente del contatore.

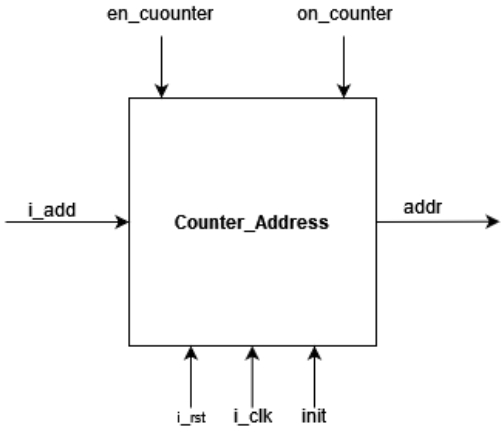

Figura 2.1: Componente Counter_Address

Il componente calcola l'indirizzo di memoria di lettura del valore, ed ad ogni fronte di salita del clock qual ora il segnale on_counter sia alto, lo incrementa di 2 byte.

2.1.2 Counter_K

Il componente **Counter_K** gestisce il conteggio delle iterazioni di elaborazione. Anche questo componente si interfaccia principalmente con il modulo FSM.

I segnali in ingresso per **Counter** sono:

- **i_clk**: il segnale di clock che sincronizza le operazioni del contatore.

- **i_rst**: un segnale di reset che riporta il contatore al suo stato iniziale.
- **init**: un segnale di reset aggiuntivo utilizzato per riportare il contatore al suo stato iniziale.
- **en_count_k**: un segnale di abilitazione che, quando attivo, carica il valore iniziale nel contatore.
- **on_count_k**: un segnale che, quando attivo, incrementa il valore corrente del contatore di 1.
- **i_k**: un vettore di logica standard di 10 bit che rappresenta il valore iniziale da cui inizia il conteggio

Il segnale in uscita è:

- **o_stop**: un segnale che indica quando il contatore ha raggiunto il valore di k.

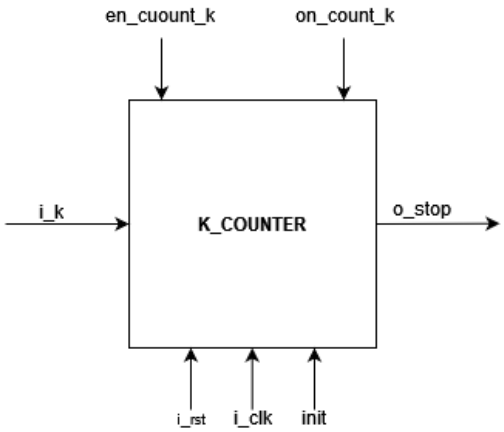

Figura 2.2: Componente Counter_K

Il componente tiene traccia del numero di iterazioni eseguite e segnala quando tutte le iterazioni richieste sono state completate.

2.1.3 Selector

Il componente **Selector** è responsabile della selezione dei dati e della loro credibilità da scrivere in memoria. Si interfaccia con il modulo FSM e con la memoria.

I segnali in ingresso per **Selector** sono:

- **i_clk**: il segnale di clock che sincronizza le operazioni del selettore.
- **i_rst**: un segnale di reset che riporta il selettore al suo stato iniziale.
- **init**: un segnale di reset aggiuntivo utilizzato per riportare il selettore al suo stato iniziale.
- **i_value_reg**: un vettore di logica standard di 8 bit che rappresenta il valore corrente nel registro dei valori.
- **i_credibility_reg**: un vettore di logica standard di 8 bit che rappresenta la credibilità corrente nel registro della credibilità.
- **i_mem_data**: un vettore di logica standard di 8 bit che rappresenta il dato corrente nella memoria.

I segnali in uscita sono:

- **o_value_is_zero**: un segnale che indica se il valore corrente letto in memoria è zero.
- **o_credibility_register**: un vettore di logica standard di 8 bit che rappresenta la credibilità da caricare nel registro della credibilità e da scrivere in memoria.
- **o_value_register**: un vettore di logica standard di 8 bit che rappresenta il valore da caricare nel registro dei valori e da scrivere in memoria.

Il componente elabora il dato letto dalla memoria e, a seconda dei casi descritti nelle specifiche generali, seleziona il dato e la credibilità da scrivere in memoria.

2.2 Finite State Machine (FSM)

L'**FSM** (Finite State Machine) gestisce il flusso di controllo tra i vari componenti. Oltre a ricevere dei segnali di ingresso propri del progetto, usa anche dei segnali generati dai vari componenti per passare da uno stato all'altro; l'FSM controlla una serie di segnali di uscita che abilitano o disabilitano i

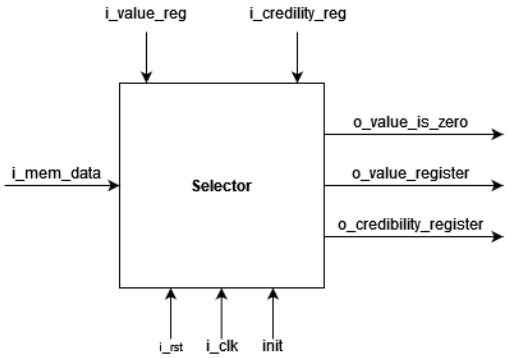

Figura 2.3: Componente Selector

vari componenti e operazioni, come la lettura e la scrittura della memoria, l'incremento dei contatori, il caricamento dei registri, ecc.

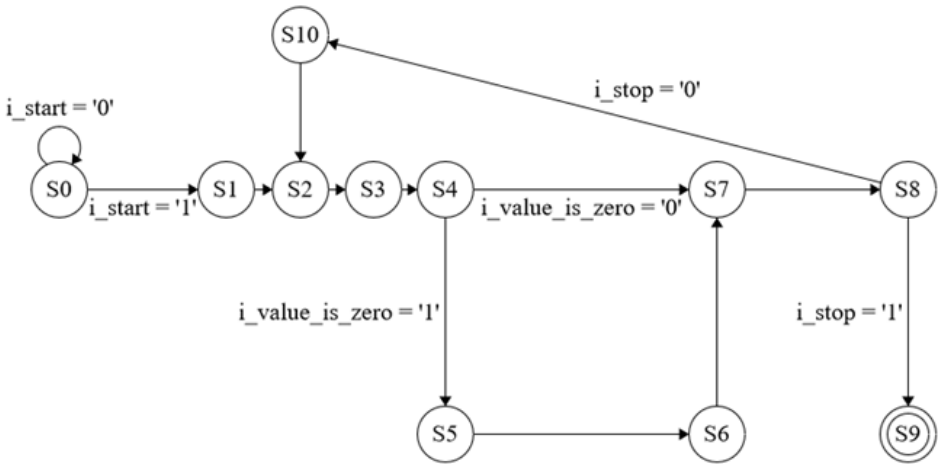

Figura 2.4: FSM

Gli stati della FSM sono:

- **s0**: Questo è lo stato iniziale. Se il segnale i_start è '1', l'FSM passa allo stato s1; altrimenti rimane nello stato s0.
- **s1**: In questo stato l'FSM abilita i contatori counter_k e counter_address in modo tale che possano leggere i valori in ingresso i_k e i_add e passa allo stato s2.

- **s2**: In questo stato l'FSM abilita la lettura dalla memoria leggendo così un valore della sequenza e passa allo stato s3.
- **s3**: Questo è uno stato di attesa della lettura durante il quale l'FSM continua ad abilitare la lettura della memoria e passa allo stato s4.
- **s4**: In questo stato avendo quindi il valore letto da memoria l'FSM abilità i registri che, grazie al selettore, riceveranno in ingresso i dati corretti. Se il valore letto corrente è zero, si passa allo stato s5, altrimenti si passa allo stato s7. In questo stato l'FSM abilita anche la conta del counter_k tenendo traccia della lettura del valore corrente della sequenza.
- **s5**: In questo stato l'FSM inizia la scrittura del valore e passa allo stato s6.
- **s6**: Questo è uno stato di attesa di scrittura durante il quale l'FSM continua a scrivere il valore e passa allo stato s7.
- **s7**: In questo stato l'FSM inizia la scrittura della credibilità e passa allo stato s8.
- **s8**: Questo è uno stato di attesa di scrittura durante il quale l'FSM continua a scrivere la credibilità. Se il contatore counter k ha raggiunto il valore di k dunque se il segnale i_stop='1', si passa allo stato s9, altrimenti si passa allo stato s10.
- **s9**: In questo stato l'FSM segnala la fine dell'elaborazione, portando o_done al valore alto. Se il segnale i_start è '0', ritorna allo stato s0.
- **s10**: In questo stato l'FSM incrementa l'indirizzo del contatore counter_address e ritorna allo stato s2 per iniziare la lettura del valore successivo.

2.3 Scelte progettuali

Nella fase di progettazione del componente, inserendo compoenti come il Counter_Address, il Counter_K e il Selector abbiamo adottato un approccio che privilegia la **scalabilità** e l'**adattabilità**.

Questa scelta è stata guidata dalla volontà di creare un sistema che non solo rispondesse alle esigenze immediate del progetto, ma che potesse essere facilmente modificato e **riutilizzato in contesti diversi**.

Per esempio, abbiamo previsto la possibilità di modificare l'indice di credibilità massimo, attualmente impostato a 31, per adattarlo a diverse esigenze. Questo permette di personalizzare il comportamento del componente in base alle specifiche necessità del progetto in cui viene utilizzato.

Inoltre, abbiamo implementato il componente in modo da poter avre diverse impostazioni di default che possono essere facilmente modificate; come ad esempio la decrementazione della credibilità ad ogni computazione successiva nel caso in cui si legge il valore 0, si può impostare la decrementazione di due o più unità al posto di una. Questo offre un ulteriore livello di flessibilità, permettendo agli sviluppatori di configurare il componente in modo da rispondere al meglio alle esigenze del loro progetto.

Queste scelte progettuali pertanto non si limitano a risolvere il problema immediato, ma anticipano anche le esigenze future, fornendo soluzioni che possono essere adattate e riutilizzate in una varietà di contesti.

Risultati sperimentali

3.1 Sintesi

Il componente ha superato la sintesi. Analizzando il report di utilizzo, sono state ricavate informazioni dettagliate sull'uso dei registri. In particolare, vengono utilizzati 75 registri Flip-Flop, di cui nessuno è un Latch. In seguito, la Slice Logic riporta l'uso delle risorse dell'FPGA da parte del componente sintetizzato:

Infine, utilizzando il report di temporizzazione, è stato calcolato uno Slack

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	67	0	0	134600	0.05
LUT as Logic	67	0	0	134600	0.05
LUT as Memory	0	0	0	46200	0.00
Slice Registers	75	0	0	269200	0.03
Register as Flip Flop	75	0	0	269200	0.03
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Figura 3.1: Slice Logic Report

di 15,783 ns, che rappresenta la differenza tra il periodo disponibile e il tempo utilizzato.

```
Slack (MET) :             15.783ns  (required time - arrival time)
```

Figura 3.2: Tempi di lavoro

Ricavando cosi la frequenza massima di lavoro:

$$f = 237{,}71\,\mathrm{MHz} \tag{3.1}$$

Test Bench

Il componente è stato testato utilizzando il test bench fornito con la specifica, nonché altri test bench per verificare il funzionamento in situazioni particolari (corner case).

4.1 Test Bench di Esempio

Il primo test effettuato è stato quello fornito come test di esempio. Questo test prevede uno scenario con 14 elementi, che inizia con un valore diverso da zero ma contiene diverse sequenze consecutive di zeri. Questo consente di verificare il corretto funzionamento del modulo quando la credibilità scritta è la precedente diminuita di 1.

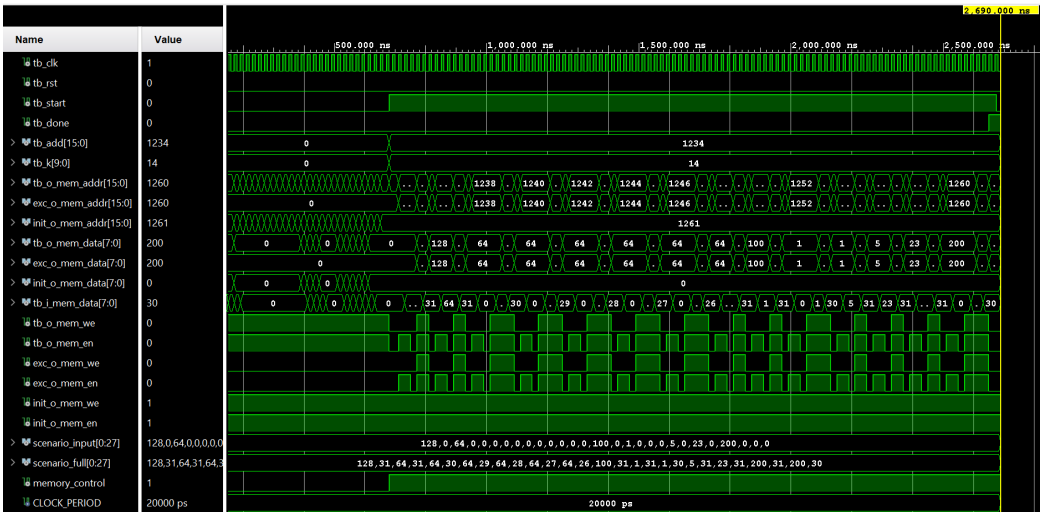

Figura 4.1: Wave Diagram

4.2 Credibilità fino a zero

Questo test bench verifica il comportamento del modulo VHDL in uno scenario con 36 valori, inclusa una lunga sequenza di zeri seguita da un recupero di valori positivi. Lo scenario è progettato per testare se il modulo riduce correttamente la credibilità fino a zero e poi recupera correttamente la credibilità quando incontra un valore positivo.

4.3 Primo valore della sequenza nullo

Questo test bench verifica il comportamento del modulo VHDL quando il primo valore della sequenza di lettura dalla memoria è nullo (zero). In questo caso, la credibilità deve essere impostata al minimo valore (zero) immediatamente.

4.4 Due computazioni consecutive

Infine, è stato verificato il comportamento del modulo quando viene sottoposto a due computazioni consecutive. Lo scopo è assicurarsi che il modulo possa gestirle correttamente. Questo test è stato effettuato sia con reset intermedio che con solo il segnale di start.

Conclusioni

Abbiamo portato a termine con successo il progetto, garantendo un'implementazione accurata e completa delle specifiche richieste. Durante la fase di sviluppo, abbiamo prestato particolare attenzione a garantire che ogni aspetto del progetto fosse **adeguatamente testato**. Questo ci ha permesso di identificare e risolvere eventuali problemi in modo tempestivo, assicurando la **robustezza** e l'**affidabilità** del nostro componente.

Abbiamo anche esplorato diverse **applicazioni pratiche** del nostro progetto. Ad esempio, il componente potrebbe essere utilizzato in dispositivi IoT per gestire e analizzare dati raccolti da sensori. Inoltre, grazie alla sua **scalabilità** e **adattabilità**, il componente può essere facilmente integrato in vari contesti e applicazioni, rendendolo un elemento versatile per una vasta gamma di progetti.

In conclusione, siamo soddisfatte del lavoro svolto. Abbiamo acquisito una comprensione più profonda dei concetti di progettazione hardware e abbiamo avuto l'opportunità di applicare queste conoscenze in un contesto pratico. Siamo fiduciose che le competenze e le esperienze acquisite durante questo progetto saranno di grande valore per i nostri futuri impegni nel campo dell'ingegneria informatica.