Esercizio: Progettazione di un Contatore Circolare a 2 bit

Obiettivo dell'Esercizio

Siete chiamati a progettare un circuito logico sequenziale che realizzi il comportamento di un contatore di numeri interi da 0 a 3. Il circuito deve generare un output a due bit ($q_0 q_1$) e deve incrementare il proprio valore **solo ed esclusivamente** quando uno specifico segnale di input, denominato "Enable" (E), è attivo ($E = 1$). Se il segnale di Enable non è attivo ($E = 0$), il contatore deve mantenere il suo stato corrente.

Consegna

Al fine di modellare il comportamento del sistema e ricavare le equazioni logiche che lo governano, si richiede di svolgere i seguenti punti:

1. **Macchina a Stati Finiti (FSM):** Disegnare la FSM del dispositivo. Rappresentare graficamente gli stati del contatore e tracciare le transizioni tra di essi, indicando chiaramente come il passaggio da uno stato all'altro dipenda dal valore del segnale Enable.

2. **Tabella della Verità:** Compilare la tabella per la funzione di stato successivo ($next\_state$). La tabella dovrà esplicitare lo stato successivo in funzione di tutte le combinazioni possibili dello stato corrente e del segnale di Enable in ingresso.

3. **Funzioni Logiche:** Partendo dalla tabella della verità, estrapolare le espressioni booleane (sotto forma di somma di prodotti) per i due bit di output: il bit meno significativo (q_1) e il bit più significativo (q_0).

4. **Semplificazione e Circuito:** Applicare i teoremi dell'algebra di Boole per semplificare al massimo le espressioni ottenute, in vista di minimizzare il numero di porte logiche necessarie. Disegnare infine lo schema logico del circuito risultante.

Soluzione dell'Esercizio

1. Macchina a Stati Finiti (FSM)

Il comportamento del dispositivo può essere modellato tramite la seguente Macchina a Stati Finiti (FSM). Ogni stato rappresenta il valore attuale del contatore in formato binario ($q_0 q_1$).

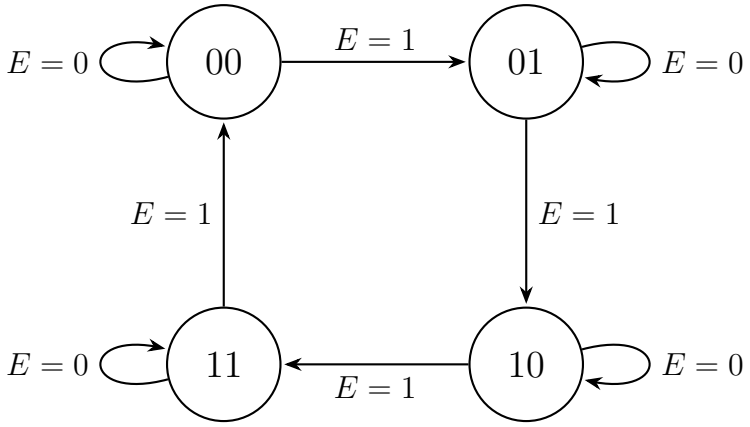

Figura 1: Diagramma degli stati (FSM) del contatore.

1.1 Come leggere questo diagramma? Ragioniamoci insieme

La Macchina a Stati Finiti (FSM) è la mappa comportamentale del nostro sistema sequenziale. I cerchi rappresentano gli *stati* del sistema, ovvero i valori che la memoria del contatore può assumere in un dato istante (i numeri binari da 0 a 3: 00, 01, 10 e 11).

Le frecce, invece, sono le *transizioni*: indicano come il sistema si evolve da uno stato all'altro in base al segnale di controllo in ingresso, ovvero **Enable (E)**:

- **Se il segnale è disattivo ($E = 0$):** Seguiamo la transizione che curva su se stessa. Questo indica che il sistema deve mantenere il dato attuale in memoria senza alterarlo. Se il contatore segnava 10 (2 in decimale), rimarrà a 10.
- **Se il segnale è attivo ($E = 1$):** Seguiamo la transizione che punta allo stato successivo. Il circuito esegue l'incremento. È importante notare che dallo stato 11 (3 in decimale), una transizione con $E = 1$ riporta il sistema allo stato 00, definendo così la natura *circolare* del contatore (overflow controllato).

Disegnare la FSM è il primissimo passo della progettazione: definisce visivamente le specifiche esatte di ciò che l'hardware dovrà fare.

2. Tabelle della Verità (*next_state*)

Di seguito sono riportate le tabelle per la funzione *next_state*, che descrivono lo stato successivo in funzione dello stato corrente e del segnale di Enable.

Current State	**Next State**	
	$E = 0$	$E = 1$
00	00	01
01	01	10
10	10	11
11	11	00

Tabella 1: Tabella standard delle transizioni di stato.

O, in altra forma equivalente:

Enable	$\mathbf{q_0q_1 = 00}$	$\mathbf{q_0q_1 = 01}$	$\mathbf{q_0q_1 = 10}$	$\mathbf{q_0q_1 = 11}$
0	$q_0q_1 = 00$	$q_0q_1 = 01$	$q_0q_1 = 10$	$q_0q_1 = 11$
1	$q_0q_1 = 01$	$q_0q_1 = 10$	$q_0q_1 = 11$	$q_0q_1 = 00$

Tabella 2: Tabella esplicita estesa.

2.1 Dalla FSM alla tabella: definire le specifiche formali

Il diagramma a stati è ottimo per comprendere il comportamento generale del sistema, ma per progettare l'hardware fisico della rete logica abbiamo bisogno di una specifica più formale e rigorosa.

La tabella della verità funge proprio da ponte tra il comportamento desiderato e le equazioni matematiche. Essa mappa in modo esauriente tutte le combinazioni possibili degli ingressi (lo *Stato Corrente* memorizzato e il valore attuale dell'*Enable*) per stabilire in modo univoco quale dovrà essere l'uscita desiderata (*Stato Successivo*).

Leggendo ad esempio la seconda tabella (quella estesa):

- Nella riga in cui l'Enable vale 0, i valori di Stato Successivo sono una copia esatta dello Stato Corrente ($00 \to 00$, $01 \to 01$, ecc.). Questo codifica la funzione di "mantenimento" (hold) della memoria.
- Nella riga in cui l'Enable vale 1, osserviamo la codifica dell'incremento binario ($00 \to 01 \to 10 \to 11 \to 00$).

Questa tabella sarà la "lista delle istruzioni" da cui estrarremo direttamente le porte logiche necessarie.

3. Funzioni Logiche e Semplificazione

La funzione logica corrispondente a *next_state* sarà data dalla somma (OR) delle combinazioni dei tre input booleani (q_0, q_1 ed E) che restituiscono output a 1.

Calcolo e semplificazione del bit più significativo (q_0):

$$\begin{aligned} next\_state(q_0) &= \overline{E}(q_0\overline{q_1} + q_0q_1) + E(\overline{q_0}q_1 + q_0\overline{q_1}) \\ &= q_0\overline{q_1}\,\overline{E} + q_0q_1\overline{E} + \overline{q_0}q_1E + q_0\overline{q_1}E \\ &= q_0\overline{q_1}(\overline{E} + E) + q_0q_1\overline{E} + \overline{q_0}q_1E \\ &= q_0\overline{q_1} + q_0q_1\overline{E} + \overline{q_0}q_1E \end{aligned}$$

Calcolo e semplificazione del bit meno significativo (q_1):

$$\begin{aligned} next\_state(q_1) &= \overline{E}(\overline{q_0}q_1 + q_0q_1) + E(\overline{q_0q_1} + q_0\overline{q_1}) \\ &= \overline{q_0}q_1\overline{E} + q_0q_1\overline{E} + \overline{q_0q_1}E + q_0\overline{q_1}E \\ &= \overline{q_0}(q_1\overline{E} + \overline{q_1}E) + q_0(q_1\overline{E} + \overline{q_1}E) \\ &= (q_1\overline{E} + \overline{q_1}E)(\overline{q_0} + q_0) \\ &= q_1\overline{E} + \overline{q_1}E \end{aligned}$$

3.1 Dalla tabella alla matematica: la Somma di Prodotti

Per tradurre la tabella della verità in un circuito reale, utilizziamo la tecnica della "Somma dei Prodotti".

L'obiettivo è scrivere un'equazione booleana per ciascun bit di uscita del prossimo stato (q_0 e q_1). Il metodo è sistematico: si analizza la colonna dell'uscita desiderata e ci si concentra **esclusivamente sulle righe in cui il risultato è 1** (i cosiddetti *mintermini*). Per ognuno di questi casi, si scrivono le condizioni di ingresso legate da un operatore AND ($\cdot$), e infine si sommano tutti i casi validi con un operatore OR ($+$).

Prendiamo ad esempio il bit meno significativo q_1. Se analizziamo la tabella di verità, notiamo che $next\_state(q_1)$ deve valere 1 in quattro scenari specifici:

1. Quando $E = 0$ e lo stato corrente è 01 (sintetizzato come $\overline{E} \cdot \overline{q_0} \cdot q_1$)
2. Quando $E = 0$ e lo stato corrente è 11 (sintetizzato come $\overline{E} \cdot q_0 \cdot q_1$)
3. Quando $E = 1$ e lo stato corrente è 00 (sintetizzato come $E \cdot \overline{q_0} \cdot \overline{q_1}$)
4. Quando $E = 1$ e lo stato corrente è 10 (sintetizzato come $E \cdot q_0 \cdot \overline{q_1}$)

Sommando tramite OR logici questi quattro blocchi, otteniamo l'espressione di partenza. Il medesimo procedimento si applica al bit q_0.

L'importanza della semplificazione nell'ingegneria hardware

Perché è essenziale applicare i teoremi dell'algebra di Boole per semplificare queste equazioni? La risposta risiede nei vincoli del mondo fisico. Ogni operatore matematico nell'equazione corrisponde a una porta logica reale (AND, OR, NOT) da implementare su silicio. Se costruissimo la rete per q_1 usando l'equazione canonica non semplificata, avremmo bisogno di ben 12 porte AND e 3 porte OR.

Questo si tradurrebbe in un circuito costoso, che occupa un'area maggiore sul chip, consuma più potenza elettrica e introduce ritardi di propagazione dei segnali più elevati. Raccogliendo a fattor comune e semplificando l'espressione fino ad arrivare a $(q_1\overline{E} + \overline{q_1}E)$, riduciamo l'implementazione a sole 2 porte AND e 1 porta OR. Abbiamo così ottimizzato radicalmente il costo e le prestazioni dell'hardware senza alterarne minimamente la funzione logica.

4. Implementazione del Circuito

Sulla base delle funzioni minimizzate, è possibile ricavare la rete combinatoria per il calcolo del *next_state* (D_0 e D_1), che piloterà i due flip-flop D. Lo schema logico sottostante ne mostra l'architettura riordinata.

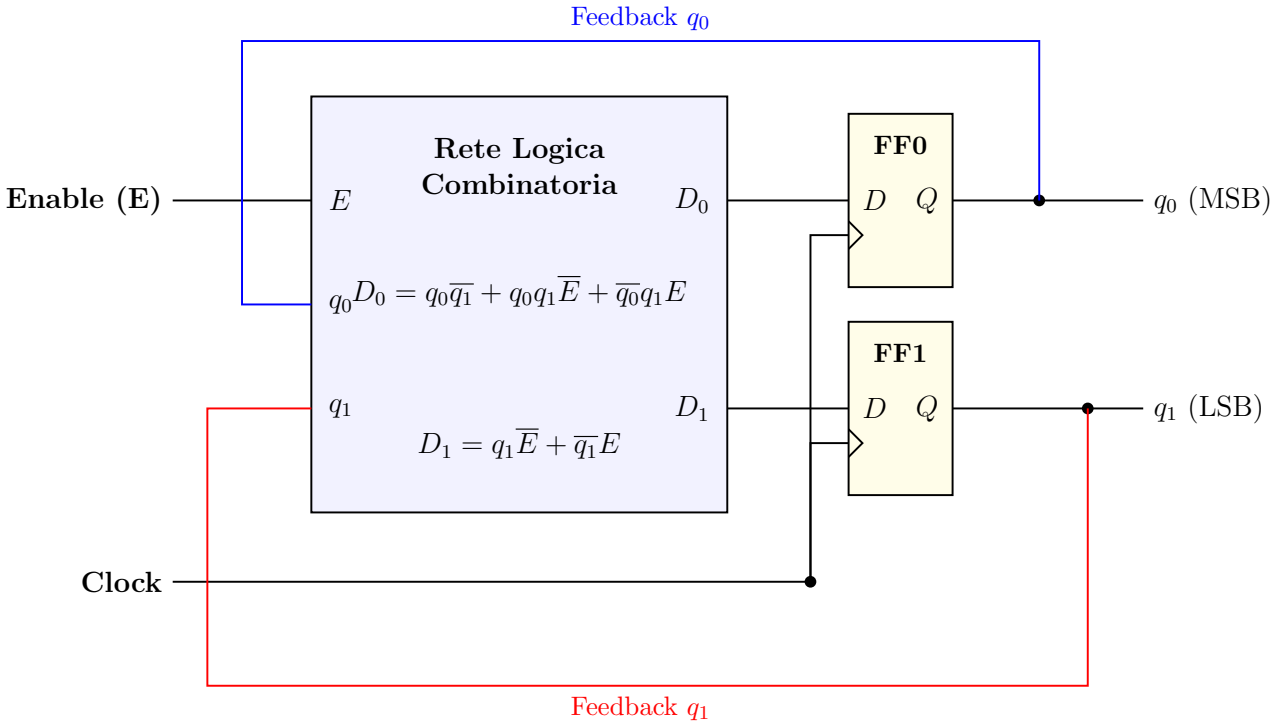

Figura 2: Schema architetturale logico del contatore circolare a 2 bit. I percorsi di feedback e il clock sono instradati esternamente per un cablaggio pulito.

4.1 Come siamo arrivati a questo circuito? Ragioniamoci insieme

Se lo costruiamo mentalmente pezzo per pezzo, vi accorgerete che è la conseguenza naturale di ciò che abbiamo appena calcolato. Facciamolo insieme, passo dopo passo.

Passo 1: Separare la "Memoria" dal "Cervello"
Nei sistemi sequenziali come questo contatore, dobbiamo sempre fare due cose: *calcolare* quale sarà il prossimo numero e *ricordare* il numero attuale. Per ricordare ci servono degli elementi di memoria. Visto che il nostro contatore produce un output a 2 bit (q_0 e q_1), ci servono esattamente due celle di memoria. Ecco perché a destra nello schema vediamo due blocchi chiamati **FF0** e **FF1** (i nostri Flip-Flop). Tutto il resto del circuito (il "cervello") serve solo a dire a questi Flip-Flop quale numero dovranno memorizzare al prossimo giro.

Passo 2: Perché usare i Flip-Flop di tipo D?
In laboratorio e negli esercizi base, il Flip-Flop D (Data) è il vostro migliore amico. La sua regola d'oro è semplicissima: *quello che metti in ingresso (D) te lo ritrovi in uscita (Q) al prossimo colpo di clock*. In formule: $Q_{next} = D$. Questo significa che le funzioni logiche che abbiamo faticosamente semplificato poco fa per lo stato successivo di q_0 e q_1 sono esattamente i segnali che dobbiamo collegare ai pin di ingresso D_0 e D_1!

Passo 3: Costruire il "Cervello" (La Rete Combinatoria)
Al centro dello schema c'è un grande rettangolo azzurro. Invece di disegnare una ragnatela incomprensibile di porte logiche AND, OR e NOT, le abbiamo "nascoste" lì dentro per tenere il disegno pulito. Cosa ha bisogno di sapere questa scatola per calcolare le formule di D_0 e D_1? Se guardate le equazioni, le variabili in gioco sono solo tre: il segnale Enable (E), q_0 e q_1. Ecco perché la scatola ha esattamente tre fili in entrata a sinistra e due fili in uscita a destra.

Passo 4: Chiudere il cerchio (Il Feedback)
Qui arriva il momento chiave, quello che trasforma una rete semplice in un sistema sequenziale.

Come fa la scatola azzurra a sapere quanto valgono q_0 e q_1 in questo preciso istante? Semplice: deve "leggerli" direttamente dalle uscite dei Flip-Flop! Ecco spiegati i fili colorati in blu e in rosso. Prendono i valori in uscita (Q) e li riportano indietro all'ingresso della rete. Questa "retroazione" (o *feedback*) è il cuore pulsante di ogni contatore: per sapere a che numero devo andare, devo prima guardare a che numero sono.

Passo 5: Sincronizzare tutti (Il Clock)
Immaginate un'orchestra senza direttore: ognuno suonerebbe quando vuole e il risultato sarebbe il caos. Il Clock è il nostro metronomo. È un segnale che fa "tic-tac" a intervalli regolari. I piccoli triangoli disegnati sui Flip-Flop indicano che essi non cambiano mai stato a caso, ma "scattano" tutti insieme, esattamente quando il segnale di Clock glielo ordina. In questo modo il passaggio da un numero all'altro è netto e perfetto.

Piccolo glossario per leggere lo schema

Per non perdervi tra le sigle, ecco un promemoria su cosa significano le lettere che vedete nel disegno:

- **FF (Flip-Flop):** I nostri mattoncini di memoria elementari, ognuno grande 1 bit.
- **D (Data):** La "porta d'ingresso" del Flip-Flop. Il valore che aspetta qui fuori entrerà nella memoria non appena scatta il clock.
- **Q (Output):** La "porta d'uscita". Mostra il valore che il Flip-Flop sta attualmente ricordando e inviando al resto del circuito.
- **E (Enable):** L'interruttore del nostro contatore. Se vale 1 diamo il "permesso" al contatore di andare avanti, se vale 0 gli diciamo di fermarsi e mantenere la posizione.
- **Clock (CLK):** Il battito cardiaco del circuito. Fa avanzare il sistema sincronizzando l'aggiornamento delle memorie.
- **MSB (Most Significant Bit):** Il "bit più significativo", ovvero quello che "pesa" di più matematicamente. In un numero binario a due cifre come q_0q_1, q_0 è la colonna dei "due".
- **LSB (Least Significant Bit):** Il "bit meno significativo", ovvero la colonna delle unità (q_1).