

ESERCITAZIONE 7

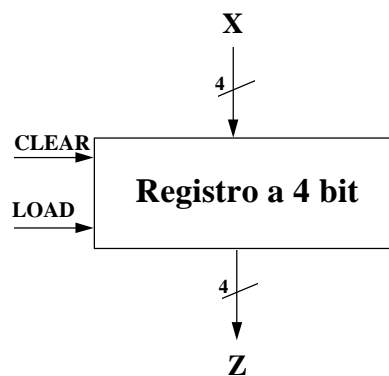
Sommario

- Linguaggio di descrizione a livello RT
- Progetto a livello RT

1. Livello a trasferimento tra registri (segue)

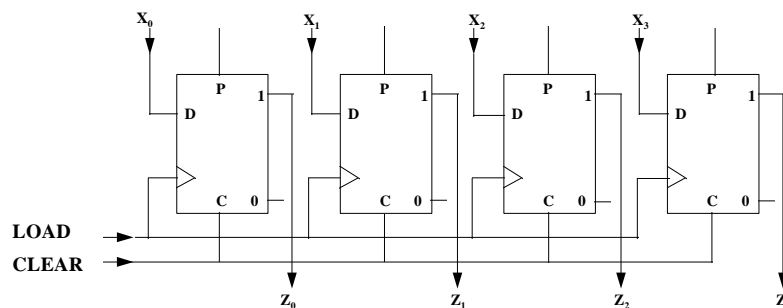
1.1. Registri

Gli elementi di memoria a livello RT sono i registri. I registri sono costituiti da elementi di memoria (Flip-Flop) collegati per memorizzare ed operare su un insieme di bit.



Il segnale di *clear* porta a zero le uscite in modo asincrono rispetto al segnale di *clock*.

Il segnale di *load* permette il caricamento dei dati nel registro.



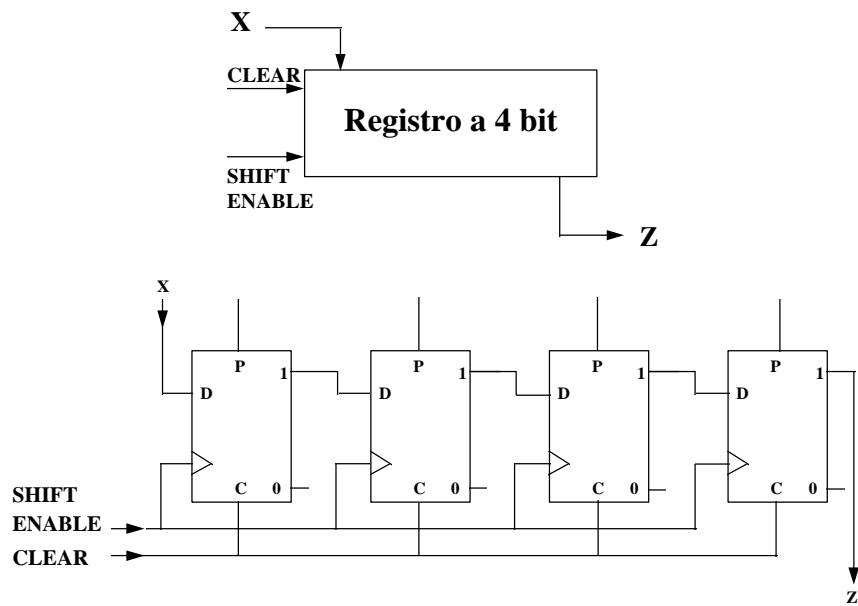
Tale registro è caratterizzato da avere un ingresso parallelo a 4 bit ed un'uscita parallela a 4 bit.

Dando un colpo di clock i valori in ingresso vengono memorizzati in uscita ai flip-flop. I dati vengono campionati in ingresso in presenza di un segnale di *clock*.

1.1.1. Registri a scalamiento

I dati vengono inseriti serialmente a partire dall'ingresso *X*; dopo un numero di colpi di clock pari al numero di flip-flop, i dati sono immagazzinati nel registro.

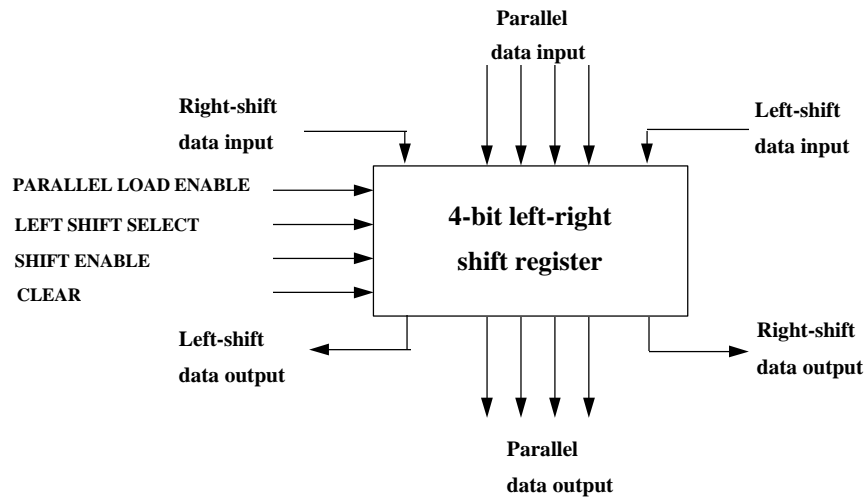
Il segnale di *shift enable* permette lo scorrimento dei dati da sinistra a destra.



1.1.2. Registro generalizzato

Svolge le seguenti funzioni:

1. ingresso parallelo;
2. ingresso seriale;
3. uscita parallela;
4. uscita seriale.

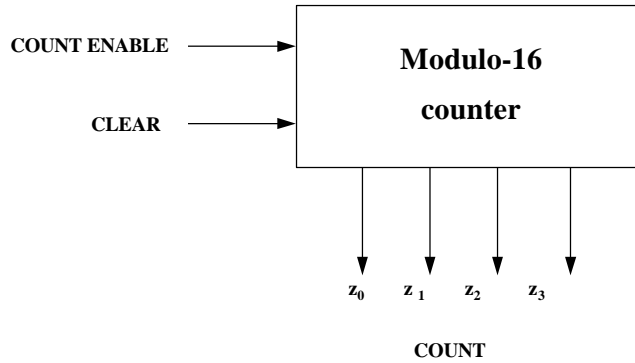


Usi dei registri ascalamento:

- memorizzazione di dati seriali (FIFO);
- conversione seriale-parallelo e parallelo-seriale;
- moltiplicazione e divisione su numeri infixed-point senza segno e con segno (notazione).

1.2. Contatori

I contatori sono moduli che hanno come solo ingresso il segnale di clock. I contatori ciclano attraverso k stati in risposta ad impulsi in ingresso; la codifica degli stati permette di contare il numero di impulsi.

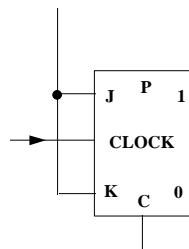


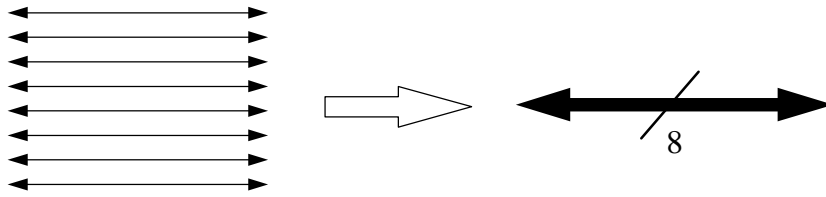
1.2.1. Usi dei contatori

- Come *Program Counter* in una macchina a stati;
- per la generazione di segnali di temporizzazione (*divisori di frequenza*);
- come contatori di eventi.

1.2.2. Contatore asincrono

Tali contatori sono detti asincroni perché il segnale di clock è generato internamente.



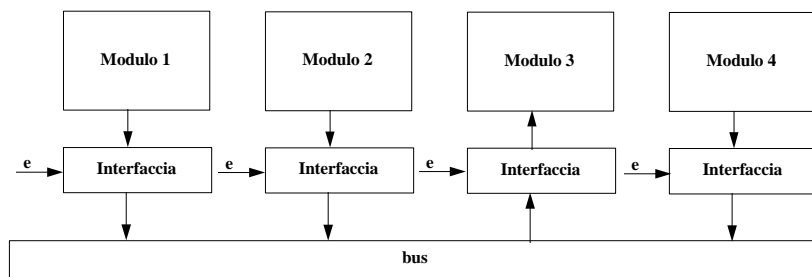


Il costo di un bus può essere elevato in termini di area di silicio (se connette moduli distanti sullo stesso IC) e dipin (se è accessibile dall'esterno del chip).

1.3.1. Connessione al bus

Ad ogni istante un solo dispositivo tra quelli i cui output sono connessi al bus deve pilotarne il valore.

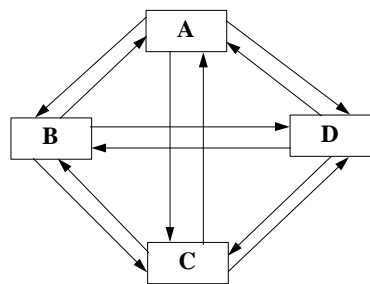
Gli altri devono assumere un valore particolare noto come Z (alta impedenza).



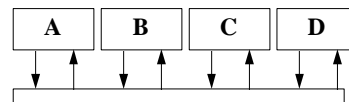
1.3.2. Tipologie di bus

Possono essere:

- *dedicati* (alto costo, elevate prestazioni);
- *condivisi* (basso costo, basse prestazioni): richiedono un meccanismo di regolamentazione degli accessi (*arbitro*).



Bus Dedicato



Bus Condiviso

2. Progetto a livello RT

2.1. Flusso di progetto

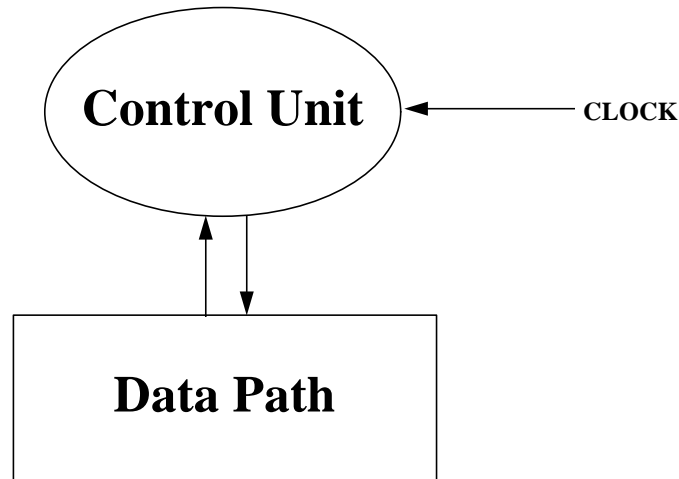
Architettura a livello RT è costituita da 2 tipi di descrizione:

1. struttura del sistema (interconnessione dei componenti);
2. comportamento (programma).

Il risultato di un progetto a livello register è un insieme di componenti connessi opportunamente (*Data-Path*) pilotati dai segnali provenienti da una Unità di Controllo (*Control Unit*).

La struttura è dunque divisa in 2 parti:

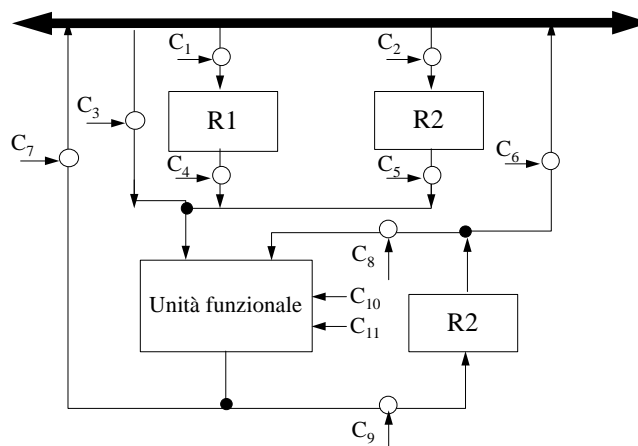
1. Unità Dati (*Data-Path*) che esegue direttamente le operazioni;
2. Unità di Controllo (*Control Unit*) che gestisce l'unità dati ed è definita a partire dal comportamento definito dalle specifiche del progetto.



- Descrivere il comportamento di quanto si vuole progettare utilizzando un linguaggio di descrizione *register transfer*.
- Costruire un diagramma a blocchi del *Data Path* istanziando un blocco per ogni operazione che appare nella descrizione, e connettendo opportunamente i blocchi.
- Definire i segnali di controllo necessari per:
 - abilitare opportunamente i blocchi funzionali
 - realizzare le connessioni corrette.

Progettare la *Control Unit*.

- Ottimizzare il progetto eliminando i duplicati.



Le connessioni non sono attive tutte contemporaneamente: l'unità di controllo seleziona i cammini che devono essere attivi per svolgere determinate operazioni (trasferimenti).

2.2. Linguaggi di descrizione a livello RT

Servono per descrivere un progetto per vari scopi:

1. documentazione
2. simulazione
3. verifica
4. valutazione delle prestazioni
5. sintesi.

Sono detti *Register Transfer Language* in quanto il loro elemento essenziale è l'istruzione di tipo register transfer:

$$X \leftarrow f(X_1, X_2, \dots, X_n)$$

Tali linguaggi hanno spesso notevoli somiglianze con i linguaggi di programmazione (ad es. hanno il costrutto condizionale).

Si veda ora un esempio di linguaggio di descrizione a livello RT:

- dichiarazione di registri: **declare register** A (0:7), B (0:7), count (0:2)
- trasferimento tra registri: COUNT ← 0
- istruzioni di controllo: **IF** COUNT ≠ 7 **THEN GOTO** ADD
- operazioni contemporanee: A ← 0, COUNT ← 0
- operazioni sequenziali: OUTBUS ← Q; OUTBUS ← A.

2.3. Esempio di progetto

Si desidera progettare un moltiplicatore per numeri fixed-point su 8 bit rappresentati in modulo e segno.

$$\text{Rappresentazione: } X = X_0 X_1 X_2 X_3 X_4, N = \sum_{i=1}^7 x_i 2^{-i}$$

- Modulo: $P_M \leftarrow X_M \times Y_M$,
- Segno: $p_0 \leftarrow x_0 \text{ EXOR } y_0$

Il calcolo di P_M avviene con un algoritmo basato su 7 passi di somma e addizione. Ad ogni passo le operazioni eseguite sono:

$$P_i \leftarrow P_i + x_{7-i} \times Y$$

$$P_{i+1} \leftarrow 2^{-1} P_i$$

La prima operazione equivale a sommare a P_i il valore di Y oppure 0, a seconda del valore di x_i .

La seconda operazione equivale a shiftare a destra di una posizione il valore di P

Le condizioni iniziali sono

$$P_0 = 0$$

Al termine

$$P_M = P_7$$

Esempio di operazione:

X_M : 1011

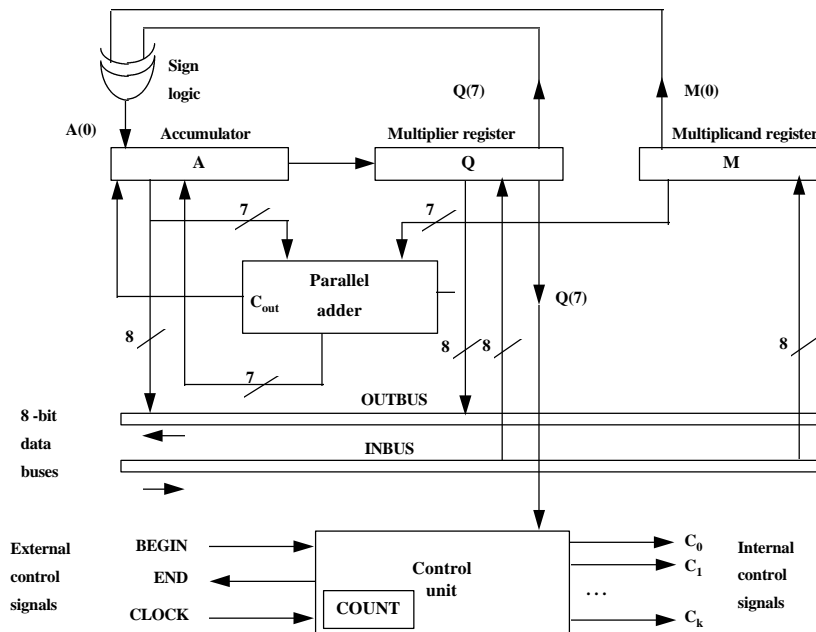
Y_M : 0101

<i>Passo</i>	<i>Operazione</i>	<i>Risultato</i>
0	Inizializzazione	00000000
1	$P_1 = P_0 + X_4 * Y$	10110000
2	$P_2 = P_1 / 2$	01011000
3	$P_3 = P_2 + X_3 * Y$	01011000
4	$P_4 = P_3 / 2$	00101100
5	$P_5 = P_4 + X_2 * Y$	11011100
6	$P_6 = P_5 / 2$	01101110
7	$P_7 = P_6 + X_1 * Y$	01101110
8	$P_8 = P_7 / 2$	00110111

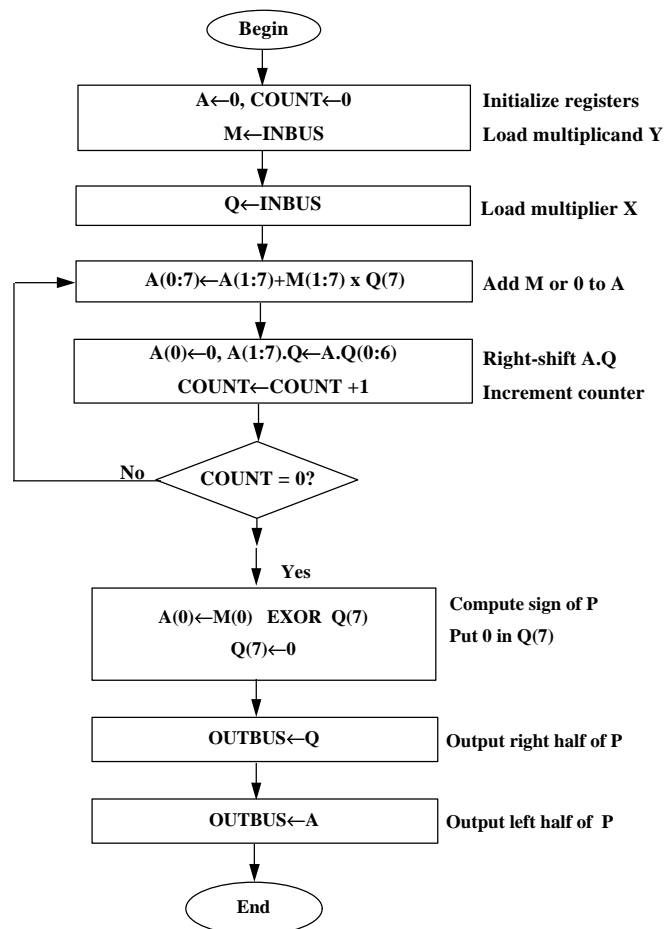
2.3.1. Progetto del DataPath

Moduli necessari:

- 1 registro ad 8 bit (Q: moltiplicatore X)
- 1 registro ad 8 bit (M: moltiplicando Y)
- 1 registro a 16 bit (A: prodotto P)
- 1 sommatore a 7 bit
- Porta Exor



2.3.2. Descrizione comportamentale



2.3.3. Programma in linguaggio RT

declare register A(0:7), M(0:7), Q(0:7), COUNT(0:2)

declare bus INBUS(0:7), OUTBUS(0:7)

BEGIN: A ← 0, COUNT ← 0;

```

INPUT:      M ← INBUS,
           Q ← INBUS;
ADD:        A(0:7) ← A(1:7)+M(1:7)×Q(7);
RIGHTSHIFT: A(0) ← 0,A(1:7).Q ← A.Q(0:6),
TEST:       COUNT ← COUNT+1;
           IF COUNT ≠ 7 THEN GOTO ADD,
FINISH:     A(0) ← M(0) EXOR Q(7), Q(7) ← 0;
OUTPUT:     OUTBUS ← Q;
           OUTBUS ← A;
END;

```

2.4. Esercizio

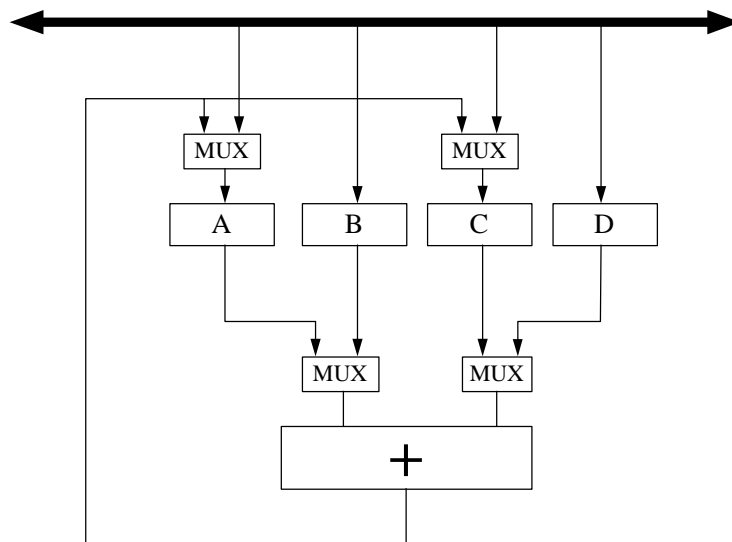
Si supponga di dover eseguire la seguente operazione

$$A \leftarrow A+B$$

$$C \leftarrow C+D$$

2.4.1. Soluzione

Si utilizza 1 solo sommatore e le somme vengono effettuate in sequenza, connettendo il sommatore prima ai registri A e B, poi ai registri C e D.



```

declare register A(0:7), B(0:7), C(0:7), D(0:7)

```

```

declare bus INBUS A(0:7)

```

```

BEGIN:      A ← INBUS;

```

```

           C ← INBUS;

```

```

CICLO:     B ← INBUS;

```

```

           D ← INBUS;

```

```

           A ← A + B;

```

```

           C ← C+D;

```

GOTO CICLO

END;

[Torna al Sommario](#)