Traccia di progetto: Interfaccia IFRANGE[1]

Un'interfaccia IFRANGE è connessa a un bus esterno $\texttt{EXTDATA}_{0-15}$ su cui transitano dati numerici paralleli sincronizzati a un clock esterno `EXTCLK`. IFRANGE riceve dalla CPU PD32 due valori numerici $V_{\min}$ e $V_{\max}$, quindi avvia l'acquisizione di una sequenza di 1024 dati `EXTDATA`, determina il numero N di valori della sequenza compresi nell'intervallo $[V_{\min},V_{\max}]$, e trasmette tale risultato alla CPU. Quest'ultima confronta il valore ricevuto di N con un valore N_0 di soglia e:

- decrementa $V_{\min}$ e incrementa $V_{\max}$ se $N < N_0$
- incrementa $V_{\min}$ e decrementa $V_{\max}$ se $N > N_0$

dopo di che riavvia il processo.

Progettare l'interfaccia IFRANGE e codificare il relativo software di pilotaggio.

* * * * * * * *

L'architettura dell'interfaccia può essere partizionata in blocchi funzionali (Fig. 1); il progetto richiesto è peraltro molto simile come struttura a quello descritto nel documento *Esempio di progetto: Interfaccia IFMAX*, al quale si rimanda per una discussione generale e per l'implementazione dei moduli

- condizionamento dati di input,
- sottosistema di controllo,
- interfaccia PD32 I/O bus.

Esamineremo qui dunque soltanto l'organizzazione del sottosistema di calcolo e quella parte del sottosistema di controllo specifica per il problema dato.

1 Acquisizione di $V_{\min}$ e $V_{\max}$

I valori $V_{\min}$ e $V_{\max}$ devono essere forniti dalla CPU prima che il processo di calcolo venga avviato; la maniera più naturale affinché la CPU possa trasmettere questi due valori all'interfaccia IFRANGE consiste nel prevedere su questa due registri e renderli accessibili alla CPU come porte di output, in maniera che la CPU vi possa accedere mediante istruzioni di `OUT`. È chiaro che a ciascuno dei registri dovremo associare un *diverso* device address, in caso contrario ad ogni `OUT` sul device address la CPU sovrascriverebbe l'informazione precedente. Ne consegue che l'interfaccia deve generare due distinti segnali di `SELECT`, che chiameremo `SEL0` e `SEL1`, associati rispettivamente a $V_{\min}$ e a $V_{\max}$. I due registri in questione ricevono gli ingressi dati dall'I/O Data Bus `IODB` e presentano in uscita i valori `VMIN` e `VMAX` da utilizzare come parametri; il loro clock è condizionato al segnale di bus `IOWR` mediante i due segnali di selezione citati (Fig. 2).

[1] Prima prova d'esame di Calcolatori Elettronici II, sessione estiva, appello del 9 luglio 2002.

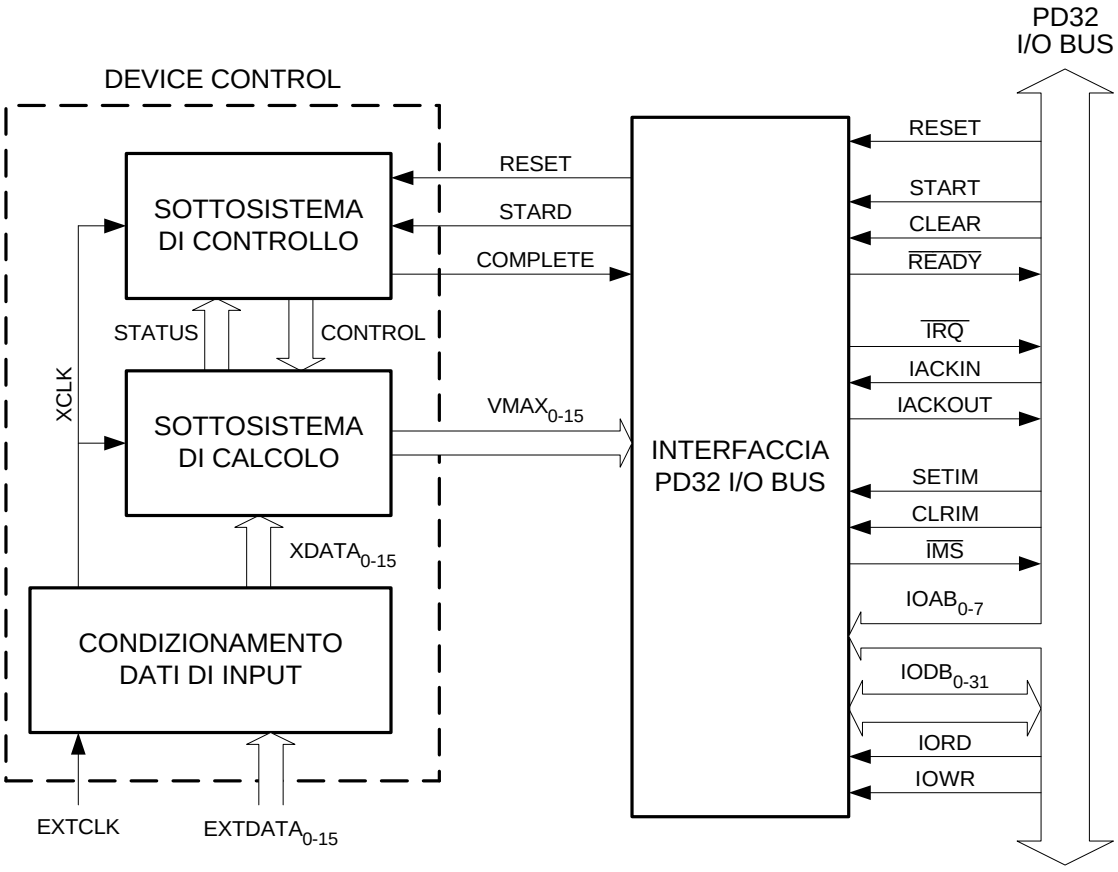

Fig. 1: Partizionamento di IFRANGE in blocchi funzionali.

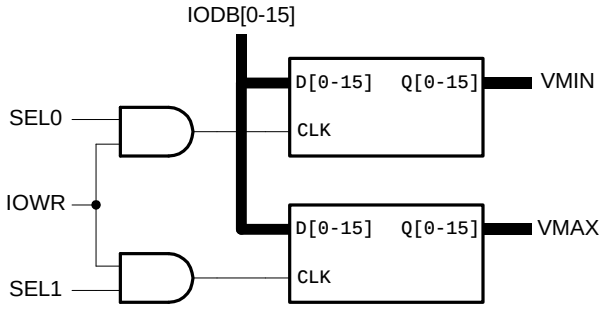

Fig. 2: Porte di output per l'acquisizione di $V_{\min}$ e $V_{\max}$.

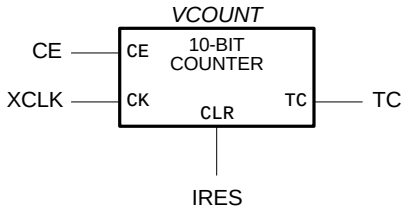

Fig. 3: Circuito di conteggio dei dati.

2 Conteggio dei dati

Le specifiche richiedono che il processo di elaborazione operi su una sequenza di 1024 valori `XDATA`; a tal fine è sufficiente predisporre un contatore modulo 1024 (dunque a 10 bit) la cui uscita Terminal Count (`TC`) ci informerà della fine della sequenza. Il contatore, che chiameremo VCOUNT, dovrà contare da 0 all'inizio del processo, e dunque il suo ingresso di Reset (`CLR`) dovrà essere connesso al segnale di Internal Reset (`IRES`) prodotto dal SCO[2], mentre dovrà essere abilitato al conteggio (`CE` $= 1$) solo durante le normali operazioni. Il circuito è illustrato in Fig. 3.

3 Confronto tra i dati

Poiché il testo del problema nulla specifica circa il formato dei dati, possiamo assumere che i valori in gioco siano assoluti; di conseguenza la comparazione può essere senz'altro realizzata facendo ricorso a comparatori standard. Dovendo realizzare due distinti confronti in un unico intervallo di clock `XCLK`, avremo bisogno di due distinti comparatori:

- COMPMIN, che produca un'uscita attiva quando $V \geq V_{\min}$
- COMPMAX, che produca un'uscita attiva quando $V \leq V_{\max}$

Tali uscite andranno poi combinate in AND per indicare l'esito positivo contemporaneo dei due confronti; inoltre, per disattivare il confronto al di fuori dalle normali operazioni, è sufficiente condizionare l'uscita generale `INRANGE` con il medesimo segnale `CE` che abilita il conteggio dei 1024 valori della sequenza (Fig. 4).

L'uscita `INRANGE` sarà dunque alta quando l'interfaccia si trova nello stato di normale operazione e quando nel contempo il valore V presente su `XDATA` è compreso nell'intervallo $[V_{\min}, V_{\max}]$.

4 Conteggio dei confronti positivi

Il conteggio dei confronti con esito favorevole si realizza semplicemente inviando l'uscita `INRANGE` dal circuito di comparazione all'ingresso di abilitazione di un contatore NCOUNT avente come

[2] Si veda il documento *Esempio di progetto: Interfaccia IFMAX* per i relativi dettagli.

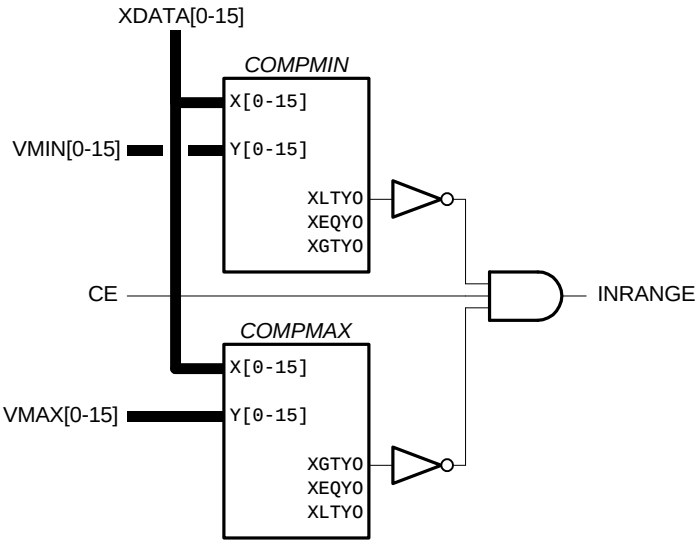

Fig. 4: Circuito di comparazione.

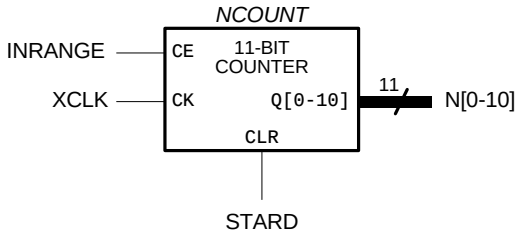

Fig. 5: Circuito per il conteggio dei confronti favorevoli.

clock il clock interno `XCLK`. Poiché possiamo avere da 0 a 1024 confronti positivi, NCOUNT dovrà poter produrre anche il conteggio 1024, e dunque dovrà essere esteso su 11 bit (10 non bastano!). Inoltre, se evitiamo che venga resettato da `IRES`, esso può anche fungere da registro di uscita, nel qual caso è sufficiente resettarlo con il segnale di `STARD` che avvia le operazioni (possiamo evidentemente presumere che, se la CPU avvia un nuovo processo, abbia già letto il valore N calcolato nel processo precedente). Il circuito relativo è illustrato in Fig. 5.

5 Lettura del risultato

La CPU leggerà il risultato N attraverso una porta di input, alla quale sarà associato un device address che potrà essere diverso dai due (`SEL0` e `SEL1`) associati ai registri `VMIN` e `VMAX`, ovvero, per ragioni di semplicità circuitale nella decodifica, potrà anche coincidere con uno di essi — diciamo con `SEL0`. Come per ogni porta di input, le uscite di NREG si attestano sull'I/O DataBus `IODB` attraverso un buffer tri-state multiplo controllato dal segnale di I/O Control

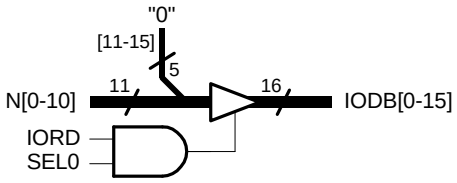

Fig. 6: Circuito per la lettura del risultato.

Bus `IORD` e da `SEL0`; il buffer tri-state dovrà essere esteso ad almeno 16 bit mediante zeri, dal momento che il valore N è un numero assoluto (Fig. 6).

6 Controllo delle operazioni

La macchina a stati finiti (FSM) per il controllo di IFRANGE è costituita semplicemente da:

- uno **stato di riposo** (R), in cui VCOUNT viene forzato al reset;
- uno **stato di normale operazione** (N), in cui VCOUNT è abilitato al conteggio mentre NCOUNT lo è solo quando il confronto ha esito positivo;
- uno **stato di completamento** (C), instabile[3], nel quale si entra quando VCOUNT raggiunge il conteggio terminale, e che produce il segnale `COMPLETE` di fine operazioni.

Il corrispondente diagramma degli stati è illustrato in Fig. 7.

La FSM dovrà essere implementata con due flip-flop, ad esempio di tipo D; assegnata una codifica di stato arbitraria, come quella illustrata in figura, i circuiti di eccitazione devono soddisfare alla seguente tavola di verità:

Stato presente	Q_1Q_0	`TC`	Stato futuro	D_1D_0	Note
R	00	*	N	01	uscita dallo stato di riposo
N	01	0	N	01	normale operazione
	01	1	C	10	fine operazioni
C	10	*	?	**	stato di completamento (instabile)
?	11	*	?	**	stato inesistente

Si noti come, essendo lo stato C instabile, il suo stato futuro venga indicato come non specificato, al fine di semplificare i circuiti di eccitazione; per la stessa ragione, anche allo stato inesistente 11 viene assegnato uno stato futuro non specificato. I circuiti di eccitazione possono allora essere realizzati come illustrato in Fig. 8.

Gli unici segnali di controllo che la FSM deve generare sono:

- `CE`, che va attivato solo nello stato N di normale operazione,

[3] Si veda il documento *Esempio di progetto: Interfaccia IFMAX* per la relativa discussione e i dettagli di implementazione.

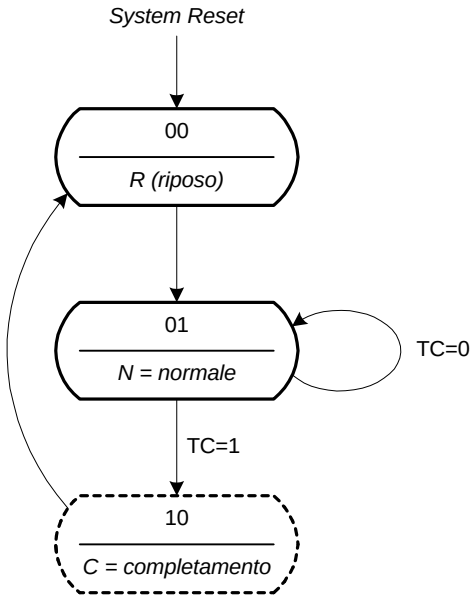

Fig. 7: Diagramma degli stati della FSM di controllo.

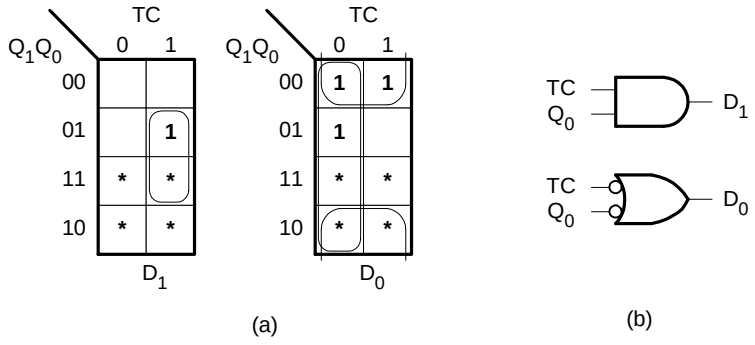

Fig. 8: Circuiti di eccitazione per la FSM di controllo: (a) mappe di Karnaugh, (b) realizzazione con porte logiche.

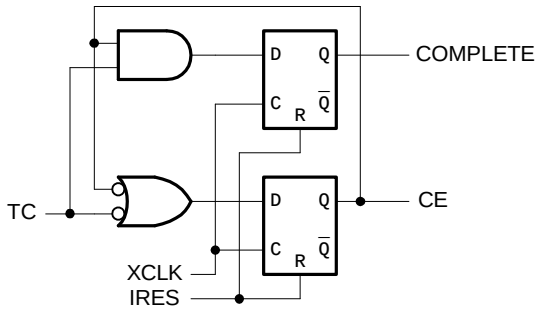

Fig. 9: Circuito della FSM di controllo.

- `COMPLETE`, che va attivato solo nello stato C di completamento,

secondo la seguente tavola di verità:

Stato presente	Q_1Q_0	`CE`	`COMPLETE`
R	00	0	0
N	01	1	0
	01	1	0
C	10	0	1
?	11	*	*

e, senza necessità di ricorrere alle mappe di Karnaugh, si vede facilmente che è `CE` $= Q_0$ e `COMPLETE` $= Q_1$. Il circuito finale per la FSM di controllo è allora quello illustrato in Fig. 9.

7 Software di pilotaggio

Assumendo che i valori $V_{\min}$, $V_{\max}$ e N_0 siano allocati in altrettante variabili a 16 bit VMIN, VMAX ed N0:

```
VMIN    DW    ???    ; valore iniziale di Vmin
VMAX    DW    ???    ; valore iniziale di Vmax
N0      DW    ???    ; valore iniziale di N0
```

e che i device address necessari siano definiti come costanti arbitrarie:

```
DEV0    EQU   30h    ; device address per Vmin, N0 e START
DEV1    EQU   31h    ; device address per Vmax
IVN     EQU   5      ; interrupt vector number per IFRANGE
```

il software di pilotaggio può comodamente essere partizionato in una *subroutine di restart*:

```
RESTART:
    MOVW  VMIN,R0    ; trasmette Vmin a IFRANGE
    OUTW  R0,DEV0
```

```
    MOVW  VMAX,R0     ; trasmette Vmax a IFRANGE
    OUTW  R0,DEV1
    START DEV0        ; avvia le operazioni
    RET
```

e in una *subroutine di acquisizione ed elaborazione*:

```
PROCESS:
    INW   DEV0,R0     ; acquisisce N
    CMPW  N0,R0       ; confronta N con N0
    JZ    NEQ         ; salta se N != N0
    RET               ; altrimenti ritorna
NEQ:
    MOVW  VMIN,R1     ; carica Vmin
    MOVW  VMAX,R2     ; carica Vmax
    JN    NLT         ; salta se N < N0
NGT:
    ADD   #1,R1       ; incrementa Vmin
    SUB   #1,R2       ; decrementa Vmax
    J     UPDATE      ; salta all'aggiornamento
NLT:
    SUB   #1,R1       ; decrementa Vmin
    ADD   #1,R2       ; incrementa Vmax
UPDATE:
    MOVW  R1,VMIN     ; aggiorna Vmin
    MOVW  R2,VMAX     ; aggiorna Vmax
    RET               ; ritorna al chiamante
```

Se il protocollo di I/O è di tipo busy-waiting, utilizzeremo allora il seguente schema di programma:

```
AGAIN:
    JSR   RESTART     ; emette i parametri e avvia le operazioni
LOOP:
    JNR   DEV0        ; aspetta il completamento
    JSR   PROCESS     ; processa il risultato e aggiorna i parametri
    J     AGAIN       ; ripete la procedura
```

Se invece il protocollo di I/O è basato su interrupt, avremo:

```
    JSR   RESTART     ; emette i parametri e avvia le operazioni
    ........
    DRIVER IVN,IFRANGE_ISR
    PUSH  R0          ; salva i registri interessati
    PUSH  R1
    PUSH  R2
    JSR   PROCESS     ; processa il risultato e aggiorna i parametri
```

```
JSR   RESTART    ; emette i parametri e avvia una nuova operazione
POP   R2         ; ripristina i registri interessati
POP   R1
POP   R0
RTI              ; ritorna dalla routine di interrupt
```