Metodo di progetto RTL

Dalle macchine a stati ai sistemi complessi

Cosa non sappiamo fare

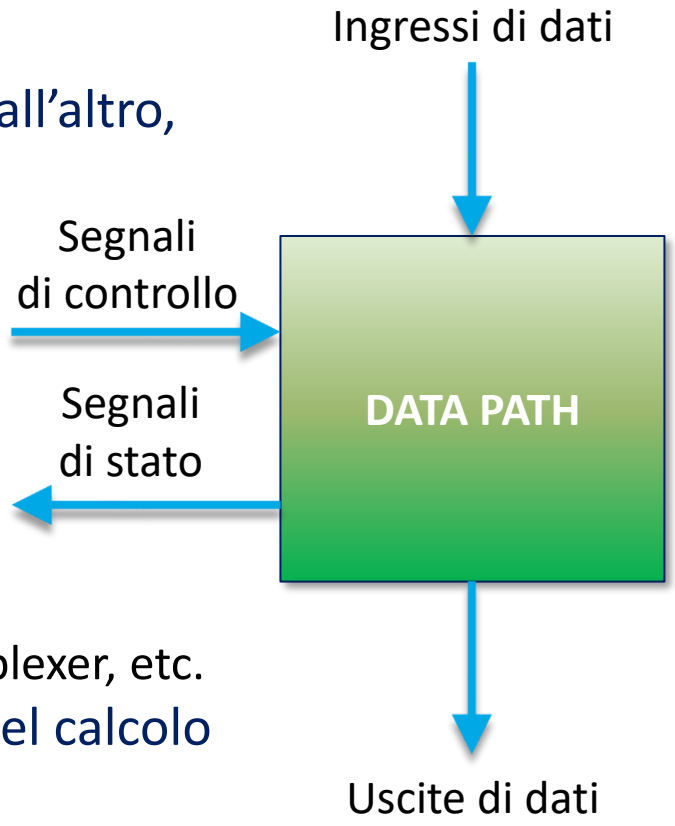
- **Gestire la complessità**
  - Abbiamo visto il progetto gerarchico e iterativo
  - Molto utile ma poco strutturato
  - Cerchiamo di generalizzare
- **Qual è in generale il problema?**
  - Come esseri umani siamo in grado di trattare poche cose alla volta
  - Nelle macchine sequenziali siamo limitati a poche decine di stati
  - Ma un semplice registro a 8 bit introduce già 256 possibili stati
  - Occorre in qualche modo trattare questi stati in maniera implicita
  - Utile utilizzare blocchi già fatti
  - Occorre però definirne con precisione l'interazione con tali blocchi

Data path e controllo

- **La complessità introdotta dai registri è solo apparente**
  - E' vero che introducono potenzialmente migliaia di stati
  - Ma generalmente non concorrono nella *logica* del funzionamento
  - Il *flusso* delle operazioni rimane generalmente pressoché indipendente, o dipendente in minima parte, dai valori dei registri
- **Separation of concerns**
  - Cerchiamo allora di separare la parte circuitale che si occupa di trattare i dati → **data path**
  - Da quella che si occupa di gestire le relative operazioni secondo un flusso logico → **unità di controllo**

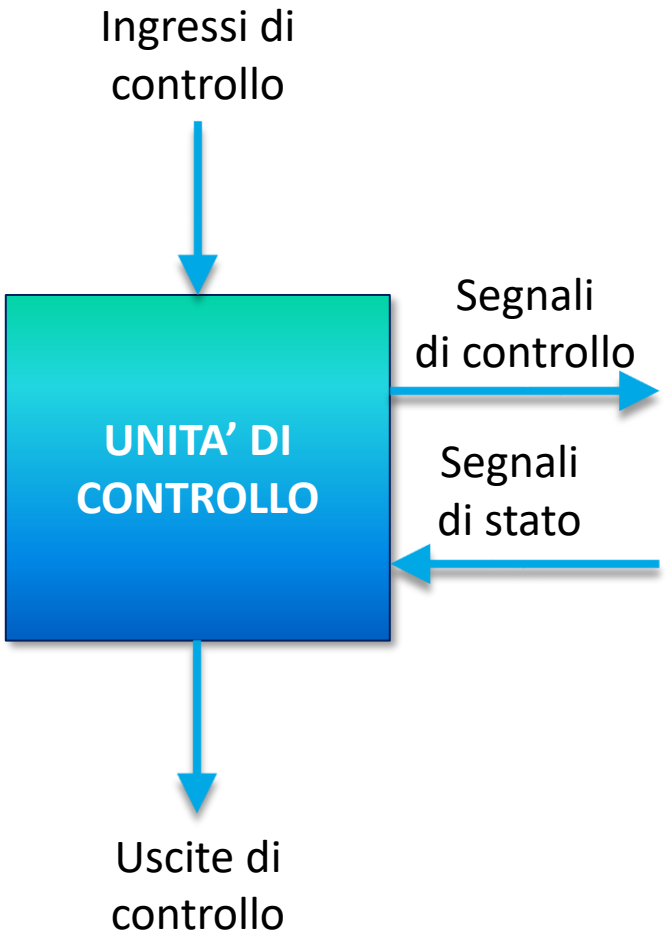
Data path

- **Contiene la parte di calcolo**
  - Circuiti combinatori aritmetici e logici
  - Registri e contatori per memorizzare i dati
  - Circuiti per indirizzare i dati da un registro all'altro, e dai registri alla logica di calcolo
- **Interfaccia**
  - Ingressi di dati (sensori, linee di comunicazione, memorie)
  - Uscite di dati (attuatori, etc.)
  - Segnali di controllo per gestire l'operatività dei componenti
    - Segnali di enable dei contatori, configurazioni delle reti di calcolo, dei multiplexer, etc.
  - Segnali di stato per indicare le condizioni del calcolo
    - Terminal count di un contatore, carry e overflow, uscite dei comparatori, etc.

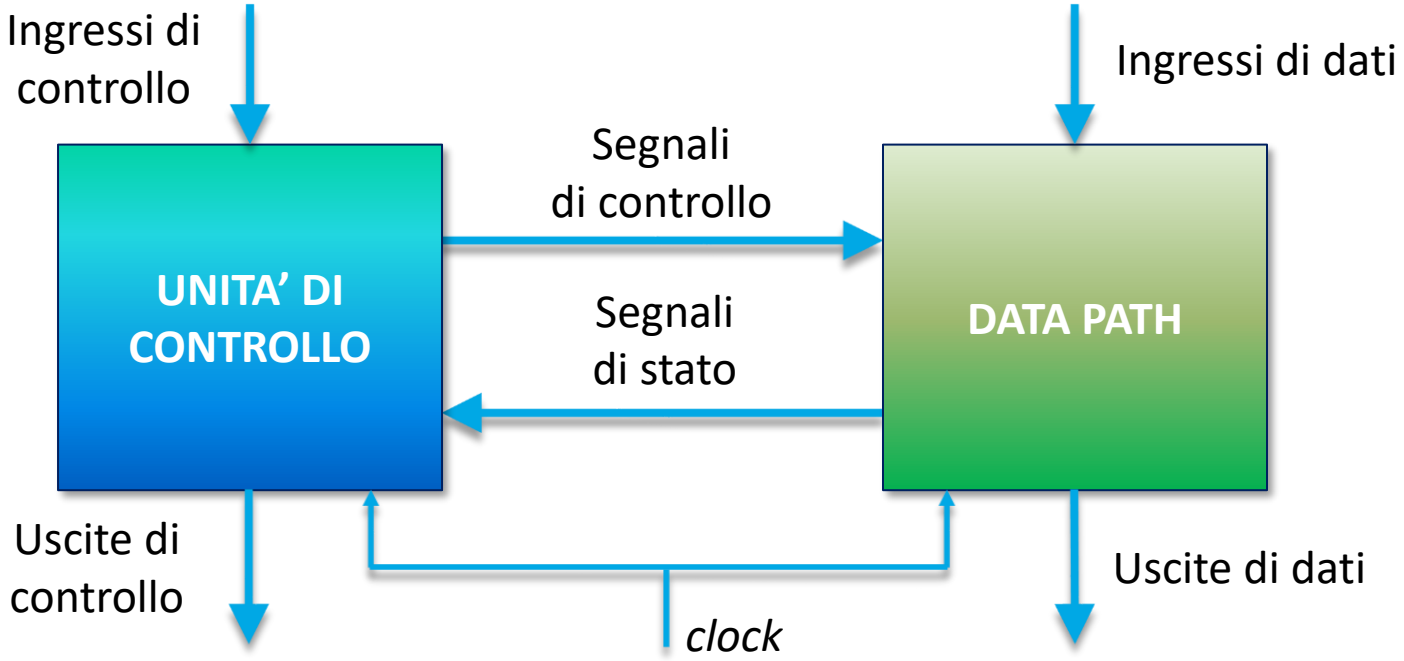


Unità di controllo

- **Contiene la parte di controllo di flusso**
  - Essenzialmente una macchina a stati
  - Esprime la logica di funzionamento del circuito
  - Astrae dai dati per contenere la complessità
  - Mantiene il solo stato necessario alla sequenzializzazione
- **Interfaccia**
  - Ingressi di controllo (bottoni, linee logiche)
  - Uscite di controllo (spie, etc.)
  - Segnali di controllo per gestire l'operatività del data path
  - Segnali di stato per prendere decisioni sul flusso operativo



Struttura di sistema



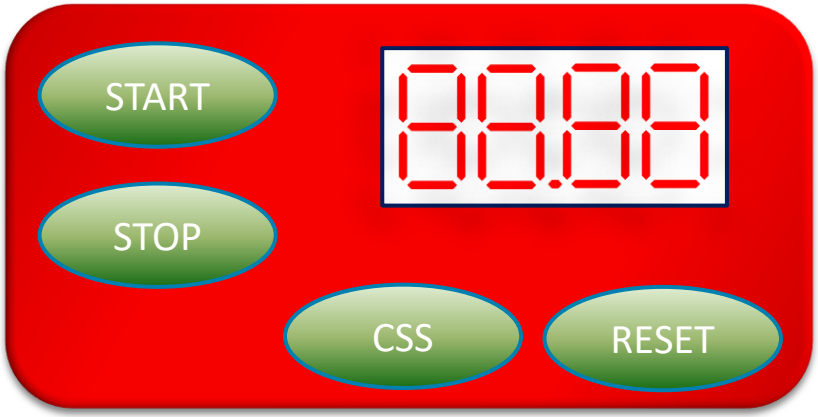
- **Entrambi i blocchi sono sequenziali, sincronizzati dallo stesso clock**
  - Non è la divisione tra parte combinatoria e sequenziale!!
  - Abbiamo invece separato concettualmente diverse prerogative
  - L'unità di controllo decide cosa fare (segnali di controllo) in base ai risultati (segnali di stato)
  - Il data path esegue le operazioni e memorizza i dati

Procedura di progetto

- **Occorre definire cosa sono i dati (variabili)**
  - Definire i registri nel data path per memorizzare i dati
- **Occorre determinare il tipo di operazioni necessarie**
  - Trasferimenti tra registri
  - Calcoli aritmetici
  - Operazioni logiche sui dati
- **Definire i segnali di controllo e di stato**
  - Necessari per determinare il tipo di operazione da svolgere
  - Per capire come fare evolvere l'elaborazione
- **Progettare il controllo**
  - Sviluppare una macchina a stati secondo la logica di specifica
- **Progetto del data path**
  - Diagramma dettagliato del circuito

Esempio

- **Si progetti un cronometro per corsa, con precisione dei centesimi di secondo, da 00.00 a 99.99 secondi**
  - Un pulsante START resetta a 0 il cronometro e fa partire il conteggio
  - Un pulsante STOP ferma il conteggio
  - Il conteggio viene mostrato su un display a 7 segmenti a 4 cifre
  - Un pulsante RESET inizializza il cronometro
- **Funzione giro veloce**
  - Un pulsante CSS (compare and store shortest) confronta il conteggio attuale con un valore memorizzato
    - Mostra il più piccolo dei due
    - Memorizza il più piccolo dei due



Identificazione di ingressi e uscite

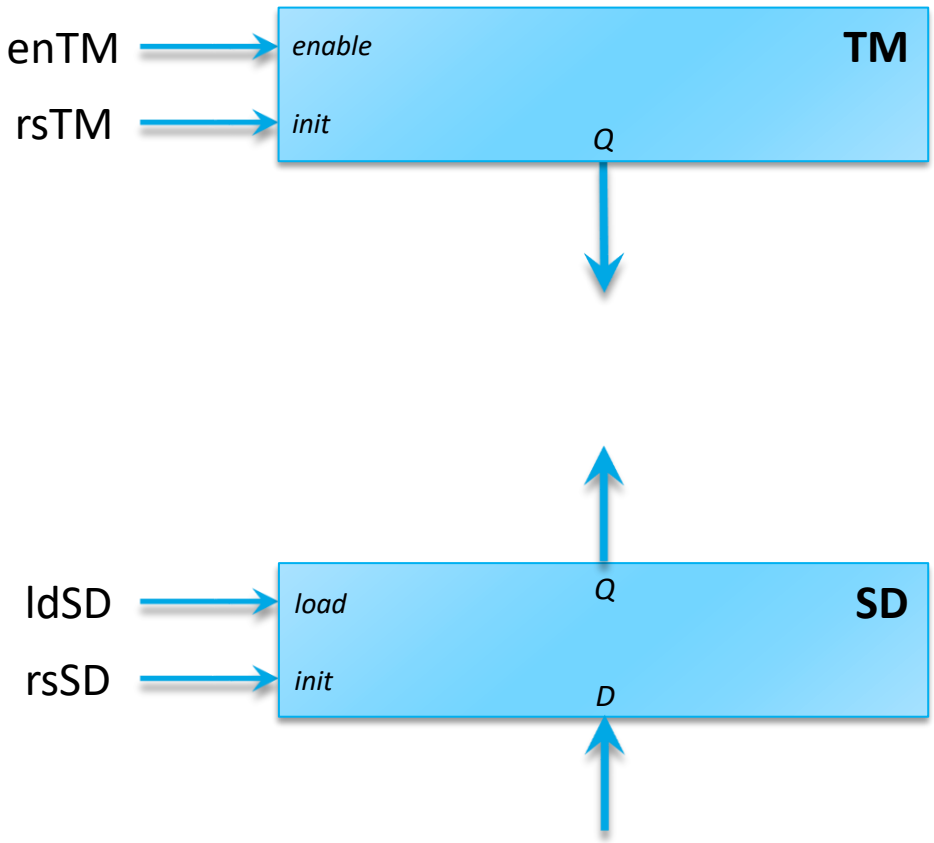
Simbolo	Funzione	Tipo
START	Inizializza e fa partire il conteggio	Ingresso controllo
STOP	Ferma e mostra conteggio	Ingresso controllo
CSS	Confronta, memorizza e mostra il tempo più breve	Ingresso controllo
RESET	Inizializza il tempo più breve	Ingresso controllo
B_1	Vettore display 1 (*a, b, c, d, e, f, g*)	Uscita dati
B_0	Vettore display 0 (*a, b, c, d, e, f, g*)	Uscita dati
B_{-1}	Vettore display -1 (*a, b, c, d, e, f, g*)	Uscita dati
B_{-2}	Vettore display -2 (*a, b, c, d, e, f, g*)	Uscita dati
PD	Punto decimale	Uscita dati

- **Non ci sono uscite di controllo o ingressi di dati**
  - Totale di 4 ingressi di controllo e 7*4+1 = 29 uscite di dato

Registri del data path

- **E' necessario un contatore**
  - O un contatore binario che possa contare da 0 a 9999 (minimo 14 bit, da 0 a16535)
  - Oppure un contatore BCD a 4 cifre (16 bit)
  - Deve anche poter essere inizializzato a 0
  - Si deve controllare l'abilitazione al conteggio per poterlo fermare
  - Scegliamo il BCD per non complicare poi la decodifica
- **Occorre memorizzare il tempo migliore**
  - Possiamo usare un registro della stessa dimensione del contatore
  - Deve poter caricare un valore arbitrario, quindi deve essere un registro parallelo con load enable
  - Deve poter essere inizializzato (si può usare il caricamento parallelo)

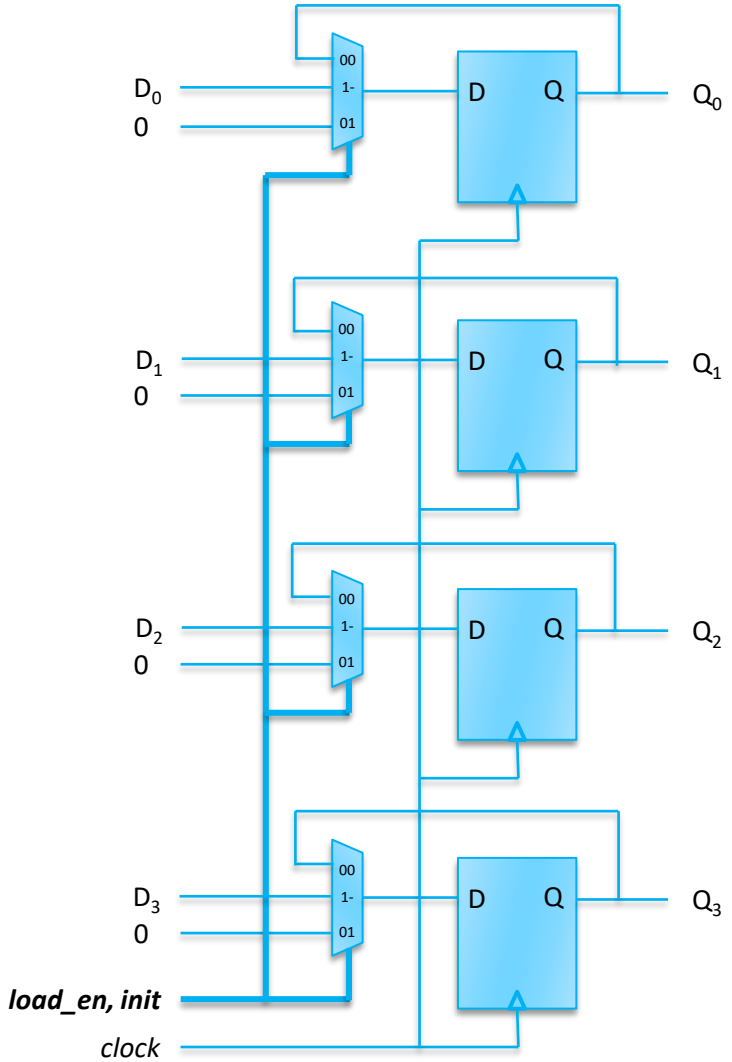
Struttura iniziale del data path



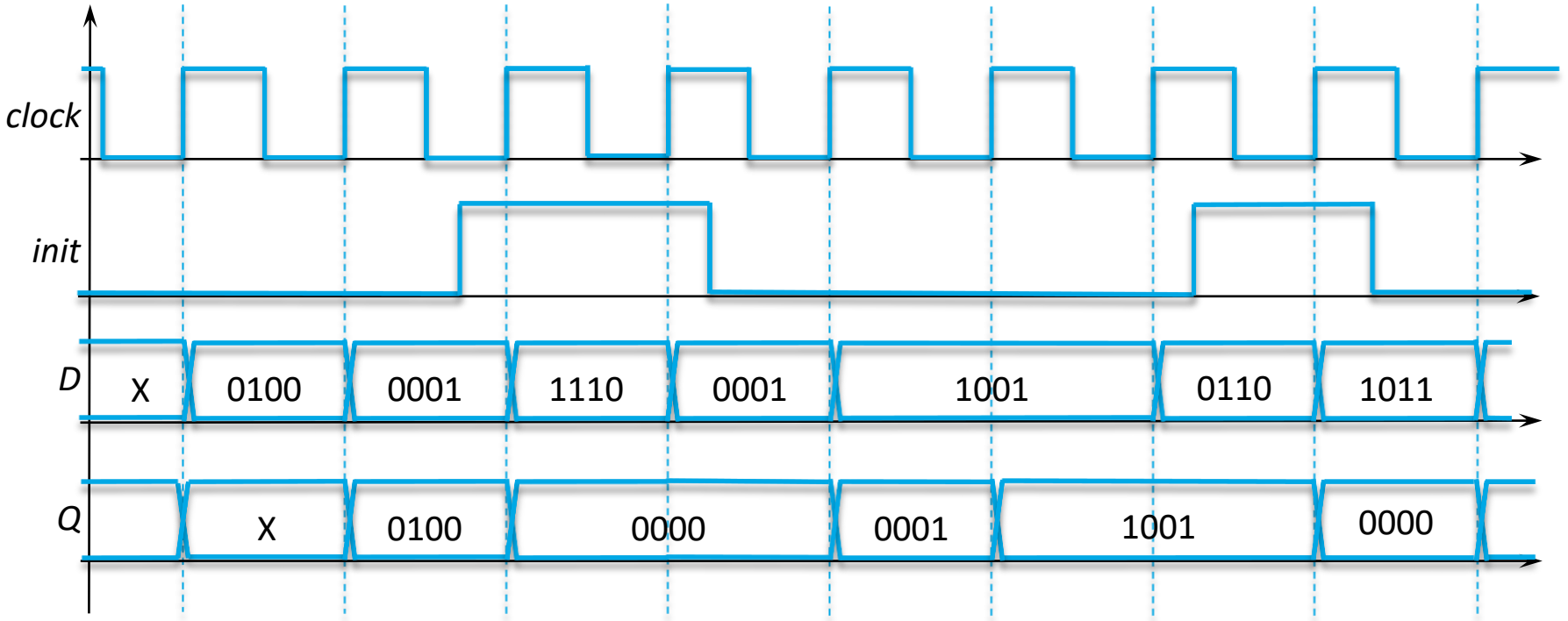
- **TM: contatore BCD a 4 cifre**
  - enTM: abilita il conteggio
  - rsTM: inizializza a 0000
    - Segnale sincrono
- **SD: registro BCD a 4 cifre**
  - ldSD: carica il valore in ingresso
  - rsSD: inizializza a 0000

Registro con init sincrono

- **Si sceglie il valore da caricare**
  - Se *init* è attivo, si carica il valore 0
  - Se *init* non è attivo, si carica il valore in ingresso
- **Come per il *load enable***
  - Volendo, usando un multiplexer a tre ingressi, si può aggiungere anche il *load enable*

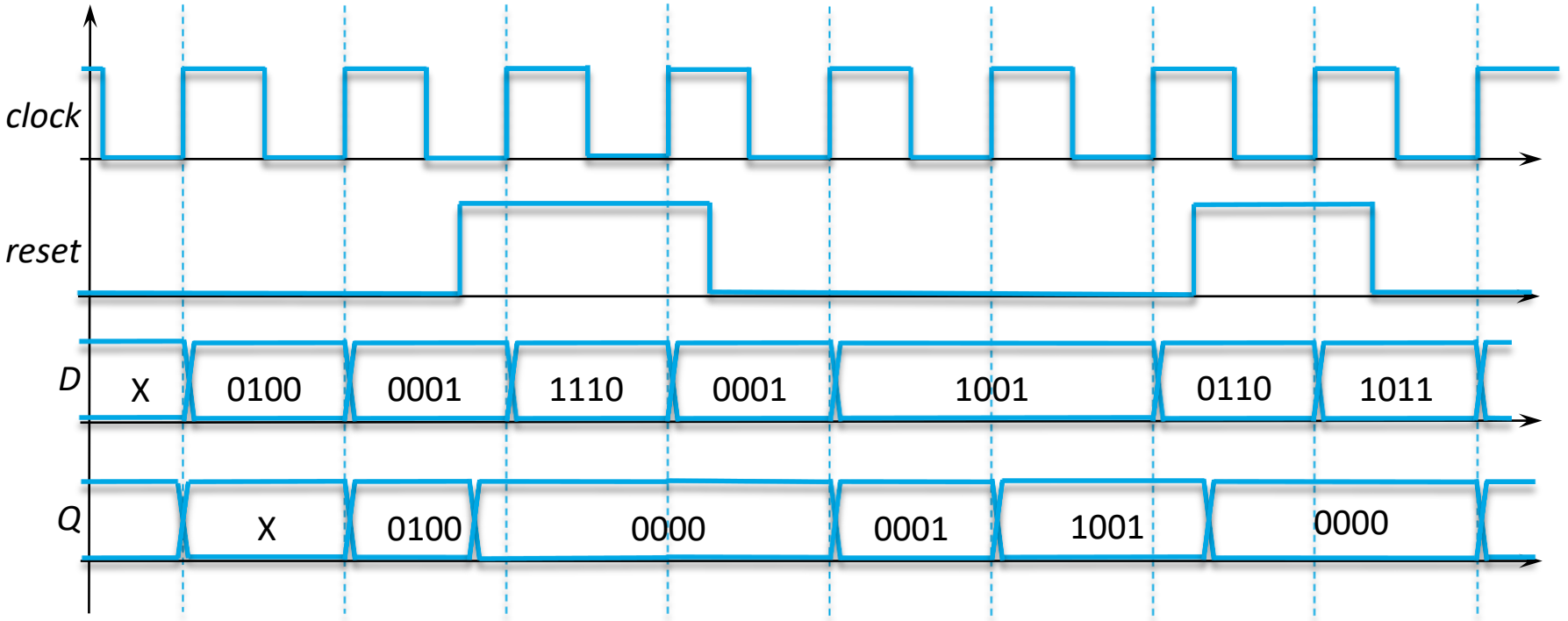


Registro con init sincrono



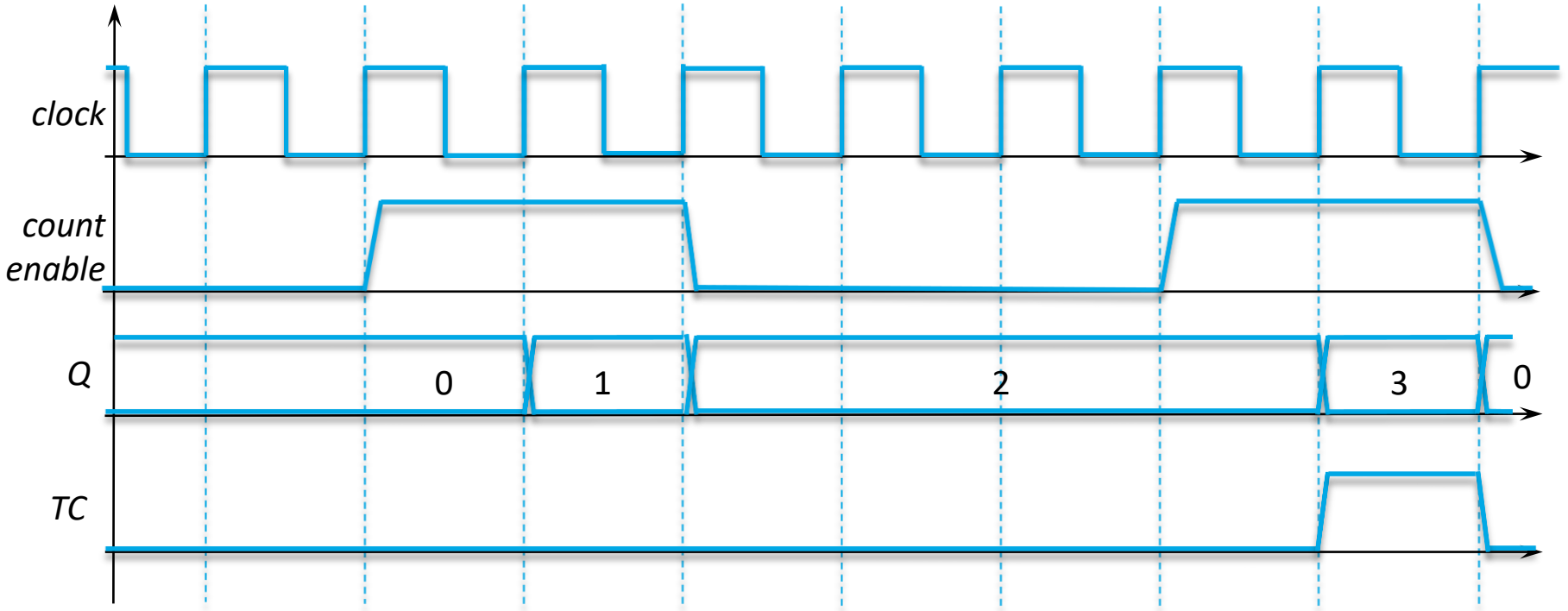
- **Init attivo alto *sincrono***
  - Impone il valore 0 al primo fronte attivo del clock
  - Facile da realizzare con un multiplexer con uno degli ingressi a 0

Registro con reset asincrono



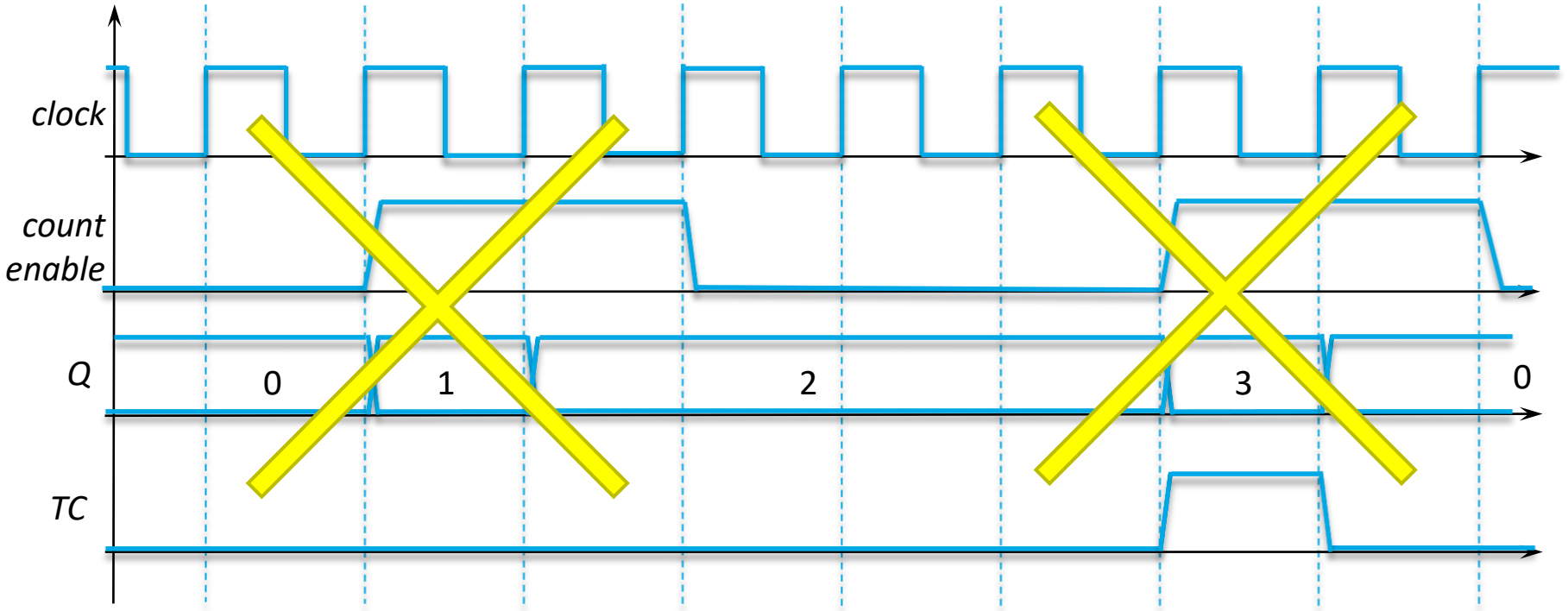
- **Reset attivo alto *asincrono***
  - Impone il valore 0 immediatamente, senza attendere il fronte del clock
  - Si realizza con gli ingressi set e reset asincroni dei flip flop
  - Si può avere un registro con sia l'init sincrono sia il reset asincrono

Contatore con count enable (modulo 4)



- **Count enable attivo alto *sincrono***
  - Normalmente l'uscita del contatore è uguale allo stato, quindi è una macchina di Moore
  - Terminal Count (TC) attivo nel ciclo in cui il contatore è al fondo scala

Contatore con count enable (modulo 4)

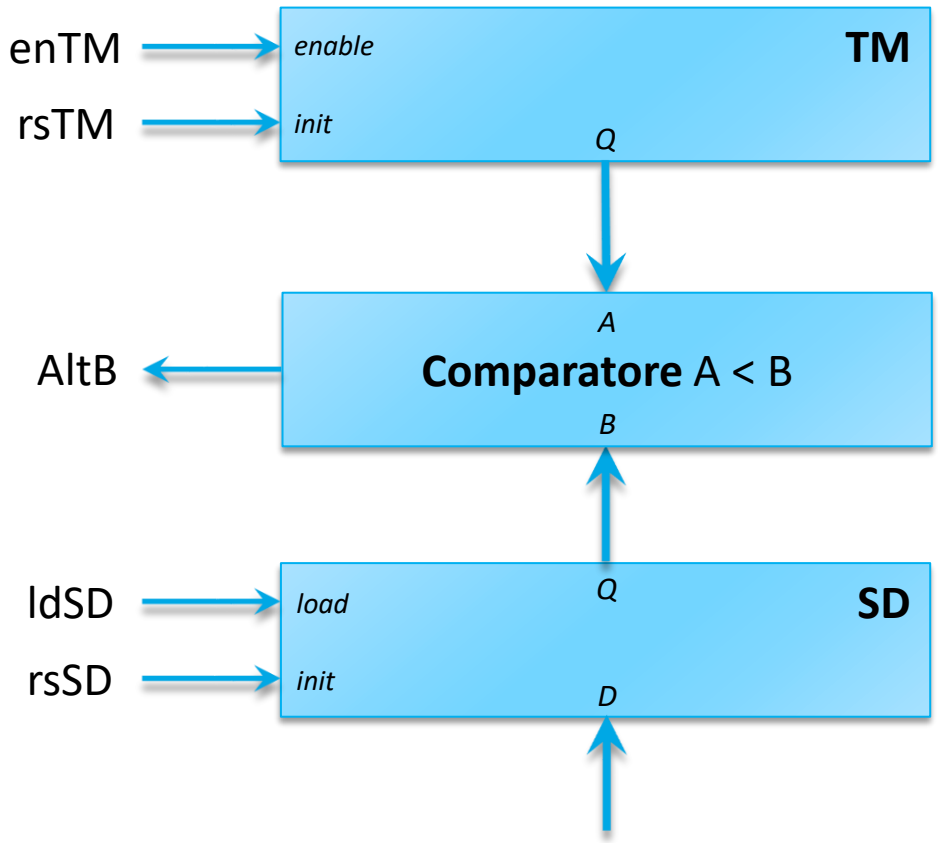


- **Count enable attivo alto *sincrono***
  - Normalmente l'uscita del contatore è uguale allo stato, quindi è una macchina di Moore
  - Terminal Count (TC) attivo nel ciclo in cui il contatore è al fondo scala

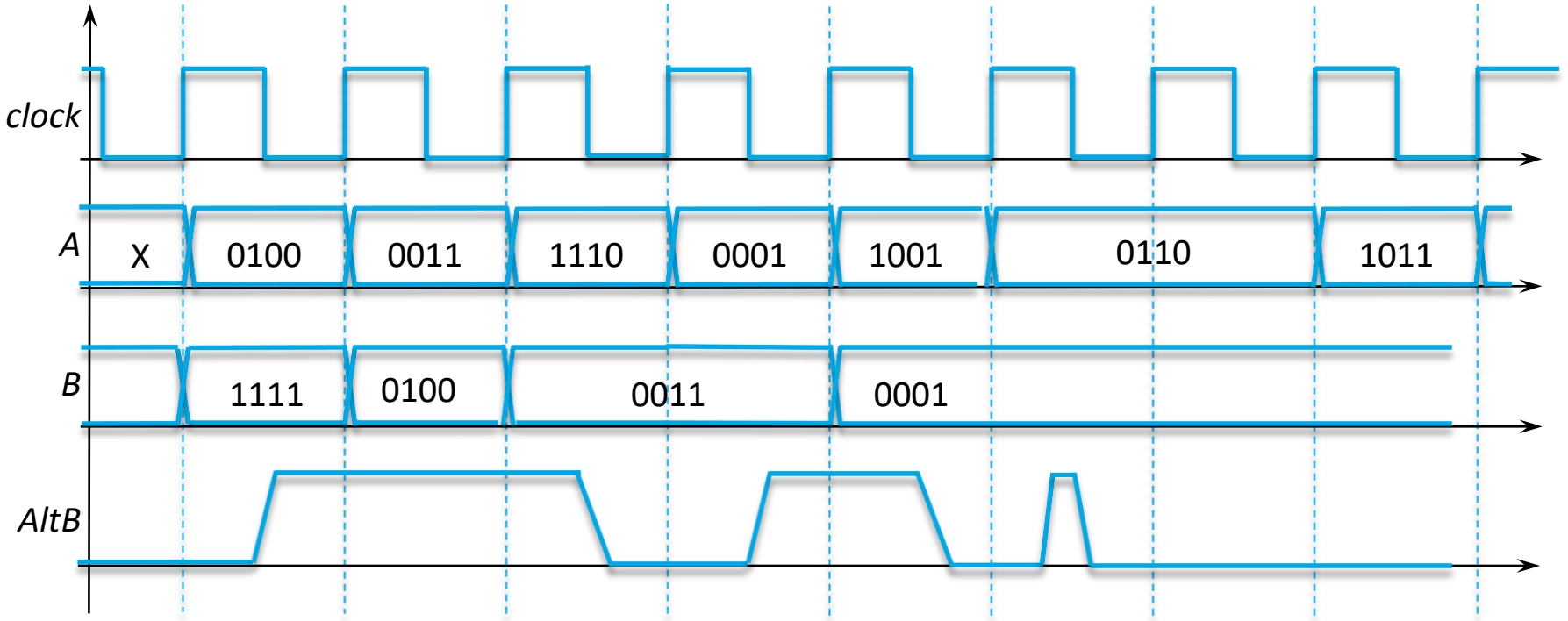
Comparatore

- **Occorre poter confrontare il contenuto del timer con quello del tempo migliore**
  - Usiamo un comparatore BCD parallelo combinatorio
  - Potete progettarlo per esercizio, in maniera iterativa (come il comparatore binario visto ad esercitazioni)
  - Riceve in ingresso due numeri BCD a 4 cifre (16 segnali ciascuno), chiamati A e B
  - Fornisce in uscita una valore binario che è pari a 1 se A < B, e 0 altrimenti

Struttura del data path



Comparatore combinatorio

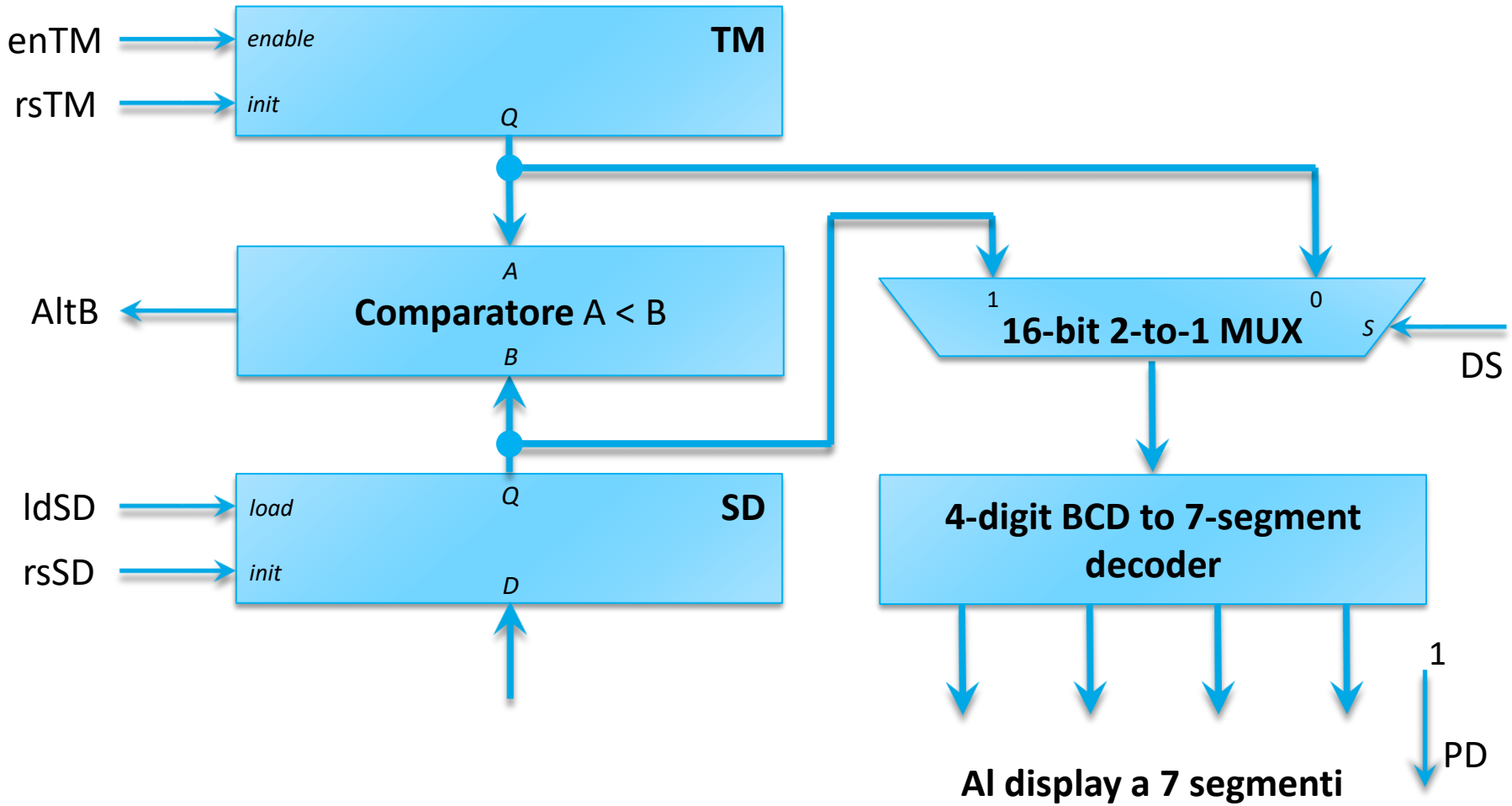


- **Il comparatore è un circuito combinatorio**
  - L'uscita cambia nello stesso ciclo di clock in cui si presentano i valori di A e B (e.g., se A < B, l'uscita va a 1 in quel ciclo)
  - Il ritardo con cui varia deve essere inferiore alla durata del ciclo
  - Possono esservi dei glitch, che devono essere ignorati dalla MSF di controllo

Display

- **Dobbiamo poter mostrare o il conteggio attuale TM, oppure il conteggio del tempo migliore SD**
  - Bisogna usare un multiplexer prima dei display
  - Conviene metterlo prima anche della decodifica 7 segmenti, così il multiplexer è più piccolo (16 bit per ingresso, invece di 28) ed usiamo una decodifica sola
  - Gli ingressi del multiplexer saranno le uscite dei registri
  - Un segnale DS (Display Select) sceglie se mostrare **TM** oppure **SD**

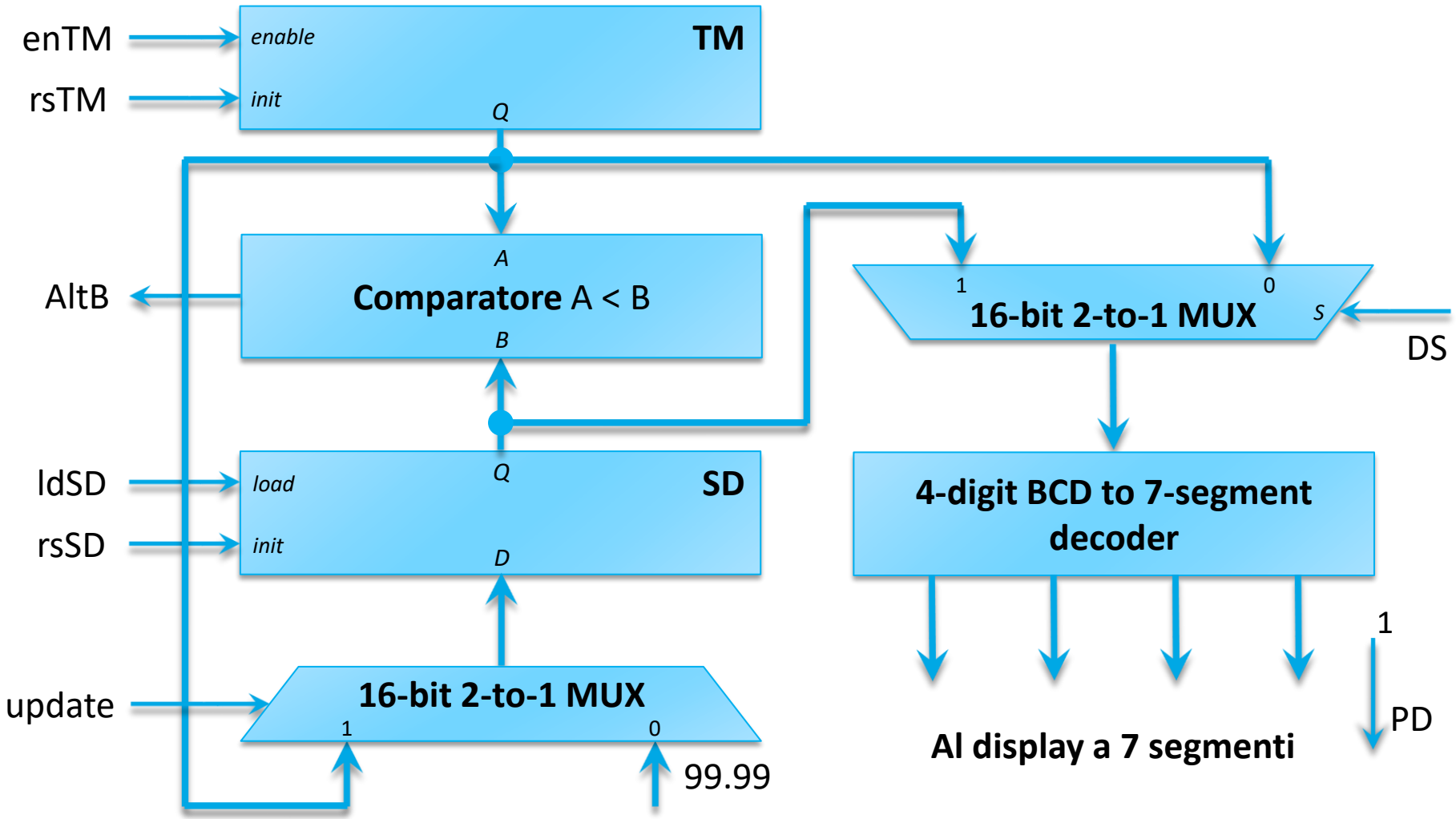
Struttura del data path



Inizializzazione di SD

- **A che valore dobbiamo inizializzare SD?**
  - Il suo init lo mette a 00.00
  - Ma facendo così non si potrà registrare alcun tempo migliore
  - Qualunque tempo non sarà minore di 00.00, quindi il registro **SD** non può essere inizializzato a 00.00
  - Occorre invece inizializzarlo al valore più alto, 99.99
- **Supponiamo il registro con init a 99.99 non esista**
  - Allora sfruttiamo la possibilità di caricamento parallelo
  - In **SD** dobbiamo caricare o 99.99 per l'inizializzazione…
  - … oppure il valore contenuto in **TM**, se migliore
  - Usiamo allora un altro multiplexer

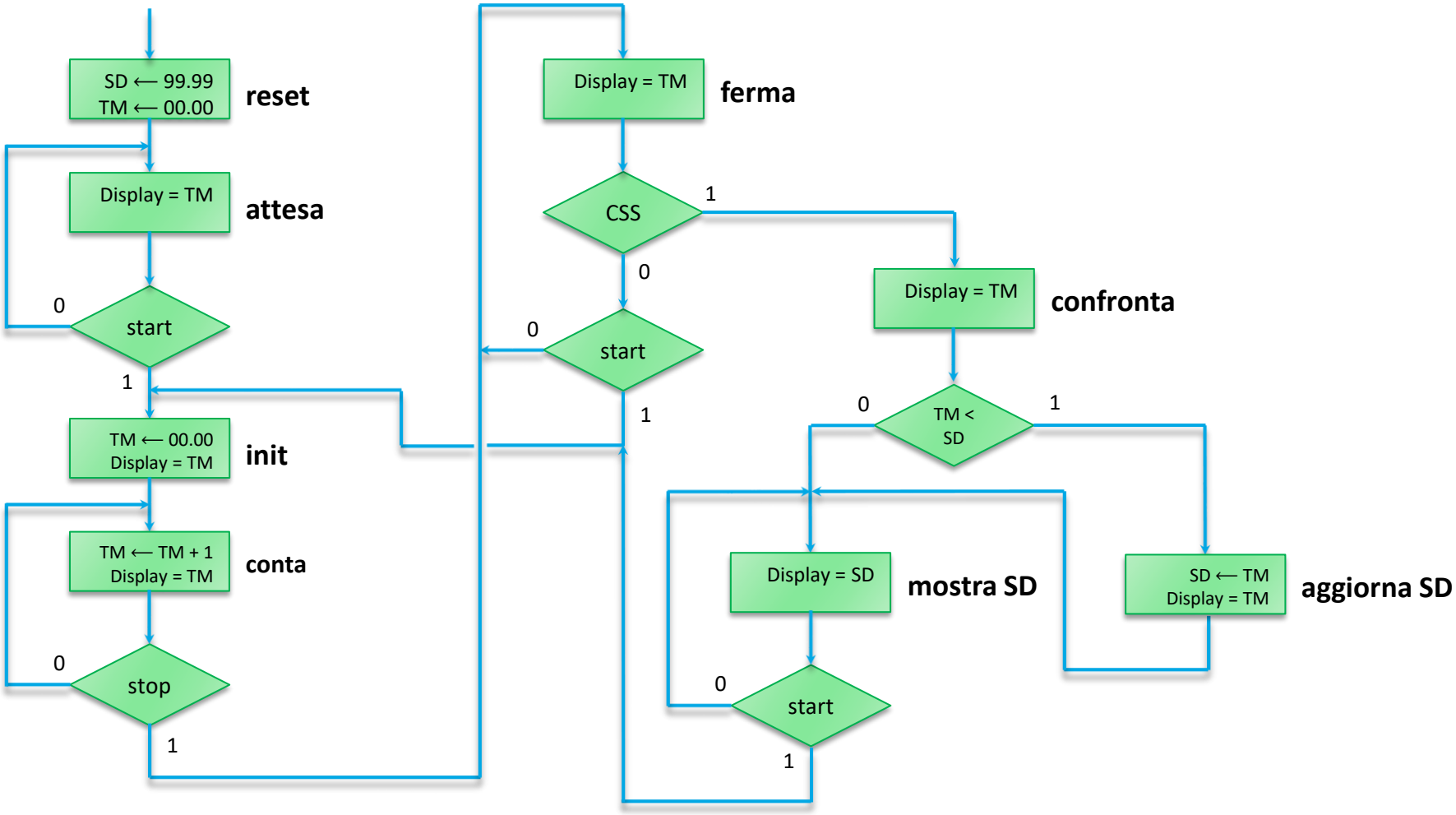
Struttura del data path



Unità di controllo

- **L'unità di controllo si occupa di gestire il data path in base agli ingressi di controllo forniti dall'utente**
  - Si realizza tramite una macchina a stati
  - In ogni stato si comandano certe operazioni del data path in modo da ottenere il risultato voluto
  - Inizialmente usiamo **operazioni simboliche**
  - Si svincola la macchina a stati dai particolari del controllo del data path
- **Register Transfer: $R_1 \leftarrow R_2$, $R_1 \leftarrow R_2 + 1$, $R_1 \leftarrow R_2 - R_3$**
  - Trasferimento di dati tra registri, con eventualmente delle operazione
- **Connessione: $OUT = R_1$**
  - Stabilisce un instradamento di valori verso le uscite o segnali interni

Macchina a stati

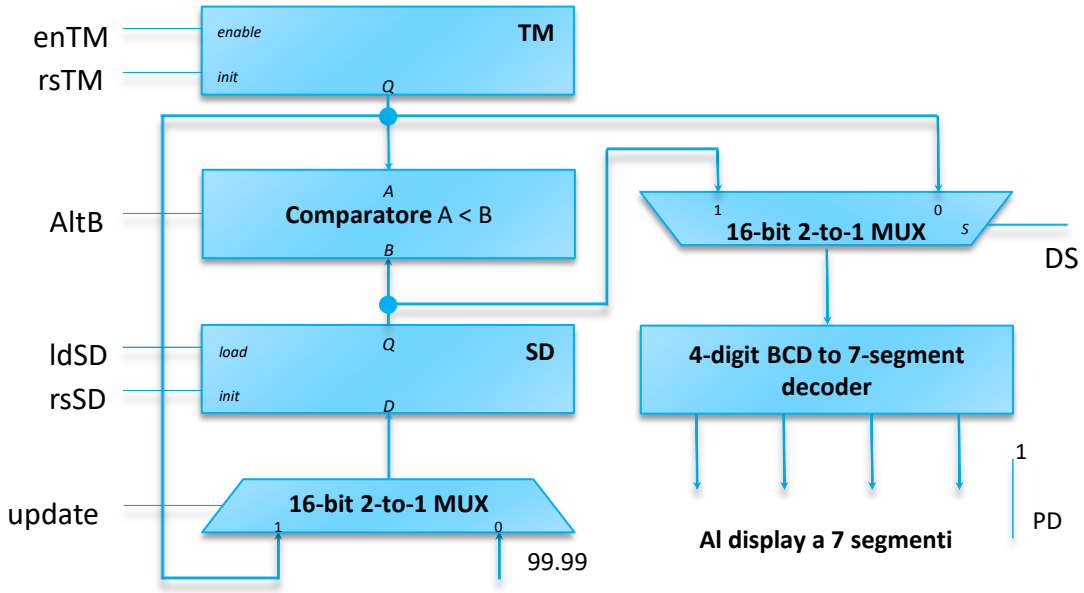


Micro-operazioni e segnali di controllo

- **Le operazioni indicate negli stati si chiamano micro-operazioni**
  - Si riferiscono ad operazioni che possono essere eseguite nel data path
  - Consistono in conti e trasferimenti di registri
- **Vanno realizzate impostando in modo opportuno i segnali di controllo**
  - Ogni micro-operazione avrà quindi una sua codifica
- **Le condizioni sulle transizioni dipendono invece da**
  - Ingressi di controllo dell'unità di controllo (START, STOP, CSS, RESET)
  - Segnali di stato dal data path (AltB)
  - Anche le condizioni avranno quindi la loro codifica

Codifica micro-operazioni

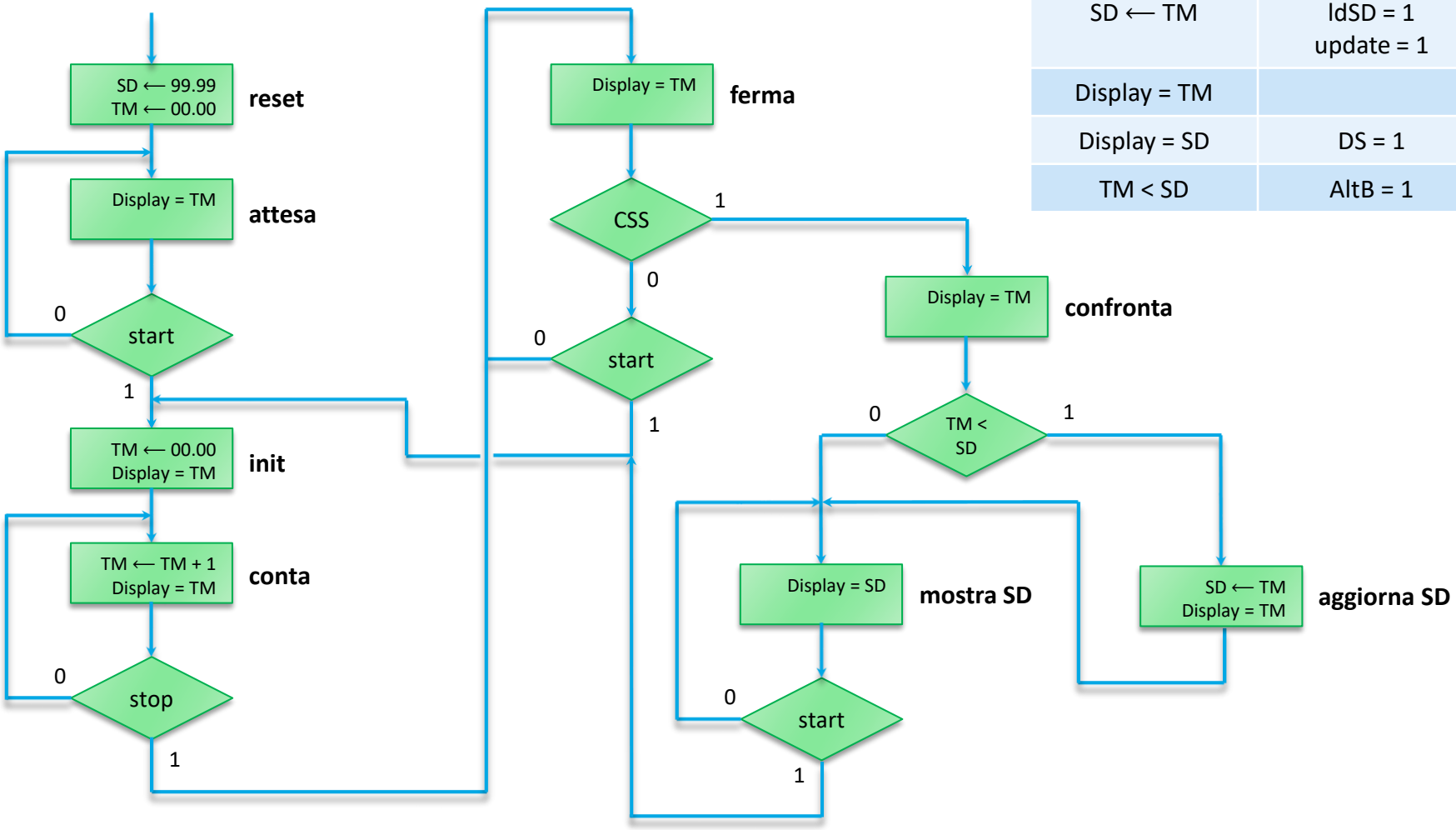
- **Di default, assumiamo che i segnali siano a 0**
  - Indichiamo solo quelli che devono andare a 1



Micro-op	Segnale
TM ⟵ 00.00	rsTM = 1
TM ⟵ TM + 1	enTM = 1
SD ⟵ 99.99	ldSD = 1
SD ⟵ TM	ldSD = 1 update = 1
Display = TM	
Display = SD	DS = 1
TM < SD	AltB = 1

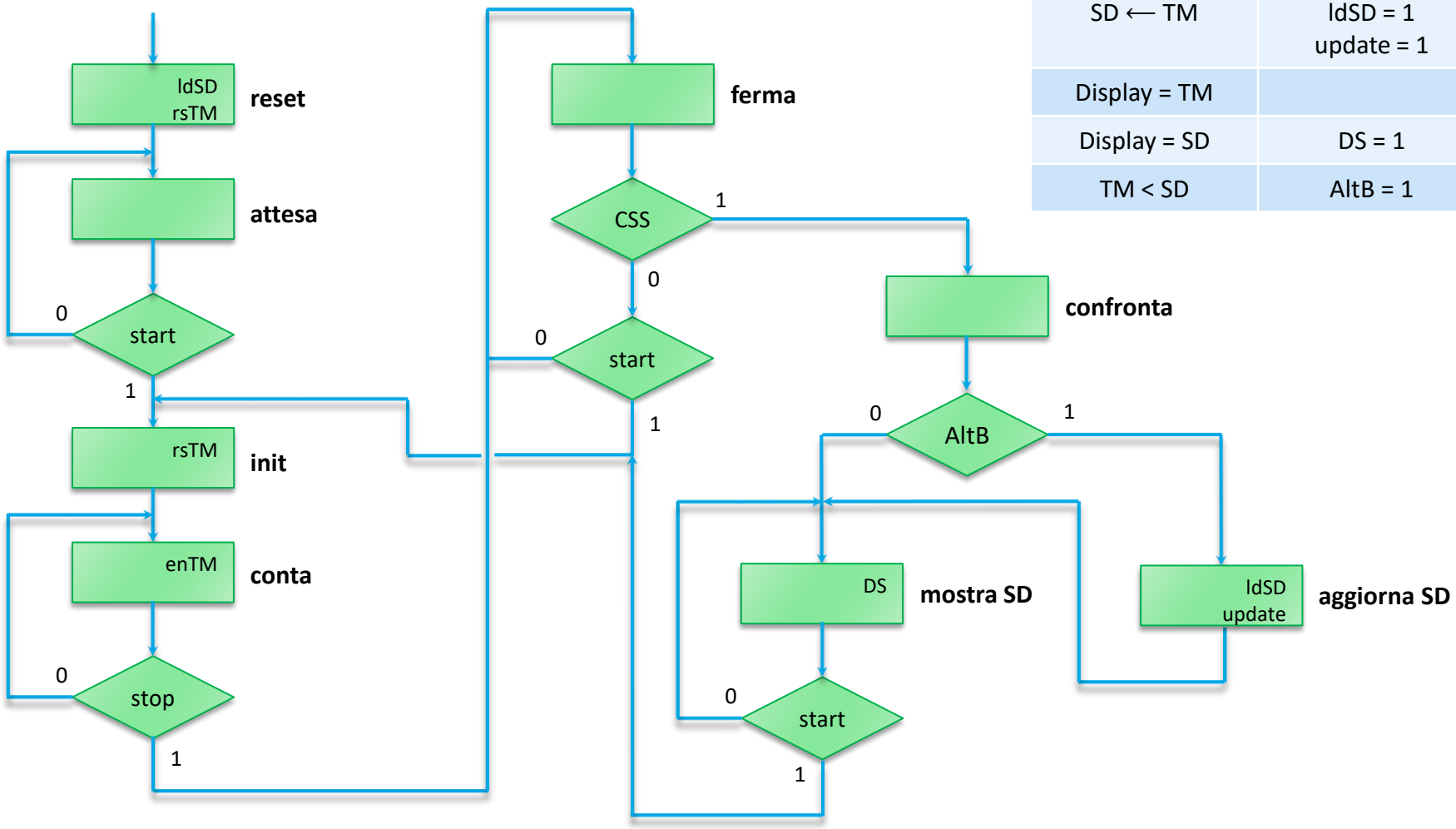
Macchina a stati

Micro-op	Segnale
TM ⟵ 00.00	rsTM = 1
TM ⟵ TM + 1	enTM = 1
SD ⟵ 99.99	ldSD = 1
SD ⟵ TM	ldSD = 1 update = 1
Display = TM	
Display = SD	DS = 1
TM < SD	AltB = 1



Macchina a stati

Micro-op	Segnale
TM ⟵ 00.00	rsTM = 1
TM ⟵ TM + 1	enTM = 1
SD ⟵ 99.99	ldSD = 1
SD ⟵ TM	ldSD = 1 update = 1
Display = TM	
Display = SD	DS = 1
TM < SD	AltB = 1

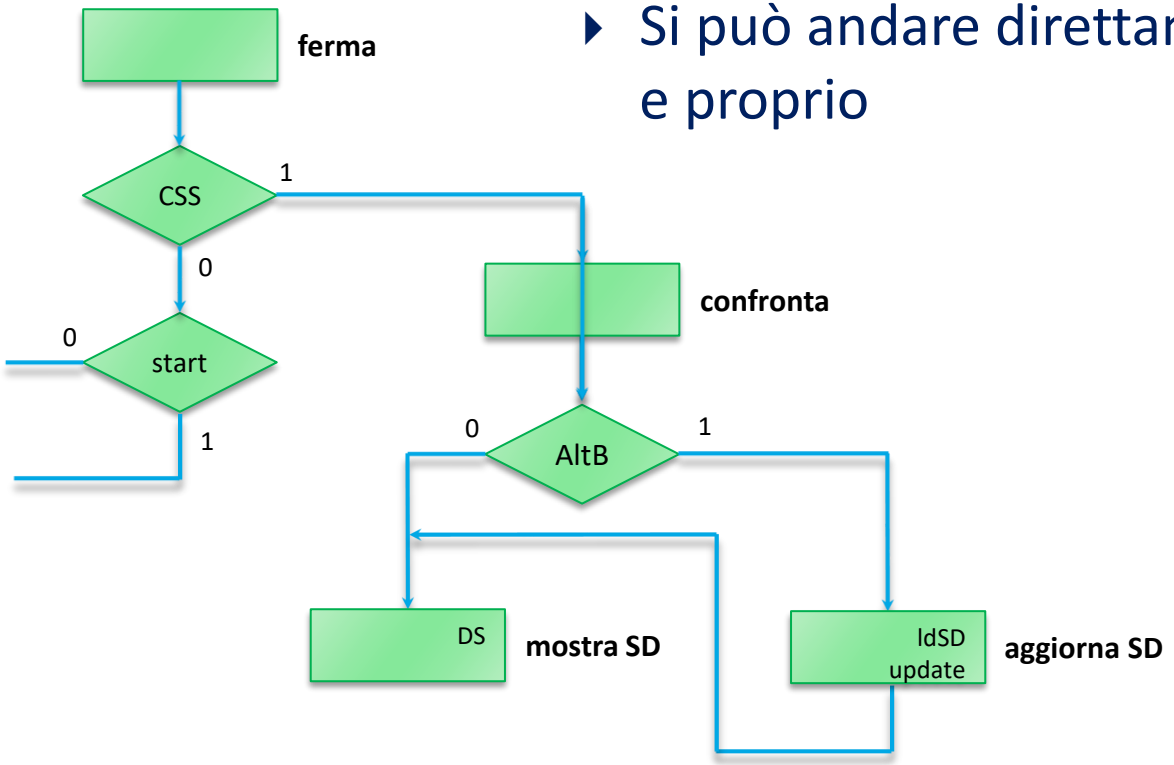


Osservazioni

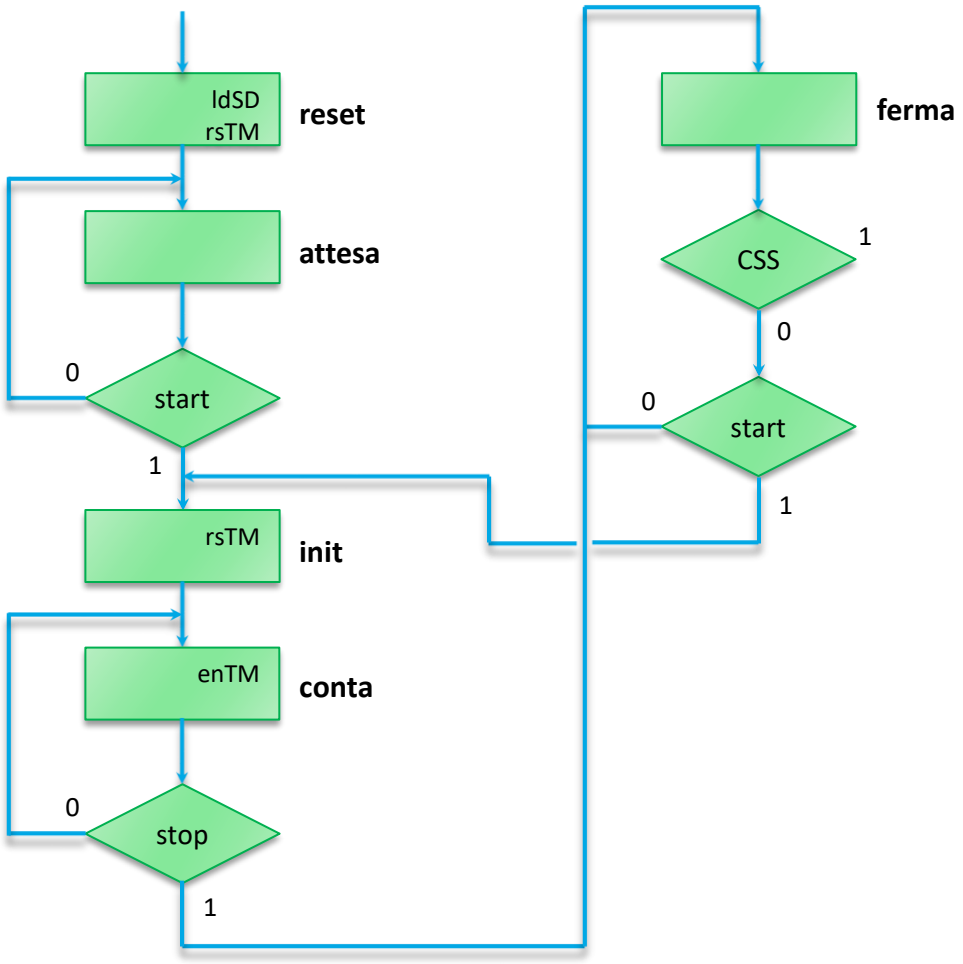
- **Attenzione a quando l'operazione viene effettivamente eseguita**
  - Se in uno stato comando il caricamento di un dato in un registro, questo sarà nel registro solo nel ciclo di clock **successivo** a quello in cui il comando è attivo
  - Quindi anche nello stato successivo
  - Le operazioni combinatorie invece si completano durante lo stesso ciclo di clock
  - Ma memorizzo il risultato solo al ciclo dopo

Ottimizzazioni

- **La macchina a stati che abbiamo realizzato non è ottimizzata**
  - Per esempio lo stato **confronta** non serve
  - Si può andare direttamente al confronto vero e proprio

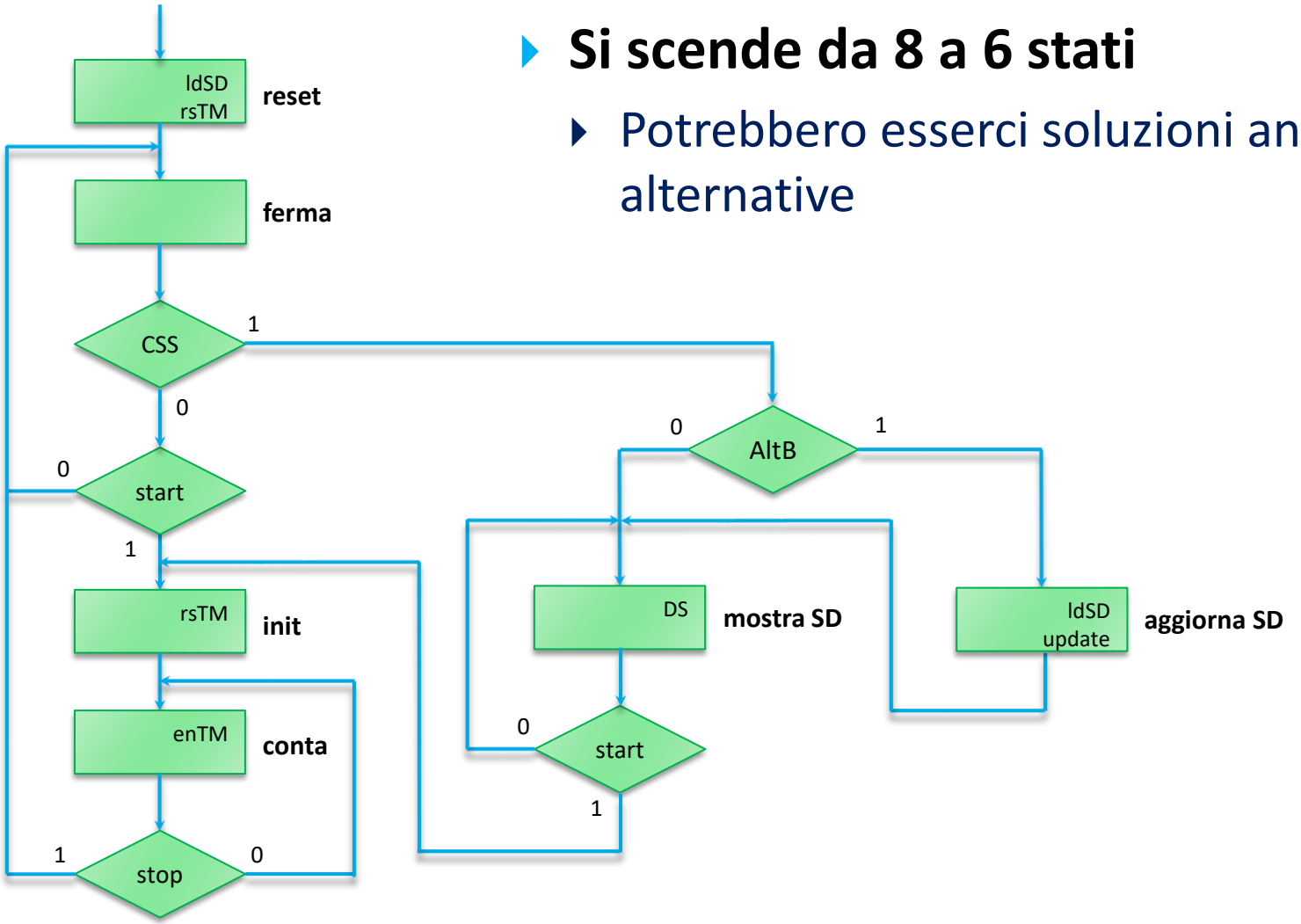


Ottimizzazioni



- **Lo stato attesa può anche essere rimosso**
  - Lo stato **ferma** fa quasi la stessa cosa (stessi output)
  - Per *start* = 0 torna in se stesso
  - Per *start* = 1 va in **init**
  - E' anche sensibile al CSS, mentre **attesa** non lo è
  - Lo stato ferma può sostituire **attesa**
  - Attenzione che ora CSS premuto all'inizio fa andare SD a 0, che sarebbe contro le specifiche

Macchina a stati



- **Si scende da 8 a 6 stati**
  - Potrebbero esserci soluzioni ancora alternative

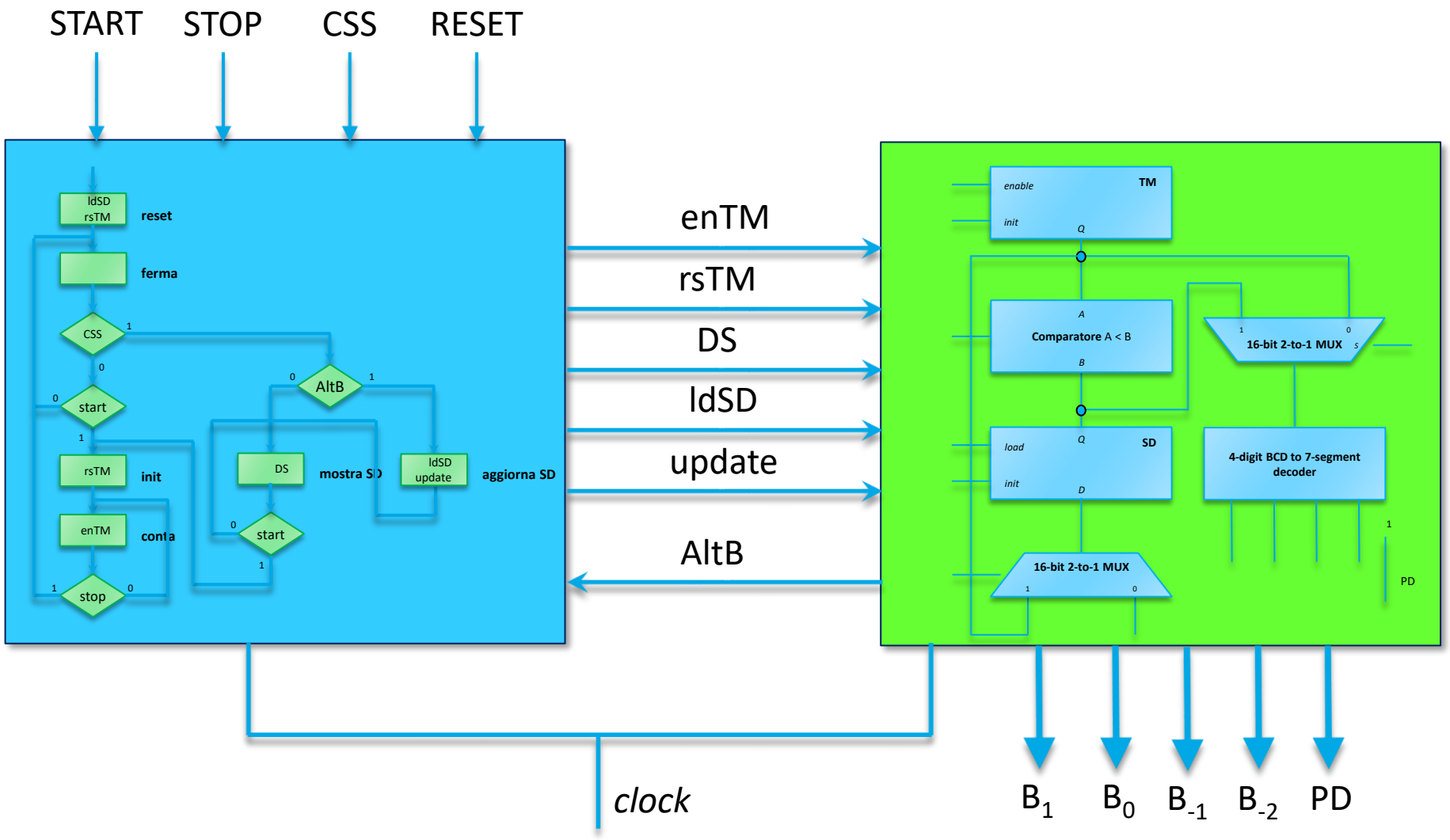
Ritardo di inizializzazione

- **Vi è un ritardo tra il momento in cui si preme START e quello in cui si comincia il conteggio**
  - C'è un ritardo perché comunque START lo si rileva solo al fronte del clock
  - E in ogni caso passiamo prima per uno stato che inizializza il timer a 0
  - Con un clock a 100 Hz può voler dire un paio di centesimi di secondo
- **Come risolvere la situazione?**
  - Usare uscite condizionate
    - Ma il ritardo rimane, visto che gli ingressi vanno comunque sincronizzati tramite un flip flop per evitare corse critiche
  - Comunque ci sarà un clock molto più veloce disponibile, facciamo andare il sistema a quella velocità
    - Richiede alcuni cambiamenti al circuito e/o alla macchina a stati
  - Inizializziamo il timer a 00.01 invece che a 00.00!!

Realizzazione finale

- **Per completare il progetto occorre finalizzare la macchina a stati**
  - Codifica degli stati
  - Rete combinatoria di calcolo dello stato futuro
  - Rete combinatoria di calcolo delle uscite
- **Occorre inoltre avere l'implementazione di tutti i componenti del data path**
  - Quelli che mancano vanno progettati secondo le specifiche
- **Quindi si mette tutto assieme**
  - Già solo questo cronometro è molto più complesso di quanto fatto finora
  - Si può cercare di verificarne il funzionamento sviluppando dei diagrammi temporali
  - Strumenti di simulazione sono usati per crearne in maniera automatica

Struttura di sistema



Take away

- **La metodologia RTL si compone dei seguenti passi**
  1. Scrittura della specifica di sistema
  2. Definizione degli ingressi e uscite primarie di dati e controllo del sistema
  3. Definizione dei registri di dati
  4. Sviluppo di una macchina a stati di controllo facente uso di operazioni simboliche di trasferimento tra registri e di condizioni
  5. Sviluppo del diagramma schematico del data path che realizzi le operazioni simboliche e le condizioni
  6. Codifica delle operazioni e delle condizioni
  7. Progetto dettagliato del data path e dell'unità di controllo
  8. Verifica del progetto
- **L'ordine non è necessariamente prefissato**
  - Il progetto del data path e del controllo potrebbero andare di pari passo
  - Il progetto può essere reso gerarchico con l'interconnessione di più sotto-sistemi differenti

Take away

- **La metodologia RTL è ancora oggi tra le più usate**
  - Riduce la complessità separando la parte dedicata all'elaborazione e la memorizzazione dei dati da quella dedicata al suo controllo
  - Normalmente coadiuvata dall'uso di linguaggi di descrizione dell'hardware, come ad esempio il Verilog o il VHDL
  - Fa uso di blocchi preconfezionati per semplificare il progetto
- **La creazione del data path e del controllo sono, appunto, delle creazioni**
  - Per oggetti semplici è facile individuare una struttura efficiente
  - Per oggetti più complessi le alternative architetturali sono molteplici
  - Non vi è una ricetta come le mappe di Karnaugh
  - L'esperienza e gli strumenti automatici (simulazione, sintesi, analisi) guidano le scelte di progetto

Progetto del data path

Paradigmi di trasferimento, operazioni

Componenti principali del data path

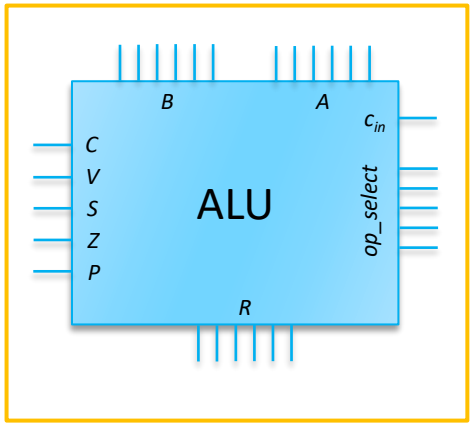
- **Operazioni logico/aritmetiche**
  - Le abbiamo già viste
  - Spesso si usano blocchi in grado di svolgere diverse funzioni secondo segnali di configurazione
  - E' utile cercare di condividere lo stesso hardware per operazioni diverse
- **Memorizzazione**
  - Registri e contatori
  - Talvolta i registri stessi incorporano delle funzioni logico/aritmetiche al loro ingresso
- **Trasferimento**
  - Insieme di collegamenti necessari per instradare i dati tra i registri, e tra registri e operatori
  - Multiplexer spesso usati per effettuare le condivisioni e le scelte

Arithmetic Logic Unit (ALU)

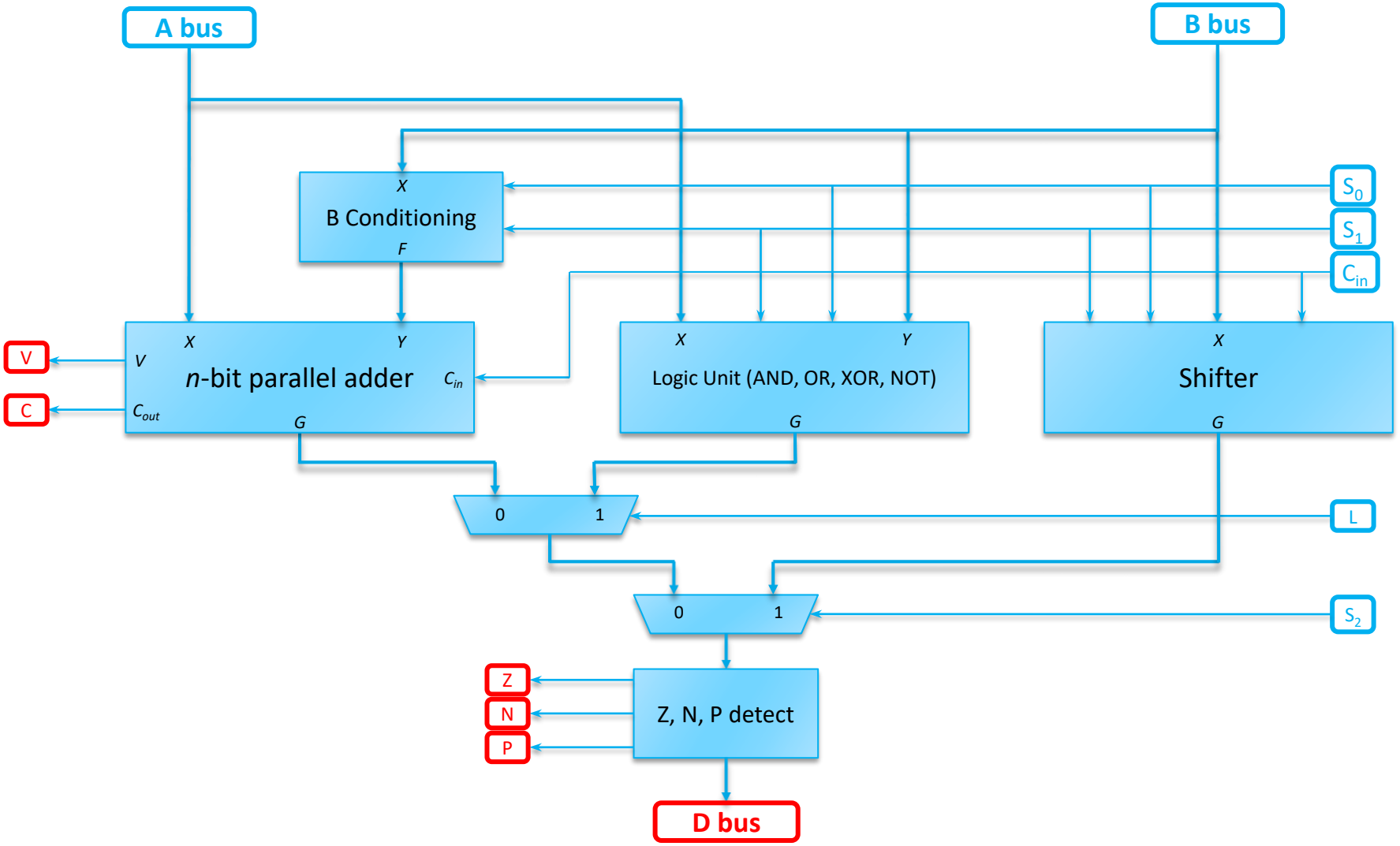
- **L'unità aritmetico/logica è un componente spesso trovato nel data path**
  - E' un oggetto generico che può essere configurato tramite dei segnali di controllo per eseguire una di diverse operazioni logiche o aritmetiche
  - Fornisce in uscita dei segnali di stato, quali il carry e l'overflow, con i quali gestire il flusso successivo di operazioni
  - La sua complessità dipende dall'applicazione che si vuole realizzare
- **Diverse classi di operazioni**
  - Operazioni aritmetiche (somma, sottrazione, moltiplicazione, divisione, confronto)
  - Operazioni logiche (NOT, AND, OR, XOR, etc.)
  - Operazioni di shift (rotazioni, traslazioni, etc.)

ALU

- **La ALU combina varie funzionalità**
  - Per esempio le manipolazioni di bit delle operazioni logiche settano i valori dei flag
    - Si può guardare se un certo bit di un registro è a 1 oppure a 0
  - Può diventare un oggetto piuttosto complesso
  - Data la sua genericità è utile soprattutto nel progetto di processori
  - Anche però nei circuiti specifici l'uso di una ALU può essere vantaggioso, perché si condivide l'hardware per varie operazioni differenti
  - Si diminuisce tuttavia il parallelismo del sistema
    - Se vogliamo fare due operazioni con una ALU dobbiamo necessariamente eseguirle in serie



ALU / Shifter

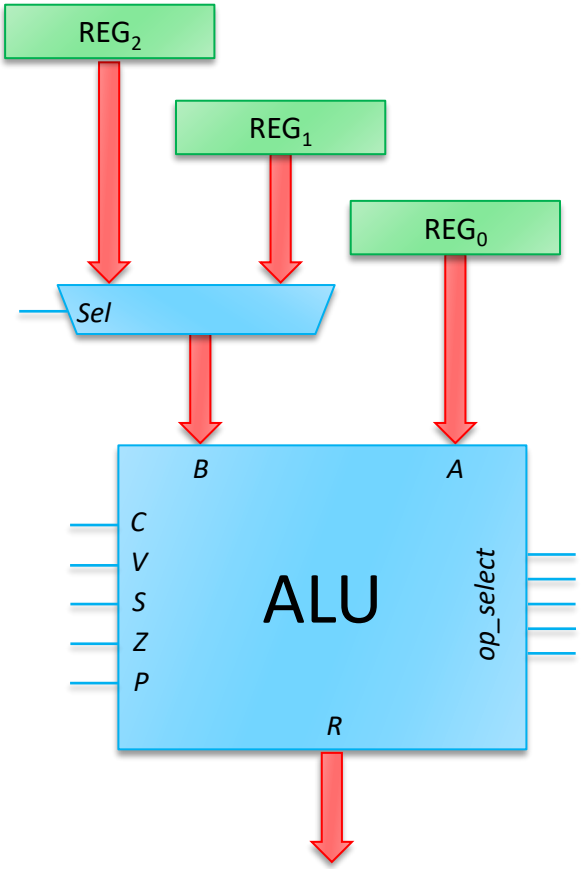

A bus
B bus
X
B Conditioning
F
S0
S1
Cin
V
C
X
Y
V
n-bit parallel adder
Cin
Cout
G
X
Y
Logic Unit (AND, OR, XOR, NOT)
G
X
Shifter
G
0
1
L
0
1
S2
Z
N
P
Z, N, P detect
D bus

Trasferimenti di dati

- **Oltre alle operazioni aritmetico/logiche occorre spesso spostare dati da un registro al resto del circuito**
  - Per esempio tra un registro ed un altro registro, come copiare il contenuto di **TM** in **SD** nel nostro cronometro
  - Oppure dai registri verso le unità di calcolo
  - E dalle unità di calcolo di nuovo verso i registri
- **Molte volte un blocco deve ricevere dati da sorgenti diverse**
  - Essenziale per poter condividere una risorsa di calcolo tra diverse operazioni
  - O per poter scegliere tra vari operandi e risultati
  - Occorre poter arbitrare l'accesso
- **Esistono varie tecniche di instradamento**
  - Basate sull'uso di multiplexer
  - Coadiuvate dall'uso di segnali di caricamento specifici ed indipendenti
  - Facenti uso di buffer tri-state

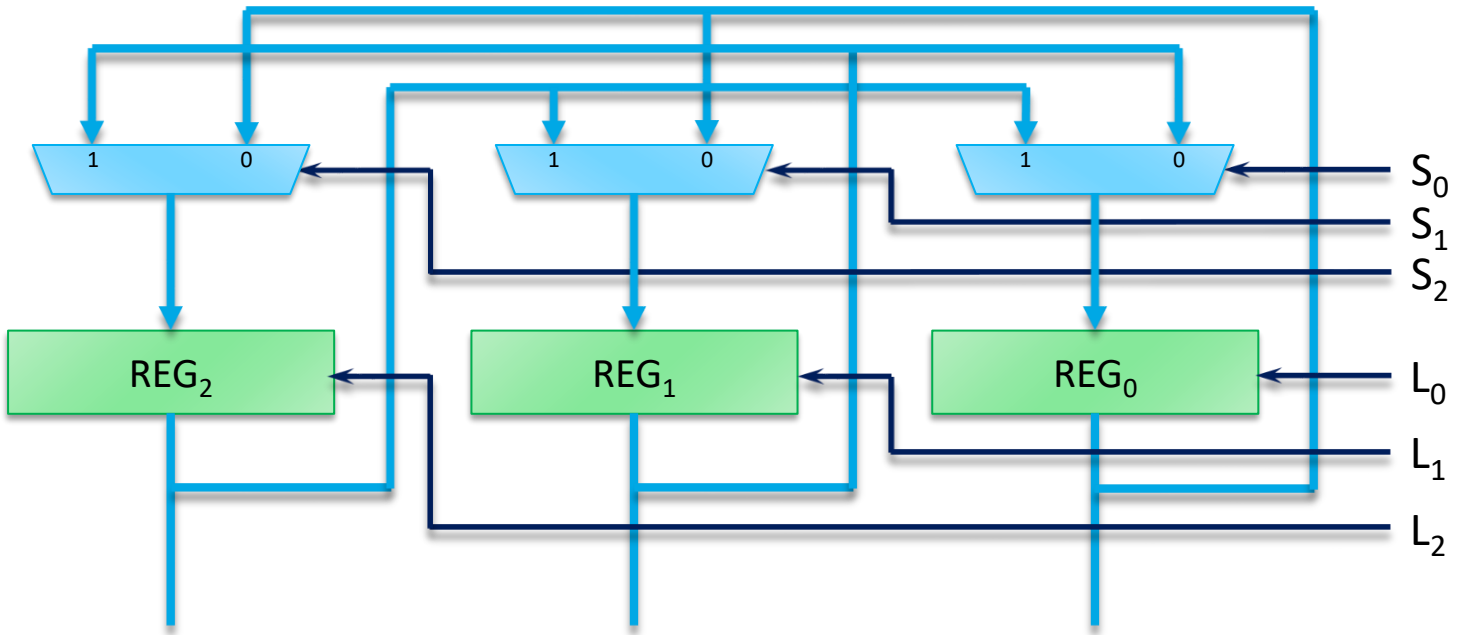
Trasferimenti con multiplexer

- **Supponiamo di voler confrontare il contenuto di un registro REG_0 con uno tra REG_1 e REG_2**
  - Per esempio in un termometro confrontiamo la temperatura attuale con quella minima e massima registrata
- **Si può eseguire la scelta tra REG_1 e REG_2 tramite un multiplexer**
  - Soluzione semplice
  - Scelta tramite i segnali di selezione del multiplexer
  - Segnali di selezione codificati in binario, quindi in numero logaritmico delle possibili scelte



Trasferimenti con multiplexer

- **Tecnica già usata nel progettare i registri a caricamento parallelo**
  - Si può scegliere se caricare un nuovo valore, costanti, shift, valore precedente, etc.
  - Si può generalizzare ad un insieme di registri



Caratteristiche

- **Possiamo eseguire tutti i trasferimenti possibili**
  - E' sufficiente selezionare la corretta combinazione di segnali S ed L
  - Posso caricare un nuovo valore in un registro, mentre trasferisco contemporaneamente il vecchio valore in un altro registro
  - Funziona perché tutto viene fatto al fronte del clock

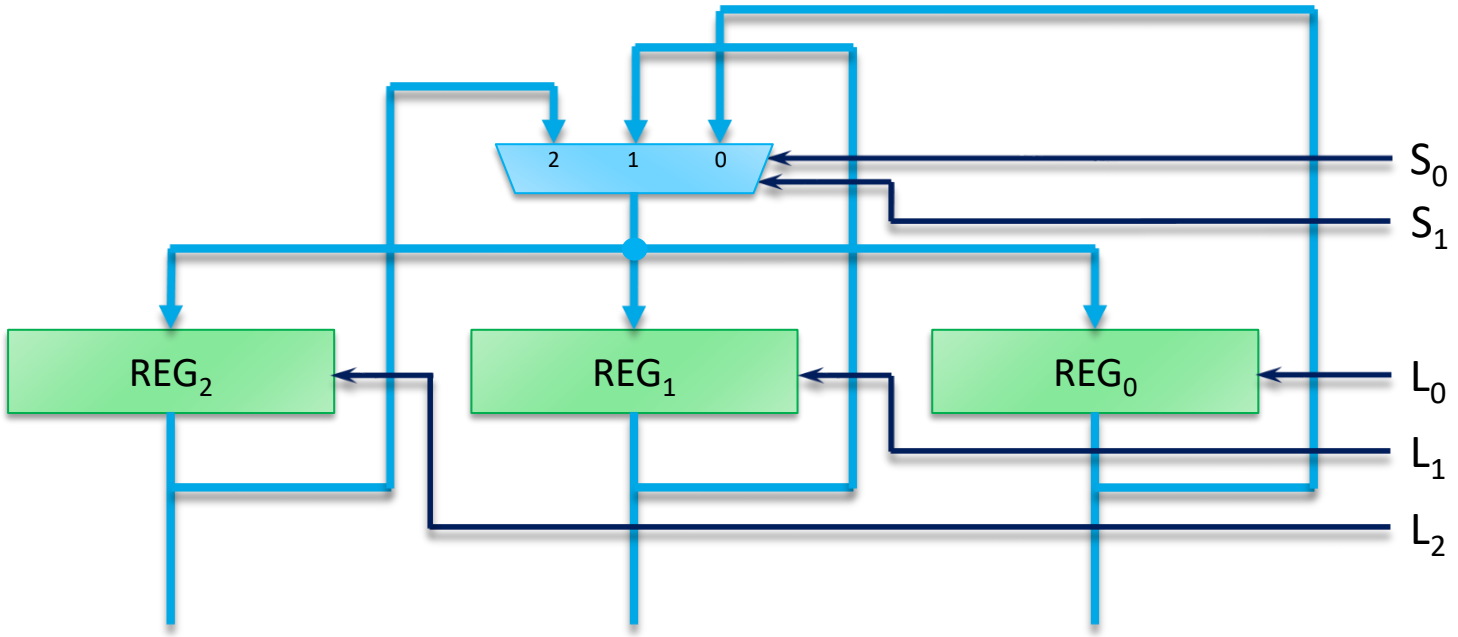
S_0	S_1	S_2	L_0	L_1	L_2	Trasferimento
0	0	-	1	1	0	$R_0 \leftarrow R_1$, $R_1 \leftarrow R_0$, $R_2 \leftarrow R_2$
-	1	0	0	1	1	$R_0 \leftarrow R_0$, $R_1 \leftarrow R_2$, $R_2 \leftarrow R_0$
1	0	1	1	1	1	$R_0 \leftarrow R_2$, $R_1 \leftarrow R_0$, $R_2 \leftarrow R_1$
1	-	-	1	0	0	$R_0 \leftarrow R_2$, $R_1 \leftarrow R_1$, $R_2 \leftarrow R_2$

Complessità

- **Tecnica poco scalabile**
  - Funziona bene per pochi registri
  - Per m registri ad n bit devo riportare indietro $m \bullet n$ fili ed ogni multiplexer avrà $(m - 1) \bullet n$ ingressi
  - L'intero circuito viene letteralmente mangiato dalle interconnessioni
  - Difficile realizzare un layout compatto anche con diversi livelli di metallizzazione
- **Difficoltà di ampliamento**
  - Qualora volessi aggiungere un nuovo registro devo modificare tutto il resto del circuito
  - Ogni multiplexer deve avere un nuovo ingresso
  - Potenzialmente anche gli ingressi di selezione devono essere aumentati

Bus e multiplexer

- **Una possibile semplificazione consiste nel trasformare parte del routing in un bus**
  - Un bus è una serie di interconnessioni condivise da diversi elementi
  - Per esempio possiamo collegare tutti gli ingressi dei registri assieme e all'uscita di un multiplexer



Caratteristiche

- **Ora non tutti i trasferimenti sono possibili**
  - Per ogni registro si può scegliere se caricare oppure no
  - Ma c'è un solo valore disponibile, uguale per tutti i registri e selezionato tramite il multiplexer
  - Certi tipi di trasferimento richiedono di suddividere l'operazione in diversi cicli di clock

S_0	S_1	L_0	L_1	L_2	Trasferimento
0	1	1	0	0	$R_0 \leftarrow R_2$, $R_1 \leftarrow R_1$, $R_2 \leftarrow R_2$
1	0	1	0	1	$R_0 \leftarrow R_1$, $R_1 \leftarrow R_1$, $R_2 \leftarrow R_1$
0	0	0	1	0	$R_0 \leftarrow R_0$, $R_1 \leftarrow R_0$, $R_2 \leftarrow R_2$
Non	pos	si	bi	le	$R_0 \leftarrow R_2$, $R_1 \leftarrow R_0$, $R_2 \leftarrow R_2$

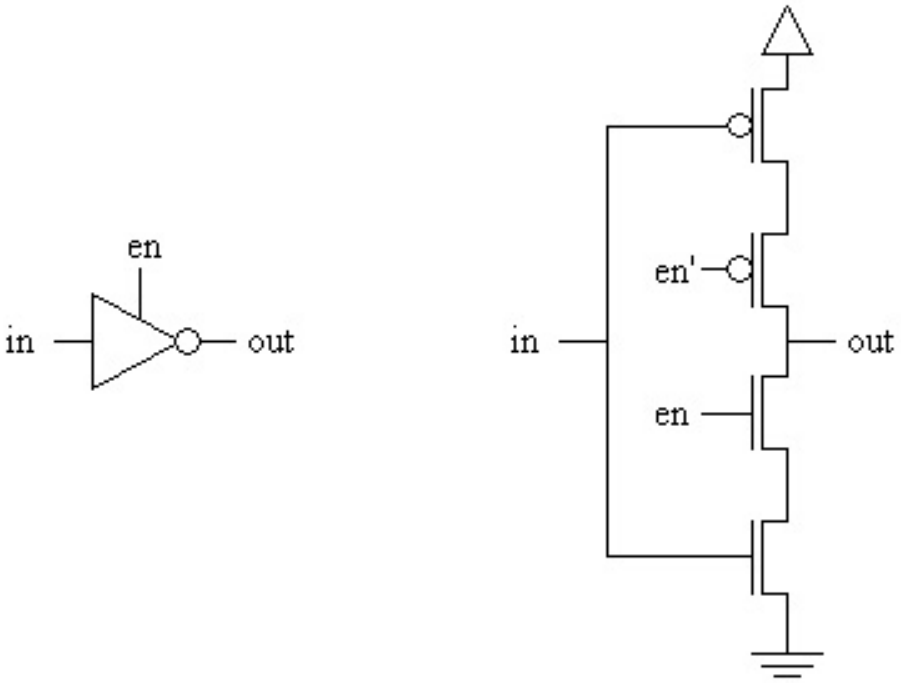
In un primo ciclo eseguo $R_1 \leftarrow R_0$
In un secondo ciclo eseguo $R_0 \leftarrow R_2$
Attenzione all'ordine delle operazioni!!

Complessità

- **Tecnica molto più scalabile**
  - Molte meno porte logiche rispetto alla soluzione precedente (un solo multiplexer, anche se più grosso)
  - Funziona bene per un numero medio di registri
  - Per m registri ad n bit devo comunque riportare indietro $m \bullet n$ fili ed il multiplexer avrà $m \bullet n$ ingressi, però i fili vanno in un posto solo
  - Le interconnessioni potrebbero ancora essere dominanti
- **Più semplice da ampliare del precedente**
  - Qualora volessi aggiungere un nuovo registro devo modificare il solo multiplexer e un po' del routing
  - Potenzialmente anche gli ingressi di selezione devono essere aumentati

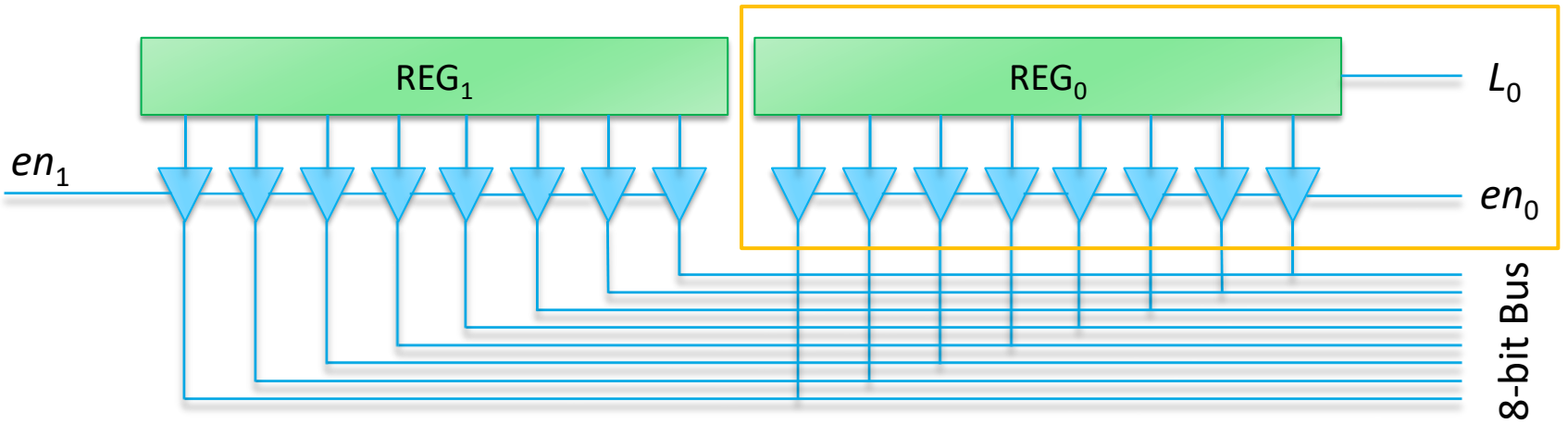
Bus tri-state

- **L'uso di buffer tri-state per realizzare strutture a bus può semplificare notevolmente l'architettura**
  - Un buffer tri-state dispone di un ingresso di enable che consente di attaccare o staccare la sua uscita dal circuito



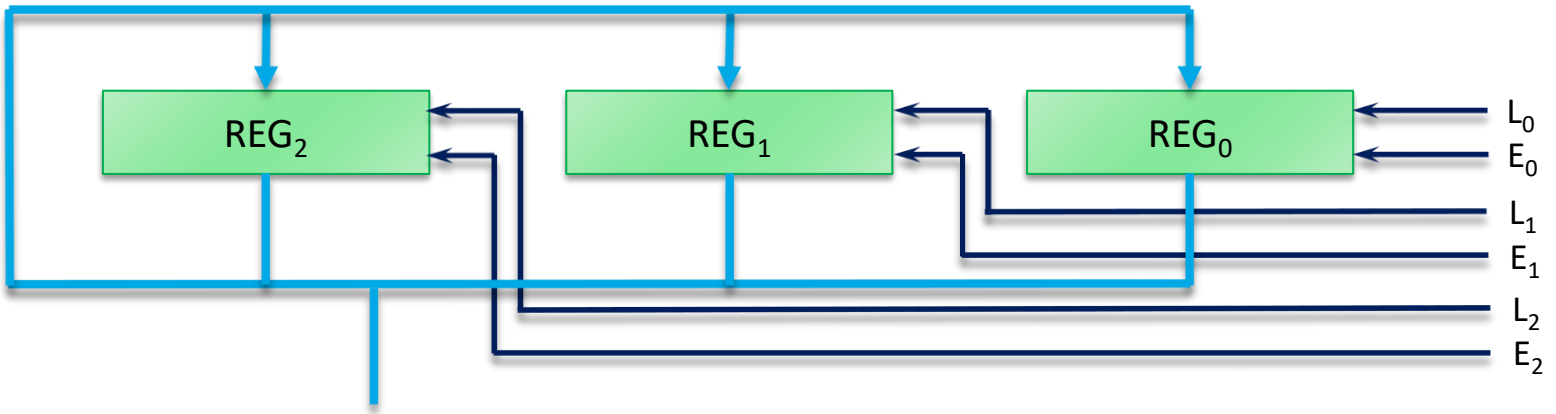
Bus tri-state

- **L'uso di buffer tri-state per realizzare strutture a bus può semplificare notevolmente l'architettura**
  - Un buffer tri-state dispone di un ingresso di enable che consente di attaccare o staccare la sua uscita dal circuito
  - Possiamo collegare più registri allo stesso bus avendo l'accortezza di non abilitarne mai due contemporaneamente
  - Costruiamo un registro direttamente con uscita tri-state ed enable



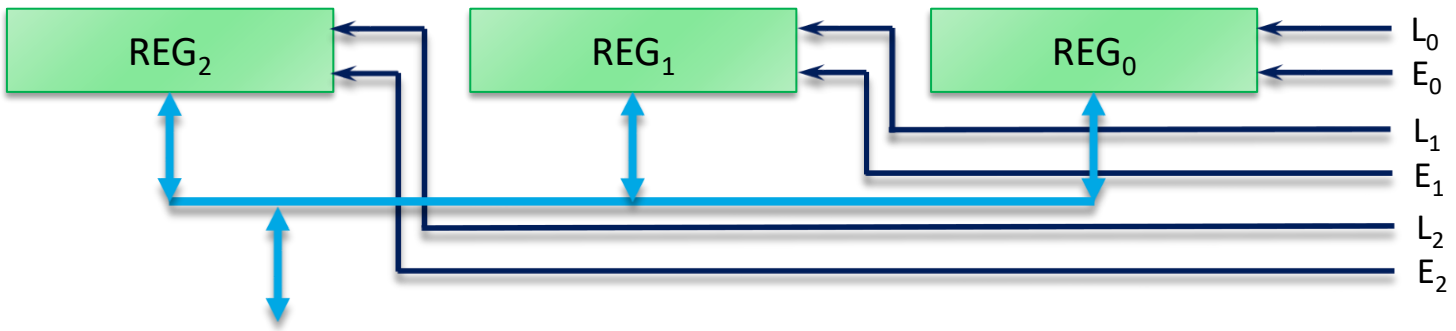
Bus tri-state

- **Una sola linea di bus collega sia le uscite sia gli ingressi**
  - I segnali di enable e di load consentono di specificare quale registro viene messo sul bus, e quali registri caricano il valore
  - Non c'è bisogno di riportare le linee indietro ad un componente centralizzato
  - In pratica il bus è una serie di linee di collegamento che attraversano i registri, in modo bidirezionale



Bus tri-state

- **Una sola linea di bus collega sia le uscite sia gli ingressi**
  - I segnali di enable e di load consentono di specificare quale registro viene messo sul bus, e quali registri caricano il valore
  - Non c'è bisogno di riportare le linee indietro ad un componente centralizzato
  - In pratica il bus è una serie di linee di collegamento che attraversano i registri, in modo bidirezionale



Caratteristiche e complessità

- **Può eseguire gli stessi trasferimenti del bus-multiplexer**
  - Un solo valore può essere trasferito alla volta
  - Più trasferimenti richiedono vari cicli di clock
- **Ma la complessità circuitale è molto inferiore**
  - Inoltre i carichi visti dalle porte sono molto inferiori ai precedenti, pertanto il circuito può essere anche più veloce
- **La scalabilità è molto buona**
  - Aggiungere un registro è solo questione di attaccarsi al bus
  - Il bus tri-state è una scelta praticamente obbligata per il bus di sistema
- **Più complesso il controllo**
  - Occorre assicurarsi che i segnali di enable non siano mai attivi contemporaneamente
  - I segnali di load e di enable potrebbero richedere anche l'uso di una decodifica

Take away

- **Nel data path concorrono varie strutture ricorrenti**
  - Unità aritmetico/logiche
  - Strutture di interconnessione e di condivisione delle risorse
- **Possibili varie ottimizzazioni**
  - Condivisione di linee di controllo tra celle diverse di un data path multi-bit
  - Varie soluzioni archietturali per le interconnessioni (multiplexer, bus e tri-state)
- **Spinta verso generalizzazione e specializzazione**
  - Generalizzazione per poter riutilizzare il data path in diverse applicazioni
  - Specializzazione per poter semplificare ed ottimizzare il progetto