

8 Progetto

Il presente capitolo riguarda la progettazione di un Apparato Centrale Computerizzato (ACC) di una stazione ferroviaria, basato sull'utilizzo del microprocessore DLX e ispirato allo schema di principio degli *ACEI I 0/16 II Serie SDO* senza segnalamento di manovra e con liberazione elastica degli enti (deviatoi). La progettazione non considera le funzionalità evolutive introdotte con il successivo schema di principio V401, come le zone di binario, il FSL, le "C" luminose, il telecomando SCC, il degrado di II° livello con luci blu da deviatoio, il segnalamento plurimo).

L'elaborato prende come base operativa il piano schematico di FIGURA 3.5 e la tabella delle incompatibilità di FIGURA 4.7, su cui si fonda la gestione di tutti i 20 itinerari previsti dalla stazione in esame. Per ragioni di brevità espositiva, si è deciso di rappresentare in dettaglio esclusivamente le reti logiche dell'itinerario 2-II, ossia quello con punto iniziale 2 e punto finale II. Tuttavia, in molte delle reti progettate è presente una rete logica generalizzata, utile a comprendere il funzionamento anche degli altri itinerari.

La parte relativa ai deviatoi si limita a quelli numerati 1 e 2, essendo coinvolti nell'itinerario 2-II. Per ciascun deviatoio si prevedono due modalità operative: la manovra manuale singola, attivabile con click del mouse da parte dell'operatore, e la manovra automatica, comandata direttamente dal processore al momento della formazione dell'itinerario. È inoltre gestita, laddove prevista, l'intallonabilità a comando dei deviatoi per gli itinerari che la prevedono. Ogni deviatoio è dotato di un'unica rete b di bloccamento, che sostituisce i relè degli ACEI b (per ogni CdB di immobilizzazione e senso di marcia) e g.

Per quanto riguarda i CdB, essi vengono codificati durante la formazione dell'itinerario sui binari di corsa, come previsto dallo schema *ACEI I 0/16 II Serie SDO*. In condizioni di impianto a riposo, i CdB sono invece alimentati a corrente fissa.

Una delle principali differenze rispetto allo SdP ACEI è che ogni itinerario possiede una rete logica dedicata per ciascuna fase di gestione (C, R, V, E, Ap, S, mF, va, mS), eliminando il concetto di raggruppamento per punto origine come avveniva per alcuni relè (V, E, Ap, S, mF, va, mS). Il comando della formazione degli itinerari è attuato dall'operatore mediante click del mouse.

Le uniche fasi gestite esclusivamente via software dal microprocessore DLX sono:

1. la valutazione di tutte le incompatibilità tra itinerari (fase R);
2. il comando di manovra dei deviatoi coinvolti nella formazione degli itinerari.

Le restanti fasi (V, E, Ap, S, mF, va, mS) vengono invece implementate tramite reti logiche dedicate in hardware.

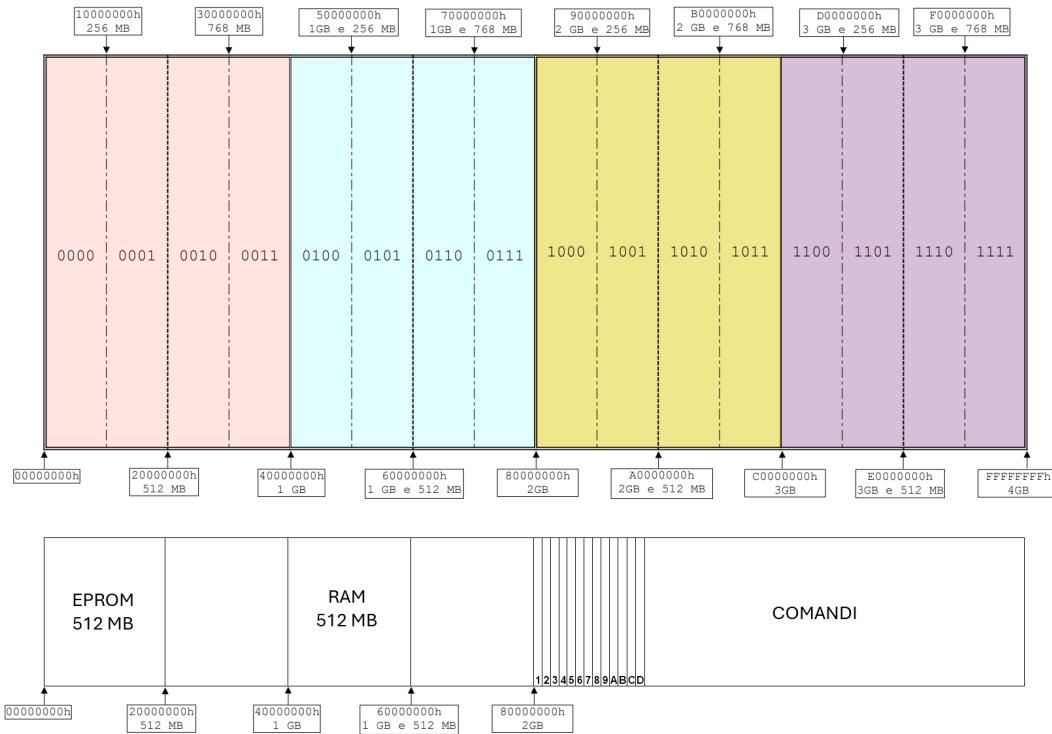
Non è prevista la fase J (indicazione), in quanto ritenuta superflua in logica elettronica. Non sono inoltre presenti reti equivalenti ai relè g di collegamento, poiché anche le incompatibilità che venivano realizzate nell'ACEI con relè g, sono gestite logicamente all'interno del software eseguito dal processore.

Per quanto riguarda le funzioni di soccorso, il progetto contempla solo quelle previste dallo schema di principio *ACEI I 0/16 II Serie SDO*. Queste sono attuabili manualmente dall'operatore sempre tramite click del mouse. Le funzioni previste sono:

- **DJtxY-Z**: distruzione itinerario;
- **TlxY-Z**: liberazione artificiale del punto origine in caso di zona d'approccio occupata;
- **TbxY-Z**: bloccamento del percorso con esclusione della fase V;
- **TmxY-Z**: bloccamento del percorso artificiale, senza attivazione di alcuna fase;
- **TzxY**: accensione manuale dei segnali di avvio;
- **TbDQ**: esclusione del CdB di immobilizzazione di un deviatoio;
- **TcDQ**: esclusione del controllo iniziale per consentire la manovra di un deviatoio;
- **TSP**: disposizione permanente a via impedita di un segnale;
- **Tsm2/3**: soppressione manuale della codifica sui CdB tra i punti 2 e 3 dell'itinerario.

8.1 Mapping

Si precisa che nel presente capitolo, tutti i valori numerici seguiti dalla lettera "h" minuscola sono espressi in formato esadecimale.

Figura 8.1: Spazio di indirizzamento del DLX e mapping del progetto⁸⁶

Si riporta di seguito una tabella riassuntiva delle principali potenze del 2, utile per comprendere in maniera più agevole il mapping che è stato effettuato nel progetto

Potenze di due				
n	2^n	$2^{(10+n)}$	$2^{(20+n)}$	$2^{(30+n)}$
0	1	1K	1M	1G
1	2	2K	2M	2G
2	4	4K	4M	4G
3	8	8K	8M	8G
4	16	16K	16M	16G
5	32	32K	32M	32G
6	64	64K	64M	64G
7	128	128K	128M	128G
8	256	256K	256M	256G
9	512	512K	512M	512G

Figura 8.2: Potenze del 2

⁸⁶I numeri indicati dentro alle colonne colorate rappresentano i 4 bit più significativi (MSB) del bus address (BA31, BA30, BA29, BA28).

EPROM 512 MB

La EPROM di capacità 512 MB è mappata agli indirizzi bassi dello spazio di indirizzamento del DLX: dall'indirizzo 00000000h al 1FFFFFFFh.

Di conseguenza per ottenere un parallelismo a 32 bit si prevede l'impiego di 4 banchi di memoria EPROM da 128 MB ciascuno, come dimostrato nel calcolo seguente:

$$512 \text{ MB} = 2^{29} \rightarrow 2^{29} / 2^2 = 2^{27} \rightarrow 4 \text{ EPROM da 128 MB ciascuna}$$

Tale memoria EPROM contiene, a partire dall'indirizzo 0, il codice assembly che sarà riportato in seguito.

RAM 512 MB

La RAM di capacità 512 MB è mappata agli indirizzi bassi dello spazio di indirizzamento del DLX: dall'indirizzo 40000000h al 5FFFFFFFh.

Di conseguenza per ottenere un parallelismo a 32 bit si prevede l'impiego di 4 banchi di memoria RAM da 128 MB ciascuno, come dimostrato nel calcolo seguente:

$$512 \text{ MB} = 2^{29} \rightarrow 2^{29} / 2^2 = 2^{27} \rightarrow 4 \text{ RAM da 128 MB ciascuna}$$

Tale memoria RAM non sarà utilizzata per salvare dati nel mio progetto, ma se ne prevede comunque la presenza in quanto può risultare utile per eventuali sviluppi futuri e ampliamenti del sistema. Inoltre, la sua integrazione consente di semplificare la compatibilità con architetture standard di microprocessori, facilitando l'eventuale debug tramite strumenti esterni e permettendo, in futuro, la gestione temporanea di variabili di stato, di code di eventi asincroni o di log di sistema. La RAM può infine offrire maggiore flessibilità nel caso si decida di inserire funzionalità avanzate come una diagnostica locale, un buffer per comunicazioni seriali, oppure l'implementazione di algoritmi di gestione più complessi.

Indirizzi dei comandi

READ_STARTUP	→ 80000000h
WRITE_0_STARTUP	→ 80000001h
DESTROY_Cd2-II	→ 80000002h
REGISTRATION_Rd2-II	→ 80000003h
READ_C	→ 80000004h ... 80000007h
READ_R	→ 80000008h ... 8000000Bh
M_DEV1	→ 8000000Ch
M_DEV2	→ 8000000Dh

CHIP SELECT delle memorie e dei comandi

I segnali di CHIP SELECT (**CS**) rappresentano, in questo progetto, la rete logica AND dei segnali di indirizzamento **BA** (bus address) e **BE** (bus enable) che vengono asseriti dal microprocessore DLX durante un ciclo di bus di lettura o scrittura. In particolare, questi segnali vengono espressi senza includere i segnali di controllo **MEMWR** e **MEMRD**, che verranno cablati a parte nelle reti successive. Lo scopo dei **CS** è quello di attivare selettivamente il banco di memoria interessato (tra i quattro presenti) oppure un determinato comando verso l'esterno. Sono quindi fondamentali per abilitare un solo componente alla volta in fase di comunicazione sul bus.

Tali segnali sono espressi secondo una codifica semplificata: ciò significa che l'AND logico include solo i segnali **BE** e **BA** strettamente necessari per distinguere in modo univoco ciascun **CS**. Per ragioni di ottimizzazione logica e risparmio di porte logiche, non sono inclusi i segnali **BA** e **BE** ritenuti ridondanti o sovrabbondanti rispetto allo scopo di identificazione del **CS**, pur restando chiaro il principio generale alla base del loro funzionamento.

<code>CS_EPROM_128MB_0</code>	$= \overline{BA31} \cdot \overline{BA30} \cdot BE0$
<code>CS_EPROM_128MB_1</code>	$= \overline{BA31} \cdot \overline{BA30} \cdot BE1$
<code>CS_EPROM_128MB_2</code>	$= \overline{BA31} \cdot \overline{BA30} \cdot BE2$
<code>CS_EPROM_128MB_3</code>	$= \overline{BA31} \cdot \overline{BA30} \cdot BE3$
<code>CS_RAM_128MB_0</code>	$= \overline{BA31} \cdot BA30 \cdot BE0$
<code>CS_RAM_128MB_1</code>	$= \overline{BA31} \cdot BA30 \cdot BE1$
<code>CS_RAM_128MB_2</code>	$= \overline{BA31} \cdot BA30 \cdot BE2$
<code>CS_RAM_128MB_3</code>	$= \overline{BA31} \cdot BA30 \cdot BE3$
<code>CS_READ_STARTUP</code>	$= BA31 \cdot \overline{BA3} \cdot \overline{BA2} \cdot BE0$
<code>CS_WRITE_0_STARTUP</code>	$= BA31 \cdot \overline{BA3} \cdot \overline{BA2} \cdot BE1$
<code>CS_DESTROY_Cd2-II</code>	$= BA31 \cdot \overline{BA3} \cdot \overline{BA2} \cdot BE2$
<code>CS_REGISTRATION_Rd2-II</code>	$= BA31 \cdot \overline{BA3} \cdot \overline{BA2} \cdot BE3$
<code>CS_READ_C</code>	$= BA31 \cdot \overline{BA3} \cdot BA2$
<code>CS_READ_R</code>	$= BA31 \cdot BA3 \cdot \overline{BA2}$
<code>CS_M_DEV1</code>	$= BA31 \cdot BA3 \cdot BA2 \cdot BE0$
<code>CS_M_DEV2</code>	$= BA31 \cdot BA3 \cdot BA2 \cdot BE1$

8.2 Memoria EPROM

Nella FIGURA 8.2 è rappresentata la rete logica di interfacciamento dei 4 banchi di memoria EPROM. Essi emettono sul bus dati **BD** il contenuto delle loro celle di memoria quando vengono asseriti, durante un ciclo di bus di lettura, contemporaneamente i segnali `CS_EPROM_128MB` corrispondenti e il segnale **MEMRD**.

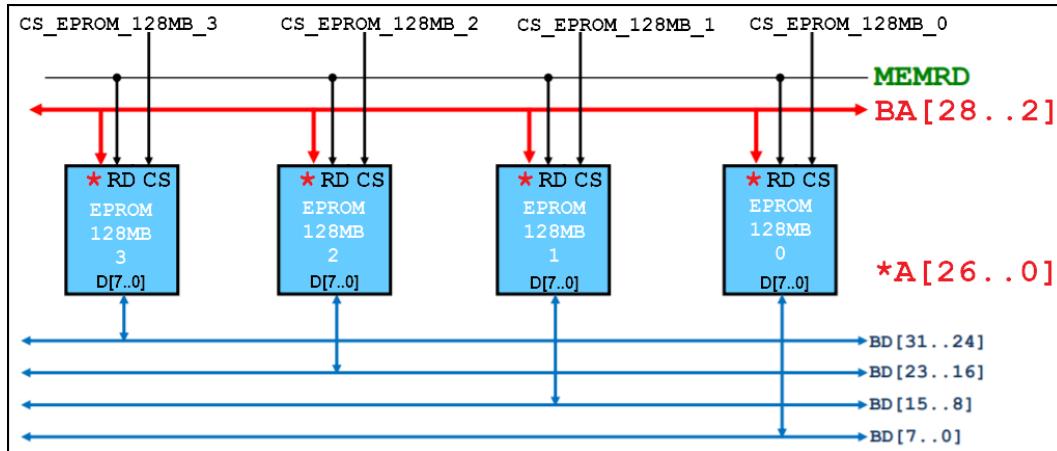


Figura 8.3: EPROM

8.3 Memoria RAM

Nella FIGURA 8.3 è rappresentata la rete logica di interfacciamento dei 4 banchi di memoria RAM. Essi emettono sul bus dati **BD** il contenuto delle loro celle di memoria quando vengono asseriti, durante un ciclo di bus di lettura, contemporaneamente i segnali **CS_RAM_128MB** corrispondenti e il segnale **MEMRD**. Invece, memorizzano nelle loro celle di memoria, il dato presente sul bus **BD**, quando vengono asseriti, durante un ciclo di bus di scrittura, contemporaneamente i segnali **CS_RAM_128MB** corrispondenti e il segnale **MEMWR**.

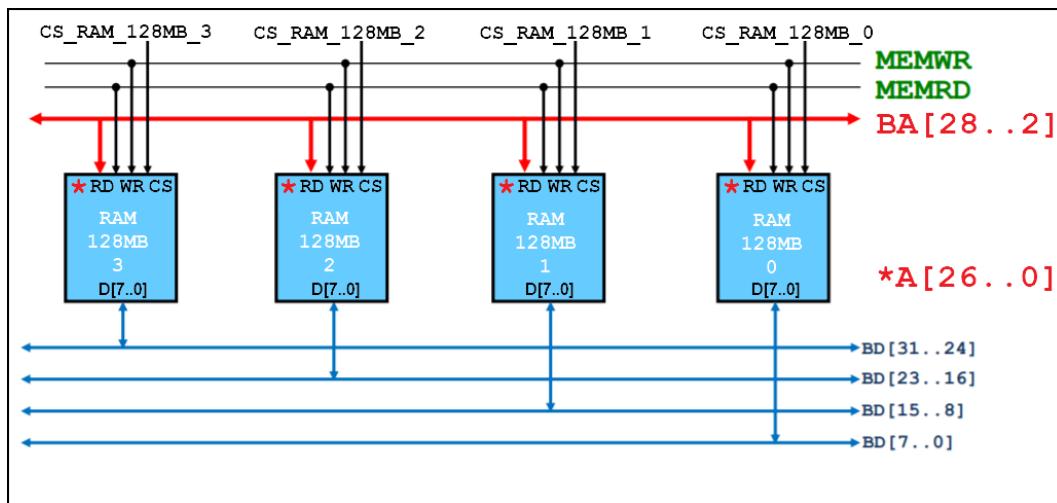


Figura 8.4: RAM

8.4 Rete RESET_SYNC

In ogni progetto elettronico è necessaria una rete di reset, poiché all'accensione non è possibile determinare con certezza lo stato logico (0 o 1) dei componenti digitali: esso, infatti, dipende da condizioni iniziali casuali dovute a variazioni fisiche

e elettriche nei semiconduttori, come cariche residue nei nodi dei transistor in tecnologia CMOS. Per questo motivo, all'accensione del processore viene asserito un segnale asincrono di **RESET** per almeno un ciclo di clock. L'applicazione di un segnale asincrono, può portare a problemi di metastabilità⁸⁷ nel momento in cui tale segnale viene posto al valore 0 (cioè in uscita dal reset). Le problematiche sono analoghe a quelle evidenziate durante il campionamento di un segnale che non rispetta i tempi di setup e hold. Una soluzione che garantisce un'uscita sincrona dal reset, è quella esposta in FIGURA 8.4. Il segnale **RESET_SYNC** sarà utilizzato per inizializzare in uno stato noto non solo il microprocessore DLX, ma anche tutte le reti logiche del progetto, garantendo così un corretto avviamento del sistema in uno stato noto.

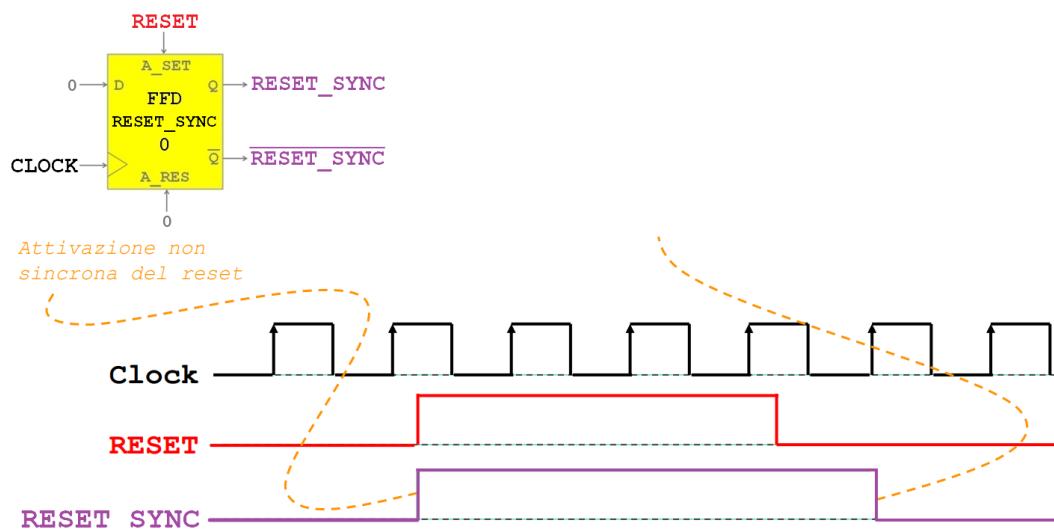


Figura 8.5: Rete **RESET_SYNC**

8.5 Rete STARTUP

La FIGURA 8.5. raffigura una rete ausiliaria necessaria per verificare se siamo all'istruzione 0 poiché è in corso l'accensione del sistema oppure è avvenuto un interrupt. Nello specifico, il FFD viene inizializzato a 1 all'avvio del sistema tramite l'ingresso asincrono **A_SET**, che riceve il segnale **RESET_SYNC**. In questo modo, il FFD memorizza il valore logico 1 finché non interviene un'operazione di scrittura da parte del processore sullo stesso FFD. Tale operazione viene effettuata attraverso un ciclo di bus di scrittura mediante l'istruzione **SB** (Store Byte), la qua-

⁸⁷ La metastabilità è una condizione anomala che può verificarsi nei circuiti digitali sequenziali, in particolare nei flip flop, quando un segnale di ingresso cambia in prossimità del fronte di clock, cioè in un momento non sufficientemente lontano dai tempi di setup o hold richiesti. In queste situazioni, il flip-flop può entrare in uno stato instabile in cui l'uscita non è né logicamente alta (1) né bassa (0), ma rimane in una condizione intermedia per un tempo imprevedibile prima di stabilizzarsi. Questo comportamento può causare malfunzionamenti nella logica sequenziale, compromettendo la correttezza del funzionamento del sistema. La metastabilità è un fenomeno fisico inevitabile, ma può essere mitigata con l'uso di tecniche di sincronizzazione, come l'inserimento di flip-flop in cascata (metastability hardening).

le asserisce il comando `CS_WRITE_0_STARTUP`. Questo segnale è cablato direttamente sul bus di indirizzo del multiplexer (MUX), e la sua attivazione permette al MUX di instradare verso l'ingresso D del FFD il valore logico 0. Esso viene campionato dal FFD sul fronte di salita del segnale di controllo `MEMWR` (Memory Write), aggiornando quindi in modo permanente il contenuto del flip flop a 0. Quando il segnale `CS_WRITE_0_STARTUP` non è asserito, il MUX instrada sull'ingresso D del FFD il segnale `STARTUP` in retroazione, permettendo così al flip-flop di "fare memoria", ossia di campionare nuovamente il proprio stesso valore durante eventuali cicli di scrittura non indirizzati a `CS_WRITE_0_STARTUP`, mantenendo invariato il contenuto memorizzato.

Il valore memorizzato nel FFD può essere letto dal processore attraverso un ciclo di lettura, mediante l'asserzione del segnale `CS_READ_STARTUP` e del segnale di controllo `MEMRD` (Memory Read). L'AND logico di entrambi abilita le uscite dei three-state⁸⁸ per consentire l'immissione sul bus dati `BD0` del bit di `STARTUP`, mentre sugli altri 7 bit del bus (`BD1-BD7`) vengono posti zeri logici, formando così un byte che può essere letto con l'istruzione `LB` (Load Byte). Il comando `CS_READ_STARTUP`, contenuto nell'istruzione 0 memorizzata nella EPROM, viene emesso ogni volta, al verificarsi di un interrupt mentre `CS_WRITE_0_STARTUP` viene utilizzato una sola volta, al termine delle operazioni software di inizializzazione effettuate dopo la prima accensione. In questo modo si garantisce un corretto passaggio dallo stato iniziale a quello operativo del sistema.

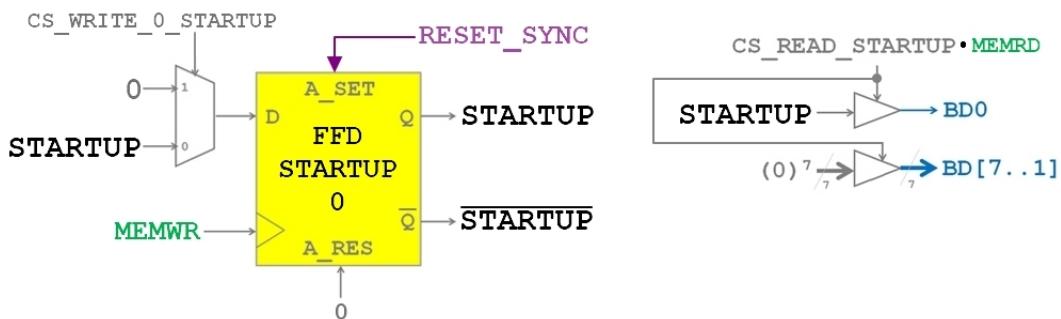


Figura 8.6: Rete STARTUP

8.6 Rete C

L'operatore avvia la formazione dell'itinerario mediante un click del mouse, generando un segnale digitale che passa a livello alto (1) durante la pressione del ta-

⁸⁸Un *three-state* (o *tristate*) è un tipo speciale di uscita logica che può assumere tre stati distinti: livello logico alto ("1"), livello logico basso ("0") oppure alta impedenza (spesso indicata come "Z" o "stato ignoto"). I primi due rappresentano i normali stati logici, mentre il terzo, l'alta impedenza, equivale a disconnettere elettricamente il segnale dal circuito, come se non fosse collegato. Questo è particolarmente utile quando più dispositivi devono condividere una stessa linea, come ad esempio un bus dati. Nel progetto, i three-state verranno impiegati per gestire tutti i segnali che condividono il bus dati, in modo da evitare conflitti tra più sorgenti e garantire che un solo componente per volta possa scrivere sul bus, mentre gli altri rimangono in stato di alta impedenza.

sto e ritorna a livello basso (0) al rilascio. Collegando questo segnale all'ingresso di clock del FFD, si effettua il campionamento sul fronte di salita ($0 \rightarrow 1$) del segnale, convertendo così un interrupt di tipo "a fronte" in un interrupt "a livello". Il flip-flop memorizza quindi uno stato stabile a livello logico alto, segnalando l'avvenuta richiesta di creazione dell'itinerario. In particolare, il segnale **Cd2-II** (relativo all'itinerario di punto iniziale 2 e finale II) genera un interrupt per il DLX attraverso l'asserzione del segnale **INT**.

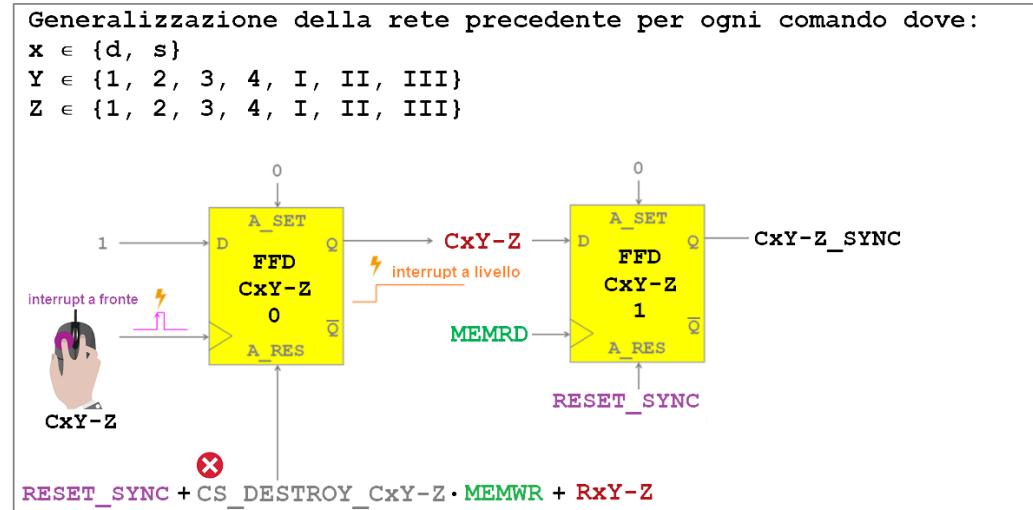
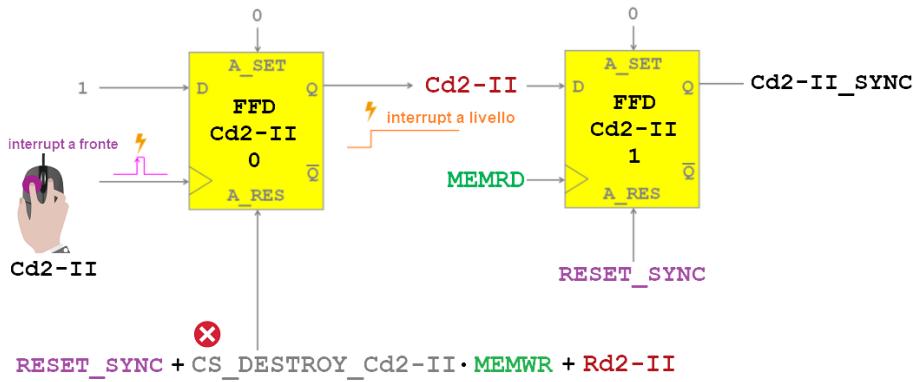


Figura 8.7: Rete C

Poiché **Cd2-II** è un segnale asincrono rispetto al clock del processore, viene sincronizzato mediante un secondo FFD che lo campiona, permettendo di ottenere il segnale **Cd2-II_SYNC**, valido all'interno del dominio di clock del DLX. Questo segnale viene letto attraverso un ciclo di bus di lettura che asserisce il comando **CS_READ_C** mediante un'istruzione **LW** (Load Word), abilitando le porte three-state. Ogni itinerario ha una posizione specifica nel bus dati, utile per la successiva fase di elaborazione in cui il processore valuterà le eventuali incompatibilità tra gli itinerari già attivi e quello richiesto.

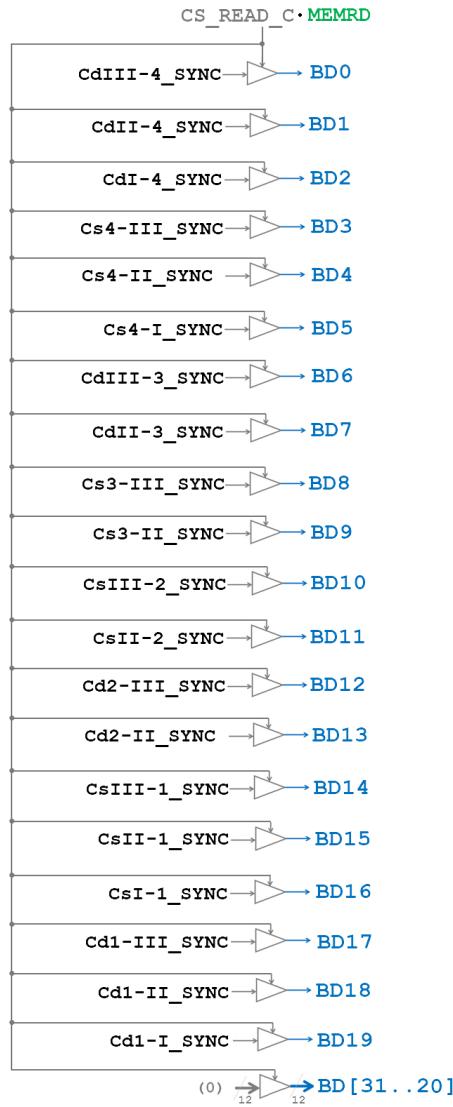


Figura 8.8: Rete CS_READ_C

Il flip-flop che memorizza il comando **Cd2-II** è inizializzato a 0 dal segnale **RESET_SYNC** all'avvio del sistema, e memorizza 1 quando l'operatore richiede l'attivazione dell'itinerario. Tale valore verrà poi forzato a 0 in due casi:

1. attraverso un ciclo di scrittura al comando **CS_DESTROY_Cd2-II** (istruzione SB), se l'itinerario 2-II risulta incompatibile con altri già attivi;
2. automaticamente all'attivazione dell'itinerario, quando il flip-flop **Rd2-II** assume valore 1 registrando l'avvenuta attuazione. Poiché **Rd2-II** rimane assunto per tutto il tempo in cui l'itinerario è formato, ogni ulteriore click del mouse, sul comando **Cd2-II**, viene ignorato, grazie alla presenza del segnale **Rd2-II** sull'ingresso asincrono **A_RES** del FFD.

Si riporta, inoltre, una rete generalizzata che rappresenta lo stesso schema per un qualsiasi altro itinerario xY-Z.

A valle, una rete logica finale effettua un OR di tutti i segnali di comandi itine-

rario che devono generare un interrupt. Questa rete è posta in AND con il segnale STARTUP negato: in tale modo, tutti gli interrupt vengono ignorati dal microprocessore DLX finché il sistema non ha completato l'inizializzazione (ovvero finché il segnale STARTUP è ancora alto).

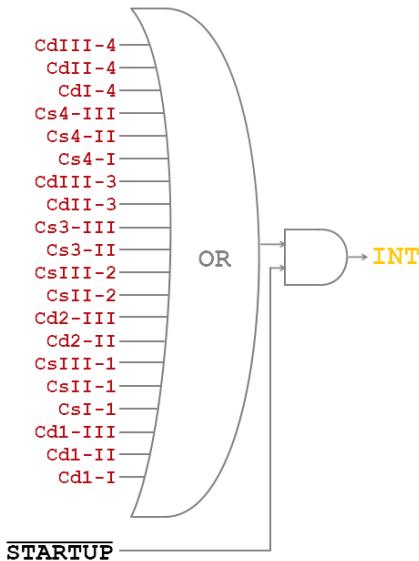


Figura 8.9: Rete INT

8.7 Rete Deviatoi

La FIGURA 8.7 mostra le reti logiche che implementano l'interfaccia tra il microprocessore DLX e gli attuatori dei deviatoi, tramite l'utilizzo di I/O PORT dedicate alla gestione delle manovre. In particolare, vengono illustrati nel dettaglio i deviatoi 1 e 2, coinvolti nella formazione dell'itinerario 2-II, e si presenta anche una rete logica generalizzata estendibile a qualsiasi altro deviatoio del piano schematico.

Le manovre dei deviatoi possono essere finalizzate ad assumere una posizione normale o una posizione rovescia.

Gli ingressi principali della I/O PORT sono:

- **M**, se a 1, abilita la ricezione dei segnali di manovra (**MN** e **MR**) mentre se a 0, la porta ignora tali ingressi;
- **MN** e **MR**, selezionano rispettivamente la manovra normale o rovescia del deviatoio, quando attivi;
- **FLAG**, segnala che le uscite della porta sono stabili e pronte per essere lette;
- **KEDa** e **KEDb**, quando posti a 1 indicano l'inserzione (diseccitazione) degli elettromagneti rispettivamente delle casse a e b;
- **KDN** e **KDR**, quando posti a 1 indicano rispettivamente il controllo di posizione normale o rovescia cumulativa del deviatoio (AND logico delle casse a e b);

- **EDa_ON** e **EDb_ON**, quando posti a 1 attivano rispettivamente la disecitazione degli elettromagneti nelle casse a e b;
- **EDa_OFF** e **EDb_OFF**, quando posti a 1 attivano rispettivamente l'eccitazione degli elettromagneti nelle casse a e b.

Nel dialogo tra il processore e un attuatore attraverso una porta I/O, è fondamentale garantire una corretta sincronizzazione per evitare perdite o sovrapposizioni di dati. A questo scopo entrano in gioco due segnali essenziali: **STB** (Strobe) e **ACK** (Acknowledge). Quando l'attuatore deve scrivere un dato nel buffer della I/O, porta **STB** a livello alto (1) per indicare l'inizio dell'operazione di scrittura. Una volta completata la scrittura, riporta **STB** a livello basso (0) per segnalare che l'operazione è terminata. In modo simmetrico, quando l'attuatore è pronto a leggere un dato presente nella porta, asserisce il segnale **ACK** (ponendolo a 1), a indicare che sta prelevando il dato. Al termine della lettura, **ACK** viene azzerato (0). Questi due segnali sono quindi fondamentali per coordinare correttamente il flusso di dati tra l'attuatore e la porta, evitando condizioni di race o letture/scritture non sincronizzate.

Di seguito si descrive in dettaglio, a campione, il funzionamento della rete del deviatoio 1.

Il segnale **M_DEV1** viene asseritoognqualvolta si richieda una manovra (manuale/automatica) e il deviatoio non è bloccato dal segnale **bD1** (il blocco avviene quando **bD1 = 0**). **M_DEV1** è collegato direttamente all'ingresso **M** della I/O PORT per abilitare la ricezione dei comandi **MN** e **MR**.

La manovra manuale (FIGURA 8.11) viene attivata mediante il click dell'operatore, generando un fronte in salita campionato da un FFD, come già visto per i comandi di itinerario. Sono previsti due segnali distinti:

- **MANUAL_MN_DEV1**, per la manovra manuale in posizione normale;
- **MANUAL_MR_DEV1**, per la manovra manuale in posizione rovescia.

Questi segnali abilitano le rispettive porte three-state, che mettono sulla linea **MN** o **MR** il bit 1 necessario alla manovra. Il reset dei segnali manuali avviene:

- all'avvio del sistema mediante il segnale **RESET_SYNC**;
- automaticamente, se il deviatoio raggiunge la posizione comandata mediante l'assunzione del valore 1 del segnale di controllo;
- oppure, fintantoché il deviatoio è bloccato da **bD1 = 0**.

Il DLX può comandare automaticamente una manovra scrivendo un byte sulla porta, con un ciclo di bus di scrittura (istruzione **SB**), che asserisce **CS_M_DEV1** e **MEMWR**. In tale ciclo:

- se il DLX vuole posizionare il deviatoio 1 in posizione normale, imposta **BD0 = 1**;
- se vuole posizionarlo rovescio, imposta **BD1 = 1**.

Tali segnali attivano gli ingressi **MN** o **MR** della I/O PORT, secondo il bit impostato, esattamente come nella manovra manuale.

La rete implementa due importanti controlli logici (vedi FIGURA 8.9):

1. condizione di posizione iniziale, la I/O PORT accetta una manovra solo se il controllo di posizione opposto (**KDR** o **KDN**) è attivo. In caso contrario, la manovra viene ignorata. Questa condizione è ottenuta mediante la prima coppia a sinistra di porte logiche AND. Infatti, anche nel caso in cui il segnale **M_DEV1** venga asserito da un comando di manovra non valido, la I/O port ignorerà tale manovra se rileva che i segnali **MN** e **MR** sono entrambi a 0 (ovvero nessun bit attivo) grazie alla forzatura impressa dai gate AND. Si ipotizza che la logica interna della porta I/O scarti tale condizione non univoca, ad esempio tramite una semplice porta logica EXOR, che consenta il passaggio solo quando uno dei due ingressi è alto e l'altro basso. Inoltre, in presenza di guasti (ad esempio, assenza iniziale di controllo), la condizione verificata dalla prima coppia di AND può essere bypassata mediante l'attivazione della funzione di soccorso **TcD1**;
2. condizione di blocco, un'ulteriore coppia di AND forza gli ingressi **MN** e **MR** a zero se il deviatoio è bloccato (**bD1** = 0), impedendo qualsiasi manovra.

Infine, per quanto riguarda il deviatoio 2, sulla porta I/O è cablato il segnale **Sd2-II**, che comanda la diseccitazione dell'elettromagnete della cassa 2b quando si forma l'itinerario 2-II. Un secondo segnale, ottenuto dalla negazione di **Sd2-II** AND **bD2**, rimuove l'elettromagnete solo dopo la distruzione dell'itinerario (caduta di **Sd2-II**) e dopo il transito del treno, ovvero quando **bD2** torna alto. Gli ingressi **EDa** e **EDb** di ciascun deviatoio saranno dunque cablati con tutti i segnali degli itinerari che prevedono una diseccitazione/eccitazione di un elettromagnete. Poiché l'itinerario 2-II coinvolge solo il deviatoio 2b, la figura raffigura solo questo caso come esempio.

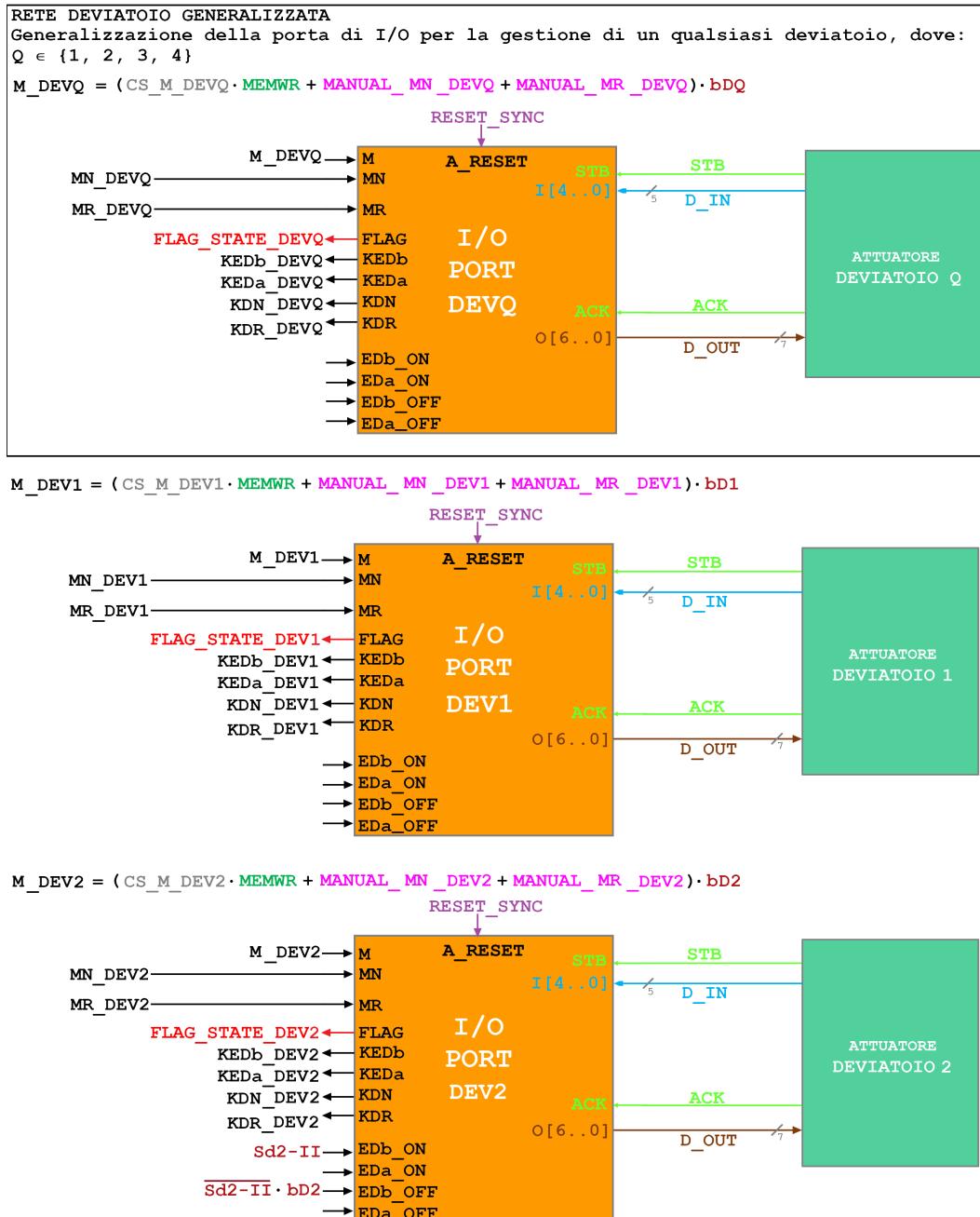


Figura 8.10: Rete Deviatoio

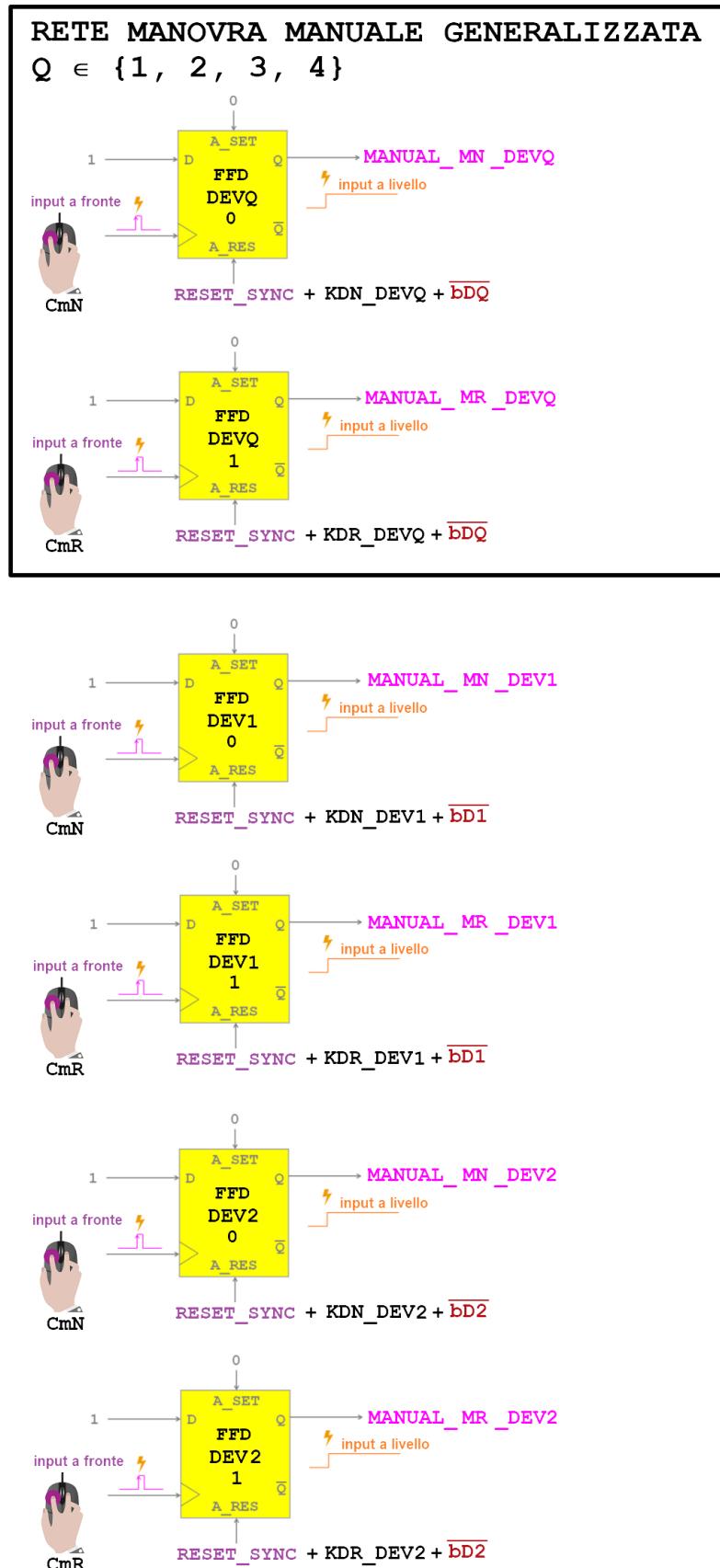


Figura 8.11: Rete manovra manuale

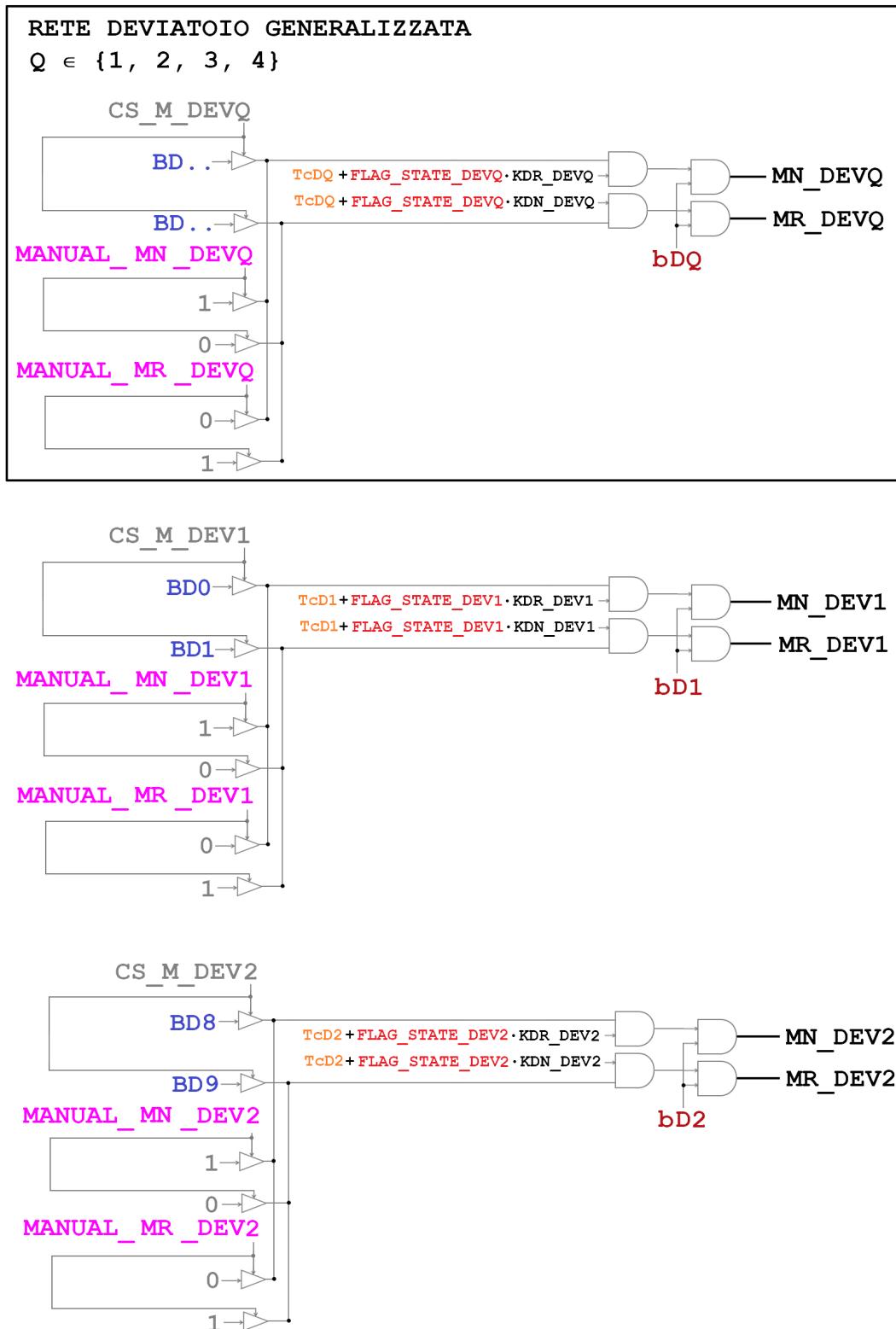


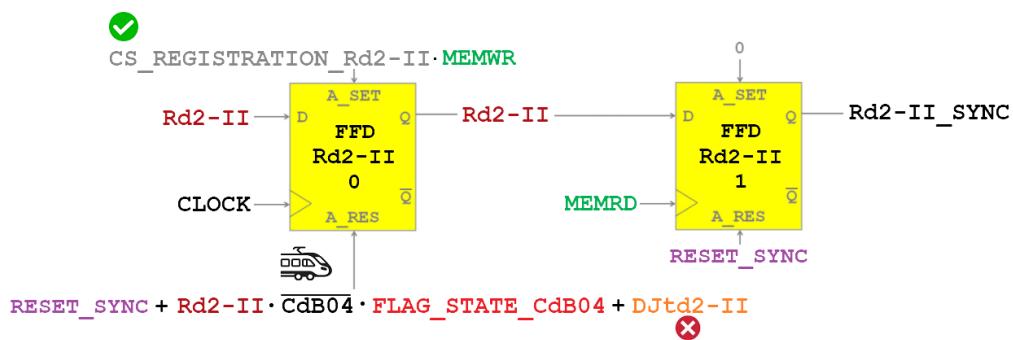
Figura 8.12: Rete MN MR

8.8 Rete R

La rete R è responsabile della memorizzazione dello stato di avvenuta registrazione degli itinerari, ovvero quando un determinato itinerario è stato autorizzato dal microprocessore DLX in quanto compatibile con quelli già attivi. Ad ogni itinerario sono associati due flip-flop D:

- il primo memorizza un 1 logico quando l'itinerario è registrato;
- il secondo ha il compito di sincronizzare tale informazione per consentire al DLX una lettura sicura e priva di fenomeni di metastabilità (come avviene anche nella rete C).

Nella FIGURA 8.8 si rappresenta la rete specifica per l'itinerario 2-II, accompagnata da una versione generalizzata, applicabile a qualsiasi altro itinerario.



Generalizzazione della rete precedente per ogni itinerario, dove:

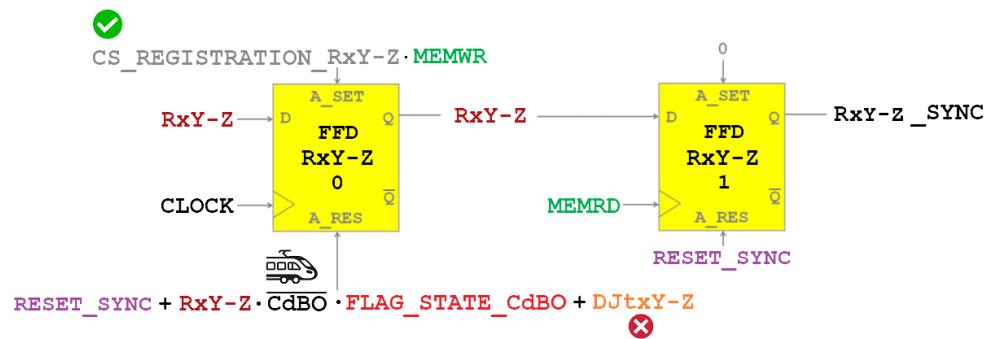
$$\begin{aligned}x &\in \{d, s\} \\Y &\in \{1, 2, 3, 4, I, II, III\} \\Z &\in \{1, 2, 3, 4, I, II, III\} \\O &\in \{01, 04, 03, 06, 07, 09, 12, 11, 14\}\end{aligned}$$


Figura 8.13: Rete R

Il primo FFD viene inizializzato a 0 all'accensione del sistema, tramite il segnale **RESET_SYNC** applicato al suo ingresso **A_RES**.

Quando l'operatore comanda la formazione dell'itinerario 2-II, si attiva la relativa rete di comando **Cd2-II**, che genera un interrupt per il DLX. Il microprocessore

gestisce tale evento all'interno dell'handler denominato **R_Cd2II**, dove valuta, in base ai segnali R già attivi, la compatibilità dell'itinerario richiesto.

Se l'itinerario risulta compatibile, il DLX effettua un ciclo di scrittura sul bus (istruzione **SB**), asserendo il comando **CS_REGISTRATION_Rd2-II**, che porta il primo FFD a 1, attivando quindi il segnale **Rd2-II**.

Il valore 1 rimane memorizzato fino a uno dei seguenti eventi:

- il treno "pesta" il circuito di binario di occupazione permanente relativo all'itinerario (nel caso del 2-II si tratta come da piano schematico del CdB 04);
- attivazione della funzione di soccorso **DJtd2-II** (corrispondente, negli impianti ACEI, all'estrazione del pulsante d'itinerario) nel caso in cui l'operatore voglia cancellare l'itinerario (se si è sbagliato a crearlo).

In entrambi i casi, il FFD viene resettato a 0, disattivando **Rd2-II**. Questo evento determina la distruzione dell'itinerario, poiché tutte le reti logiche delle fasi successive (V, b, Ap, S, mF, va, mS) tornano automaticamente allo stato di riposo. Tale comportamento è garantito dal fatto che ciascuna rete è condizionata in AND con il segnale **Rd2-II**.

Per garantire una lettura sicura e sincronizzata dello stato degli itinerari, il valore contenuto nel primo FFD viene propagato al secondo FFD tramite un segnale di clock controllato dal segnale **MEMRD**, ottenendo così una versione sincronizzata del segnale R.

Il DLX può leggere contemporaneamente tutti i 20 segnali R, relativi ai diversi itinerari, attraverso un ciclo di lettura (istruzione **SW**) al comando **CS_READ_R**. Quest'ultimo abilita i buffer three-state che immettono sul bus dati i segnali sincronizzati R.

Anche in questo caso, la disposizione dei segnali R sul bus dati (**BD**) riveste un ruolo fondamentale per la corretta esecuzione, via software, della verifica delle incompatibilità: ad esempio, l'itinerario 2-II è cablato coerentemente nella rete C, nella rete R e nella tabella delle incompatibilità memorizzata nei registri del DLX, occupando la quattordicesima posizione logica, corrispondente alla tredicesima linea dati (considerando un indice a partire da zero).

Nei paragrafi successivi, si descrivono tutte le principali reti logiche dedicate alle fasi ACEI V, b, E, Ap, S, mF, va, mS relative alla formazione dell'itinerario 2-II, senza generalizzazione. Si rammenta che per ogni itinerario, è presente una rete specifica, che non accomuna i punti origine, in modo da consentire una gestione indipendente e dedicata di ogni itinerario. Tali reti descritte di seguito sono cablate con un contatto del segnale **Rd2-II** in AND, al fine di essere automaticamente azzerate (portate a 0) quando l'itinerario viene distrutto (ossia quando **Rd2-II** = 0).

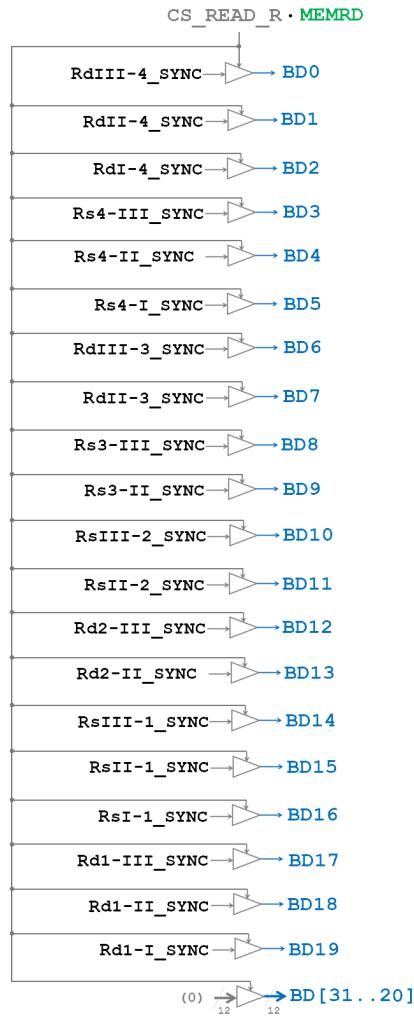


Figura 8.14: Rete CS_READ_R

8.9 Rete V

La rete **Vd2-II** ha il compito di verificare la libertà dei circuiti di binario (CdB) richiesti liberi secondo la tabella delle condizioni per l'itinerario 2-II, comprendendo sia i CdB di percorso che quelli di uscita. Solo nel caso in cui tutti i CdB siano liberi, e il segnale **Rd2-II** sia attivo (1), la rete V si pone anch'essa a 1. In caso contrario, rimane a 0, impedendo la prosecuzione della formazione itinerario.

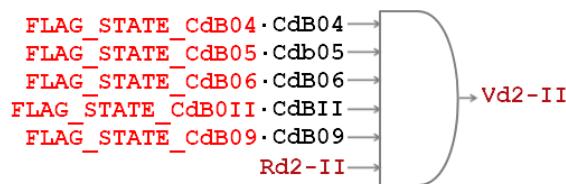


Figura 8.15: Rete V

8.10 Rete b

La rete **bD2** è dedicata al bloccamento del deviatoio 2, e il segnale corrispondente **bD2** si pone a 1 solo quando sono soddisfatte tutte le seguenti condizioni:

1. i CdB di immobilizzazione associati al deviatoio 2 risultano liberi, oppure in presenza di occupazione indebita, la condizione viene scartata tramite la funzione di soccorso **TbD2** attivata dall'operatore;
2. tutti i segnali Ap degli itinerari che utilizzano il deviatoio 2 sono alti, condizione che consente la liberazione elastica del deviatoio.
3. tutti i segnali R degli itinerari che coinvolgono il deviatoio 2 sono a 0, cioè nessun itinerario sta impiegando attivamente quella risorsa;
4. non vi è conflitto tra comandi manuali e automatici, né tra comandi manuali incoerenti (normale e rovescio simultanei). Tali condizioni sono rilevate da porte NAND che forzano **bD2** a 0 in caso di tali errori;
5. il segnale **Tmd2-II** vale 0. Se viene, infatti, attivato a 1 impone $bD2 = 0$, bloccando il deviatoio. Questo è utile per azioni di emergenza attivate tramite la funzione di soccorso **Tmd2-II**, che permette di fare una bloccamento "artificiale" del percorso.

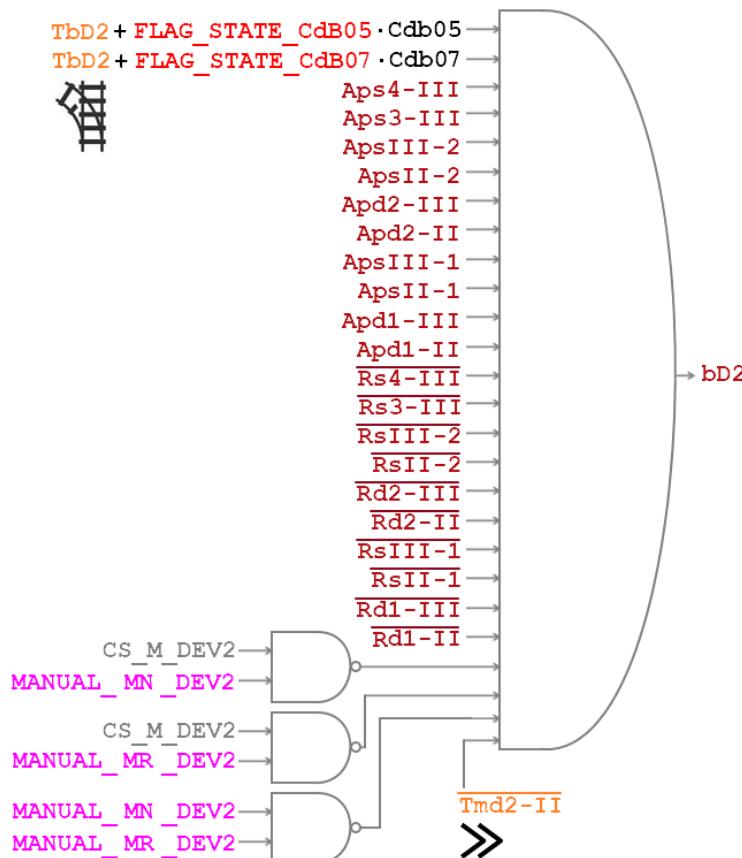


Figura 8.16: Rete b

8.11 Rete E

La rete **Ed2-II** si attiva quando si verificano contemporaneamente le seguenti condizioni:

1. è stato registrato il comando itinerario (**Rd2-II** = 1);
2. si è verificata positivamente la libertà della via (**Vd2-II** = 1), oppure questa è stata scartata tramite funzione di soccorso **Tbd2-II** in caso di guasto ai CdB;
3. è stato ottenuto il controllo coerente dei deviatoi coinvolti (**KDN_DEV1** = 1 e **KDN_DEV2** = 1) e il relativo bloccamento (**bD1** = 0 e **bD2** = 0).

Il passaggio di **Ed2-II** a 1 consente l'attivazione della successiva fase Ap, bloccando il punto di origine dell'itinerario.

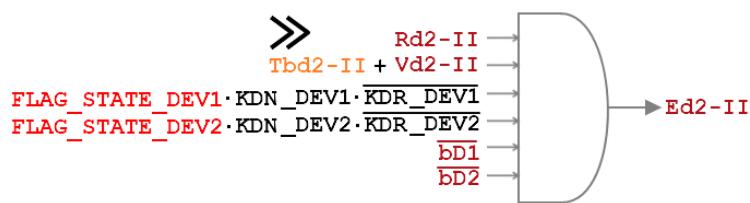


Figura 8.17: Rete E

8.12 Rete Ap

Il segnale **Apd2-II** si comporta come i segnali Ap negli apparati ACEI: è a 1 quando l'itinerario non è bloccato, e passa a 0 quando la rete E si attiva, indicando l'avvenuto bloccamento. **Apd2-II** ritorna a 1 solo quando simultaneamente:

1. si verifica la distruzione dell'itinerario (**Rd2-II** = 0), a seguito del passaggio del treno sul CdB 04, di occupazione permanente per l'itinerario 2-II;
2. il treno deve aver già transitato completamente sul CdB04 (è richiesta infatti la condizione di libertà del segnale CdB04);
3. la zona di approccio è libera, ovvero le tratte di blocco a monte (0002 e 002) sono libere (segnaletica APPROACH_ZONE = 1). Se la zona di approccio non è libera, la liberazione del segnale Ap richiede due azioni successive dell'operatore: prima la distruzione itinerario (**DJtd2-II**), poi la funzione di soccorso **Tld2-II**. Questa procedura è stata progettata anche negli ACEI al fine di impedire lo sbloccamento dei deviatoi con una sola distruzione dell'itinerario (manipolazione di **DJtd2-II**) se il treno è ravvicinato al segnale di protezione (tratte 0002 e/o 002 occupate).

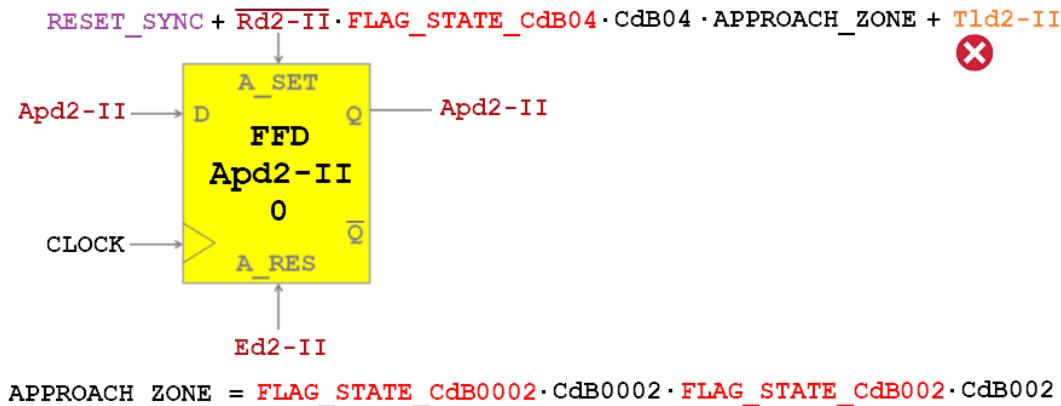


Figura 8.18: Rete Ap

8.13 Rete S

Questa rete verifica che tutte le fasi precedenti (inclusa Ap) siano attive. L'uscita **Sd2-II** abilita la diseccitazione dell'elettromagnete nella cassa 2b (vedi rete deviatoi) e permette l'intallonabilità a comando, propedeutica all'attivazione del segnale.

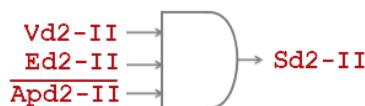


Figura 8.19: Rete S

8.14 Rete mF

Il segnale **mFd2-II** si pone a 1 dopo l'avvenuto controllo positivo dell'elettromagnete 2b per l'itinerario 2-II. Questo segnale consente la fase di codifica dei CdB, avviando la generazione del codice appropriato sui binari di corsa (vedi rete CdB).



Figura 8.20: Rete mF

8.15 Rete va

Quando tutti i CdB coinvolti nell'itinerario 2-II risultano correttamente codificati (segnali vd a 1), la rete **vad2-II** passa a 1. Questo segnale autorizza la successiva fase di manovra e selezione dell'aspetto dei segnali ferroviari B e Avv.B.

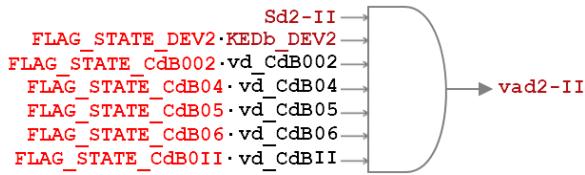


Figura 8.21: Rete va

8.16 Rete mS

La rete finale, **mSd2-II**, si attiva solo dopo il completamento di tutte le fasi precedenti, in particolare quando **vad2-II** è alto. Il segnale **mSd2-II** chiude il ciclo di formazione dell’itinerario, consentendo la manovra dei segnali ferroviari e selezionando l’aspetto corretto sia del segnale ferroviario B (protezione) che del relativo avviso associato.

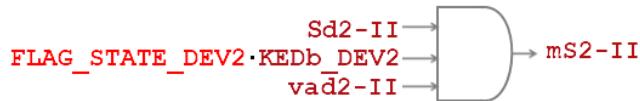


Figura 8.22: Rete mS

8.17 Rete CdB

La I/O PORT di ciascun CdB dispone dei seguenti ingressi:

- Ck/d_OFF e Ck/s_OFF, disattivano rispettivamente qualsiasi codifica per itinerario destro o sinistro precedentemente attivata, mantenendo l’alimentazione fissa del CdB;
- Ck/d75, Ck/d120, Ck/d180, Ck/d270, selezionano la codifica con frequenze specifiche per itinerari destri;
- Ck/s75, Ck/s120, Ck/s180, Ck/s270, selezionano le codifiche equivalenti per itinerari sinistri;
- FLAG, segnale che indica quando le uscite della I/O PORT sono in uno stato stabile per poter essere lette (non in fase di transizione);
- CdB_STATE, segnale d’uscita che indica se il CdB è occupato (0) o libero (1);
- vd e vs, segnali di controllo che indicano l’effettiva presenza della codifica contro-treno, rispettivamente per itinerari destri e sinistri.

Di seguito si descrive la rete che rappresenta il comportamento logico e funzionale della I/O PORT associata al **CdB04**, circuito di binario di occupazione permanente dell’itinerario 2-II. Viene presentata come rete campione, da cui derivare per analogia tutte le altre reti I/O PORT dei CdB.

Durante la formazione dell’itinerario 2-II, il segnale **mFd2-II** si porta a livello alto. Questo segnale rappresenta il comando logico per attivare la codifica corretta.

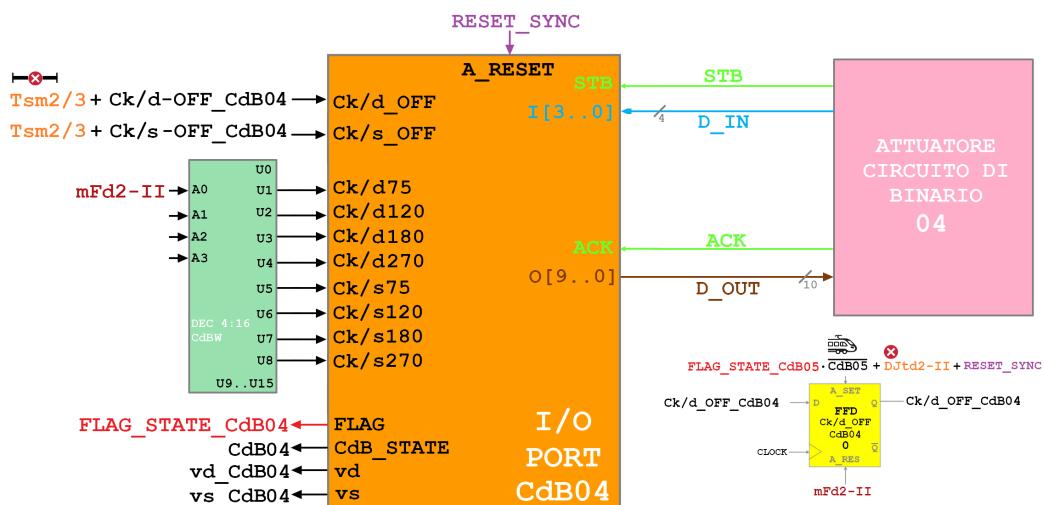
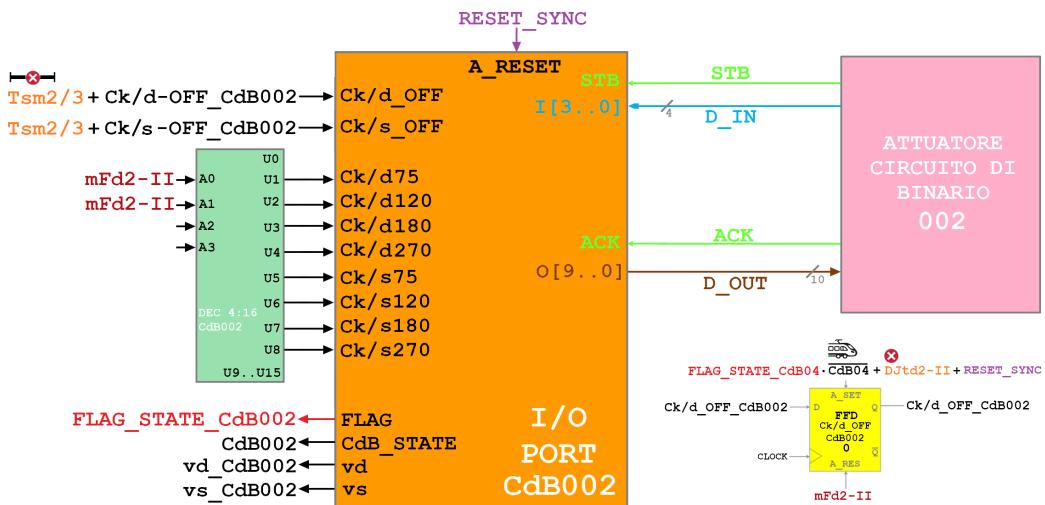
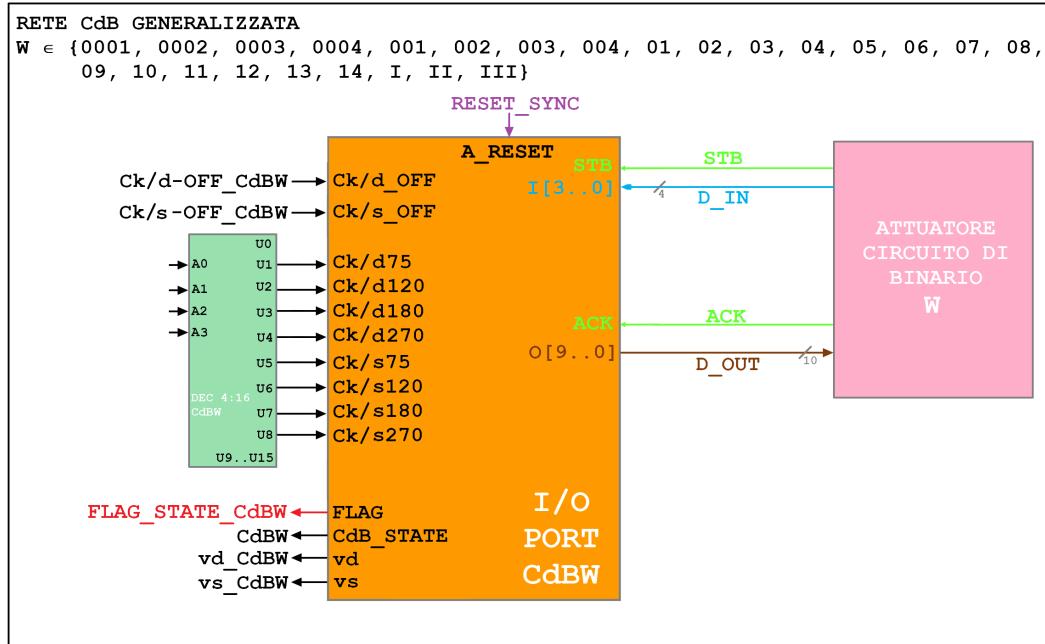


Figura 8.23: CdB

Infatti, **mFd2-II** (= 1) viene inviato al decoder binario (3:8), che converte il valore 0001 attivando l'uscita 1. L'uscita 1 è collegata all'ingresso Ck/d75 della I/O PORT, pertanto il CdB04 viene codificato a 75 per l'itinerario destro. Tutti gli altri segnali di codifica restano a 0. Vengono omessi i segnali **mF** relativi agli altri itinerari, che dovranno essere opportunamente collegati agli ingressi di indirizzo del decoder (**A0...A3**) per garantire, in base all'itinerario attivo, la corretta selezione del codice di codifica. Quando **mFd2-II** si dissecchia (per distruzione dell'itinerario), il decoder commuta automaticamente su **U0** (uscita non connessa), portando tutti gli ingressi della I/O PORT a 0. I selettori di codice della porta sono sensibili al fronte di salita, quindi la codifica rimane attiva fino all'arrivo esplicito di un comando di OFF, evitando instabilità o rimozioni premature.

La rimozione della codifica da Ck/d75 (CdB04) avviene non contestualmente alla distruzione dell'itinerario, ma solo quando il treno ha completamente abbandonato il CdB04, transitando su CdB05. Le condizioni che pongono a 1 il segnale **Ck/d_OFF_CdB04** (che disattiva la codifica) sono una delle seguenti opzioni riportate:

- CdB05 occupato (= 0);
- distruzione manuale dell'itinerario mediante comando operatore (tramite funzione di soccorso **DJtd2-II**);
- funzione di soppressione manuale della codifica tramite funzione di soccorso **Tm2/3** attivabile sempre con click del mouse.

Il segnale **Ck/d_OFF_CdB04** è implementato con un flip-flop D, inizializzato a 1 tramite **RESET_SYNC** all'avvio del sistema. Questo garantisce che inizialmente nessun CdB sia codificato.

8.18 Rete Segnali

La FIGURA 8.19 mostra la rete logica relativa alla I/O PORT di comando dei segnali ferroviari, che agisce da interfaccia tra le reti interne del sistema e l'attuatore fisico del segnale. Tale I/O PORT è progettata per generare i corretti comandi di aspetto in base allo stato degli itinerari attivi, attraverso una serie di ingressi che definiscono il comportamento visivo del segnale e di uscite di stato che forniscono un riscontro sulla configurazione attiva.

Gli ingressi della I/O PORT includono i segnali logici **mS**, ciascuno dei quali mantiene il segnale in un determinato aspetto fintantoché resta asserito. Nello specifico:

- **mS_R**, mantiene la prima luce del segnale rossa;
- **mS_G**, mantiene la prima luce gialla;
- **mS_Gx**, mantiene la prima luce gialla lampeggiante;
- **mS_V**, mantiene la prima luce verde;

- **mS_R/G**, **mS_R/V**, **mS_G/V**, **mS_Gx/Vx**, gestiscono combinazioni a due luci (prima e seconda luce), con possibilità anche di lampeggio per l'unico aspetto di giallo (prima luce) e verde (seconda luce).

Il segnale **FLAG**, se asserito, indica che tutte le uscite della porta hanno raggiunto uno stato stabile e sicuro da leggere, mentre i segnali **KS** (quando asseriti a 1) rappresentano il riscontro di attivazione per ciascuno degli aspetti visualizzati, garantendo coerenza tra il comando logico e lo stato dell'attuatore.

La rete presentata è generalizzata per essere adattata a qualsiasi segnale ferroviario. Successivamente, ne vengono rappresentate due istanze specifiche: il segnale **B** e il relativo segnale di avviso **AvvB**, entrambi comandati dall'attivazione dell'itinerario 2-II. In questi casi, un decoder 3:8 viene utilizzato per selezionare l'aspetto corretto del segnale in funzione dei segnali **mS** provenienti dagli itinerari.

A differenza delle reti dei circuiti di binario o dei deviatoi, progettate qui esclusivamente per il solo itinerario 2-II, la rete dei segnali è stata completamente cablata, permettendo la selezione dell'aspetto corretto per ogni possibile itinerario. I segnali **mS** degli itinerari sono collegati direttamente agli ingressi del decoder (**A0**, **A1**, **A2**), permettendo una logica di selezione automatica dell'aspetto. Ad esempio, per l'itinerario 2-II:

- il segnale **mSd2-II** attiva l'ingresso **A0** del decoder del segnale **B** (configurazione 001), selezionando l'uscita **U1** che comanda l'aspetto rosso;
- contemporaneamente, il segnale **mSd2-II** attiva l'ingresso **A1** del decoder dell'avviso **AvvB** (configurazione 010), attivando l'uscita **U2** che comanda l'aspetto giallo lampeggiante.

Quando l'itinerario 2-II viene distrutto, il segnale **mSd2-II** si disecca assumendo valore logico 0; di conseguenza, entrambi i decoder ricevono in ingresso la configurazione 000, che abilita l'uscita **U0** e determina il ritorno dei segnali all'aspetto di default dell'impianto a riposo, ovvero rosso per il segnale di protezione **B** e giallo per il segnale di avviso **AvvB**.

Infine, è prevista anche una modalità di intervento manuale tramite la funzione di soccorso **TSB**, che l'operatore può attivare con un semplice click del mouse. Tale funzione forza entrambi i decoder nella configurazione 000, imponendo agli attuatori gli aspetti più restrittivi previsti dallo stato di riposo dell'impianto.

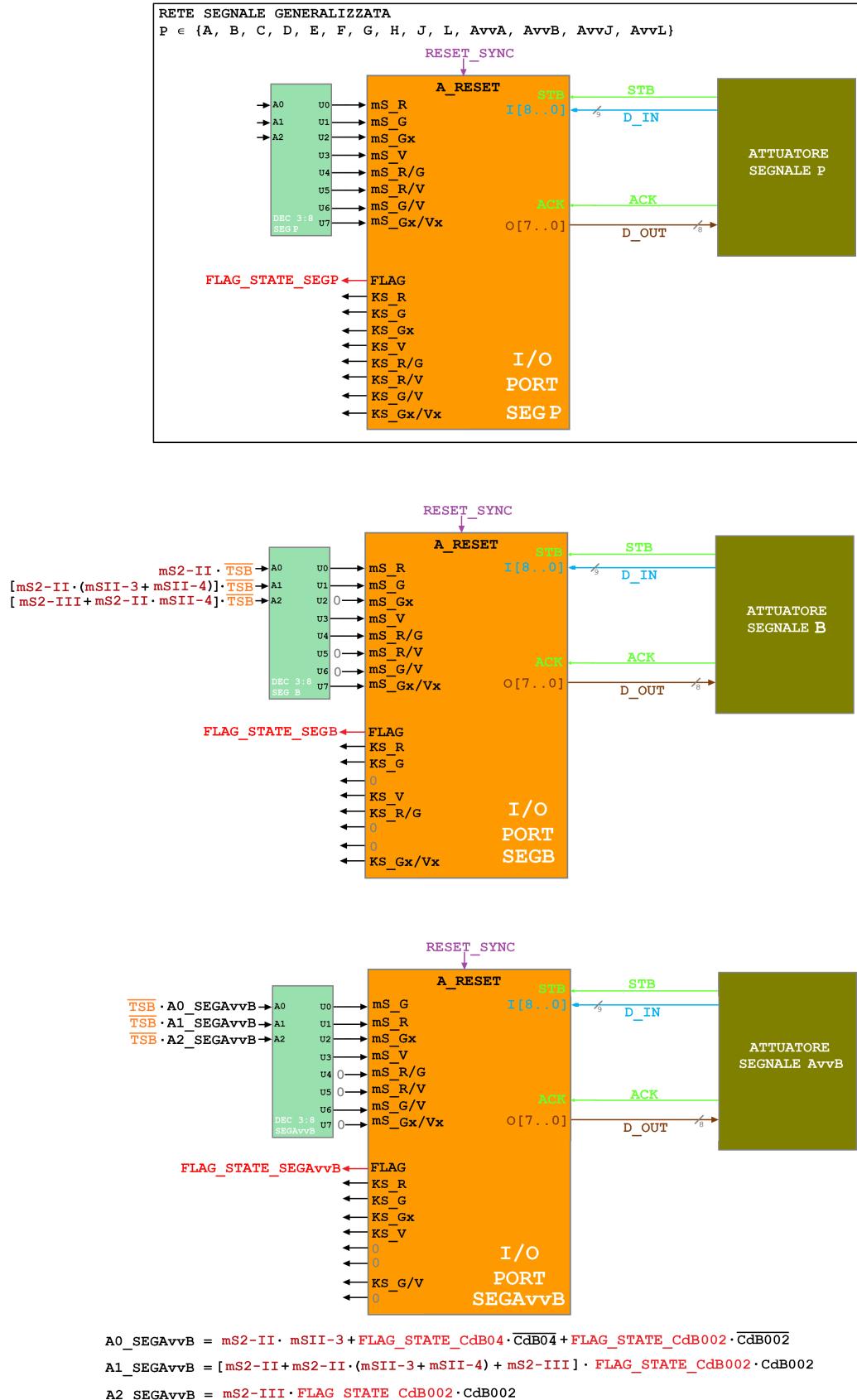


Figura 8.24: Segnale

Nella FIGURA 8.20 sono riportate due tabelle che illustrano chiaramente il cablaggio dei segnali **mS** relativi agli itinerari sugli ingressi dei decoder del segnale B e del segnale Avv.B. Ciascuna tabella mostra le diverse configurazioni degli ingressi del decoder, associandole all'aspetto corrispondente del segnale e agli itinerari che determinano tale configurazione.

SEGNALE AvvB					
A2	A1	A0	Uscita abilitata	Aspetto	
0	0	0	U0	G	DEFAULT (a riposo)
0	0	1	U1	R	$\overline{Cdb002} + \overline{Cdb04}$
0	1	0	U2	Gx	2-II + (2-II • II-4)
0	1	1	U3	V	2-II • II-3
1	0	0	U4	-	-
1	0	1	U5	-	-
1	1	0	U6	G/V	2-III
1	1	1	U7	-	-

SEGNALE B					
A2	A1	A0	Uscita abilitata	Aspetto	
0	0	0	U0	R	DEFAULT (a riposo)
0	0	1	U1	G	2-II
0	1	0	U2	-	-
0	1	1	U3	V	2-II • II-3
1	0	0	U4	R/G	2-III
1	0	1	U5	-	-
1	1	0	U6	-	-
1	1	1	U7	Gx/Vx	2-II • II-4

Figura 8.25: Tabella di codifica dei bit d'indirizzo dei decoder

8.19 Funzioni di Soccorso

Nel presente paragrafo si descrivono le 9 reti logiche relative alle funzioni di soccorso implementate nel prototipo di ACC. Ogni rete è associata a una funzione ispirata allo schema ACEI I 0/16 II Serie SDO. Le funzioni di soccorso **DJt**, **Tl**, **Tb** si attivano con 1 click del mouse, disattivandosi automaticamente mediante i segnali presenti sugli ingressi **A_RES** dei FFD. Invece, le rimanenti funzioni di soccorso si attivano sempre con un click del mouse, ma rimangono attive finché l'operatore non le disattiva con un altro click successivo.

DJt – Distruzione Itinerario

La funzione **DJtxY-Z** è impiegata per distruggere manualmente un itinerario attivo. All'avvio del sistema, il Flip-Flop D (FFD) è inizializzato a 0 tramite il segnale **RESET_SYNC**. L'operatore può attivare la funzione con un click del mouse, il cui fronte di salita viene campionato e memorizza il valore logico 1 nel FFD. Questa attivazione rimane valida finché l'itinerario xY-Z non viene completamente distrutto, condizione che si verifica quando tutte le relative reti (da R fino a mS) sono poste

a 0. Il segnale **mSxY-Z** è l'ultimo che diseccitandosi, provoca la disattivazione automatica della funzione resettando il FFD. Inoltre, la presenza del segnale negato **mSxY-Z** rende la funzione inattuabile qualora l'itinerario non sia attivo, evitando quindi interventi inutili.

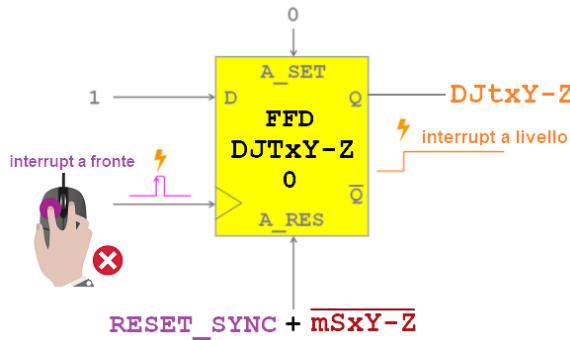
$$\begin{aligned} \mathbf{x} &\in \{\mathbf{d}, \mathbf{s}\} \\ \mathbf{Y} &\in \{1, 2, 3, 4, \mathbf{I}, \mathbf{II}, \mathbf{III}\} \\ \mathbf{Z} &\in \{1, 2, 3, 4, \mathbf{I}, \mathbf{II}, \mathbf{III}\} \end{aligned}$$


Figura 8.26: DJt

Tl – Liberazione Punto Origine

La rete **TlxY-Z** consente di liberare il punto origine di un itinerario quando, pur essendo stato distrutto, la zona di approccio è occupata, impedendo alla rete **ApxY-Z** di assumere il valore 1. Per forzare la liberazione tale segnale, si attiva la funzione di soccorso **Tl**. Il FFD è inizializzato a 0 con il segnale **RESET_SYNC** all'avvio del sistema. La funzione si attiva con un click dell'operatore, campionato sul fronte di salita, che memorizza il valore 1 nel FFD.

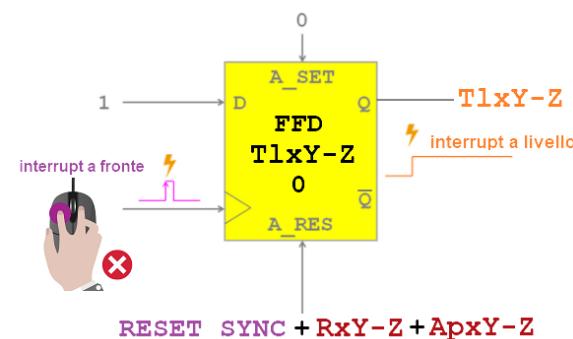
$$\begin{aligned} \mathbf{x} &\in \{\mathbf{d}, \mathbf{s}\} \\ \mathbf{Y} &\in \{1, 2, 3, 4, \mathbf{I}, \mathbf{II}, \mathbf{III}\} \\ \mathbf{Z} &\in \{1, 2, 3, 4, \mathbf{I}, \mathbf{II}, \mathbf{III}\} \end{aligned}$$


Figura 8.27: Tl

La rete si disattiva automaticamente quando **ApxY-Z** ritorna a 1, grazie al collegamento del segnale **ApxY-Z** all'ingresso **A_RES** del FFD. Inoltre, la presenza del segnale **Rxy-Z** su **A_RES** impedisce l'attivazione della funzione finché la rete R dell'itinerario non è stata prima posta a 0, ovvero finché l'itinerario non è stato distrutto.

Tb – Scarto della Fase V

Questa funzione consente di scartare manualmente la fase V nel caso in cui un circuito di binario, coinvolto nell'itinerario xY-Z, sia occupato indebitamente durante la formazione dello stesso. Il FFD è resettato a 0 all'avvio tramite **RESET_SYNC**. L'attivazione avviene tramite click del mouse campionato sul fronte di salita, che memorizza il valore 1 nel FFD. La rete si disattiva automaticamente grazie al segnale negato **Rxy-Z**, collegato ad **A_RES**, che si asserisce una volta distrutto l'itinerario. Inoltre, il segnale **Vxy-Z** impedisce l'attivazione della funzione se la fase V si è già verificata correttamente. Una particolarità della rete **TbxY-Z** è che la sua attivazione, se seguita dalla diseccitazione di **ApxY-Z**, causa automaticamente l'attivazione del segnale di chiamata (vedi rete **Tz**) senza dover attivare la funzione **Tz** manualmente. I segnali di chiamata si spengono automaticamente con la disattivazione della funzione **Tb**, senza ulteriori interventi da parte dell'operatore.

```

x ∈ {d, s}
Y ∈ {1, 2, 3, 4, I, II, III}
Z ∈ {1, 2, 3, 4, I, II, III}

```

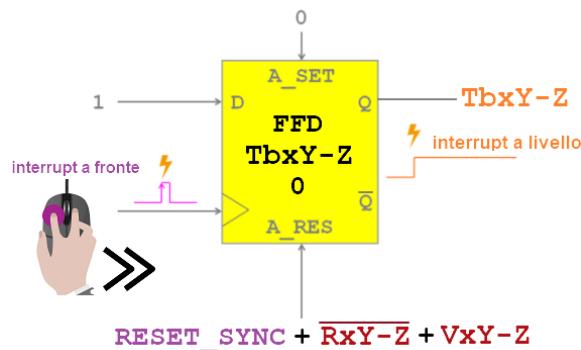


Figura 8.28: Tb

Tm – Bloccamento Manuale del Punto Origine

La rete **TmxY-Z** consente all'operatore di bloccare manualmente il punto origine di un itinerario, portando la rete **ApxY-Z** a 0, causando a sua volta la diseccitazione dei blocchatori b di tutti i deviatoi coinvolti. Questo tipo di bloccamento è definito "alla cieca" poiché viene effettuato senza il supporto delle fasi automatiche dell'apparato (come R, V, E, ecc.). Il FFD parte da 0 grazie a **RESET_SYNC** e si attiva con

un click del mouse campionato sul fronte di salita. Tuttavia, a differenza delle precedenti, la funzione non si disattiva automaticamente: una seconda pressione del mouse è necessaria affinché il valore negato di Q (posto a 0) venga retroazionato sull'ingresso D del FFD, disattivando la funzione. Inoltre, il segnale RxY-Z , cablato su A_{RES} , impedisce l'attivazione della funzione se l'itinerario corrispondente è attivo.

```

 $x \in \{d, s\}$ 
 $Y \in \{1, 2, 3, 4, I, II, III\}$ 
 $Z \in \{1, 2, 3, 4, I, II, III\}$ 

```

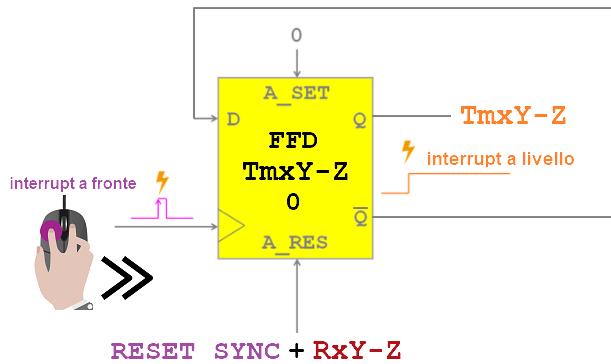


Figura 8.29: Tm

Tz – Attivazione Segnale di Chiamata

Questa rete serve ad attivare il segnale di chiamata, configurando il sistema in modalità di degrado di I° livello. Il FFD è inizializzato a 0 tramite RESET_SYNC .

```

 $x \in \{d, s\}$ 
 $Y \in \{1, 2, 3, 4, I, II, III\}$ 
 $Z \in \{1, 2, 3, 4, I, II, III\}$ 

```

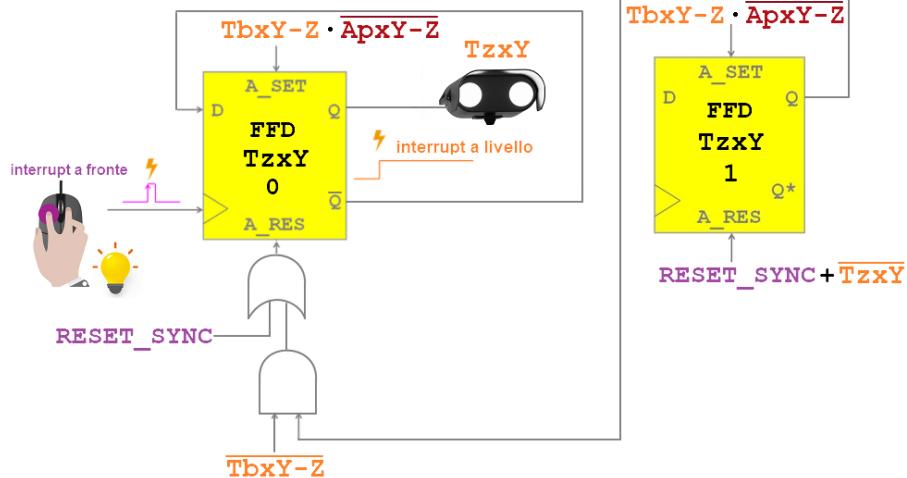


Figura 8.30: Tz

Il segnale può essere attivato manualmente con un click del mouse, campionato sul fronte di salita, che memorizza un 1 nel FFD. Una seconda pressione del mouse disattiva la funzione tramite retroazione del valore negato di Q. Tuttavia, l'attivazione può avvenire anche automaticamente: se la funzione **TbxY-Z** è attiva e **textcolorbordeauxApxY-Z** è a 0, si accendono automaticamente entrambi i FFD coinvolti nella rete **Tz**, accendendo così i segnali di chiamata. Quando la funzione **Tb** viene disattivata, il secondo FFD ausiliario provoca la disattivazione automatica del segnale di chiamata e si resetta anch'esso.

TbD – Ignora CdB di Immobilizzazione Deviatoio

La funzione **TbD** consente di ignorare il controllo del circuito di binario di immobilizzazione di un deviatoio generico Q, utile in caso di guasto o occupazione indebita del CdB stesso. Il FFD parte da 0 grazie al **RESET_SYNC** e si attiva con un click del mouse campionato sul fronte di salita. La funzione rimane attiva finché non viene eseguita un'altra pressione del mouse che, tramite retroazione del valore Q negato a 0, spegne la funzione. L'attivazione è impedita se il CdB è eccitato correttamente: infatti, il segnale **CdBW** con annesso flag di stabilità blocca la funzione in condizioni normali.

```

Q ∈ {1, 2, 3, 4}
W ∈ {0001, 0002, 0003, 0004, 001, 002, 003, 004, 01, 02, 03,
04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, I, II, III}

```

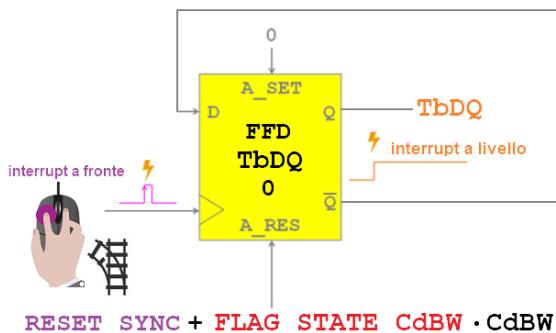


Figura 8.31: **TbD**

TcD – Ignora Controllo di posizione iniziale Deviatoio

La funzione **TcD** è progettata per ignorare il controllo di posizione iniziale necessario per movimentare un deviatoio generico Q. Come le precedenti, il FFD è inizializzato a 0 tramite **RESET_SYNC** e si attiva con un click del mouse campionato sul fronte di salita. Per disattivare la funzione, è necessaria una seconda pressione del mouse che retroaziona il valore negato di Q a 0. Tuttavia, la funzione è inibita se il deviatoio dispone del controllo di posizione: infatti, i segnali **KDN_DEVQ** e **KDR_DEVQ**,

con relativo **FLAG** di stabilità, ne impediscono l'attivazione in condizioni operative regolari.

$$Q \in \{1, 2, 3, 4\}$$

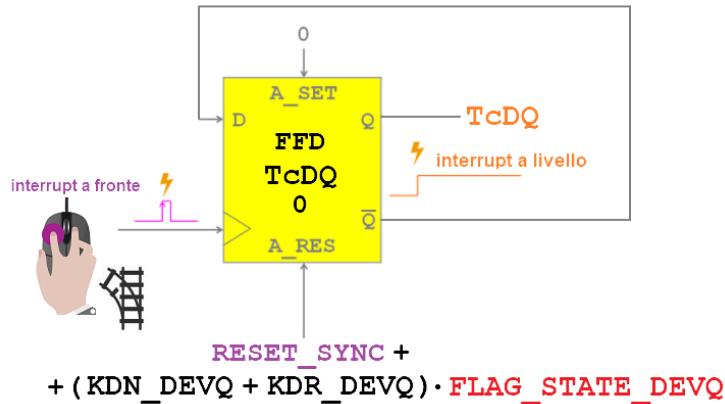


Figura 8.32: TcD

TS – Chiusura Segnale Ferroviario

La rete **TS** permette all'operatore di forzare manualmente la chiusura di un segnale ferroviario generico P. Il FFD è inizializzato a 0 tramite **RESET_SYNC** e si attiva con un click del mouse campionato sul fronte di salita. L'uscita rimane attiva finché non viene effettuata un'ulteriore pressione del mouse, che provoca la retroazione del valore **Q** negato (0) e disattiva la funzione. Non vi sono condizioni di blocco esterne, quindi la funzione è sempre attuabile.

$$P \in \{A, B, C, D, E, F, G, H, J, L, \text{AvvA}, \text{AvvB}, \text{AvvJ}, \text{AvvL}\}$$

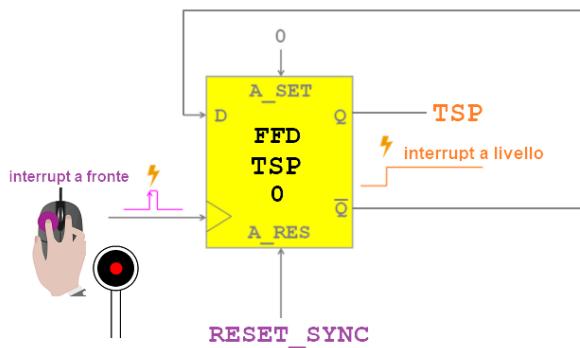


Figura 8.33: TSP

Tsm – Rimozione Codifica sui Binari di Corsa

Questa funzione consente la rimozione manuale della codifica sui binari di corsa tra i punti Y e Z, forzando l'alimentazione fissa dei relativi CdB. Il FFD è resettato a 0 all'avvio grazie al segnale **RESET_SYNC**. La funzione si attiva con un click del mouse

campionato sul fronte di salita, e rimane attiva finché l'operatore non la disattiva con un secondo click, il quale, tramite la retroazione del valore Q negato a 0, spegne la funzione.

```
Y ∈ {1, 2, 3, 4, I, II, III}
Z ∈ {1, 2, 3, 4, I, II, III}
W ∈ {0001, 0002, 0003, 0004, 001, 002, 003, 004, 01, 02, 03,
      04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, I, II, III}
```

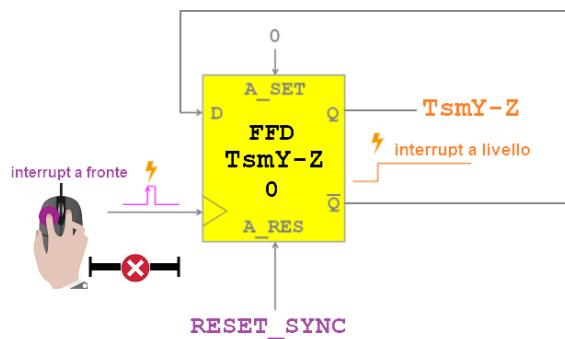


Figura 8.34: Tsm

8.20 Codice sorgente

Prima di analizzare il codice sorgente, è necessario spiegare come è stata programmata, all'interno del codice DLX, la tabella delle incompatibilità degli itinerari ACEI, rappresentata in FIGURA 8.35.

Figura 8.35: Tabella delle Incompatibilità ACEI

Osservando la tabella, si nota che ciascuna riga j ha lo stesso contenuto della corrispondente colonna j , con l'unica eccezione delle incompatibilità relative al libero transito, indicate con la lettera T (le incompatibilità sono espresse per colonna).

Al fine di ottenere tutte le incompatibilità di un determinato itinerario espresse per riga (aspetto chiave per la successiva gestione nel codice), le incompatibilità di tipo T sono state ridistribuite nelle posizioni simmetriche della tabella (vedi matrice di destra nella FIGURA 8.35).

Completata questa riorganizzazione, si è proseguito inserendo 0 in tutte le caselle vuote della tabella, incluse quelle sulla diagonale, poiché un itinerario non è incompatibile con sé stesso. Questo comportamento è comunque gestito nella rete logica C, che impedisce la ri-creazione di un itinerario già attivo quando il corrispondente segnale **RdxY-Z** (cablato sull'ingresso **A_RES** del FFD) assume lo stato logico 1. Le celle occupate sono state invece riempite con 1, ottenendo così una rappresentazione binaria completa per ciascun itinerario (FIGURA 8.36).

ITINERARI INCOMPATIBILI																			R E G I S T R I D L X	
1-I	1-II	1-III	I-1	II-1	III-1	2-II	2-III	II-2	III-2	3-II	3-III	II-3	III-3	4-I	4-II	4-III	I-4	II-4	III-4	
1-I	0	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	1	0	0	R1
1-II	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	R2
1-III	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	R3
I-1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	R4
II-1	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	R5
III-1	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	R6
2-I	0	1	1	0	1	1	0	1	1	1	1	1	1	0	1	1	0	1	1	R7
2-II	0	1	1	0	1	1	1	0	1	1	1	1	1	0	1	1	0	1	1	R8
2-III	0	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0	R9
II-2	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	R10
III-2	0	1	1	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	0	R11
3-II	0	1	1	0	1	0	1	1	1	0	0	1	1	0	1	1	0	1	1	R12
3-III	0	1	1	0	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	R13
II-3	0	0	1	0	0	0	0	1	0	0	1	1	0	1	0	1	1	0	1	R14
III-3	0	1	1	0	0	0	1	1	0	1	1	0	0	1	1	0	1	1	1	R15
4-I	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	R16
4-II	0	1	1	0	1	0	1	1	1	0	1	1	1	1	0	1	1	1	1	R17
4-III	0	1	1	0	0	1	1	0	1	1	1	1	1	1	1	0	1	1	1	R18
I-4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	R19
II-4	0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	1	R20
III-4	0	1	1	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	0	R20

(ITINERARI INCOMPATIBILI) ₂	(IT,INC) ₁₆	(16MSB) ₂	(16MSB) ₁₆	(Imm _{16bit}) ₂	(Imm _{16bit}) ₁₆	R
1-I	00000000000000000011111000000000100100	0007C024	0000000000000000111	0007	110000000000100100	C024 R1
1-II	000000000000001011111111111011011	000BFFDB	000000000000001011	000B	1111111111011011	FFDB R2
1-III	000000000000001101111111111011011	000DFFDB	000000000000001101	000D	11111111111011011	FFDB R3
I-1	0000000000000011101100000000000000	000EC000	000000000000001110	000E	110000000000000000	C000 R4
II-1	0000000000000011110111110000000000	000F7C00	000000000000001111	000F	011110000000000000	7C00 R5
III-1	0000000000000011111011111000000000	000FB0D8	000000000000001111	000F	10111101000001000	B0D8 R6
2-II	000000000000001101101111111011011	0006DFDB	000000000000001110	0006	1101111111011011	DFDB R7
2-III	000000000000001101111111111011011	0006EFDB	000000000000001110	0006	1110111111011011	EFDB R8
II-2	0000000000000011011111010000000000	0006F400	000000000000001110	0006	111101000000000000	F400 R9
III-2	0000000000000011011111100100000000	0006F908	000000000000001110	0006	11111001000001000	F908 R10
3-II	00000000000000110101110111011011	0006B9DB	000000000000001110	0006	1011100111011011	B9DB R11
3-III	00000000000000110101110111011011	000676DB	000000000000001110	0006	01110110111011011	76DB R12
II-3	000000000000001100001001101011011	0002135B	000000000000001110	0006	0001001101011011	135B R13
III-3	00000000000000110000110011110011011	0006339B	000000000000001110	0006	0011001110011011	339B R14
4-I	0000000000000010010000000000000000	0009001F	000000000000001001	0009	00000000000011111	001F R15
4-II	00000000000000101011101111101111	0006BBEF	000000000000001110	0006	1011101111101111	BBEF R16
4-III	00000000000000110011101111110111	000677F7	000000000000001110	0006	0111011111101111	77F7 R17
I-4	0000000000000000000000000000000000	00000003B	000000000000000000	0000	000000000011011	003B R18
II-4	0000000000000000000000000000000000	000213FD	000000000000000000	0002	0011001111111101	13FD R19
III-4	0000000000000000000000000000000000	000633FE	000000000000000000	0006	0011001111111110	33FE R20

Figura 8.36: Tabella delle Incompatibilità ACC

Questa codifica consente, nel codice, di eseguire un'operazione **AND** tra il registro contenente gli itinerari attivi (**R30**) e un registro che rappresenta le incompatibilità di un determinato itinerario. L'operazione fornisce un risultato nullo (32 zeri) solo se tutti gli itinerari attivi sono compatibili con quello che si desidera attivare, condizione necessaria affinché la creazione possa proseguire. È fondamentale, a tal fine, che ogni itinerario occupi la stessa posizione bit a bit sia nel registro de-

gli attivi che in quello delle incompatibilità. Ad esempio, l'itinerario 2-II occupa la quattordicesima posizione da destra in entrambe le rappresentazioni.

Dopo aver codificato ciascuna riga della tabella in formato binario, si sono aggiunti 12 bit a sinistra per ottenere una lunghezza di 32 bit compatibile con l'architettura del DLX. La stringa binaria risultante è stata poi convertita in esadecimale (terza colonna FIGURA 8.36) e suddivisa nei suoi 16 bit più e meno significativi, in modo da agevolare il caricamento nel DLX tramite le istruzioni **LHI** e **ADDUI**, che lavorano con immediati a 16 bit. Infine, nell'ultima colonna della tabella è riportato in quale registro del DLX viene memorizzata ogni riga: ad esempio, la riga che rappresenta le incompatibilità dell'itinerario 2-II viene salvata nel registro **R7**.

Il codice è scritto in linguaggio assembly e rappresenta la sequenza di istruzioni che il processore DLX eseguirà. La prima istruzione imposta il valore esadecimale **0x8000000** nel registro **R21**, a cui segue una **LBU**, ovvero un ciclo di bus di lettura, che asserisce il comando **CS_READ_STARTUP**: il byte letto è contenuto su **BD[7...0]** e in particolare il bit **BD0** rappresenta il segnale **STARTUP**. Questo byte è esteso a 32 bit e salvato nel registro **R22**. La terza istruzione confronta il valore in **R22** con zero: se è zero, significa che **STARTUP** è basso, quindi il sistema non si è appena acceso ma è avvenuto un interrupt, e in tal caso si salta direttamente all'**handler**. Se invece **STARTUP** è alto, l'esecuzione continua alla riga 14h, indicando che siamo nella fase di accensione del sistema. Da qui fino alla riga 88h si memorizza la tabella delle incompatibilità (FIGURA 8.30) nei registri da **R1** a **R20**, ciascuno rappresentante una riga della tabella, utilizzando istruzioni **LHI** e **ADDUI** con il registro **R23** come supporto per comporre i 32 bit da salvare. Dato che gli itinerari sono 20, i primi 20 bit di ogni registro contengono informazioni utili, mentre i restanti 12 bit sono impostati a zero; ad esempio, **R7** memorizza le incompatibilità dell'itinerario 2-II. Alla riga 90h si esegue un ciclo di scrittura con l'istruzione **SB** per resettare il segnale **STARTUP** a 0, scrivendo il valore 0 contenuto in **R0** sul **BD**, che in realtà non è una scrittura vera e propria ma serve solo ad attivare il comando **CS_WRITE_STARTUP_0**. Successivamente, con l'istruzione di jump alla riga 98h, si passa al **main** dove il programmatore può eseguire routine di diagnostica e sicurezza non dettagliate nel codice. Il DLX rimane nel **main** finché non riceve un interrupt sull'ingresso **INT**, che indica un comando di itinerario da parte dell'operatore (vedi rete C). In questo caso, l'esecuzione riparte dall'istruzione 0 e, trovando **STARTUP** a 0, salta all'**handler**. In esso si legge una parola a 32 bit dallo stato dei 20 FFD della rete C tramite una **LW** che attiva **CS_READ_C**, salvando il risultato in **R24**: i primi 20 bit rappresentano lo stato dei comandi, con un bit a 1 se l'itinerario corrispondente è stato comandato. Dalla riga A0h si costruisce una maschera con solo il 20° bit a 1 (con **LHI**) salvata in **R25**, si esegue un **AND** tra **R24** e **R25** salvando il risultato in **R26**: se **R26** è 0, si passa a controllare l'itinerario successivo; se è diverso da 0, si esegue l'handler specifico **R_Cd1-I**. Seguendo questo schema, si individua l'itinerario con

priorità più alta tra quelli comandati e si esegue il suo handler specifico; l'ordine di priorità è arbitrario e configurabile, ma nel caso in esame si è assegnata priorità massima all'itinerario in 20^a posizione (1-I) e minima a quello in 1^a posizione (III-4). Quando viene rilevato il comando per l'itinerario 2-II (bit 14), si esegue l'handler **R_Cd2II** che valuta la possibilità di creare l'itinerario leggendo lo stato degli itinerari attivi tramite una **LW** che asserisce **CS_READ_R**, salvando il valore in **R30**. Un **AND** tra **R30** e **R7** (che contiene le incompatibilità del 2-II) viene salvato in **R27**: se **R27** ha almeno un bit a 1, significa che c'è incompatibilità con un itinerario già attivo e il DLX non crea l'itinerario, ma spegne il comando dell'operatore con una **SB** che attiva **CS_DESTROY_Cd2-II** scrivendo 0 da **R0** (anche qui si tratta di una falsa scrittura mirata esclusivamente ad azionare il relativo **CS**). Poi, una **LW** su **CS_READ_C** aggiorna **R24** per riflettere eventuali nuovi comandi ricevuti nel frattempo prima di tornare all'**handler**. Se invece **R27** è 0, cioè 2-II è compatibile con gli attivi (se presenti), si prosegue con **RUN_Cd2II**. In esso si comandano i deviatoi coinvolti: **R28** è inizializzato con solo **BD0** a 1, e una **SB** attiva **CS_M_DEV1** comandando la manovra normale del deviatoio 1, seguito da una seconda **SB** alla riga 100h che asserisce **CS_M_DEV2** scrivendo **BD8** a 1, comandando la manovra normale del deviatoio 2. Anche se i deviatoi fossero già normali, le **SB** sarebbero ininfluenti grazie alla rete logica che blocca manovre non necessarie, evitando così cicli di lettura-preventiva che sarebbero più onerosi in termini di tempo. Dopo aver comandato i deviatoi, si registra l'itinerario con una **SB** che attiva **CS_REGISTRATION_Rd2-II** scrivendo 0 da **R0** (falsa scrittura) per impostare a 1 il FFD textcolorbordeaux**Rd2-II**. Questa registrazione avviene solo dopo le manovre, perché attivare prima il bit textcolorbordeaux**Rd2-II** causerebbe il blocco dei deviatoi tramite la rete dei **bD**. Infine, si aggiorna **R24** con una **LW** per rilevare nuovi interrupt tornando all'**handler** con un jump. Se non sono presenti altri itinerari attivi, l'istruzione **RFE** conclude l'esecuzione dell'**handler** e ritorna (nel **main**) all'istruzione successiva a quella che ha attivato l'ultima gestione terminata.

Riassumendo:

- a) nella sezione **handler** si individua l'itinerario da attuare, secondo una politica di priorità prestabilita;
- b) ogni itinerario prevede due sezioni dedicate (nel codice sono state implementate quelle relative all'itinerario 2-II, ma le sezioni di qualsiasi altro itinerario sono facilmente deducibili per analogia). La prima sezione verifica se l'itinerario è attuabile, considerando l'eventuale presenza di altri itinerari attivi, mentre la seconda si occupa del comando dei deviatoi coinvolti nell'itinerario.

Si precisa infine che, nel codice, le stringhe che iniziano con il prefisso «**0x**» o terminano con il suffisso «**h**» rappresentano numeri in formato esadecimale. Le due notazioni sono perfettamente equivalenti.

Il codice riportato nella presente pagina, è disponibile anche all'interno del repository GitHub del progetto, accessibile al link: <https://github.com/nicolovaldiserri/railway-ACC-with-DLX.git>.

```

0h init:    LHI    R21,0x8000      ; set R21 = 0x8000000h (command address)
4h      LBU    R22,0x0000(R21)   ; CS_READ_STARTUP (read STARTUP signal into R22)
8h      BEQZ   R22,handler     ; if (STARTUP == 0) then jump to handler
Ch
10h
14h      LHI    R23,0x0007      ; begin preamble: perform all startup initialization tasks here
18h      ADDUI  R1,R23,0xC024    ; storage of the incompatibility table
1Ch      LHI    R23,0x000B      ; set R23 = 0x000B000h
20h      ADDUI  R2,R23,0xFFDB    ; set R2 = 0x000BFDBh
24h      LHI    R23,0x000D      ; set R23 = 0x0000000h
28h      ADDUI  R3,R23,0xFFDB    ; set R3 = 0x000DFDBh
2Ch      LHI    R23,0x000E      ; set R23 = 0x0000000h
30h      ADDUI  R4,R23,0xC000    ; set R4 = 0x000FC00h
34h      LHI    R23,0x000F      ; set R23 = 0x000F000h
38h      ADDUI  R5,R23,0x7C00    ; set R5 = 0x000F7C0h
3Ch      ADDUI  R6,R23,0xB088    ; set R6 = 0x000FB08h
40h      LHI    R23,0x0006      ; set R23 = 0x0006000h
44h      ADDUI  R7,R23,0xDFDB    ; set R7 = 0x0006DFDBh
48h      ADDUI  R8,R23,0xEFDB    ; set R8 = 0x0006EFDBh
4Ch      ADDUI  R9,R23,0xF400    ; set R9 = 0x0006F40h
50h      ADDUI  R10,R23,0xF908   ; set R10 = 0x0006F908h
54h      ADDUI  R11,R23,0xB9DB    ; set R11 = 0x0006B9DBh
58h      ADDUI  R12,R23,0x76DB    ; set R12 = 0x000676DBh
5Ch      ADDUI  R13,R23,0x135B    ; set R13 = 0x0006135Bh
60h      ADDUI  R14,R23,0x339B    ; set R14 = 0x0006339Bh
64h      LHI    R23,0x0009      ; set R21 = 0x0009000h
68h      ADDUI  R15,R23,0x001F    ; set R15 = 0x0009001Fh
6Ch      LHI    R23,0x0006      ; set R21 = 0x0006000h
70h      ADDUI  R16,R23,0xB8EF    ; set R16 = 0x0006B8EFh
74h      ADDUI  R17,R23,0x77F7    ; set R17 = 0x000677F7h
78h      ADDUI  R18,R0,0x003B    ; set R18 = 0x0000003Bh
7Ch      LHI    R23,0x0002      ; set R23 = 0x0002000h
80h      ADDUI  R19,R23,0x13FD    ; set R19 = 0x000213FDh
84h      LHI    R23,0x0006      ; set R23 = 0x0006000h
88h      ADDUI  R20,R23,0x33FE    ; set R20 = 0x000633FE
8Ch
90h      SB     R0,0x0001(R21)   ; end of storage
94h
98h      J     main           ; end of preamble
9Ch
A0h handler:  LW     R24,0x0004(R21)   ; CS_READ_C (read interrupt into R24)
A4h      LHI    R25,0x0008      ; set R25 = 0x0008000h (only the 20th bit is equal to 1)
A8h      AND    R26,R24,R25
ACh      BNEZ   R26,R_CdII
B0h      LHI    R25,0x0004      ; if BD19 is equal to 1, the Cd1-I route is managed
B4h      AND    R26,R24,R25
B8h      BNEZ   R26,R_CdII
BCh
C0h
C4h      ANDI   R26,R24,0x2000    ; turn off all bits except the 14th one which represents Cd2-II_SYNC
C8h      BNEZ   R26,R_CdII
CCh
D0h
D4h      RFE
D8h
DCh R_Cd2II: LW     R30, 0x0008(R21)   ; CS_READ_R (read the active itineraries into R30)
E0h      AND    R27,R7,R30
E4h      BEQZ   R27,DEV_Cd2II
E8h      SB     R0,0x0002(R21)   ; CS_DESTROY_Cd2-II (reset Cd2-II)
ECh      LW     R24,0x0004(R21)   ; CS_READ_C (read interrupt into R24)
F0h      J     handler        ; jump to handler
F4h
F8h RUN_Cd2II: ADDUI R28,R0,0x0001
FCh      SB     R28,0x000C(R21)   ; CS_M_DEV1 (with normal operation)
100h      SB     R28,0x000D(R21)   ; CS_M_DEV2 (with normal operation)
104h      SB     R0,0x0003(R21)   ; CS_REGISTRATION_Rd2-II
108h
10Ch      LW     R24,0x0004(R21)   ; CS_READ_C (read interrupt into R24)
110h      J     handler        ; jump to handler
114h
118h main:          ; DLX routine activities

```

8.21 Spunti di adattamento del progetto ad un core RISC-V

Il core DLX utilizzato nel presente progetto appartiene alla famiglia dei processori RISC (Reduced Instruction Set Computer), e si ispira ad un'architettura didattica pensata per rendere chiari i concetti fondamentali di una CPU pipeline con set di istruzioni semplificato. Tuttavia, nel contesto dell'evoluzione tecnologica, uno standard aperto e sempre più diffuso è rappresentato dal RISC-V. Esso è una moderna architettura RISC open source, sviluppata con l'obiettivo di offrire un ISA libero da licenze, modulare e flessibile, già ampiamente utilizzato in ambito accademico e industriale.

A livello concettuale, il DLX e il RISC-V condividono i principi fondamentali delle architetture RISC: istruzioni di lunghezza fissa (32 bit nella base), load/store architecture, utilizzo di registri general purpose, pipeline e set di istruzioni semplici. Tuttavia, si distinguono in vari aspetti: il RISC-V ha un set di istruzioni più esteso e modulare (diviso in base, estensioni standard e personalizzabili), usa convenzioni e codifiche leggermente diverse, ha registri numerati da x0 a x31 (dove x0 è hard-wired a 0, come R0 nel DLX), e impone una sintassi assembler differente. Inoltre, nel RISC-V le istruzioni di accesso alla memoria operano solo su offset firmati e non esistono istruzioni come **LHI** (load high immediate), il che richiede strategie diverse per la costruzione di costanti a 32 bit (ad esempio, con LUI seguito da ADDI).

Per adattare il progetto corrente, basato su DLX, a un core RISC-V, sarebbe dunque necessario:

- convertire il codice assembly riscrivendo le istruzioni secondo la sintassi e le convenzioni RISC-V. Ad esempio LW x10,0(x11) invece di **LW R10,0x0000(R11)**;
- rivedere la logica di inizializzazione delle costanti a 32 bit (sostituendo **LHI** e **ADDUI** con LUI e ADDI);
- riconfigurare la mappa di memoria e i segnali di controllo (**CS**, **MEMRD**, **MEMWR**) per adeguarli alla gestione del bus nel nuovo core;
- adattare eventuali differenze nei meccanismi di interrupt e gestione del flusso (J, RFE) secondo il modello di interrupt a privilegi del RISC-V (registri CSR accessibili mediante istruzioni dedicate come CSRRW, CSRRS, CSRRC);
- infine, riprogettare i segnali specifici con interfacce standard come AXI o Wishbone, se si impiega un core RISC-V reale anziché il modello didattico del DLX.

In sintesi, l'adattamento sarebbe perfettamente fattibile e rappresenterebbe un naturale passo evolutivo del progetto, permettendo anche la possibilità di implementazione su piattaforme hardware RISC-V esistenti (es. FPGA con core PicoRV32 o RocketChip), offrendo maggiore compatibilità e scalabilità per sviluppi futuri.