UNIVERSITÀ DEGLI STUDI DI GENOVA

Dipartimento di Ingegneria Navale, Elettrica, Elettronica e delle Telecomunicazioni (DITEN)

Corso di **LAUREA MAGISTRALE**
in **Ingegneria Elettronica**

Progettazione e sviluppo su dispositivi FPGA, di un circuito digitale che implementa le funzionalità Master del protocollo di comunicazione seriale I3C e l'interfaccia verso un microprocessore "RISC-V"

RELATORE:

Prof. Maurizio Valle

CANDIDATO:

Ancarani Francesco

CORRELATORI:

Ing.Daniele Grosso

Ing. Luca Noli

Anno Accademico 2023/2024

Abstract

This thesis focuses on the design, simulation, Register Transfer Level verificationand FPGA implementation of a digital circuit module. The main goal is to realize a programmable I3C Master circuit that acts as the interface between the TARDIS, a customized RI5CY microprocessor version having a RISC-V architecture, and a I3C bus.The circuit manages data communications from-and-to the RI5CY microprocessor, providing the typical protocol signals, and to implementing the I3C basic functions. The proposed methodology makes use of a top-down approach for the module design phase and uses SystemVerilog constructs for both design and RTL verification phases. Module verification phase uses testbenches that emulates microprocessor and peripherals behavior, in order to provide the Master_I3C significant input stimuli and verify the output correctness. Subsequent FPGA implementation allow us to have a circuit performance estimation in terms of occupied area, power consumption and maximum clock speed.

Indice

Capitolo 1

Introduzione

1.1 Obiettivi

L'obiettivo dell'attività di tesi è la progettazione di un circuito digitale con funzionalità di Master_I3C che permetta a un microprocessore di tipo RI5CY di comunicare sul bus seriale attraverso il protocollo I3C.

1.2 Motivazioni

Si è scelto di realizzare il circuito digitale Master_I3C in collaborazione con l'azienda INSYDE s.r.l.s. L'azienda disponeva di una versione per il protocollo I^2C, il Master_I^2C e c'era la necessità di aggiornare il circuito per adempire alle funzioni del protocollo seriale più recente, ovvero l'I3C. L'azienda possiede una versione personalizzata del microprocessore RI5CY, il TARDIS.

1.3 Metodologia utilizzata

L'attività di tesi si divide in due fasi fondamentali: la fase di progettazione e la fase di verifica delle funzioni implementate. Per la fase di progettazione si è scelto di lavorare sul precedente progetto del Master_I^2C, dopo averne compreso la struttura è stato possibile modificare il progetto in modo che si adattasse al protocollo I3C. Il modulo Master_I3C è stato progettato con un approccio top-down, definendo prima i moduli ad alto livello.

Il linguaggio di descrizione hardware utilizzato per la scrittura dell'RTL è il SystemVerilog (*IEEE 1800 standard*), un'estensione del Verilog (*IEEE 1346 standard*) nata nel 2005. Introduce i costrutti della programmazione orientata agli oggetti e per questo risulta ideale sia per la fase di progettazione dell'RTL ma anche per la fase di verifica.

La fase di verifica consiste nel validare le funzionalità implementate nella fase di progettazione. Il testbench è stato concepito in modo che emulasse il comportamento del microprocessore e delle periferiche collegate al bus. Ogni funzionalità è stata verificata attraverso la generazione

di sequenze di input, fornite al Master_I3C sia dal lato del microprocessore che dal lato delle periferiche. La verifica di una funzionalità si conclude con il confronto tra la risposta ricevuta dal Master_I3C e il valore atteso.

Dopo le fasi di progettazione e di verifica, l'RTL è stato sintetizzato su una FPGA (*Field Programmable Gate Array*). Sono stati ricavate le stime sulle prestazioni del circuito in termini di timing, consumi in potenza, risorse hardware necessarie alla realizzazione.

1.4 Organizzazione del documento di tesi

Questo documento è suddiviso in 7 capitolo, organizzati come segue:

- Il Capitolo 1 necessario per la presentazione del lavoro.
- Il Capitolo 2 ha il compito di descrivere il protocollo I3C da un punto di vista teorico. Questo capitolo fa riferimento allo standard fornito nella documentazione ufficiale rilasciata da *MIPI Alliance*.
- Il Capitolo 3 è dedicato alla descrizione del microprocessore utilizzato, per capire l'interfaccia disponibile per la comunicazione con il Master_I3C è stato necessario descrivere il microprocessore RI5CY.
- Il Capitolo 4 è dedicato alla descrizione approfondita dei moduli che sono stati progettati a livello RTL per la realizzazione del Master_I3C.
- Il Capitolo 5 è dedicato alla descrizione della metodologia utilizzata per la fase di verifica e ai risultati ottenuti nei test effettuati.
- Il Capitolo 6 è dedicato alla realizzazione su dispositivo FPGA e all'analisi dei risultati ottenuti.
- Il Capitolo 7 è dedicato alle conclusioni finali e alle proposte per eventuali sviluppi futuri.

Capitolo 2

Il Protocollo I3C

Questo capitolo è dedicato alla descrizione del protocollo I3C che è stato implementato dal circuito Master_I3C. Verranno affrontati gli aspetti del protocollo I3C ritenuti più importanti.

2.1 Introduzione al Protocollo I3C

La comunicazione tra due o più dispositivi elettronici avviene attraverso un insieme di linee chiamate “bus”. L’insieme delle regole che garantiscono una corretta comunicazione tra i dispositivi prende il nome di protocollo.

Il protocollo I3C *(Improved Inter Integrated Circuit)* è il nuovo standard per la comunicazione seriale tra più dispositivi elettronici, anche conosciuto come MIPI I3C, fu annunciato nel novembre 2014 dalla compagnia *Mobile Industry Processor Interface Alliance (MIPI Alliance)* e rilasciato al pubblico nel dicembre 2017. È stato sviluppato principalmente per rispondere alle esigenze, sempre più crescenti, di connettività e prestazioni nel settore dei dispositivi embedded. Si pensi al settore dell’automotive o a tutte le soluzioni per l’IoT (Internet of Things), dove un processore per comunicare con molti sensori spesso deve ricorrere a diversi tipi di interfaccia (ad esempio I^2C o SPI) aumentando il numero di pin richiesti e lo spazio sul PCB. l'I3C si presenta come una soluzione efficiente, consentendo una connessione di tutti i dispositivi su un unico bus. Inoltre, l'I3C offre una totale compatibilità con i dispositivi già esistenti che utilizzano il protocollo I2C, garantendo una transizione fluida verso le nuove tecnologie. Grazie ai suoi comandi standard e alla gestione dinamica degli indirizzi, l'I3C permette di velocizzare e ottimizzare la comunicazione, migliorando complessivamente le prestazioni dei sistemi embedded.

2.2 Terminologia utilizzata e aspetti fondamentali

ACK: abbreviazione per “acknoledge”.

Arbitraggio degli Indirizzi: processo per determinare la precedenza di trasmissione degli indirizzi sulla linea SDA durante la fase di contesa.

Broadcast: comando indirizzato a più dispositivi Slave che utilizza l’indirizzo Broadcast 7’h7E.

Bus: implementazione delle linee SDA e SCL.

BCR (Bus Characteristic Register): registro proprio di ogni dispositivo contenente specifiche funzionali.

CCC (Common Command Code): insieme di comandi standard per i dispositivi I3C.

Frame: Inizia con START, segue l’indirizzo target, i byte di dati e si conclude con STOP.

HDR (High Data Rate): modalità che incrementa la velocità di trasmissione dei dati.

Hot – Join: meccanismo che permette agli Slave, che si uniscono al bus dopo la configurazione, di ottenere un indirizzo dinamico dal Master.

IBI (In – Band Interrupt): metodo che permette ai dispositivi Slave di inviare il proprio indirizzo al Master per notificare un’interruzione.

Indirizzo: set di bit per identificare un dispositivo o la posizione di un registro.

Indirizzo dinamico: l’indirizzo di un dispositivo viene assegnato dinamicamente durante l’inizializzazione del bus.

Legacy I^2C: completa retrocompatibilità con dispositivi I^2C.

Master: dispositivo delegato al controllo del bus, l’unico che pilota la linea SCL.

NACK: abbreviazione di “not acknoledge”.

Open Drain: configurazione di uscita dei pin, con solo possibilità di Pull – Down.

Pull - Down: attivazione della linea mettendo a livello logico “0” il bus.

Pull - Up: attivazione della linea mettendo a livello logico “1” il bus.

Push - Pull: configurazione di uscita dei pin, con possibilità di Pull - Up e Pull - Down.

SDR (Single Data Rate): modalità standard di comunicazione I3C.

START: definisce l'inizio della comunicazione con transizione della linea SDA da 1 a 0, mentre la linea SCL rimane a livello logico 1.

STOP: definisce la fine della comunicazione con transizione della linea SDA da 0 a 1, mentre la linea SCL rimane a livello logico 1.

T-Bit: Transition Bit, alternativa introdotta con I3C a ACK/NACK.

2.2.1 Aspetti importanti

Il protocollo I3C è una versione avanzata del protocollo I^2C che, in aggiunta ai messaggi privati, ha la possibilità di inviare messaggi standard ai soli dispositivi I3C. Questa tipologia di messaggi si divide in messaggi Broadcast, inviati a tutti i dispositivi I3C presenti sul bus e messaggi Direct, indirizzati a uno specifico dispositivo.

Utilizzando il protocollo I3C è possibile garantire una totale retrocompatibilità con i dispositivi che supportano solo I^2C, questo genere di comunicazione prende il nome di Legacy I^2C.

Tutte le modalità di comunicazione presenti sul bus avvengono mediate due linee:

- SDA: acronimo di "Serial Data", è la linea bidirezionale attraverso la quale i dispositivi scambiano i dati.
- SCL: acronimo di "Serial Clock", è la linea monodirezionale, attraverso la quale il dispositivo "principale" invia il segnale di sincronizzazione.

Una delle novità più importanti introdotte è la possibilità di cambiare la velocità del segnale di sincronizzazione trasmesso sulla linea SCL. Durante una trasmissione in modalità Legacy I^2C la velocità non può superare i 3.4 [Mbit/s], in accordo con la velocità massima supportata dal protocollo I2C. In modalità Single Data Rate (SDR) è possibile incrementare la velocità fino a 12.5 [Mbit/s] grazie alla configurazione di uscita dei pin di tipo Push–Pull.

Esiste una modalità più veloce, denominata High Data Rate (HDR), che permette di incrementare ulteriormente la velocità di trasmissione, superando i 30 [Mbit/s]. È importante precisare che il bus I3C viene sempre inizializzato in modalità SDR e mai in HDR. In **figura 2.1** si possono vedere le velocità permette dalle diverse configurazioni.

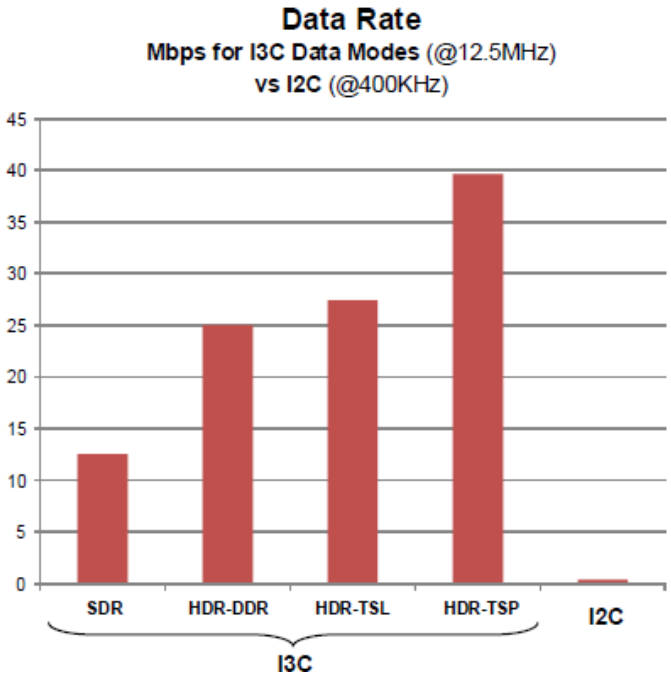

***Figura 2.1**: Velocità e data rate per I3C e I^2C*

2.3 Configurazione del Bus I3C

Ogni comunicazione sul bus avviene tramite una trasmissione o una ricezione di dati tra due o più dispositivi attraverso la linea SDA, in base al ruolo che ricoprono. Questo significa che i ruoli di trasmettitore e ricevitore non sono permanenti, ma cambiano in base alla direzione di trasmissione dei dati. Il Master è l'unico dispositivo che genera il segnale di sincronizzazione sulla linea SCL e lo trasmette agli altri dispositivi. Tutti gli altri dispositivi, diversi dal Master, sono definiti Slave. In altre parole, il Master è il dispositivo di riferimento che sincronizza e controlla la comunicazione.

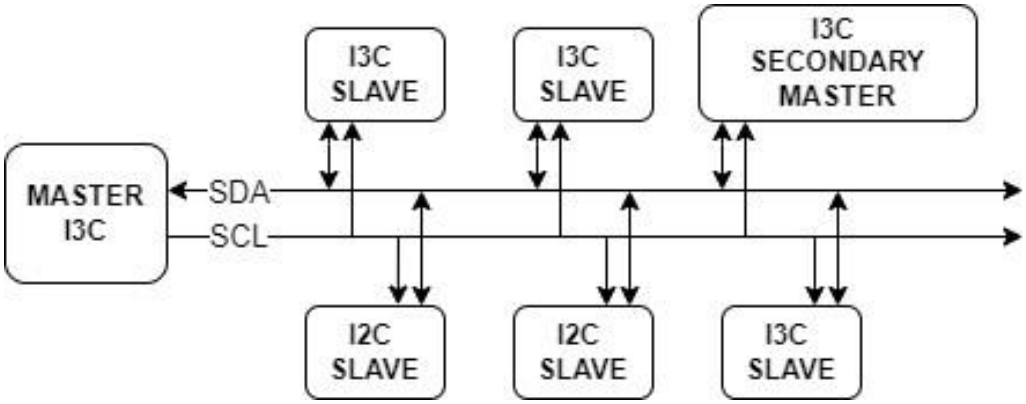

***Figura 2.2**: Bus con dispositivi I3C e I2C*

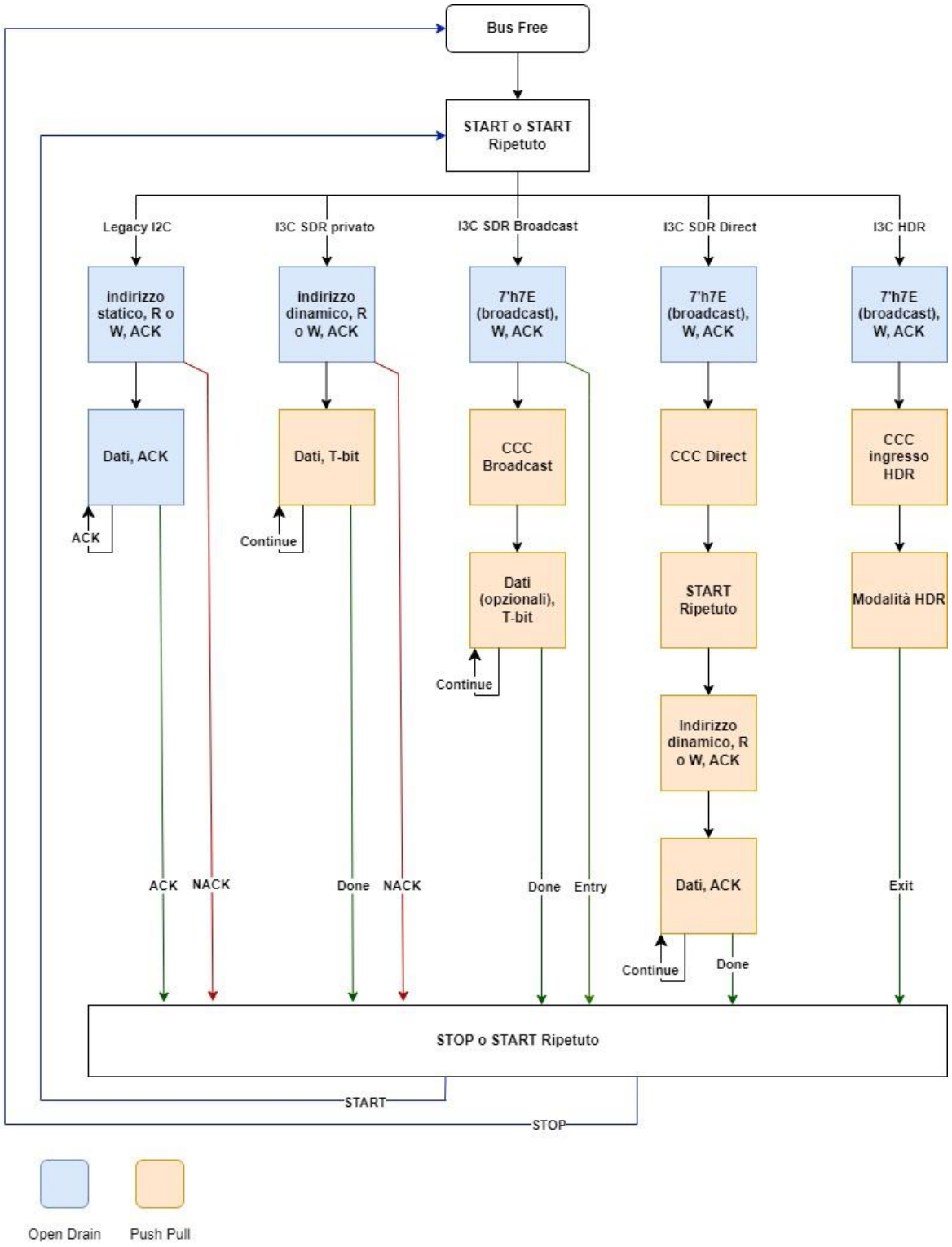

***Figura 2.3**: Flusso di comunicazione*

In **figura 2.3** è mostrato il flusso di comunicazione, a cui si aggiungono due considerazioni: il processo di assegnazione degli indirizzi dinamici (ENTDAA CCC) non è stato raffigurato, a questo verrà dedicato un capitolo a parte.

2.3.1 Modalità Open Drain e Push Pull

Come accennato precedentemente il segnale di sincronizzazione è configurato e generato esclusivamente dal Master. Nel protocollo I3C ci sono due possibili configurazioni per i pin di uscita: Open Drain e Push Pull (vedi **figura 2.3**).

La configurazione Open Drain è la più comune ed è l'unica condivisa anche dai dispositivi I2C. in questa configurazione il pin di uscita può essere impostato attivamente solo a livello logico basso, quello che in gergo si chiama Pull–Down, mentre lo stato a livello logico alto è gestito solamente in maniera passiva. Questo permette ai dispositivi che utilizzano la configurazione Open Drain di pilotare la linea SDA solamente da livello logico alto a livello logico basso.

Con il passaggio da protocollo I^2C a protocollo I3C è stata introdotta la configurazione Push Pull. È implementata solamente nei dispositivi I3C e ha la possibilità di pilotare attivamente la linea SDA sia a livello logico alto che basso. In questo caso di definisce attivo sia il Pull-Down che il Pull-Up.

Dal punto di vista circuitale la differenza tra le due configurazioni è data da una diversa implementazione della resistenza di Pull-Up. La configurazione Open Drain fa uso di una resistenza esterna al circuito integrato che ha il compito di "far salire" la tensione di uscita quando il transistor viene spento, altrimenti quest'ultima rimarrebbe in uno stato di alta impedenza. Nel caso di configurazione Push Pull la resistenza di Pull-Up è implementata con transistor PMOS ed è interna al circuito integrato, questo permette di "far salire" la tensione di uscita quando il segnale di ingresso al gate è uguale a zero.

Grazie all'implementazione della resistenza di Pull-Up con un transistor PMOS i tempi di salita della forma d'onda vengono ridotti notevolmente, permettendo una velocità del segnale di clock di 12.5 MHz.

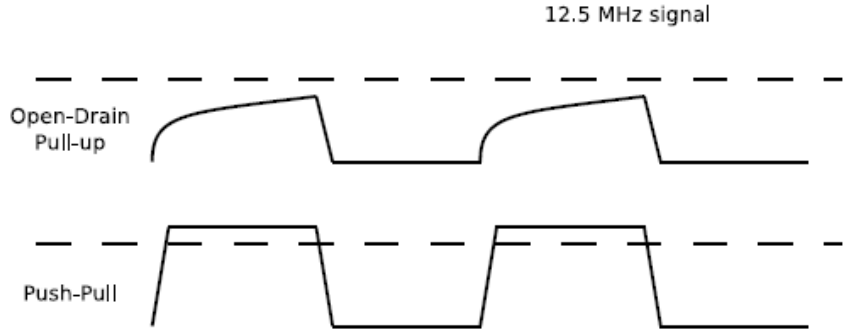

***Figura 2.4**: Forma d'onda per configurazione Open Drain e Push Pull*

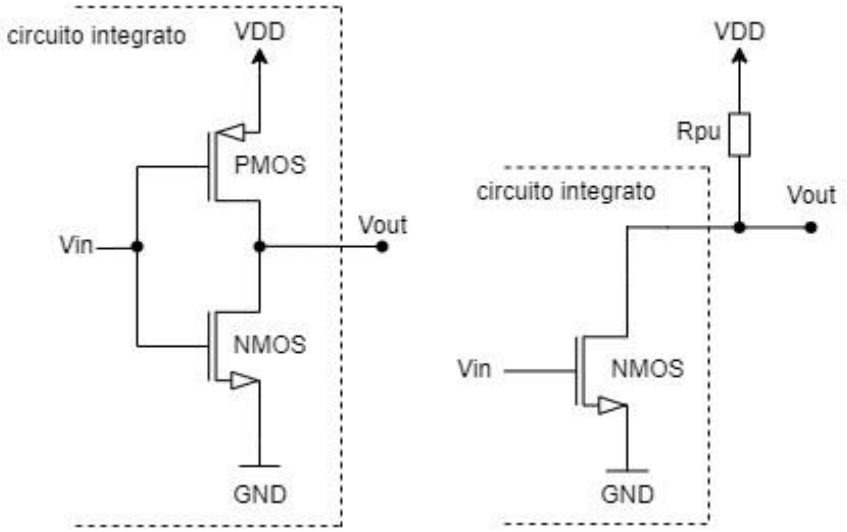

***Figura 2.5**: Configurazione Push-Pull (a sinistra) e Open Drain (a destra)*

2.3.2 Condizioni di START, STOP e START Ripetuto

Come illustrato in **figura 2.3**, l'inizio di qualunque tipo di comunicazione avviene con un segnale di START (S) e si conclude con un segnale di STOP (P) o START Ripetuto (Sr). La condizione di START è definita dalla transizione dal livello logico alto a basso della linea SDA, quando la linea SCL è a livello logico alto. Al contrario, la condizione di STOP è definita dalla transizione da un livello logico basso ad alto della linea SDA, mentre la linea SCL è a livello logico alto. Il segnale di START Ripetuto indica che il trasmettitore intende continuare la comunicazione con un ricevitore; quindi, il bus continua a rimanere occupato e segue una nuova comunicazione. Quando possibile è utile utilizzare lo START Ripetuto invece di passare dalla condizione di STOP e successivamente di START.

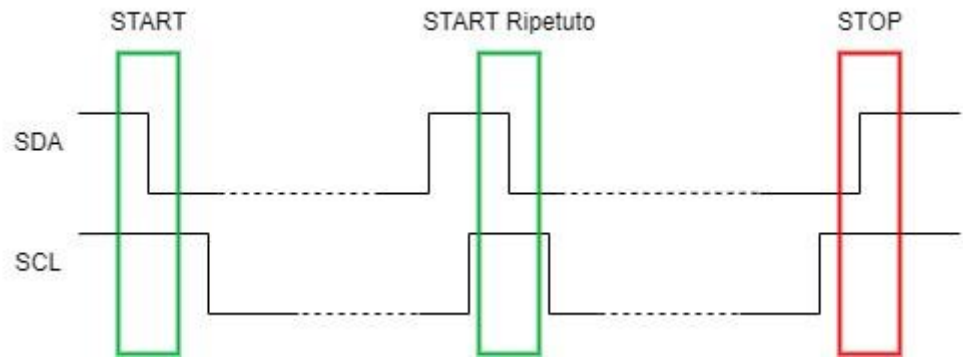

***Figura 2.5**: condizioni di START, STOP, START Ripetuto*

2.3.3 Acknoledge e Not- Acknoledge

L'utilizzo e la definizione dei segnali di ACK e NACK sono identici al protocollo I^2C.

Il bit di ACK viene trasmesso dal ricevitore per comunicare al trasmettitore la corretta ricezione dei dati. Ricevuto l'ACK è possibile proseguire con la comunicazione, in alternativa un segnale NACK comporta a un errore di comunicazione e all'interruzione della stessa. Le cause che possono portare un ricevitore a trasmettere un segnale di NACK sono diverse ma sempre legate a un errore nella trasmissione dei dati. Il bit ACK è sempre trasmesso in Open Drain.

Il segnale ACK è definito come segue: il trasmettitore, dopo l'ultimo bit di dato trasmesso lascia la linea SDA permettendo al ricevitore di abbassarla a livello logico basso. Il trasmettitore interpreta come bit ACK se la linea SDA è impostata a livello logico basso durante il semiperiodo alto del segnale SCL. In alternativa, se il ricevitore imposta la linea SDA a livello logico alto significa che è stato generato un segnale di NACK.

2.4 Trasmissione dei dati

In questa trattazione ci concentreremo sull'analisi della trasmissione dati nella modalità principale utilizzata dal protocollo I3C, ovvero la Single Data Rate (SDR). Questa presenta una struttura del Frame simile allo standard I^2C per ognuna delle tipologie di comunicazione: I3C SDR privato, Legacy I^2C, I3C SDR Broadcast e Direct.

La scelta dell'utilizzo della configurazione Open drain o Push Pull nella trasmissione dell'intestazione dell'indirizzo verrà affrontata nel **capitolo 2.5**, dedicato all'arbitraggio e alla gestione del clock.

2.4.1 Intestazione dell'indirizzo

L'intestazione dell'indirizzo ha la stessa struttura utilizzata nel protocollo I^2C, inizia con una condizione di START o START Ripetuto, seguono 8 bit (1 byte) suddivisi in 7 bit di indirizzo e 1 bit di lettura o scrittura (R/W) e si conclude con la condizione di ACK/NACK. Ha il compito di inizializzare una comunicazione motivo per cui deve essere trasmesso in Open drain per rispettare la condizione di arbitraggio. Di seguito viene descritta la struttura del Frame.

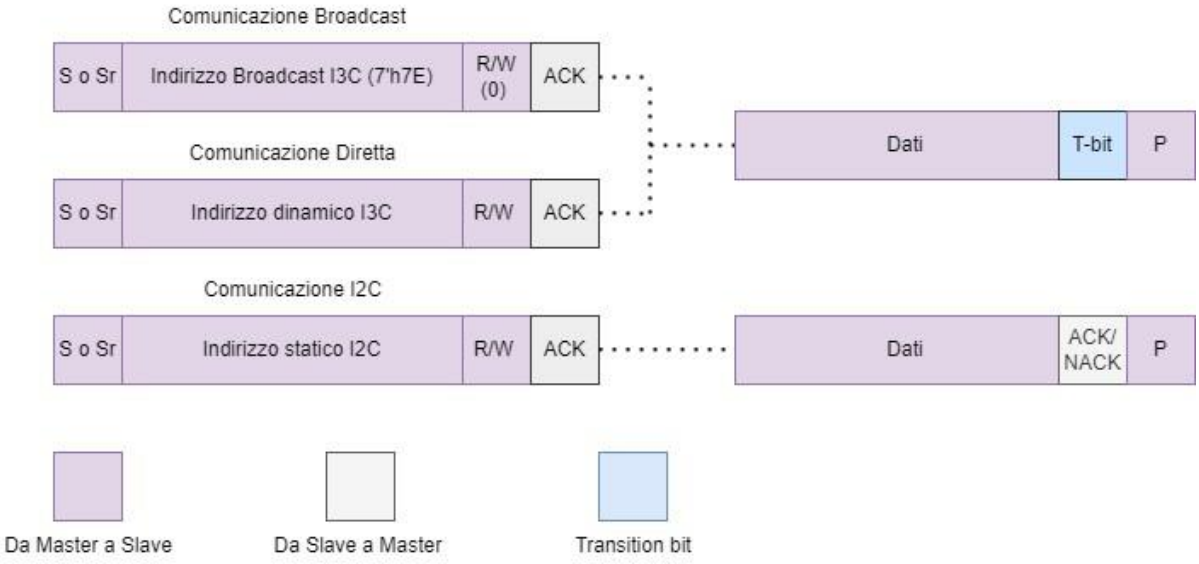

***Figura 2.7**: Comparazione dell'intestazione dell'indirizzo*

È importante aggiungere che il bit di lettura/scrittura nel caso di comunicazione con indirizzo Broadcast deve sempre essere impostato in scrittura, quindi a zero. Questo significa che l'intestazione dell'indirizzo, nel caso di comunicazioni Broadcast, avrà necessariamente i primi 8 bit con la configurazione seguente: 8'b11111100. Come precedentemente visto (vedi **figura 2.3**) le modalità di comunicazione SDR sono diverse e nel corso di questo capitolo verranno descritte singolarmente.

2.4.2 Legacy I^2C

La modalità Legacy I^2C si riferisce alla completa retrocompatibilità con lo standard I^2C, garantendo sia l'indirizzamento a 7 bit che a 10 bit. La struttura del Frame nel caso di lettura e scrittura è uguale per l'indirizzamento a 7 bit, cambia solo la direzione di comunicazione. Per l'indirizzamento a 10 bit vengono utilizzati due byte per trasmettere l'indirizzo, del secondo byte vengono considerati solo i tre bit meno significativi. In questo elaborato il protocollo I^2C è trattato in modo marginale, per una trattazione più approfondita si rimanda alla documentazione dell'I^2C.

***Figura 2.8**: Scrittura e lettura I2C*

2.4.3 Messaggio privato I3C (SDR)

La comunicazione privata I3C può essere sia di lettura che di scrittura, inizia sempre con il Master che trasmette sulla linea SDA l'indirizzo dinamico di una periferica (Slave), seguito dal bit di lettura/scrittura. Nel protocollo I3C non è previsto l'indirizzamento a 10 bit; quindi, l'intestazione dell'indirizzo sarà sempre formata da un solo byte seguito dal bit di ACK.

I possibili formati di trasferimento dati sono i seguenti:

- Master trasmettitore e Slave ricevitore. Il Master trasmette l'indirizzo dinamico dello Slave con il quale vuole comunicare seguito dal bit R/W a 0, dichiarando l'intenzione di scrittura. Lo Slave a cui corrisponde l'indirizzo inviato sul bus risponderà con ACK, mentre tutti gli altri si limiteranno a ignorare la richiesta e quindi a lasciare la linea SDA libera. Successivamente il Master inizia a trasmettere i dati in pacchetti da 8 bit, dopo ogni pacchetto viene concatenato e trasmetto il T-bit con la funzione di *Parity bit.*
- Master ricevitore e Slave trasmettitore. Il Master trasmette l'indirizzo dinamico dello Slave con il quale vuole comunicare seguito dal bit R/W a 1, dichiarando l'intenzione di lettura. Analogamente a caso precedente lo Slave a cui corrisponde l'indirizzo inviato sul bus risponderà con ACK. Successivamente lo Slave prende il controllo della linea SDA e inizia a trasmettere i dati in pacchetti da 8 bit, dopo ogni pacchetto viene concatenato e trasmetto il T-bit con la funzione di *End of Data.*

È inoltre possibile combinare i due formati: terminando la scrittura (o lettura) con uno START Ripetuto è possibile trasmettere un nuovo indirizzo con il bit R/W invertito rispetto al formato precedente.

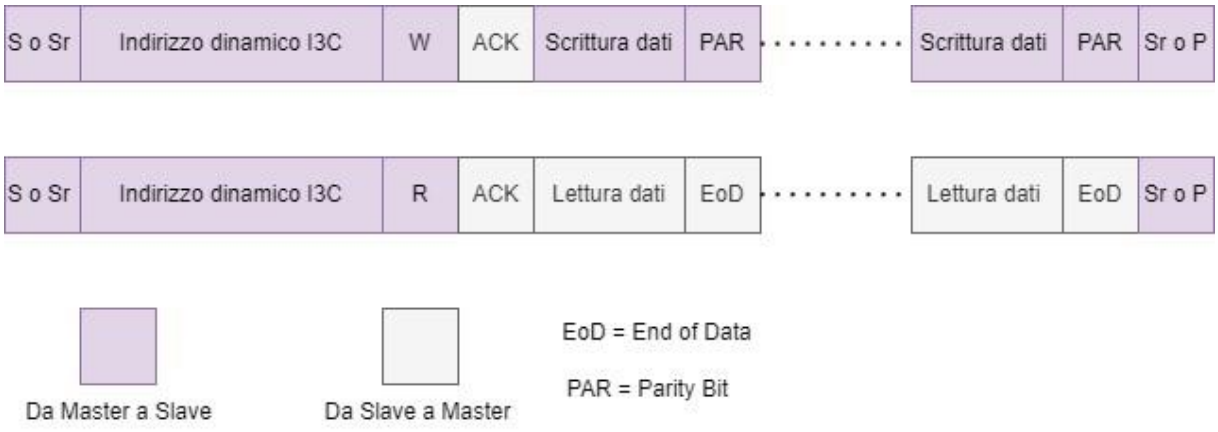

***Figura 2.9**: Scrittura e lettura per messaggio privato I3C*

2.4.3.1 T-Bit

Si inserisce un capitolo per descrivere il funzionamento del bit di transizione (T-bit). Si tratta di un'alternativa all'utilizzo del bit di ACK (utilizzato dal precedente protocollo I^2C) durante la trasmissione dei dati e ha due funzioni differenti a seconda del tipo di trasferimento dati (trasmissione o ricezione). L'utilizzo del T-bit rispetto al bit di ACK è preferibile perché la direzione di comunicazione rimane invariata durante tutto il trasferimento di molteplici byte. Si pensi a una semplice scrittura di tipo I^2C di 2 byte trasmessi da un Master verso una periferica, dopo il primo byte il Master deve lasciare la linea SDA allo Slave e attendere il bit di ACK e solo successivamente può riprendere il controllo della linea per trasmettere il secondo byte. Grazie all'introduzione del T-bit lo stesso trasferimento dei 2 byte avviene mantenendo invariata la direzione. Di seguito vengono approfondite le due tipologie di T-bit:

- ***Parity Bit**:* in caso di Master trasmettitore e Slave ricevitore il T-bit è il risultato del calcolo della parità dispari sul byte trasmesso, in altre parole il bit di parità dispari è il risultato dell'operazione di XOR sugli 8 bit di dato trasmessi. Ad esempio, se il byte trasmesso dal Master è composto dai seguenti bit: 11011001, l'operazione XOR(11011001) darà come risultato 0. Il Master manda la sequenza 110110010, lo Slave effettua l'operazione di XOR sui primi 8 bit e confronta il valore del T-bit. Se il T-bit non corrisponde vuol dire che c'è stato un errore nella comunicazione. per errore lo Slave riceve 110010010, l'operazione XOR(11001001) dà come risultato 1 che è diverso dal T-bit mandato dal Master.

- ***End of data**:* in caso di Master ricevitore e Slave trasmettitore il T-bit rappresenta l'intenzione dello Slave di continuare a trasmettere. Dopo ogni byte trasmesso se la linea SDA è impostata a livello logico basso dalle Slave significa che non ci sono più byte da inviare. In alternativa, se la linea SDA è impostata a livello logico alto significa che è presente un altro byte e quindi lo Slave continua a trasmettere dati.

2.4.4 Messaggio CCC Broadcast I3C (SDR)

I messaggi Broadcast sono indirizzati solamente agli Slave I3C che condividono l'indirizzo Broadcast (7'h7E) presente nell'intestazione. Questo permette di "filtrare" gli Slave I^2C e garantire che solo i dispositivi I3C riceveranno i successivi dati inviati dal Master. Questo genere di comunicazione può essere solamente in scrittura, quindi il bit R/W che segue l'indirizzo Broadcast è sempre impostato a 0. Successivamente all'intestazione dell'indirizzo, solamente gli Slave I3C risponderanno con ACK.

Generalmente l'indirizzo Broadcast viene utilizzato per inviare i *Common Command Code (CCC)* (vedi **capitolo 2.6)**, si tratta di comandi standard utilizzati solo dai dispositivi I3C, motivo per cui è necessario escludere i dispositivi I^2C da questo genere di comunicazione. I CCC si dividono in Broadcast o Direct. I CCC Broadcast sono indirizzati a tutti i dispositivi I3C.

La struttura del Frame per messaggi di tipo Broadcast è analoga a quelle precedenti: inizio della comunicazione con condizione di START o START Ripetuto, 7 bit di indirizzo, 1 bit di R/W (impostato a 0), 1 bit di ACK, 8 bit di dati (CCC), 1 bit di parità, eventuali byte di dati se previsti dal CCC e per concludere una condizione di STOP o START Ripetuto.

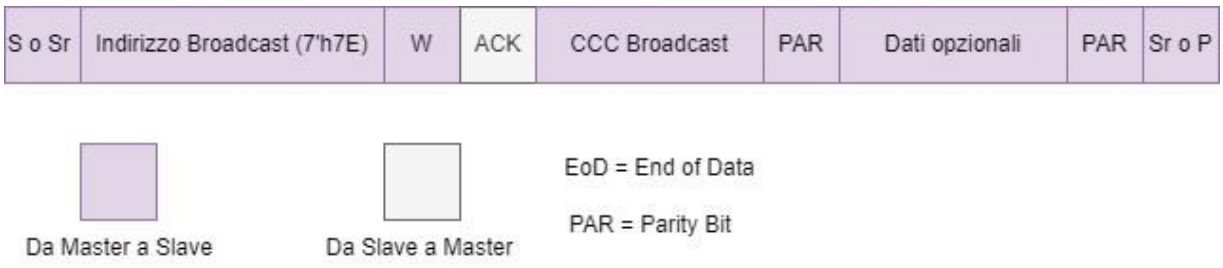

***Figura 2.10**: I3C CCC Broadcast*

2.4.5 Messaggio CCC Direct I3C (SDR)

A differenza dei messaggi CCC Broadcast, i comandi Direct sono indirizzati a uno specifico dispositivo I3C, motivo per cui, dopo l'indirizzo Broadcast 7'h7E e il comando CCC è necessario trasmettere l'indirizzo della periferica con la quale si vuole comunicare.

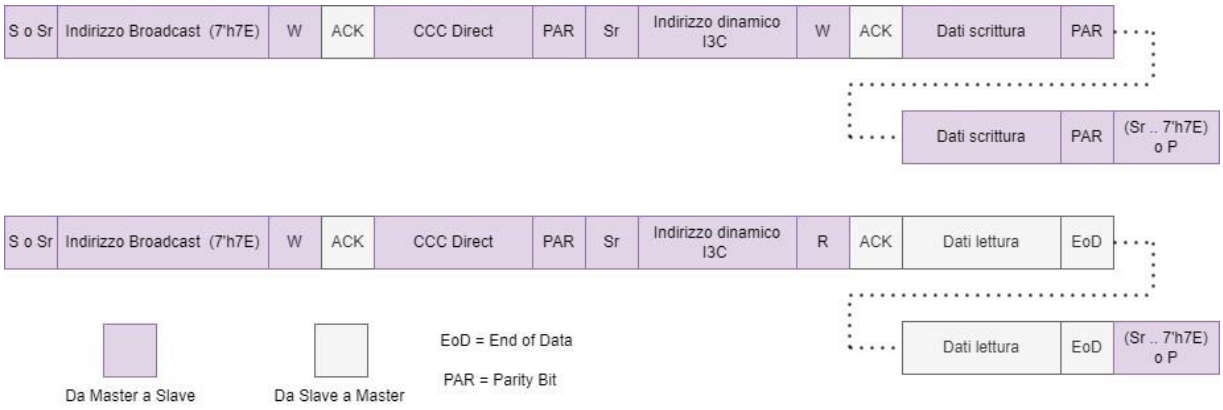

***Figura 2.11**: I3C CCC Direct*

2.5 Arbitraggio

Nel contesto del protocollo I3C, il concetto di arbitraggio si riferisce al processo attraverso il quale i dispositivi competono per accedere al bus e trasmettere dati. Poiché il bus I3C permette comunicazioni multi-Master ed eventi di interruzione (vedi **capitolo 2.9**) è necessario un meccanismo per gestire le richieste di accesso simultanee dei vari dispositivi. Il processo di arbitraggio dell'indirizzo determina quale dispositivo avrà il diritto di trasmettere prima i dati sul bus. L'intestazione dell'indirizzo che segue una condizione di START e non di START Ripetuto è soggetto al processo di arbitraggio, per permettere sia ai dispositivi I^2C che I3C di partecipare, durante l'arbitraggio l'indirizzo viene trasmesso in modalità Open Drain. Concluso il processo e deciso il dispositivo che ha "vinto" la contesa del bus si può passare alla modalità Push Pull se si tratta di comunicazione I3C.

Per comprendere meglio come avviene il processo di arbitraggio si propone un esempio di due Slave I3C, abilitati a eventi di interruzione, che vogliono trasmettere simultaneamente il proprio indirizzo sul bus. Per ipotesi il primo Slave ha indirizzo 7h5F (1011111) e il secondo 7h77 (1110111). Inizialmente il Master è in uno stato di idle, con la linea SCL alta e la linea SDA

alta, dato che il bus risulta libero entrambi gli Slave “tirano giù” la linea SDA per iniziare la comunicazione. Si ricorda che è sempre il Master a fornire il segnale di sincronizzazione; quindi, a seguito dell’evento di interruzione degli Slave il Master procede a trasmettere il segnale sulla linea SCL. A questo punto entrambi gli Slave trasmettono il proprio indirizzo a partire dal bit più significativo (MSB). Al secondo ciclo di clock l’indirizzo 7h5F tira giù la linea SDA per trasmettere 0, mentre 7h77, dovendo trasmettere 1 lascia la linea SDA a livello logico alto. Successivamente entrambi gli Slave devono trasmettere 1 e lasciano la linea SDA a livello logico alto. Il Risultato del processo di arbitraggio, ovvero il valore sulla linea SDA che il Master riceve al terzo ciclo di clock è 101. A questo punto, l’arbitraggio è stato vinto dallo Slave 7h5F che trasmette l’intero indirizzo, mentre lo Slave 7h77 rimane in attesa che il bus si liberi per riprovare a trasmettere. Attraverso questo processo, la priorità di accesso al bus è assegnata agli indirizzi con valore inferiore. Di seguito si propone un esempio grafico che descrive quanto detto.

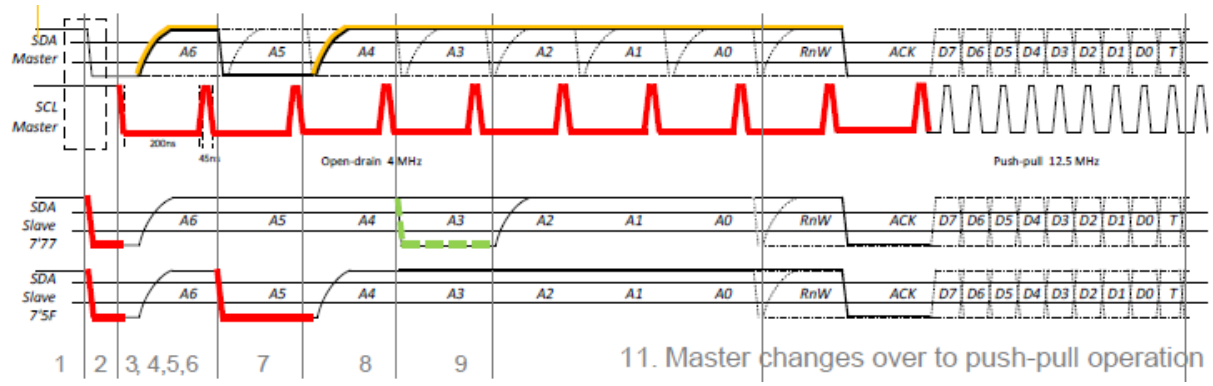

***Figura 2.12**: Arbitraggio tra indirizzo 7’h5F e 7’h77*

2.6 Common Command Code

I comandi CCC sono comandi standard, globali e supportati da ogni dispositivo I3C, possono essere trasmessi direttamente a specifiche periferiche I3C attraverso i Command Code di tipo Direct, oppure possono essere trasmessi simultaneamente a tutte le periferiche I3C presenti sul bus, attraverso i Command Code di tipo Broadcast. La trasmissione dei comandi CCC avviene solamente in modalità SDR e inizia sempre con l’indirizzo Broadcast 7’h7E e il bit R/W in scrittura: a livello logico basso. L’indirizzo 7’h7E è dedicato alle comunicazioni CCC ed è condiviso solamente dai dispositivi I3C, questo garantisce che siano solo queste periferiche a rispondere ai comandi Command Code.

Campo	**Definizione**	
S o **Sr**	Un CCC inizia sempre con START O START Ripetuto	
7'h7E / W / ACK	**7'h7E**	Indirizzo Broadcast in modo che tutti gli Slave I3C sul bus possano ricevere il CCC che segue.
	W	Il bit R/W è impostato a zero perché è sempre il Master a trasmettere i CCC.
	ACK	SDA impostato a livello logico basso da uno o più Slave.
Command Code / T	Il CCC è definito da 8 bit, seguito dal T-bit.	
Dati (opzionali) / T	Se previsto vengono trasmessi ulteriori 8 bit seguiti sempre dal T-bit.	
Sr / indirizzo Broadcast o P	Un comando CCC si conclude sempre con una condizione di STOP oppure START Ripetuto e indirizzo Broadcast.	

Tabella 2.1*: Struttura del Frame per Common Command Code*

Il sistema che permette di distinguere tra comando Broadcast e comando Direct è basato su una codifica a 8 bit di 255 valori divisi nel seguente modo:

- I valori esadecimali da 0x00 a 0x7F, che in binario corrispondono al range da 0000 0000 a 0111 1111 sono dedicati ai comandi di tipo Broadcast.
- I valori esadecimali da 0x80 a 0xFF, che in binario corrispondono al range da 1000 0000 a 1111 1111 sono dedicati ai comandi di tipo Direct.

Si sottolinea che la notazione 0x rappresenta un numero esadecimale. Spesso nei linguaggi di descrizione hardware come Verilog i numeri esadecimali sono rappresentati dalla notazione 8h in modo da specificare la base di rappresentazione, in questo esempio 8 bit.

Un aspetto importante derivato dalla codifica dei comandi CCC è la possibilità di distinguere la tipologia di comando esclusivamente dal bit più significati, cioè dal bit MSB (*Most Significant Bit*). Infatti, se il bit MSB ha valore 0 il comando è di tipo Broadcast, se ha valore 1 è di tipo Direct.

2.6.1 Command Code Broadcast

I Command Code Broadcast essendo indirizzati a tutte le periferiche I3C collegate al bus non richiedono di specificare l'indirizzo di destinazione. La **tabella 2.2** descrive alcuni dei CCC di tipo Broadcast più importanti.

Codifica Comando	**Tipologia**	**Nome Comando**	**Descrizione**
0x00	Broadcast	ENEC (Enable events Command)	Abilita l'interruzione del bus da parte di tutti Slave
0x01	Broadcast	DISEC (Disable Events Command)	Disabilita l'interruzione del bus da parte di tutti Slave
0x02-0x05	Broadcast	ENTAS0 – ENTAS3 (Enter Activity State 0-5)	Tutti gli Slave I3C entrano nello stato denominato Activity State per un tempo specifico.
0x06	Broadcast	RSTDAA (Reset Dynamic Address Assignment)	Elimina la precedente configurazione degli indirizzi dinamici
0x07	Broadcast	ENTDAA (Enter Dynamic Address Assignment)	Ingresso nel processo di assegnazione degli indirizzi dinamici

***Tabella 2.2**: Common Command Code Broadcast*

2.6.2 Command Code Direct

A differenza de Command Code Broadcast, l'utilizzo di quelli Direct prevede che l'intestazione dell'indirizzo sia sempre seguita dall'indirizzo specifico della periferica. La **tabella 2.3** descrive alcuni dei CCC di tipo Direct più importanti.

Codifica Comando	Tipologia	Nome Comando	Descrizione
0x80	Direct	ENEC	Abilita l'interruzione del bus da parte dello specifico Slave
0x81	Direct	DISEC	Disabilita l'interruzione del bus da parte dello specifico Slave
0x82-0x85	Direct	ENTAS0 – ENTAS3 (Enter Activity State 0-5)	Lo Slave I3C entra nello stato denominato Activity State per un tempo specifico.
0x06	Direct	RSTDAA (Reset Dynamic Address Assignment)	Elimina l'indirizzo dinamico dello Slave.
0x87	Direct	SETNEWDA (Set New Dynamic Address)	Il Master imposta un nuovo indirizzo dinamico allo specifico Slave.

***Tabella 2.3**: Common Command Code Direct*

2.7 ID provvisorio e registri BCR e DCR

Nel contesto del protocollo I3C è necessario che ogni dispositivo abbia dei registri identificativi che permettano di definirne le caratteristiche e le funzionalità supportate. Di seguito verranno descritti i registri BCR, DCR e l'ID provvisorio. Esiste anche il registro LVR, acronimo di Legacy Virtual Register, un registro a 8 bit di sola lettura utilizzato esclusivamente dai dispositivi I^2C contenente informazioni importanti come l'indirizzo statico e le modalità di comunicazione supportate, in questa trattazione verrà omessa la descrizione dettagliata di questo registro.

2.7.1 Bus Characteristics Register (BCR)

Ogni dispositivo connesso al bus I3C deve possedere il BCR. Si tratta di un registro a 8 bit di sola lettura che descrive le regole di utilizzo per l'assegnazione degli indirizzi dinamici e dei Common Command Code. In **tabella 2.4** è descritta la struttura del BCR. Risulterà di particolare interesse il valore di BCR[2].

BIT	Nome	Descrizione
BCR [7]	Ruolo del dispositivo [1]	2'b00 – Slave I3C 2'b01 – Master I3C 2'b10 – Riservato MIPI 2'b11 – Riservato MIPI
BCR [6]	Ruolo del dispositivo [0]	
BCR [5]	Riservato MIPI	0 - default
BCR [4]	Dispositivo Ponte	0 – dispositivo non ponte 1 – dispositivo ponte
BCR [3]	Capacità offline	0 – dispositivo risponde sempre 1 - dispositivo non risponde sempre
BCR [2]	IBI Mandatory data	0 – nessun dato a seguito di IBI 1 – dati obbligatori a seguito di IBI
BCR [1]	Richiesta IBI	0 – Slave abilitato a IBI 1 – Slave non abilitato a IBI
BCR [0]	Limitazione velocità dati	0 – nessuna limitazione 1 – nessuna limitazione

***Tabella 2.4**: Bus Characteristics Register*

2.7.2 Device Characteristics Register (DCR)

Ogni dispositivo I3C connesso al bus I3C deve possedere il DCR. Si tratta di un registro a 8 bit di sola lettura che descrive la tipologia di dispositivo (accelerometro, giroscopio, memoria etc.). In **tabella 2.5** è descritta la struttura del DCR.

BIT	Nome	Descrizione
DCR [7]	Device ID [7]	Codifica a 255 bit che descrive la tipologia di periferica. Valore di default: 8'b0.
DCR [6]	Device ID [6]	
DCR [5]	Device ID [5]	
DCR [4]	Device ID [4]	
DCR [3]	Device ID [3]	
DCR [2]	Device ID [2]	
DCR [1]	Device ID [1]	
DCR [0]	Device ID [0]	

***Tabella 2.5**: Device Characteristics Register*

2.7.3 ID provvisorio

A supporto di una corretta gestione degli indirizzi dinamici ogni dispositivo I3C connesso al bus I3C deve possedere un codice a 48 bit identificativo con la seguente struttura:

- **Bit [47:33]:** 15 bit riservati a ID di produzione MIPI.
- **Bit [32]:** definisce se i successivi 32 bit che corrispondono al ID provvisorio sono Random o definiti dal venditore.
- **Bit [32]:** 1'b1.
    - **Bit [31:0]:** valori Random generati dal dispositivo, con la possibilità di definirli attraverso il Command Code Enter Test Mode (ENTTM)
- **Bit [32]:** 1'b0.
    - **Bit [31:0]:** valori definiti dal venditore.

2.8 Gestione degli indirizzi dinamici

È il Master responsabile della procedura di assegnazione degli indirizzi dinamici che deve avvenire come primo processo a seguito dell'inizializzazione del bus.

1. **Configurazione Iniziale**: Il Master deve essere in uno stato adeguatamente configurato e deve avere memorizzati dal microprocessore il numero di dispositivi I3C che devono ricevere un indirizzo dinamico.

2. **Assegnazione degli Indirizzi Dinamici**: Il Master assegna gli indirizzi dinamici ai dispositivi I3C con indirizzo statico noto, utilizzando i comandi SETDASA o SETAASA. Se ci sono dispositivi che supportano solo SETDASA, vengono loro assegnati gli indirizzi dinamici individualmente. Se ci sono dispositivi sia I3C che I^2C, si invia prima il comando SETAASA per gli Slave solo I^2C, e poi il comando ENTDAA per assegnare gli indirizzi dinamici vai dispositivi I3C.

3. **Inizio della Procedura di Assegnazione Dinamica degli Indirizzi**: Il Master invia il comando di broadcast ENTDAA per iniziare la procedura.

4. **Riconoscimento dei Dispositivi**: Il Master invia uno START Ripetuto e l'indirizzo di broadcast I3C. Ogni dispositivo I3C che non ha ancora un indirizzo dinamico risponde all'indirizzo di broadcast.

5. **Comunicazione del Master**: Il Master continua a inviare solo il segnale di clock (SCL) e rilascia il segnale di dati (SDA) in modo che i dispositivi possano trasmettere il loro ID provvisorio, il BCR e il DCR.
6. **Trasferimento delle Caratteristiche dei Dispositivi**: I dispositivi che hanno risposto trasmettono le loro caratteristiche concatenando in 8 byte l'ID Provvisorio, il BCR e il DCR fino a quando perdono l'arbitraggio dell'indirizzo dinamico.
7. **Arbitraggio degli Indirizzi**: Il dispositivo con il valore concatenato più basso vince l'arbitraggio.
8. **Assegnazione dell'Indirizzo Dinamico**: Il Master trasferisce un indirizzo dinamico al dispositivo vincente.
9. **Ripetizione del Procedimento**: Il procedimento viene ripetuto fino a quando non si riceve un ACK da tutti i dispositivi presenti sul bus, cioè finché non si esauriscono i dispositivi che necessitano di un indirizzo dinamico.
10. **Fine della Procedura**: Il Master termina la procedura emettendo un segnale di STOP.

Per una comprensione migliore del Command Code ENTDAA in **figura 2.13** è descritta la struttura del Frame.

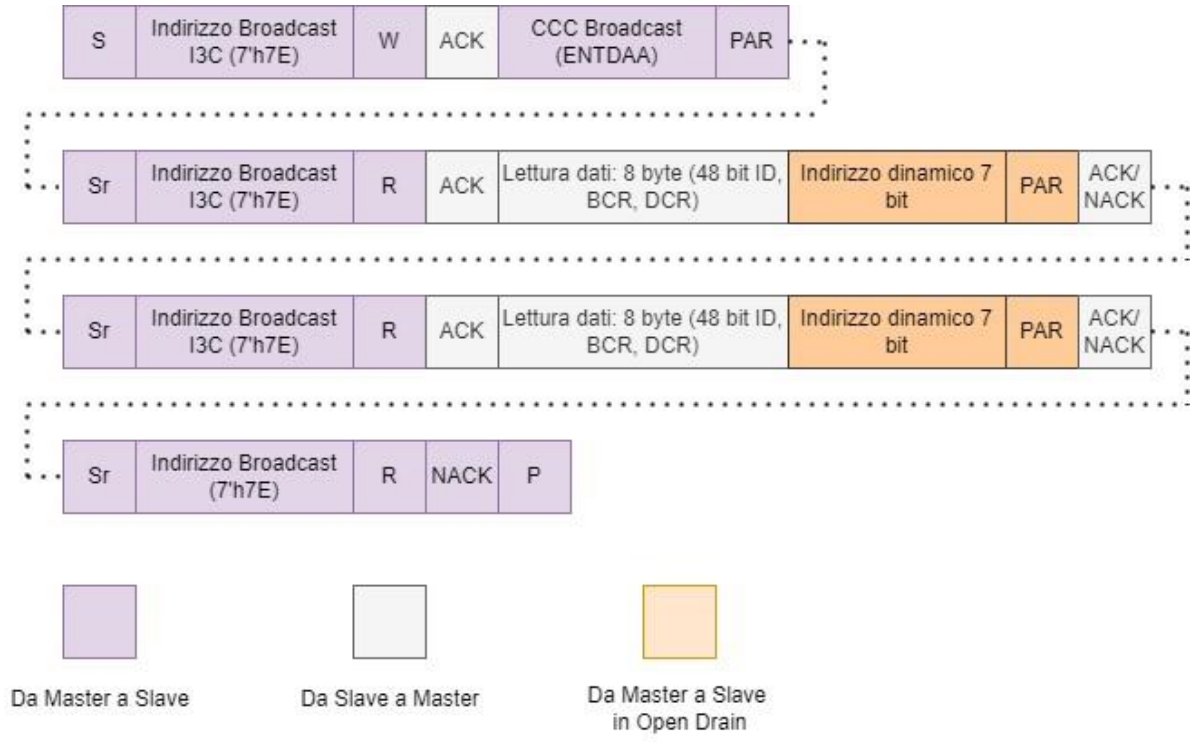

Figura 2.13: ENTDAA

Si aggiungono alcune considerazioni importanti:

- A seguito di una condizione di START l'intestazione dell'indirizzo è sempre trasmessa in Open Drain.
- Esclusivamente nel caso del comando ENTDAA il bit R/W che segui l'indirizzo Broadcast è impostato in lettura, cioè a 1.
- L'indirizzo dinamico inviato dal Master allo Slave è sempre trasmesso in Open Drain.
- Attraverso il comando RSTDAA è possibile resettare tutti gli indirizzi dinamici salvati per riassegnarli.

2.8.1 Scelta degli indirizzi dinamici

La scelta degli indirizzi dinamici che verranno assegnati alle periferiche I3C è lasciata al microprocessore, il quale ha il compito di inviare al Master il numero necessario di indirizzi e assicurarsi che non ci siano due o più indirizzi uguali. Il range completo di indirizzi a 7 bit va da 000 0000 a 111 1111, ma non tutti sono disponibili per essere assegnati alle periferiche, ad esempio si pensi all'indirizzo 111 1110 (7'h7E) che è riservato alle comunicazioni di tipo broadcast oppure all'indirizzo 000 0010 (7'h02) riservato agli eventi Hot Join.

Gli indirizzi disponibili per l'assegnazione sono 108 e sono distribuiti come segue:

- Da 000 1000 a 011 1101 (7'h08 – 7'h3D) per un totale di 54 indirizzi.
- Da 011 1111 a 101 1101 (7'h3F – 7'h5D) per un totale di 31 indirizzi.
- Da 101 1111 a 110 1111 (7'h5F – 7'h6D) per un totale di 15 indirizzi.
- Da 110 1111 a 111 0101 (7'h6F – 7'h75) per un totale di 7 indirizzi.
- 111 0111 (7'h77)

2.9 Eventi di interruzione

Rappresentano una novità introdotta con il protocollo I3C che permette ai dispositivi I3C di generare segnalazioni di interruzione verso il Master. Questo è un modo efficiente per notificare al Master eventi critici o richieste di comunicazione, riducendo la latenza rispetto agli approcci tradizionali che richiedono la scansione continua da parte del Master di tutti i dispositivi.

2.9.1 In–Band Interrupt (IBI)

***Figura 2.14**: In-Band Interrupt (IBI)*

L'evento di *In–Band Interrupt* (IBI) permette alle periferiche di inviare un segnale di interruzione sulla linea SDA per richiedere l'attenzione del Master.

- L'evento di interruzione è permesso solo se il bus si trova in uno stato libero dal almeno 5 microsecondi, ovvero se le linee SDA e SCL sono entrambe a livello logico alto.
- L'interruzione avviene quando una periferica imposta la linea SDA a livello logico basso.
- L'evento di interruzione è permesso solo se precedentemente è stata fatta l'assegnazione degli indirizzi dinamici.
- È sempre il Master a fornire il segnale di sincronizzazione sulla linea SCL.
- L'evento di interruzione è disabilitato sotto richiesta del microprocessore attraverso il Command code DISEC.
- Lo Slave trasmette il proprio indirizzo in modalità Open Drain per rispettare le condizioni di arbitraggio, di conseguenza gli Slave con valore dell'indirizzo più basso hanno precedenza di accesso al bus.
- L'indirizzo letto dal Master viene mandato al modulo di memoria dedicato ad associare i dati ID, BCR, DCR con l'indirizzo dinamico (nel **capitolo 4** verrà approfondita la struttura di questo modulo). Come descritto in **tabella 2.4** è previsto che gli Slave che hanno il bit BCR [2] uguale a '1' trasmettano altri dati successivi all'indirizzo.
- La lettura degli eventuali dati avviene in configurazione Push Pull.
- La conclusione della comunicazione avviene con un segnale di STOP, riportando il bus in uno stato libero.

2.9.2 Hot Join

Un dispositivo che viene collegato al bus I3C, dopo che questo è stato configurato, può richiedere al Master l'assegnazione di un indirizzo dinamico attraverso il processo di Hot join. Questo permette una gestione più flessibile delle periferiche che vengono collegate al bus, senza il bisogno di riconfigurare tutti gli indirizzi dinamici quando un nuovo dispositivo si aggiunge.

La nuova periferica manda una segnalazione di interruzione in modo analogo agli eventi *In-Band Interrupt* in modo da richiedere l'attenzione del Master. Successivamente invia l'indirizzo dedicato al processo di Hot Join, ovvero 000 0010 (7'h02), il Master riconosce l'indirizzo e procede ad assegnare al nuovo dispositivo un indirizzo dinamico. È necessario che sia il microprocessore a fornire al master il nuovo indirizzo da inviare alla periferica.

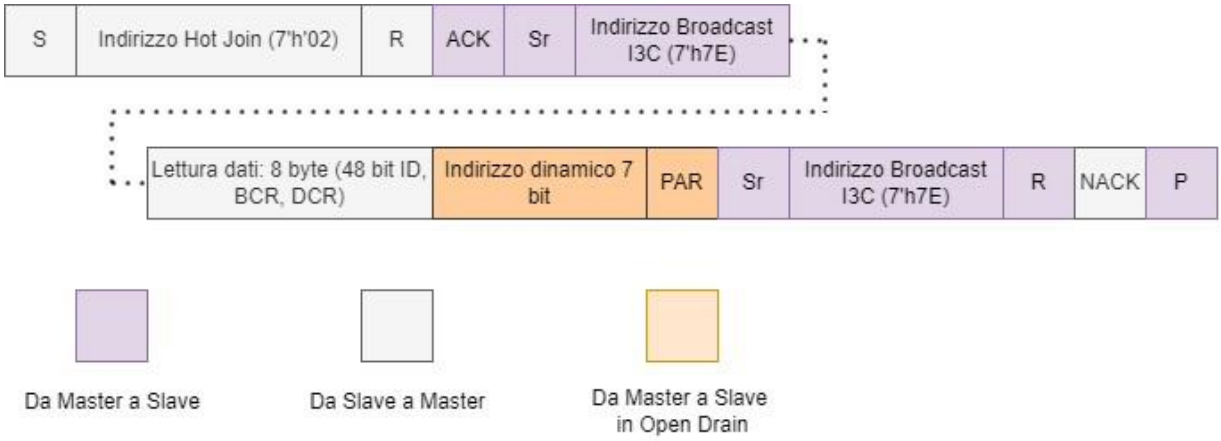

Figura 2.15: *Hot Join*

2.10 Conclusioni

In questo capitolo è stato descritto il funzionamento del protocollo I3C facendo stretto riferimento alla documentazione ufficiale rilasciata da MIPI Alliance. Sono state descritte solamente le funzioni implementate nel lavoro di tesi. Per una descrizione completa si rimanda alla documentazione ufficiale in bibliografia.

Capitolo 3

Architettura ISA RISC-V

Il RISC-V, noto anche come "risc-five", è un “insieme di istruzioni”, definito anche come ISA (*Instruction Set Architecture*) basato sul concetto di Reduced Instruction Set Computer (RISC) ed è definito come uno standard aperto. A differenza di molte altre architetture ISA disponibili sul mercato, il RISC-V è distribuito sotto una licenza open source, il che significa che non richiede l'acquisto di licenze proprietarie per essere utilizzato. Questa caratteristica ha portato molte aziende a offrire o annunciare hardware basato su questo standard, mentre numerosi sistemi operativi open source sono compatibili con esso. Inoltre, sono disponibili diverse toolchain che supportano l'implementazione e lo sviluppo basato su questo insieme di istruzioni.

3.1 L’architettura RISC

Come già menzionato, RISC indica l'abbreviazione di *Reduced Instruction Set Computer*. Questo termine denota un approccio di progettazione per microprocessori che mira a semplificare e rendere più lineare l'architettura. Ma per avere una comprensione completa, è opportuno iniziare dall'inizio e fornire un contesto graduale.

Negli anni '60, l'assenza di compilatori costringeva i programmatori a scrivere direttamente in codice macchina o assembly. Per agevolare questo processo, i progettisti di hardware iniziarono ad includere nel set di istruzioni dei loro processori alcune istruzioni, anche complesse, per simulare le funzioni di alto livello dei linguaggi di programmazione. Inoltre, per semplificare il lavoro dei programmatori, il set di istruzioni dei processori cominciò a supportare metodi di indirizzamento dati complessi che permettevano di manipolare direttamente i dati in memoria, senza passare attraverso i registri interni del processore. Tuttavia, poiché il processore può elaborare solo dati contenuti nei registri, quando venivano utilizzati questi metodi di indirizzamento particolari, il processore doveva copiare i dati in un registro interno per eseguire l'operazione e successivamente salvare il risultato in memoria.

Un altro vantaggio dell'utilizzo di un set di istruzioni complesso riguardava l'occupazione di memoria. Le istruzioni complesse permettevano di realizzare programmi con poche righe di codice, il che era un vantaggio considerando la costosa memoria dell'epoca. Questo spiega la ricchezza dei set di istruzioni dei processori e la loro codifica a lunghezza variabile per massimizzare la densità di istruzioni in memoria.

Le CPU disponevano di pochi registri per due motivi principali:

1. La realizzazione di registri all'interno dei processori era costosa, poiché richiedeva più transistor, sacrificando le altre unità funzionali del processore.
2. I molti registri richiedevano più bit per l'indirizzamento, aumentando il fabbisogno di memoria per l'immagazzinamento delle istruzioni e la latenza.

Questi fattori spinsero i progettisti a sviluppare istruzioni con metodi di accesso alla memoria complessi, ad esempio istruzioni in grado di caricare, sommare e salvare dati in memoria con una sola istruzione. Processori con queste caratteristiche nel loro set di istruzioni furono definiti come CISC, ovvero *Complex Instruction Set Computer*.

L'obiettivo principale era realizzare processori in grado di eseguire operazioni con qualsiasi tipo di indirizzamento.

Tuttavia, alla fine degli anni '70, si dimostrò che la maggior parte dei modi di indirizzamento disponibili non venivano utilizzati all'interno dei programmi. Questo fu dovuto alla diffusione dei compilatori che ignoravano i modi di indirizzamento più complessi, concentrandosi su quelli più semplici. Inoltre, molti istruzioni "esotiche" risultavano poco utilizzate e a volte più lente del codice scritto con istruzioni generiche.

In quel periodo, le velocità di CPU e memoria erano simili, ma si prevedeva un divario crescente a favore delle CPU in futuro. Per ridurre il problema, si pensò di includere più registri e cache all'interno dei processori per ridurre gli accessi alla memoria. Tuttavia, ciò richiedeva molto spazio, quindi si cercò di semplificare la complessità dei processori. Si scoprì che le CPU dell'epoca erano sovradimensionate rispetto alle reali esigenze dei programmi.

Di conseguenza, si iniziò a concentrarsi sulle operazioni più comuni, rendendo l'esecuzione più veloce eliminando istruzioni e modi di indirizzamento poco usati. Ciò portò all'ideazione dell'approccio RISC: un set di istruzioni ridotto con metodi di accesso semplici e lineari,

ottimizzati per essere eseguiti rapidamente, spesso in un solo ciclo di clock. Questo approccio consentì l'implementazione di soluzioni di tipo pipeline, che rendono i processori più efficienti.

In sintesi, i vantaggi dell'architettura RISC includono un'architettura semplificata e snella, semplificando la parte hardware, l'avere a disposizione un set di istruzioni costituito da comandi semplici e, per concludere, possedere una decina di registri interni al processore così dal velocizzare l'accesso alla memoria. Tuttavia, il vantaggio della semplicità comporta una grande occupazione di memoria da parte del codice. La semplicità delle istruzioni è il motivo per cui il codice per architetture RISC tende ad essere più corposo rispetto alle architetture CISC. Inoltre, si complica il lavoro per chi traduce le istruzioni complesse ad alto livello in istruzioni semplici implementate utilizzando i comandi ISA.

A titolo esemplificativo si mettono a confronto un processore RISC, il PowerPC G4 e un processore CISC, il Pentium4. Il primo è un processore a 32 bit che fu utilizzato da Apple tra il 2001 e il 2006 nei suoi PowerBook G4. Il secondo è un processore prodotto da Intel commercializzati tra il 2000 e il 20008

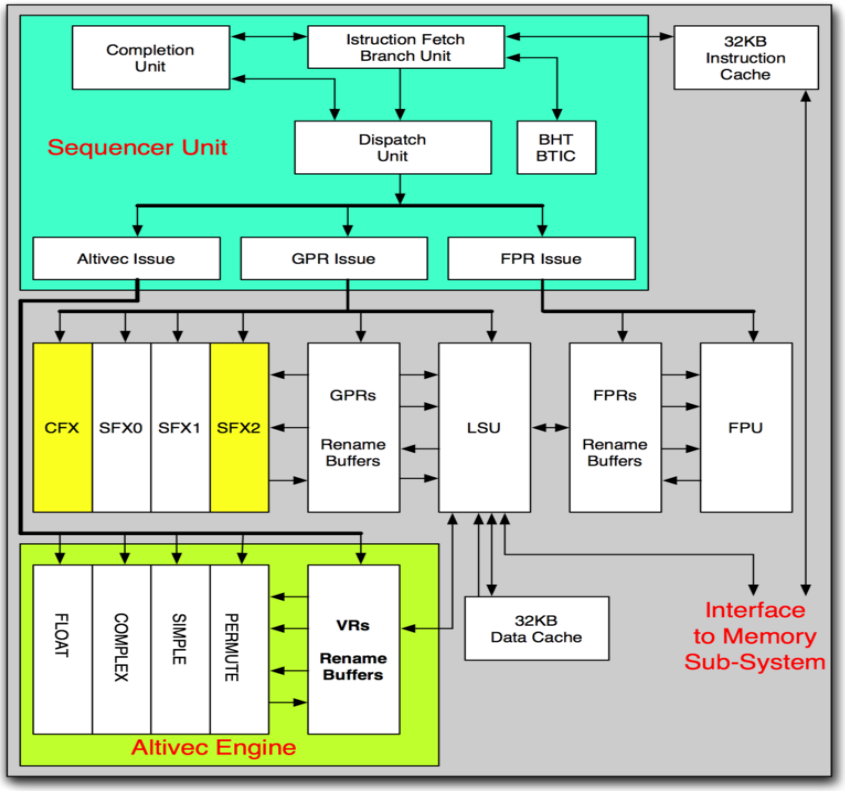

Figura 3.1: *Architettura del PowerPC G4,*

si osservi la semplicità di questa architettura.

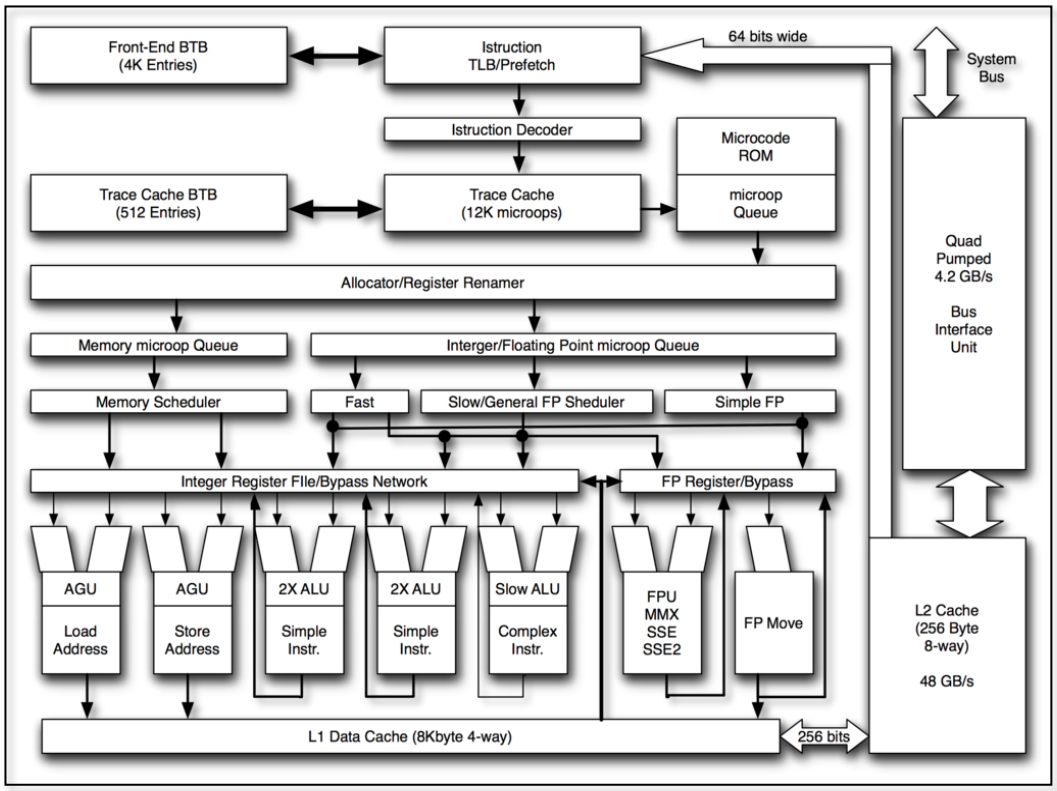

Figura* 3.2: Architettura del Pentium4; notare la differente complessità architetturale rispetto al PowerPC G4*

3.2 Il RISC-V

Il Dottor Krste Asanović e i due laureati Yunsup Lee e Andrew Waterman hanno iniziato a collaborare sullo schema RISC-V nel maggio del 2010 come parte del Parallel Computing Laboratory (Par Lab) presso l'Università di Berkeley in California, dove il Dottor David Patterson dirigeva il laboratorio. Il linguaggio di descrizione dell'hardware Chisel17, adoperato per la progettazione di numerosi processori RISC-V, è stato altresì elaborato nel Par Lab.

Il Par Lab costituiva un'iniziativa di ricerca quinquennale volta a promuovere gli studi sui sistemi di calcolo parallelo, finanziata da Intel e Microsoft con un fondo di 10 milioni di dollari distribuiti su cinque anni, precisamente dal 2008 al 2013. Nel corso del suo ciclo di attività, il laboratorio ha ricevuto supporto finanziario da varie imprese e dallo stato della California. Sebbene il progetto nel suo insieme non abbia ricevuto finanziamenti federali, Yunsup Lee e Andrew Waterman hanno ottenuto qualche finanziamento dal progetto Fototonico del DARPA18, utilizzato per sostenere parte dello sviluppo dell'implementazione del processore (ma non dell'ISA RISC-V). Tutti i progetti interni al Par Lab erano caratterizzati da un modello

open source e si avvalevano della licenza Berkeley Software Distribution (BSD), tra cui RISC-V e Chisel. Nel maggio del 2011, il Par Lab ha reso pubblico un primo rapporto al fine di delineare il set di istruzioni del RISC-V.

Nel 2015 è stata istituita la "RISC-V Foundation", un'organizzazione senza scopo di lucro con l'obiettivo di creare una comunità aperta di innovatori nell'ambito del software e dell'hardware basati sull'ISA RISC-V. Inizialmente, i membri interni della RISC-V Foundation erano 36. Attualmente, la comunità conta oltre 750 membri, tra cui aziende di rilievo come Google, Microchip, Nvidia, Cadence, Xilinx e Samsung.

3.3 ISA del RISC-V

Un'architettura RISC-V si caratterizza per la sua struttura modulare, essendo definita come un'architettura base a interi, la quale rappresenta un requisito fondamentale in ogni sua implementazione, cui possono poi essere aggiunte diverse estensioni. Le strutture a base intera si presentano simili ai primi processori RISC, ma con notevoli differenze quali l'assenza dei cosiddetti slot di ritardo nei branch e la possibilità di adottare una codifica delle istruzioni a lunghezza variabile. L'architettura base è vincolata a un insieme essenziale di istruzioni, atte a fornire una base solida per compilatori, assemblatori, linker e sistemi operativi, costituendo così un "scheletro" di strumenti ISA e software attorno ai quali possono essere costruite architetture di processori più specializzate. Nonostante comunemente si parli di ISA RISC-V, in realtà RISC-V comprende una famiglia di diverse ISA, attualmente rappresentate da quattro ISA di base. Ogni insieme di istruzioni a base intera si distingue per la dimensione dei registri, la dimensione dello spazio degli indirizzi e il numero di registri utilizzati.

Le due principali varianti della base intera, RV32I e RV64I19, offrono rispettivamente spazi di indirizzamento a 32 e 64 bit e utilizzano una rappresentazione in complemento a due per i valori interi con segno.

RISC-V è stato progettato per consentire una vasta gamma di personalizzazioni. Ogni set di istruzioni può essere esteso con una o più estensioni opzionali; il codice di ogni insieme di istruzioni RISC-V è diviso in tre categorie distinte: standard, riservato e personalizzato. Le codifiche standard sono definite dalla Fondazione e non entrano in conflitto con altre estensioni standard per la stessa ISA di base. Le codifiche riservate non sono attualmente utilizzate ma sono riservate per estensioni standard future. Le codifiche personalizzate non devono mai essere

impiegate per estensioni standard e sono disponibili per estensioni non standard specifiche del produttore.

Per agevolare lo sviluppo del software, sono definite diverse estensioni standard che implementano operazioni come la moltiplicazione, la divisione, le operazioni atomiche e le operazioni aritmetiche in virgola mobile, sia a singola che doppia precisione. L'ISA base, indicata con "I" (preceduta da RV32 o RV64, a seconda della dimensione del registro intero), include istruzioni per il calcolo su interi, il caricamento e l'archiviazione di interi e le istruzioni di controllo del flusso. L'estensione standard per la moltiplicazione e la divisione di interi è chiamata "M" e aggiunge istruzioni per eseguire tali operazioni sui registri. L'estensione standard per le operazioni atomiche è denominata "A" e include istruzioni per la lettura, la modifica e la scrittura atomica di valori in memoria. L'estensione per la virgola mobile a singola precisione è indicata con "F" e introduce registri a virgola mobile, istruzioni di calcolo e istruzioni di caricamento e archiviazione, tutte a singola precisione. L'estensione per la virgola mobile a doppia precisione è chiamata "D" e amplia le funzionalità già presentate con l'estensione a singola precisione. L'estensione standard "C" fornisce un set di istruzioni "compresse", rappresentando una versione a 16 bit delle istruzioni più comuni.

3.3.1 Formato delle istruzioni

l'ISA RISC-V di base possiede istruzioni di lunghezza fissa pari a 32 bit. Tuttavia,lo schema di codifica RISC-V standard è progettato per supportare estensioni ISA con istruzioni di lunghezza variabile, in cui ogni istruzione può essere un qualsiasi numero di pacchetti di lunghezza pari a 16 bit. Nella versione RV32I dell'ISA esistono quattro tipi di istruzione (R /I /S /U). Tutti i tipi sopra elencati hanno una lunghezza fissa di 32 bit e devono essere allineati su un limite di quattro byte in memoria.

Name	Description	Version	Status[a]
Base			
RVWMO	Weak Memory Ordering	2.0	Ratified
RV32I	Base Integer Instruction Set, 32-bit	2.1	Ratified
RV32E	Base Integer Instruction Set (embedded), 32-bit, 16 registers	1.9	Open
RV64I	Base Integer Instruction Set, 64-bit	2.1	Ratified
RV128I	Base Integer Instruction Set, 128-bit	1.7	Open
Extension			
M	Standard Extension for Integer Multiplication and Division	2.0	Ratified
A	Standard Extension for Atomic Instructions	2.1	Ratified
F	Standard Extension for Single-Precision Floating-Point	2.2	Ratified
D	Standard Extension for Double-Precision Floating-Point	2.2	Ratified
G	Shorthand for the base integer set (I) and above extensions (MAFD)	N/A	N/A
Q	Standard Extension for Quad-Precision Floating-Point	2.2	Ratified
L	Standard Extension for Decimal Floating-Point	0.0	Open
C	Standard Extension for Compressed Instructions	2.0	Ratified
B	Standard Extension for Bit Manipulation	0.92	Open
J	Standard Extension for Dynamically Translated Languages	0.0	Open
T	Standard Extension for Transactional Memory	0.0	Open
P	Standard Extension for Packed-SIMD Instructions	0.2	Open
V	Standard Extension for Vector Operations	0.9	Open
N	Standard Extension for User-Level Interrupts	1.1	Open
H	Standard Extension for Hypervisor	0.4	Open
ZiCSR	Control and Status Register (CSR)	2.0	Ratified
Zifencei	Instruction-Fetch Fence	2.0	Ratified
Zam	Misaligned Atomics	0.1	Open
Ztso	Total Store Ordering	0.1	Frozen

a. ^ Frozen parts are expected to have their final feature set and to receive only clarifications before being ratified.

***Figura 3.3**: Tabella riassuntiva su ISA di base e sue estensioni.*

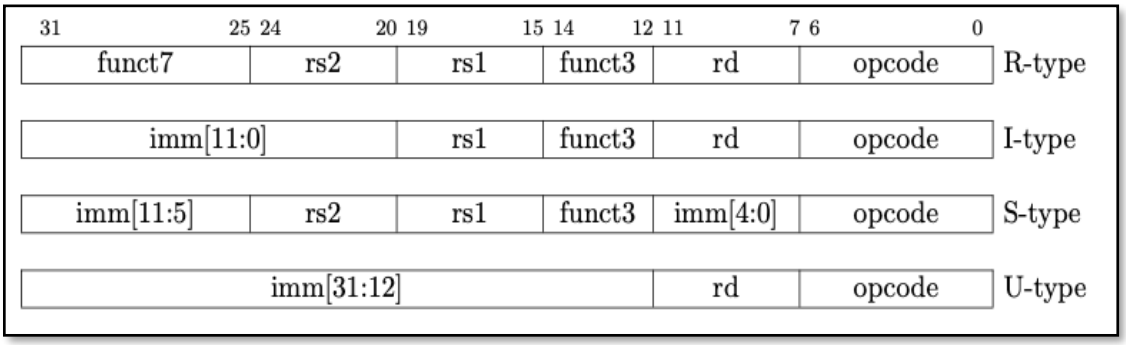

Figura 3.4: *istruzioni base del RISC-V a 32 bit*

Esistono altre due varianti, (B/ J) basate sulla gestione del campo *immediate*:

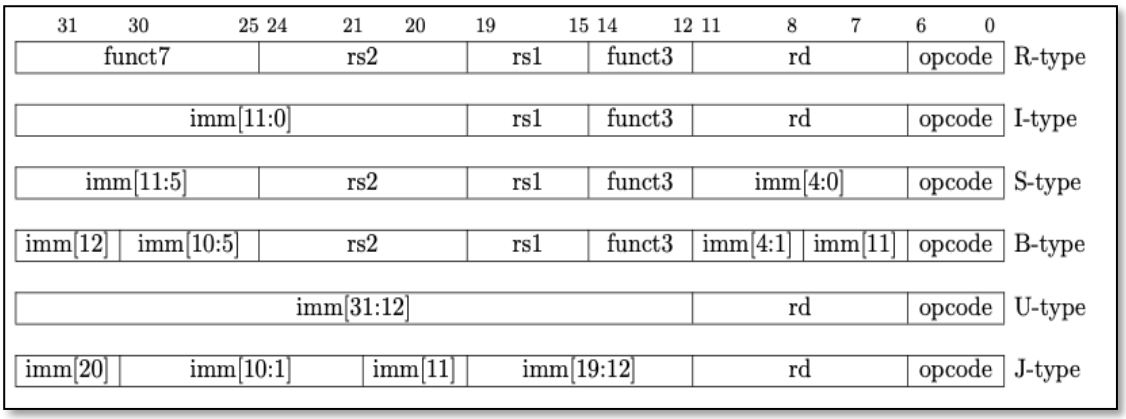

Figura 3.5: *set completo delle istruzioni del RISC-V a 32 bit.*

3.3.2 Register Set

RISC-V possiede 32 registri non privilegiati a valori interi e, quando implementa l'estensione, altri 32 a valori in virgola mobile.

Ad eccezione delle istruzioni che fanno accesso alla memoria, le istruzioni indirizzano solamente i registri interni. Il primo registro interno è il "registro di zero" e i rimanenti sono registri di uso generale [21]. Esiste un ulteriore registro non privilegiato ed è il *Program Counter*.

Register name	Symbolic name	Description	Saved by
32 integer registers			
x0	Zero	Always zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5	t0	Temporary / alternate return address	Caller
x6–7	t1–2	Temporary	Caller
x8	s0/fp	Saved register / frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function argument / return value	Caller
x12–17	a2–7	Function argument	Caller
x18–27	s2–11	Saved register	Callee
x28–31	t3–6	Temporary	Caller
32 floating-point extension registers			
f0–7	ft0–7	Floating-point temporaries	Caller
f8–9	fs0–1	Floating-point saved registers	Callee
f10–11	fa0–1	Floating-point arguments/return values	Caller
f12–17	fa2–7	Floating-point arguments	Caller
f18–27	fs2–11	Floating-point saved registers	Callee
f28–31	ft8–11	Floating-point temporaries	Caller

Figura 3.6: *Register sets*

3.3.3 Accesso alla memoria

Come molti progetti di architettura RISC, RISC-V presenta un'organizzazione load-store; cosa comporta ciò? Significa che le istruzioni si rivolgono esclusivamente ai registri; viene eseguita un'operazione di ‘load' per trasferire un dato dalla memoria al registro e un'operazione di ‘store' per fare il contrario.

La maggior parte delle operazioni di load e store contiene un offset di 12 bit e due identificatori di registro. Uno rappresenta il registro base, mentre l'altro funge da punto di partenza per una “store” (o come destinazione per una “load”).

L'offset viene sommato al registro base per ottenere l'indirizzo. L'unione dell'offset e del registro base consente alle singole istruzioni di accedere alle strutture dati. Ad esempio, se il registro

base indica l'inizio di uno stack, le singole istruzioni possono accedere alle variabili all'interno dello stesso.

3.4 Introduzione al RI5CY

In questo paragrafo verranno discussi solamente gli aspetti architetturali del microprocessore che sono serviti per poter svolgere il lavoro di tesi.

RI5CY è un *processor-core* con architettura RISC-V a 4 stadi e a 32 bit. L'ISA del RI5CY è stata estesa per poter supportare alcune istruzioni aggiuntive, tra cui *loop* hardware, istruzioni di caricamento/memorizzazione post-incremento e ulteriori istruzioni ALU che non fanno parte dello standard RISC-V. Lo schema a blocchi di tale processore è riportato nella figura successiva:

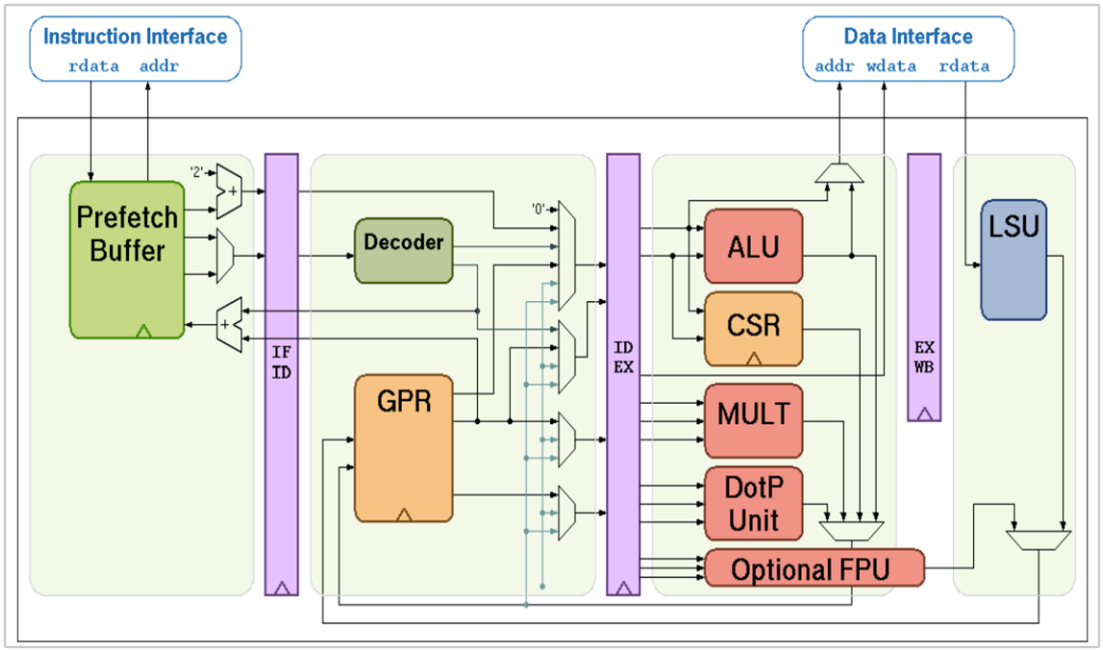

***Figura 3.7**: schema a blocchi del core RI5CY.*

Il processore supporta i seguenti tipi di istruzione:

- *RV32I Base Integer Instruction Set*
- *RV32C Standard Extension for Compressed Instruction*
- *RV32M Integer Multiplication and Division Instruction Set Extension*
- *RV32F Single Precision Floating Point Extension* (opzionale)

- *PULP specific extensions (Hw loops, MAC, ALU extensions...)*

Il RI5CY supporta la sintesi ASIC. L'intero design è completamente sincrono e fa uso di *positive-edge triggered flip-flops* eccezion fatta per il *register file*, che può essere implementato sia tramite *latches* sia tramite flip-flops. La sintesi FPGA è supportata per il RI5CY solamente quando il *register file* viene implementato tramite flip-flops. Questo perché i latches non sono ben supportati sulle FPGA; è allora fondamentale selezionale il *register file* basato su flip-flops.

Viene ora analizzato un altro aspetto importante del processore: il protocollo utilizzato per la comunicazione con la memoria. Per questo scopo introduciamo la tabelladei segnali che vengono usati dalla LSU[23] durante questa fase:

Segnale	**Direzione**	**Descrizione**
data_req_o	*output*	*Request ready, questo segnale deve rimanere 'alto' finché data_gnt_i non va 'alto' per un intero ciclo di clock.*
data_addr_o [31:0]	*output*	*Address*
data_we_o	*output*	*Write Enable, deve rimanere 'alto' per le scritture e andare a livello logico 'basso' per le letture. Inviato insieme a data_req_o.*
data_be_o [3:0]	*output*	*Byte Enable. È attivo per i byte da leggere/scrivere, inviato insieme a data_req_o.*
data_wdata_o [31:0]	*output*	*Dati che devono essere scritti in memoria, segnale inviato insieme a data_req_o.*
data_rdata_i [31:0]	*input*	*Dati da leggere dalla memoria.*
data_rvalid_i	*input*	*data_rdata_i contiene dati validi quando il segnale data_rvalid_i è a livello logico 'alto'. Questo segnale rimane attivo per un ciclo di clock a richiesta.*
data_gnt_i	*input*	*Segnale che indica che l'altro lato ha accettato la richiesta. Il segnale data_addr_o può cambiare il prossimo ciclo di clock.*

Tabella 3.1: *Tabella che elenca e descrive i segnali usati dalla LSU*

Il protocollo utilizzato dalla LSU per comunicare con una memoria funzione ne modo seguente.

Inizialmente la LSU fornisce alla memoria un indirizzo valido sul bus ‘data_addr_o' ed imposta, contemporaneamente, il segnale ‘data_req_o' a livello logico alto. La memoria, di conseguenza, risponde con un segnale ‘data_gnt_i', anch'esso a livello alto, non appena è pronta per servire la richiesta inoltrata dal processore. Ciò può accadere nello stesso ciclo di clock in cui è stata inviata la richiesta, oppure in un numeroqualsiasi di cicli successivi. Una volta ricevuta la risposta, l'indirizzo presente sul bus può essere modificato dalla LSU (già il ciclo di clock successivo).

A questo punto i segnali ‘data_wdata_o' , ‘data_we_o' e ‘data_be_o' possono essere modificati tranquillamente, poiché si presume che la memoria abbia già elaborato e memorizzato tali informazioni. Questo valeva per il processo di scrittura. Per la fase di lettura il procedimento è simile. Dopo aver inviato il segnale di risposta, tramite il segnale attivo alto ‘data_gnt-i', la memoria risponde ulteriormente con il segnale ‘data_rvalid_i', impostato a livello logico alto, se i valori presenti sul bus ‘data_rdata_i' sono validi. Ciò può accadere dopo uno o più cicli di clock dopo l'aver ricevuto il segnale ‘data_gnt_i'. Si noti che ‘data_rvalid_i' deve essere impostato a livello logico alto anche quando è stata eseguita un'operazione di scrittura, sebbene i valori presenti sul bus ‘data_rdata_i' non abbiano alcun significato in questo caso. La figura seguente servirà a riassumere graficamente quanto detto fino ad ora.

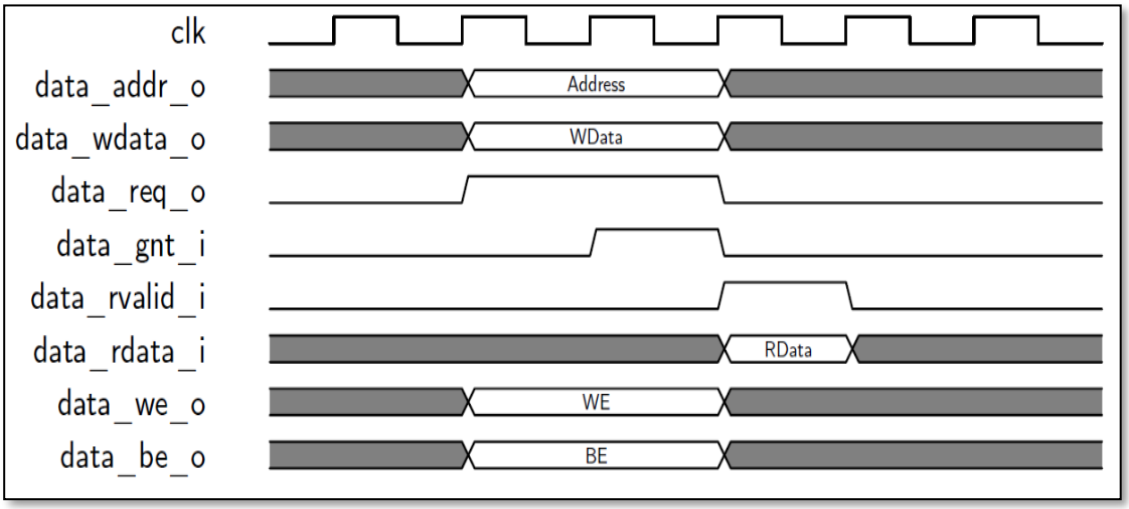

Figura 3.8: *Segnali di temporizzazione per le operazioni di ‘load' e di ‘store'.*

3.5 Interfaccia tra microprocessore e bus I3C

Come accennato precedentemente, il circuito Master_I3C è stato concepito per essere posto tra il microprocessore RI5CY ed un bus sul quale è utilizzato il protocollo I3C. Così facendo viene reso possibile il dialogo tra il RI5CY e le periferiche cheutilizzano questo tipo di protocollo per lo scambio di informazioni, che altrimenti sarebbeimpossibile.

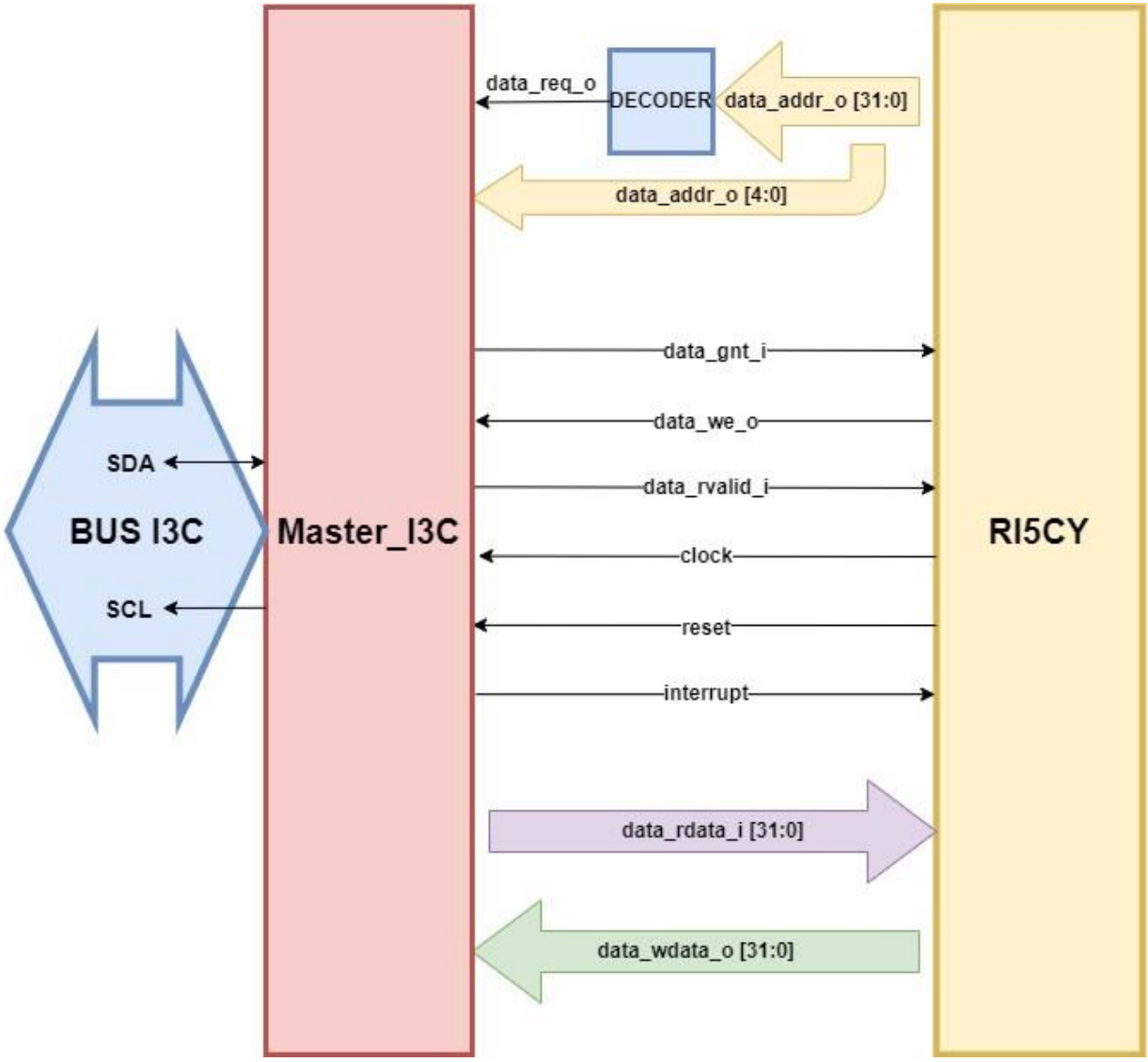

***Figura 3.9**: schema delle connessioni del Master_ I3C.*

Capitolo 4

Progetto a livello Register Transfer del circuito Master I3C

Questo capitolo è dedicato alla progettazione circuitale del Master I3C, verrà descritta la struttura di ogni sotto-modulo e tutte le funzionalità che sono state implementate nel lavoro di tesi.

4.1 Introduzione al Master I3C

Il dispositivo Master I3C implementato in questo lavoro di tesi è stato realizzato con l'obiettivo di potersi interfacciare ad una versione personalizzata del processore RISC-V, il TARDIS. Il modulo realizzato deve essere in grado di gestire la comunicazione tra il processore e il bus seriale I3C. Nel **capitolo 2** è stato descritto il protocollo I3C, con particolare attenzione alla possibilità di interfacciarsi con dispositivi di natura diversa. A fronte di questa considerazione il Master I3C deve potersi adattare al tipo di periferica con la quale comunica e garantire una corretta "coesistenza" di molteplici dispositivi. Per questo motivo il modulo è stato progettato affinché:

- Sia configurabile dal microprocessore in modo da poter comunicare sia in modalità Open Drain, con frequenza massima di 1 [Mbit/s], sia in modalità Push Pull, con frequenza massima di 12,5 [Mbit/s].
- Garantisca completa retrocompatibilità con dispositivi I^2C.
- Sia in grado di effettuare trasmissioni e ricezioni di un numero arbitrario di byte, sia in modalità privata che in modalità Command Code.
- Sia in grado di riconoscere e gestire autonomamente il Command Code ricevuto dal processore.
- Sia in grado di gestire l'assegnazione degli indirizzi dinamici, salvando in registri interni le informazioni importanti di ogni Slave.

L'obiettivo di questo lavoro di tesi è permettere al processore RI5CY di utilizzare il Master_I3C come una sorta di periferica, caricandovi le operazioni che dovranno essere eseguite senza dover gestire nessun aspetto comunicativo con le periferiche. In questo modo il Master I3C lavorerà in autonomia, gestendo la comunicazione sul bus I3C senza coinvolgere ulteriormente il processore, che potrà svolgere altre operazioni.

4.2 Ingressi e uscite del modulo *Master I3C*

La **figura 4.1** illustra i segnali di ingresso e uscita utilizzati dal Master I3C per comunicare con il microprocessore RI5CY e con il bus I3C.

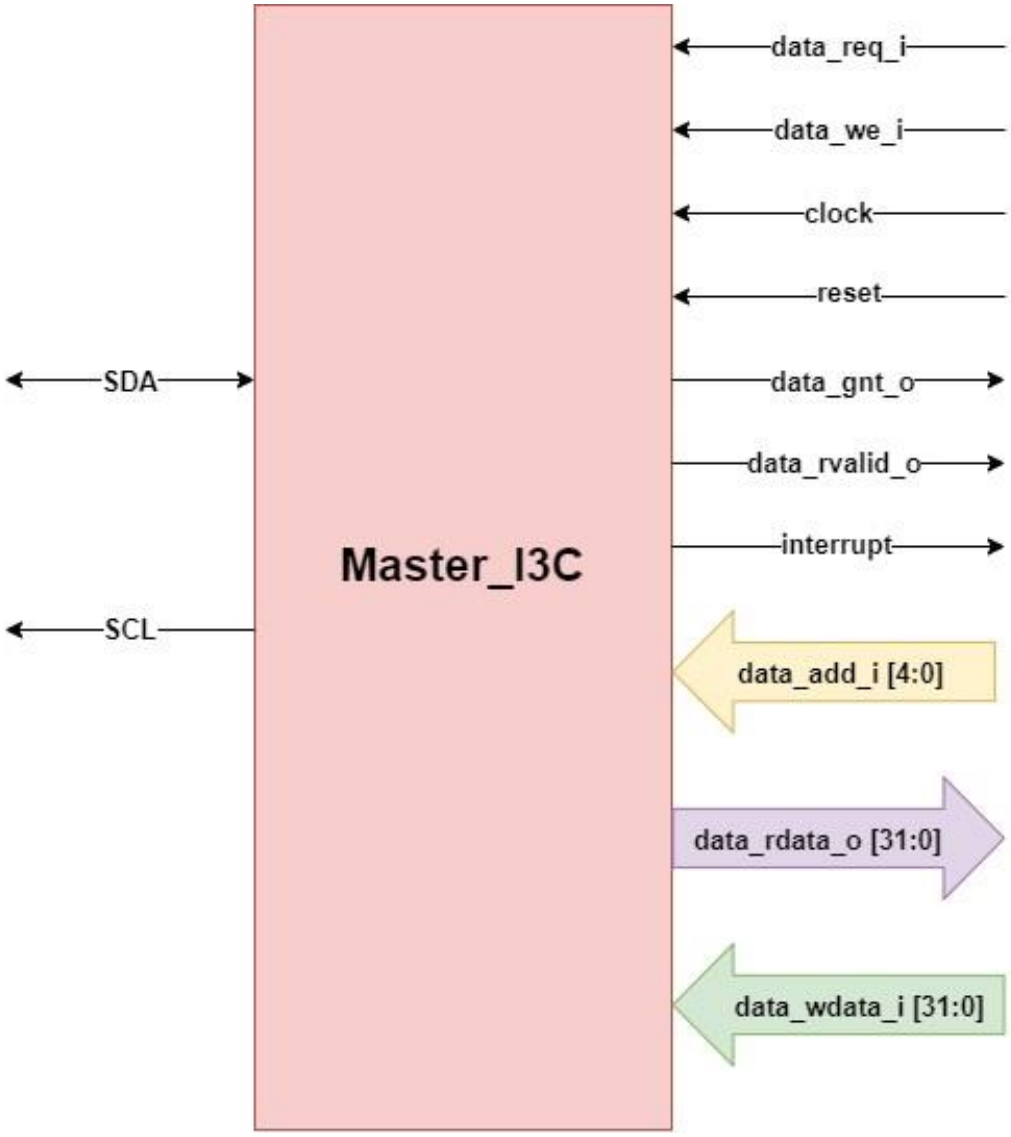

Figura 4.1: *Ingressi e uscite del modulo Master_I3C*

- ***data_addr_i [4:0]*** sono i cinque bit meno significativi dei trentadue del bus *data_addr_i [31:0]* e selezionano i registri presenti all'interno del modulo *Registers & memory manager.*
- ***data_wdata_i [31:0]*** è un bus sul quale il microprocessore invia al Master_I3C i dati da scrivere al suo interno.
- ***data_rdata_o [31:0]*** è un bus utilizzato per trasferire i dati che dovranno essere letti dal RI5CY.
- ***clock*** è il segnale di sincronizzazione fornito dal TARDIS. Il Master_I3C è stato progettato per lavorare sul fronte di salita del segnale di clock.
- ***reset*** è il segnale di azzeramento ed è fornito dal TARDIS.
- ***interrupt*** è una linea che ha lo scopo di segnalare al microprocessore l'avvenimento di eventi particolari all'interno del modulo I3C.
- ***data_req_i*** viene utilizzata per segnalare l'intenzione, da parte del microprcessore, di effettuare un'operazione di lettura o scrittura sulla periferica.
- ***data_we_i*** viene usata insieme al segnale *data_req_i* e specifica l'operazione che il micro vuole eseguire sulla periferica: '**1**' indica un'operazione di scrittura,'**0**' un'operazione di lettura.
- ***data_gnt_o*** viene generato dal Master_I3C ogniqualvolta sia pronto a gestire una richiesta di servizio avanzata dal microprocessore.
- ***data_rvalid_o*** è una linea utilizzata per comunicare al micro che i dati forniti in uscita dopo un'operazione di lettura sono validi.
- ***SDA*** *(Serial data)* è la linea bidirezionale utilizzata per lo scambio dati con le periferiche presenti sul bus I3C.
- ***SCL*** *(Serial clock)* è la linea attraverso la quale il Master fornisce il segnale di sincronizzazione per la comunicazione con le periferiche.

4.3 Schema a blocchi del *Master_I3C*

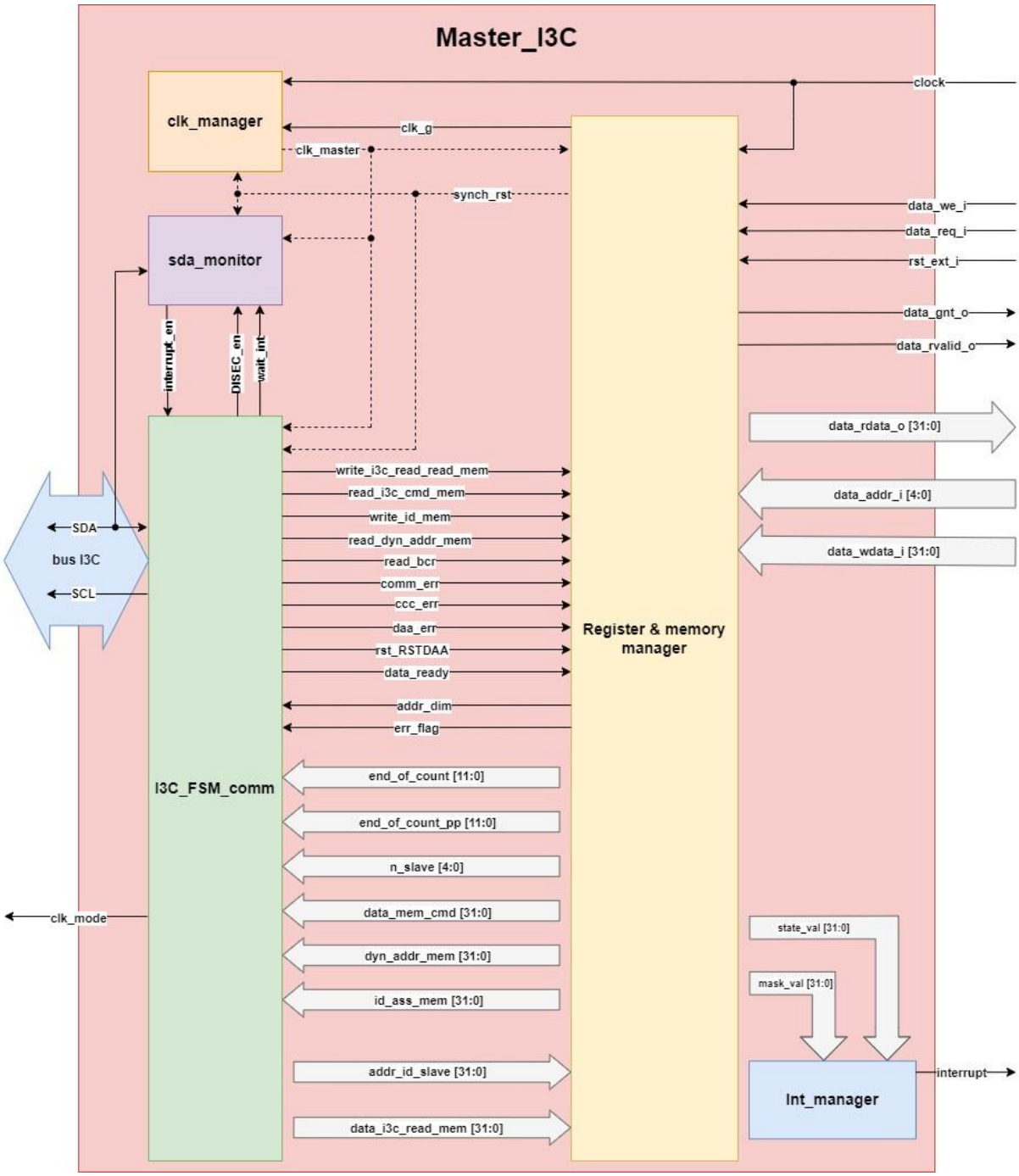

***Figura 4.2**: Schema a blocchi del modulo Master_I3C*

In **figura 4.2** è rappresentato lo schema a blocchi interno al Master_I3C. il circuito contiene 5 sotto-circuiti, ognuno dei quali ha un ruolo specifico per una corretta comunicazione:

- ***Int_manager*** ha lo scopo di generare il segnale di interrupt da inviare al RI5CY.
- ***I3C_FSM_comm*** è la macchina a stati finiti che si occupa della trasmissione e ricezione dei dati sul bus I3C e della generazione del segnale di sincronizzazione sulla linea SCL.
- ***clk_manager*** è il modulo di clock gating.
- ***Register & memory manager*** è l'unico modulo che comunica con il processore ed è dedicato a gestire tutti gli elementi di memoria del circuito.
- ***sda_monitor*** è il modulo dedicato a monitorare la linea SDA per la gestione degli eventi di interruzione.

4.4 Modulo *clk_manager*

Il modulo clk_*manager* ha il compito di generare il segnale di temporizzazione utilizzato all'interno del Master_I3C. Attraverso una cella di clock gating fornisce a tutti i moduli del *Master_I3C* il segnale di temporizzazione. Il clock gating è una tecnica utilizzata nella progettazione dei circuiti digitali per ridurre il consumo di potenza dei circuiti integrati. Nei circuiti digitali, il clock è il segnale che regola il tempo e sincronizza le operazioni dei vari componenti del circuito. Quando un circuito è attivo, il clock è costantemente in funzione, anche se alcune parti del circuito potrebbero non essere utilizzate in determinati momenti. Questo comporta uno spreco di energia.

Il clock gating risolve questo problema introducendo un meccanismo che spegne il clock per le parti del circuito che non sono in uso, riducendo così il consumo di potenza. Questo viene fatto aggiungendo un elemento logico che controlla il segnale del clock in modo che venga disattivato quando non è necessario, ad esempio quando il modulo del circuito associato non deve eseguire alcuna operazione. Nel nostro caso quando il processore non ha necessità di comunicare con il bus I3C è possibile disattivare il clock interno al modulo Master_I3C.

4.4.1 Segnali di ingresso e uscita

***Figura 4.3**: segnali di ingresso e uscita di clk_manager*

- ***clk*** è il segnale di clock di sistema proveniente dal microprocessore.
- ***clk_g_i*** è un bit che viene impostato all'interno del modulo *Register & memory manager* e ha lo scopo di abilitare o meno la cella di clock gating. Se *clk_g_i* è uguale a '1' allora il *clk* viene passato in uscita al segnale *clk_master_o,* se *clk_g_i* è uguale a '0' allora clk_master_o viene impostato a '0', disabilitato il clock interno al modulo Master_I3C.
- ***rst_n_i*** è il bit di reset sincrono proveniente da *Register & memory manager*, ha lo scopo di resettare *clk_manager*.
- ***clk_master_o*** è il clock interno al modulo Master_I3C.

4.5 Modulo *Int_manager*

Il modulo Int_manager ha lo scopo di generare il segnale di interrupt da inviare al RI5CY. Le sorgenti di interrupt sono legate a possibili errori di comunicazione sul bus, alle capacità delle memorie interne al modulo *Register & memory manager* e alla presenza di nuovi dati in memoria. L'interfaccia con il RI5CY prevede una sola linea di uscita; quindi, si è scelto di utilizzare un registro a 32 bit chiamato *state_register* che racchiude tutte le possibili cause del segnale di interrupt. Per risalire alle singole cause il microprocessore deve leggere il contenuto di *state_register* con la possibilità di disabilitare una o più fonti di possibili interrupt utilizzando il registro *mask_register*. Il "mascheramento" delle fonti di interrupt avviene attraverso un'operazione di *AND* bit a bit tra i due registri. dettagli sulla struttura dello *state_register* verranno spiegati nel capitolo **4.8.8.3.**

4.5.1 Segnali di ingresso e uscita

***Figura 4.4**: segnali di ingresso e uscita di Int_manager*

- ***status_reg_i [31:0]*** è un bus tramite il quale viene inviata al modulo una copia del contenuto del registro *state_register*.
- ***mask_reg_i [31:0]*** è un bus tramite il quale viene inviata al modulo una copia del contenuto del registro *mask_register*.
- ***interrupt*** è la linea che segnala al RI5CY l'avvenimento di eventi particolari.

4.6 Modulo *sda_monitor*

Il modulo sda_monitor ha lo scopo di monitorare la linea SDA nel caso ci fossero eventi di interruzione come In-Band Interrupt o Hot Join. Se le condizioni lo permettono (vedi **capitolo 2.9**) e uno Slave richiede l'attenzione del Master_I3C, ad esempio attraverso un evento IBI allora il modulo *sda_monitor* comunica al modulo *I3C_FSM_comm* la richiesta. Sarà poi compito del modulo *I3C_FSM_comm* fornire il segnale di sincronizzazione sulla linea SCL e prepararsi a ricevere i dati.

4.6.1 Segnali di ingresso e uscita

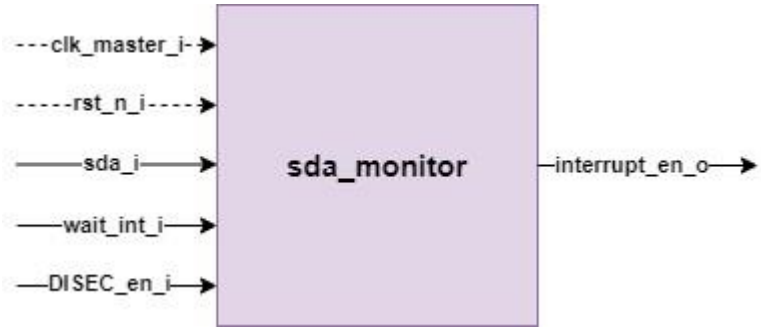

***Figura 4.5**: Segnali di ingresso e uscita di sda_monitor*

- ***clk_master_i*** è il segnale di sincronizzazione in uscita da *clk_manager*.
- ***rst_n_i*** è il reset sincronizzato in uscita da *Register & memory manager*.
- ***sda_i*** è la linea di SDA in ingresso.
- ***wait_int_i*** è un segnale in uscita da *I3C_FSM_comm* e ha il compito di avvertire che il bus è in uno stato di inattività da sufficiente tempo ed il *Master_I3C* è pronto a riceve eventuali eventi di interruzione. In particolare, se la macchina a stati è in uno stato di *IDLE* (con SDA e SCL a livello logico alto) da più di 5 microsecondi il segnale wait_int_i è impostato a '1', in tutti gli altri casi ha valore '0'.
- ***DISEC_en_i*** è un segnale in uscita da *I3C_FSM_comm* e ha il compito di disabilitare gli eventi di interruzione. Il microprocessore attraverso il Command Code DISEC può disabilitare gli eventi di interruzione impostando il segnale *DISEC_en_i* a '1', in tutti gli altri casi ha valore '0'.
- ***Interrupt_en_o*** è un segnale diretto al modulo *I3C_FSM_comm* e ha il compito di preparare il Master_I3C a ricevere l'indirizzo inviato dallo Slave che ha fatto richiesta e quindi di fornire il segnale di sincronizzazione sulla linea SCL. In particolare, se *DISEC_en_i*, *wait_int_i* sono uguale a '1' e la linea *sda_i* passa da livello logico alto a livello logico basso allora *interrupt_en_o* è impostato a '1', in tutti gli altri casi ha valore '0'.

4.7 Modulo I3C_FSM_comm

Il modulo *I3C_FSM_comm* è la macchina a stati che ha il compito di comunicare con il bus I3C. Si occupa della trasmissione dei dati contenuti nelle memorie dinamiche *I3C_cmd_mem* e *write_dyn_addr_mem*; nella comunicazione con la memoria statica *addr_id_mem*; nella scrittura dei dati ricevuti da una periferica nelle memorie dinamiche *I3C_read_mem* e *read_id_mem*. Inoltre, il modulo *I3C_FSM_comm* è responsabile della generazione del segnale SCL e del passaggio dalla configurazione Open Drain a quella Push Pull. Infine gestisce la scrittura di alcuni campi del registro *state_register: data_ready, comm_err, ccc_err* e *daa_err*.

La struttura della *I3C_FSM_comm* è quella di una macchina a stati finiti di Moore; quindi, le uscite sono in funzione dei soli stati correnti.

4.7.1 Segnali di ingresso e uscita

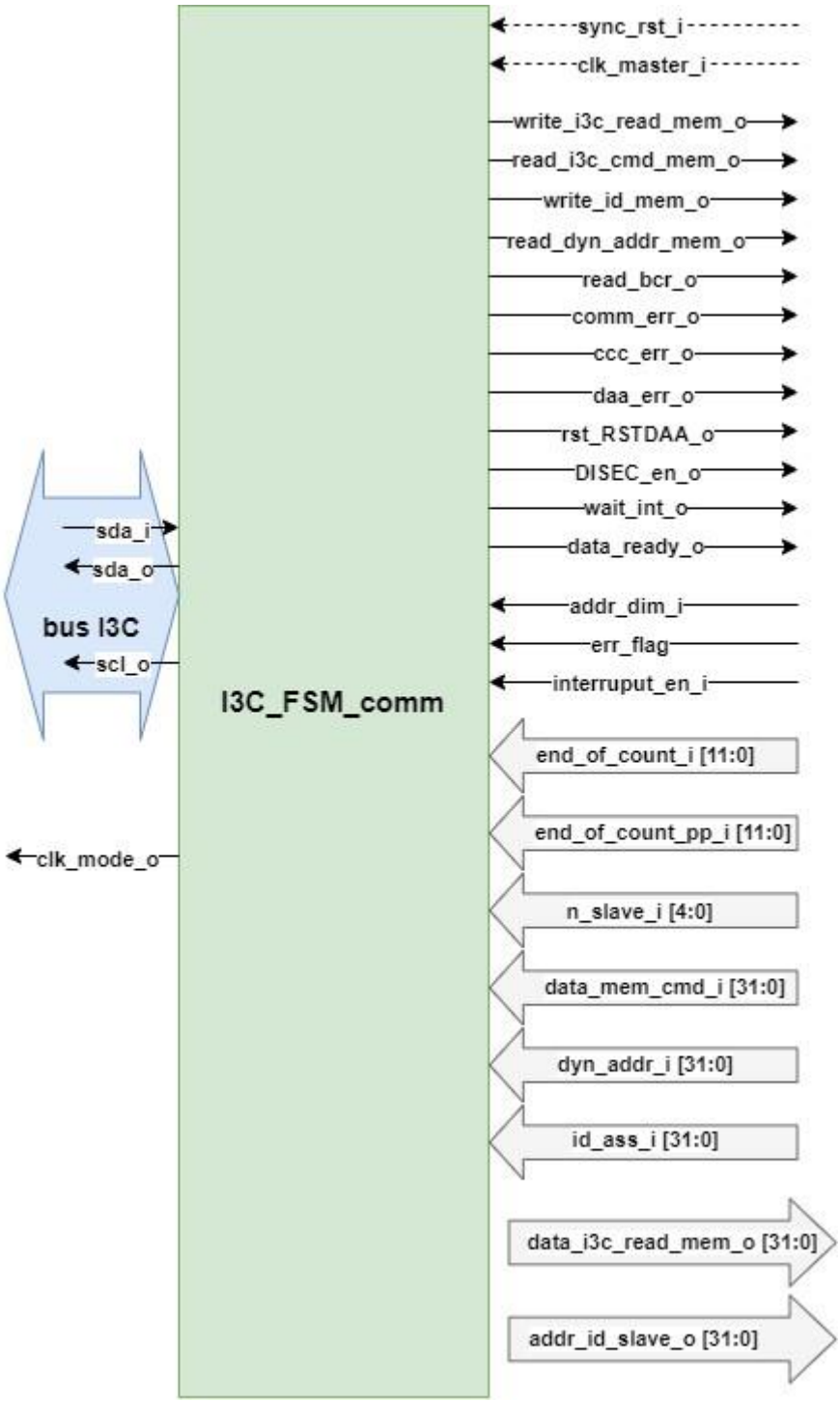

***Figura 4.6**: Segnali di ingresso e uscita di I3C_FSM_comm*

- ***sync_rst_i*** è il reset sincronizzato in uscita da *Register & memory manager*.
- ***clk_master_i*** è il segnale di sincronizzazione in uscita da *clk_manager*.
- ***addr_dim_i*** è un bit attraverso il quale viene specificata la dimensione dell'indirizzo I^2C della periferica. Il bit ha valore '0' se la periferica ha indirizzo a 10 bit, altrimenti ha valore '1' se la periferica ha indirizzo a 7 bit. Si precisa che le periferiche I3C non supportano indirizzi a 10 bit; quindi, questo bit è utilizzato solo per comunicare con periferiche I^2C.
- ***err_flag*** è un bit in uscita dal modulo *register_file* e ha il compito di interrompere la comunicazione sul bus nel caso si verificasse un errore.
- ***sda_i*** è la linea SDA in ingresso a *I3C_FSM_comm.*
- ***interrupt_en_i*** è un bit in uscita dal modulo *sda_monitor* e serve ad avvertire la presenza di un evento di interruzione.
- ***end_of_count_i [11:0]*** bus contenente l'informazione sul rapporto tra la frequenza del clock di sistema e la frequenza sul bus I3C in configurazione Open Drain.
- ***end_of_count_pp_i [11:0]*** ha lo stesso compito di *end_of_count_i [11:0]* ma per la configurazione Push-Pull.
- ***n_slave_i [4:0]*** è un bus contenente il numero di periferiche connesse al bus che necessitano l'assegnazione di un indirizzo dinamico.
- ***data_mem_cmd_i [31:0]*** bus tramite il quale vengono ricevuti dati immagazzinati nella memoria *I3C_cmd_mem.*
- ***dyn_addr_i [31:0]*** bus tramite il quale vengono ricevuti i dati immagazzinati nella memoria *write_dyn_addr_mem.*
- ***id_ass_i [31:0]*** bus tramite il quale vengono letti i dati relativi alle periferiche a cui è stato assegnato indirizzo dinamico. È un segnale in uscita della memoria statica *addr_id_mem* che ha il compito di associare a ogni indirizzo dinamico le impostazioni ricevute dalla periferica.
- ***data_i3c_read_mem_o [31:0]*** bus tramite il quale la macchina a stati invia i dati ricevuti da una periferica alla memoria *I3C_read_mem.*
- ***addr_id_slave_o [31:0]*** bus tramite il quale la macchina a stati invia le informazioni relative a una periferica, ovvero i registri ID, DCR e BCR, alla memoria statica *addr_id_mem* e alla memoria dinamica *read_id_mem.*
- ***sda_o*** linea SDA in uscita da *I3C_FSM_comm.*
- ***scl_o*** linea SCL in uscita da *I3C_FSM_comm.*

- ***write_i3c_read_mem_o*** segnale di abilitazione per la scrittura dei dati, diretto alla memoria *I3C_read_mem.* Viene inviato ogni volta che *I3C_FSM_comm* deve scrivere un nuovo dato ricevuto sul bus I3C.
- ***read_i3c_cmd_mem_o*** segnale di abilitazione alla lettura dei dati, diretto alla memoria *I3C_cmd_mem*.
- ***write_id_mem_o*** segnale analogo a *write_i3c_read_mem_o,* ma rivolto alla memoria *read_id_mem.* Viene inviato ogni volta che una periferica invia i propri dati (ID, DCR, BCR) per il processo di assegnazione degli indirizzi dinamici.
- ***read_dyn_addr_mem_o*** segnale analogo a *read_i3c_cmd_mem_o,* ma rivolto alla memoria *write_dyn_addr_mem.* Viene inviato ogni volta che è necessario trasmettere a una periferica il suo indirizzo dinamico.
- ***read_bcr_o*** è un segnale diretto alla memoria statica *addr_id_mem.* Viene inviato ogni volta che è necessario leggere i dati contenuti nel registro BCR di una specifica periferica.
- ***comm_err_o*** è un segnale diretto al modulo *register_file* con lo scopo di comunicare una mancato ACK durante una comunicazione.
- ***ccc_err_o*** è un segnale diretto al modulo *register_file* con lo scopo di comunicare che il Command Code caricato non è valido o non è supportato.
- ***daa_err_o*** è un segnale diretto al modulo *register_file* con lo scopo di comunicare un mancato ACK durante il processo di assegnazione degli indirizzi dinamici, durante il Command Code ENTDAA.
- ***rst_RSTDAA_o*** è un segnale diretto alla memoria statica *addr_id_mem* e alle memorie dinamiche *read_id_mem, write_dyn_addr_mem.* Viene inviato ogni volta che il microprocessore, attraverso il Command Code RSTDAA, vuole resettare tutti gli indirizzi dinamici.
- ***DISEC_en_o*** è un segnale diretto al modulo *sda_monitor* e viene utilizzato ogni volta che il microprocessore, attraverso il Command Code DISEC, vuole disabilitare gli eventi di interruzione sul bus causati dalle periferiche.
- ***clk_mode_o*** è un segnale che ha il compito di tenere traccia della frequenza di comunicazione sul bus. In particolare, se ha valore ‘0’ la comunicazione è in configurazione Open Drain, altrimenti se ha valore ‘1’ è in configurazione Push-Pull.
- ***data_ready_o*** è un segnale che ha il compito di comunicare al RI5CY la presenza di nuovi dati in memoria *I3C_read_mem*

- ***wait_int_o*** è un segnale diretto al modulo *sda_monitor* per comunicare che è trascorso il tempo necessario per ricevere interruzioni sul bus da parte delle periferiche. Un contatore tiene traccia dei cicli di clock di sistema, raggiunto un determinato valore il bit *wait_int_o* viene impostato a '1', altrimenti ha valore '0'.

4.7.2 Diagrammi di flusso

Di seguito si propongono i diagrammi di flusso della macchina a stati finiti *I3C_FSM_comm.* Per via della complessità e per una più facile comprensione verrà descritto un diagramma di flusso per ognuna delle seguenti modalità: scrittura e lettura in modalità Legacy I^2C; scrittura e lettura in modalità privata SDR; scrittura broadcast in modalità CCC (Common Command Code); scrittura e lettura diretta in modalità CCC; operazione di assegnazione degli indirizzi dinamici (ENTDAA); evento di interruzione (In-Band Interrupt); evento Hot Join.

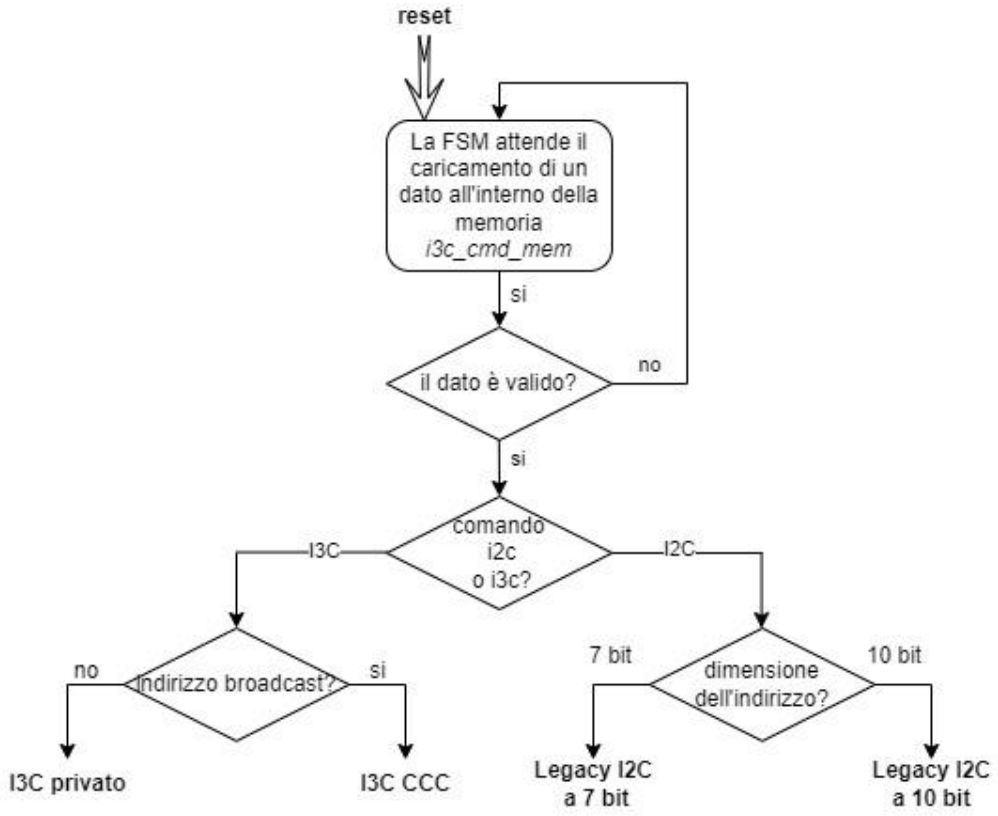

Figura 4.7*: configurazione comunicazione da Master a Slave*

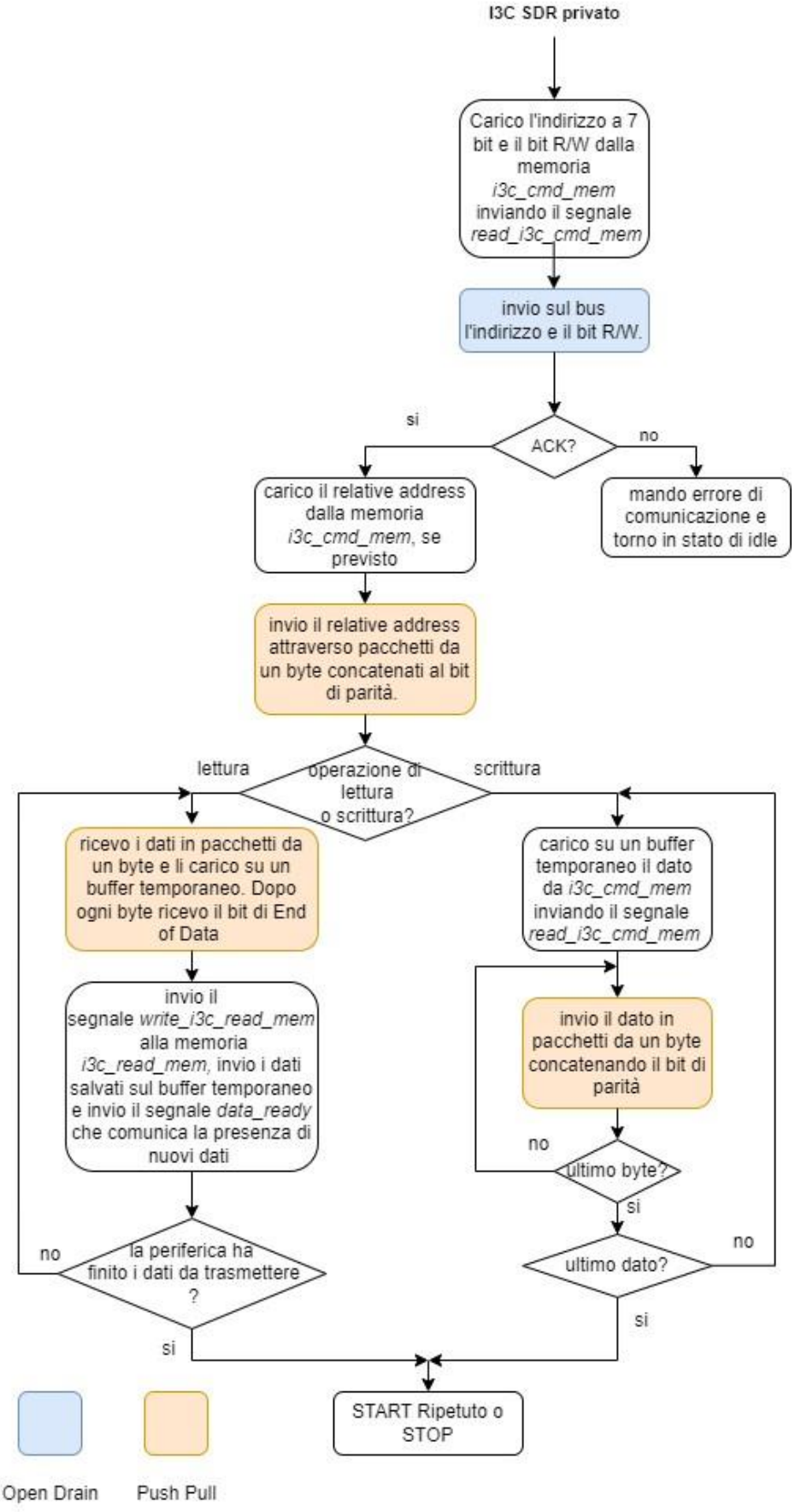

***Figura 4.8**: comunicazione I3C privata in lettura e scrittura*

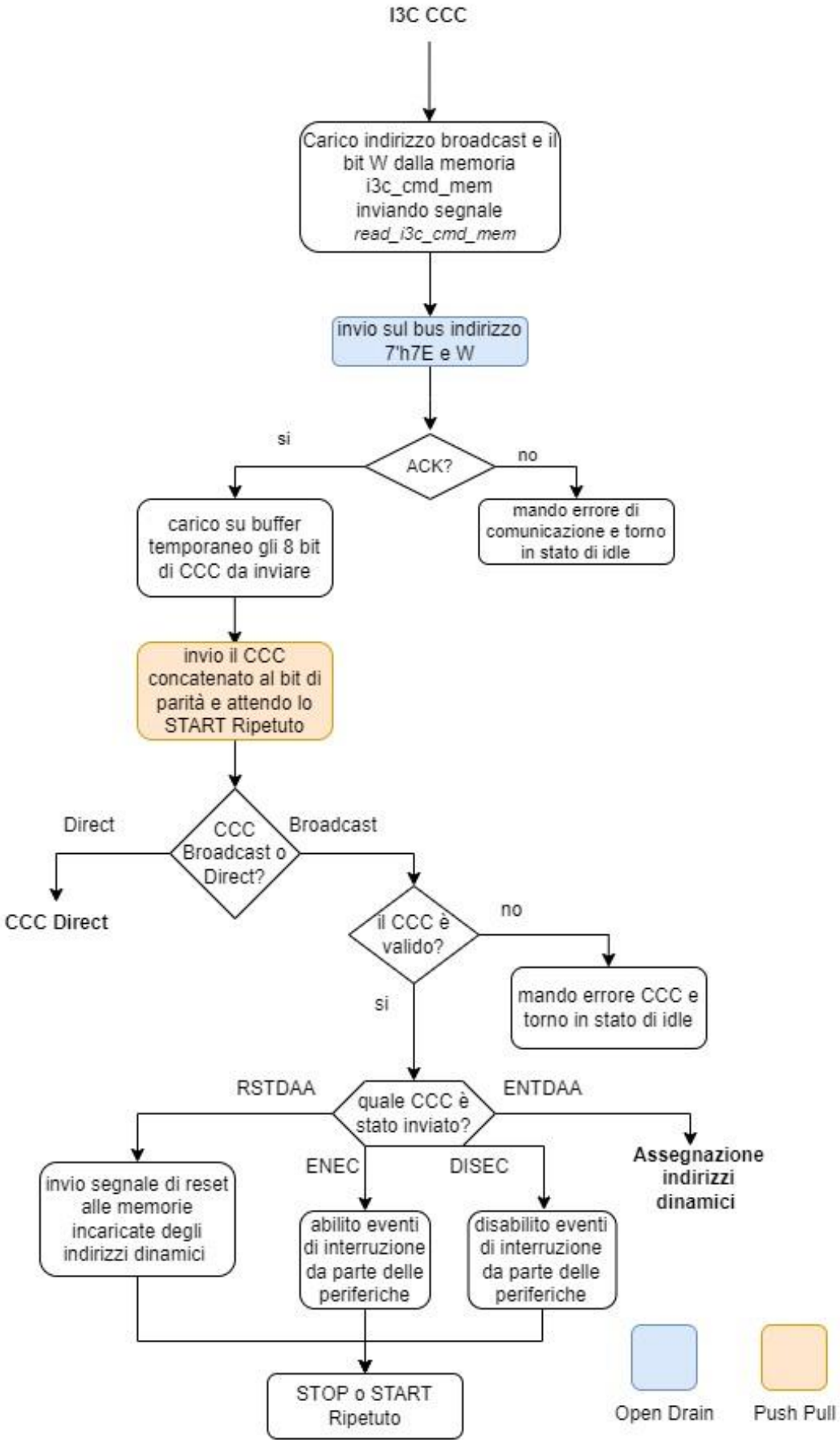

***Figura 4.9**: configurazione CCC e comunicazione CCC Broadcast*

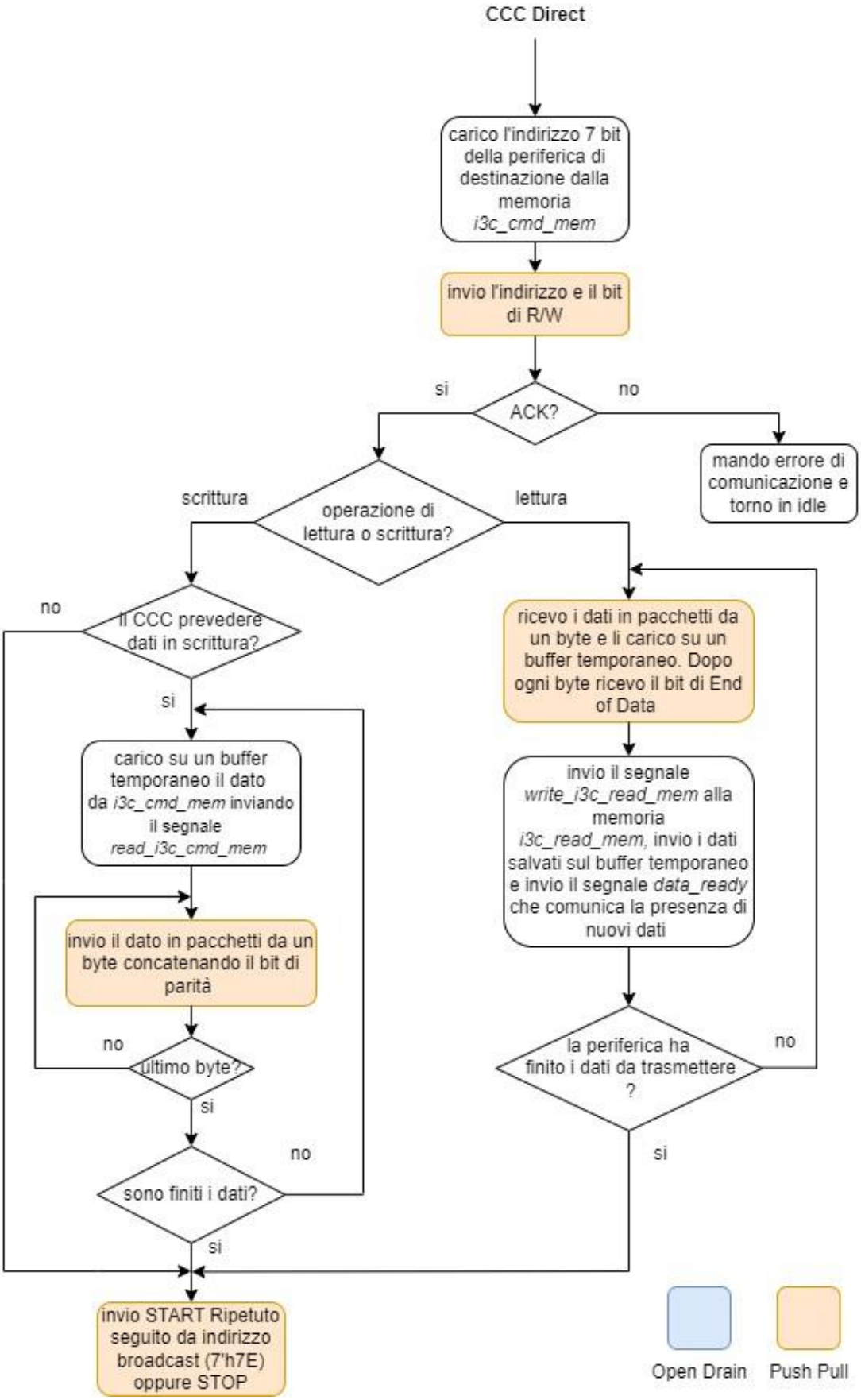

***Figura 4.10**: comunicazione CCC Direct in scrittura e lettura*

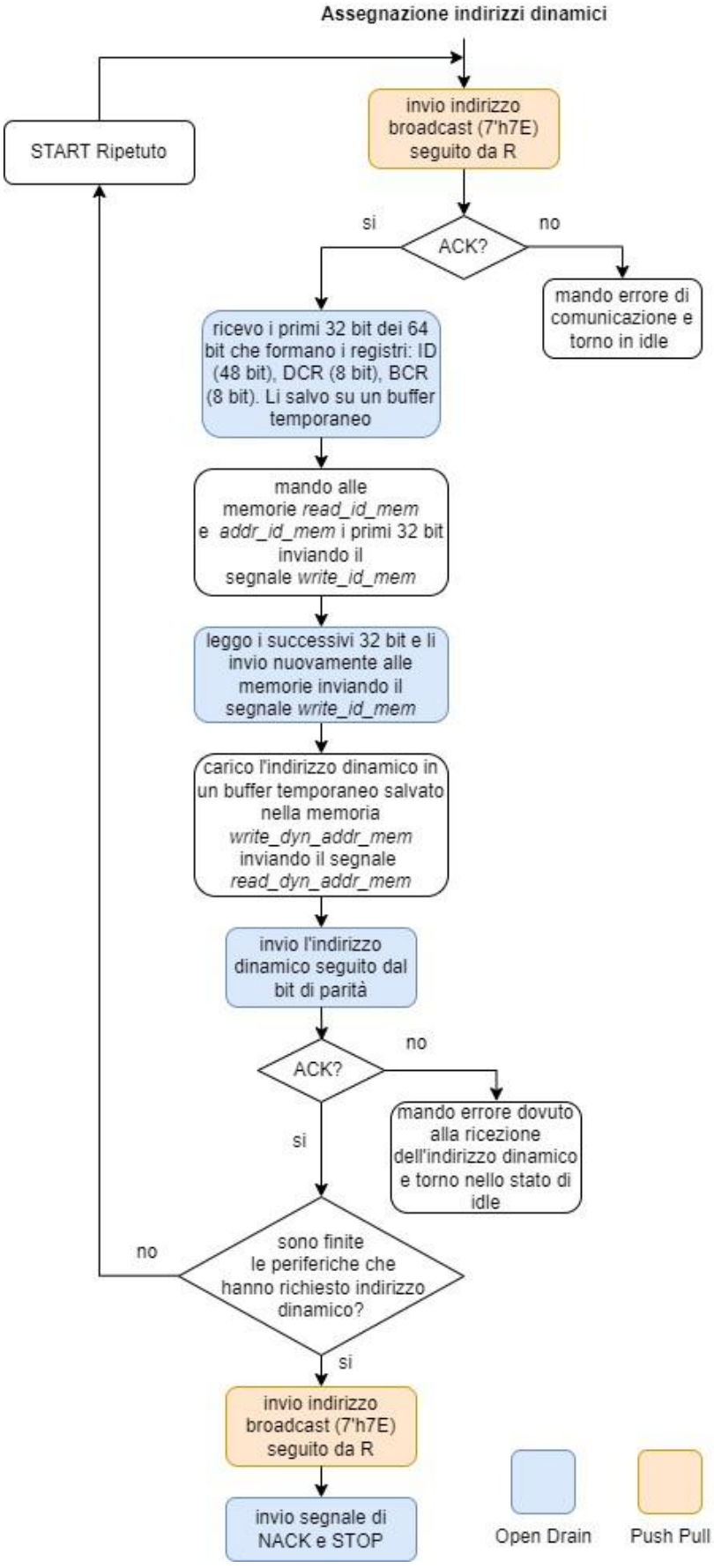

***Figura 4.11**: Assegnazione degli indirizzi dinamici*

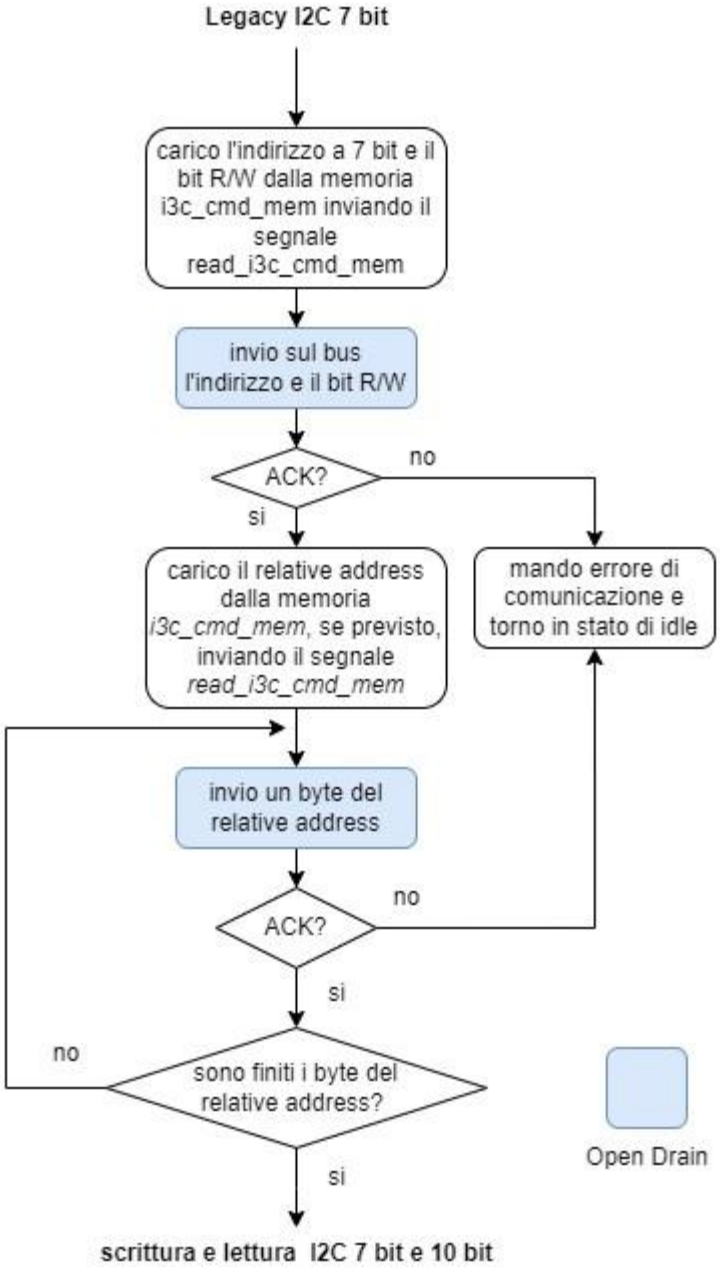

***Figura 4.12**: comunicazione I2C con indirizzo a 7 bit*

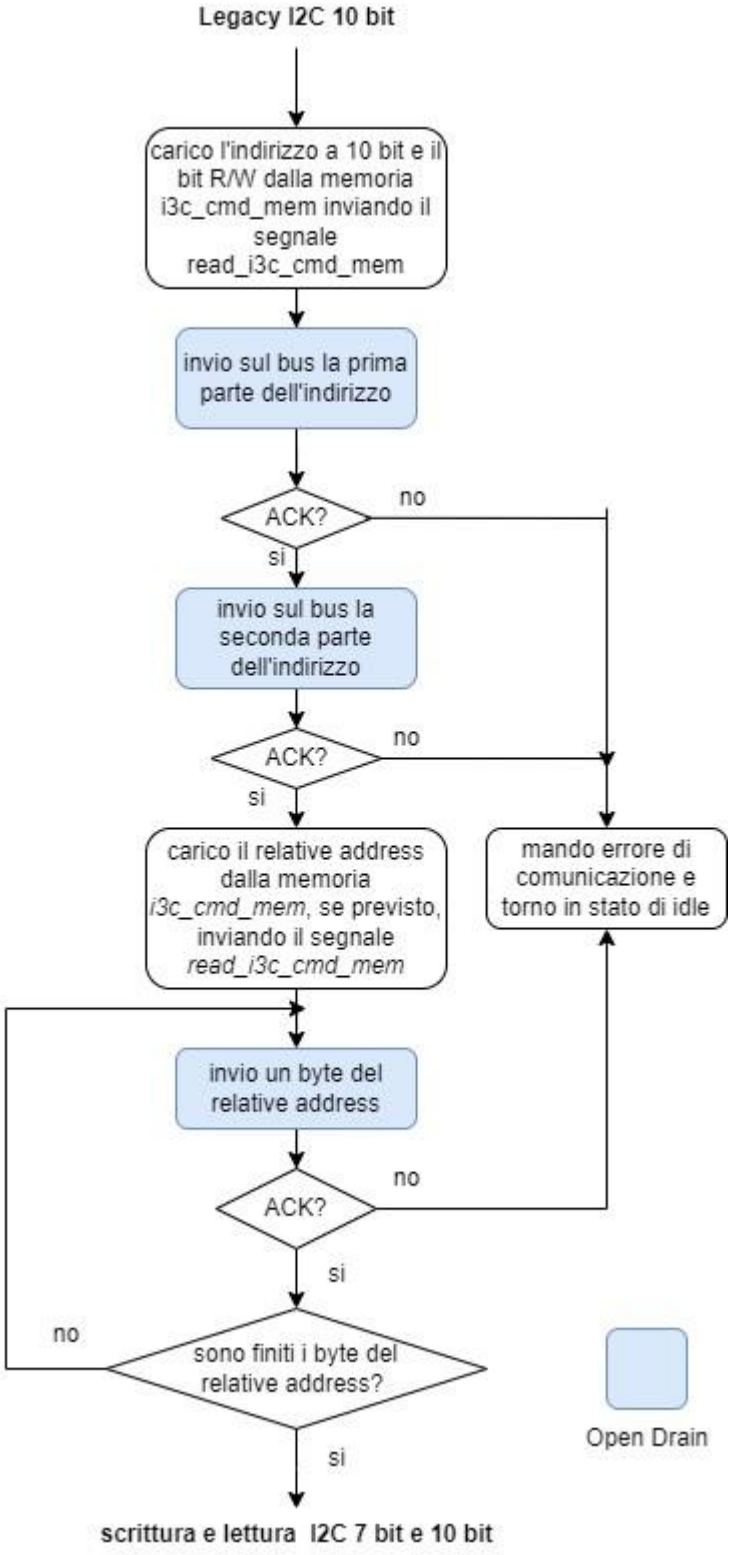

***Figura 4.14**: comunicazione I2C a 7 bit*

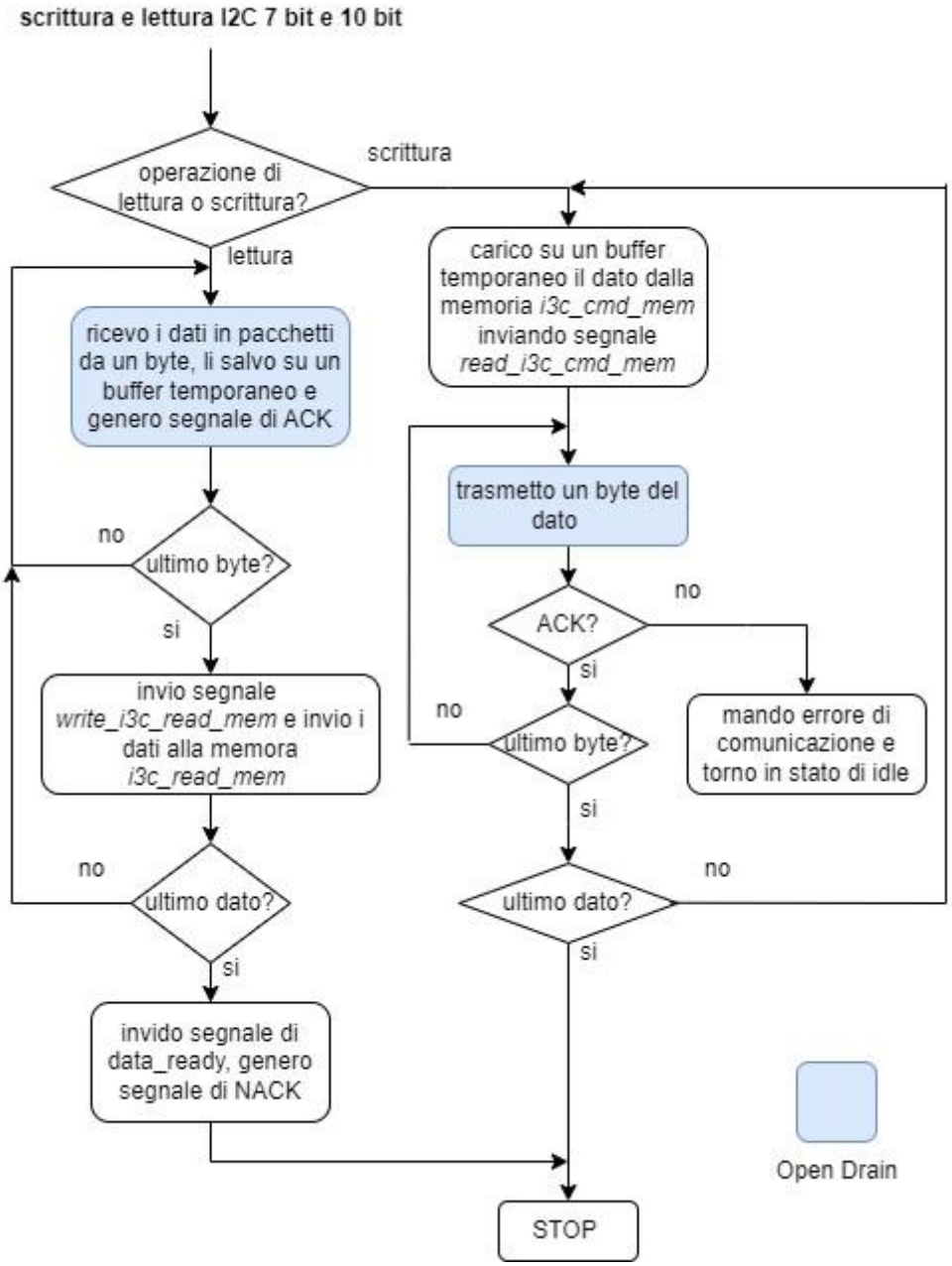

***Figura 4.15**: lettura e scrittura I2C a 7 bit e 10 bit*

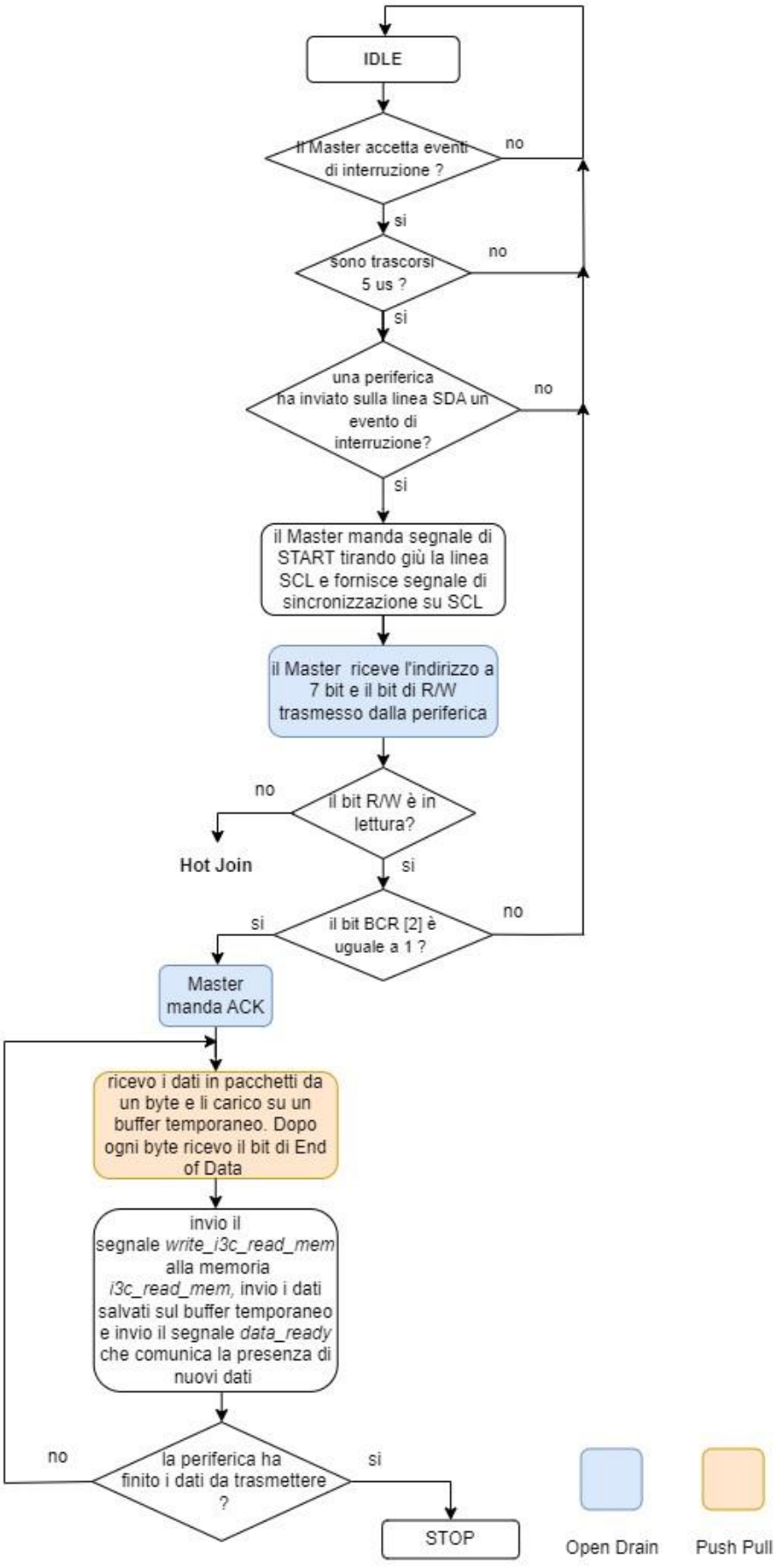

***Figura 4.16**: Evento di interruzione (IBI)*

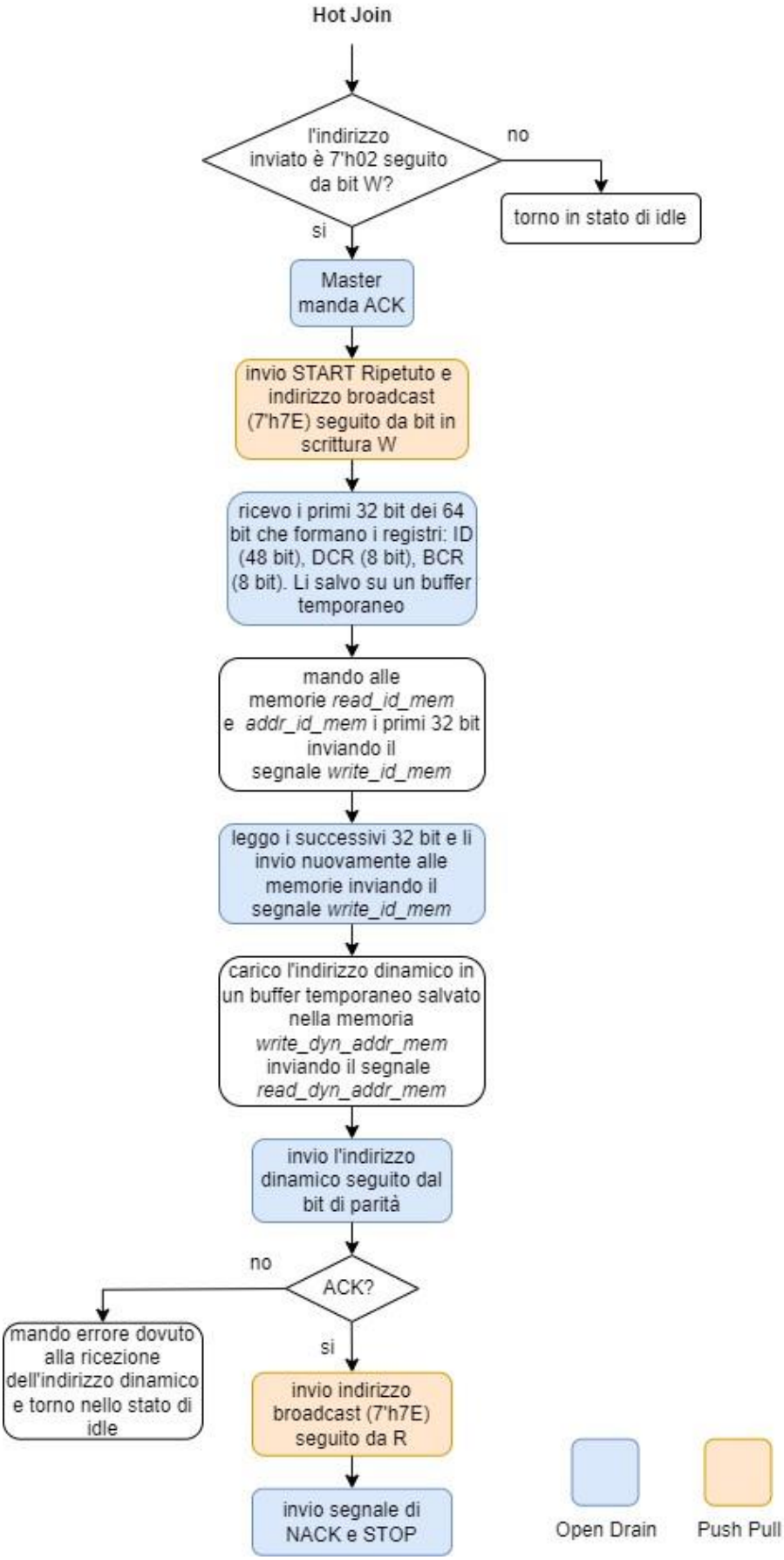

***Figura 4.17**: Evento Hot Join*

4.8 Il modulo *Register & memory manager*

Il modulo *Register & memory manager* è l'unico che comunica con il microprocessore. È tramite questo modulo che il RI5CY comunica con il Master_I3C e con il bus I3C attraverso un bus a 32 bit in lettura e uno in scrittura. tutte le operazioni che il Master_I3C deve eseguire vengono prima memorizzate dal modulo *Register & memory manager* e solo in un secondo momento vengono lette. All'interno di questo modulo sono presenti nove sotto moduli.

4.8.1 Segnali di ingresso e uscita

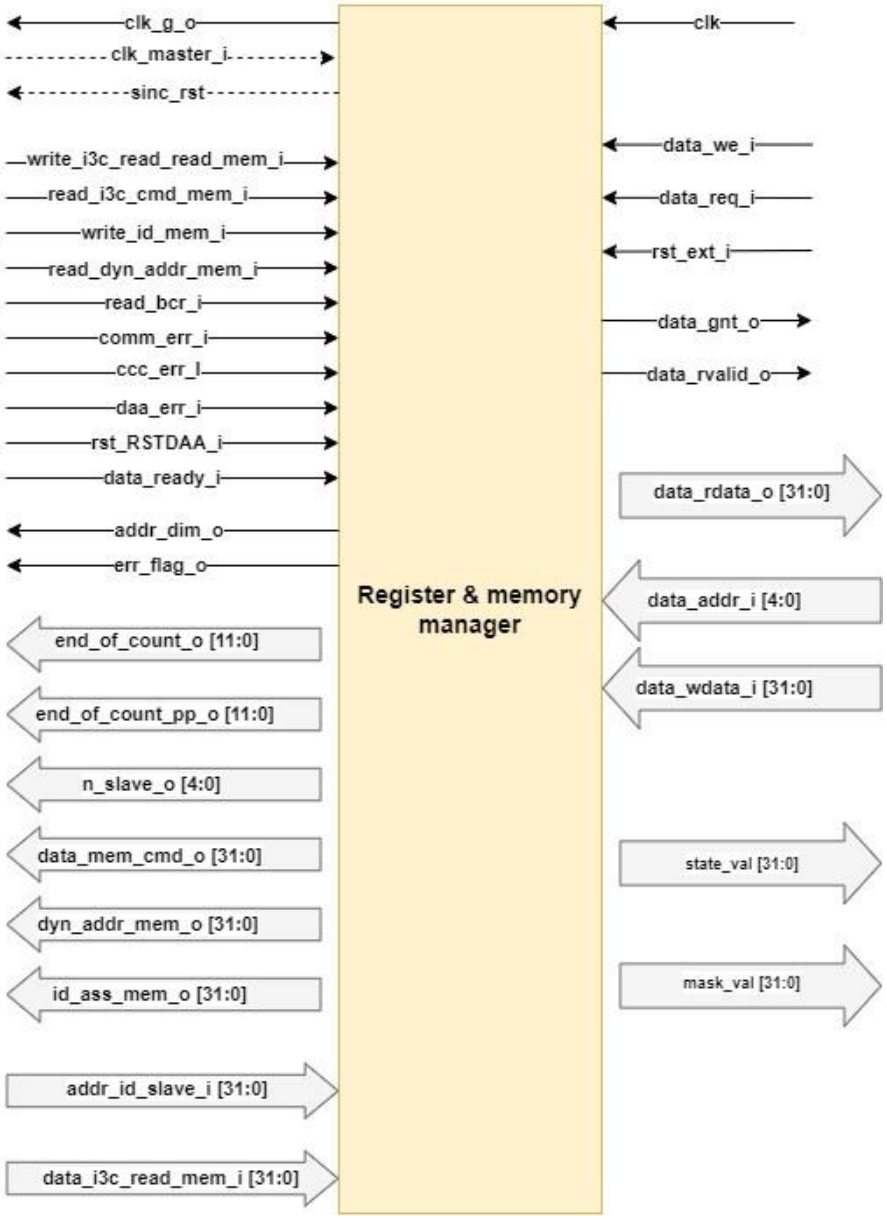

***Figura 4.18**: segnali di ingresso e uscita di Register & memory manager*

- ***sync_rst_i*** è il reset sincronizzato diretto alla macchina a stati *I3C_FSM_comm.*
- ***clk_master_i*** è il segnale di sincronizzazione in uscita da *clk_manager*.
- ***addr_dim_o*** è un bit attraverso il quale viene specificata la dimensione dell'indirizzo I^2C della periferica, diretto a *I3C_FSM_comm*. Il bit ha valore '0' se la periferica ha indirizzo a 10 bit, altrimenti ha valore '1' se la periferica ha indirizzo a 7 bit. Si precisa che le periferiche I3C non supportano indirizzi a 10 bit; quindi, questo bit è utilizzato solo per comunicare con periferiche I^2C.
- ***err_flag_o*** è un bit in uscita dal modulo *register_file*, diretto a *I3C_FSM_comm* e ha il compito di interrompere la comunicazione sul bus nel caso si verificasse un errore.
- ***end_of_count_o [11:0]*** bus diretto a *I3C_FSM_comm* contenente l'informazione sul rapporto tra la frequenza del clock di sistema e la frequenza sul bus I3C in configurazione Open Drain.
- ***end_of_count_pp_o [11:0]*** ha lo stesso compito di *end_of_count_o [11:0]* ma per la configurazione Push-Pull.
- ***n_slave_o [4:0]*** è un bus che salva l'informazione dal *configuration register* sul numero di periferiche connesse al bus che necessitano l'assegnazione di un indirizzo dinamico. È diretto a *I3C_FSM_comm.*
- ***data_mem_cmd_o [31:0]*** bus tramite il quale vengono inviati a *I3C_FSM_comm* i dati immagazzinati nella memoria *I3C_cmd_mem.*
- ***dyn_addr_o [31:0]*** bus tramite il quale vengono inviati a *I3C_FSM_comm* i dati immagazzinati nella memoria *write_dyn_addr_mem.*
- ***id_ass_mem_o [31:0]*** bus tramite il quale vengono inviati a *I3C_FSM_comm* i dati relativi alle periferiche a cui è stato assegnato indirizzo dinamico. La macchina a stati interroga la memoria *addr_id_mem* inviando un indirizzo, la memoria risponde inviando su *id_ass_mem_o[31:0]* ID, BCR o DCR. Essendo che le informazioni occupano 64 bit è necessario che la macchina a stati comunichi quali dati caricare su *id_ass_mem_o[31:0].*
- ***data_i3c_read_mem_i [31:0]*** bus ricevuto da *I3C_FSM_comm* e in ingresso a *I3C_read_mem.* Serve a salvare i dati in arrivo da una periferica connessa al bus.
- ***addr_id_slave_i [31:0]*** bus tramite il quale *I3C_FSM_comm* invia le informazioni relative a una periferica, ovvero i registri ID, DCR e BCR, alla memoria statica *addr_id_mem* e alla memoria dinamica *read_id_mem.*

- ***write_i3c_read_mem_i*** segnale di abilitazione per la scrittura dei dati, diretto alla memoria *I3C_read_mem.* Viene inviato ogni volta che *I3C_FSM_comm* deve scrivere un nuovo dato ricevuto sul bus I3C.
- ***read_i3c_cmd_mem_i*** segnale di abilitazione alla lettura dei dati, proveniente da *I3C_FSM_comm* e diretto alla memoria *I3C_cmd_mem*.
- ***write_id_mem_i*** segnale analogo a *write_i3c_read_mem_i,* ma rivolto alla memoria *read_id_mem.* Viene inviato ogni volta che una periferica invia i propri dati (ID, DCR, BCR) per il processo di assegnazione degli indirizzi dinamici.
- ***read_dyn_addr_mem_i*** segnale analogo a *read_i3c_cmd_mem_i,* ma rivolto alla memoria *write_dyn_addr_mem.* Viene inviato ogni volta che è necessario trasmettere a una periferica il suo indirizzo dinamico.
- ***read_bcr_i*** è un segnale in arrivo da *I3C_FSM_comm* e diretto alla memoria statica *addr_id_mem.* Viene inviato ogni volta che è necessario leggere i dati contenuti nel registro BCR di una specifica periferica.
- ***comm_err_i*** è un segnale in arrivo da *I3C_FSM_comm* e diretto al modulo *register_file* con lo scopo di comunicare un mancato ACK durante una comunicazione.
- ***ccc_err_i*** è un segnale in arrivo da *I3C_FSM_comm* e diretto al modulo *register_file* con lo scopo di comunicare che il Command Code caricato non è valido.
- ***daa_err_i*** è un segnale in arrivo da *I3C_FSM_comm* e diretto al modulo *register_file* con lo scopo di comunicare un mancato ACK durante il Command Code ENTDAA.
- ***rst_RSTDAA_i*** è un segnale in arrivo da *I3C_FSM_comm* e diretto alla memoria statica *addr_id_mem* e alle memorie dinamiche *read_id_mem, write_dyn_addr_mem.* Viene inviato ogni volta che il microprocessore, attraverso il Command Code RSTDAA, vuole resettare tutti gli indirizzi dinamici.
- ***data_ready_i*** è un segnale che ha il compito di comunicare al RI5CY la presenza di nuovi dati in memoria *I3C_read_mem.*

4.8.2 Struttura interna

Il modulo *Register & memory manager* è formato da 7 sotto moduli sincroni e 3 sotto moduli di sola logica combinatoria. Ognuno di questi moduli può essere selezionato attraverso il bus *data_addr_i [4:0]* secondo i seguenti valori:

- "0" seleziona il *reset_register.*
- "1" seleziona il *configuration_register.*
- "2" seleziona lo *state_register.*
- "3" seleziona il *mask_register.*
- "4" seleziona la memoria dedicata ai dati in scrittura *I3C_cmd_mem.*
- "5" seleziona la memoria dedicata ai dati in lettura *I3C_read_mem.*
- "11" seleziona la memoria dinamica *write_dyn_addr_mem* e la memoria statica *addr_id_mem,* entrambe dedicate al processo di assegnazione degli indirizzi.
- "12" seleziona la memoria dinamica *read_id_mem*, dedicata alla lettura di ID provvisorio, DCR e BCR in arrivo dalle periferiche.

I 7 moduli sincroni sono:

- ***Register_file:*** è costituito dall'insieme di 3 registri a 32 bit: il *configuration_register*, il *mask_register* e lo *state_register;* inoltre è presente il registro *reset_register* a 2 bit.
- ***I3C_cmd_mem:*** è la memoria dedicata alla scrittura dei dati che il microprocessore vuole inviare alle periferiche sul bus I3C. È composta da 16 registri a 32 bit organizzati in una struttura FIFO circolare.
- ***I3C_read_mem:*** è la memoria nella quale i dati ricevuti dalle periferiche sul bus I3C vengono scritti e resi disponibili per essere letti dal RI5CY. Ha la stessa struttura di *I3C_cmd_mem.*
- ***Write_dyn_addr_mem:*** è la memoria dedicata alla scrittura degli indirizzi dinamici che il RI5CY vuole inviare alle periferiche che lo richiedono. Ha una struttura FIFO analoga alla memoria *I3C_cmd_mem.*
- ***Read_id_mem:*** è la memoria nella quale i 64 bit che compongono ID provvisorio, DCR e BCR vengono scritti e resi disponibili per essere letti dal RI5CY. Ha una struttura analoga a *I3C_read_mem.*
- ***Addr_id_mem:*** è una memoria statica che viene scritta sia dal RI5CY, che dal bus I3C. ha il compito di associare ad ogni indirizzo dinamico scritto dal microprocessore i 64 bit

ricevuti dalle periferiche. Viene interrogata dalla macchina a stati *I3C_FSM_comm* quando ha bisogno di leggere le informazioni associate a una periferica (vedi processo di In-Band Interrupt). È composta da una struttura dati che contiene due campi: un registro a 32 bit che memorizza le informazioni provenienti dalle periferiche e un registro a 7 bit che memorizza gli indirizzi provenienti dal RI5CY.

- ***FSM_uC_intf:*** è la macchina stati finiti dedicata alla gestione della comunicazione tra RI5CY e Master_I3C. ha il compito di interfacciarsi con il microprocessore e abilitare la lettura o scrittura nelle 5 memorie sopra descritte.

I 3 moduli di logica combinatoria sono:

- ***Decoder_0:*** in base alla valore di *data_addr_i[4:0]* e di *data_req_i* fornisce il segnale di write enable (in uscita da *FSM_uC_intf)* ai registri interni al *register_file* oppure alle memorie *I3C_cmd_mem, write_dyn_addr_mem* e *addr_id_mem.*
- ***Decoder_1:*** ha la stessa struttura di *decoder_0* ma ha il compito di fornire il segnale di read enable alle memorie *I3C_read_mem* e *read_id_mem.*
- ***Output decoder:*** è un multiplexer che fornisce in uscita alcuni registri o valori interni al modulo. Le uscite sono abilitare con alcune combinazioni del bus *data_addr_i[4:0]:*
    - "1" seleziona il *configuration_register.*
    - "2" seleziona lo *state_register.*
    - "3" seleziona il *mask_register.*
    - "5" seleziona i dati in memoria *I3C_read_mem.*
    - "6" seleziona il valore del puntatore in scrittura della memoria *I3C_cmd_mem.*
    - "7" seleziona il valore del puntatore in lettura della memoria *I3C_cmd_mem.*
    - "8" seleziona il valore del puntatore in scrittura della memoria *I3C_read_mem.*
    - "9" seleziona il valore del puntatore in lettura della memoria *I3C_read_mem.*
    - "12" seleziona i dati in memoria *read_id_mem.*

Di seguito si propone un esempio di funzionamento nel caso di scrittura e di lettura da parte del microprocessore.

Nel caso di scrittura nella memoria *I3C_cmd_mem* il microprocessore come prima cosa imposta *data_addr_i[4:0]* nella combinazione "1", così da selezionare il *configuration_register* e settare alcune impostazioni importati per il funzionamento del Master_13C. Successivamente il RI5CY invia la combinazione "4" per indirizzarsi alla memoria *I3C_cmd_mem.* Ora può mandare i dati sul bus *data_wdata_i[31:0]* e attivare *data_req_i,* in modo da attivare la macchina a stati

FSM_uC_intf. Per abilitare la *FSM_uC_intf* alla scrittura deve inviare anche il segnale *data_we_i.* Giunti a questo punto saranno la *FSM_uC_intf* e il *decoder_o* a occuparsi di indirizzare i dati alla memoria *I3C_cmd_mem.* Qualora il microprocessore debba scrivere su registri o memorie diverse da *I3C_cmd_mem* è sufficiente che invii l'indirizzo corretto sul bus *data_addr_i[4:0].*

Nel caso di lettura della memoria *I3C_read_mem* il procedimento è simile: il RI5CY invia sul bus *data_addr_i[4:0]* la combinazione "5" e, dopo che la macchina a stati *FSM_uC_intf* ha inviato read enable, sul bus di uscita *data_rdata_o[31:0]* son forniti i dati presenti all'interno di *I3C_read_mem.*

Per la gestione degli indirizzi dinamici il procedimento è simile, con la differenza che in caso di scrittura nelle memorie *wrtie_dyn_addr_mem* e *addr_id_mem* la combinazione di *data_addr_i[4:0]* è "11". In caso di lettura della memoria *read_id_mem* la combinazione è "12".

4.8.3 La memoria I3C_cmd_mem

È la memoria nella quale il microprocessore scrive i dati da inviare al bus. È selezionabile attraverso l'indirizzo 5'b00100 di *data_addr_i[4:0],* ovvero la sopracitata combinazione "4". Si tratta di una memoria di tipo FIFO (First_In First-Out) circolare, questo significa che i dati vengono salvati in maniera temporanea per poi essere recuperati nell'ordine in cui sono stati inseriti. Contiene 16 registri, ognuno composto da 32 bit. Per garantire il funzionamento FIFO circolare si utilizza un puntatore in scrittura e un puntatore in lettura. Ciò permette di scrivere i dati nel registro (locazione di memoria) selezionato dal puntatore di scrittura e di leggere i dati dal registro selezionato dal puntatore di lettura. L'utilità di questa struttura è garantire un preciso ordine dei dati.

Ad esempio, al segnale di reset la memoria viene svuotata ed entrambi i puntatori selezionano la locazione 0, quando il processore scrive nella memoria *I3C_cmd_mem* 4 valori il puntatore in scrittura verrà incrementato fino al valore 4, mentre quello in lettura rimarrà a 0. Durante la trasmissione dei dati, per mano della *I3C_FSM_comm,* ad ogni dato trasmetto il puntatore in lettura verrà incrementato. Quando il puntatore in lettura raggiunge quello in scrittura (quindi valore 4) vuol dire che tutti i dati sono stati letti. Sono stati implementati dei controlli sullo stato dei puntatori per avere traccia sullo stato di riempimento della memoria.

4.8.3.1 Segnali di ingresso e uscita

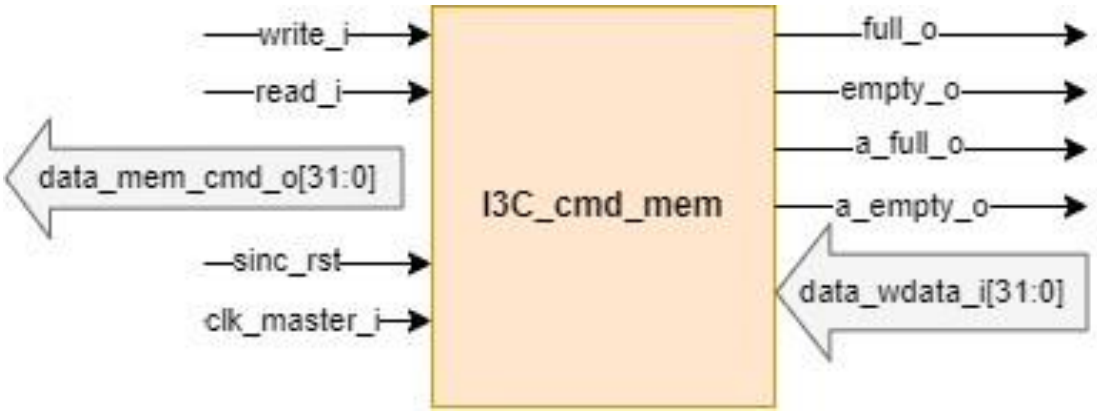

***Figura 4.19**: memoria I3C_cmd_mem*

- ***data_wdata_i [31:0]*** è il bus attraverso il quale il RI5CY invia i dati da scrivere in memoria.
- ***write_i*** è il segnale di write enable che abilita la scrittura della memoria, è generato dal microprocessore.
- ***read_i*** è il segnale di read enable che abilita la lettura della memoria, è generato dalla macchina a stati *I3C_FSM_comm.*
- ***data_mem_cmd_o [31:0]*** è il bus tramite il quale la memoria invia i dati alla macchina a stati *I3C_FSM_comm.*
- ***sinc_rst*** segnale di reset sincronizzato, generato dal modulo *register_file*.
- ***clk_master_i*** segnale di sincronizzazione a cui è stato applicato il clock gating.
- ***full_o*** segnale che comunica al microprocessore di non avere più locazioni libere per la scrittura, il valore viene scritto in *state_register.*
- ***empty_*** segnale che comunica al microprocessore di essere vuota, il valore viene scritto in *state_register*.
- ***a_full_o*** segnale che comunica al microprocessore di essere quasi piena, il valore viene scritto in *state_register.*
- ***a_empty_o*** segnale che comunica al microprocessore di essere quasi vuota, il valore viene scritto in *state_register*.

4.8.3.2 Organizzazione del contenuto

Il contenuto della I3C_cmd_mem è composto da una prima stringa di comando chiamata *command_string* (32 bit) seguita da una stringa di indirizzo. Quest'ultima può assumere il ruolo di *relative_address*, ovvero l'indirizzo interno della periferica selezionata con la *command_string.* Oppure, nel caso di comunicazione CCC Direct, la seconda stringa in memoria fa riferimento all'indirizzo a 7 bit *address_slave* della periferica I3C a cui si rivolge il comando CCC specificato nella *command_string.* A seguire gli N dati da trasmettere.

Si precisa che le stringhe di dati e la stringa *relative_address* possono essere composte da 8, 16, 24 o 32 bit, a seconda di come specificato nella *command_string*. Per la stringa *address_slave* sono presi in considerazione solo i primi 7 bit, corrispondenti all'indirizzo della periferica.

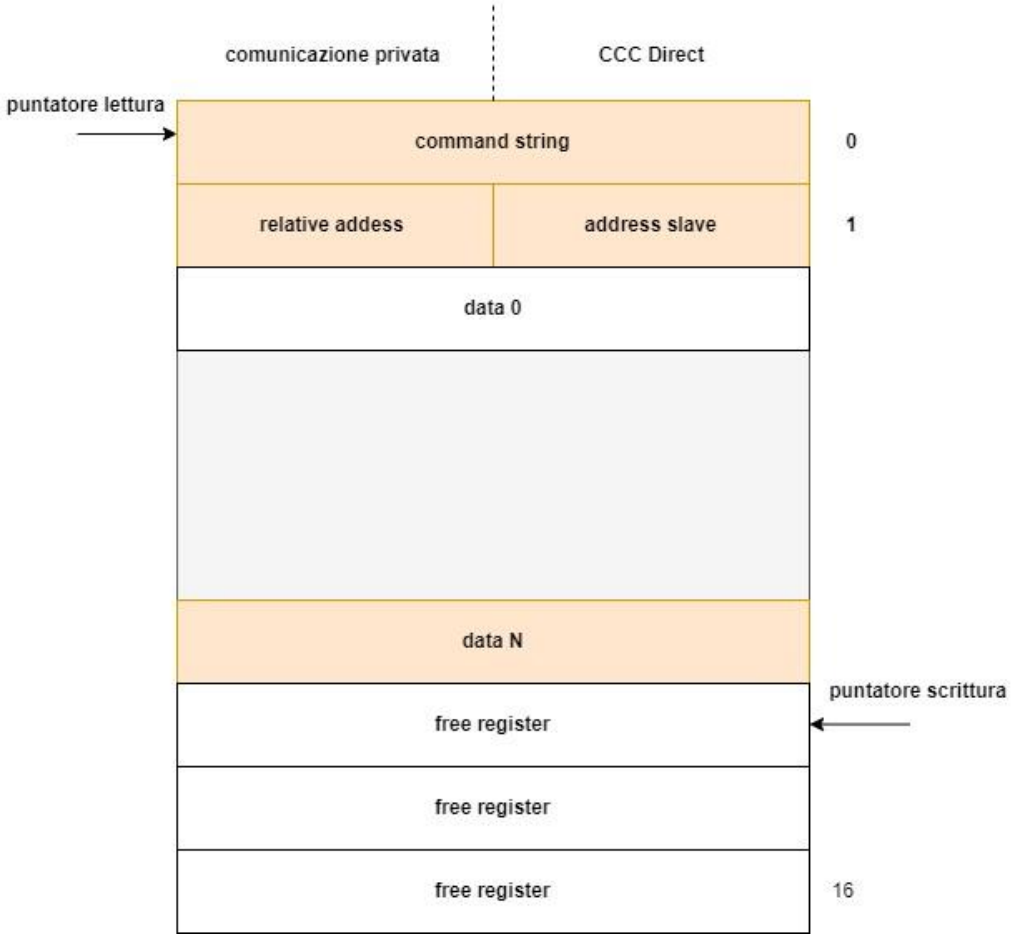

***Figura 4.20**: contenuto della memoria I3C_cmd_mem*

Per una corretta comunicazione sul bus è importante che il microprocessore rispetti l'ordine di riempimento della memoria.

La *command_string* contiene informazioni utili per la macchina a stati *I3C_FSM_comm* riguardo al tipo di comunicazione, per questo è la prima stringa che deve essere letta per una corretta comunicazione. La sua struttura è descritta di seguito:

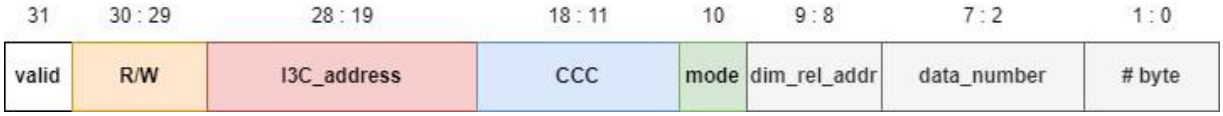

***Figura 4.21**: command string*

- ***valid*** è il bit, attivo alto, utilizzato dal microprocessore per comunicare al Master_I3C la presenza di un dato valido per la comunicazione.
- ***R/W*** sono 2 bit utilizzati per impostare la comunicazione in lettura o scrittura. hanno la seguente codifica:
  - "00" scrittura complessa, richiede il *relative address* in quanto scrittura interna.
  - "10" scrittura semplice.
  - "11" lettura semplice.
  - "10" lettura complessa, richiede il *relative address* in quanto lettura interna.
- ***I3C_address*** è il campo in cui viene inserito l'indirizzo della periferica con la quale si vuole comunicare, compreso l'indirizzo broadcast (7'h7E). per gli indirizzi a 7 bit vengono considerati solo i 7 bit meno significativi.
- ***CCC*** è il campo contenente il byte del Common Command Code, viene letto solamente se richiesto dall'operazione. In una comunicazione I^2C non verrà utilizzato dato che supporta i comandi CCC.
- ***mode*** è un bit che definisce il tipo di comunicazione che il microprocessore vuole iniziare. Se questo bit ha valore "1" allora la comunicazione sarà di tipo I3C, se ha valore "0" sarà di tipo I^2C.
- ***dim_rel_addr*** sono 2 bit che specificano la dimensione in byte del *relative address.* in questo modo è possibile comunicare con indirizzi in formato 8, 16, 24 e 32 bit. La codifica di questo campo è la seguente:
  - "00" corrisponde a un byte (8 bit)
  - "01" corrisponde a due byte (16bit)

- “10” corrisponde a tre byte (24 bit)
- “11” corrisponde a quattro byte (32 bit)

- ***data_number*** contiene l’informazione sul numero di dati che devono essere trasmessi o letti, in altre parole il numero di dati contenenti nelle memorie.
- ***# byte*** sono due bit che specificano la dimensione in byte dei dati da inviare o leggere. Ha la stessa codifica di *dim_rel_addr*.

Si aggiunge una considerazione sulla stringa *address_slave:* essendo prevista solo durante comunicazioni di tipo CCC Direct, la macchina a stati *I3C_FSM_comm* terrà conto solo dei 7 bit meno significativi.

4.8.4 La memoria I3C_read_mem

La memoria *I3C_read_mem* ha la stessa struttura di *I3C_cmd_mem* è selezionabile attraverso l’indirizzo 5’b00101 di *data_addr_i[4:0],* ovvero la combinazione “5*”*. È dedicata alla memorizzazione dei dati inviati dalle periferiche, che saranno poi resi disponibili sul bus *data_rdata_o [31:0].*

4.8.4.1 Segnali ingresso e uscita

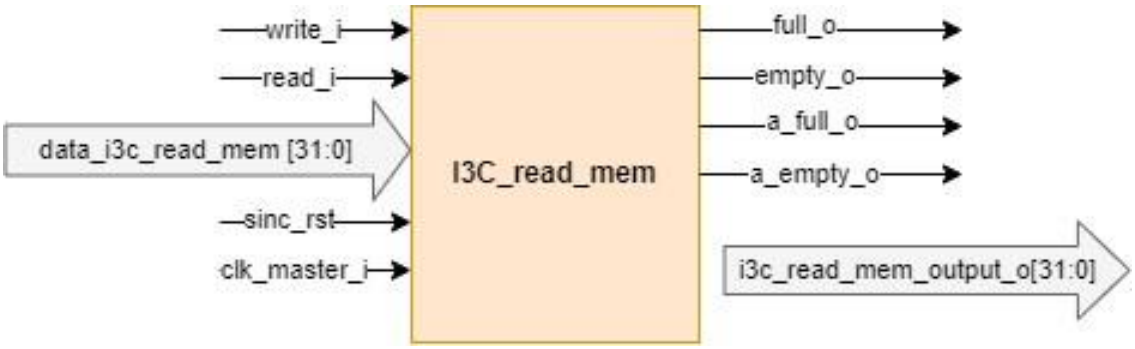

Figura 4.22*: memoria I3C_read_mem*

- ***data_i3c_read_mem [31:0]*** è un bus in arrivo dalla *I3C_FSM_comm* contenente i dati ricevuti dalla periferica.
- ***i3c_read_mem_output_o [31:0]*** è un bus diretto all’*output decoder,* il quale passerà il suo valore a *data_rdata_i [31:0]* quando il microprocessore vorrà leggerne il contenuto.

4.8.5 La memoria write_dyn_addr_mem

È la memoria nella quale il microprocessore carica gli indirizzi dinamici attraverso il bus *data_wdata_i [31:0]* che verranno letti dalla macchina a stati *I3C_FSM_comm* durante la fase di assegnamento. È selezionabile con il valore 5'b01011 di *data_addr_i [4:0]*, ovvero con la combinazione "11". La struttura è la simile alle memorie descritte precedentemente. Si è scelto di utilizzare una memoria FIFO per il processo di assegnazione degli indirizzi dinamici perché anche in questo caso è importante l'ordine di scrittura e lettura dei dati (vedi **figura 4.11**).

4.8.5.1 Segnali di ingresso e uscita

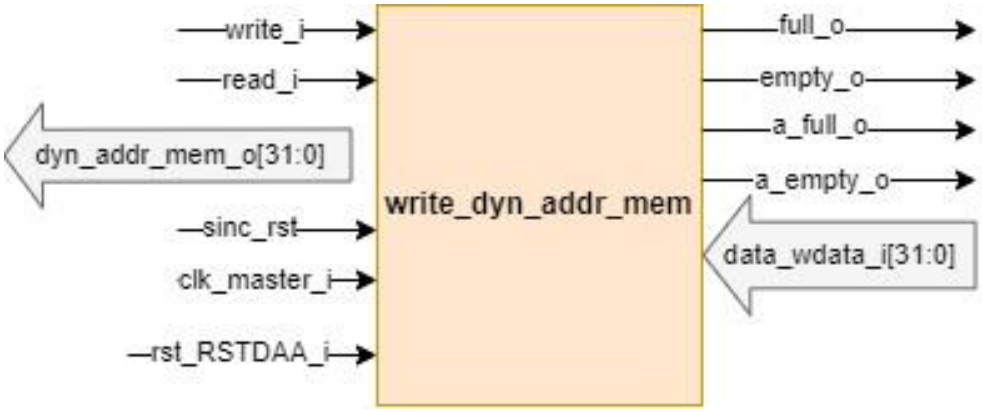

***Figura 4.23**: memoria write_dyn_addr_mem*

- ***rst_RSTDAA_i*** è un segnale in arrivo dalla macchina a stati *I3C_FSM_comm* con il compito di resettare la memoria nel caso di Command Code RSTDAA che prevede il reset degli indirizzi dinamici.
- ***dyn_addr_mem_o [31:0]*** è il bus diretto alla macchina a stati *I3C_FSM_comm* contenente l'indirizzo dinamico per la periferica. Vengono inviati solo i 7 bit meno significativi del bus.

4.8.6 La memoria read_id_mem

È la memoria dedicata a salvare i dati riguardo ID provvisorio, DCR e BCR in arrivo dalle periferiche. È selezionabile attraverso il valore 5'b01100 di *data_addr_i [4:0]*, ovvero la combinazione "12". Ha la stessa struttura di *write_dyn_addr_mem*, ma riceve i dati dalla macchina a stati e li fornisce in uscita sul bus *data_rdata_i [31:0]*.

4.8.6.1 Segnali di ingresso e uscita

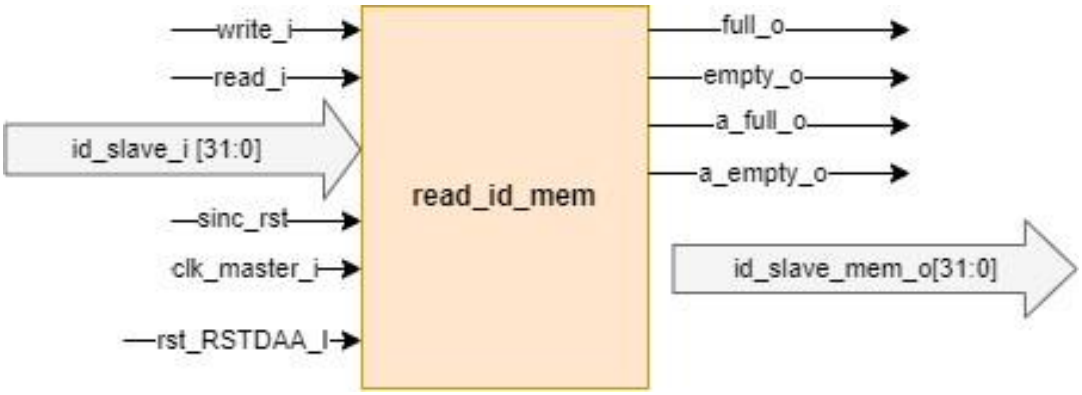

***Figura 4.24**: memoria read_id_mem*

- ***id_slave_i [31:0]*** è il bus in arrivo dalla macchina a stati *I3C_FSM_comm.* Ogni periferica manda al Master_I3C 64 bit (48 bit per ID provvisorio, 8 bit per DCR e 8 bit per BCR). Quindi il bus *id_slave_i [31:0]* viene sempre caricato due volte per ogni periferica che richiede l'indirizzo dinamico.
- ***Id_slave_mem_o [31:0]*** è il bus che diretto al microprocessore che manda a *data_rdata_o [31:0]* il contenuto della memoria. Analogamente a *id_slave_i [31:0]* per ogni periferica il microprocessore dovrà richiedere due volte la lettura della memoria *read_id_mem.*

4.8.7 La memoria addr_id_mem

È la memoria dedicata ad associare ad ogni indirizzo dinamico mandato dal microprocessore le informazioni riguardo ID provvisorio, DR e BCR in arrivo dalla periferica. Il RI5CY scrive contemporaneamente l'indirizzo nella memoria *write_dyn_addr_mem* e *addr_id_mem*, sempre attraverso il *bus_wdata_i [31:0]* e con la combinazione "11" di *data_addr_i [4:0]*.

Per chiarire il funzionamento si propone un esempio di funzionamento. Attraverso il *configuration register* il microprocessore comunica al Master_I3C il numero di periferiche che necessitano di un indirizzo (ipotizziamo 4). Successivamente, attraverso il bus *data_wdata_i[31:0]* il RI5CY invia 4 indirizzi dinamici che vengono salvati nelle memorie *write_dyn_addr_mem* e *addr_id_mem*. Durante la fase di assegnazione degli indirizzi i 64 bit (ID, DCR e BCR) vengono salvati in *addr_id_mem* in una struttura associata all'indirizzo precedentemente salvato. Questo è necessario per poter associare ad ogni indirizzo dinamico assegnato le informazioni sulla specifica periferica.

La memoria è stata implementata attraverso una struttura a 32 celle di memoria, ogni cella è composta da due registri a 32 bit per la memorizzazione dei 64 bit (ID, DCR, BCR) e un registro a 32 bit per la memorizzazione dell'indirizzo dinamico. I primi due registri vengono scritti dalla *I3C_FSM_comm* con il segnale *write_id* abilitato. L'ultimo registro è scritto dal RI5CY con *write_dyn* abilitato.

Quando la macchina a stati *I3C_FSM_comm* ha bisogno di leggere le informazioni associate a una periferica invia *read_bcr* per abilitare la lettura del BCR e attraverso *dyn_addr_i [31:0]* manda il valore dell'indirizzo richiesto. La memoria invia a *I3C_FSM_comm* il valore del BCR attraverso il bus *id_ass_mem_o [31:0]*.

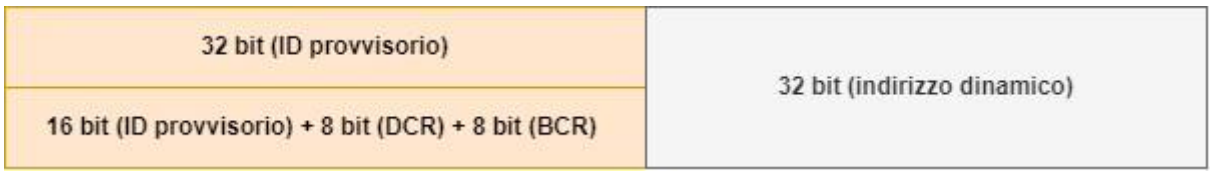

***Figura 4.25**: struttura di una delle 32 celle di memoria interne a addr_id_mem*

4.8.7.1 Segnali di ingresso e uscita

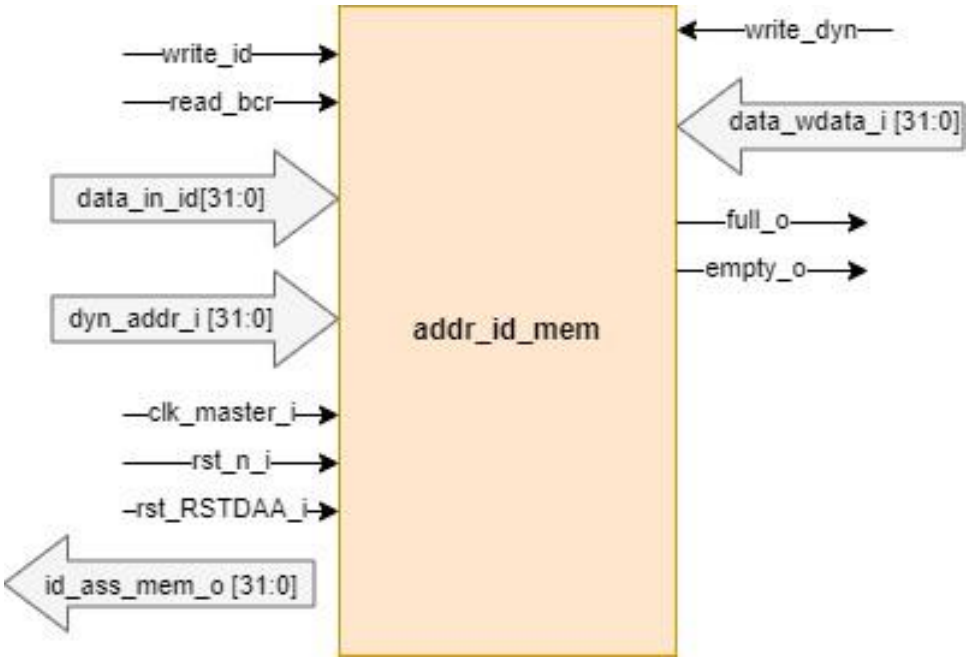

***Figura 4.26**: memoria addr_id_mem*

- ***write_id*** è il segnale in arrivo da *I3C_FSM_comm* che abilita la scrittura dei 64 bit di informazione della periferica.
- ***read_bcr*** è il segnale in arrivo da *I3C_FSM_comm* e ha il compito di abilitare la lettura da parte della macchina a stati del BCR.
- ***write_dyn*** è il segnale che abilita la scrittura dell'indirizzo dinamico da parte del RI5CY.
- ***clk_master_i*** è il clock sincrono.
- ***rst_n_i*** è il reset sincrono.
- ***rst_RSTDAA_i*** è il reset in arrivo da *I3C_FSM_comm* a seguito del CCC RSTDAA.
- ***full_o*** è il segnale che ha il compito di informare il microprocessore che la memoria è piena.
- ***empty_o*** è il segnale che ha il compito di informare il microprocessore che la memoria è vuota.
- ***data_in_id [31:0]*** è l bus in arrivo da *I3C_FSM_comm* contenente i bit di ID, DCR e BCR.

- ***dyn_addr_i [31:0]*** è il bus in arrivo da *I3C_FSM_comm* contenente l'indirizzo dinamico di cui si vuole leggere le informazioni dei registri ID, DCR o BCR.
- ***id_ass_mem_o [31:0]*** è il bus attraverso il quale la memoria invia a *I3C_FSM_comm* i registri ID, DCR o BCR corrispondenti all'indirizzo *dyn_addr_i [31:0]*.
- ***data_wdata_i [31:0]*** è il bus attraverso il quale il RI5CY invia gli indirizzi dinamici.

4.8.8 Register file

Questo modulo è composto da 3 registri da 32 bit ed un registro da 2 bit. Nel *register_file* sono contenute tutte le informazioni riguardanti lo stato del circuito Master_I3C e le impostazioni di configurazione per la comunicazione sul bus I3C. Ogni registro può essere scritto dal microprocessore ed il suo contenuto può essere letto per ottenere informazioni, per esempio, sulla presenza o meno di interrupt. I registri a 32 bit sono lo *state_register* il *mask_register* ed il *configuration_register* mentre il registro a due bit è il *reset_register*.

4.8.8.1 Configuration register

È il registro nel quale il RI5CY carica le impostazioni sulla configurazione necessarie al circuito Master_I3C per potersi adattare al tipo periferica con la quale deve instaurare una comunicazione. Essendo il registro che contiene l'informazione sul clock gating, il *configuration_register* è l'unico registro ad aggiornarsi sul fronte di salita del clock di sistema e non sul fronte del segnale *clk_master_o*. I campi del registro di configurazione sono riportati nella figura seguente:

***Figura 4.27**: Struttura del configuration register*

- **CG** è il bit di abilitazione per il clock gating. Se il valore all'interno di questo campo è '0' allora il clock gating è disabilitato, in caso contrario il clock gating è abilitato. Questo bit viene inviato al modulo *clk_manager* che lo utilizzerà per generare il segnale clk_master_o.

- ***I2C_addr_dim*** è il bit in cui viene specificata la dimensione dell'indirizzo I^2C della periferica con la quale il circuito deve effettuare lo scambio dei dati. Lo '**0**' indica che la dimensione è pari a 10 bit mentre l' '**1**' indica una dimensione di 7bit. Questo campo verrà inviato alla *I3C_FSM_comm* la quale lo utilizzerà percostruire l'indirizzo da trasmettere sulla linea SDA.
- ***end_of_count*** questo campo sarà utilizzato dalla *I3C_FSM_comm. End_of_count* contiene l'informazione riguardante il rapporto fra le due frequenze, di sistema e della trasmissione I^2C, necessario per poter rispettare le temporizzazioni, imposte dal protocollo, sui segnali SCL ed SDA in modalità Open Drain.
- ***end_of_count_pp*** ha la stessa funzione di *end_of_count* ma è necessario per poter rispettare le temporizzazioni sui segnali SCL ed SDA in modalità Push Pull.
- ***# slave*** è il campo attraverso il quale il RI5CY comunica al Master_I3C il numero di periferiche connesse al bus I3C che necessitano di un indirizzo dinamico.

4.8.8.2 State register

Lo *state_register* è il registro incaricato di memorizzare lo stato del circuito. È selezionabile con la combinazione "2" di *data_addr_i [4:0]*. Questo registro viene scritto dal Master_I3C il quale utilizza i campi dello *state register* per segnalare al RI5CY l'avvenimento di eventi particolari al suo interno. Questo è anche il registro che viene letto dal microprocessore in caso di interrupt. L'unica operazione di scrittura concessa al microprocessore è quella relativa all'azzeramento dei flag di interrupt in modo da segnalare al Master che il motivo che ha causato l'interruzione è stato risolto. La figura seguente mostra la struttura del registro:

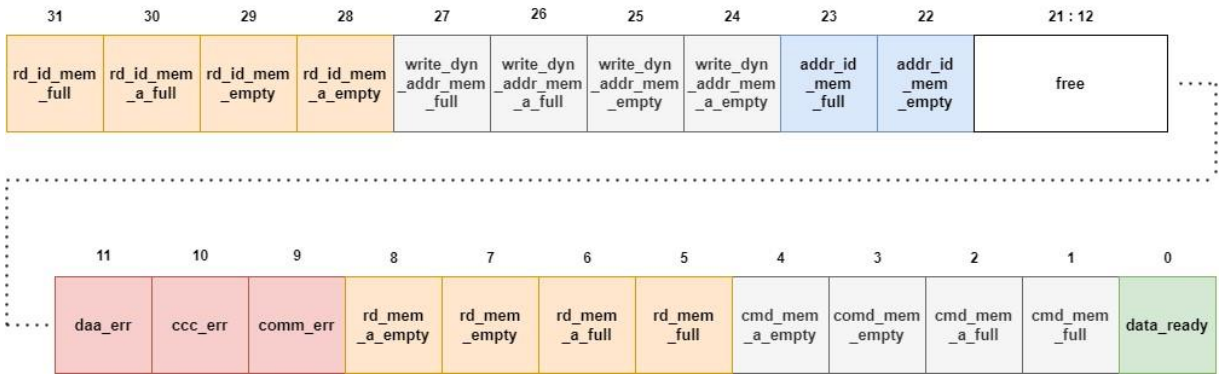

***Figura 4.28**: struttura dello state register*

- ***data_ready*** è un bit che serve ad avvisare il microprocessore che un'operazione di lettura sul bus è terminata. Il dato ricevuto dalla periferica, memorizzato momentaneamente nella *I3C_read_mem,* è pronto per essere letto dal RI5CY. Questo bit può essere resettato solamente dal processore essendo l'unico dispositivo autorizzato a leggere quei dati.
- ***cmd_mem_full*** comunica al RI5CY che la *I3C_cmd_me*m è piena. Se il microprocessore provasse ad effettuare un'operazione di scrittura, il nuovo dato andrebbe perso.
- ***cmd_mem_a_full*** è un bit che serve per comunicare al RI5CY che la *I3C_cmd_mem* è quasi piena. La soglia per questo valore è stata impostata a ¾ della capienza massima delle due memorie.
- ***cmd_mem_empty*** comunica al microprocessore che la memoria è completamente vuota.
- ***cmd_mem_a_empty*** è l'analogo del bit cmd_mem_a_full ma per il caso di memoria quasi vuota. La soglia in questo caso è stata impostata ad ¼ della capacità massima.
- ***rd_mem_full*** indica che la memoria I3C_read_mem è piena. In questo caso, se dovesse arrivare un nuovo dato da una periferica connessa al bus I3C, questo andrebbe perso poiché non ci sarebbe la possibilità di salvarlo.
- ***rd_mem_a_full*** è il bit analogo a *cmd_mem_a_full* ma per la memoria *I3C_read_mem.* Anche in questo caso è settato a ¾ della capacità massima.
- ***rd_mem_empty*** comunica al RI5CY che la memoria di lettura è vuota.
- ***rd_mem_a_empty*** indica che la memoria di lettura è quasi vuota. La soglia, anche in questo caso, è stata impostata ad ¼ della capacità massima.
- ***comm_err*** è un bit per la segnalazione degli errori di comunicazione avvenuti sul bus I3C. Nel nostro caso specifico viene scritto ad ‘1' dalla macchina a stati I3C_FSM_comm ogniqualvolta, durante uno scambio dati con una periferica, non riceve il bit di acknowledge.
- ***ccc_err*** analogo a *comm_err* ma viene scritto quando l’errore riguarda la lettura di un CCC non valido.
- ***daa_err*** analogo a *comm_err,* viene letto quando l’errore di ACK è originato dal processo ENTDAA.

- ***rd_id_mem_full*** comunica al RI5CY che la *read_id_me*m è piena. Se una periferica provasse ad effettuare un'operazione di scrittura, il nuovo dato andrebbe perso.
- ***rd_id_mem_a_full*** è un bit che serve per comunicare al RI5CY che la *read_id_mem* è quasi piena. La soglia per questo valore è stata impostata a ¾ della capienza massima delle due memorie.
- ***rd_id_mem_empty*** comunica al microprocessore che la memoria è completamente vuota.
- ***rd_id_a_empty*** è l'analogo del bit *rd_id_mem_a_full* ma per il caso di memoria quasi vuota. La soglia in questo caso è stata impostata ad ¼ della capacità massima.
- ***write_dyn_addr_mem_full*** indica che la memoria *write_dyn_addr_mem* è piena. In questo caso, se dovesse arrivare un nuovo indirizzo dal microprocessore, questo andrebbe perso poiché non ci sarebbe la possibilità di salvarlo.
- ***write_dyn_addr_mem_a_full*** è il bit analogo a *cmd_mem_a_full* ma per la memoria *write_dyn_addr_mem.* Anche in questo caso è settato a ¾ della capacità massima.
- ***write_dyn_addr_mem_empty*** comunica al RI5CY che la memoria w*rite_dyn_addr_mem* è vuota.
- ***write_dyn_addr_a_empty*** indica che la memoria di lettura è quasi vuota. La soglia, anche in questo caso, è stata impostata a ¼ della capacità massima.
- ***comm_err*** è un bit per la segnalazione degli errori di comunicazione avvenuti sul bus I3C. Viene scritto ad ‘1' dalla macchina a stati *I3C_FSM_comm* ogniqualvolta, durante uno scambio dati con una periferica, non riceve il bit di acknowledge.
- ***ccc_err*** è un bit per la segnalazione degli errori legati ai Common Command Code. Viene scritto ad ‘1’ dalla macchina a stati *I3C_FSM_comm* ogniqualvolta il microprocessore comunica all’interno della *command string* CCC non supportato da *I3C_FSM_comm.*
- ***daa_err*** è un bit per la segnalazione degli errori di comunicazione avvenuti sul bus I3C durante il processo di assegnazione degli indirizzi dinamici. Viene scritto ad ‘1' dalla macchina a stati *I3C_FSM_comm* ogniqualvolta, durante l’invio dell’indirizzo nel processo ENTDAA, non riceve il bit di acknowledge.

4.8.8.3 Mask register

Il *mask register* è un registro strutturalmente identico allo *state register* ed è scritto dal RI5CY per abilitare o disabilitare alcune delle fonti di interrupt. è selezionabile con la combinazione "3" di *data_addr_i [4:0]*. Ogni volta che il microprocessore scrive uno '0' all'interno di uno dei campi del *mask register* significa che disabilita la sorgente di interrupt corrispondente situata all'interno dello *state register*. Al contrario, scrivendo '1' consente a quella sorgente di essere abilitata. Questa meccanica è stata implementata attraverso un'operazione di *AND* bit a bit tra le i due registri.

4.8.8.4 Reset register

Il registro *reset register* è rivolto alla generazione di un segnale di reset sincrono da utilizzare all'interno del circuito Master_I3C. la struttura adottata da questo registro è quella di un sincronizzatore a doppio flip – flop ed è raffigurata nello schema seguente:

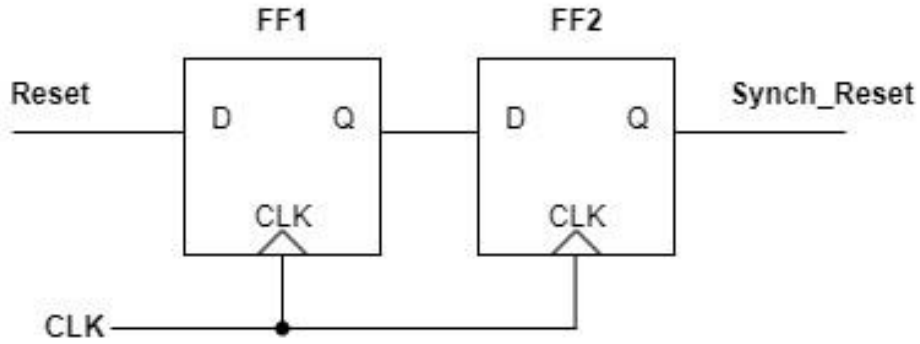

***Figura 4.29**: Struttura del reset register*

L'obiettivo è quello di sincronizzare un segnale asincrono, ovvero il reset di sistema proveniente dal TARDIS. Essendo sia FF1 che FF2 temporizzati con lo stesso clock, assegnando l'uscita di FF1 all'ingresso di FF2 il risultato è un segnale con lo stesso valore di *Reset,* sincronizzato con *CLK,* ovvero *Synch_Reset.*

Si tratta di una tecnica comunemente utilizzata in elettronica digitale per ridurre gli effetti di *metastabilità.* Teoricamente più sono i flip-flop in cascata migliore sarà la sincronizzazione, si è scelto che utilizzarne due fosse un giusto compromesso tra latenza e sincronizzazione.

4.8.9 Il modulo FSM_uC_intf

Il modulo *FSM_uC_intf* è la macchina a stati interna a *Register & memory manager*, ha il compito di processare le richieste in arrivo dal RI5CY e sincronizzare i decoder di lettura e scrittura in funzione delle richieste in arrivo.

Ogni operazione di scrittura è caratterizzata dall'invio dei dati sul bus *data_wdata_i [31:0]*, accompagnati dal segnale di richiesta di operazione *data_req_i* e dal segnale *data_we_i*, impostato a '1' per operazioni di scrittura. Subito dopo la ricezione di *data_req_i* la macchina a stati deve mandare il segnale *data_gnt_o* e il segnale di write enable *write_i3c_o.* sul fronte di salita successivo la macchina a stati genera il segnale *data_rvalid_o* per comunicare la validità dei dati.

L'operazione di lettura è simile a quella di scrittura, il microprocessore invia la richiesta sulla linea *data_req_i*, accompagnata dalla scrittura '0' sulla linea *data_we_i.* in questo caso la macchina a stati abilita il segnale di read enable *read_uc_o* invece di *write_i3c_o.*

Si precisa che questa macchina a stati gestite solo le operazioni di lettura che partono dal microprocessore; quindi, tutte le operazioni di interruzione del bus da parte delle periferiche non vengono gestite

4.8.9.1 Segnali di ingresso e uscita

***Figura 4.30**: macchina a stati FSM_uC_intf*

- ***data_req_i*** è un segnale che viene utilizzato per comunicare l'intenzione, da parte del microprocessore, di effettuare un'operazione di lettura o scrittura sul Master_I3C.

- ***data_we_i*** è un segnale utilizzato insieme al segnale *data_req_i* e specifica il tipo di operazione che il microprocessore vuole eseguire. Il valore ‘1’ indica un’operazione di scrittura, ‘0’ per la lettura.
- ***data_gnt_o*** segnale generato dal MasterI3C ogni volta che è pronto a gestire una nuova richiesta.
- ***data_rvalid_o*** è un segnale utilizzato per comunicare al RI5CY che i dati forniti in uscita dopo un’operazione sono validi e disponibili in memoria.
- ***clk*** è il clock di sistema, la *FSM_uC_intf* si aggiorna sul fronte di salita del clock di sistema perché deve essere in grado di abilitare la scrittura del registro *configuration register.*
- ***rst_n_i*** segnale di reset sincrono generato dal modulo *register file.*
- ***write_i3c_o*** è il segnale di write enable. È diretto al *decoder_0,* il quale lo manda alla macchina a stati *I3C_FSM_comm*oppue ai registri interni al *register file*, in base al valore di *data_addr_i [4:0]*.
- ***read_uc_o*** segnale di read enable. È diretto a *decoder_1,* il quale lo manda al decoder in output per gestire i dati da passare a *data_rdata_o [31:0],* in base al valore di *data_addr_i [4:0],*

4.9 Conclusioni

In questo capitolo è stato descritto il progetto a livello implementativo. Per ogni modulo è stato dedicato un capitolo per la descrizione dei segnali di ingresso e uscita, al fine di fornire una documentazione quanto più completa possibile per la comprensione del circuito. Inoltre, si è ritenuto importante introdurre i diagrammi di flusso in modo da migliorare la comprensione della macchina a stati e del suo funzionamento.

Capitolo 5

Verifica funzionale del Master_I3C

Questo capitolo è dedicato alla descrizione della metodologia utilizzata per verificare il funzionamento del Master_I3C. verranno testate alcune funzionalità, commentandone i risultati.

5.1 Metodologia utilizzata

Al fine di verificare le funzionalità del Master_I3C sono stati realizzati dei testbench che emulano il comportamento del microprocessore e delle periferiche presenti sul bus I3C. Lo scopo di questi test è fornire in ingresso al dispositivo una serie di stimoli e verificare che i segnali di uscita si comportino come previsto da protocollo. Il microprocessore RI5CY esegue le richieste di lettura e scrittura attraverso due task implementati nel file di testbench. Come descritto più avanti il funzionamento delle periferiche, ovvero la risposta in arrivo sulla linea *sda_i* è gestita manualmente direttamente all'interno del file di testbench. Tutti i test sono stati eseguiti attraverso il software SimVision, proprietà di Cadence.

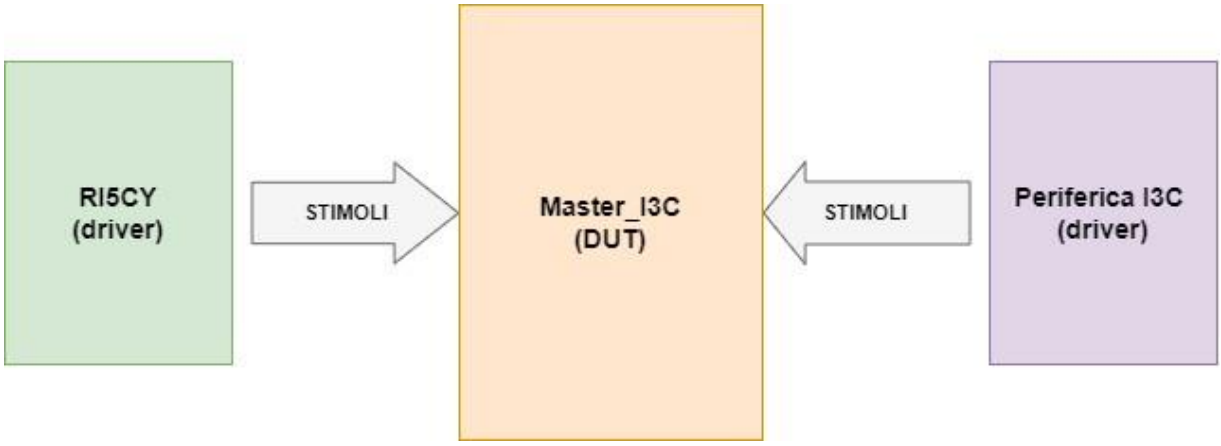

***Figura 5.1**: struttura dei test*

```
//write task

task write_task (input [4:0] addr_test, input [31:0] data_in_test);

        data_req_i <= 1'b1;
        data_we_i <= 1'b1;
        data_addr_i <= addr_test;
        #1;
        data_wdata_i <= data_in_test;

        @(negedge clk);
          if (data_gnt_o) begin
                $display("gnt ok");
          end
          else begin
                $display("gnt not ok");
          end

        @(posedge clk);
        #1;
        while (!data_gnt_o) begin
           @(posedge clk);
           #1;
        end
        data_req_i <= 1'b0;

endtask : write_task
```

Figura 5.2: *script per il task di scrittura*

```
//read task

task read_uC_mem (input [4:0] addr_test);

        data_req_i <= 1'b1;
        data_we_i <= 1'b0;
        data_addr_i <= addr_test;
        #1;

        @(negedge clk);
          if (data_gnt_o) begin
                $display("gnt ok");
          end
          else begin
                $display("gnt not ok");
          end

        @(posedge clk);
        #1;
        while (!data_gnt_o) begin
           @(posedge clk);
           #1;
        end
        data_req_i <= 1'b0;

endtask : read_uC_mem
```

Figura 5.3: *script per il task di lettura*

5.2 Funzionalità testate

Di seguito un elenco delle funzionalità testate:

- Scrittura interna di una periferica I3C.
- Lettura multipla proveniente da periferica I3C.
- Caricamento di dati nella memoria *I3C_cmd_mem* da parte del microprocessore.
- Lettura della memoria *I3C_read_mem* da parte del microprocessore.
- Funzionamento del Common Command Code DISEC e ENEC.
- Processo di assegnazione e gestione degli indirizzi dinamici suddiviso come segue:
    - Common Command Code ENTDAA con una periferica.
    - Evento di Hot Join.
    - Common Command Code RTSDAA.

5.3 Risultati dei test

Il primo test eseguito è descritto dal seguente script e si divide in tre parti: una trasmissione di tre stringhe che rappresentano la *command string*, il *relative address* e il dato da scrivere; la ricezione di due dati; la lettura dei precedenti dati scritti in memoria *I3C_read_mem.*

```
//OP: scrittura i3c + lettura i3c


  rst_ext_i <= 1'b1;
  sda_i <= 1'b0;
  data_wdata_i <= 32'd0;
  rst_ext_i <= 1'b0;
  #20;
  rst_ext_i <= 1'b1;
  #10;
  @(posedge clk);
  write_task (5'd1, 32'b00000000000000001100000000111011);  //configuration register
  write_task (5'd4, 32'b10000011011100000011010000000011);  //command string
  write_task (5'd4, 32'b11111111111111111111111111111111);  //relative_address a 8 bit
  write_task (5'd4, 32'b11100011100011100011100011100011);  //data to send 1

  #25000;
  @(posedge clk);
  write_task (5'd4, 32'b11100001010100000000011100000011);  //command string
  #3000;
  sda_i <= 1'b1;
  #8000;
  sda_i <= 1'b0;
  #30000;
  read_uC_mem(5'd5);
  read_uC_mem(5'd5);
  #30000;
```

Figura 5.4: *script per il primo test*

Come previsto da protocollo la prima stringa ricevuta dal microprocessore deve essere indirizzata al *configuration register* per definire le impostazioni di comunicazione. Successivamente, come illustrato in **figura 5.5** viene caricata la memoria *I3C_cmd_mem* con il *relative address* e *la command string.*

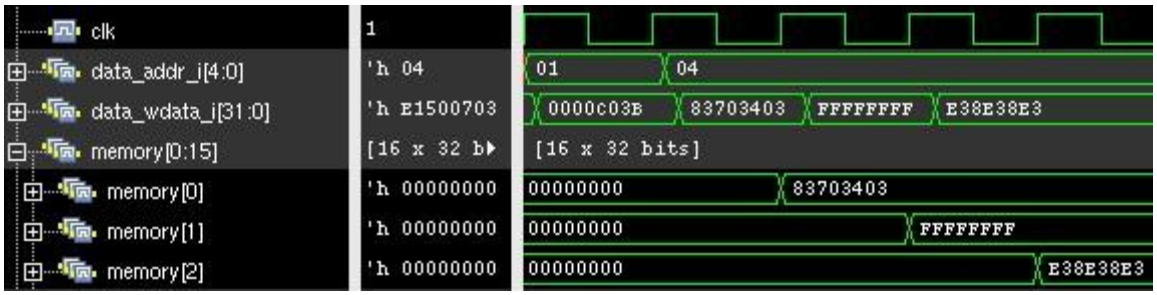

Figura 5.5: *caricamento in memoria*

La trasmissione dell'indirizzo avviene prima in configurazione Open Drain, per rispettare le condizioni di arbitraggio, successivamente i dati vengono inviati in Push Pull, come illustrato in **figura 5.6**. Si noti che, dopo ogni lettura della memoria I3C_cmd_mem viene eliminato il dato in memora.

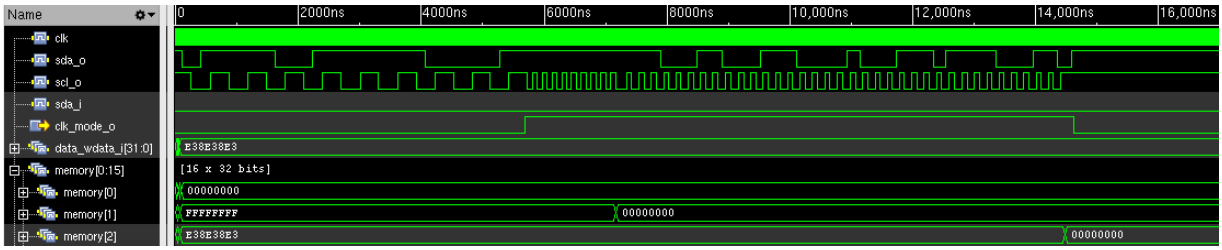

Figura 5.6: *trasmissione dei dati*

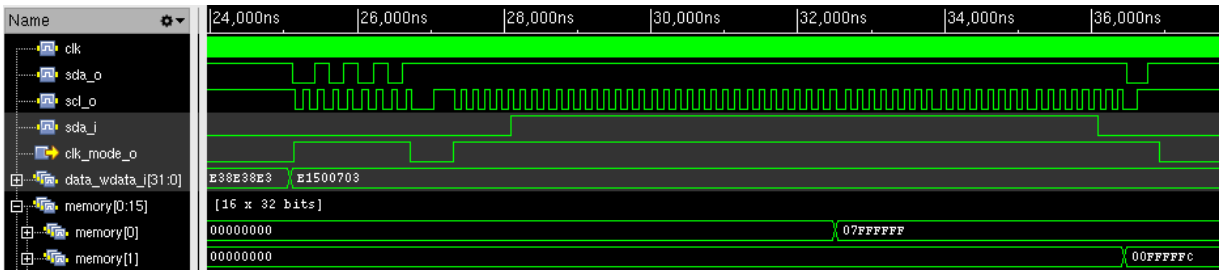

Figura 5.7: *ricezione dei dati*

La ricezione dei dati avviene tutta in Push Pull perché non sono passati 5 [ns], quindi non è necessaria la condizione di arbitraggio. I dati vengono caricati nella memoria *I3C_read_mem*. la **figura 5.7** descrive quanto appena detto.

Il test si conclude con la lettura dei dati immagazzinati nella memoria *I3C_read_mem*. la **figura 5.8** descrive come il valore di *data_addr_i [4:0]* viene impostato a '5' e i dati in memoria vengono passati al bus *data_rdata_o [31:0]* diretto al microprocessore.

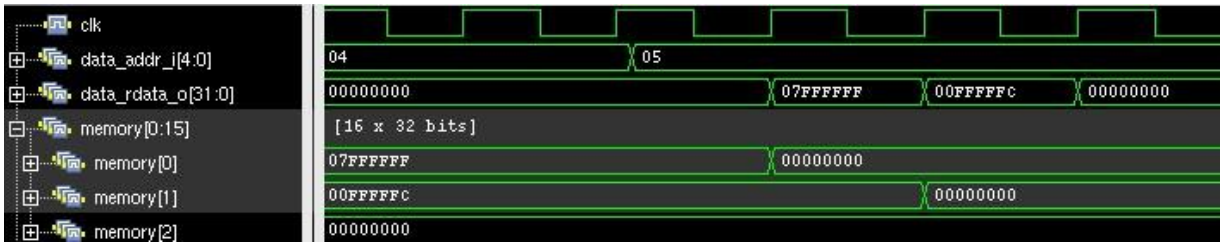

Figura 5.8: *lettura della memoria*

Il secondo test consiste nel disabilitare gli eventi di interruzione attraverso il comando DISEC e successivamente nel riabilitarli. Di seguito si propone lo script di test.

```
//OP: DISEC + ENEC

  rst_ext_i <= 1'b1;
  data_wdata_i <= 32'd0;
  sda_i <= 1'b0;
  rst_ext_i <= 1'b0;
  #20;
  rst_ext_i <= 1'b1;
  #10;
  @(posedge clk);//per allinearci sul fronte di salita del clock
  write_task (5'd1, 32'b00000000000000000110000000111011);  //configuration register
  write_task (5'd4, 32'b10100011111100000000111100000011);  //command string
  #15000;
  sda_i <= 1'b1;
  #3000;
  sda_i <= 1'b0;
  #3000;
  write_task (5'd4, 32'b10100011111100000000011100000011);  //command string
  #15000;
  sda_i <= 1'b1;
  #3000;
  sda_i <= 1'b0;
  #300000;
```

Figura 5.9*: script per il secondo test*

La figura seguente mostra prima la trasmissione del Command Code DISEC, il quale disabilita gli eventi di interruzione. Infatti, si può notare un primo tentativo da parte di una periferica di abbassare la linea *sda_i* senza avere risposta dal Master. Successivamente, il microprocessore carica il Command Code ENEC, il quale abilita gli eventi di interruzione. Infatti, il secondo evento di interruzione viene accettato dal master che risponde abilitando il segnale SCL in attesa di leggere l'indirizzo inviato dal Master. Si noti il comportamento segnale *DISEC_en_i* che gestisce questa funzionalità. Notiamo che il segnale di sincronizzazione interno al Master_I3C

(*clk*) è sensibilmente più veloce a confronto del segnale di sincronizzazione generato da *I3C_FSM_comm* sulla linea SCL.

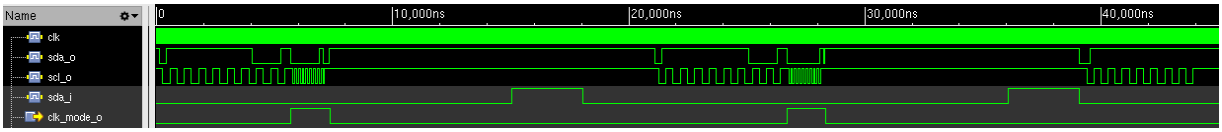

Figura 5.10: *DISEC e ENEC*

In **figura 5.11** si propone uno "zoom" sulla trasmissione dell'indirizzo broadcast (7'b1111110) e del bit di R/W in scrittura (1'b0); a seguire viene trasmesso in Push Pull il CCC e il parity bit, in questo caso DISEC che ha valore 8'b00000001, di conseguenza il parity bit sarà a zero. Confrontando con la **tabella 2.1** si conclude che la trasmissione avviene in maniera corretta.

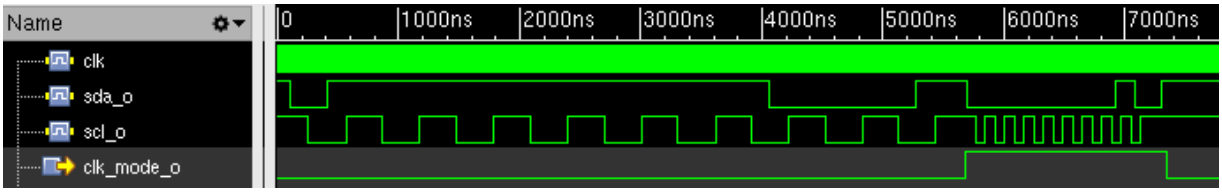

Figura 5.11*: trasmissione del CCC*

Per il terzo test si è voluto gestire l'assegnazione degli indirizzi dinamici, inizialmente attraverso il Command code ENTDAA è stato assegnato l'indirizzo dinamico a una periferica, in un secondo momento, attraverso l'evento di Hot Join una nuova periferica si è unica al bus ricevendo un indirizzo dinamico. Il test si conclude resettando la memoria degli indirizzi attraverso il comando RSTDAA.

Di seguito lo script utilizzato:

```
//OP: ENTDAA + HOT JOIN + RSTDAA


  rst_ext_i <= 1'b1;
  data_wdata_i <= 32'd0;
  sda_i <= 1'b0;       //ack non gestito, sda_i sempre bassso
  rst_ext_i <= 1'b0;
  #20;
  rst_ext_i <= 1'b1;
  #10;
  @(posedge clk);
   write_task (5'd1, 32'b00000100000000001100000000111011);  //config_reg.
   write_task (5'd4, 32'b10000011111100000011110000000111);  //command string
   write_task (5'd11, 32'b00000000000000000000000010111111);  //dynamic address 1
   write_task (5'd11, 32'b00000000000000000000000010100011);  //dynamic address 2

  #15000;
  sda_i <= 1'b1;
  #30000;
  sda_i <= 1'b0;
  #30000;
  sda_i <= 1'b1;
  #2000;
  sda_i <= 1'b0;
  #20000;
  sda_i <= 1'b1;
  #7000;
  sda_i <= 1'b0;
  #3650;
  sda_i <= 1'b1;
  #300;
  sda_i <= 1'b0;
  #3000;
  write_task (5'd11, 32'b10101010101010101010101010010001);  //dynamic address 3
  #1000;
  sda_i <= 1'b1;
  #40000;
  sda_i <= 1'b0;
  #100;
  @(posedge clk);
  #10000;
  write_task (5'd4, 32'b10000011111100000011010000000111);  //command string
  write_task (5'd4, 32'b10000011111100000011010000000111);  //command string
  #100000;
```

***Figura 5.12**: script per il terzo test*

Dal grafico in **figura 5.13** notiamo la struttura della memoria *addr_id_mem.* Dopo aver salvato in memoria i due indirizzi dinamici nella struttura *data [31:0]* il Master riceve e salva, sempre in *data [31:0]*, i 64 bit che compongono ID, DCR e BCR in arrivo dalla prima periferica. Dopo la ricezione dei dati da ogni periferica il Master invia l'indirizzo dinamico. Di seguito i risultati del test:

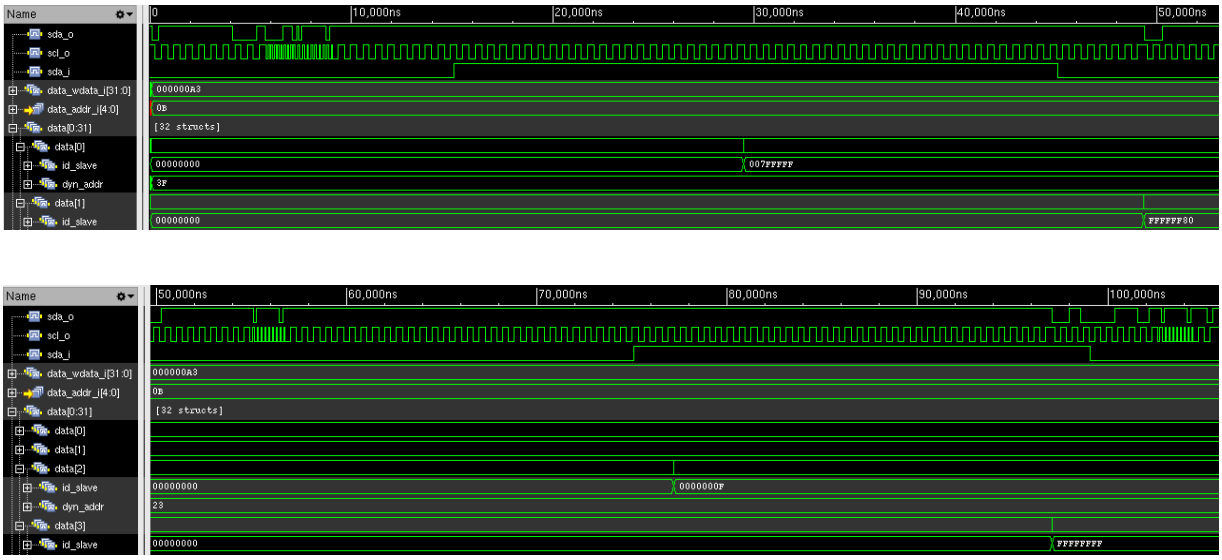

Figura 5.13: *ENTDAA*

A seguito di una richiesta di Hot Join, ovvero dell'invio dell'indirizzo 8'b00000100 da parte di una periferica, il Master procede assegnando un nuovo indirizzo dinamico. Come descritto in **figura 5.14** sarà compito del microprocessore caricare il nuovo indirizzo dinamico.

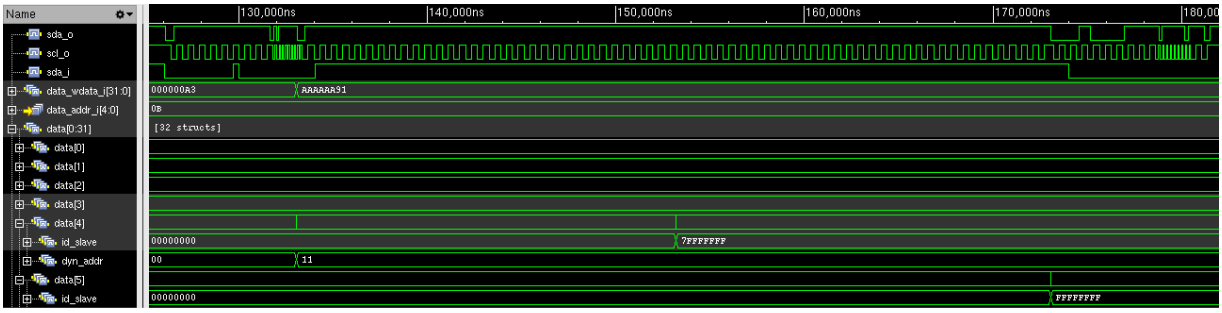

Figura 5.14: *Hot Join*

Si conclude il test mostrando il Command Code RSTDAA, notiamo che a seguito di questo comando l'intera memoria degli indirizzi viene azzerata. Anche in questo caso, non essendo trascorso abbasta tempo per permettere alle periferiche di contendersi il bus, l'indirizzo broadcast è inviato in modalità Push Pull. La **figura 5.15** mostra quanto detto.

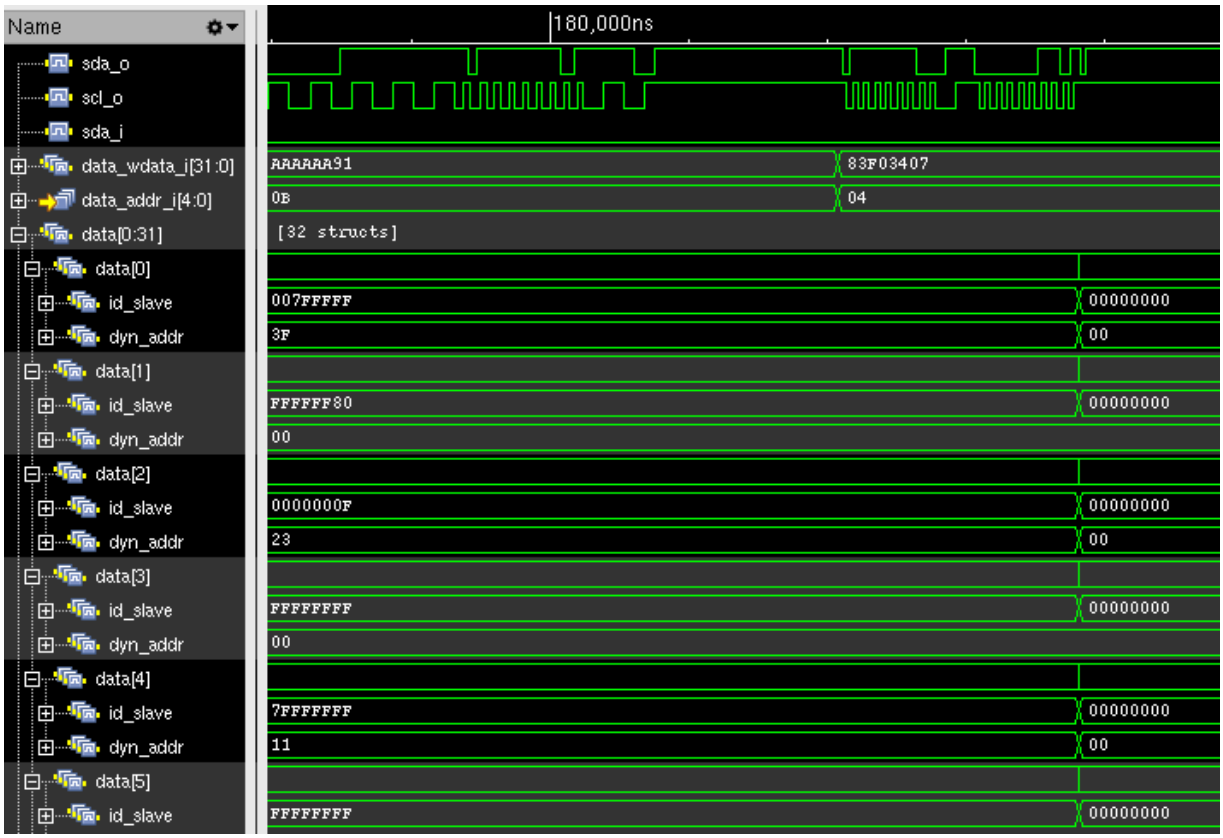

Figura 5.15: *RTSDAA*

5.4 Conclusioni

In questo capitolo sono stati descritti alcuni dei risultati ottenuti nella fase di verifica, effettuata sul circuito a livello RTL. In alcune figure (vedi **figura 5.6**) è mostrato il segnale di clock di sistema a 100 [MHz]. È stato voluto per permettere di vedere il rapporto tra il clock di sistema e il segnale trasmesso su SCL. Si è ritenuto opportuno soffermarsi su una verifica effettuata riguardo la gestione degli indirizzi dinamici in quanto rappresenta una delle novità più importanti introdotte dal protocollo I3C.

Capitolo 6

Sintesi su dispositivo FPGA e risultati

Per la realizzazione del circuito su dispositivo FPGA è stato utilizzato il software Vivado di proprietà di Xilinx. Il processo è automatizzato e trasforma la descrizione RTL del circuito in una Netlist di porte elementari che mantengono la stessa funzionalità. Successivamente sono stati ricavati i report relativi al circuito ottenuto con il processo di sintesi.

6.1 FPGA utilizzata

Per la sintesi dell'RTL è stata utilizzata una FPGA della Xilinx appartenente alla famiglia Zynq 7000 il cui modello è: 7z030sbv485-1. La scelta è ricaduta su questo dispositivo in quanto è lo stesso su quale è stata realizzata l'IP del microprocessore. Il dispositivo è dotato di:

- 78600 LUT: sono le Look Up Table riprogrammabili per l'implementazione delle funzioni logiche.
- 157200 FF: sono i Flip – Flop di tipo D
- 150 I/O: sono i pin di ingresso e di uscita.
- 32 BUFG: sono i buffer globali, utilizzati per utilizzati per rigenerare il segnale di clock che sincronizza tutti i moduli.
- 39300 F7 Muxes: sono i multiplexer a 7 ingressi.
- 19650 F8 Muxes: sono i multiplexer a 8 ingressi.

6.2 Risultati della sintesi

In **figura 6.1** si propone il layout della FPGA ottenuto dopo il processo di sintesi e implementazione, in azzurro sono rappresentate le celle utilizzate dal circuito Master_I3C implementato col l'RTL.

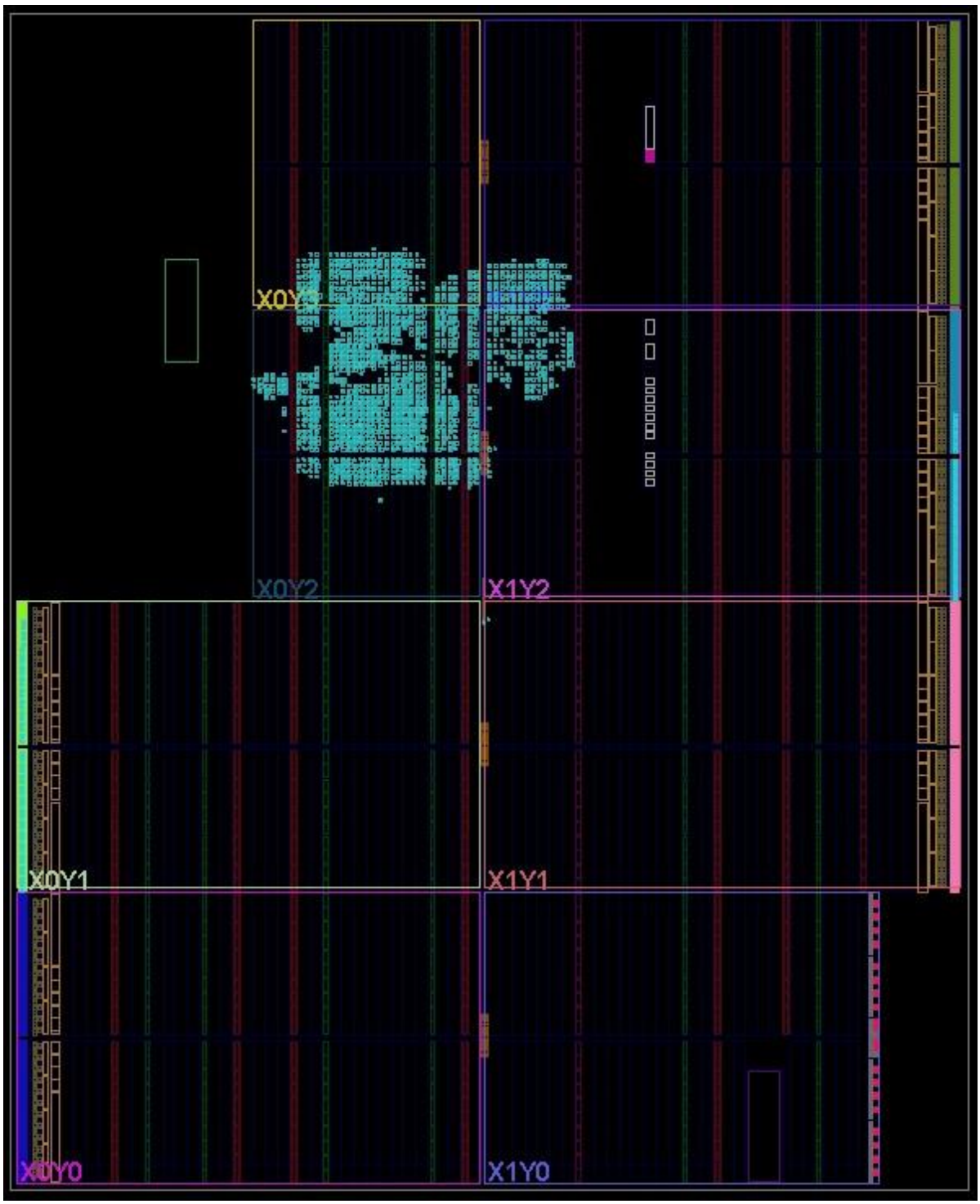

***Figura 6.1**: realizzazione su FPGA*

Notiamo che la superficie occupata dal circuito è solo una minima parte del dispositivo FPGA utilizzato. In **figura 6.2** è rappresentata la differenza in termini di risorse utilizzate tra il Master_I^{2}C (sviluppato precedentemente) e il Master_I3C.

Resource	Utilization	Available	Utilization %
LUT	3546	78600	4.51
FF	2252	157200	1.43

Resource	Utilization	Available	Utilization %
LUT	1991	78600	2.53
FF	1313	157200	0.84

***Figura 6.2**: risorse utilizzate da Master_I3C (sopra) e Master_I^{2}C (sotto)*

In **figura 6.3** è rappresentata la distribuzione delle risorse all'interno dei moduli del Master_I3C. Si noti che la maggior parte delle risorse è impiegata dal *Register & memory manager*, essendo il modulo più grande.

Name	Slice LUTs (78600)	Slice Registers (157200)	F7 Muxes (39300)	F8 Muxes (19650)	Bonded IOB (150)	BUFGCTRL (32)
master_i3c	3546	2252	211	92	79	1
i3C_FSM_comm (i3c_fsm_comm)	1305	273	5	1	0	0
Register (top_register)	2240	1977	206	91	0	0
addr_id_mem (mem_ass_addr)	95	142	0	0	0	0
FSM_uC_intf (comm_fsm)	2	2	0	0	0	0
i3c_cmd_mem (mem)	715	534	64	32	0	0
i3c_read_mem (mem_0)	546	530	64	20	0	0
read_id_mem (mem_addr)	489	528	64	32	0	0
rf (register_file)	242	86	0	0	0	0
write_dyn_addr_mem (mem_add	147	123	14	7	0	0
sda_i_monitor (sda_monitor)	1	2	0	0	0	0

Figura 6.3: *distribuzione delle risorse utilizzate*

Per i risultati sulla potenza e sui vincoli di timing è stata impostata la frequenza di funzionamento del microcontrollore e quella utilizzata nella fase di simulazione, ovvero 100 [MHz].

I risultati sui vincoli di setup e di hold sono i seguenti:

Setup		Hold	
Worst Negative Slack (WNS):	4,015 ns	Worst Hold Slack (WHS):	0,061 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6416	Total Number of Endpoints:	6416

Figura 6.4: *vincoli di timing*

- **Setup** si riferisce al minimo tempo che un segnale deve essere stabile prima del bordo del bordo di clock del trigger affinchè il dato venga correttamente campionato. In questo caso il percorso peggiore (WNS) è rappresentato da un tempo di setup di 4,015 [ns].
- **Hold** si riferisce al tempo minimo per cui il dato deve essere mantenuto stabile dopo il bordo di clock di trigger affinchè venga correttamente campionato. In questo caso il percorso peggiore (WHS) è rappresentato da un tempo di hold di 0,061 [ns].

I risultati sul consumo di potenza quando il Master_I3C funziona a una frequenza di 100 [MHz] sono descritti di seguito:

Total On-Chip Power:	**0.159 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25,4°C**
Thermal Margin:	59,6°C (22,7 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	2,6°C/W
Power supplied to off-chip devices:	0 W

Figura 6.5*: consumo totale in potenza*

Il consumo totale del circuito digitale dopo la sintesi e l'implementazione è di 0,159 [W] al quale corrisponde una temperatura di giunzione di 25,4 [°C] che rappresenta un risultato ottimo in termini di dissipazione termica.

La **figura 6.6** mostra come viene distribuita la potenza all'interno del circuito. La potenza dissipata durante il funzionamento è di 35 [mW], ovvero la potenza dinamica. Si tratta di un risultato buono considerando che il dispositivo verrà inserito all'interno di un circuito più complesso.

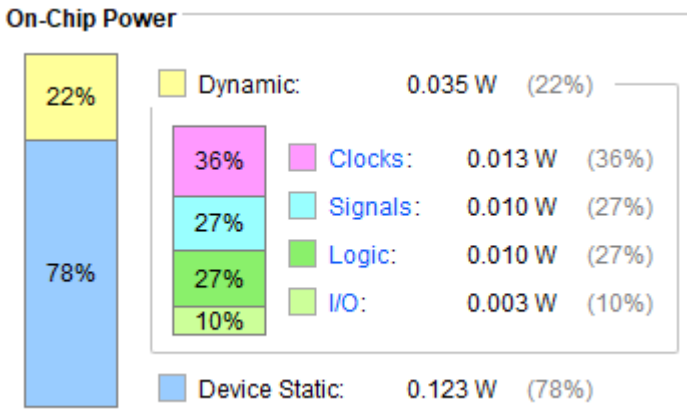

Figura 6.6: *distribuzione della potenza totale*

6.3 Conclusioni

I risultati ottenuti dai report di Vivado riguardo risorse utilizzate, timing e potenza sono conformi con quanto ci si aspettava. Possiamo concludere che il dispositivo è in grado di funzionare alla frequenza di 100 [MHz]

Capitolo 7

Conclusione e sviluppi futuri

Durante l'attività di tesi è stato progettato e testato in System Verilog un circuito digitale Master I3C. successivamente è stato sintetizzato su dispositivo programmabile FPGA. L'obiettivo del lavoro era progettare un dispositivo che permettesse al processore RI5CY e ad altre periferiche di comunicare su un bus I3C.

La prima parte del lavoro è stata dedicata alla comprensione del protocollo I3C e successivamente alla realizzazione pratica. L'approccio implementativo è stato quello di dividere ogni ruolo all'interno del circuito in un modulo, questo ha permesso di facilitare il processo di sintesi su FPGA. La seconda parte è stata dedicata alla fase di verifica del funzionamento dell'RTL attraverso la costruzione di un modulo di testbench.

Le simulazioni eseguite descrivono un funzionamento in linea con quanto ci si aspettava. Solamente dopo il successo della fase di verifica si è passato alla sintesi su FPGA. Dai report ricavati si nota che le risorse utilizzate dal circuito sono una minima parte di quelle disponibili. Questo sicuramente è un dato positivo in quanto il Master_I3C dovrà essere inserito all'interno di un circuito più grande; quindi, è bene che abbia un'occupazione di area e un utilizzo di risorse contenuto. Inoltre, anche i risultati riguardo i vincoli di tempi sono positivi in quanto vengono rispettati i vincoli di setup e di hold per la frequenza di funzionamento (100MHz). Riguardo alla potenza dissipata è stato ottenuto un valore totale di 0,158 Watt che rappresenta un risultato accettabile.

Si conclude proponendo alcuni sviluppi possibili che sarebbe interessante portare avanti:

- Aggiungere ulteriori comandi CCC per sfruttare in maniera completa le novità introdotte dal protocollo I3C.
- Inserire il circuito in un ambiente di verifica UVM (*Universal Verification Metodology*).

Bibliografia

- MIPI Alliance, “Specification for I3C Basic^{SM} v1.0”, MIPI Alliance, 2016-2018.
- Bottini Nicola, “Progettazione di un circuito digitale dedicato alla gestione dell’interfaccia fra un microprocessore di tipo RISC-V ed il bus di comunicazione seriale I^2C (Master) e realizzazione su dispositivi FPGA”, Università degli Studi di Genova, 2020.
- “RISC-V Specifications”, https://riscv.org/technical/specifications/ .

Appendice

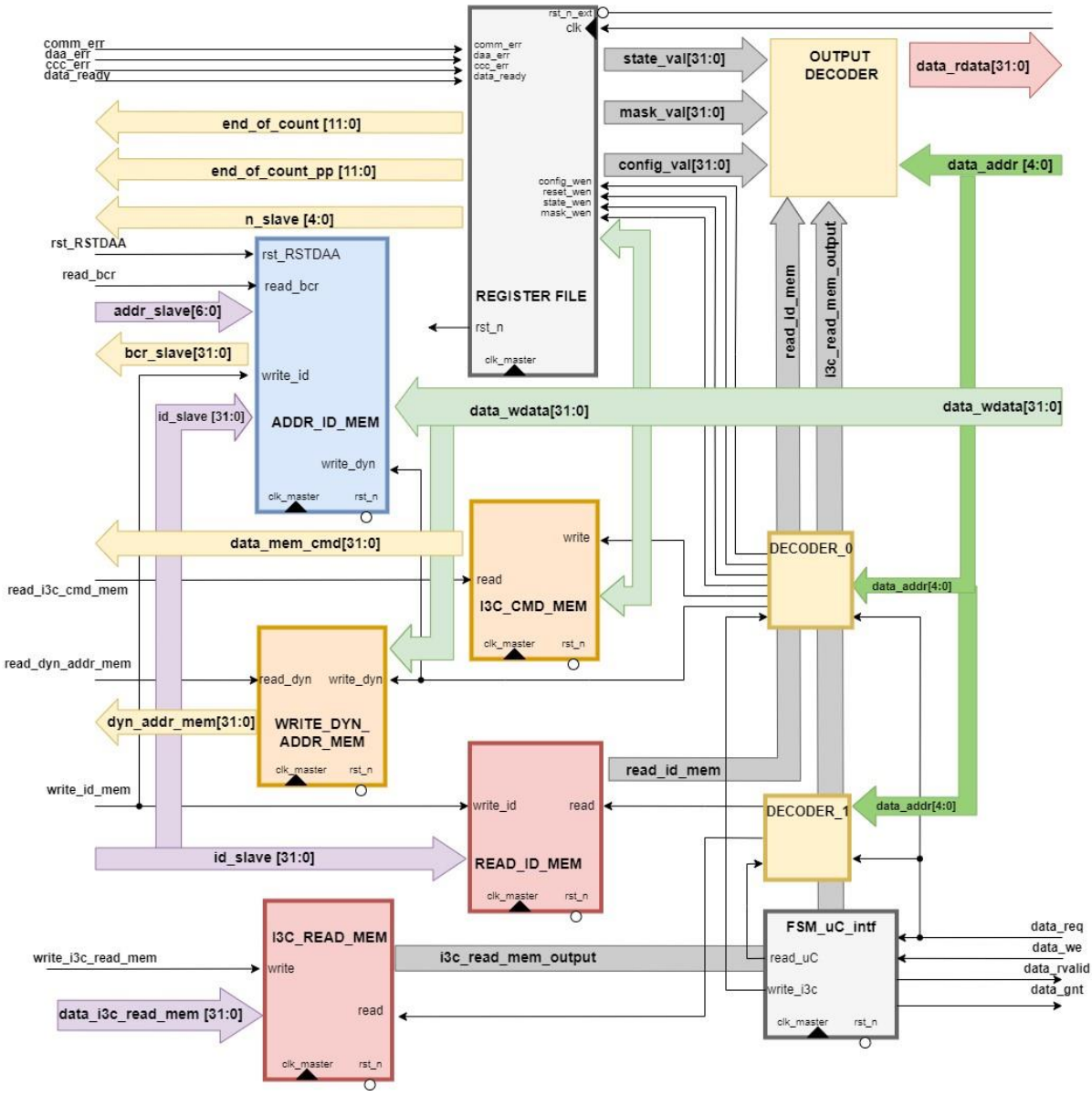

***Figura**: Schema a blocchi interno del modulo Register & memory manager*

Si propone uno schema a blocchi semplificato del modulo *Register & memory manager* con particolare attenzione sulla gestione della comunicazione tra microprocessore e macchina a stati *I3C_FSM_comm.* L'interfaccia con la macchina a stati è a sinistra, mentre quella con il microprocessore è a destra. Al fine di semplificare la rappresentazione e facilitarne la lettura alcuni segnali non sono stati rappresentati.