

Politecnico di Torino
Scuola di Ingegneria e Architettura

Sistemi Digitali Integrati

Prof. Massimo Rou Roch

Prof. Maurizio Zamboni

Relazione FFT



**Politecnico
di Torino**

Federico Cobianchi
Onice Mazzi
Antonio Telmon

A.A. 2025/2026

Indice

1	Introduzione	1
2	Data Flow Diagram	2
2.1	Specifiche sui blocchi operazionali	2
2.2	Approccio ASAP	2
2.3	Approccio ALAP	2
2.4	Approccio scelto	2
3	Datapath	4
3.1	ROM Rounding	6
4	Control Unit	9
5	Appendice	10

1 Introduzione

La FFT (Fast Fourier Transform) è un'operazione fondamentale per tutti i sistemi di elaborazione dei segnali digitali. È utilizzata nelle telecomunicazioni, nell'elaborazione audio e nei sistemi embedded ad alte prestazioni. L'algoritmo FFT si basa sull'operazione butterfly, che è una struttura di manipolazione dei dati che esegue combinazioni lineari di dati complessi mediante somma, sottrazione e moltiplicazione con coefficienti complessi.

Lo scopo di questo progetto è progettare un'unità di elaborazione dedicata per eseguire la Butterfly FFT, utilizzando tecniche di microprogrammazione e considerando vincoli realistici dell'architettura hardware. Più specificamente, questo progetto si occupa della gestione di dati complessi in una rappresentazione frazionaria a complemento a due di 24 bit, dell'uso della Scansione in Virgola Mobile a Blocco Incondizionata per gestire il sovraccarico e dell'implementazione di un datapath ottimizzato dati i vincoli di risorse computazionali limitate e pipeline interna. Il lavoro include la derivazione del diagramma di flusso dei dati dell'algoritmo, l'ottimizzazione del datapath e dell'unità di controllo, la completa descrizione dell'architettura in VHDL e la verifica funzionale attraverso simulazioni. Infine, la Butterfly implementata deve essere utilizzata come blocco di base per l'implementazione e il collaudo di una FFT 16x16, che ne dimostra la validità e la scalabilità della soluzione.

Per creare la singola butterfly sono stati seguiti i seguenti passi:

- Creazione del Data Flow Diagram
- Stima del tempo di vita delle variabili
- Creazione del Datapath
- Creazione della Control Unit (CU)
- Test finali

Data la necessità di utilizzare diversi blocchi logici quali moltiplicatori, sommatore, sottrattori, registri e multiplexer sono state eseguite delle simulazioni intermedie rispetto ai punti appena descritti per facilitare il lavoro di debug. Si è proceduto nel modo descritto in quanto è da preferire rispetto ad un approccio "trial and error" dove tutti i blocchi non vengono testati e si procede solamente al test finale della butterfly. Nel caso fosse stata scelta questa strategia progettuale sarebbe stato pressoché impossibile andare a trovare dove fosse l'errore nel caso si fosse verificato qualche malfunzionamento.

Una volta completata la singola butterfly è stato creato il processore che esegue la FFT unendo tra loro le varie unità necessarie per adempiere alla richiesta finale del progetto. Una volta implementato il tutto il sistema è stato testato nella sua interezza per constatare l'effettivo funzionamento.

2 Data Flow Diagram

In questo capitolo si parlerà delle specifiche imposte sui blocchi operazionali. Si andrà a confrontare gli approcci "As Soon As Possible" *ASAP* e "As Late As Possible" *ALAP*. Infine verrà illustrato l'approccio che è stato utilizzato per ottimizzare le tempistiche dell'algoritmo.

2.1 Specifiche sui blocchi operazionali

In questo Progetto si supponeva di poter utilizzare per ciascuna Butterfly un unico blocco moltiplicatore. In grado di poter svolgere sia l'operazione di moltiplicazione tra due numeri in ingresso sia come un moltiplicatore per 2 di un dato in ingresso. Le due operazioni erano selezionabili attraverso un segnale di controllo esterno al blocco operatore. Si è supposto che il moltiplicatore avesse due livelli di pipeline, ovvero che il risultato fosse disponibile al registro di uscita dopo 3 colpi di clock. Mentre l'operazione di shift (moltiplicazione per 2) avesse un livello di pipeline e quindi l'uscita sarebbe disponibile dopo 2 colpi di clock. Il blocco moltiplicatore è stato rappresentato in 1 con il blocco *verde* e mentre l'operazione di shift è stata rappresentata con il blocco *viola*.

I blocchi sommatore e sottrattore avevano entrambi un livello di pipeline e sono rappresentati rispettivamente dal blocco *rosso* e *blu*.

Al fine di rappresentare anche l'operazione di ROM Rounding presente alla fine dell'algoritmo è stato deciso di impiegare un colpo di clock per l'operazione di arrotondamento (blocco *azzurro*) e utilizzare successivamente un registro controllato esternamente per mantenere in vita le variabili di uscita della butterfly.

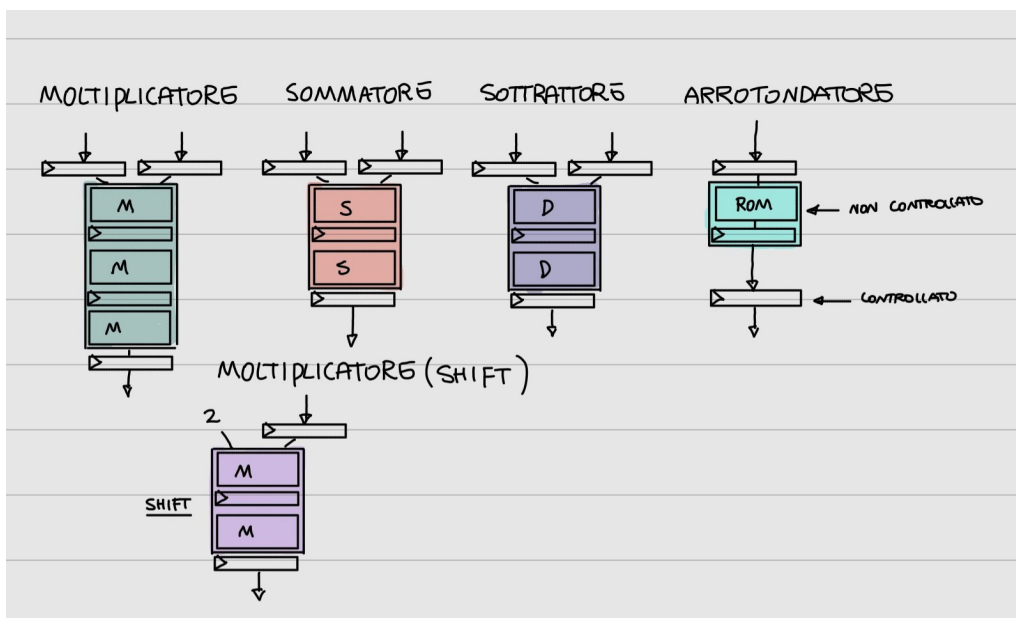


Figura 1: Blocchi elementari del data flow diagram

2.2 Approccio ASAP

L'approccio "As Soon As Possible", è un approccio che predilige lo svolgimento delle operazioni non appena si ha disponibilità. da continuare. inserire foto.

2.3 Approccio ALAP

L'approccio "As Late As Possible", questo approccio predilige lo svolgimento delle operazioni il più tardi possibile. da continuare. inserire foto.

2.4 Approccio scelto

Per ottimizzare le tempistiche dell'algoritmo avendo delle restrizioni sul numero di operatori si è optato per il data flow diagram illustrato in 2

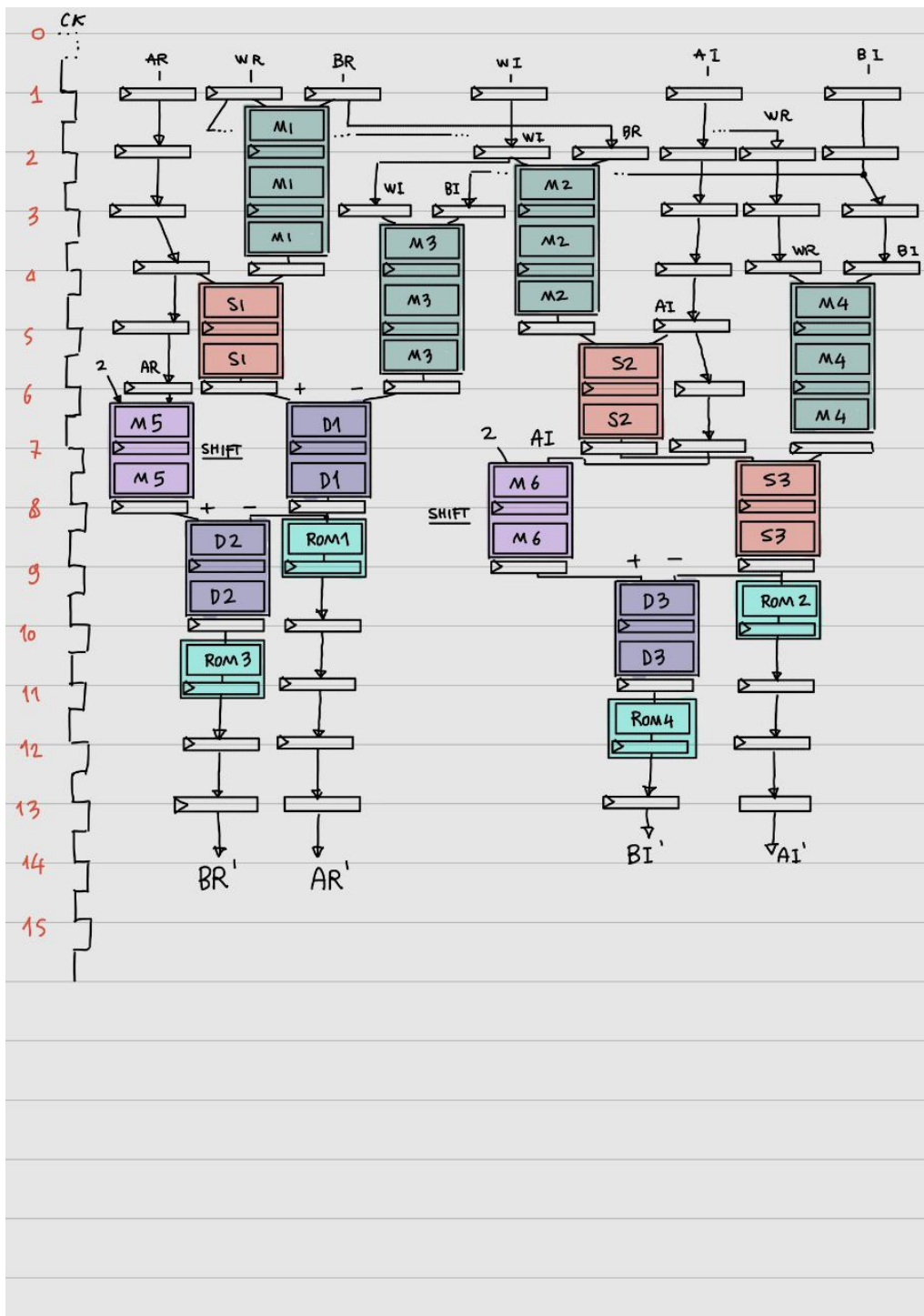


Figura 2: Data flow diagram

3 Datapath

Dopo aver stimato e valutato il tempo di vita delle variabili si è iniziato a progettare il datapath necessario a svolgere tutte le operazioni richieste della CU. Il primo datapath studiato è rappresentato in *Fig. 3*. Come si vede dallo schema non è stato apportato ancora nessun miglioramento volto all'ottimizzazione del numero di BUS e/o al loro parallelismo.

Successivamente si è preso come riferimento il Data Flow Diagram (DFD) e si sono apportate delle migliorie per ciò che concerne l'efficienza dello schema circuitale. Come si vede da *Fig. 4* il register file è stato mantenuto e tutti i segnali che devono entrare all'interno del Datapath passano attraverso di esso. A valle sono stati inseriti dei MUX con lo scopo di selezionare i vari dati da mandare ai blocchi logici. Dal DFD è chiaramente visibile il fatto che i vari segnali, durante il loro tempo di vita, entreranno solo in specifici blocchi logici, risulta perciò superfluo e deleterio avere dei collegamenti (BUS) tra ogni uscita del register file e ogni blocco logico. Dato che l'uscita di un blocco logico potrebbe dover essere riutilizzata in uno step successivo si è fatto uso di registri intermedi che permettono di memorizzare e di riportare il dato in ingresso quando risulta necessario. Ciò è conveniente in quanto, facendo in questo modo, si risparmiano molte scritture su BUS che risultano essere lente e dispendiose in termini energetici. Infine, è stato esplicitato il blocco ROM Rounding che serve per arrotondare.

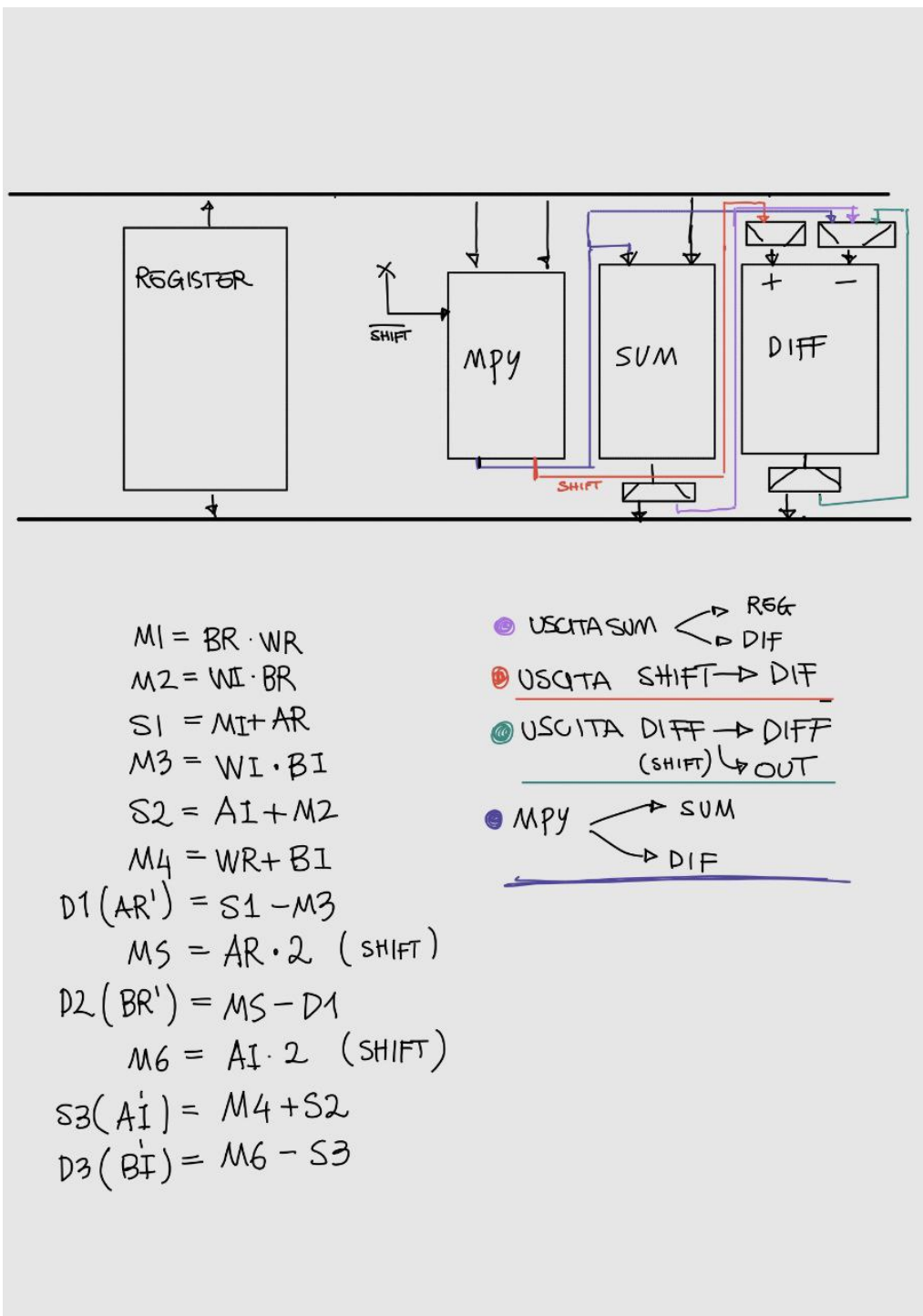


Figura 3: Schema del datapath iniziale

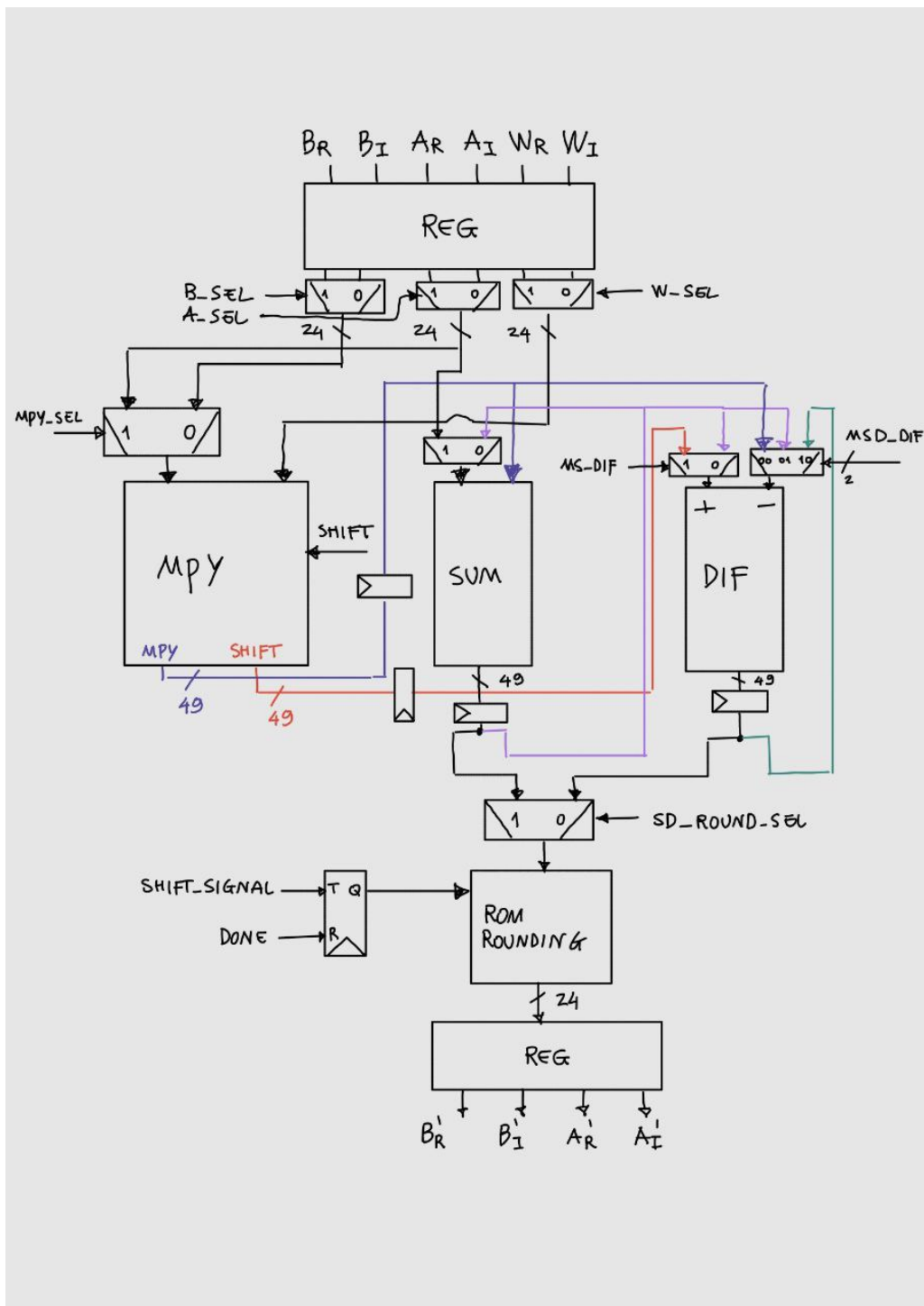


Figura 4: Schema del datapath finale

3.1 ROM Rounding

Come richiesto nella consegna del progetto per avere un uscita nel formato Q1.23 è necessario fare uso del ROM rounding. Questa tecnica consiste nell'arrotondare gli N bit in ingresso al blocco arrotondatore con una look-up-table salvata all'interna di una memoria ROM. Per leggere i dati presenti all'interno della memoria sarà necessario fornire all'ingresso un indirizzo. La scelta del numero di bit dell'indirizzo, e conseguentemente del numero di celle della memoria, è una scelta critica per il progetto in quanto un indirizzo di pochi bit consente di avere una memoria piccola (e che quindi richiede poco spazio su Silicio) ma permette un arrotondamento peggior. Nel caso sia necessario ottenere un arrotondamento più preciso, e quindi con errore minore, allora risulta obbligatorio aumentare il numero di bit di indirizzo per permettere l'indirizzamento di più celle di memoria. Considerando che si volevano salvare 5 bit per riga è stato scelto come numero di bit per l'indirizzo 5. Si riporta di seguito una tabella che mette in relazione il numero di bit dell'indirizzo con il numero di righe dell'ROM e con il numero di bit totali da memorizzare.

bit indirizzo	righe ROM	bit totali
3	8	40
5	32	160
7	128	640
9	512	2560

Tabella 1: Relazione tra il numero di bit di indirizzo della ROM, il numero di righe ed il numero totale di bit memorizzati

Bisogna anche considerare il bias e l'errore medio. Avendo come specifica di progetto l'utilizzo del metodo "Round to Nearest Even" si è dovuto scegliere un indirizzo composto da un numero di bit della mantissa e bit di scarto disposti in maniera tale da minimizzare sia il bias che l'errore. Di seguito si riportano i test effettuati:

bit indirizzo	bias	errore
3		
5		
7		
9		

Tabella 2: Relazione tra il numero di bit di indirizzo della ROM, il bias e l'errore

Dopo aver considerato tutte le opzioni, sia dal lato di area occupata che dal lato bias/errore, è stato scelto di comporre l'indirizzo della ROM con gli ultimi 3 bit della mantissa (LSB mantissa) e con i primi 2 bit dello scarto (MSB scarto). Si riporta in *Fig. 5* lo schema del ROM rounding implementato. Si può apprezzare la presenza della ROM, un registro posto in ingresso e uno in uscita usati per rendere i dati disponibili sul fronte del clock dato che la ROM è puramente combinatoria e, infine, il parallelismo dei bus espresso col numero di fianco al bus stesso.

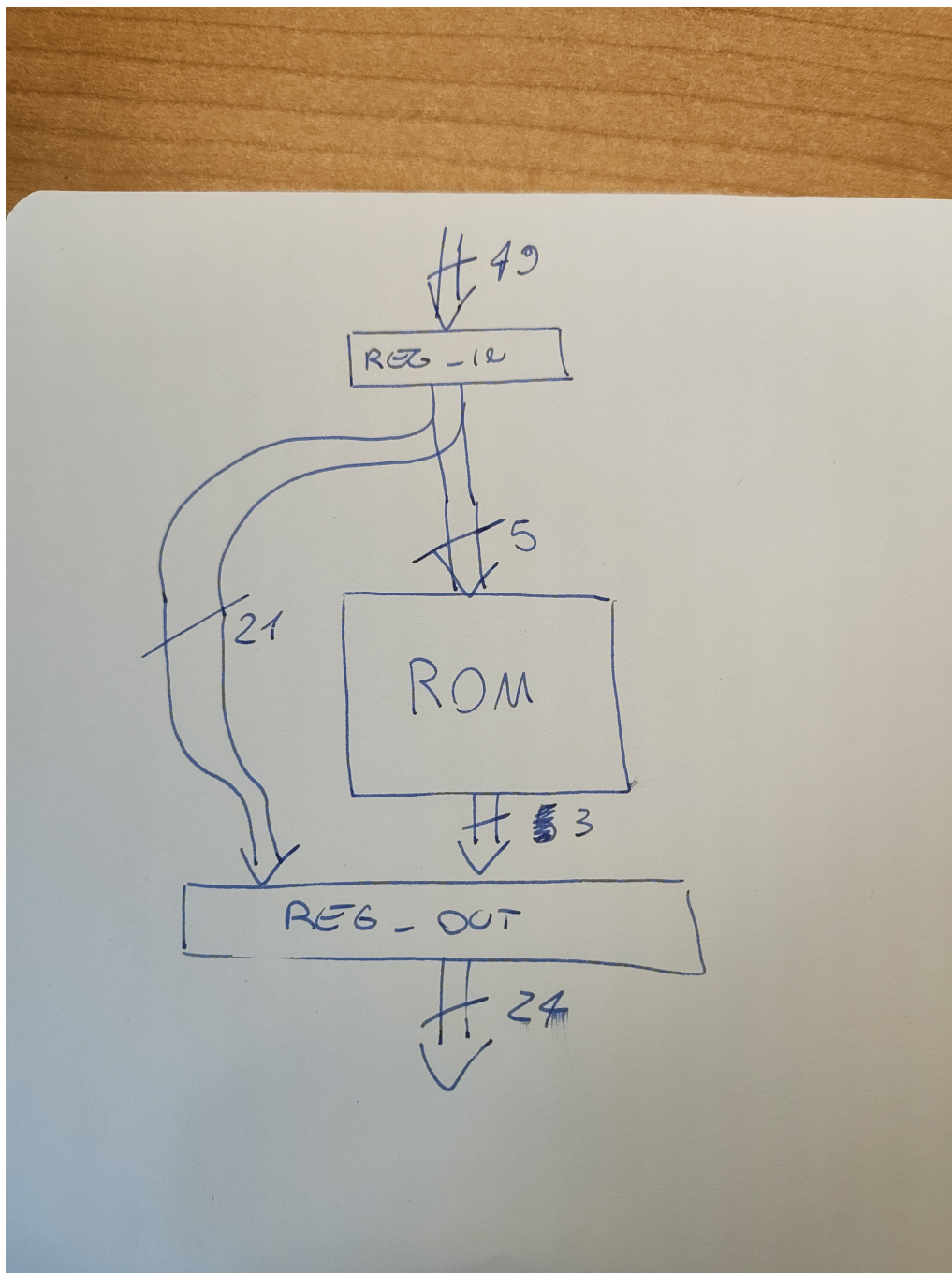


Figura 5: Schema del ROM rounding implementato

4 Control Unit

5 Appendice

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5
6  -- Creazione entity
7  entity rounding is
8      port(
9          Clock: IN      STD_LOGIC; -- Clock
10         rounding_in : IN std_logic_vector(48 downto 0); -- 49 bit da arrotondare
11         rounding_out: OUT std_logic_vector(23 downto 0); -- 24 bit arrotondati
12         shift_signal: IN STD_LOGIC -- segnale per shiftare
13     );
14 end entity;
15
16 -- Architecture del blocco rounding
17 architecture behavioural of rounding is
18
19     -----
20     -- Inizializzazione componenti
21     -----
22     component ROM is
23     port(
24         address : IN std_logic_vector(4 downto 0);
25         memory_out: OUT std_logic_vector(2 downto 0));
26     end component;
27
28     component FD is
29     generic(
30         bus_length: INTEGER:= 24
31     );
32     port ( D:      in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
33           E: in STD_LOGIC;          --ENABLE attivo alto
34           CK:      in      STD_LOGIC;
35           Q:      out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
36     end component;
37
38     -----
39     -- Segnali interni al blocco rounding
40     -----
41
42     signal mantissa : std_logic_vector(23 downto 0):= (others=>'0');
43     signal dummy_memory_out: std_logic_vector(2 downto 0):= (others=>'0');
44     signal address_memory : std_logic_vector(4 downto 0):= (others=>'0');
45     signal reg_in : std_logic_vector(23 downto 0):= (others=>'0');
46     signal bit_scarto : std_logic_vector(1 downto 0):= (others=>'0');
47     signal shift_dummy: std_logic_vector(48 downto 0) := (others=>'0');
48
49 begin
50
51     -----
52     -- Port map
53     -----
54     pm_reg_rom_out : FD
55         generic map (
56             bus_length => 24
57         )
58         port map (
59             D => reg_in,
60             E => '1',
61             CK => Clock,
62             Q => rounding_out
63         );
64
```

```

65     pm_ROM : ROM
66         port map(
67             address => address_memory,
68             memory_out => dummy_memory_out
69         );
70
71
72     -----
73     -- Shift senza processo logico (non impiega colpi di clock)
74     -----
75     shift_dummy <=
76         '0' & '0' & rounding_in(48 downto 2) when shift_signal = '1' else '0' &
            rounding_in(48 downto 1);
77
78     -----
79     -- Creazione mantissa e bit di scarto
80     -----
81
82     mantissa    <= shift_dummy(46 downto 23);
83     bit_scarto  <= shift_dummy(22 downto 21);
84
85     -----
86     -- Creazione dell'indirizzo per leggere dalla ROM
87     -----
88
89     -- address = (3 bit LSB mantissa) + (1 bit MSB scarto) + (1 bit OR con tutti gli
        altri dello scarto)
90     address_memory <= mantissa(2 downto 0) & bit_scarto;
91
92     -----
93     -- Inserimento dati nel registro d'uscita del blocco
94     -----
95
96     reg_in <= mantissa(23 downto 3) & dummy_memory_out; -- 21 bit di mantissa & 3 bit
        arrotondamento
97
98 end architecture behavioural;

```

Listing 1: Rounding_single_clock