UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Architetture dei sistemi dinamici

Esame scritto: progetto di un sistema digitale

Anno Accademico 2023/24

Prof.ssa

Alessandra De Benedictis

candidato
Luca Pisani
matr. M63001627

Indice

Esercizio

Traccia

Progettare, implementare in VHDL e simulare il seguente sistema. Un sistema è composto da tre nodi:

- il nodo A che, tramite handshaking completo, invia al nodo C un codice operativo cop a 2 bit ed un operando ad 8 bit op1;
- il nodo B che, tramite handshaking completo, invia al nodo C un operando ad 8 bit op2 e un valore X;
- il nodo C che contiene un'unità aritmetica in grado di eseguire somme e sottrazioni e una memoria ROM da N locazioni, ciascuna da 8 bit.

La comunicazione fra A e C e fra B e C avviene in parallelo (con handshaking per ogni valore inviato).

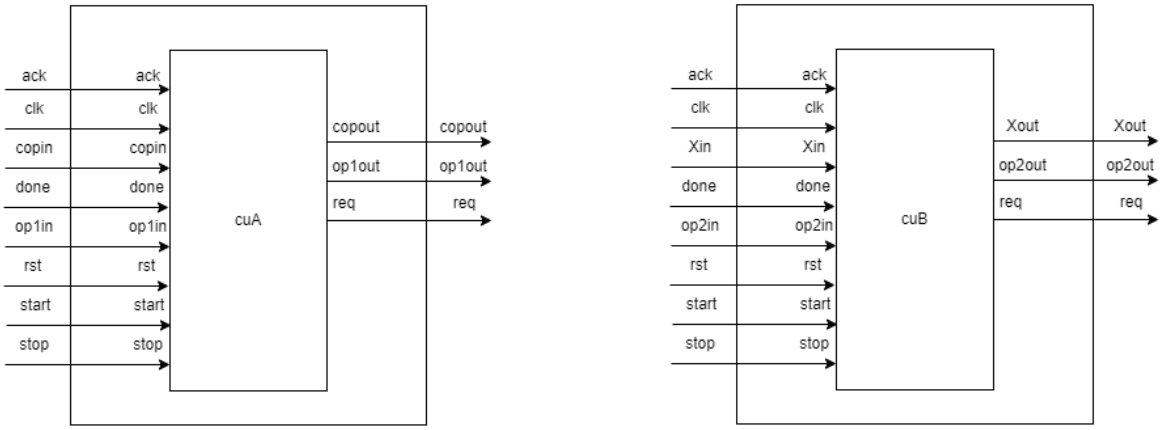
Quando il nodo C riceve dal nodo A i valori cop e op1 e dal nodo B i valori op2 e X, esegue l'operazione richiesta dal cop (per esempio, se cop è 00, esegue la somma): se il risultato dell'operazione è minore o uguale del valore X, il nodo C inserisce il risultato nella sua memoria interna e invia ai nodi A e B un ack per indicare di esser pronto a ricevere un nuovo insieme di dati; se il risultato dell'operazione è maggiore di X, il nodo C memorizza una stringa nulla nella memoria interna e interrompe la comunicazione con i nodi A e B inviando un segnale di stop.

Il sistema complessivo deve essere sviluppato seguendo un approccio strutturale per

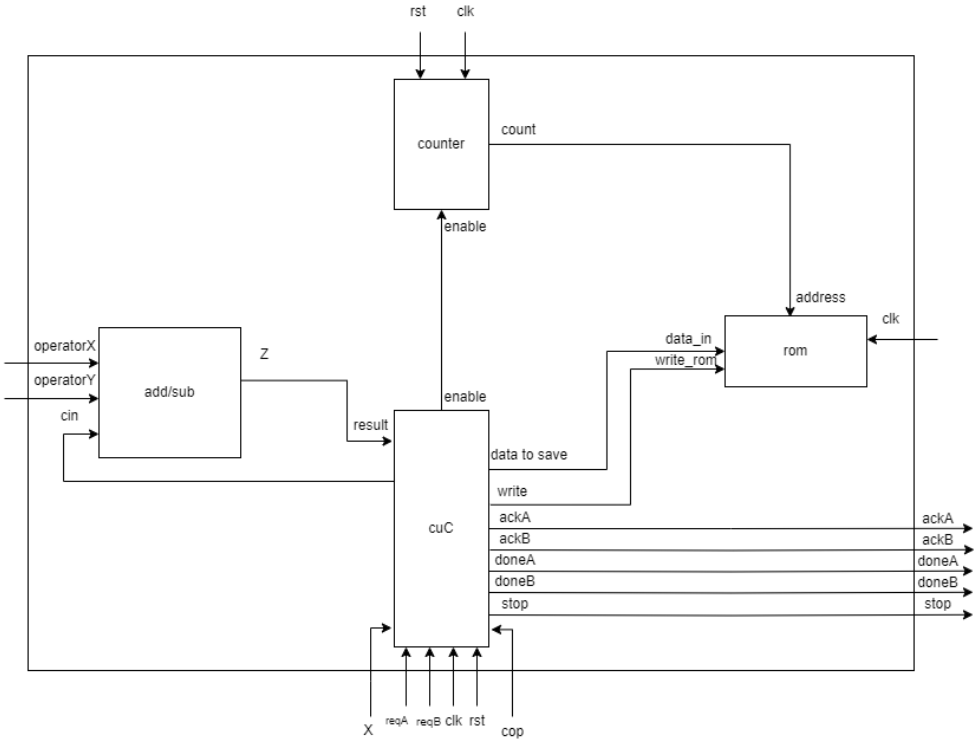
l’interconnessione dei componenti. Ad accezione del sommatore/sottrattore nel nodo C, che deve essere realizzato con approccio strutturale, gli altri componenti possono essere realizzati mediante approccio comportamentale.

1.1.1 Progetto e architettura

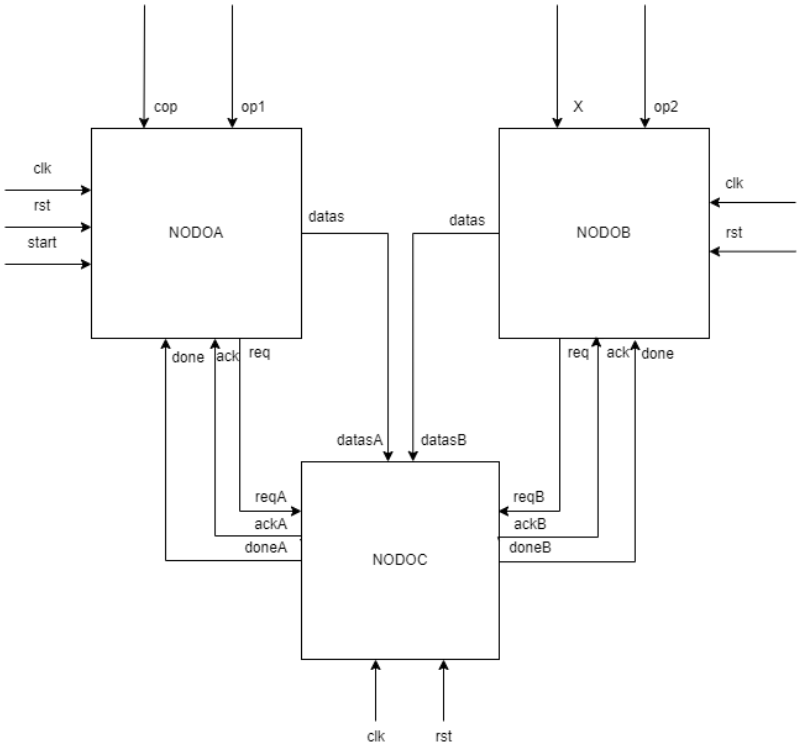
Per realizzare questo design si é deciso di adottare un approccio di tipo strutturale in modo da poter utilizzare diversi componenti ed interconnetterli in modo da formare il sistema finale. La traccia richiede di realizzare tre nodi, A, B e C. A e B non fanno altro che ricevere dall’esterno degli input (comportandosi da registri), in questo caso i valori « cop, op1, X, op2 » e fornire in uscita gli stessi valori nel momento in cui il nodo C é pronto a riceverli.



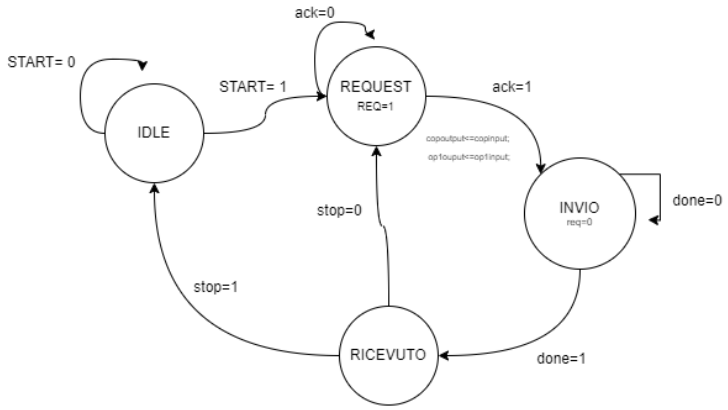
Il nodo C riceverá in ingresso i dati provenienti dai due sistemi (in modo simultaneo) e provvederá ed effettuare le dovute elaborazioni. Esso contiene un sommatore/sottrattore, un contatore modulo m, una control unit ed una memoria Rom. Il nodo riceverá da A e B i due operandi della macchina aritmetica, un valore di confronto X ed un opcode per selezionare somma o sottrazione e salverá eventualmente il risultato nella propria memoria interna.



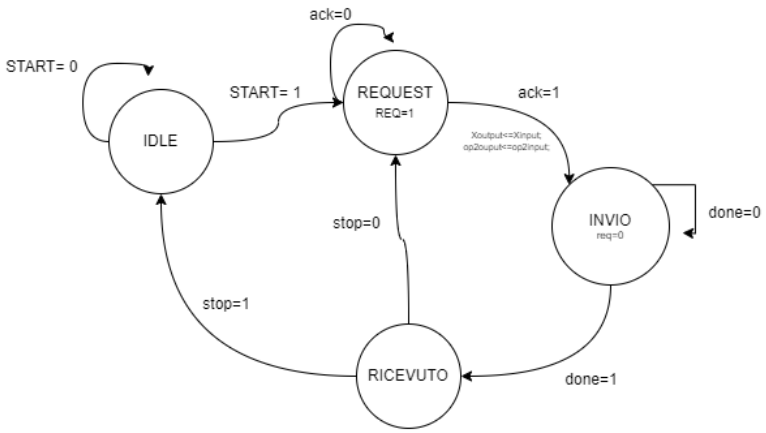
La comunicazione tra i tre nodi viene garantita dal protocollo di Hanshaking (completo): esso prevede lo scambio di specifici messaggi di controllo tra le entitá in modo che ciascuna trasmetta o riceva solo se effettivamente nelle condizioni di farlo.



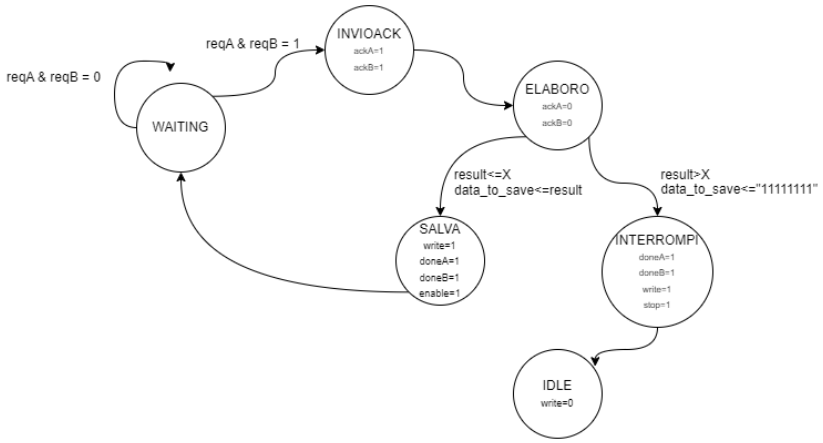
Le control unit dei tre nodi rappresentano l'implementazione dei rispettivi automi a stati finiti dei sistemi :



Per quanto riguarda il nodo A, il sistema permane nello stato « IDLE » fino a quando un segnale start arriva dall'esterno segnalando l'inizio della trasmissione dei dati. Il sistema transita quindi nello stat « REQUEST » in cui invia un primo segnale di request al nodo C segnalando la volontá di voler inviare un dato da elaborare. Il nodo C risponderá con un segnale di Acknowledgment (ack) con il quale segnalerá al nodo A la propria disponibilitá. In caso di ack=1 il sistema transiterá nello stato « INVIO » in cui il dato viene effettivamente trasmesso. Nel momento in cui il sistema C avrá ricevuto il dato ed effettuato le proprie elaborazioni, manderá al sistema A un segnale done=1 che indicherá la disponibilitá nel ricevere un nuovo dato per una nuova elaborazione. Per questa ragione in caso di done=1 si passerá prima allo stato « RICEVUTO » dal quale si ritornerá in « REQUEST » se non sará rilevato alcun segnale di stop con valore logico '1' Il nodo B avrá lo stesso comportamento e implementazione del nodo A.



Per quanto riguarda il nodo C invece, la control unit presenta un comportamento di questo tipo :



Il sistema C sará in uno stato di attesa « WAITING » in cui aspetta che i segnali reqB e reqA in arrivo dai due nodi esterni passino a valore logico alto. In tal caso il sistema transiterá nello stato « INVIOACK » in cui verranno effettivamente inviati i due segnali di ack per i due nodi in modo da segnalare loro la disponibilitá nell'accogliere i dati. Questi verranno quindi trasmessi dai nodi A e B ed elaborati nello stato « ELABORATE ». I segnali di ack vengono azzerati in modo da segnalare ai nodi esterni una elaborazione in corso. I due operandi sono caricati all'intero del sommatore/sottrattore e la control unit invia uncarry cin pari a 0 o 1 in base al valore di cop ricevuto da A. Se il risultato dell'operazione risulta essere minore o uguale del valore X inviato da B allora il sistema transiterá nello stato « SALVA » in cui il risultato sará memorizzato all'interno della ROM del nodo, il contatore sará incrementato in modo da fornire l'indirizzo successivo per memorizzare eventualmente il risultato di una prossima elaborazione e verranno infine inviati i segnali di doneA=1 e doneB=1 ai due nodi in modo da segnalare l'avvenuta elaborazione. I due nodi segnaleranno l'avvenuta lettura di done se dei nuovi request saranno inoltrati al sistema C. Nel caso in cui il risultato dell'elaborazione dovesse essere maggiore del valore X, il sistema si porterá nello stato « INTERROMPI » in cui provvederá a salvare nella propria rom una determinata stringa (é stata scelta la stringa '11111111') e a segnalare l'interruzione della comunicazione con i due nodi attraverso un segnale di stop che passerá da valore logico basso a valore logico alto. Dopodiché il sistema resterá in uno stato di standby « IDLE » in attesa di un reset.

1.1.2 Implementazione

Per quanto riguarda l'implementazione dei vari componenti é stato necessario adottare approcci differenti in base alle diverse esigenze. Il componente **adder/subtractor** é stato implementato per composizione di componenti via via piú semplici (ripple_carry e full_adder) precedentemente presenti in libreria. Allo stesso modo sono stati utilizzati dei componenti precedentemente sviluppati quali : **ROM** N(8) locazioni da 8 bit ciascuna , un **Contatore mod 8** per scandire in ordine le locazioni della memoria. Si procede a presentare l'implementazione delle control unit dei tre sistemi (in ordine A,B e C) realizzate con approccio behavioral (2 process pattern):

```
1
2
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6
7
8  entity cuA is
9    Port (
10   signal start: in std_logic;
11   signal clk: in std_logic;
12   signal rst: in std_logic;
13   signal copinput: in std_logic_vector(1 downto 0);
14   signal op1input: in std_logic_vector(7 downto 0);
15   signal copoutput: out std_logic_vector(1 downto 0);
16   signal op1ouput: out std_logic_vector(7 downto 0);
17
18   signal req: out std_logic;
19   signal ack: in std_logic;
20   signal done:in std_logic;
21
22   signal stop: in std_logic
23    );
24 end cuA;
25
26 architecture Behavioral of cuA is
27 type stato is (idle, request, invio, ricevuto);
28 signal curr_state: stato := idle;
29 signal next_state: stato := idle;
30 begin
31
32 stato_uscita: process(curr_state, start, ack)
33     begin
34 --    copoutput<="00";
35 --    op1ouput<="00000000";
36     req<='0';
37
38     case curr_state is
39         when idle =>
40             if (start = '1') then
41                 next_state <= request;
42             else
43                 next_state <= idle;
44             end if;
45
46         when request =>
47             req<='1';
48             if ack = '0' then
49                 next_state <= request;
50
51             else
52                 copoutput<=copinput;
53                 op1ouput<=op1input;
54                 next_state <= invio;
55              end if;
56
57         when invio =>
58             req<='0';
59             if done = '1' then
60                 next_state <= ricevuto;
61             else
62                 next_state <= invio;
63             end if;
64
65         when ricevuto =>
66             if stop ='1' then
67                 next_state<=idle;
68             else
69                 next_state<=request;
70             end if;
71
72     end case;
73 end process;
74 mem: process(clk)
75     begin
76         if rising_edge(clk) then
77             if rst = '1' then
78                 curr_state <= idle;
79             else
80                 curr_state <= next_state;
81             end if;
82         end if;
83     end process;
84 end behavioral;
```

```
1
2
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6
7
8 entity cuB is
9   Port (
10  clk: in std_logic;
11  rst: in std_logic;
12  start: in std_logic;
13  Xinput: in std_logic_vector(7 downto 0); -- valore confronto, un valore rappresentabile su 8 bit
14  op2input: in std_logic_vector(7 downto 0);
15  Xoutput: out std_logic_vector(7 downto 0);
16  op2ouput: out std_logic_vector(7 downto 0);
17
18  signal req: out std_logic;
19  signal ack: in std_logic;
20  signal done:in std_logic;
21
22  signal stop: in std_logic
23   );
24 end cuB;
25
26 architecture Behavioral of cuB is
27 type stato is (idle, request, invio, ricevuto);
28 signal curr_state: stato := idle;
29 signal next_state: stato := idle;
30 begin
31
32 stato_uscita: process(curr_state, start, ack, done)
33     begin
34 --    Xoutput<="00000000";
35 --    op2ouput<="00000000";
36     req<='0';
37
38     case curr_state is
39         when idle =>
40             if (start = '1') then
41                 next_state <= request;
42             else
43                 next_state <= idle;
44             end if;
45
46         when request =>
47             req<='1';
48             if ack = '0' then
49                 next_state <= request;
50             else
51                 Xoutput<=Xinput;
52                 op2ouput<=op2input;
53                 next_state <= invio;
54              end if;
55
56         when invio =>
57             req<='0';
58             if done = '1' then
59                 next_state <= ricevuto;
60             else
61                 next_state <= invio;
62             end if;
63
64         when ricevuto =>
65             if stop ='1' then
66                 next_state<=idle;
67             else
68                 next_state<=request;
69             end if;
70
71     end case;
72 end process;
73 mem: process(clk)
74     begin
75         if rising_edge(clk) then
76             if rst = '1' then
77                 curr_state <= idle;
78             else
79                 curr_state <= next_state;
80             end if;
81         end if;
82     end process;
83 end behavioral;
```

CU_C

```
2
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6  use IEEE.std_logic_unsigned.ALL;
7
8
9  entity cuC is
10   Port (
11   signal clk: in std_logic;
12   signal rst: in std_logic;
13
14   signal enable: out std_logic;
15   signal write: out std_logic;
16
17   signal result_from_add: in std_logic_vector(7 downto 0);
18   signal data_to_save: out std_logic_vector(7 downto 0);
19   signal cop: in std_logic_vector(1 downto 0);
20   signal cin: out std_logic;
21   signal X: in std_logic_vector(7 downto 0);
22
23   signal reqA: in std_logic;
24   signal ackA: out std_logic;
25   signal doneA:out std_logic;
26
27   signal reqB: in std_logic;
28   signal ackB: out std_logic;
29   signal doneB:out std_logic;
30
31   signal stop: out std_logic
32    );
33 end cuC;
34
35 architecture Behavioral of cuC is
36 type stato is (waiting, invioack, elaboro, salva, interrompi, idle);
37 signal curr_state: stato := waiting;
38 signal next_state: stato := waiting;
39 signal appoggio: std_logic;
40
41 begin
42
43
44  stato_uscita: process(curr_state, reqA, reqB )
45     begin
46 write<='0';
47 cin<='0';
48 ackA<='0';
49 doneA<='0';
50 ackB<='0';
51 doneB<='0';
52 stop<='0';
53 --data_to_save<="00000000";
54 enable<='0';
55     case curr_state is
56         when waiting =>
57             write<='0';
58             data_to_save<="00000000";
59             cin<='0';
60             enable<='0';
61             ackA<='0';
62             doneA<='0';
63             ackB<='0';
64             doneB<='0';
65             stop<='0';
66             if (reqA ='1' AND reqB = '1') then
67                 next_state<=invioack;
68             else
69                 next_state<= waiting;
70             end if;
```

```
        when invioack =>
            write<='0';
            doneA<='0';
            data_to_save<="00000000";
            enable<='0';
            doneB<='0';
            stop<='0';
            write<='0';
            ackA<='1';
            ackB<='1';
            if (cop="00") then
                cin<='0';
            elsif (cop="11") then
                cin<='1';
            end if;
            next_state<=elaboro;

        when elaboro =>
            write<='0';
            write<='0';
            enable<='0';
            cin<='0';
            ackA<='0';
            doneA<='0';
            ackB<='0';
            doneB<='0';
            stop<='0';
            if (conv_integer(result_from_add)<X OR conv_integer(result_from_add)=X) then
                data_to_save<=result_from_add;

                next_state<=salva;
            else
                data_to_save<="11111111";

                next_state<= interrompi;
            end if;

        when salva =>
            write<='1';
            cin<='0';
            ackA<='0';
            ackB<='0';
            stop<='0';
            doneA<='1';
            doneB<='1';

            enable<='1';
            appoggio<='1';
            next_state<=waiting;

        when interrompi =>
            write<='1';
            stop<='1';
            enable<='0';
            cin<='0';
            ackA<='0';
            ackB<='0';
            stop<='0';
            doneA<='1';
            doneB<='1';
            next_state<=idle;

        when idle=>
            write<='0';
            next_state<=idle;

        end case;
end process;

mem: process(clk)
   begin
        if rising_edge(clk) then
            if rst = '1' then
                curr_state <= waiting;
            else
                curr_state <= next_state;
            end if;
        end if;
    end process;

end behavioral;
```

Di seguito le implementazioni dei rispettivi nodi A, B e C :

NODO A

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7  entity NODOA is
8    Port (
9    signal clk: in std_logic;
10   signal rst: in std_logic;
11   signal start: in std_logic;
12
13   signal cop: in std_logic_vector(1 downto 0);
14   signal op1: in std_logic_vector(7 downto 0);
15
16   signal cop_to_C: out std_logic_vector(1 downto 0);
17   signal op1_to_C: out std_logic_vector(7 downto 0);
18
19   signal req: out std_logic;
20   signal ack: in std_logic;
21   signal done:in std_logic;
22
23   signal stop: in std_logic
24    );
25 end NODOA;
26
27 architecture structural of NODOA is
28
29 begin
30
31 CUA: entity work.cuA port map(
32     clk=>clk,
33     rst=>rst,
34     start=>start,
35     copinput=>cop,
36     op1input=>op1,
37     copoutput=>cop_to_C,
38     op1ouput=>op1_to_C,
39
40     req=>req,
41     ack=>ack,
42     done=>done,
43
44     stop=>stop
45 );
46
47 end structural;
48
```

NODO B

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7  entity NODOB is
8    Port (
9    signal clk: in std_logic;
10   signal rst: in std_logic;
11   signal start: in std_logic;
12
13   signal X: in std_logic_vector(7 downto 0);
14   signal op2: in std_logic_vector(7 downto 0);
15
16   signal X_to_C: out std_logic_vector(7 downto 0);
17   signal op2_to_C: out std_logic_vector(7 downto 0);
18
19   signal req: out std_logic;
20   signal ack: in std_logic;
21   signal done:in std_logic;
22
23   signal stop: in std_logic
24    );
25 end NODOB;
26
27 architecture structural of NODOB is
28
29 begin
30
31 CUB: entity work.cuB port map(
32     clk=>clk,
33     start=>start,
34     rst=>rst,
35     Xinput=>X,
36     op2input=>op2,
37     Xoutput=>X_to_C,
38     op2ouput=>op2_to_C,
39
40     req=>req,
41     ack=>ack,
42     done=>done,
43
44     stop=>stop
45 );
46
47 end structural;
48
```

NODO C

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity NODOC is
  Port (

    signal clk: in std_logic;
    signal rst: in std_logic;

    --dati
    signal cop: in std_logic_vector(1 downto 0);
    signal op1: in std_logic_vector(7 downto 0);
    signal X: in std_logic_vector(7 downto 0);
    signal op2: in std_logic_vector(7 downto 0);

    --hs
    signal reqA: in std_logic;
    signal ackA: out std_logic;
    signal doneA: out std_logic;
    signal reqB: in std_logic;
    signal ackB: out std_logic;
    signal doneB: out std_logic;

    --stop
    signal stop: out std_logic

   );
end NODOC;

architecture structura of NODOC is
signal result_to_cu: std_logic_vector(7 downto 0);
signal write_rom_temp: std_logic;
signal counter_to_rom: std_logic_vector(2 downto 0);
signal finaldata: std_logic_vector(7 downto 0);
signal counter_enable_temp: std_logic;
signal cintemp: std_logic;


begin

CUC: entity work.cuC port map(
    clk=>clk,
    rst=>rst,
    X=>X,
    enable=>counter_enable_temp,
    write=>write_rom_temp,
    result_from_add=>result_to_cu,
    data_to_save=>finaldata,
    cop=>cop,
    cin=>cintemp,

    reqA=>reqA,
    ackA=>ackA,
    doneA=>doneA,

    reqB=>reqB,
    ackB=>ackB,
    doneB=>doneB,

    stop=>stop
);

ADDSUB: entity work.adder_sub port map(
    X=>op1,
    Y=>op2,
    cin=>cintemp,
    Z=>result_to_cu
);

ROM: entity work.rom port map(
    clk=>clk,
    write_rom=>write_rom_temp,
    address=>counter_to_rom,
    data_in=>finaldata
);

COUNTER: entity work.counter_mod8 port map(
    clk=>clk,
    reset=>rst,
    enable=>counter_enable_temp,
    count=>counter_to_rom

);



end structura;

```

Infine si presenta il top module :

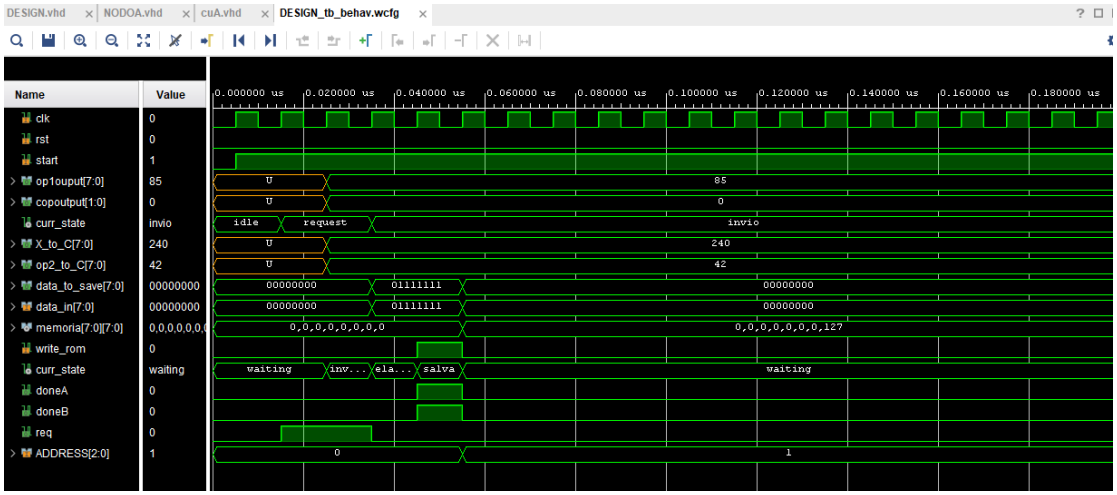
DESIGN

```
1
2
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity DESIGN is
7   Port (
8     signal clk: in std_logic;
9     signal rst: in std_logic;
10    signal start: in std_logic;
11
12    signal cop: in std_logic_vector(1 downto 0);
13    signal op1: in std_logic_vector(7 downto 0);
14    signal X: in std_logic_vector(7 downto 0);
15    signal op2: in std_logic_vector(7 downto 0)
16
17
18   );
19 end DESIGN;
20
21 architecture structural of DESIGN is
22
23 signal acktempA: std_logic;
24 signal reqtempA: std_logic;
25 signal donetempA: std_logic;
26 signal acktempB: std_logic;
27 signal reqtempB: std_logic;
28 signal donetempB: std_logic;
29 signal stoptemp: std_logic;
30
31 signal coptemp: std_logic_vector(1 downto 0);
32 signal op1temp: std_logic_vector(7 downto 0);
33 signal Xtemp: std_logic_vector(7 downto 0);
34 signal op2temp: std_logic_vector(7 downto 0);
35
36 begin
37
38 NODOA: entity work.nodoA port map(
39     clk=>clk,
40     rst=>rst,
41     start=>start,
42     cop=>cop,
43     op1=>op1,
44     cop_to_C=>coptemp,
45     op1_to_C=>op1temp,
46
47     ack=>acktempA,
48     req=>reqtempA,
49     done=>donetempA,
50     stop=>stoptemp
51
52 );
53
54 NODOB: entity work.nodoB port map(
55     clk=>clk,
56     rst=>rst,
57     start=>start,
58     X=>X,
59     op2=>op2,
60     X_to_C=>xtemp,
61     op2_to_C=>op2temp,
62
63     ack=>acktempB,
64     req=>reqtempB,
65     done=>donetempB,
66     stop=>stoptemp
67 );
68
69 NODOC: entity work.nodoC port map(
70     clk=>clk,
71     rst=>rst,
72
73     cop=>coptemp,
74     op1=>op1temp,
75     X=>xtemp,
76     op2=>op2temp,
77
78     ackA=>acktempA,
79     reqA=>reqtempA,
80     doneA=>donetempA,
81
82     ackB=>acktempB,
83     reqB=>reqtempB,
84     doneB=>donetempB,
85     stop=>stoptemp
86     );
87
88
89
90 end structural;
91
```

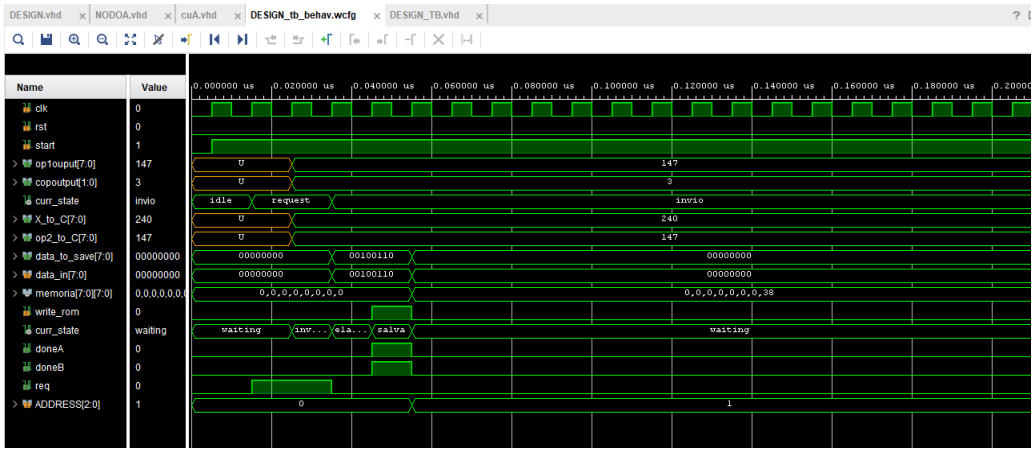
1.1.3 Simulazione

Attraverso un componente testbench é stato simulato il comportamento del design complessivo in diverse casistiche :

1- Caso op1+op2 < M



2- Caso op1-op2 < M



3- Caso op1+op2 > M