Mini Router

Damiano Barone

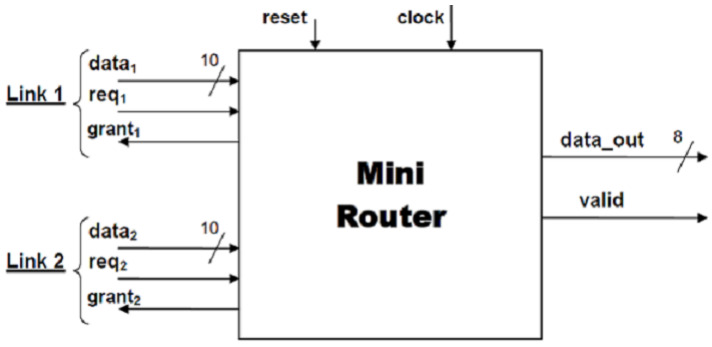
January 4, 2015

Contents

Chapter 1

Introduzione

Si richiede di realizzare la descrizione VHDL di un mini router, che presenta in uscita i dati proveniente da uno dei due link. Un link presenta un dato valido se ha req a 1. Ogni link ha un campo data di 10 bit dove i 2 bit piu' significativi rappresentano la priorità. Nel caso in cui ci siano due richieste contemporanee si serve prima quello con priorità maggiore, se hanno la stessa priorita si utilizza l'algoritmo di Round Robin. Quando viene scelto un link il grant corrispondete e valid vanno ad 1 per un ciclo di clock. In uscita vengono inviati solo gli 8 bit meno significati di data e quindi vengono troncati quelli della priorità.



Chapter 2

Possibili scenari

Durante il funzionamento del mini router possono avvenire vari scenari . Di seguito elenchero tutti possibili scenari e come sono stati gestiti.

2.1 Una sola richiesta

Nel caso che arriva una sola richiesta da uno dei due link, il router accetta la richiesta, mette ad 1 grant e valid e collega l'input con l'ouput, però senza i bit di priorità. Al clock successivo rimetterà il grant e valid a 0. Si nota che quando rimetterà grant e valid a 0 potrebbe accadere che arriva una richiesta dall'altro link, in questo caso valid rimarebbe a 1 e verrebbe gestita l'altra richiesta.

2.2 Due richieste contemporanee

Nel caso di due richiesta contemporanee con:

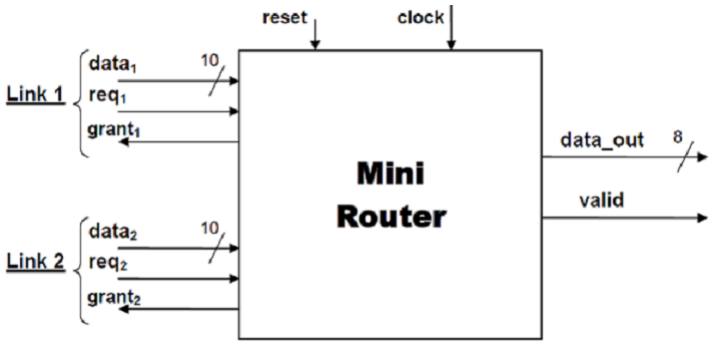
- Priorità diverse: si gestisce prima il link con priorità maggiore e al ciclo successivo l'altro.Si nota che può accadere che alla gestione della seconda richiesta, il link servito per prima può fare una nuova richiesta, in questo caso valid non lo porto a 0 ma gestisco la richiesta.
- Priorità uguali: si gestice con l'algoritmo del Round Robin, la prima volta viene servito il link1 e la volta successiva che ricapita verrà servito il link2 e cosi via...

Chapter 3

Descrizione dell'architettura

3.1 Diagramma a blocchi

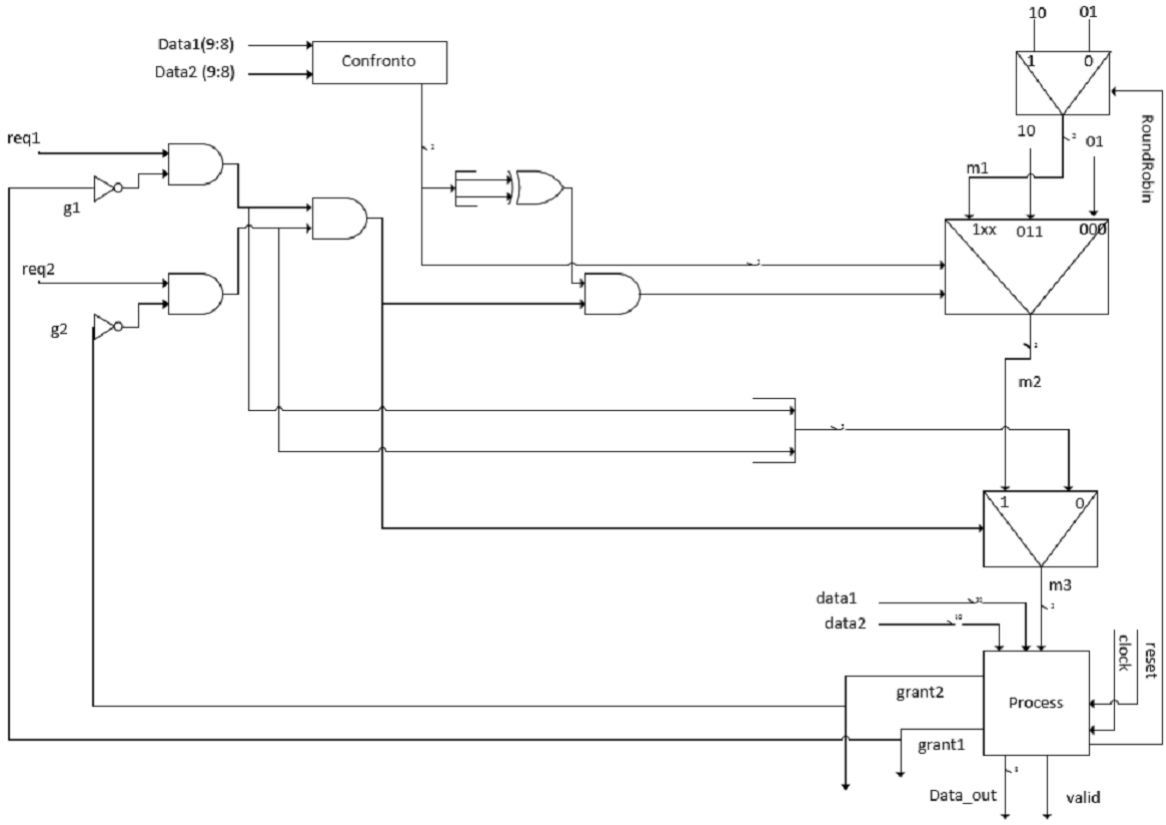
Dalla figura si può notare la vista esterna del router. Il campo data sono i



dati di input di 10 bit di cui 2 bit per la priorità, req sono i bit che dicono se una richiesta è valida, in uscita del router si hanno i grant che completano handshake con i relativi link che notificano che la richiesta è stata inoltrata. Inoltre si ha dataout che manda in uscita il collegamento data ma con il troncamento dei bit di priorità, in più c'è valid che indica a valle che i dati sono validi.
La vista interna del mini router è composta da una rete combinatoria. Si è notato che il router deve fare tre operazioni fondamentali:1)servire il link1, 2)servire il link2 oppure non fare niente cioè avere tutti gli output a 0. La rete combinatoria identifica le tre operazioni con questo comportamento: Una richiesta di un link si può definire valida se ha req=1 e grant=0, questo perchè se grant è ad 1 significa che req è ancora 1 perchè la componente a monte non si è ancora accorta che è stato consegnato il suo dato, al clock successivo se req è ancora uguale ad 1 significa che gli è stato notificato, ma vuole

proporre un altro dato. Nella rete è presente un rete combinatoria detta confronto, questa rete fa il confronto delle priorità se uno dei due è più grande restituisce 00 o 11 invece se hanno la stessa priorità restituisce 10. Grazie alla XOR si identifica se hanno la stessa priorità. L'uscita della XOR si collega in ingresso ad una AND che ha come altro ingresso un collegamento che vale 1 se ci sono due richieste contemporanee valide altrimenti 0. In questo modo si pilota il multiplexer che decide se mettere in uscita il dato del multiplexer del RoundRobin o la codifica del link con priorità maggiore. Il multiplexer successivo (m3) decide se inserire in ingresso al process il valore dei multiplexer precedenti (caso di richiesta insieme) oppure una richiesta singola di uno dei due link, codificati in "01" per il link1 e "10" per il link 2. Infine il processo grazie ha questi 2 bit in ingresso decide se stare in pausa (grant e valid a 0) oppure presentare i dati in output e mettere ad 1 valid e il rispettivo grant. Round Robin viene aggiornato dal process nel caso in cui ci siano due richieste con la stessa priorità. Si nota che grant1 e grant2 in uscita dal processo si sdoppiano per rientrare nella rete combinatoria.

Di seguito è riportata la rete combinatoria del router:



3.2 Workbench

Per testare l'implementazione dell'algoritmo è stata scritta una rete di workbench e due file contenente dei dati di input. La rete di workbench è stata realizzata attraverso un processo che legge dati da read1 e read2 che rappresentano i due link e scrive l'output su un file chiamato write.txt. Ogni read.txt ha varie righe dove ogni riga è composta dal bit req e il dato in decimale che verrà convertito in 10 bit. Si nota che i 2 bit della priorità sono nella parte più significatica. I file sono stati preparati in maniera tale da simulare uno scenario di funzionamemento del router. Sono stati provati tutti i casi possibili di evoluzione della rete. Si nota che un link non cambia il dato finchè non vede che grant è andato a 1. In più è stato creato un file in cpp che semantiamente è uguale al mini router in vhdl. Il programma cpp legge sempre da due file e stampa l'output in un file. Dopo l'esecuzione del programma è stato fatto il confronto (comando fc dal terminale di windows) tra i due file di output uno generato con vhdl e l'altro in cpp ed è risultato che i 2 file sono identici.

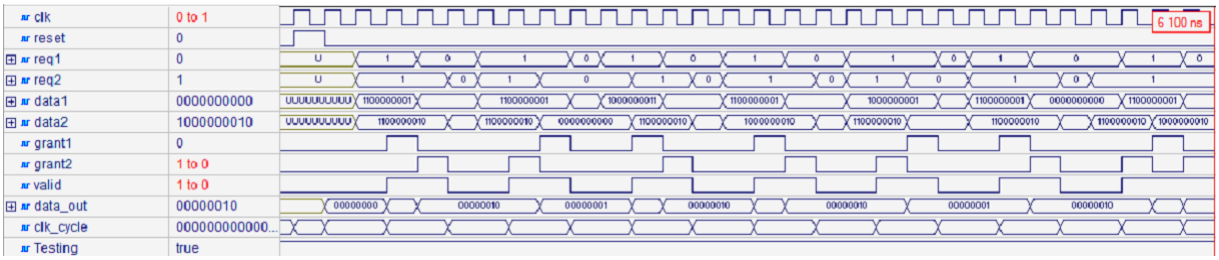
3.3 File di test

I file read1.txt e read2.txt sono compositi da 28 righe dove simulano più scenari della rete. Di seguito riporto cosa è stato testato:

- Riga 1-4 Richiesta contemporanee con stessa priorità usa l'algoritmo RoundRobin (verrà scelto il link1);
- Riga 5-8 Richiesta contemporanee con stessa priorità usa l'algoritmo RoundRobin (verrà scelto il link2);
- Riga 9-10 Una sola richiesta (link1);
- Riga 10-11 Una sola richiesta (link2);
- Riga 12 Pausa (si aspetta finisca handshake con il link2);
- Riga 13-16 Richiesta contemporanee con priorità diverse (link1 maggiore);
- Riga 17-19 Richiesta contemporanee con priorità diverse (link2 maggiore);
- Riga 21-24 Richiesta contemporanee con stessa priorità usa l'algoritmo RoundRobin (verrà scelto il link1);

- Richiesta 25-28 richieste casuali.

Tramite l'evoluzione della temporizzazione e il confronto con i due output (uno fatto in vhdl e l'altro in cpp) si ha un certa sicurezza che l'implementazione dell'algoritmo sia giusta.
Di seguito si può notare la temporizzazione dello scenario descritto in precedenza:



Chapter 4

Codice

4.1 Mini Router

```
1  library IEEE;
   use IEEE.STD_LOGIC_1164.all;
3  entity progetto_Mini_Router is

5  generic (N : INTEGER:=10);
   port(
7          clk : in std_logic;
           reset : in std_logic;
9          -------- LINK 1--------
           data1 : in std_logic_VECTOR (N-1 downto 0);
11         req1 : in std_logic;       --request1
           grant1 : out std_logic :='0';
13         -------- LINK 2--------
           data2 : in std_logic_vector(N-1 downto 0);
15         req2 : in std_logic;       --request2
           grant2 : out std_logic :='0';
17         --------OUTPUT a valle--------
           valid  : out std_logic :='0';
19         data_out : out std_logic_vector(N-3 downto 0)); --not have bit of priority

21 end progetto_Mini_Router;

23 architecture progetto_Mini_Router of progetto_Mini_Router is

25  SIGNAL  g1 : std_logic:='0';      --return grant1 to input
    SIGNAL  g2 : std_logic:='0';      --return grant2 to input
27  SIGNAL  confronto : std_logic_vector(1 downto 0);--output of combinational
        network confronto
    SIGNAL  R1_valid : std_logic:='0';    --output of AND between r1 and grant1
29  SIGNAL  R2_valid : std_logic:='0';    --output of AND between r2 and grant2
```

```
SIGNAL Multiplexer1_2: std_logic;     --second input of variable control of
    multiplexer2
SIGNAL Multiplexer2_3: std_logic;     --input of variable control of multiplexer3
SIGNAL RoundRobin : std_logic:='0';
SIGNAL M1 : std_logic_vector(1 downto 0);  --output multiplexer1
SIGNAL M2 : std_logic_vector(1 downto 0);   --output multiplexer2
SIGNAL M3 : std_logic_vector(1 downto 0);   --output multiplexer3
BEGIN
grant1<=g1;          --connect g1 to grant1
grant2<=g2;          --connect g2 to grant2
confronto <= "00" when data1(9 downto 8) > data2(9 downto 8) else -- check
    priority
          "01" when data1(9 downto 8) = data2(9 downto 8) else
          "11" ;
R1_valid <= (NOT g1) AND req1;      --the request1 is valid when req1=1 e
    grant1=0
R2_valid <= (NOT g2) AND req2;      --the request2 is valid when req2=1 e
    grant2=0
Multiplexer2_3 <= R1_valid AND R2_valid;   -- control variable of a multiplexer3
Multiplexer1_2 <= Multiplexer2_3 and (confronto(0) XOR confronto(1));  --
    control variable of a multiplexer2
M1 <= "01" WHEN RoundRobin='0' else "10" ;      --output multiplexer1
M2 <= "01" WHEN (Multiplexer1_2 & confronto)= "000" ELSE   --output
    multiplexer2
     "10" WHEN(Multiplexer1_2 & confronto)="011" ELSE M1;
M3 <= M2 WHEN Multiplexer2_3 ='1' else R2_valid & R1_valid; --output
    multiplexer3
  start:PROCESS(clk)
  BEGIN
   IF (clk'EVENT AND clk='1') THEN
       IF (reset='1') then   --RESET
        g1 <= '0';
        g2<='0';
        valid<='0';
        data_out<="00000000";
     ELSE
        CASE M3 IS
           WHEN "00" =>      --case not have new input, all outputs 0
              g1 <= '0';
              g2<='0';
              valid<='0';
           WHEN "01" =>       --case data1 is connect to output
              data_out <= data1(7 downto 0);  --connect data_out with data1
              g1<= '1';       --update grant1
              g2<='0';        --in case the previous clock was served link2
              valid<='1';       --data_out is valid
           WHEN "10" =>       -- case data2 is connect to output
              data_out <= data2(7 downto 0);  --connect data_out with data2
              g1 <= '0';       --in case the previous clock was served link1
```

```
                g2<='1';        --update grant1
                valid<='1';     --data_out is valid
            WHEN others => null;
        END CASE;
        if(((CONFRONTO(0) XOR CONFRONTO(1))= '1')AND R1_valid='1'
            AND R2_valid='1') then --update control variable of round robin
                roundrobin <= not roundrobin;          --when have two request with
                    equal priority
        END IF;
    END IF;
  END IF;
  end process start;
end progetto_Mini_Router;
```

4.2 Workbech

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use STD.textio.all;
use IEEE.numeric_std.all;
USE ieee.std_logic_arith.ALL;
use ieee.std_logic_unsigned.all;
entity test_minirouter is
end test_minirouter;
architecture test_minirouter of test_minirouter is

  COMPONENT progetto_Mini_Router

  generic (N : INTEGER:=10);

  port(clk : in std_logic;
    reset : in std_logic;
    -------INPUT LINK 1-------
    data1 : in std_logic_VECTOR (N-1 downto 0);
    req1 : in std_logic;
    grant1 : out std_logic :='0';
    -------INPUT LINK 2-------
    data2 : in std_logic_vector(N-1 downto 0);
    req2 : in std_logic;
    grant2 : out std_logic :='0';
    -------OUTPUT-------
    valid  : out std_logic :='0';
    data_out : out std_logic_vector(N-3 downto 0));

  END COMPONENT;
  CONSTANT N : INTEGER := 10; -- Bus Width
```

```
31  CONSTANT MckPer : TIME := 200 ns; -- Master Clk period
    CONSTANT TestLen : INTEGER := 50; -- No. of Count (MckPer/2) for test
33  SIGNAL clk : std_logic := '0';
    SIGNAL reset: std_logic := '0';
35  SIGNAL req1 :std_logic_vector(0 downto 0);
    SIGNAL req2 :std_logic_vector(0 downto 0);
37  SIGNAL data1:std_logic_vector(N-1 downto 0);
    SIGNAL data2:std_logic_vector(N-1 downto 0);
39
    -----signal output-----
41  SIGNAL grant1 :std_logic;
    SIGNAL grant2 : std_logic;
43  SIGNAL valid :std_logic;
    SIGNAL data_out: std_logic_vector(N-3 downto 0);
45  SIGNAL clk_cycle : INTEGER;
    SIGNAL Testing: Boolean := True;
47
    BEGIN
49
    I : Progetto_Mini_Router GENERIC MAP(N=>10)
51          PORT MAP(clk,reset,data1,req1(0),grant1,data2,req2(0),grant2,valid,
                data_out);
    -- Generates clk
53  clk <= NOT clk AFTER MckPer/2 WHEN Testing ELSE '0';
    Test_Proc: process(clk)
55  variable temp : integer;

57  variable buf_in,buf_out: LINE;  --buffers used to read and write on file
    variable count: INTEGER:= 0;
59  file test_vectors1 : text open read_mode is "C:/My_Designs/miniRouter/read1.txt
        "; --oper read1.txt
    file test_vectors2 : text open read_mode is "C:/My_Designs/miniRouter/read2.txt
        "; --oper read2.txt
61  file file_pointer : text open write_mode is "C:/My_Designs/miniRouter/write.txt"
        ; --create write.txt
    begin
63  IF (clk'EVENT AND clk='1') THEN
      clk_cycle <= (count+1)/2;
65    case count is
        when 0 => reset <= '1';
67      when 1 => reset <= '0';
        when (TestLen -1) => Testing <= FALSE;
69      when others =>
        if ((not(endfile(test_vectors1)))and (not (endfile(test_vectors2)))) then
71      --link1
        readline(test_vectors1, buf_in);    --read line of read1.text
73      read(buf_in, temp);       --read req1
        req1 <=std_logic_vector(to_signed(temp,1));
75      read(buf_in, temp);       --read data1
```

```
            data1<=std_logic_vector(to_signed(temp,10));
77          --link2
            readline(test_vectors2, buf_in);    --read line of read2.text
79          read(buf_in, temp);       --read req2
            req2 <=std_logic_vector(to_signed(temp,1));
81          read(buf_in, temp);       --read data2
            data2<=std_logic_vector(to_signed(temp,10));
83          write(buf_out,"valid:");      --write valid
            write(buf_out,conv_integer(valid));
85          write(buf_out,"␣data_out:");
               write(buf_out,conv_integer(data_out));   --write data_out
87          write(buf_out,"␣grant1:");
            write(buf_out,conv_integer(grant1));   --write grant1
89          write(buf_out,"␣grant2:");
             write(buf_out,conv_integer(grant2));    --write grant2
91           writeline(file_pointer,buf_out) ;
          end if;
93      end case;
        count:= count + 1;
95    end if;
    end process Test_Proc;
97  end test_minirouter;
```

4.3 Codice cpp

Si nota che nel codice cpp ho dovuto stampare i dati nella lettura successiva per far combiaciare il file di output con quello fatto in vhdl.

```
1   #include <iostream>
    #include <fstream>
3   using namespace std;
    //input sono req1,req2,data1,data2,reset
5   //output sono valid, data_out, grant1, grant2,;
    unsigned char req1=0;
7   unsigned char req2=0;
    unsigned short int data1=0;
9   unsigned short int data2=0;
    unsigned int grant1=0;
11  unsigned int grant2=0;
    unsigned int valid=0;
13  unsigned char rr=0;
    unsigned short int data_out=0;
15  string cestino;
    ofstream desoutput;
17  void stampa()
    {
19      desoutput <<"valid:␣"<< valid ;
```

```
    desoutput<<"␣data_out:␣" << data_out;
    desoutput<<"␣grant1:␣" <<grant1;
    desoutput <<"␣grant2:␣"<< grant2<<"\n" ;
}
void assegna0()
{
    stampa();   //pausa, not new input
    grant1=0;
    valid=0;
    grant2=0;

}
void assegna1()
{
    stampa();
    data_out=data1 & 0x00FF ; //cut bit priorities and connect data1 to output
    grant1=1;    //update the bit of handshake
    valid=1;
    grant2=0;
}
void assegna2()
{
    stampa();
    data_out=data2 & 0X00FF; //cut bit priorities and connect data2 to output
    grant2=1;   //update the bit of handshake
    valid=1;
    grant1=0;
}
int main ()
{

    ifstream desinput1("input1.txt"); //open files
    ifstream desinput2("input2.txt");
    desoutput.open("output.txt");
    stampa();
    for (int i=0; i<27;i++)
    {
        desinput1>>req1;  //read request1
        desinput2>>req2;  //read request2
        desinput1>>data1;  //read data1
        desinput2>>data2;  //read data2
        desinput1>>cestino;  //read a comment
        desinput2>>cestino;  //read a comment
    if (((req1=='1') && !grant1)&&(req2=='1' && !grant2)) // two request
    {
        if (data1>>8 > data2>>8) //link1 have priority
                assegna1();
        else
        {
```

```
69            if (data1>>8 == data2>>8) //same priority
                {
71                  if (rr==0)
                    {
73                      assegna1(); //turn of link1
                        rr=1;  //update varible of round robin
75                  }
                    else //turn of link2
77                  {
                        assegna2();
79                      rr=0; //update varible of round robin
                    }
81              }
            else
83              assegna2();
        }
85  }
    else
87  {
        if ((req1=='1')&& !grant1) //only request of link1
89          assegna1();
        else
91      {
            if((req2=='1') && !grant2) //only request of link2
93              assegna2();
            else
95              assegna0();    //pause, has not valid request
        }
97  }
}
99 desinput1.close();   //close files
desinput2.close();
101 desoutput.close();
}
```