



UNIVERSITÀ DI PISA

Computer Engineering

Electronics and Communication Systems

Mini Router Report

Student: Federico Montini

Academic Year: 2021/2022

Chapter 1 – Introduction

1.1 Problem Description

The Mini Router is a synchronous component which have two input links, a link is composed by three elements:

1. data: Which contains the data to propagate in the 8 MSB and the priority in the remaining 2 bits
2. req: When this variable is equal to '1' the value inside 'data' is valid
3. grant: Required to notify to the sender of the data that the propagation has been completed. Setting it to '1' for one clock cycle result in completing the handshake.

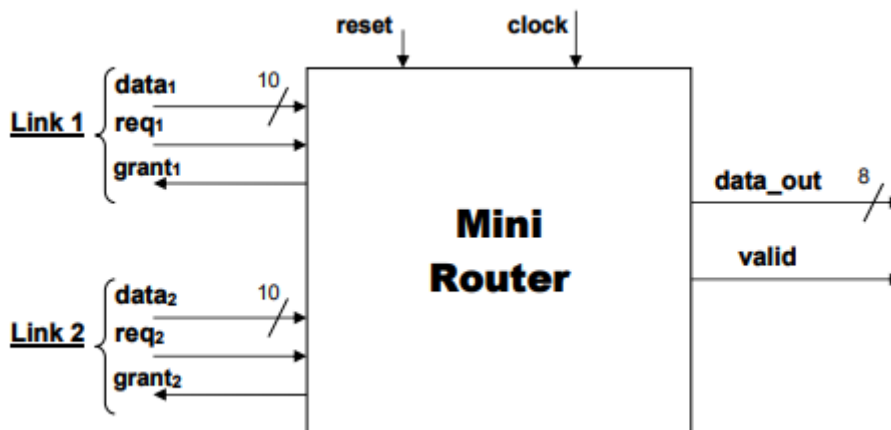


Figure 1: Block representation of a Mini Router

The Mini Router's goal is to propagate the data received on its link: propagate 'data1' to 'data_out' if 'req1' is set or 'data2' if 'req2' is set.

According to the specifications, if the req1 and req2 are both set, the choice will be to choose the link to propagate by a priority principle provided by the last two bits of the 'data' channel. Those two bits can assume four different values: '00', '01', '10', '11' and each one represents a level of priority that goes from 0 (min. priority) to 3 (max priority).

In the case that both 'req' are set and the priority is the same the choice is to propagate one link chosen by the Round-Robin algorithm so:

1. first conflict -> propagate data1
2. second conflict -> propagate data2

3. third conflict -> propagate data1
4. and so on...

To notify the validity of the output the Mini Router has another output link called 'valid'.

It's important to notice that the output will be valid for just one clock cycle and the data must be propagated without the priority bits.

1.2 Applications

The main application for a mini router is the hardware implementation for a priority check and its resolution. This may help the programmer and the processing unit to reduce the overhead during the computations leaving more time to the CPU to elaborate the data.

We can also imagine, in the case of more than two input links, more than one mini router in series or in parallel, but this configuration won't work due to a missing 'grant' in input from the previous mini routers.

This implementation cannot be used to solve the routing at all, the problem is that a router has more than one input links and output links in a way that he can redirect the signal to the correct destination. Our mini router has only one output link, so he cannot redirect the packages.

1.3 Possible Architectures

There are two possible architectures:

1. A pure combinatory architecture that at each clock cycle gets the input choose the output and it get propagate on the subsequent clock rising edge. This solution may have many problems, such as the fact that we don't know at what our mini router will be attached to. The input data must be ready $t_{p-logic} + t_{hold}$ before the clock rising due to the propagation in a combinatory circuit and the time that the register need to see a stable value.
2. Try to reduce the critical path putting some registers at the input links to memorize the data received and leaving the combinatory logic between the registers of input/output to manage and optimize the whole path. The combinatorial logic between the register is what develop the logic of the mini router itself. With this architecture the mini router will be more reliable.

For these reasons I choose to develop and exploit the second option.

Chapter 2 – Architecture Description

The architecture that I chosen to implement is the following:

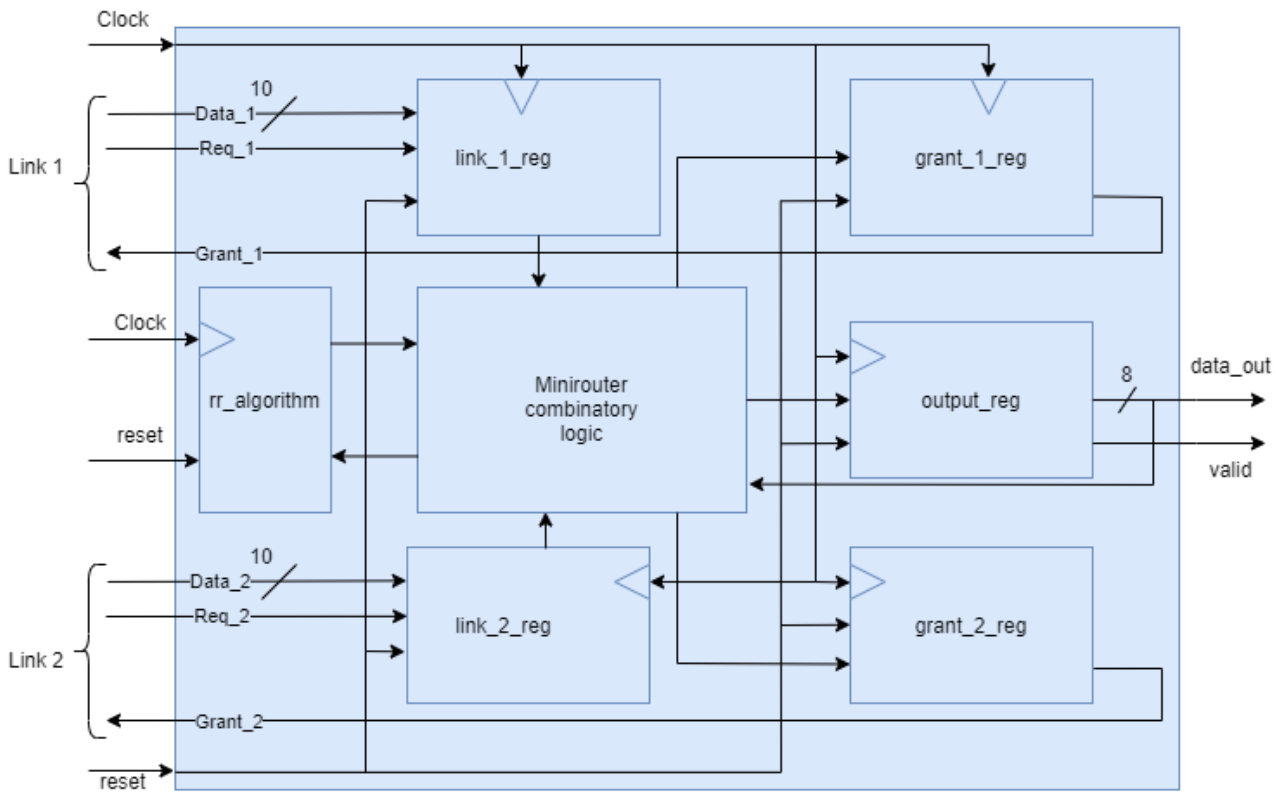


Figure 2: Mini Router Architecture

The Mini Router is composed by seven subcomponents needed to implement the functionalities:

- **link_1_reg & link_2_reg:** Used to sample the incoming message on their respective link in order to have a finite, optimizable critical path
- **rr_algorithm:** Used to provide the combinatorial network with the ability to choose between the links that have both req='1' and same priority
- **grant_1_reg & grant_2_reg:** Used to keep in a register the value to send as an output to the data sender to notify it the propagation of the data that he is sending to the Mini Router.
- **output_reg:** Used to keep in output the data to be propagated and the validity signal for one clock cycle.

Let's see how these components are implemented one by one.

2.1 Link_reg

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY link_in_reg IS
  GENERIC( N_BIT : positive :=10);
  PORT (
    -- ----Synchronizing logic---- --
    reset      : in std_logic;
    clock      : in std_logic;

    -- ----Link 1---- --
    data_link  : in std_logic_vector(N_BIT - 1 DOWNTO 0);
    req_link   : in std_logic;

    -- ----Final outputs---- --
    data_out   : out std_logic_vector(N_BIT - 1 DOWNTO 0);
    req_out    : out std_logic
  );
END link_in_reg;

ARCHITECTURE Behavioral OF link_in_reg IS
BEGIN
  Clock_Event:PROCESS(clock,reset)
  BEGIN
    IF(reset='0') THEN
      data_out<=(OTHERS => '0');
      req_out<='0';
      -- Asynchronous reset signal
    ELSIF (clock'EVENT AND clock='1') THEN
      data_out<=data_link;
      req_out<=req_link;
      -- At the rising edge of the clock set the output
    END IF;
  END PROCESS Clock_Event;
END Behavioral;
```

Figure 3: link_reg VHDL code

The reset signal has been managed as an asynchronous signal that will reset the value of the ‘data’ and ‘required’ link to ‘0’.

The main section of the process is activated with a rising edge event on clock signal. In this case the register will sample the input from the link and keeps it stable for the elaborations executed inside the combinatory network up ahead.

Overall, we have two ten-bits registers and one one-bit register for each link.

2.2 Rr_algorithm

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY rr_algorithm IS
  PORT (
    -- ----Synchronizing logic---- --
    reset      : in std_logic;
    clock      : in std_logic;

    -- ----RR choice---- --
    rr_choice   : out std_logic;          -- The choice of the RR algorithm, codified in '0' if is the turn to choose link 1 and '1' otherwise
    rr_notify   : in std_logic          -- Used to notify to the register that we used that value so he can flip it
  );
END rr_algorithm;

ARCHITECTURE Behavioral OF rr_algorithm IS
BEGIN
  Clock_Event:PROCESS(clock,reset)
  BEGIN
    IF(reset='0') THEN
      rr_choice<='0';
    ELSIF (clock'EVENT AND clock='1') THEN
      IF(rr_notify='1') THEN
        rr_choice<='1';
      ELSIF(rr_notify='0') THEN
        rr_choice<='0';
      END IF;
    END IF;
  END PROCESS Clock_Event;
END Behavioral;
```

Figure 4: “rr_algorithm” VHDL code

This component is used to keep track of the last link prioritized during the execution of the previous propagations. It is linked to “Minirouter_combinatorial_logic” from which it receives a signal change each time the value has been used.

The register, as soon as it notices the value change in “rr_notify”, change the out value of the signal “rr_choice”.

2.3 Grant_reg

The “grant_reg” module is implemented as a D-Flip-Flop, it saves the information on the grant data and propagate it through the output to notify the sender on its respective link that the data he is sending has already been propagated.

Its output is controlled each clock cycle from the combinatory network that implement all the logic functions inside the router, we will see it in the next paragraph.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY grant_reg IS                                     --It is a D-Flip-Flop
    GENERIC( N_BIT : positive :=10);
    PORT (
        -- ----Synchronizing logic---- --
        reset      : in std_logic;
        clock      : in std_logic;

        -- ----Input---- --
        d          : in std_logic;           -- Input link
        q          : out std_logic          -- Output link
    );
END grant_reg;

ARCHITECTURE Behavioral OF grant_reg IS
BEGIN
    Clock_Event:PROCESS(clock,reset)
    BEGIN
        IF (clock'EVENT AND clock='1') THEN
            IF(reset='0') THEN
                q<='0';
            ELSE
                q<=d;                         -- At the rising edge of the clock set the output depending on the input
            END IF;
        END IF;
    END PROCESS Clock_Event;
END Behavioral;

```

Figure 5: "grant_reg" VHDL code

2.4 Minirouter_combinatorial_logic

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY minirouter_combinatorial_logic IS
    GENERIC( N_BIT : positive :=10);
    PORT (
        -- ----RR choice---- --
        rr_choice   : in std_logic;           -- The choice of the RR algorithm, codified in '0' if is the turn to choose link 1 and '1' otherwise
        rr_notify   : out std_logic;          -- Used to notify to the register that we used that value so he can flip it

        -- ----Link 1---- --
        data_link_1 : in std_logic_vector(N_BIT - 1 DOWNTO 0); -- Contains the data to propagate, if any.
        req_link_1  : in std_logic;           -- Show if the data in 'data_link_1' has to be propagated

        -- ----Link 2---- --
        data_link_2 : in std_logic_vector(N_BIT - 1 DOWNTO 0); -- Contains the data to propagate, if any.
        req_link_2  : in std_logic;           -- Show if the data in 'data_link_2' has to be propagated

        -- ----Current Output---- --
        current_out : in std_logic_vector(N_BIT - 3 DOWNTO 0); -- Current output, used to keep the previous value in the case there isn't any data to propagate

        -- ----Final outputs---- --
        data_out    : out std_logic_vector(N_BIT - 1 DOWNTO 2);
        valid_out   : out std_logic;

        -- ----Grant values---- --
        grant_1     : out std_logic;
        grant_2     : out std_logic
    );
END minirouter_combinatorial_logic;

```

Figure 6: "minirouter_combinatorial_logic" VHDL code

This is the entity of the combinatory logic section. As we can see in the code it is connected to the other modules described before. It uses them as memory holders for useful data, as long as it is necessary for the signal to traverse the network.

```

ARCHITECTURE Behavioral OF minirouter_combinatorial_logic IS
BEGIN
    -- No process needed is a combinatory net -> no clock

    -- We need to assign the data to the output in the case that at least one of the two links wants to propagate data, so we need to assign something to data_out only when at
    -- least one 'req' is set.

    data_out<=data_link_1(N_BIT-1 DOWNTO 2) WHEN ((req_link_1='1' AND req_link_2='0') OR
        (req_link_1='1' AND req_link_2='1' AND(
            (data_link_1(1 DOWNTO 0) > data_link_2(1 DOWNTO 0)) OR
            (data_link_1(1 DOWNTO 0) = data_link_2(1 DOWNTO 0) AND rr_choice='0')))) ELSE
        data_link_2(N_BIT-1 DOWNTO 2) WHEN ((req_link_2='1' AND req_link_1='0') OR
            (req_link_2='1' AND req_link_1='1' AND(
            (data_link_2(1 DOWNTO 0) > data_link_1(1 DOWNTO 0)) OR
            (data_link_2(1 DOWNTO 0) = data_link_1(1 DOWNTO 0) AND rr_choice='1')))) ELSE
        current_out;

    --Base case, it doesn't actually matter what's put in the output due to the value of 'valid_out' that is going to change, so to avoid dynamic power consumption
    -- I keep the same value.

    -- valid_out should be set for each case except the case in which there is no data to propagate. So is equal to '1' if at least one of the two links have req setted.
    valid_out<= '1' WHEN (req_link_1='1' OR req_link_2='1') ELSE
        '0';

    -- The grant value must be chosen according to the same criteria for which a certain data is propagated to the output
    grant_1<= '1' WHEN ((req_link_1='1' AND req_link_2='0') OR
        (req_link_1='1' AND req_link_2='1' AND(
            (data_link_1(1 DOWNTO 0) > data_link_2(1 DOWNTO 0)) OR
            (data_link_1(1 DOWNTO 0) = data_link_2(1 DOWNTO 0) AND rr_choice='0')))) ELSE
        '0';

    grant_2<= '1' WHEN ((req_link_1='0' AND req_link_2='1') OR
        (req_link_1='1' AND req_link_2='1' AND(
            (data_link_1(1 DOWNTO 0) < data_link_2(1 DOWNTO 0)) OR
            (data_link_1(1 DOWNTO 0) = data_link_2(1 DOWNTO 0) AND rr_choice='1')))) ELSE
        '0';

    -- It notify to the rr section that he used the current value so he has to flip it
    rr_notify<= '1' WHEN ((data_link_1(1 DOWNTO 0) = data_link_2(1 DOWNTO 0)) AND rr_choice='0' AND (req_link_1='1' AND req_link_2='1')) ELSE
        '0' WHEN ((data_link_1(1 DOWNTO 0) = data_link_2(1 DOWNTO 0)) AND rr_choice='1' AND (req_link_1='1' AND req_link_2='1')) ELSE
        rr_choice;

END Behavioral;

```

Figure 7: Behavioural of "minirouter_combinatorial_logic" VHDL code

As we can see in the code the behavioural is just a combinatory net which assign to the output links a combination of logic functions applied on the input. Each assignment is explained in the comments that are present inside the code.

Inside the file 'Minirouter.vhd' all these components are linked accordingly to the block diagram at page 4 of this report.

Chapter 3 – Test-plan & Testbench

In order to check the correctness of the system the following scenarios are verified:

1. **Check a single request:** We need to verify that in case of a single request all the operations are executed in the right way. This is the easiest case in which we can be, so it's important to check it first.
2. **No request**
3. **Check a multiple request:** This option is divided into two subcases
 - 2.1 **Same priority case**
 - 2.2 **Different priority**

In this case we need to be very careful. For the data-sender to send a new data to be propagated, it must first have seen that the previous data has been sent.

To do this, it reads the 'grant' input of its own link but, to correctly perform the handshake, it is logical to assume that at least one clock cycle passes between the notification and the sending of a new data.

3.1 Validation program

To validate the VHDL implementation I used a Python algorithm that simulate the mini router behavioural. Let's start defining a function:

```
def link_1_out():
    data_out=data_1[1:9]
    valid=1
    granted_1=1
    grant_1=granted_1
    granted_2=0
    grant_2=granted_2
    print("    --->data_out: " + data_out + "\n    --->valid: " + str(valid)
          +"\n    --->grant_1: "+ str(granted_1)+"\n    --->grant_2: "+str(granted_2))
```

Figure 8: Link_output_print

This function print the output that the mini router should return given an input on its first link. There is also another function called 'link_2_out' that make the same thing but for the other link.

The main body of the algorithm is just a cycle that get inputs from 'input.csv' and test various situations in which the mini router can find itself.

```

import sys
import os
import csv
import pandas as pd

initial_row=0
i=0
for i in range(18):
    print("New Cycle")
    data_1=inputFile.iloc[initial_row,1]
    req_1=inputFile.iloc[initial_row,0]
    print("    Data first Link: " + data_1 + "\n    Req: " + str(req_1))
    initial_row=initial_row+1
    data_2=inputFile.iloc[initial_row,1]
    req_2=inputFile.iloc[initial_row,0]
    initial_row=initial_row+1
    print("    Data second Link: " + data_2 + "\n    Req: " + str(req_2))
    priority_1=data_1[9]+data_1[10]
    priority_2=data_2[9]+data_2[10]
    print("    Priority of First Link: " + priority_1)
    print("    Priority of Second Link: " + priority_2)
    if (req_1==1 and grant_1==0) and (req_2==1 and grant_2==0):
        print("    ****Double request****")
        if priority_1==priority_2:
            print("    ****Same Priority****")
            if rr==0:
                print("    ****RoundRobin Algorithm chooses link n° 1****")
                rr=1
                link_1_out()
            else:
                print("    ****RoundRobin Algorithm chooses link n° 2****")
                rr=0
                link_2_out()
        elif (req_1==1 and req_2==0) or priority_1>priority_2 :
            print("    ****Link 1 has higher pririty****")
            link_1_out()
        else:
            print("    ****Link 2 has higher pririty****")
            link_2_out()
    elif (req_1==1 and grant_1==0) and (req_2==0):
        print("    ****Single Request on first link****")
        link_1_out()
    elif (req_2==1 and grant_2==0) and (req_1==0):
        print("    ****Single Request on second link****")
        link_2_out()
    else:
        print("    ****No request****")

```

The test plan is: try, at the beginning, one single request on each link; then keep on going with two consecutive simultaneous requests to check the behaviour of the Round-Robin Algorithm to end with some verifications on double requests with different priorities.

3.2 Validation results

At the end of the test-plan, checking the output that can be found in the file 'output.txt', we can state that the algorithm behaves coherently to the specs. In the following pictures we can see some of the most critical cases.

```
New Cycle
Data first Link: "0110111101"
Req: 1
Data second Link: "0110000101"
Req: 1
Priority of First Link: 01
Priority of Second Link: 01
****Double request****
****Same Priority****
****RoundRobin Algorithm chooses link n° 1****
--->data_out: 01101111
--->valid: 1
--->grant_1: 1
--->grant_2: 0

New Cycle
Data first Link: "0000000000"
Req: 0
Data second Link: "0110000101"
Req: 1
Priority of First Link: 00
Priority of Second Link: 01
****Single Request on second link****
--->data_out: 01100001
--->valid: 1
--->grant_1: 0
--->grant_2: 1

New Cycle
Data first Link: "0000000000"
Req: 0
Data second Link: "0000000000"
Req: 0
Priority of First Link: 00
Priority of Second Link: 00
****No request****

New Cycle
Data first Link: "1101001010"
Req: 1
Data second Link: "1101011110"
Req: 1
Priority of First Link: 10
Priority of Second Link: 10
****Double request****
****Same Priority****
****RoundRobin Algorithm chooses link n° 2****
--->data_out: 11010111
--->valid: 1
--->grant_1: 0
--->grant_2: 1
```

Figure 9: RR algorithm results

Figure 10: Round-Robin Check

```

New Cycle
Data first Link: "1010010000"
Req: 1
Data second Link: "0100101011"
Req: 1
Priority of First Link: 00
Priority of Second Link: 11
****Double request****
****Link 2 has higher priority****
--->data_out: 01001010
--->valid: 1
--->grant_1: 0
--->grant_2: 1

```

Figure 11: Generic double request

These results will be our expected values. We're going to give as input to the VHDL implementation the same values to see if it behaves like expected.

3.3 Simulation through ModelSim

The first operation to start the simulation is compile the whole code checking for syntax error.

```

# Reading pref.tcl
# Loading project Minirouter
# Compile of link_in_reg.vhd was successful.
# Compile of link_out_reg.vhd was successful.
# Compile of grant_reg.vhd was successful.
# Compile of minirouter_combinatorial_logic.vhd was successful.
# Compile of rr_algorithm.vhd was successful.
# Compile of Minirouter.vhd was successful.
# Compile of minirouter_tb.vhd was successful.
# 7 compiles, 0 failed with no errors.

```

Figure 12: Model-Sim compilation results

As is easily deduced from Figure 12, there appear to be no syntax errors. So, we can safely begin our test-bench to check the Mini-router behaviour.

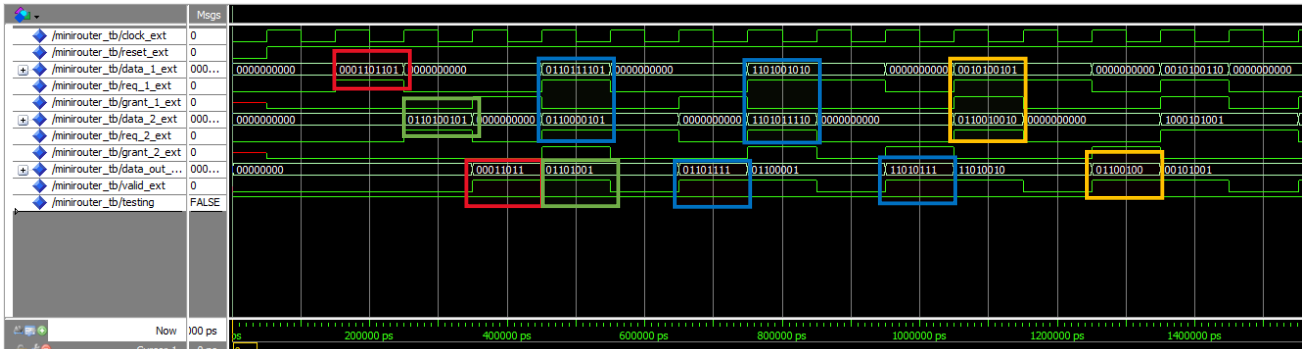


Figure 13: Test-bench part 1

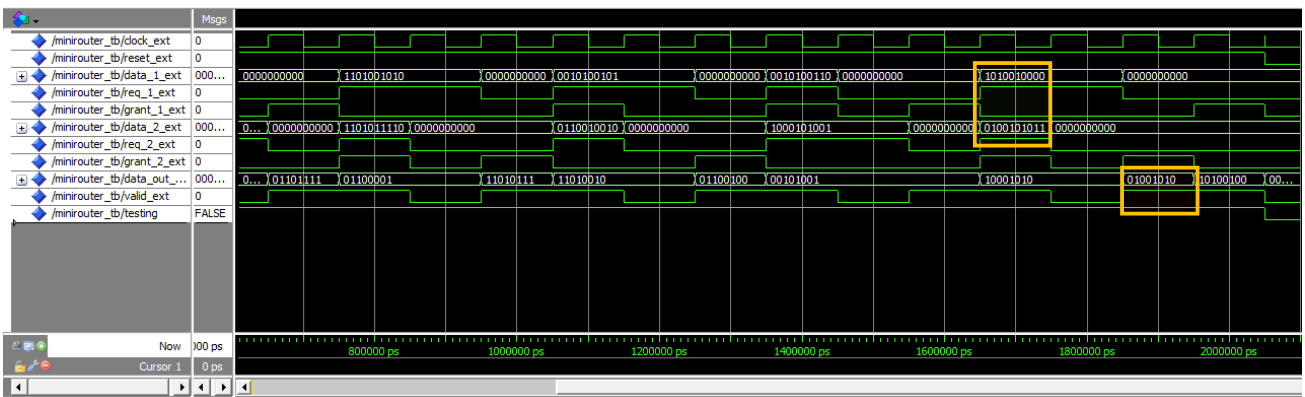


Figure 14: Test-bench part 2

Double-checking the results given from the simulation we can verify that the code behaves coherently to the specs and gives the same results of the validation code.

It's important to notice that **the output is shifted to right of one clock cycle** due to the two registers encountered from the input to the output by the data.

In Figure 13 we can observe some cases, highlighted by coloured rectangles:

- **Green and Red:** We can see two single requests on each link
- **Blue:** Two requests with same priority to check Round-Robin Algorithm
- **Yellow:** Double requests with different priority value

All these cases are managed correctly by the implementation.

Chapter 4 – Synthesis & Implementation

The next step is to give the VHDL code to Vivado Synthesis Tool to synthesize the code into a real and doable implementation.

Before heading through the Synthesis another double-check has been made to verify the correctness of the system. This is done by comparing the schemas obtained through the RTL analysis of Vivado, with the original block scheme, the result is the following:

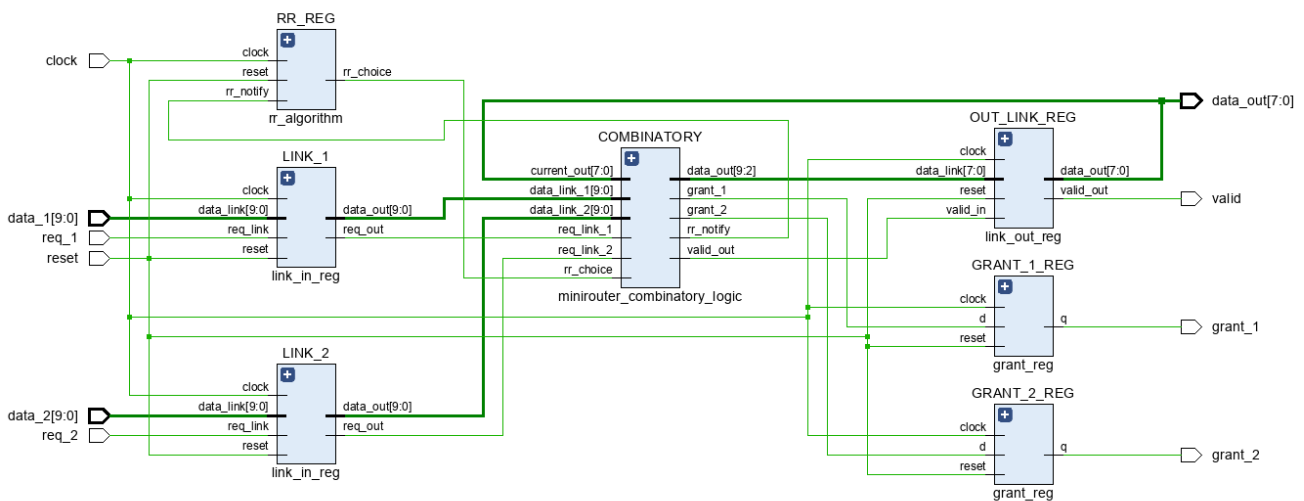


Figure 15: RTL Analysis Schematic

Everything is like expected so we can proceed through the synthesis.

All things considered the results that will be shown in this chapter are obtained after the Synthesis phase with the clock constraint and in Out-of-Context mode.

4.1 Synthesis warnings

After the synthesis Vivado doesn't show any warning given by this operation, as we can see in the following picture that represent the synthesis results.

Synthesis	
Status:	✓ Complete
Messages:	No errors or warnings
Part:	xc7z010clg400-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	None

Figure 16: Synthesis result

No warnings were given, this means that I can go through the implementation to see the results.

4.2 Implementation Results and Timing

After the execution of the implementation no errors or warning were shown:

Implementation		Summary	Route Status
Status:	✓ Complete		
Messages:	No errors or warnings		
Part:	xc7z010clg400-1		
Strategy:	Vivado Implementation Defaults		
Report Strategy:	Vivado Implementation Default Reports		
Incremental implementation:	None		

Figure 17: Implementation Results

I executed the implementation with the following constraint:

```
create_clock -period 8.000 -name clock -waveform {0.000 4.000} -add [get_ports -filter { NAME =~ "*clock*" &&
DIRECTION == "IN" }]
```

Which specify the clock frequency to $f_{clock} = \frac{1}{8ns} = 125MHz$. I chose this frequency because it is the one at which the Zybo board operates.

Running the implementation with this clock constraint returns the following results:

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 5,361 ns	Worst Hold Slack (WHS): 0,135 ns	Worst Pulse Width Slack (WPWS): 3,500 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 12	Total Number of Endpoints: 12	Total Number of Endpoints: 34
All user specified timing constraints are met.		

Figure 18: Time Report

As we can state from Figure 18 the Worst Negative Slack (WNS) is **positive**, this mean that we can drive our implementation at a maximum frequency, calculated as:

$$f_{max} = \frac{1}{T_{clock} - WNS} = 378.931 \text{ MHz}$$

Where T_{clock} is equal to the period of the clock: **8ns**.

The WSN corresponds to the lowest slack among the slacks of all the critical paths:

Name	Slack ^{^1}	Levels	High Fanout	From	To
↳ Path 1	5.361	2	12	LINK_2/req_out_reg/C	RR_REG/rr_choice_reg/D
↳ Path 2	5.369	2	10	LINK_1/data_out_reg[0]/C	OUT_LINK_REG/..._out_reg[1]/D
↳ Path 3	5.520	2	10	LINK_1/data_out_reg[0]/C	OUT_LINK_REG/..._out_reg[2]/D
↳ Path 4	5.545	2	10	LINK_1/data_out_reg[0]/C	OUT_LINK_REG/..._out_reg[4]/D
↳ Path 5	5.716	2	10	LINK_1/data_out_reg[0]/C	GRANT_1_REG/q_reg/D
↳ Path 6	5.736	2	10	LINK_1/data_out_reg[0]/C	GRANT_2_REG/q_reg/D
↳ Path 7	5.769	2	10	LINK_1/data_out_reg[0]/C	OUT_LINK_REG/..._out_reg[0]/D
↳ Path 8	5.801	2	10	LINK_1/data_out_reg[0]/C	OUT_LINK_REG/..._out_reg[5]/D
↳ Path 9	5.834	2	10	LINK_1/data_out_reg[0]/C	OUT_LINK_REG/..._out_reg[3]/D
↳ Path 10	5.871	2	10	LINK_1/data_out_reg[0]/C	OUT_LINK_REG/..._out_reg[6]/D

Figure 19: Critical paths report

So, we can conclude that the path that goes from Link_2 to RR_Reg is the one that has the highest impact on the delay and on the frequency of the clock.

4.3 Resource Utilization

The resource utilization by this architecture is the following:

Resource	Utilization	Available	Utilization %
LUT	13	17600	0.07
FF	34	35200	0.10

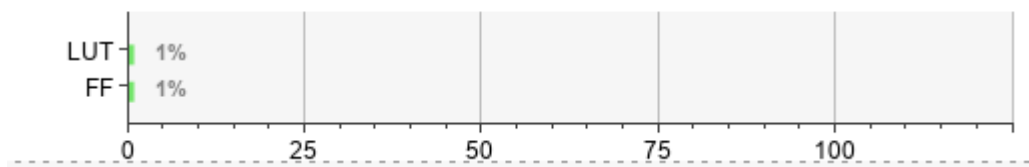


Figure 20: Resource Utilization Report

As we can expect, the used resources are quite low compared to those available, due to the simplicity of the operations that the mini router must do, especially for the Look Up Tables and the Flip-Flops, which are equal or less than 1% of the total. The most used resource is the I/O, which will require 35 pins out of 100.

This architecture is then realizable on the Zybo Zynq 7000 board. The I/O utilization is not specified because I defined it to resolve some Out-Of-Context warnings as I will show in the next paragraphs.

4.4 Power Consumption

The power consumption of this architecture is the following:

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 0.09 W
Design Power Budget: Not Specified
Power Budget Margin: N/A
Junction Temperature: 26,0°C
Thermal Margin: 59,0°C (5,0 W)
Effective θ_{JA} : 11,5°C/W
Power supplied to off-chip devices: 0 W
Confidence level: Medium

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

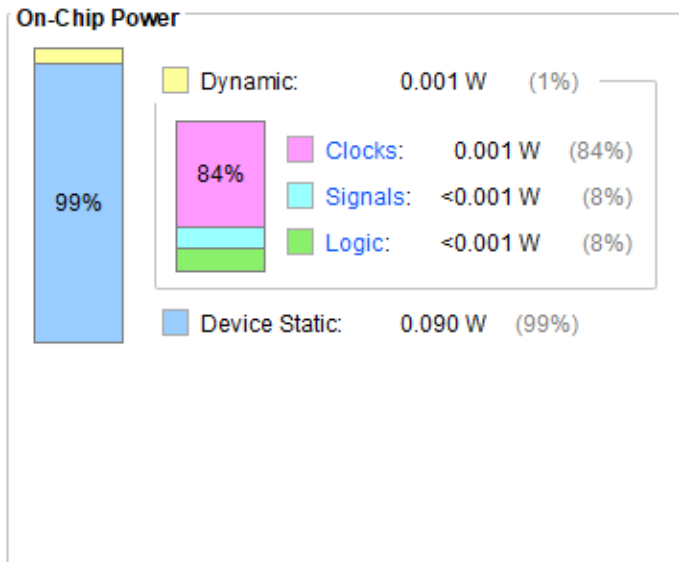


Figure 21: Power Consumption Report

As we can see in Figure 21 a total of **0.09W** of power are needed, which is **highly unbalanced** between static and dynamic power: Dynamic cover roughly 1% of the whole consumption (0.001W) and the Static that cover the remaining consumption (0.090W). The most relevant contribute for the Dynamic power is given by the Clock that cover the 84% of the power consumption, the remaining 16% is equally divided .

4.5 DRC Violation

The only warning that results from the implementation is:


 ZPS7 #1	Warning	The PS7 cell must be used in this Zynq design in order to enable correct default configuration.
---	---------	---

Figure 22

This warning result to be irrelevant in our case because it refers to the ARM processor environment¹.

¹ [Simple connection between LED and switch through the PL \(xilinx.com\)](#)

These results were given if the Implementation is done without the option ‘-mode out_of_context’, given that we don’t want to really test the project on the Zybo board.

The constraint that I set are:

```
create_clock -period 8.000 -name Clock_Constr -waveform {0.000 4.000} -add [get_ports -filter { NAME =~ "*clock*" && DIRECTION == "IN" }]
set_property HD.CLK_SRC BUFGCTRL_X0Y16 [get_ports -filter { NAME =~ "*clock*" && DIRECTION == "IN" }]

set_property HD.PARTPIN_LOCS INT_R_X1Y0 [get_ports -filter { NAME =~ "*data*" && DIRECTION == "IN" }]
set_property HD.PARTPIN_LOCS INT_R_X1Y0 [get_ports -filter { NAME =~ "*req*" && DIRECTION == "IN" }]
set_property HD.PARTPIN_LOCS INT_R_X1Y0 [get_ports -filter { NAME =~ "*reset*" && DIRECTION == "IN" }]
```

Figure 23: Constraints

Except the clock constraint that I have already discussed before, we can see four other constraints that are needed for the OOC Implementation.

The PARTPIN_LOC constraints are needed to define a specific interconnect tile for the specified port to be routed.

The CLOCK_SRC used in the OOC implementation to tell the implementation tools if a clock buffer will be used outside the out-of-context module. The value should be the location of the clock buffer instance.

5 - Conclusions

The objective of this work was to design and implement an architecture for a mini-router working on a Zybo Zynq 7000, given the implementation results and the testbenches done correctly we can assume that the work is concluded.

In the end we can assume that our implementation is an architecture that should be extended to implement more complex components to fulfil real, and more complicated, necessities.

With the results we have obtained, however, we can hypothesize some evolutions that this project could have, for example:

- Evaluate the use of another FPGA that has a lower number of ports and is more suited to our needs.
- Evaluate the use of another FPGA with a higher clock frequency to take advantage of the potential frequency increase.
- Evaluate to add more than just one output link to correctly perform the routing, and/or even more input links.

In the end, we may end on thinking about some optimizations that can be done on our architecture, for example we can try to add registers within the combinatory net that implement the logic functions of the mini-router, looking for better optimization of the clock constraint.