UNIVERSITÀ DI PISA

Electronics Systems

VHDL implementation of a Mini Router

Alessandro Noferi

2018

Contents

1. Introduction

In this project we develop a *mini router* that given data came from two links, forwards only one of them, according to some rules. In particular, each link is composed of a `req` signal, that must be *high* in order to forward the `data` signal assigned to that link. In case of both links have the `req` signal high, the link with the highest *priority* will be forwarded (the bits of priority are the last two bits of the `data` signal). There is still the case where both links have also the same priority (in addition to the high `req` signal), in this situation a Round Robin algorithm will privilege one of the two. The output data is paired with the `valid` signal that guarantees the validity of the data. This operation will be done every clock cycle.

2. Architecture Implementation

The mini router that we have to implement is shown in Figure 2.1

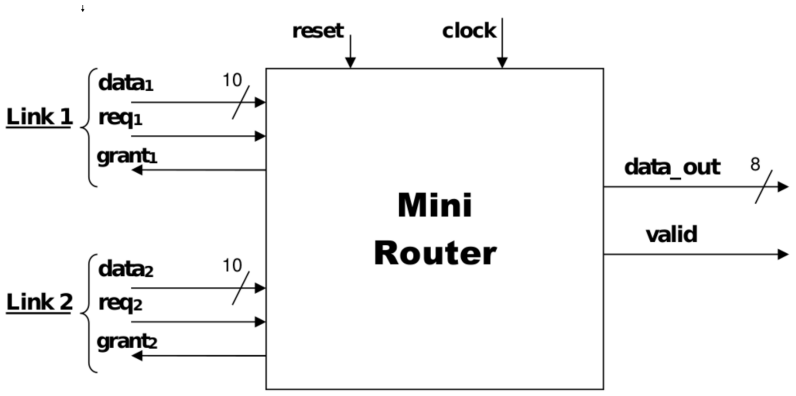


Figure 2.1: Mini Router

Where:

- **clock** represents the *input* clock signal;
- **reset** represents the *input* reset signal, we use it as "active-low";
- **Link*i*** represents all the information relative to its channel:
  - **data$_i$** is a 10 bits *input* signal that represents 8 bits of data and 2 bits of priority;
  - **req$_i$** is a 1-bit *input* signal that "validate" the request of forward;
  - **grant$_i$** is a 1-bit *output* signal that informs the link$_i$ that its data has been forwarded;
- **data_out** is a 8 bits *output* signal that represents the data forwarded without the 2 priority bits.
- **valid** is a 1-bit *output* signal that validates the data on output;

During the first phases of the development some a solutions were evaluated:

1 Implement the mini router that every clock cycle gets the inputs, chooses the output and on the next rising edge clock puts it out. This solution it is not very clear at all. The mini router should be a combinatorial logic, that given inputs it produces outputs as soon as possible (the delay between inputs and output is the propagation logic time $t_{p-logic}$, used subsequently) and we could use separate registers in order to reduce the critical path caused by combinatorial logics.

2 Create an architecture that have input and output registers with a pure combinatorial logic between them. The combinatorial logic will be the mini router. In this way the whole architecture is clearer and each component does its own job.

This report examines the second solution.

Block Diagram and VHDL implementation

Looking inside the Mini Router box, the architecture is the following:

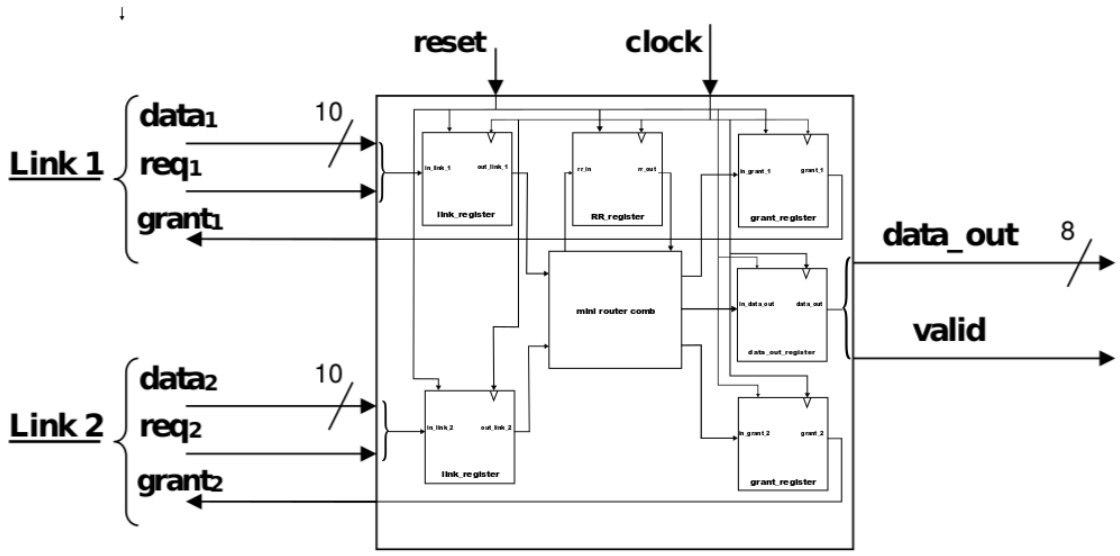


Figure 2.2: Block Diagram

`link` and the output signals are VHDL records:

```
-- author: Alessandro Noferi
-- 4 spaces per tab indentation

library ieee;
use ieee.std_logic_1164.all;

-- This package let u

package router_pkg is
    type link is record
      data  : std_logic_vector(9 downto 0); -- data signal 10 bits (8 data + 2 priority)
      req : std_logic;                      -- req signal 1 bit
    end record  link;

    type router_data_out is record
      data_out  : std_logic_vector(7 downto 0); -- output data signal 8 bits (input data wo 2
 ↪  priority bits)
      valid : std_logic;                        -- valid signal 1 bit
    end record router_data_out;
end package router_pkg;
```

Consequently, `link_register` and `data_out_register` are respectively:

```
library IEEE;
use IEEE.std_logic_1164.all;

use work.router_pkg.all; -- Use of link record and router_data_out

------------------------------------------------------------------
-- entity declaration        link_register                       |
------------------------------------------------------------------
```

```
entity link_register         is
        port (
                in_link : in link;
                clk        : in std_logic;
                rst        : in std_logic;
                out_link: out link
        );
end link_register;

-------------------------------------------------------------
-- architecture: link_register                              |
-------------------------------------------------------------
architecture bhv of link_register is
begin
        reg_p: process(clk)
                begin
                        if (clk = '1' and clk'event) then -- every rising edge clk
                            if rst = '0' then -- synchronous reset active-low
                                -- reset output
                                    out_link.data <= (others => '0');
                                    out_link.req <= '0';
                            else
                                -- set output
                                    out_link.data <= in_link.data;
                                    out_link.req <= in_link.req;
                            end if;
                        end if;
                end process;
end bhv;


library IEEE;
use IEEE.std_logic_1164.all;

use work.router_pkg.all; -- Use of link record and router_data_out

-------------------------------------------------------------
-- entity declaration        data_out_register              |
-------------------------------------------------------------

entity data_out_register is
        port (
                in_data_out : in router_data_out;
                clk        : in std_logic;
                rst        : in std_logic;
                out_data_out: out router_data_out
        );
end data_out_register;

-------------------------------------------------------------
-- architecture: data_out_register                          |
-------------------------------------------------------------
architecture bhv of data_out_register is
begin
        reg_p: process(clk)
                begin
                        if (clk = '1' and clk'event) then -- every rising edge clk
                            if rst = '0' then -- synchronous reset active-low
                                -- reset output
                                    out_data_out.data_out <= (others => '0');
                                    out_data_out.valid <= '0';
                                else
                                -- set output
                                    out_data_out.data_out  <= in_data_out.data_out;
                                    out_data_out.valid <= in_data_out.valid;
                            end if;
                        end if;
                end process;
end bhv;
```

The other registers are simple 1-bit registers that store the state of the signal.

The `RR_register` is a register that stores the value of the Round Robin algorithm. Finally, the combinatorial logic `mini_router_comb`, is the main part of the router; this block chooses the correct data to forward according to `req`, `priority` and `RR`.

For readability reason, in this report we will show only the entity of this block (complete sources are on the project folder):

```
-------------------------------------------------------------
-- entity declaration        mini router_comb               |
```

```
------------------------------------------------------------------

entity mini_router_comb is
        port (
        -- inputs
        link_1  : in link; -- record with data and req signals
        link_2  : in link; -- record with data and req signals
        -- rr alg
        rr_in     : in std_logic;
        rr_out          : out std_logic;
        -- outputs
        grant_1     : out std_logic;
        grant_2     : out std_logic;
        router_out : out router_data_out  -- record with output data and valid signals
        );
end mini_router_comb;
```

The whole architecture that represent the box shown on Figure 2.1 has been implemented interconnecting all the components described above.

Again, I will show only a snipped of the entire `Mini_router.vhd` file:

```
------------------------------------------------------------------
-- entity declaration        mini router                        |
------------------------------------------------------------------

entity mini_router is
        port (
        -- inputs
        link_1  : in link; -- record with data and req signals
        link_2  : in link; -- record with data and req signals
        rst     : in std_logic; --  active-low
                                clk             : in std_logic;

        -- outputs
        grant_1     : out std_logic; -- record with output data and valid signals
        grant_2     : out std_logic;
        router_out : out router_data_out  -- record with output data and valid signals
        );
end mini_router;

------------------------------------------------------------------
-- architecture: mini_router                                     |
------------------------------------------------------------------
architecture bhv of mini_router is
    -- components declaration


    ...
       -- signals to interconnect the components (regisgters to mini router)
        signal reg_in_link_1    : link;
        signal reg_in_link_2    : link;
        signal out_reg_data_out : router_data_out;
        signal out_reg_grant_1  : std_logic;
        signal out_reg_grant_2  : std_logic;

        signal rr_in :  std_logic := '0'; -- rr starts at 0
        signal rr_out : std_logic;

    begin
        -- component definition
        --set input registers
        IN_LINK1_REG: link_register
        port map(link_1, clk, rst, reg_in_link_1);

        IN_LINK2_REG: link_register
        port map(link_2, clk, rst, reg_in_link_2);

        --set mini router comb
        MINI_ROUTER: mini_router_comb
        port map(reg_in_link_1, reg_in_link_2, rr_in, rr_out,
                 out_reg_grant_1, out_reg_grant_2,out_reg_data_out);

        --set output registers
        OUT_REG: data_out_register
        port map(out_reg_data_out,clk, rst, router_out);

        ...
```

3. Test plan

In order to see if the combinatorial logic works correctly, a test-bench file has been built up that shows the behavior on some cases, like same priority requests and `req` signal at 0.
Then, it has been developed a test bench for the entire router that shows the evolution, clock after clock, of all the signals involved.

Mini Router Combinatorial Test-Bench

Figure 3.1 shows the test-bench of the component inside the router that chooses with data link forwards out, let call it **arbiter**.

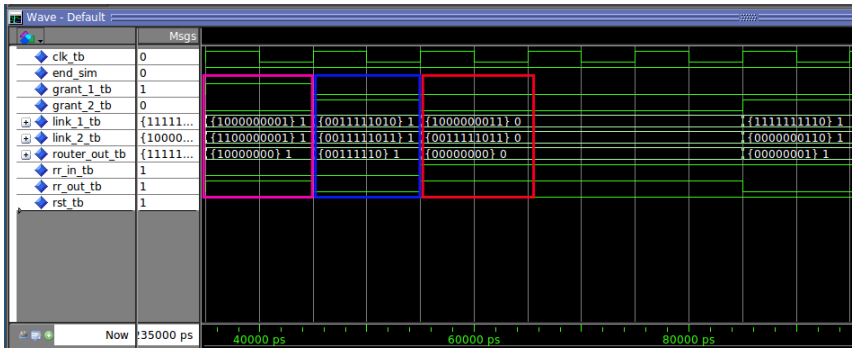


Figure 3.1: Mini Router Combinatorial TB

- **Pink square**: both `link_1` and `link_2` have the `req` bit at 1, the arbiter then checks the priority and finds that are the same, in this case it looks at the round robin bit state and since it is 0, it sends out `link_1` data's without the last two priority bits (10000000) and actives the `grant_1` signal and put to 0 the `grant_2` in order to inform the senders. Note that it flips the round robin bit to 1 (`rr_out`).

- **Blue square**: both `link_1` and `link_2` have the `req` bit at 1, but in this case `link_2` has a higher priority and so the arbiter sends out (00111100) with refer of `link_2`.

- **Red square**: in this case, both the request bits are 0, so the arbiter will send out `valid` at 0, no matter the what the output data is.

Mini Router Test-Bench

Figure 3.2 shows the test-bench of the mini router when same priority links want to be forwarded, with different priority and a round robin change that changes the output data.

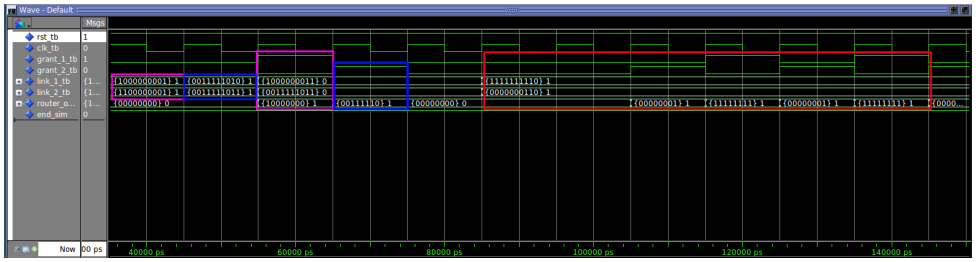


Figure 3.2: Mini Router TB

In this test-bench the blue and pink squares represent the same behavior as in the explanation of the combinatorial test.

The interesting part is the red square that shows the work of the round robin algorithm: from a certain point, always the same data has been presented to the mini router with both `req` bits at 1 and same priority (10), (11111111) for `link_1` and (00000001) for `link_2`, but according to the round robin, at each clock the output data changes between the value of the two links.

4. Synthesis

After the development of the architecture we synthesized it on the Zybo (ZYnq BOard)[1] using Xilinx VIVADO Design Suite.

RTL Design

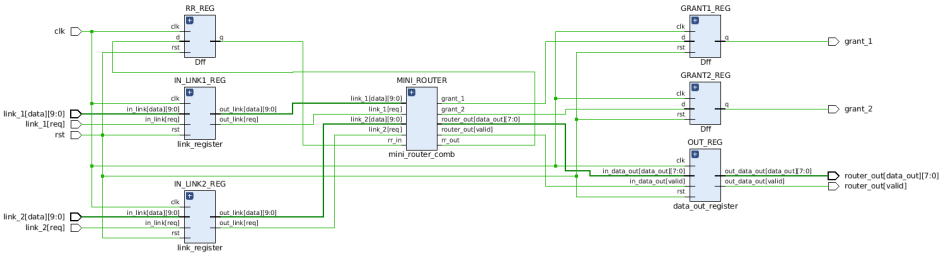


Figure 4.1: Mini Router TB

In Figure 4.1 we can see the input register and the register that stores the turn of which link to forward on the left, the combinatorial logic in the center and the output registers on the right.

Synthesis

For the synthesis, we have chosen a clock speed constraint of 10ns, thus a frequency of 100MHz.
We then run it and it successfully completed with some warnings:

- **[Constraints 18-5210]No constraint will be written out.**
  According to the Xilinx Forum[2] this warning can be safely ignored.
- **ZPS7 The PS7 cell must be used in this Zynq design in order to enable correct default configuration.**
  Event in this case, from the Xilinx Forum[3], we can ignore it since it refers to the ARM Processor environment which is out of our scopes.
- **35 out of 35 logical ports use I/O standard (IOSTANDARD) value 'DEFAULT', instead of a user assigned specific value.** [...].

[1] XC7Z020-Data-Sheet.pdf
[2] https://forums.xilinx.com/t5/Synthesis/Meaning-of-synthesis-warning-Constraints-18-5210-No-constraint/m-p/881007
[3] https://forums.xilinx.com/t5/Welcome-Join/How-to-instantiate-the-PS7-block/m-p/333953

This is due to we did not configure I/O pins of the boards and since we will not test the mini router with the board, also this warning can be safely ignored.

- **TIMING Warning: An input delay is missing on `signal` relative to clock(s) mini_router_clk.** These warnings affect all the input and output signals, except the clock and it is due to that we haven't set input and output delays of the wires that in practice are present. To avoid this warnings we can set these delays using commands like: `set_input_delay -clock [options]`. These stuff are out of our purposes and so we can ignore even them.

Timing Report

The timing report of the synthesis is shown in Figure 4.2:

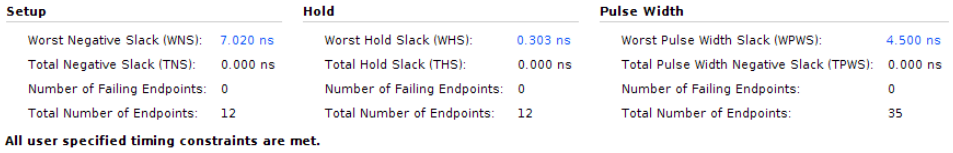


Figure 4.2: Timing Report

The clock time constraint previously choose is met.

As we could expect, the worst critical path is one of the paths that starts from the input register and end at an output register. In particular, the path with the worst slack time goes from `link_1_register` and finishes to the `data_out_register`, as shown in Figure 4.3

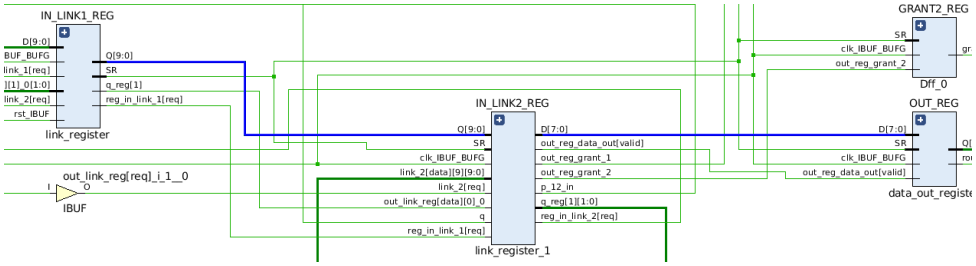


Figure 4.3: Critical Path after Implementation

Figure 4.3 can be confusing, it seems that `link_1` register signals passes trough `link_2` register, where in reality they are two independent registers. The reason is that, looking at inside `link_2` register, Vivado inserts on it all the LUTs used for the combinatorial logic of the mini router and obviously, they require `link_1` signals as well.

Max Operating Frequency

We know that the clock speed must be

$$t_{clk} \geq t_{su} + t_{p-logic} + t_{c-q}$$

where:

- t_{su} is the time during which the input shall be stable before the clock edge;
- t_{c-q} is the time after the clock edge necessary for the stabilization on the output;
- $t_{p-logic}$ is the time necessary for the combinatorial logic to produce the output given the input

and all this three parameters are fixed.
In our case, we also have the slack time t_{slack} that represent the amount of time that the data is stable, before the clock edge, in addition to the setup time t_{su}:

$$t_{clk} = t_{su} + t_{p-logic} + t_{c-q} + t_{slack}$$

To minimize the clock period we have to be in the limit case where the t_{slack} is equals to 0:

$$t_{clk_{min}} = t_{su} + t_{p-logic} + t_{c-q} + 0$$

.
The last equation is inserted in the t_{clock} equation :

$$t_{clk} = t_{clk_{min}} + t_{slack}$$

Since we know both $t_{clk} = 10\text{ns}$ and $t_{slack} = 7,020\text{ns}$ we can easily compute

$$t_{clk_{min}} = t_{clock} - t_{slack} = 10 - 7,020 = 2,98\text{ns}$$

And finally

$$f_{max} = \frac{1}{t_{clk_{min}}} = 335,57\text{MHz}$$

A synthesis with the clock constraint equal to $2,98\text{ns}$ confirmed our calculations returning a null slack time.

Note that the values reported after the synthesis are *estimated* values and they do not take into account placement or routing information. More accurate values will be reported after the implementation phase, see Chapter 5.

5. Zybo Implementation

We have also run the implementation in order to see where and which components of the board would be used if we had loaded the mini router system on the board.
As we previously said, the timing report during the synthesis is an estimation, in fact, after the implementation, the slack it has been increased to $7,251$ ns, so now the max operating frequency is:

$$f_{max} = 363,77\text{MHz}$$

Besides, also the path relative to the worst negative slack has changed, even is still the path between one input register and an out register. Now it starts from `link_2` register and ends at the same register though the same signal.

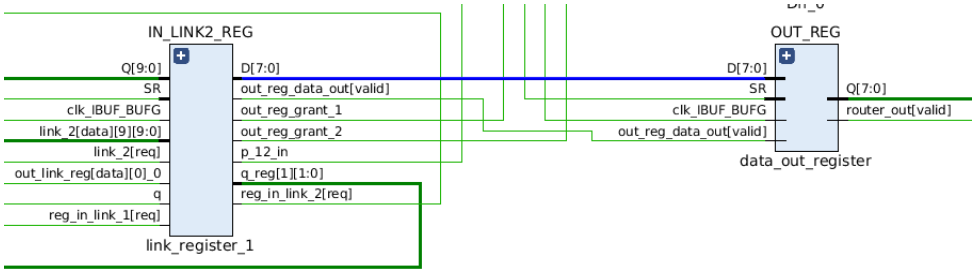
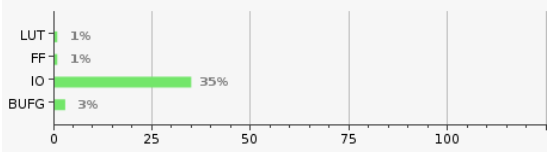


Figure 5.1: Critical Path after Synthesis

Resources Utilization

Resource	Utilization	Available	Utilization %
LUT	15	17600	0.09
FF	34	35200	0.10
IO	35	100	35.00
BUFG	1	32	3.13

(a) Used resources



(b) Used resources of the total available on the board

Figure 5.2: Resources Utilization

Figure 5.2a shows the amount of resources used on the board to implement the system, while Figure 5.2b represents the percentage of the used resources with respect to the total available on the Zybo: as we can see, except I/O pins, the values are low. This is legitimate, since this project is very small with

respect to the complexity that the board is able to perform, the LUTs and the Flip Flops required are a small percentage of the total.

The story is slightly different with respect to the I/O pins, this is due to the number of signal involved on the mini router, i.e. link and output channels, that require respectively 12 and 9 bits.

The real implementations of the system on the board is shown in Figure 5.3

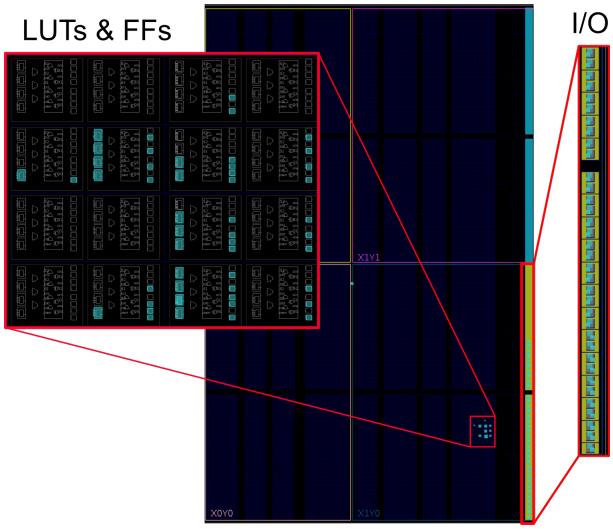


Figure 5.3: Mini Router device Implementation

Power Utilization

As we can see in Figure 5.4, nearly the total of the dissipated power is static (95%). The remain 5% is attributable to the clock and as we said for the resources utilization, the I/O.

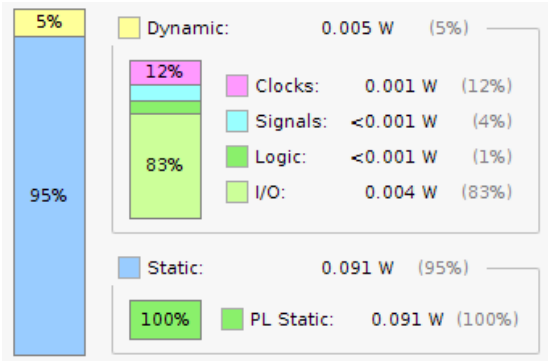


Figure 5.4: Power Utilization

6. Conclusion

This mini router could be seen as the first step for a more complex one. For example, we could add more output channels and even add a FIFO queue in order to not discard the data of the links that have not been selected. This is what happen in this project: the link that is not selected, after the Round Robin choice is discarded.

We should see that "mini router" refers to many meanings, not just to *Internet* world. On the Internet we can find a more complex technology that has some relations with this project, the **Network on a Chip**. It is a network-based communications subsystem on an integrated circuit.

NoC technology applies the theory and methods of computer networking to on-chip communication and brings notable improvements over conventional bus and crossbar communication architectures. Networks-on-chip come in many network topologies, many of which are still experimental as of 2018.

For example, a very common NoC used in personal computers is a graphics processing unit (GPU).[1].

[1] Source: Wikipedia