Computer Engineering

Electronics and Communication Systems

Mini Router

Project Documentation

Edoardo Geraci

2022/2023

Table of Contents

1 Introduction

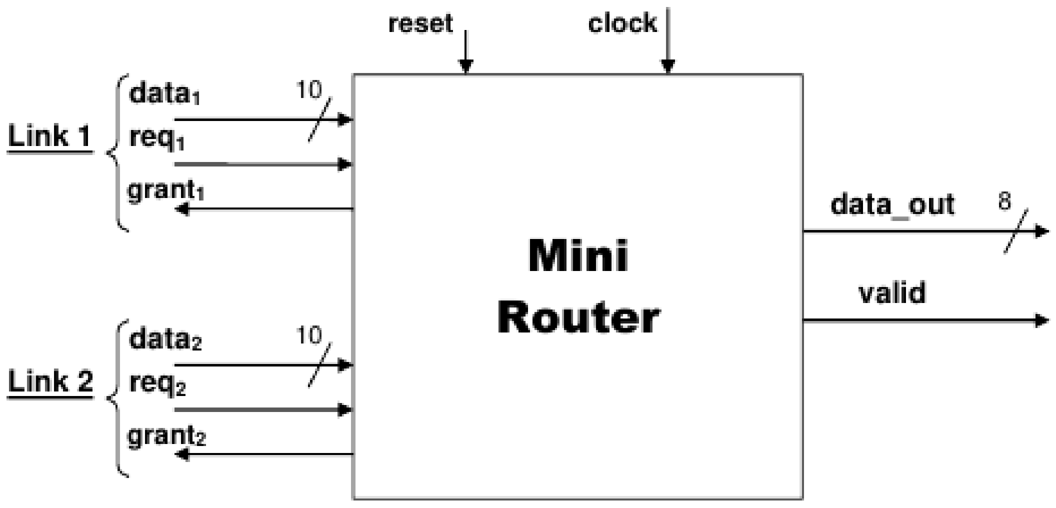
1.1 Assignment:

The project involves designing and implementing a synchronous mini-router using VHDL. The mini-router has two input sources, or "links." Each link consists of two values: "data" and "req."

The "data" value is a 10-bit vector, where the two most significant bits represent the priority of the link, and the other eight bits are the actual data being transmitted. The "req" signal represents the transmission request of the link: if "req" is high, the link is transmitting data, while if it is low, the link is ignored.

For each link, the mini-router has an output signal called "grant", which indicates to the link whether its data has been selected or not. The mini-router selects which link to use based on the following logic:

- If only one of the two "req" signals is high in a clock cycle, then the data from that link is propagated to the output.
- If both "req" signals are high, the mini-router selects the link with the higher priority. If both links have the same priority (a "data conflict"), then the mini-router uses a Round Robin algorithm to select which link to use. After the reset, the first conflict is resolved by selecting link 1, the second conflict is resolved by selecting link 2, and so on.



The data is propagated without the priority bits and remains valid for one clock cycle.

1.2 Possible Applications:

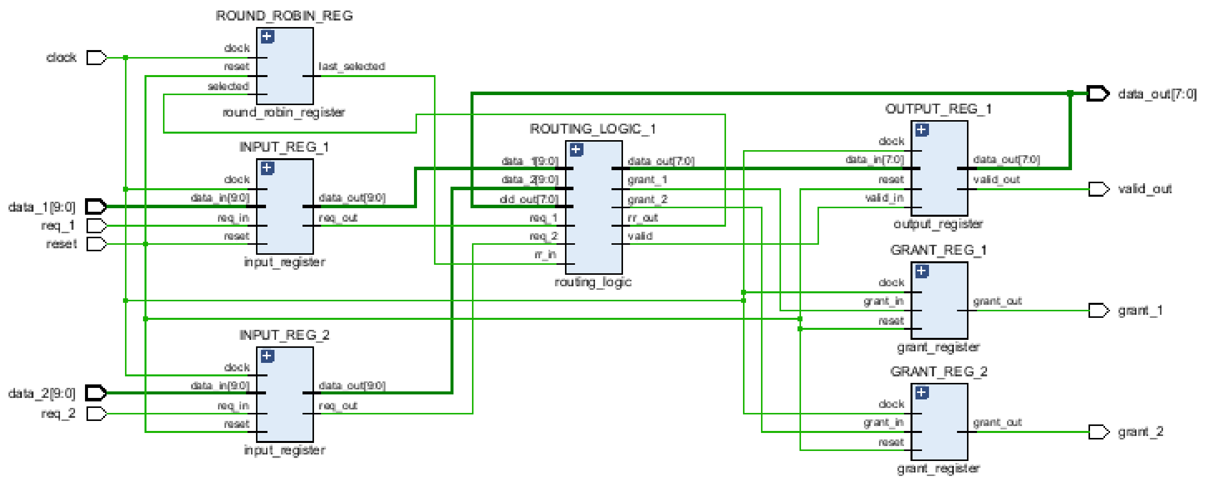
This circuit is primarily used in systems that require the multiplexing of two signals that implement priority. The circuit, as implemented in this project, is not modular, but it could become modular by adding a grant input to the mini-router. With this modification, we could use multiple copies of the mini-router to implement a routing logic for n links. However, to implement the same logic as the original mini-router, we would need to modify the handling of the Round Robin logic to suit the specific needs of the system.

1.3 Possible Architectures:

The mini-router could be implemented as simple combinational logic, but this assumes that all inputs required by the system are stable for the duration of a clock and are ready $t_p + t_{hold}$ before the rising edge of the clock.

However, since we have no information regarding the type of system that uses the mini-router, we can add registers to memorize inputs and propagate outputs. This not only makes the mini-router more reliable but also reduces the critical path of the developed system.

2 Architecture Description



As can be seen in more detail in the following chapters, the architecture relies on a central component that implements the routing logic described in the specifications, as well as six registers that are used as input and output registers.

2.1 Input Register

```
library IEEE;
 use IEEE.std_logic_1164.all;
entity input_register is
 generic (
     N : natural := 10
 );
 port (
   -- general inputs
   reset     : in std_logic;
   clock     : in std_logic;
   -- inputs
   data_in   : in std_logic_vector(N - 1 downto 0);
   req_in    : in std_logic;
   -- outputs
   data_out  : out std_logic_vector(N - 1 downto 0);
   req_out   : out std_logic
 );
end entity;

architecture behavioral of input_register is
begin
   on_clock: process(clock, reset)
   begin
       -- reset

if(reset = '1') then
           data_out <= (others => '0');
           req_out <= '0';
       elsif (rising_edge(clock)) then
           data_out <= data_in;
           req_out <= req_in;
       end if;
   end process;
end architecture;
```

The reset is handled asynchronously and sets the data_out and req_out to 0. On every rising edge of the clock, the component samples the inputs from the links and propagates them to the combinational logic implemented in the routing_logic component.

2.2 Round Robin Register

```vhdl
-- this element implements the round robin scheduling needed when the two
input links have the same
-- priority
library IEEE;
 use IEEE.std_logic_1164.all;

entity round_robin_register is
 generic (
     N : natural := 10
 );
 port (
   -- general inputs
   reset         : in std_logic;
   clock         : in std_logic;
   -- inputs
   selected      : in std_logic;
   -- outputs
   last_selected : out std_logic
 );
end entity;

architecture behavioral of round_robin_register is
begin
   on_clock: process(clock, reset)
   begin
       -- reset
       if(reset = '1') then
           last_selected <= '0';
       elsif (rising_edge(clock)) then
           if(selected = '1')then
             last_selected <= '1';
           else
             last_selected <= '0';
           end if;
       end if;
   end process;
end architecture;
```

As mentioned earlier, if both input signals have the same priority and the req signals are up, the system must implement round-robin logic to determine which input should be propagated.

This is achieved by using a register to store the last selected input in this scenario. The logic behind it is straightforward: the component takes the selected input from the routing logic as input and uses that data as output for the logic itself. Since we only have two inputs, the information about the last selected input is stored using just one bit ('0' for input 1, '1' for input 2).

This way, the logic can always determine the last selected input in the round-robin scenario and act accordingly. The reset is handled asynchronously and sets the last_selected bit to 0, thus resetting the round-robin logic.

2.3 Routing Logic

```vhdl
library IEEE;
  use IEEE.std_logic_1164.all;
entity routing_logic is
  generic (
      N_in : natural := 10;
      N_out : natural := 8
  );
  port (
      -- link_1
      data_1      : in std_logic_vector(N_in - 1 downto 0);
      req_1       : in std_logic;
      grant_1   : out std_logic;
      -- link_2
      data_2      : in std_logic_vector(N_in - 1 downto 0);
      req_2       : in std_logic;
      grant_2   : out std_logic;
      -- data round robin
      rr_in       : in std_logic;
      rr_out      : out std_logic;
      -- data outputs
      old_out   : in std_logic_vector(N_out - 1 downto 0);
      data_out  : out std_logic_vector(N_out - 1 downto 0);
      valid       : out std_logic
  );
end entity;
```

```vhdl
architecture struct of routing_logic is
begin
  -- data output selection
  data_out <= data_1(N_in - 3 downto 0) when
        (
           (req_1 = '1' and req_2 = '0') or
           (req_1 = '1' and (data_1(N_in - 1 downto N_in - 2) > data_2(N_in - 1 downto N_in - 2))) or
           (req_1 = '1' and (data_1(N_in - 1 downto N_in - 2) = data_2(N_in - 1 downto N_in - 2)) and rr_in = '0')
         )
         else data_2(N_in - 3 downto 0) when
         (
           (req_2 = '1' and req_1 = '0') or
           (req_2 = '1' and (data_2(N_in - 1 downto N_in - 2) > data_1(N_in - 1 downto N_in - 2))) or
           (req_2 = '1' and (data_2(N_in - 1 downto N_in - 2) = data_1(N_in - 1 downto N_in - 2)) and rr_in = '1')
         )
         else old_out;
  -- grant up
  grant_1 <= '1' when
         (
           (req_1 = '1' and req_2 = '0') or
           (req_1 = '1' and (data_1(N_in - 1 downto N_in - 2) > data_2(N_in - 1 downto N_in - 2))) or
           (req_1 = '1' and (data_1(N_in - 1 downto N_in - 2) = data_2(N_in - 1 downto N_in - 2)) and rr_in = '0')
         )
         else '0';
  grant_2 <= '1' when
         (
           (req_2 = '1' and req_1 = '0') or
           (req_2 = '1' and (data_2(N_in - 1 downto N_in - 2) > data_1(N_in - 1 downto N_in - 2))) or
           (req_2 = '1' and (data_2(N_in - 1 downto N_in - 2) = data_1(N_in - 1 downto N_in - 2 and rr_in = '1')
         )
         else '0';
  -- valid up
  valid <= '1' when (req_1 = '1' or req_2 = '1') else '0';

  -- round robin handling
  rr_out <= '0'  when
         (
           req_1 = '1' and req_2 = '1' and (data_2(N_in - 1 downto N_in - 2) = data_1(N_in - 1 downto N_in - 2))
           and rr_in = '1'
         )
         else '1' when
         ( req_1 = '1' and req_2 = '1' and (data_2(N_in - 1 downto N_in - 2) = data_1(N_in - 1 downto N_in - 2))
           and rr_in = '0'
         )
         else rr_in;
end architecture;
```

This is the core component of the system. It takes input data and req signals from both links, as well as the last selected link for the round-robin logic and the last output produced. The combinatory logic implements the desired behavior.

If neither input is selected (for example, if both req signals are '0'), the data remains the same as the last produced output, and only the valid signal changes (becoming '0'). This is possible due to the existence of the valid signal itself, which communicates the validity of the output without modifying the data.

Otherwise, everything works as described in the system description. The grant signal is propagated to a grant register, and the data and valid signal are propagated to the output register.

2.4 Grant Register

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

-- basically this is a D Flip-Flop
entity grant_register is
  port (
      -- general inputs
      reset       : in std_logic;
      clock       : in std_logic;
      -- inputs
      grant_in    : in std_logic;
      -- outputs
      grant_out   : out std_logic
  );
end entity;

architecture behavioral of grant_register is
begin
      on_clock: process(clock, reset)
      begin
      -- reset
      if(reset = '1') then
            grant_out <= '0';
      elsif (rising_edge(clock)) then
            grant_out <= grant_in;
      end if;
      end process;
end architecture;
```

2.5 Output Register

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

entity output_register is
  generic (
      N : natural := 8
  );
  port (
      -- general inputs
      reset        : in std_logic;
      clock        : in std_logic;
      -- inputs
      data_in    : in std_logic_vector(N - 1 downto 0);
      valid_in   : in std_logic;
      -- outputs
      data_out   : out std_logic_vector(N - 1 downto 0);
      valid_out    : out std_logic
  );
end entity;

architecture behavioral of output_register is
begin
      on_clock: process(clock, reset)
      begin
      -- reset
      if(reset = '1') then
            data_out <= (others => '0');
            valid_out <= '0';
      elsif (rising_edge(clock)) then
            data_out <= data_in;
            valid_out <= valid_in;
      end if;
      end process;
end architecture;
```

3 Test Strategy

3.1 Test plan:

The behaviors to be tested are:

- No link requests communication;
- Only one link requests communication;
- Both the links request communication with different priorities;
- Both the links request communication with the same priority.

3.2 Inputs and desired Outputs:

Only one req signal high, the priority must be ignored and the link with the req signal high must be served	
data 1: 11|11111111 req 1: 1 data 2: 11|11111111 req 2: 0	output data: 11111111 valid: 1 grant 1: 1 grant 2: 0
Same, with the other channel	
data 1: 11|11111111 req 1: 0 data 2: 00|00000000 req 2: 1	output data: 00000000 valid: 1 grant 1: 0 grant 2: 1
No req signal is high. Valid must be down and the output data remains the same as before	
data 1: 10|10101010 req 1: 0 data 2: 10|10101010 req 2: 0	output data: 00000000 valid: 0 grant 1: 0 grant 2: 0
Start of the priority tests	
data 1: 01|11111111 req 1: 1 data 2: 00|00000000 req 2: 1	output data: 11111111 valid: 1 grant 1: 1 grant 2: 0

data 1: 10|00000000 req 1: 1 data 2:01|11111111 req 2: 1	output data: 00000000 valid: 1 grant 1: 1 grant 2: 0
The round robin logic is triggered for the first time	
data 1: 11|11111111 req 1: 1 data 2: 11|00000000 req 2: 1	output data: 11111111 valid: 1 grant 1: 1 grant 2: 0
data 1: 10|11111111 req 1: 1 data 2:11|00000000 req 2: 1	output data: 00000000 valid: 1 grant 1: 0 grant 2: 1
data 1: 11|11111111 req 1: 0 data 2:10|00000000 req 2: 0	output data: 00000000 valid: 0 grant 1: 0 grant 2: 0
The round robin logic is triggered for the second time	
data 1: 11|11111111 req 1: 1 data 2: 11|10101010 req 2: 1	output data: 10101010 valid: 1 grant 1: 0 grant 2: 1

3.3 Testbench and results

The following code is the most remarkable part of the testbench, the one that implements the tests described earlier:

```vhdl
if (rising_edge(clk_tb)) then
  case (clock_cycle) is
      when 0 =>
            reset_tb <= '0';
      when 1 =>
            req_1_tb <= '1';
            req_2_tb <= '0';
            data_1_tb <= (9 downto 0 => '1');
            data_2_tb <= (9 downto 0 => '1');
      -- data_out:11111111|valid:1|grant_1:1|grant_2:0
      when 2 =>
            req_1_tb <= '0';
            req_2_tb <= '1';
            data_1_tb <= (9 downto 0 => '1');
            data_2_tb <= (9 downto 0 => '0');
      -- data_out:00000000|valid:1|grant_1:0|grant_2:1
      when 3 =>
            req_1_tb <= '0';
            req_2_tb <= '0';
            data_1_tb <= "1010101010";
            data_2_tb <= "1010101010";
      -- data_out:00000000|valid:0|grant_1:0|grant_2:0

      -- start of the priority tests
      when 4 =>
            req_1_tb <= '1';
            req_2_tb <= '1';
            data_1_tb <= "01"&(7 downto 0 => '1');
            data_2_tb <= (9 downto 0 => '0');
      -- data_out:11111111|valid:1|grant_1:1|grant_2:0
      when 5 =>
            req_1_tb <= '1';
            req_2_tb <= '1';
            data_1_tb <= "10"&(7 downto 0 => '0');
            data_2_tb <= "01"&(7 downto 0 => '1');
      -- data_out:00000000|valid:1|grant_1:1|grant_2:0
      -- FIRST RR
      when 6 =>
            req_1_tb <= '1';
            req_2_tb <= '1';
            data_1_tb <= "11"&(7 downto 0 => '1');
            data_2_tb <= "11"&(7 downto 0 => '0');
      -- data_out:11111111|valid:1|grant_1:1|grant_2:0
      when 7 =>
            req_1_tb <= '1';
            req_2_tb <= '1';
            data_1_tb <= "10"&(7 downto 0 => '1');
            data_2_tb <= "11"&(7 downto 0 => '0');
      -- data_out:00000000|valid:1|grant_1:0|grant_2:1
      when 8 =>
            req_1_tb <= '0';
            req_2_tb <= '0';
            data_1_tb <= "11"&(7 downto 0 => '0');
            data_2_tb <= "10"&(7 downto 0 => '1');
      -- data_out:00000000|valid:0|grant_1:0|grant_2:0
      when 9 =>
            req_1_tb <= '1';
            req_2_tb <= '1';
            data_1_tb <= "1011111111";
            data_2_tb <= "1010101010";
      -- data_out:10101010|valid:1|grant_1:0|grant_2:1
      when others =>
            run_simulation <= '0';
  end case;
  clock_cycle := clock_cycle + 1;
end if;
```

The results are shown in fig 3.3.1.
The delay of 1 clock cycle in the output is due to the presence of output registers.

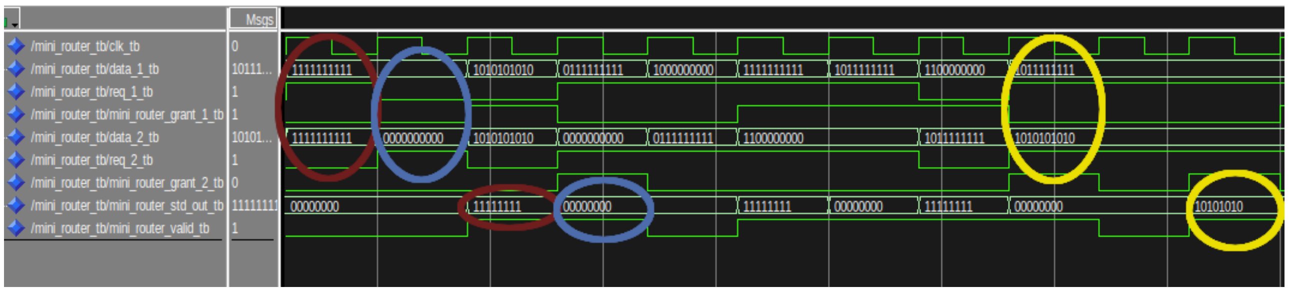


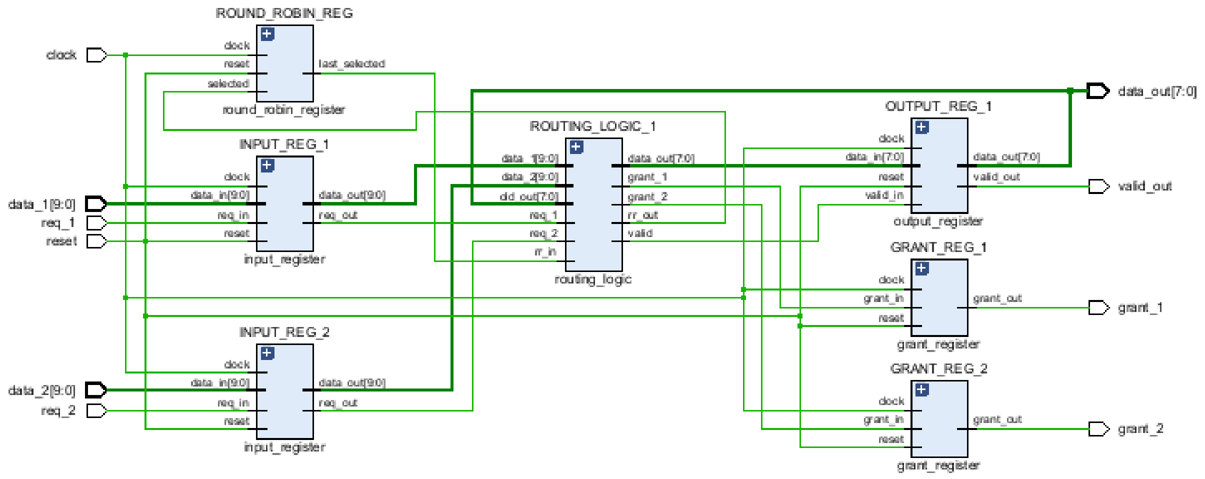
fig 3.3.1

Some of the results were highlighted, such as the first test for the functioning of the req signals (red and blue) and the second occurrence of the round robin logic.

4 Logic Synthesis

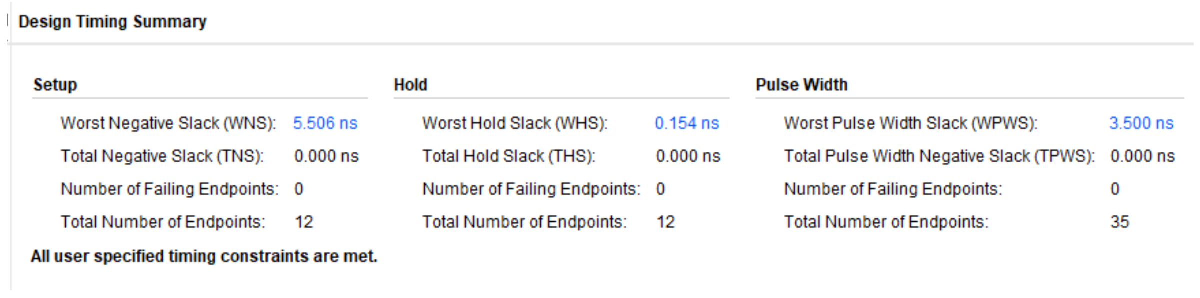
4.1 Synthesis

The logic synthesis was performed using Xilinx Vivado 2022.2 with no issues to be found. The design produced is the following:



4.1.1 Critical Path

When using a clock of 125 MHz, we observe that the critical path (worst negative slack) is significantly greater than 0. This not only indicates that the selected clock is suitable, but also suggests that the mini router can be used with higher frequency signals.



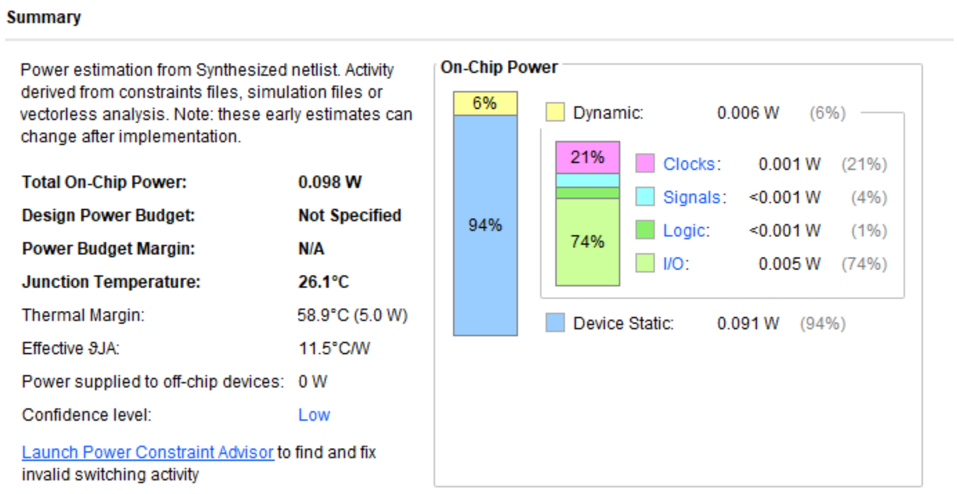
Design Timing Summary

Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	5.506 ns	Worst Hold Slack (WHS):	0.154 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:	35

All user specified timing constraints are met.

4.1.2 Power Consumption

The following is the summary report regarding power consumption:


Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.098 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.1°C**
Thermal Margin:	58.9°C (5.0 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:		0.006 W	(6%)
	Clocks:	0.001 W	(21%)
	Signals:	<0.001 W	(4%)
	Logic:	<0.001 W	(1%)
	I/O:	0.005 W	(74%)
Device Static:		0.091 W	(94%)

The power consumption summary divides it into two categories: static and dynamic. The majority of the energy (94%) is spent on static consumption, while the remaining 6% is spent on dynamic consumption.

The majority of the dynamic consumption is related to I/O, which is understandable given that our component will likely change its input and output every clock cycle, while the actual logic is relatively simple.

5 Conclusion

Overall, this component is relatively simple yet interesting to analyze. The main idea can be extended to an N-link router with some modifications and additions, making it much more useful than a router designed only for 2 links. However, this component can still be used in systems that require choosing between two different sources using a predefined logic. This logic could be even more complex than the one analyzed here, and yet be implemented with only a few input and output registers and a central combinatory logic.

As a result, this component is reliable and capable of handling high-frequency signals.