WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Designing an 8-Bit Computer using the VHDL Language

Nguyen Chi Vi[1], Chu Manh Tuyen[2], Le Ngoc Giang[3*]

[1] Master, Head of the Department of Electrical Engineering Fundamentals, Faculty of Fundamental Technics, Air Force Officer's College of Viet Nam.
[2] Master, Lecturer, Metrology Department, Faculty of Fundamental Technical, AD-AF Academy of Viet Nam, Son Tay, Ha Noi, Viet Nam
[3] PhD, Head of Metrology Department, Faculty of Fundamental Technical, AD-AF Academy of Viet Nam, Son Tay, Ha Noi, Viet Nam
DOI: https://doi.org/10.55248/gengpi.5.0324.0752

A B S T R A C T

Very High Speed Integrated Circuit Hardware Description Language (VHDL) is one of the most popular and widely used hardware description languages (HDL). VHDL provides a powerful means to describe electronic circuits and digital systems, enabling engineers and researchers to design, simulate, and test circuits efficiently. This article focuses on guiding students through the process of designing a simple 8-bit computer using the VHDL language. First, the IC system is described in VHDL using IC design software sponsored by Xilinx and Altera. Finally, the described circuit will be tested for functionality through simulation software before being loaded and run on the FPGA kit. The results of this research contribute to the development and application of VHDL in the field of IC design.

Keywords: VHDL, hardware description languages, 8-bit computer, FPGA kit, IC design

1. Introduction

In recent years, the remarkable development of electronic technology has opened up a new world of Very Large Scale Integrated (VLSI) circuits capable of integrating millions of transistors. This advancement has led to numerous new applications in information technology, electronics, telecommunications, automation, and other fields, meeting growing societal needs. In this context, application-specific integrated circuit (ASIC) technology has emerged as an alternative to traditional digital systems, aiding in reducing time and costs in the research and production processes. Additionally, Field Programmable Gate Arrays (FPGA) and Complex Programmable Logic Devices (CPLD) have gained popularity, allowing for the optimization of the design and assembly processes while providing high flexibility.

To describe the overall structure of a computer system, high-level block diagrams are utilized. These diagrams display the main components and their relationships within the system. They often include blocks representing crucial components, such as:

Central Processing Unit (CPU): Serving as the heart of the system, the CPU performs calculations and controls system operations.

Memory: consisting of main memory (RAM) and storage memory (ROM), where data and programs are stored.

To illustrate detailed operations, this article will employ the design of a simple 8-bit computer system. Figure 1 depicts a block diagram for an 8-bit computer system.

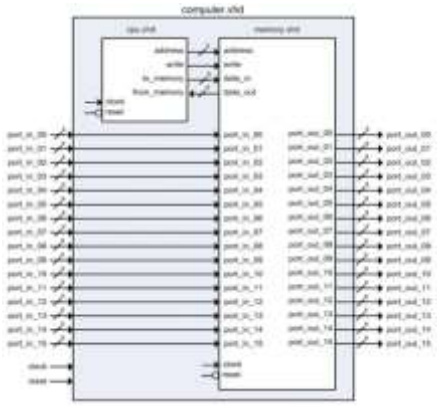


Fig.1- Block diagram of an 8-bit computer

2. Build the internal blocks of the computer

2.1 Build the computer's memory

The memory system includes program memory, data memory, and input/output ports. Figure 2 shows the block diagram of the memory system.

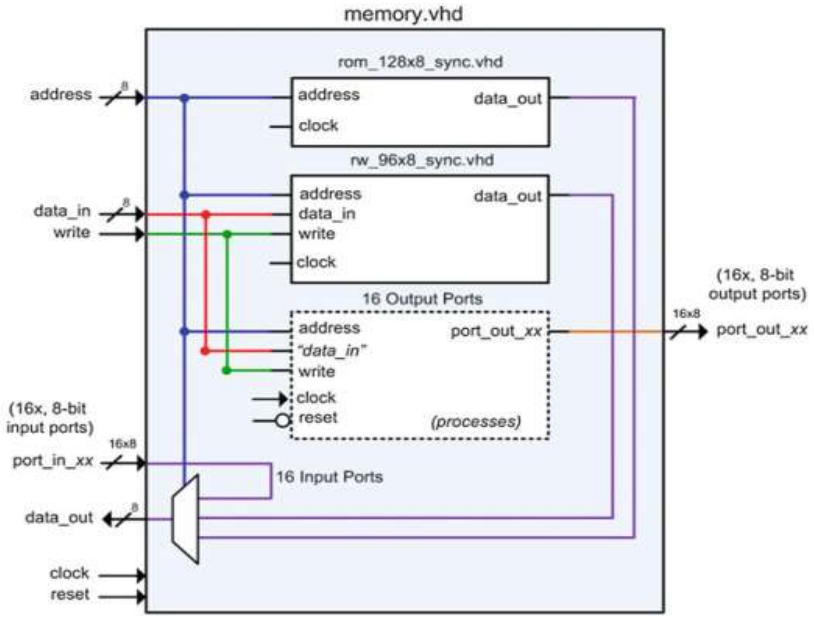


Fig.2- Block diagram of the memory system

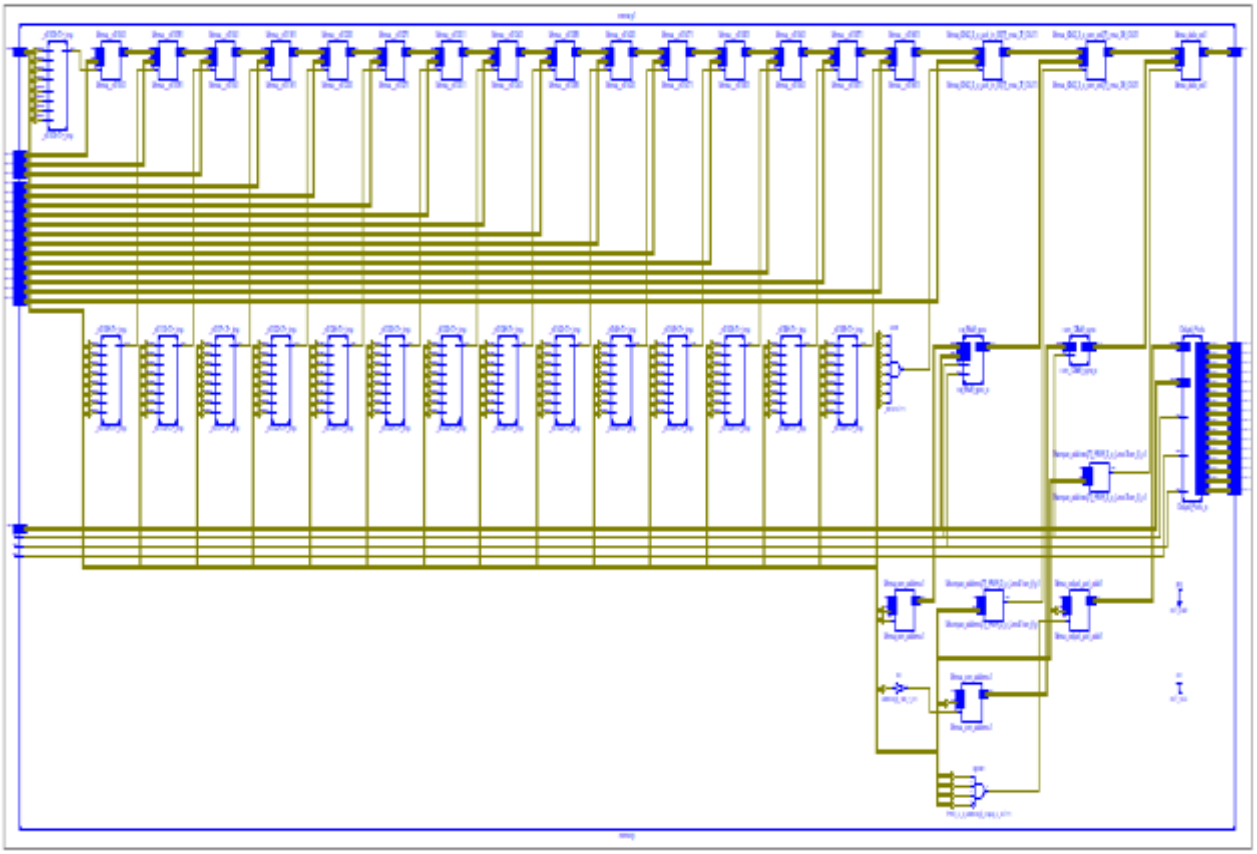
Program memory and data memory will be implemented using low-level components (rom_128x8_sync.vhd and rw_96x8_sync.vhd), while input and output ports can be emulated using a combination of RTL and combinational logic processes. The program memory and data memory components contain separate circuitry to handle their address ranges. Each output port also contains its own circuitry to handle its unique address. A switch is used to handle the routing of signals back to the CPU based on the provided address.

The computer's memory is composed of three components: Program memory (rom_128x8), data memory (rw_96x8) and output ports. Below is the program for the computer's memory:

```
            library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity memory is
   port (
      address : in  std_logic_vector(7 downto 0);
      data_in : in  std_logic_vector(7 downto 0);
      write   : in  std_logic;
      clock   : in  std_logic;
      reset   : in  std_logic;
      data_out: out std_logic_vector(7 downto 0);
      port_in : in  std_logic_vector(15 downto 0);
      port_out: out std_logic_vector(15 downto 0)
   );
end memory;
architecture Behavioral of memory is
   component rom_128x8_sync
```

```
    port (
      address : in  std_logic_vector(6 downto 0);
      data_out: out std_logic_vector(7 downto 0);
      clock   : in  std_logic
    );
  end component;
  component rw_96x8_sync
    port (
      address : in  std_logic_vector(6 downto 0);
      data_in : in  std_logic_vector(7 downto 0);
      write   : in  std_logic;
      clock   : in  std_logic;
      data_out: out std_logic_vector(7 downto 0)
    );
  end component;
  component Output_Ports
    port (
      address    : in  std_logic_vector(3 downto 0);
      data_in    : in  std_logic_vector(7 downto 0);
      write      : in  std_logic;
      clock      : in  std_logic;
      reset      : in  std_logic;
      port_out   : out std_logic_vector(7 downto 0)
    );
  end component;
  signal rom_out, ram_out        : std_logic_vector(7 downto 0);
  signal output_port_addr        : std_logic_vector(3 downto 0);
  signal ram_address, rom_address: std_logic_vector(6 downto 0);
begin
  ram_address <= address(6 downto 0) when address(7) = '1' else "0000000";
  rom_address <= address(6 downto 0) when address(7) = '0' else "0000000";
  output_port_addr <= address(3 downto 0) when address(7 downto 4) = x"E" else "0000";
  rom_128x8_sync_u: rom_128x8_sync port map (
    address => rom_address,
    clock   => clock,
    data_out=> rom_out
  );
  rw_96x8_sync_u: rw_96x8_sync port map (
    address => ram_address,
```

```
        data_in => data_in,
        write   => write,
        clock   => clock,
        data_out=> ram_out
    );
    Output_Ports_u: Output_Ports port map (
        address   => output_port_addr,
        data_in   => data_in,
        write     => write,
        clock     => clock,
        reset     => reset,
        port_out  => port_out(to_integer(unsigned(output_port_addr)))
    );
    data_out <=
        rom_out when address < x"80" else
        ram_out when address < x"E0" else
        port_in(to_integer(unsigned(address(7 downto 4)))) when address(7 downto 4) = x"F" else
        x"00";
end Behavioral;
```



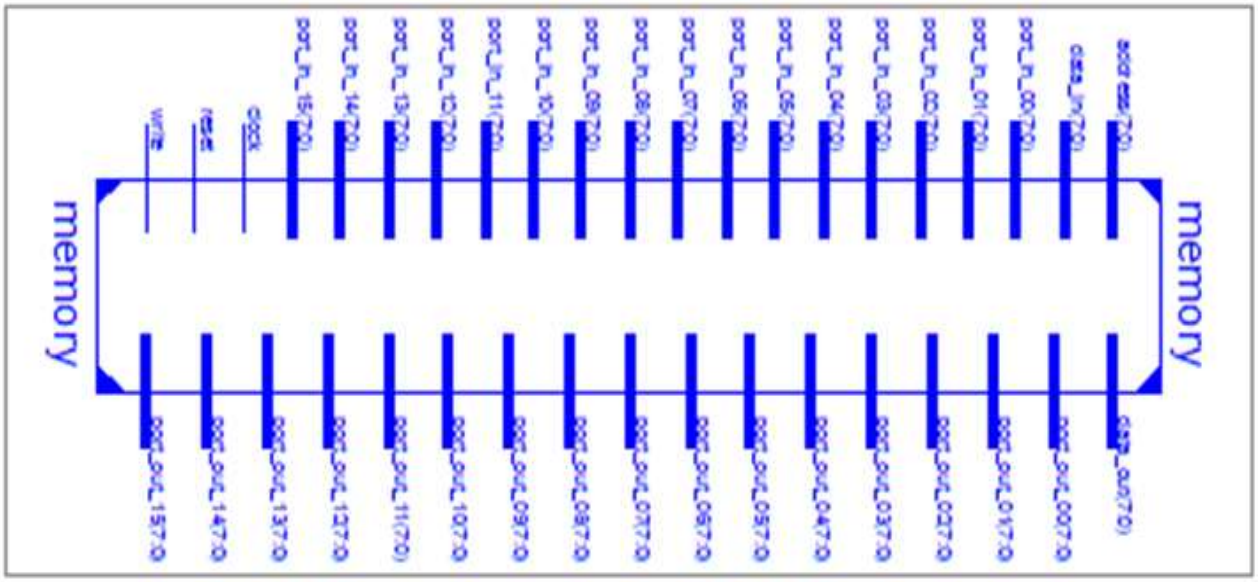


Figure 3. Memory structure diagram described using VHDL

Build the CPU in VHDL

The CPU consists of two components: the control unit (CU) and the data path (data_path). The data path contains all the registers and the ALU (Arithmetic Logic Unit). The ALU is implemented as a child component in the datapath (alu.vhd). The data path also contains a bus system to facilitate the movement of data between registers and memory. The bus system is implemented with two multichannel switches controlled by a control unit. The control unit contains a finite-state machine that generates all control signals for the data path as it performs the fetch-decode-execute steps of each instruction. Figure 4 shows the block diagram of a CPU in a simple 8-bit computer.

Fig.4- CPU structure diagram of a simple 8-bit computer

a) Build a CPU program in VHDL

A computer's CPU is composed of two components: the control unit (CU) and the data path (data_path). The data path contains all the registers and the ALU (Arithmetic Logic Unit). Below is the program for the computer's CPU:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity cpu is
   port (
      clock     : in  std_logic;
      reset     : in  std_logic;
      address   : out std_logic_vector(7 downto 0);
      from_memory: in  std_logic_vector(7 downto 0);
      write     : out std_logic;
      to_memory  : out std_logic_vector(7 downto 0)
   );
end cpu;
architecture Behavioral of cpu is
   component control_unit is
      port (
         clock     : in  std_logic;
```

```
        reset     : in  std_logic;
        IR_Load   : out std_logic;
        IR        : in  std_logic_vector(7 downto 0);
        MAR_Load  : out std_logic;
        PC_Load   : out std_logic;
        PC_Inc    : out std_logic;
        A_Load    : out std_logic;
        B_Load    : out std_logic;
        ALU_Sel   : out std_logic_vector(2 downto 0);
        CCR_Result: in  std_logic_vector(3 downto 0);
        CCR_Load  : out std_logic;
        Bus2_Sel  : out std_logic_vector(1 downto 0);
        Bus1_Sel  : out std_logic_vector(1 downto 0);
        write     : out std_logic
    );
end component;
component data_path is
    port (
        clock     : in  std_logic;
        reset     : in  std_logic;
        IR_Load   : in  std_logic;
        IR        : out std_logic_vector(7 downto 0);
        MAR_Load  : in  std_logic;
        address   : out std_logic_vector(7 downto 0);
        PC_Load   : in  std_logic;
        PC_Inc    : in  std_logic;
        A_Load    : in  std_logic;
        B_Load    : in  std_logic;
        ALU_Sel   : in  std_logic_vector(2 downto 0);
        CCR_Result: out std_logic_vector(3 downto 0);
        CCR_Load  : in  std_logic;
        Bus2_Sel  : in  std_logic_vector(1 downto 0);
        Bus1_Sel  : in  std_logic_vector(1 downto 0);
        from_memory: in  std_logic_vector(7 downto 0);
        to_memory  : out std_logic_vector(7 downto 0)
    );
end component;
signal IR_Load, MAR_Load, PC_Load, PC_Inc, A_Load, B_Load, CCR_Load: std_logic;
signal IR: std_logic_vector(7 downto 0);
```

```
    signal ALU_Sel, CCR_Result, Bus2_Sel, Bus1_Sel: std_logic_vector(2 downto 0);
    signal write: std_logic;
begin
    control_unit_module: control_unit port map (
        clock      => clock,
        reset      => reset,
        IR_Load    => IR_Load,
        IR         => IR,
        MAR_Load   => MAR_Load,
        PC_Load    => PC_Load,
        PC_Inc     => PC_Inc,
        A_Load     => A_Load,
        B_Load     => B_Load,
        ALU_Sel    => ALU_Sel,
        CCR_Result => CCR_Result,
        CCR_Load   => CCR_Load,
        Bus2_Sel   => Bus2_Sel,
        Bus1_Sel   => Bus1_Sel,
        write      => write
    );
    data_path_u: data_path port map (
        clock       => clock,
        reset       => reset,
        IR_Load     => IR_Load,
        IR          => IR,
        MAR_Load    => MAR_Load,
        address     => address,
        PC_Load     => PC_Load,
        PC_Inc      => PC_Inc,
        A_Load      => A_Load,
        B_Load      => B_Load,
        ALU_Sel     => ALU_Sel,
        CCR_Result  => CCR_Result,
        CCR_Load    => CCR_Load,
        Bus2_Sel    => Bus2_Sel,
        Bus1_Sel    => Bus1_Sel,
        from_memory => from_memory,
        to_memory   => to_memory
    );
```

end Behavioral;

b) CPU structure diagram described by VHDL

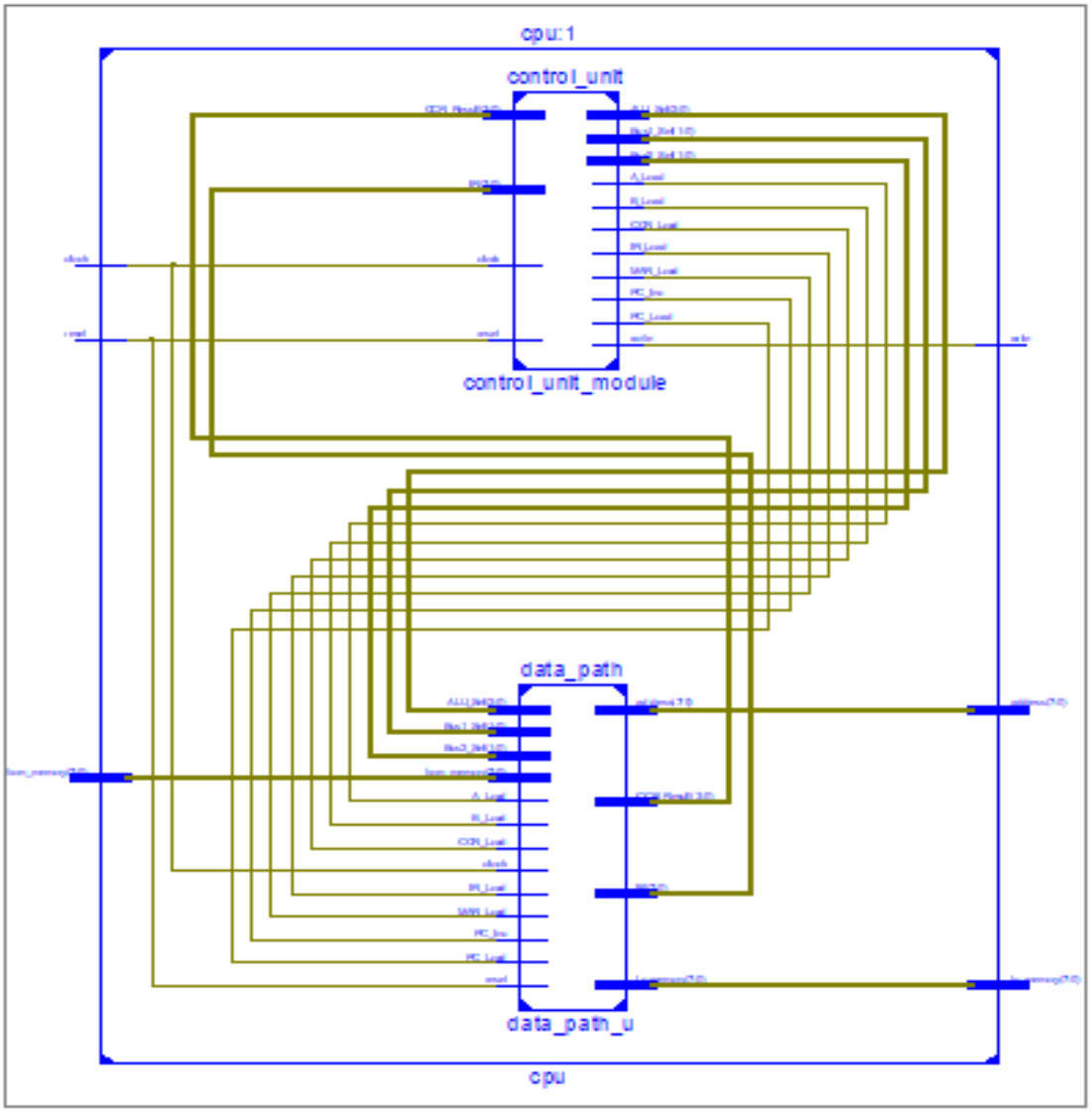


Fig.5- CPU structure diagram described by VHDL

3. Design and simulate the operation of a simple 8-bit computer

3.1 Design a simple 8-bit computer on VHDL

a) Program on VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity computer is
   port (
      clock       : in  std_logic;
      reset       : in  std_logic;
      port_in     : in  std_logic_vector(15 downto 0);
      port_out    : out std_logic_vector(15 downto 0)
```

```
    );
end computer;
architecture Behavioral of computer is
    component cpu is
        port (
            clock      : in  std_logic;
            reset      : in  std_logic;
            address    : out std_logic_vector(7 downto 0);
            from_memory: in  std_logic_vector(7 downto 0);
            write      : out std_logic;
            to_memory  : out std_logic_vector(7 downto 0)
        );
    end component;
    component memory is
        port (
            address   : in  std_logic_vector(7 downto 0);
            data_in   : in  std_logic_vector(7 downto 0);
            write     : in  std_logic;
            data_out  : out std_logic_vector(7 downto 0)
        );
    end component;
    signal address, data_in, data_out: std_logic_vector(7 downto 0);
    signal write: std_logic;
begin
    cpu_u: cpu port map (
        clock       => clock,
        reset       => reset,
        address     => address,
        write       => write,
        to_memory   => data_in,
        from_memory => data_out
    );
    memory_unit: memory port map (
        address  => address,
        data_in  => data_in,
        write    => write,
        data_out => data_out
    );
    port_out <= port_in;
```

end Behavioral;

b) Block diagram depicting a simple 8-bit computer on VHDL

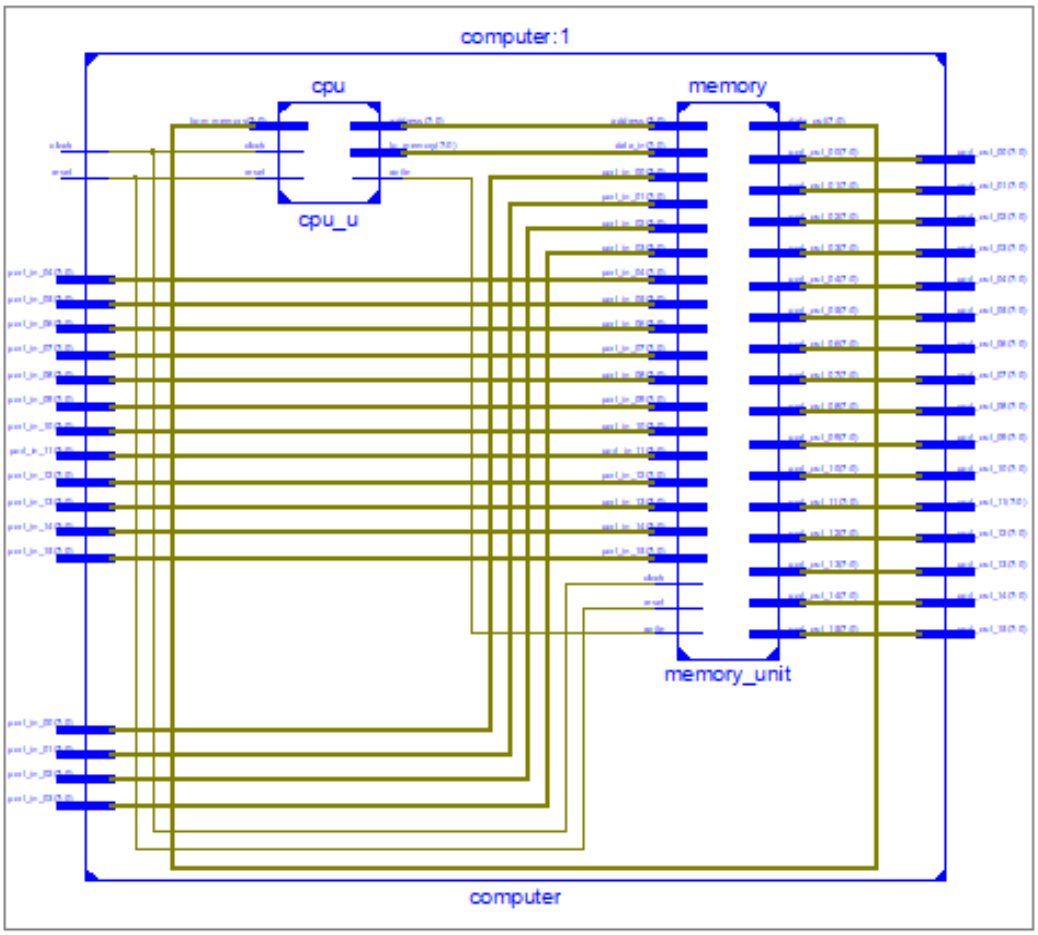


Fig.6- Block diagram depicting a simple 8-bit computer on VHDL

Simulate the operation of a simple 8-bit computer

a) Program on VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
entity computer_TB is
end entity;
architecture computer_TB_arch of computer_TB is
  constant t_clk_per : time := 20 ns;  -- Period of a 50MHz Clock
  component computer
    port (
      clock       : in  std_logic;
      reset       : in  std_logic;
      port_in     : in  std_logic_vector(15 downto 0);
      port_out    : out std_logic_vector(15 downto 0)
    );
  end component;
```

```
 signal clock_TB, reset_TB: std_logic;
 signal port_in_TB: std_logic_vector(15 downto 0);
 signal port_out_TB: std_logic_vector(15 downto 0);
begin
 microcontroller_unit : computer
  port map (
   clock    => clock_TB,
   reset    => reset_TB,
   port_in  => port_in_TB,
   port_out => port_out_TB
  );
 CLOCK_STIM : process
 begin
  clock_TB <= '0'; wait for 0.5*t_clk_per;
  clock_TB <= '1'; wait for 0.5*t_clk_per;
 end process;
 RESET_STIM : process
 begin
  reset_TB <= '0'; wait for 0.25*t_clk_per;
  reset_TB <= '1'; wait;
 end process;
 PORT_STIM : process
 begin
  port_in_TB <= x"00112233445566778899AABBCCDDEEFF";
  wait;
 end process;
end architecture;
```

b) Simple 8-bit computer simulation results

When completing the SMC description program, we run the simulation test bench file, checking the internal signals and output of the computer:

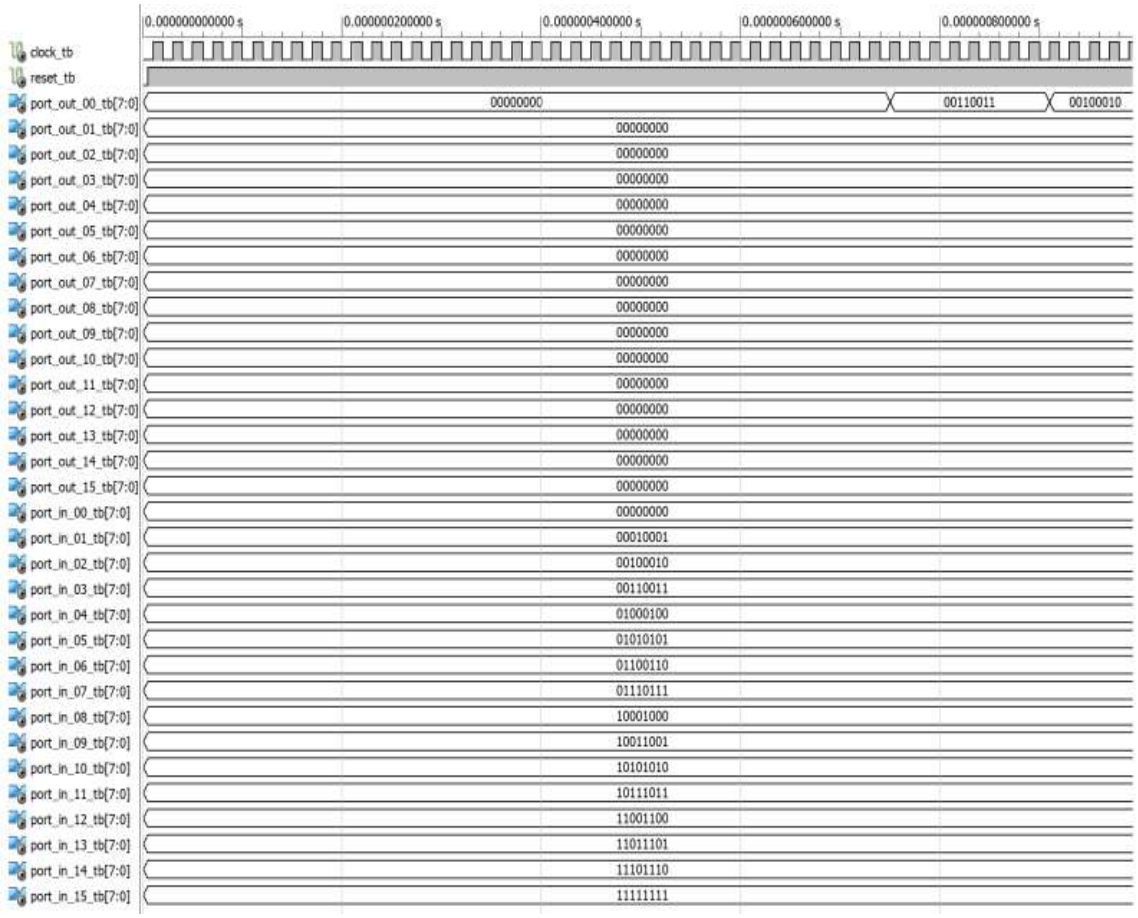


Fig.7- Simulation results of a simple 8-bit computer

Conclusion

In this article, we have examined the important role of the VHDL hardware description language in IC design, especially in building a simple 8-bit computer. By using VHDL, we had the opportunity to effectively simulate and test the circuit before implementing it on the FPGA kit, saving time and increasing the reliability of the design process. The article delves into the operating structure of a simple computer and provides specific instructions on building its basic components using VHDL. Simulation and evaluation of results using ISE software have shown the success of this method and created a solid basis for continued research and development of more complex ICs in the future. . This research not only contributes to the development of the VHDL language in the field of IC design but also provides an important opportunity for students to better understand how computers work through practice design and simulation.

References

E. Ayeh, K. Agbedanu, Y. Morita, O. Adamo, P. Guturu. “FPGA Implementation of an 8-bit Simple Processor”. University of North Texas, Denton, 2008.

V. S. Balakrishnan, H. Pottinger, F. Ercal, M. Agarwal. “Design and Implementation of an FPGA-based Processor for Compressed Images”. In Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays.

R. Fryer. “FPGA-based CPU Instrumentation for Hard Real-time Embedded System Testing”. April 2005, pp. 39-42.

P. Yiannacouras, J. Rose, and J. G. Steffan. “The Microarchitecture of FPGA-based Soft Processors”. In Proceedings of the 2005 International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, pp. 202-212.

G. Achery, C. Trinitis, R. Buchty. “CPU-independent Assembler in an FPGA”. In Field Programmable Logic and Applications, 2005. pp. 519-522.

Y Nagaonkar and M. L. Manwaring. “An FPGA-based Experiment Platform for Hardware-Software Codesign and Hardware Emulation”. In Proceedings of The 2006 World Congress in Computer Science, Compute Engineering, and Applied Computing, pp.169-174.