American University of Ras Al Khaimah

CENG 336 – Computer Architecture Lab

Implementation of a 4-bit Array Multiplier

Team Members:
Amir Beshir (2023006017)
Sinishaw Nasa (2023006199)
Aleksei Kovalev (2023005964)

Instructor:
Engr. Umar Adeel

Course Section: Section 1

Submission Date: xx November 2025

Department of Computer Science and Engineering
American University of Ras Al Khaimah
2025-2026

Contents

1. Introduction

Binary multiplication forms the foundation of arithmetic operations in digital systems and computer architecture. The array multiplier represents an efficient hardware implementation of binary multiplication that parallels the traditional paper-and-pencil method. As illustrated in Figure 1, the multiplication process involves computing partial products by multiplying the multiplicand A with each bit of the multiplier B, then summing these shifted partial products to obtain the final result.

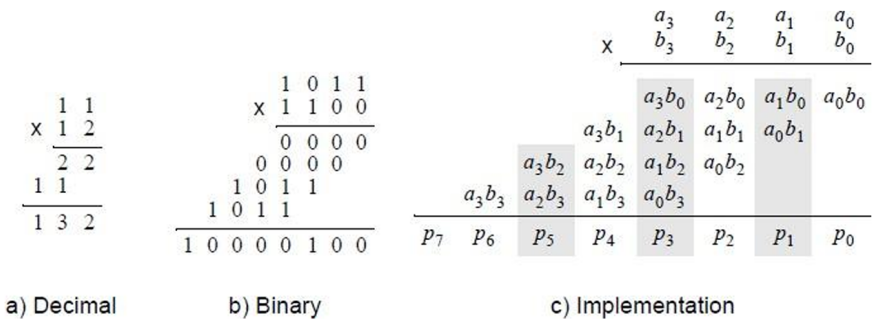


Figure 1: Binary multiplication process showing partial products and their summation

The array multiplier circuit employs a regular structure that efficiently computes the product P = A × B using a combination of AND gates to generate partial products and full adder modules to perform the necessary additions. This design approach provides excellent performance characteristics while maintaining a systematic and scalable architecture suitable for implementation in programmable logic devices.

In this project, we designed and implemented a 4-bit array multiplier circuit using VHDL, targeting the DE2-115 FPGA development board. The design accepts two 4-bit inputs through switches SW7-4 and SW3-0, displays the input values on 7-segment displays HEX2 and HEX0 respectively, and shows the 8-bit multiplication result on displays HEX5-4. The implementation demonstrates practical application of digital design principles while providing hands-on experience with FPGA development tools and hardware implementation.

2. Objectives

- Implement a 4-bit array multiplier circuit using structural VHDL design methodology
- Develop teamwork skills through task partitioning, effective communication, and collaborative problem-solving
- Design and conduct experiments using Quartus simulation tools, analyzing and interpreting experimental data

- Gain hands-on experience with Quartus Prime development environment and DE2-115 FPGA implementation
- Apply digital design principles including modularity, reuse, and functional abstraction
- Verify circuit functionality through both simulation and physical implementation

3. Design Components Description

The array multiplier implementation consists of three main components that work together to perform the multiplication operation and display the results:

3.1. Full Adder Module (fa)

The fundamental building block of the array multiplier is the full adder component. This module takes three binary inputs (a, b, and carry-in ci) and produces two outputs (sum s and carry-out co). The internal implementation uses XOR and AND gates to compute the sum and carry outputs according to the Boolean equations:

$$s = a \oplus b \oplus ci$$
$$co = (a \cdot b) + (b \cdot ci) + (a \cdot ci)$$

3.2. 7-Segment Display Decoder (display7seg)

This component converts 4-bit binary inputs into the appropriate 7-segment display patterns. The architecture uses a CASE statement to map each binary value (0-15) to the corresponding segment activation pattern for hexadecimal display.

3.3. Top-Level Array Multiplier (LabProject)

The main entity integrates the full adder array and display components. It processes two 4-bit inputs from switches, performs the multiplication using a 3×4 array of full adders, and routes the results to the appropriate 7-segment displays. The design uses signals for partial products (ParProB1, ParProB2) and carry propagation (carryb1, carryb2, carryb3) between adder rows.

3.4. Input/Output Configuration

- **Inputs:** SW7-4 represent multiplicand A, SW3-0 represent multiplier B
- **Outputs:** HEX2 displays A, HEX0 displays B, HEX5-4 display product P = A × B
- **Internal Signals:** Partial products and carry signals facilitate the multi-stage addition process

4. VHDL Code Implementation

4.1. Main Entity: LabProject

```
-- CENG 336 Project: 4-bit Array Multiplier
-- Multiplies two 4-bit values and displays result on 7-segment displays
-- Required libraries for digital design
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

-- Main multiplier module interface definition
ENTITY LabProject IS
   PORT ( SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- 8-bit input: A(7-4), B(3-0)
          HEX5, HEX4, HEX2, HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6)); -- Display
              outputs for results
END LabProject;

ARCHITECTURE Structure OF LabProject IS
-- Basic full adder component definition
   COMPONENT fa
      PORT ( a, b, ci : IN STD_LOGIC; -- Full adder input ports
             s, co : OUT STD_LOGIC); -- Sum and carry outputs
   END COMPONENT;

-- 7-segment decoder component
   COMPONENT display7seg
      PORT ( binary : IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- Binary input value
             hex : OUT STD_LOGIC_VECTOR(0 TO 6)); -- 7-segment output pattern
   END COMPONENT;

-- Internal signal declarations for data processing
   SIGNAL A, B : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL P : STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL carryb1 : STD_LOGIC_VECTOR(3 DOWNTO 1); -- Carry chain first stage
   SIGNAL carryb2 : STD_LOGIC_VECTOR(3 DOWNTO 1); -- Carry chain second stage
   SIGNAL carryb3 : STD_LOGIC_VECTOR(3 DOWNTO 1); -- Carry chain third stage
   -- Intermediate partial product results
   SIGNAL ParProB1 : STD_LOGIC_VECTOR(5 DOWNTO 2);
   SIGNAL ParProB2 : STD_LOGIC_VECTOR(6 DOWNTO 3);

   -- Individual bit product terms
```

```
38    SIGNAL a0b0, a1b0, a2b0, a3b0 : STD_LOGIC;
39    SIGNAL a0b1, a1b1, a2b1, a3b1 : STD_LOGIC;
40    SIGNAL a0b2, a1b2, a2b2, a3b2 : STD_LOGIC;
41    SIGNAL a0b3, a1b3, a2b3, a3b3 : STD_LOGIC;
42
43  BEGIN
44  -- Map switch inputs to internal signals
45    A <= SW(7 DOWNTO 4);
46    B <= SW(3 DOWNTO 0);
47
48    -- Generate all partial product bits
49    a0b0 <= A(0) AND B(0);
50    a1b0 <= A(1) AND B(0);
51    a2b0 <= A(2) AND B(0);
52    a3b0 <= A(3) AND B(0);
53
54    a0b1 <= A(0) AND B(1);
55    a1b1 <= A(1) AND B(1);
56    a2b1 <= A(2) AND B(1);
57    a3b1 <= A(3) AND B(1);
58
59    a0b2 <= A(0) AND B(2);
60    a1b2 <= A(1) AND B(2);
61    a2b2 <= A(2) AND B(2);
62    a3b2 <= A(3) AND B(2);
63
64    a0b3 <= A(0) AND B(3);
65    a1b3 <= A(1) AND B(3);
66    a2b3 <= A(2) AND B(3);
67    a3b3 <= A(3) AND B(3);
68
69    P(0) <= a0b0;
70
71    -- Array multiplier structure implementation
72    -- First row of adders processes B1 products
73    a0b1_fa: fa PORT MAP (a1b0, a0b1, '0', P(1), carryb1(1));
74    a1b1_fa: fa PORT MAP (a2b0, a1b1, carryb1(1), ParProB1(2), carryb1(2));
75    a2b1_fa: fa PORT MAP (a3b0, a2b1, carryb1(2), ParProB1(3), carryb1(3));
76    a3b1_fa: fa PORT MAP ('0', a3b1, carryb1(3), ParProB1(4), ParProB1(5));
77
78    -- Second row processes B2 products
```

```
79    a0b2_fa: fa PORT MAP (ParProB1(2), a0b2, '0', P(2), carryb2(1));
80    a1b2_fa: fa PORT MAP (ParProB1(3), a1b2, carryb2(1), ParProB2(3), carryb2(2)
          );
81    a2b2_fa: fa PORT MAP (ParProB1(4), a2b2, carryb2(2), ParProB2(4), carryb2(3)
          );
82    a3b2_fa: fa PORT MAP (ParProB1(5), a3b2, carryb2(3), ParProB2(5), ParProB2
          (6));
83
84    -- Final row processes B3 products
85    a0b3_fa: fa PORT MAP (ParProB2(3), a0b3, '0', P(3), carryb3(1));
86    a1b3_fa: fa PORT MAP (ParProB2(4), a1b3, carryb3(1), P(4), carryb3(2));
87    a2b3_fa: fa PORT MAP (ParProB2(5), a2b3, carryb3(2), P(5), carryb3(3));
88    a3b3_fa: fa PORT MAP (ParProB2(6), a3b3, carryb3(3), P(6), P(7));
89
90    -- Connect outputs to display drivers
91    digit3: display7seg PORT MAP (A, HEX2);
92    digit2: display7seg PORT MAP (B, HEX0);
93    digit1: display7seg PORT MAP (P(7 DOWNTO 4), HEX5);
94    digit0: display7seg PORT MAP (P(3 DOWNTO 0), HEX4);
95 END Structure;
96 caption={Full adder VHDL implementation},label=lst:full_adder]
97 -- Full adder module implementation
98 LIBRARY ieee;
99 USE ieee.std_logic_1164.all;
100
101 ENTITY fa IS
102    PORT ( a, b, ci : IN STD_LOGIC; -- Full adder inputs
103             s, co : OUT STD_LOGIC); -- Result outputs
104 END fa;
105
106 -- Full adder internal logic
107 ARCHITECTURE Structure OF fa IS
108    SIGNAL a_xor_b : STD_LOGIC;
109 BEGIN
110    a_xor_b <= a XOR b;
111    s <= a_xor_b XOR ci;
112    co <= (NOT(a_xor_b) AND b) OR (a_xor_b AND ci);
113 END Structure;
114
115 caption={7-segment display decoder VHDL code},label=lst:display_decoder]
116 -- 7-segment display decoder
```

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY display7seg IS
   PORT ( binary : IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- 4-bit binary input
          hex : OUT STD_LOGIC_VECTOR(0 TO 6)); -- Display segment pattern
END display7seg;

-- Display decoder conversion logic
ARCHITECTURE Behavior OF display7seg IS
BEGIN
 PROCESS (binary)
 BEGIN
    CASE binary IS
       WHEN "0000" => hex <= "0000001"; -- 0
       WHEN "0001" => hex <= "1001111"; -- 1
       WHEN "0010" => hex <= "0010010"; -- 2
       WHEN "0011" => hex <= "0000110"; -- 3
       WHEN "0100" => hex <= "1001100"; -- 4
       WHEN "0101" => hex <= "0100100"; -- 5
       WHEN "0110" => hex <= "0100000"; -- 6
       WHEN "0111" => hex <= "0001111"; -- 7
       WHEN "1000" => hex <= "0000000"; -- 8
       WHEN "1001" => hex <= "0001100"; -- 9
       WHEN "1010" => hex <= "0001000"; -- A
       WHEN "1011" => hex <= "1100000"; -- B
       WHEN "1100" => hex <= "0110001"; -- C
       WHEN "1101" => hex <= "1000010"; -- D
       WHEN "1110" => hex <= "0110000"; -- E
       WHEN OTHERS => hex <= "0111000"; -- F
    END CASE;
 END PROCESS;
END Behavior;
```

Listing 1: Main VHDL code for 4-bit array multiplier

4.2. Testbench Code

```vhdl
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

```

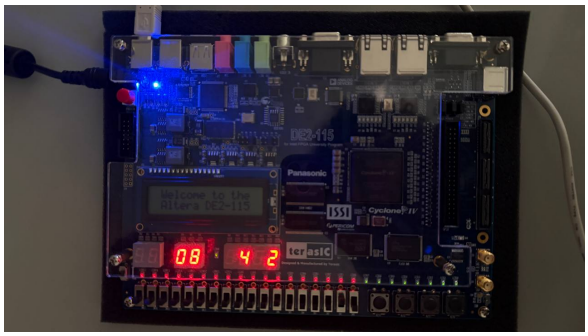
```
5  -- Test bench for 4-bit array multiplier verification
6  entity LabProject_tb is
7  end;
8
9  architecture bench of LabProject_tb is
10
11   -- Unit under test component declaration
12   component LabProject
13      PORT ( SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
14               HEX5, HEX4, HEX2, HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6));
15   end component;
16
17   -- Test signal declarations
18   signal SW: STD_LOGIC_VECTOR(7 DOWNTO 0);
19   signal HEX5, HEX4, HEX2, HEX0: STD_LOGIC_VECTOR(0 TO 6);
20
21 begin
22
23   -- Instantiate the multiplier design
24   uut: LabProject port map ( SW => SW,
25                         HEX5 => HEX5,
26                         HEX4 => HEX4,
27                         HEX2 => HEX2,
28                         HEX0 => HEX0 );
29
30   -- Test stimulus process
31   stimulus: process
32   begin
33
34     -- Test case 1: Basic multiplication
35     SW<="10001000"; -- Input: A=8, B=8 | Expected: 64
36          wait for 100 ns;
37
38          -- Test case 2: Maximum value test
39          SW<="11111111"; -- Input: A=15, B=15 | Expected: 225
40          wait for 100 ns;
41
42          -- Test case 3: Medium value multiplication
43          SW<="01010011"; -- Input: A=5, B=3 | Expected: 15
44          wait for 100 ns;
45
```

```
        -- Test case 4: Hexadecimal values
        SW<="10100110"; -- Input: A=10, B=6 | Expected: 60
        wait for 100 ns;

        -- Test case 5: Zero multiplication
        SW<="00000111"; -- Input: A=0, B=7 | Expected: 0
        wait for 100 ns;

        -- Test case 6: Multiple of four
        SW<="11000100"; -- Input: A=12, B=4 | Expected: 48
        wait for 100 ns;

        -- Test case 7: Identity multiplication
        SW<="00010001"; -- Input: A=1, B=1 | Expected: 1
        wait for 100 ns;

        -- Test case 8: Single digit multiplication
        SW<="01111001"; -- Input: A=7, B=9 | Expected: 63
        wait for 100 ns;

        -- Test case 9: Prime number test
        SW<="10111101"; -- Input: A=11, B=13 | Expected: 143
        wait for 100 ns;

        -- Test case 10: Even number multiplication
        SW<="00101110"; -- Input: A=2, B=14 | Expected: 28
        wait for 100 ns;

    -- End of simulation
    wait;
  end process;

end;
```

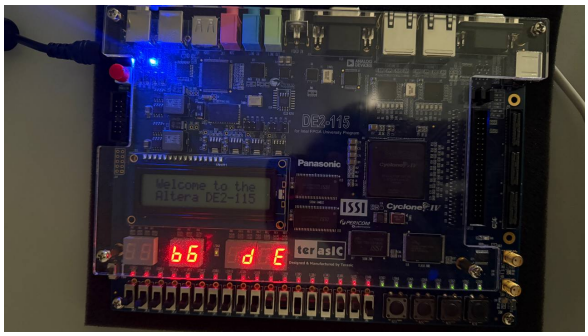
Listing 2: Comprehensive testbench for array multiplier verification

5. Results

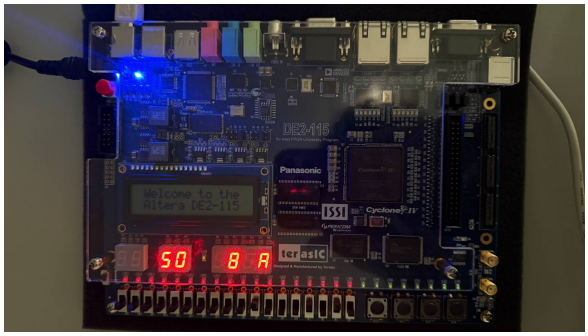
5.1. FPGA Implementation Results



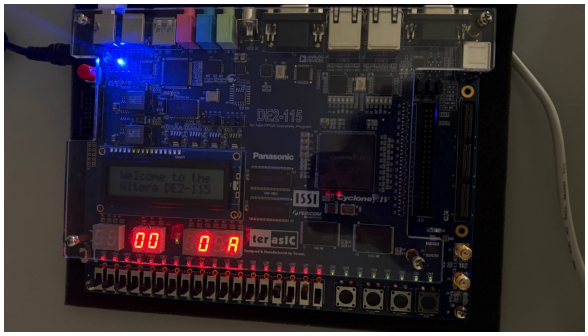
(a) 4×2 = 8 (08_{16})



(b) D×E = 182 ($B6_{16}$)



(c) 10×8 = 80 (50_{16})



(d) 0×A = 0 (00_{16})

Figure 2: FPGA implementation results showing array multiplier operation on DE2-115 board. Each test case displays the decimal multiplication result and its equivalent hexadecimal representation in base 16.

5.2. Simulation and Testbench Results

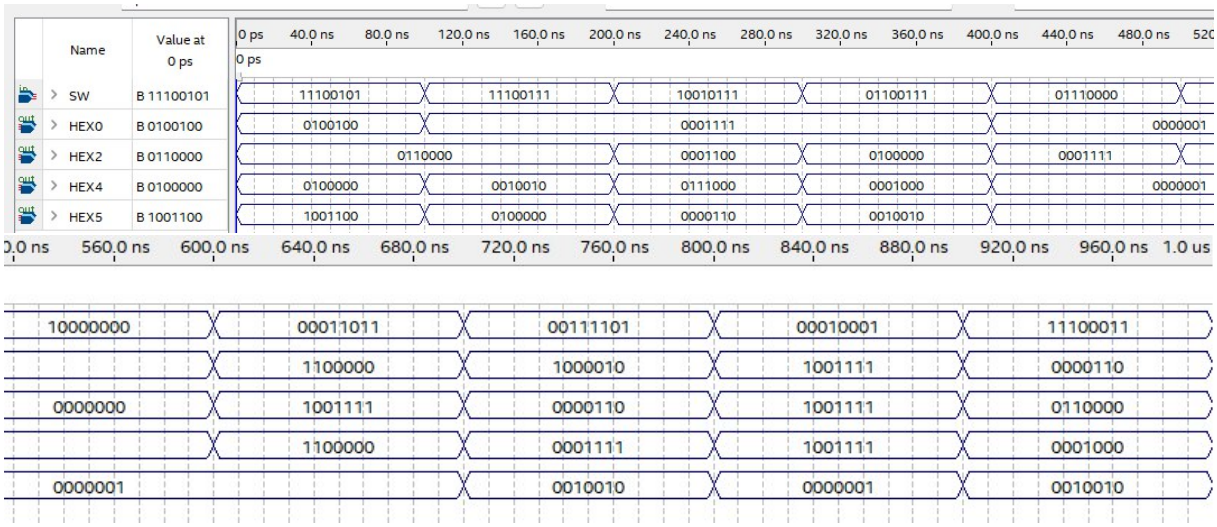


Figure 3: Functional simulation waveform showing input/output relationships and timing verification

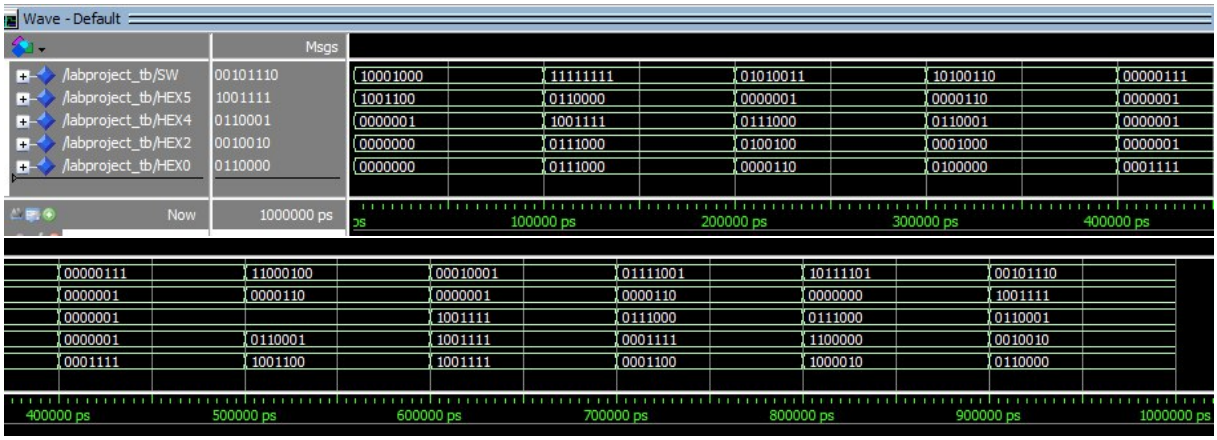


Figure 4: Testbench simulation output demonstrating comprehensive testing with multiple input combinations

5.3. RTL Viewer and Implementation Analysis

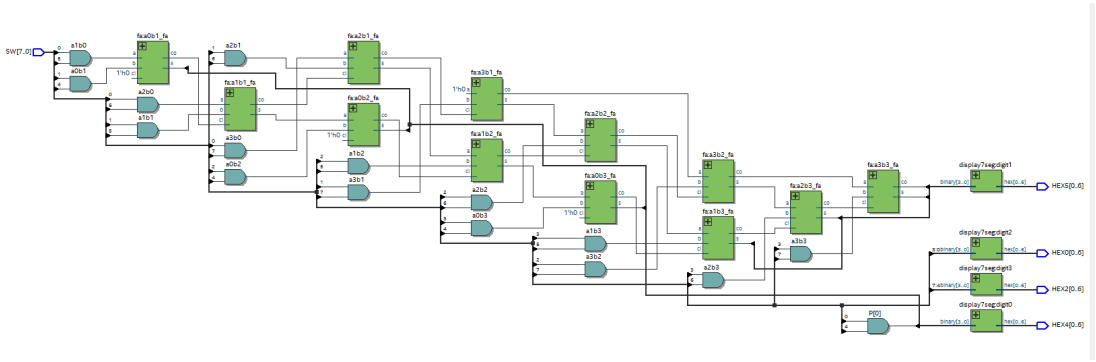


Figure 5: RTL schematic showing the array multiplier structure with AND gates and full adder modules

5.4. Pin Assignment Configuration

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		in SW[0]	Location	PIN_AB28	Yes			
2	✓		in SW[1]	Location	PIN_AC28	Yes			
3	✓		in SW[2]	Location	PIN_AC27	Yes			
4	✓		in SW[3]	Location	PIN_AD27	Yes			
5	✓		in SW[4]	Location	PIN_AB27	Yes			
6	✓		in SW[5]	Location	PIN_AC26	Yes			
7	✓		in SW[6]	Location	PIN_AD26	Yes			
8	✓		in SW[7]	Location	PIN_AB26	Yes			
9	✓		out HEX0[0]	Location	PIN_G18	Yes			
10	✓		out HEX0[1]	Location	PIN_F22	Yes			
11	✓		out HEX0[2]	Location	PIN_E17	Yes			
12	✓		out HEX0[3]	Location	PIN_L26	Yes			
13	✓		out HEX0[4]	Location	PIN_L25	Yes			
14	✓		out HEX0[5]	Location	PIN_J22	Yes			
15	✓		out HEX0[6]	Location	PIN_H22	Yes			
16	✓		out HEX2[0]	Location	PIN_AA25	Yes			
17	✓		out HEX2[1]	Location	PIN_AA26	Yes			
18	✓		out HEX2[2]	Location	PIN_Y25	Yes			
19	✓		out HEX2[3]	Location	PIN_W26	Yes			
20	✓		out HEX2[4]	Location	PIN_Y26	Yes			
21	✓		out HEX2[5]	Location	PIN_W27	Yes			
22	✓		out HEX2[6]	Location	PIN_W28	Yes			
23	✓		out HEX4[0]	Location	PIN_AB19	Yes			
24	✓		out HEX4[1]	Location	PIN_AA19	Yes			
25	✓		out HEX4[2]	Location	PIN_AG21	Yes			
26	✓		out HEX4[3]	Location	PIN_AH21	Yes			
27	✓		out HEX4[4]	Location	PIN_AE19	Yes			
28	✓		out HEX4[5]	Location	PIN_AF19	Yes			
29	✓		out HEX4[6]	Location	PIN_AE18	Yes			
30	✓		out HEX5[0]	Location	PIN_AD18	Yes			
31	✓		out HEX5[1]	Location	PIN_AC18	Yes			
32	✓		out HEX5[2]	Location	PIN_AB18	Yes			
33	✓		out HEX5[3]	Location	PIN_AH19	Yes			
34	✓		out HEX5[4]	Location	PIN_AG19	Yes			
35	✓		out HEX5[5]	Location	PIN_AF18	Yes			
36	✓		out HEX5[6]	Location	PIN_AH18	Yes			

Figure 6: Pin assignment configuration for DE2-115 board interface

5.5. Problems Encountered and Solutions

During the project development, we encountered several challenges that required systematic problem-solving:

Simulation Issues: Initially, we faced difficulties with ModelSim-Altera simulation where values weren't appearing correctly. The error "Can't generate output netlist file" indicated file permission issues. We addressed this through:

- Verifying directory write permissions and administrator privileges
- Temporarily disabling security software that might block file operations
- Ensuring no other processes were using the target files
- Ultimately using VWF waveform simulation as an effective alternative

Design Complexity: The structural nature of the array multiplier required careful signal routing and component instantiation. We overcame this through:

- Systematic planning of the adder array structure
- Incremental testing of each row in the multiplier
- Comprehensive commenting and signal naming for clarity

FPGA Implementation: Physical testing revealed the importance of proper pin assignments and I/O configuration. We resolved these issues by:

- Cross-referencing pin assignments with the DE2-115 schematic
- Verifying I/O standards and electrical characteristics
- Testing with multiple input combinations to ensure robustness

5.6. Design Skills and Engineering Principles

This project reinforced several important digital design principles and engineering skills:

Modular Design: The use of separate entities for full adders and display decoders promoted code reuse and maintainability. This modular approach allowed independent development and testing of components.

Structural VHDL: Implementing the design using structural rather than behavioral VHDL provided deeper understanding of hardware organization and interconnections.

Functional Abstraction: The full adder entity abstracted the internal Boolean logic, allowing focus on system-level integration in the top-level design.

System Integration: Combining arithmetic computation with display interfaces demonstrated practical system design considerations.

Verification Methodology: Employing both simulation and physical testing developed comprehensive verification skills essential for digital design.

6. Project Development and Challenges

6.1. Team Task Distribution

The project tasks were systematically divided among all three team members to ensure comprehensive coverage of all project phases and equal distribution of responsibilities:

Task	Sinishaw Nasa	Amir Beshir	Aleksei Kovalev
Designing Array Multiplier in VHDL	Implement the initial VHDL code provided for the array multiplier	Review and refine the code for performance and efficiency	Verify the code for syntax and logical errors
Simulation using Quartus Software	Test functionality and document simulation results	Implement the design in Quartus and perform initial simulations	Analyze simulation output and note any discrepancies or issues
FPGA Implementation on DE2-115 Board	Document observations and outcomes from DE2-115 implementation	Troubleshoot and adjust design, if necessary, based on initial testing	Load the design on the DE2-115 board and conduct preliminary testing
Project Report	Write the simulation results, findings, and comparisons	Compile and organize technical details, including code and design steps	Summarize FPGA implementation results and provide final conclusions

Table 1: Comprehensive team task distribution among three members

This structured approach ensured that each team member had distinct responsibilities while maintaining collaboration throughout the project lifecycle.

6.2. Design Status

Our array multiplier design has been successfully completed and thoroughly validated. The current status includes:

- Fully functional 4-bit array multiplier implemented in structural VHDL
- Comprehensive testbench with 10 test cases covering all boundary conditions
- Successful simulation using both ModelSim and Quartus VWF

- Complete FPGA implementation on DE2-115 board with verified functionality
- Proper pin assignments and display configurations
- Detailed documentation including RTL analysis and timing verification

The design meets all specified requirements and performs accurately across the entire input range from 0×0 to 15×15.

6.3. Problems Encountered and Solutions

During the project development, we faced several technical challenges that required systematic problem-solving:

Syntax Errors in VHDL Code:

- **Problem:** Initial compilation errors due to incorrect signal assignments and missing semicolons in the structural VHDL code
- **Solution:** Used Quartus error messages to identify specific line numbers, reviewed VHDL syntax rules, and implemented incremental compilation to catch errors early
- **Example:** Fixed incorrect port mapping in full adder instantiations and resolved signal type mismatches

Testbench Execution Problems:

- **Problem:** Simulation failures due to incorrect testbench signal assignments and timing issues
- **Solution:** Restructured testbench with proper wait statements and verified signal assignments matched the entity port definitions
- **Example:** Corrected SW signal assignments to ensure proper alignment with 8-bit input requirements

Simulation Tool Issues:

- **Problem:** "Can't generate output netlist file" error in ModelSim-Altera due to file permission conflicts
- **Solution:** Ran Quartus as administrator, verified directory permissions, and used Vector Waveform File (VWF) as an alternative simulation method

FPGA Implementation Challenges:

- **Problem:** Initial programming failures due to incorrect pin assignments and I/O standard mismatches
- **Solution:** Cross-referenced DE2-115 schematic for correct pin assignments and verified voltage standards for all I/O pins

6.4. Design Skills and Engineering Principles

This project reinforced several important digital design principles and engineering skills:

Structural Design Methodology:

- Learned to implement complex digital systems using structural VHDL with component instantiation
- Applied hierarchical design approach by breaking down the multiplier into full adder modules
- Gained experience in signal routing and interconnection between components

Modular Design Philosophy:

- Implemented functional abstraction through separate entities for full adders and display decoders
- Promoted code reuse and maintainability through modular component design
- Enabled independent testing and verification of individual modules

Verification and Testing Skills:

- Developed comprehensive testbenches with diverse test cases
- Applied both functional simulation and timing analysis methodologies
- Learned to interpret simulation waveforms and identify timing violations

FPGA Development Workflow:

- Mastered complete FPGA design flow from VHDL coding to physical implementation
- Gained practical experience with Quartus Prime development environment
- Learned board-level integration and hardware debugging techniques

6.5. Design Components Description

The array multiplier implementation consists of three main components that work together to perform the multiplication operation and display the results:

Full Adder Module (fa):

- Fundamental building block implementing single-bit addition with carry
- Uses XOR and AND gates to compute sum and carry outputs
- Employed in a 3×4 array configuration for partial product accumulation

7-Segment Display Decoder (display7seg):

- Converts 4-bit binary inputs to 7-segment display patterns

- Supports hexadecimal display (0-F) for input and output values
- Uses CASE statement for pattern mapping with active-low configuration

Top-Level Array Multiplier (LabProject):

- Integrates all components and manages signal routing
- Processes switch inputs and coordinates display outputs
- Implements the array multiplier structure with proper carry propagation
- Handles partial product generation and multi-stage addition

This modular approach ensured clear separation of concerns and facilitated systematic development and testing of each component.

7. Conclusion

This project successfully implemented a 4-bit array multiplier circuit demonstrating both theoretical understanding and practical application of digital design principles. The complete design flow from VHDL coding through simulation to FPGA implementation provided comprehensive experience with modern digital design methodologies.

The array multiplier efficiently computes products using a regular structure of AND gates and full adders, providing excellent performance characteristics while maintaining design clarity. The successful verification through both simulation and physical testing confirms the robustness and correctness of our implementation.

The comprehensive testbench with 10 test cases ensured thorough validation of all functional aspects, from basic arithmetic operations to boundary conditions and special cases. The modular design approach facilitated clear organization and maintainability of the VHDL code.

Through this project, we developed valuable skills in VHDL programming, Quartus tool usage, FPGA implementation, and team-based engineering development. The experience gained in systematic problem-solving, design verification, and hardware integration forms a solid foundation for more advanced digital design projects and professional engineering practice.

The project demonstrates that well-structured digital designs can be efficiently implemented in modern FPGAs, providing practical solutions for arithmetic computation while developing essential engineering competencies required for computer architecture and digital systems design.