

Spartan6 - DSP48A1

Under the supervision of

Eng. Kareem Wasseem

Eng. Ziad Kassem

Developer: Ramy Salah

August 2025

Table of Contents

| | | |
|-------|--|----|
| 0.1 | Abstract | 2 |
| 0.2 | introduction | 3 |
| 0.2.1 | Background on FPGA Technology | 3 |
| 0.2.2 | Overview of Xilinx Spartan-6 FPGA | 3 |
| 0.2.3 | The DSP48A1 Slice: Architecture and Applications | 3 |
| 0.2.4 | Project Objectives | 4 |
| 0.3 | Project Specifications | 5 |
| 0.3.1 | Functional Requirements | 5 |
| 0.3.2 | Hardware Specifications | 9 |
| 0.4 | Design Methodology | 11 |
| 0.4.1 | System Architecture | 11 |
| 0.4.2 | Verilog HDL Implementation | 12 |
| 0.5 | Testbench Development | 14 |
| 0.5.1 | Reset Verification | 14 |
| 0.5.2 | Directed Functional Verification (4 data paths) | 14 |
| 0.6 | Simulation Waveforms | 17 |
| 0.7 | Synthesis and Implementation | 19 |
| 0.7.1 | Overview | 19 |
| 0.7.2 | Elaboration and Synthesis | 19 |
| 0.7.3 | Implementation | 21 |
| 0.8 | Conclusion | 23 |
| 0.9 | References | 23 |

0.1 Abstract

This project presents the design and implementation of a high-speed arithmetic unit using the DSP48A1 block on the Xilinx Spartan-6 FPGA. The design is described in Verilog HDL and verified through simulation using a dedicated testbench. The synthesis process is carried out to map the design onto the FPGA hardware, leveraging the dedicated DSP resources to achieve optimal performance. The project also provides a detailed analysis of resource utilization, timing constraints, and functional verification. Results show that using the DSP48A1 block significantly improves computational efficiency compared to implementations relying solely on general-purpose logic.

0.2 introduction

0.2.1 Background on FPGA Technology

Field Programmable Gate Arrays (FPGAs) are reconfigurable integrated circuits that allow designers to implement custom digital logic without the need for fabricating a new chip. They are widely used in applications that require high-speed processing, parallelism, and hardware flexibility, such as signal processing, telecommunications, and embedded systems.

0.2.2 Overview of Xilinx Spartan-6 FPGA

The Xilinx Spartan-6 family is a cost-effective and power-efficient FPGA series designed for high-performance applications. It provides a wide range of resources including Configurable Logic Blocks (CLBs), Block RAM, and dedicated DSP slices. These devices are optimized for applications like wireless communications, video processing, and automotive systems.

0.2.3 The DSP48A1 Slice: Architecture and Applications

The DSP48A1 slice is a dedicated hardware block inside the Spartan-6 FPGA optimized for digital signal processing tasks. It can perform multiply, add, and accumulate operations in a single clock cycle. This enables the implementation of high-speed arithmetic operations without heavily relying on general-purpose logic resources. Applications include filtering, FFT, matrix operations, and other computationally intensive tasks.

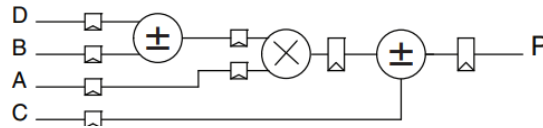


Figure 1: Simplified Version of the DSP48A1 Slice

0.2.4 Project Objectives

The main goal of this project is to design and implement a high-speed arithmetic unit using the DSP48A1 block on the Xilinx Spartan-6 FPGA. The design is described in Verilog HDL, simulated using a dedicated testbench, and synthesized for implementation on real hardware. Performance metrics such as resource utilization, maximum frequency, and timing slack are analyzed.

0.3 Project Specifications

0.3.1 Functional Requirements

DSP48A1 Slice Interface

The DSP48A1 slice is the primary computational block used in this design. Figure 2 shows the pin-level interface of the slice, including its data ports (A, B, C, D), control signals (OPMODE, clock, resets, enables), and outputs (P, M, carry signals). Understanding this interface is essential to correctly map the required operations to the hardware.

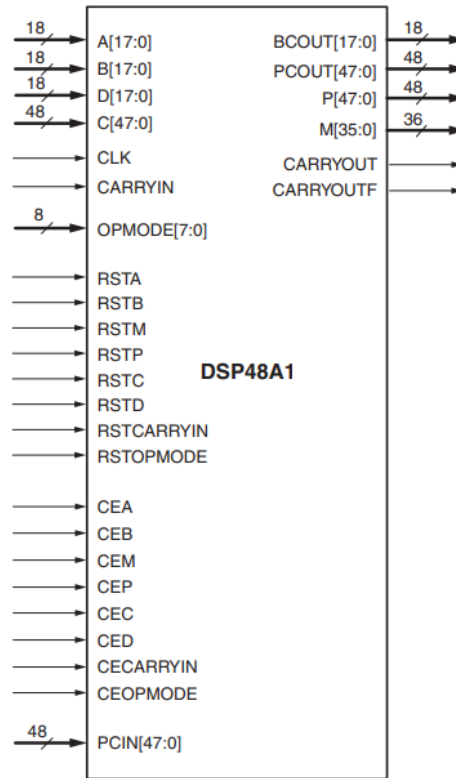


Figure 2: Pin-level interface of the DSP48A1 slice, showing data, control, and output signals.

System Block Diagram

The system block diagram in Figure 3 illustrates how the DSP48A1 slice is integrated into the FPGA design. It highlights the interconnection between

the DSP slice, input logic modules, and output logic. This diagram defines the overall signal flow and ensures that the functional requirements are met.

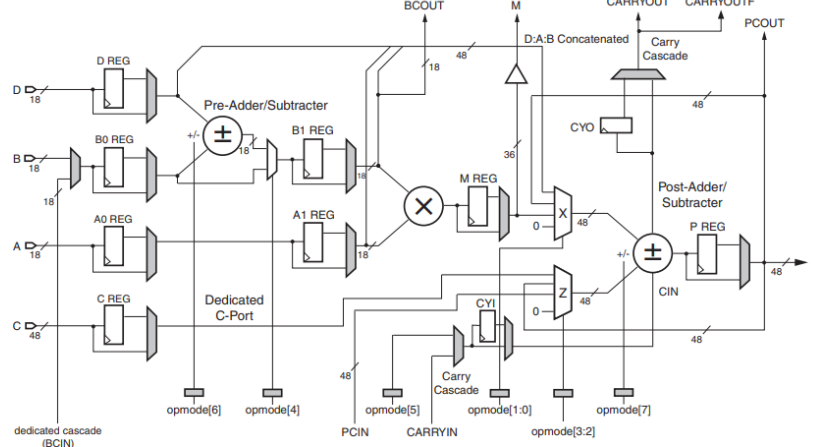


Figure 3: High-level block diagram of the FPGA system using the DSP48A1 slice.

Input Logic for DSP48A1 Operations

The DSP48A1 slice requires specific input configurations for each operation mode. The following subsections describe the main input ports and their roles.

A Input The **A[17:0]** port provides the first operand for the DSP48A1 slice. It is typically used as the multiplicand in multiplication operations or as one of the operands in addition/subtraction.

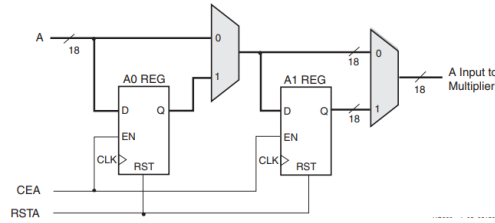
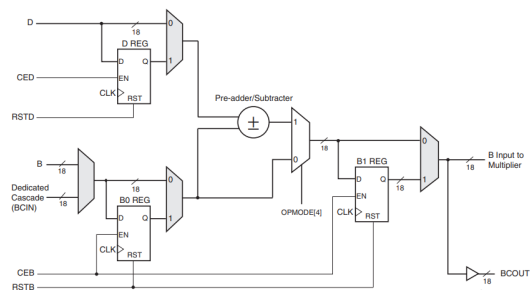
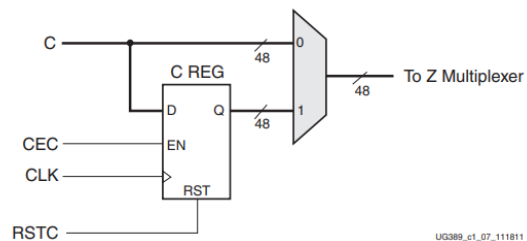


Figure 4: Logic configuration and bit-width details for the A input.

B Input The **B[17:0]** port acts as the second operand, most often serving as the multiplier in arithmetic operations.



C Input The **C[47:0]** port provides the third operand, mainly used for accumulation in multiply-accumulate (MAC) operations.



OPMODE Input The **OPMODE[7:0]** control bus defines the operational mode of the DSP48A1 slice. Each bit pattern corresponds to a specific function such as multiplication, addition, subtraction, or logical operations.

Figure 7: Control signal configuration and mapping of OPMODE bits to operations.

Summary Table of Inputs

Table 1 summarizes the bit-width and main function of each primary input port.

| Signal | Bit-width | Function |
|--------|-----------|---|
| A | 18 bits | First operand (multiplicand or add/sub operand) |
| B | 18 bits | Second operand (multiplier or add/sub operand) |
| C | 48 bits | Accumulator operand for MAC operations |
| D | 18 bits | Pre-adder operand for $(A \pm D) \times B$ |
| OPMODE | 8 bits | Operation mode selection |

Table 1: Summary of DSP48A1 primary input ports.

0.3.2 Hardware Specifications

FPGA Device Information

The project was implemented on a Xilinx Spartan-6 XC6SLX45 FPGA, which offers a balance of logic resources, DSP slices, and block RAM suitable for DSP applications.

Table 2: XC6SLX45 FPGA Device Specifications

| Parameter | Value |
|-------------------------|--------------|
| Logic Cells | 43,661 |
| Slices | 6,822 |
| Look-Up Tables (LUTs) | 27,288 |
| Flip-Flops | 54,576 |
| Block RAM (BRAM) | 2,088 Kb |
| DSP48A1 Slices | 58 |
| Maximum User I/O Pins | 218 |
| Maximum Clock Frequency | 200 MHz |

DSP48A1 Slice Specifications

The DSP48A1 is a dedicated slice optimized for high-performance arithmetic operations.

Table 3: DSP48A1 Slice Key Specifications

| Feature | Specification |
|-----------------------------|---|
| Multiplier Width | 18 x 18 bits |
| Pre-adder Width | 25 bits |
| Accumulator Width | 48 bits |
| Supported Operations | Multiply, Add, Subtract, MAC, Logic Ops |
| Pipelining Stages | Up to 3 |
| Maximum Operating Frequency | 250 MHz (speed grade -3) |

Performance and Timing

The maximum performance achievable depends on the selected speed grade and pipeline configuration.

Table 4: Maximum Clock Frequency for DSP48A1 (Typical)

| Operation Mode | Max Frequency (-3 Grade) |
|---------------------|--------------------------|
| Multiply Only | 250 MHz |
| Multiply-Accumulate | 230 MHz |
| Add/Subtract Only | 270 MHz |
| Logic Operations | 270 MHz |

0.4 Design Methodology

0.4.1 System Architecture

The project design was structured into multiple functional modules to ensure clarity, ease of debugging, and modular verification. Each module was responsible for a specific part of the DSP48A1 operation, as listed below:

- **bmux** – Implements the B input selection logic.
- **carrycascade** – Handles the carry propagation between arithmetic stages.
- **multiplier** – Performs the core multiplication operation.
- **mux_reg_sync** – Implements a synchronous register-based multiplexer.
- **mux_reg_async** – Implements an asynchronous register-based multiplexer.
- **mux_reg** – Generic multiplexer with register output.
- **mux4x1** – A 4-to-1 multiplexer for signal selection.
- **postadder** – Executes the final addition/subtraction stage after multiplication.
- **preadder** – Performs pre-addition/subtraction before multiplication.

All modules were integrated into the top-level design module, **maindsp**, using a structural modeling approach to clearly define interconnections between submodules.

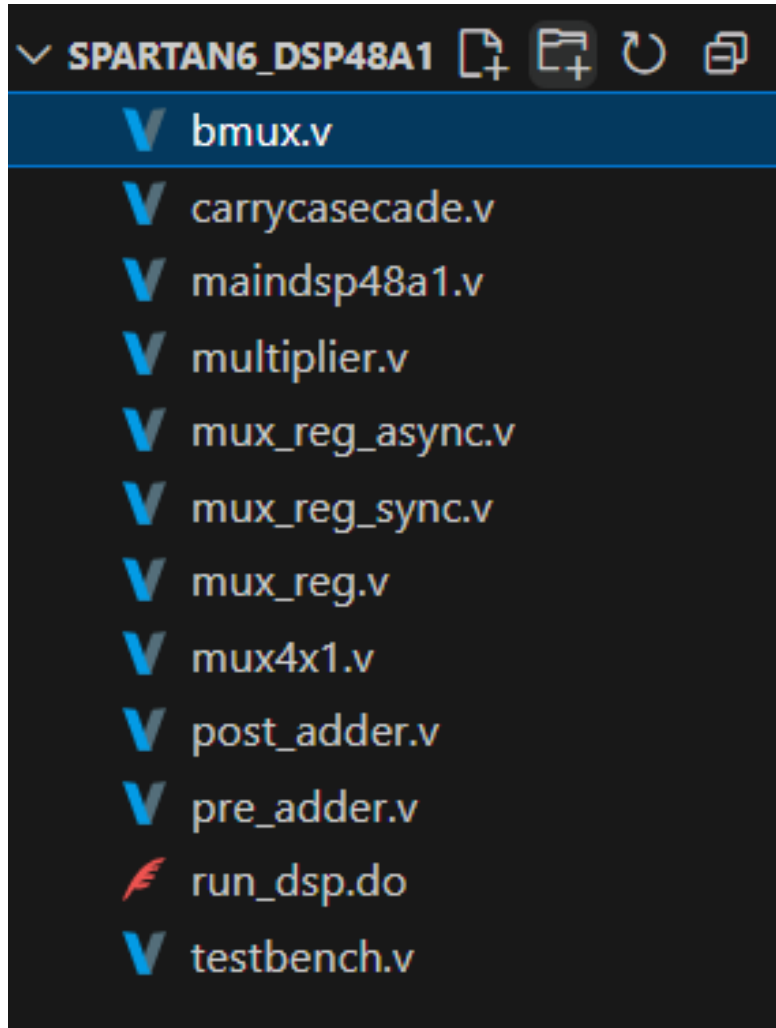


Figure 8: System modules and their interconnections in the DSP design.


0.4.2 Verilog HDL Implementation

Different design methodologies were applied to optimize each module:

- Some modules were implemented using a **behavioral** modeling style (e.g., *multiplier*, *preadder*, *postadder*) for faster development and easier modification.
- Others were implemented using **dataflow** modeling (e.g., *bmux*, *carrycasecade*) for clear representation of signal transformations.

- The top-level **maindsp** module was written using a **structural** modeling approach, explicitly instantiating and connecting the submodules.

A simulation automation script (**run.do** file) was also developed to streamline the compilation and waveform generation process, reducing repetitive manual commands.

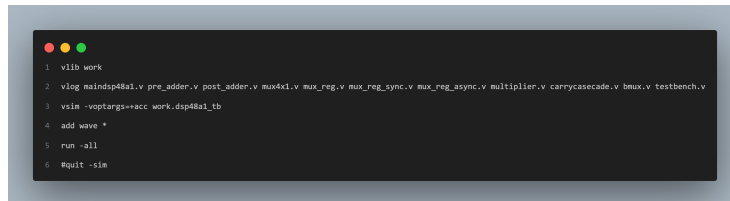


```

1  /******
2  //from Beginning till Multiplication
3  /******
4
5  mux_reg_operation #( .width(width3), .REG_STATE(OPMODEREG), .RSTTYPE(RSTTYPE))
6  opmode(.clk(clk), .cei(CEOPMODE), .rst(RSTOPMODE), .in(OPMODE), .out(W_OPMODE));
7
8  bmux #( .B_INPUT(B_INPUT), .width(width1))
9  bin0(.in0(B), .in1(BCIN), .out(B0_wire));
10
11 mux_reg_operation #( .width(width1), .REG_STATE(B0REG), .RSTTYPE(RSTTYPE))
12 b0reg(.clk(clk), .cei(CEB), .rst(RSTB), .in(B0_wire), .out(B0_wire2));
13
14 mux_reg_operation #( .width(width1), .REG_STATE(DREG), .RSTTYPE(RSTTYPE))
15 d0reg(.clk(clk), .cei(CED), .rst(RSTD), .in(D), .out(D_wire));
16
17 pre_adder #( .width(width1))
18 preadding(.in1(D_wire), .in2(B0_wire2), .out(pre_addout), .opmodeadd(W_OPMODE[6]), .opsmodesel(W_OPMODE[4]));
19
20 mux_reg_operation #( .width(width1), .REG_STATE(B1REG), .RSTTYPE(RSTTYPE))
21 b1reg(.clk(clk), .cei(CEB), .rst(RSTB), .in(pre_addout), .out(pre_addoutreg));
22
23 mux_reg_operation #( .width(width1), .REG_STATE(ABREG), .RSTTYPE(RSTTYPE))
24 ABreg(.clk(clk), .cei(CEA), .rst(RSTA), .in(A), .out(A0_wire));
25

```

Figure 9: Part of the top-level **maindsp** Verilog code showing module instantiations.



```

1  vlib work
2  vlog maindsp48a1.v pre_adder.v post_adder.v mux4x1.v mux_reg.v mux_reg_sync.v mux_reg_async.v multiplier.v carrycascade.v bmux.v testbench.v
3  vsim -voptargs+acc work.dsp48a1_tb
4  add wave *
5  run -all
6  #quit -sim

```

Figure 10: Excerpt from the **run.do** simulation automation script.

0.5 Testbench Development

The design verification strategy uses a **directed testbench** approach to exercise the top-level module (`maindsp`) under specific, well-defined scenarios. The testbench focuses on two main tasks:

1. **Reset verification:** ensure all registers and state machines initialize correctly.
2. **Functional verification of four data paths:** exercise the four distinct data-flow paths implemented in the design and verify correct outputs and internal handshaking/signalling.

0.5.1 Reset Verification

A dedicated reset test is applied at simulation start to confirm that all internal registers, control signals, and accumulator values are set to known states after reset deassertion. Place the waveform snapshot for the reset check below.

0.5.2 Directed Functional Verification (4 data paths)

The testbench includes four directed stimuli sequences. Each sequence targets one of the possible data paths in the design (for example: direct multiply, pre-adder multiply, accumulate with C, cascade path). Below are four placeholders where you should insert the verification screenshots for each path.

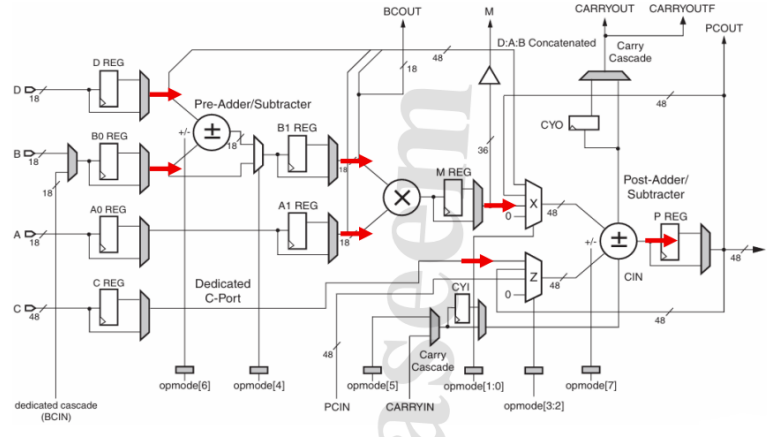


Figure 11: Verification snapshot — Data Path 1 (e.g., multiply only).

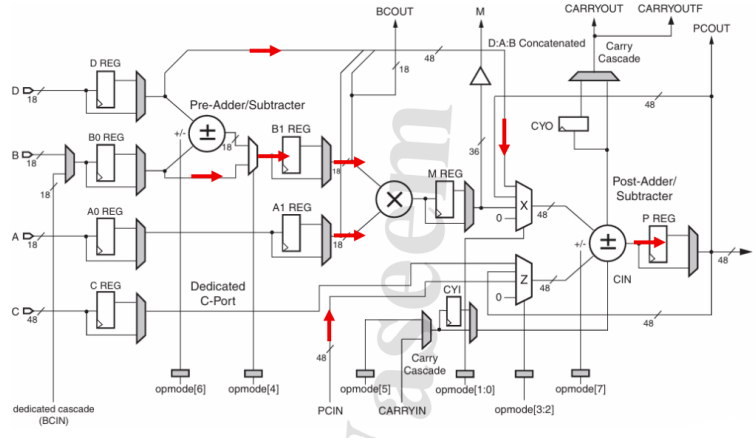


Figure 14: Verification snapshot — Data Path 4 (e.g., cascade / carry path).

0.6 Simulation Waveforms

This section presents selected waveform snippets captured during simulation. Each snapshot highlights key functional behavior and timing relationships (you should replace the placeholders below with actual waveform images).

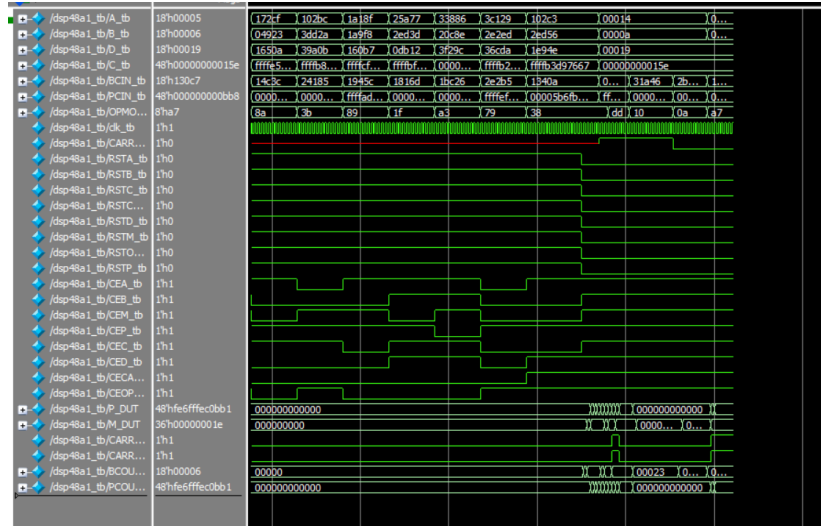


Figure 15: Representing a capture from the waveform.

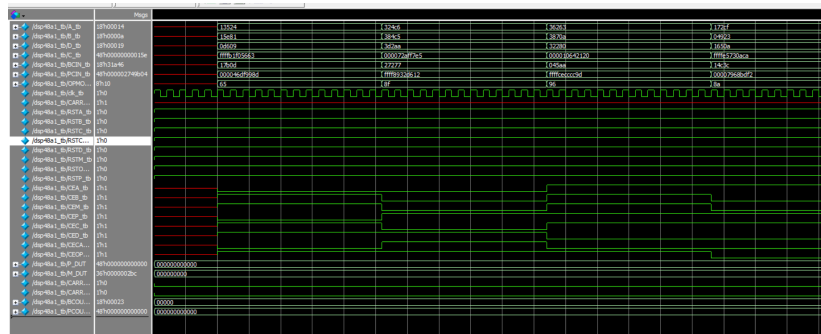


Figure 16: Representing a capture from the waveform.

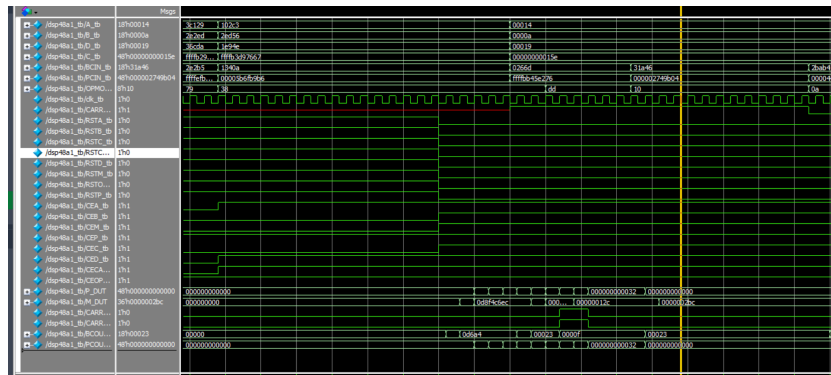


Figure 17: Representing a capture from the waveform.

0.7 Synthesis and Implementation

0.7.1 Overview

Prior to synthesis, the RTL was checked with a static linting tool (e.g., Questa/Questa Lint) to ensure coding style and inference compatibility with the target FPGA primitives. After RTL validation, synthesis and implementation were performed using the Xilinx toolflow. Although Spartan-6 devices are typically targeted with Xilinx ISE/XST, the project flow below uses the Vivado/Xilinx tools for synthesis and implementation (adjust tool names to match your setup).

0.7.2 Elaboration and Synthesis

The source files were elaborated and synthesized to validate that the design can be mapped to the target device and to obtain resource utilization estimates. Below place:

- an elaborated schematic (block-level netlist) snapshot,
- timing report snippet (post-synthesis),
- resource utilization table (post-synthesis).

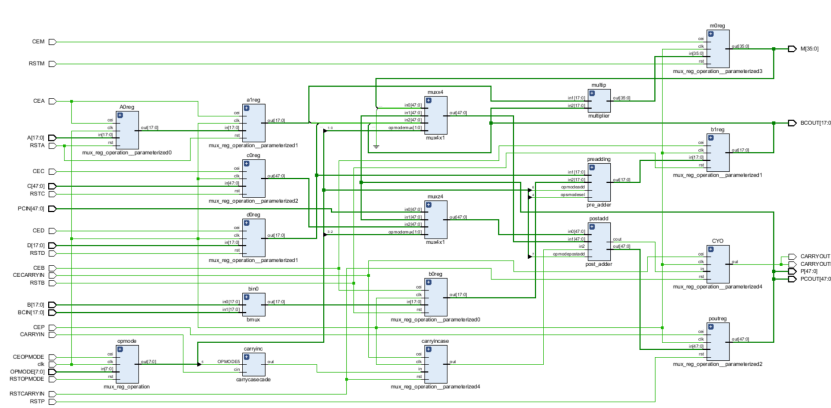


Figure 18: Elaboration schematic (post-elaboration netlist view).

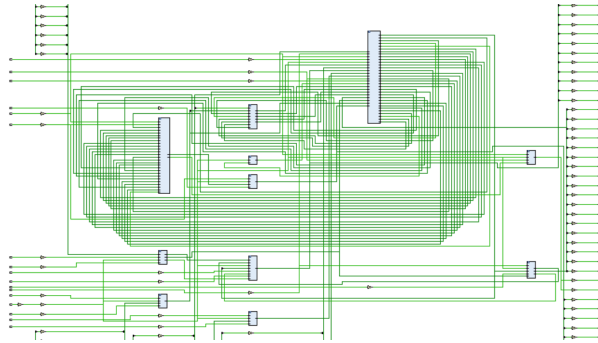


Figure 19: Synth schematic

| Design Timing Summary | | | | | |
|--|--|----------------------------------|--|---|--|
| Setup | | Hold | | Pulse Width | |
| Worst Negative Slack (WNS): 2.639 ns | | Worst Hold Slack (WHS): 0.225 ns | | Worst Pulse Width Slack (WPWS): 4.500 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: 103 | | Total Number of Endpoints: 103 | | Total Number of Endpoints: 162 | |
| All user specified timing constraints are met. | | | | | |

Figure 20: Synthesis timing report excerpt (critical path and Fmax).

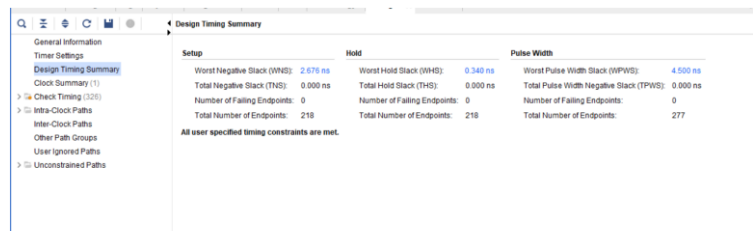
| Hierarchy | | | | | |
|---------------------------|-----------------------|----------------------------|------------------|---------------------|------------------|
| Name | Slice LUTs (20800) | Slice Registers (41600) | DSP s (90) | Bonded IOB (106) | BUFGCTRL (32) |
| maindsp48a1 | 336 | 160 | 1 | 327 | 1 |
| a1reg (mux_reg_opera...) | 19 | 18 | 0 | 0 | 0 |
| b1reg (mux_reg_opera...) | 19 | 18 | 0 | 0 | 0 |
| c0reg (mux_reg_opera...) | 25 | 48 | 0 | 0 | 0 |
| carryincase (mux_reg_...) | 0 | 1 | 0 | 0 | 0 |
| CYO (mux_reg_operati...) | 1 | 1 | 0 | 0 | 0 |
| d0reg (mux_reg_opera...) | 10 | 18 | 0 | 0 | 0 |
| m0reg (mux_reg_oper...) | 1 | 0 | 1 | 0 | 0 |
| opmode (mux_reg_op...) | 234 | 8 | 0 | 0 | 0 |
| postadd (post_adder) | 25 | 0 | 0 | 0 | 0 |
| poutreg (mux_reg_ope...) | 1 | 48 | 0 | 0 | 0 |
| preadding (pre_adder) | 1 | 0 | 0 | 0 | 0 |

Figure 21: Synthesis resource utilization (LUTs, FFs, BRAM, DSP slices).

0.7.3 Implementation

The implementation stage (place & route) produced final timing and utilization reports and a device-level view of placed logic on the FPGA. Place the following artifacts here:

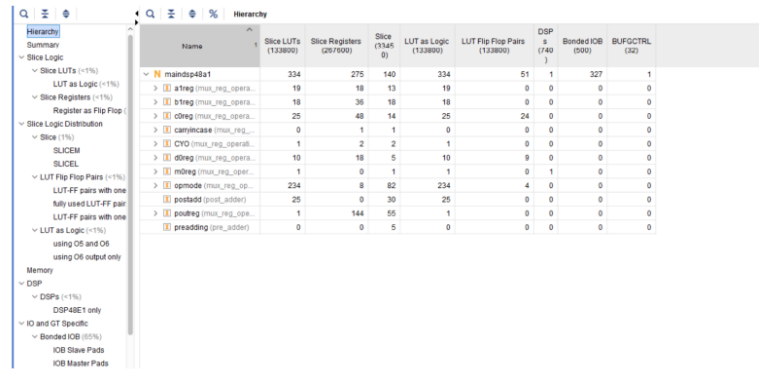
- post-implementation timing report (STA)
- post-implementation utilization summary
- device view (placed DSPs / highlighted P/CARRY chains)



| Setup | Hold | Pulse Width |
|--------------------------------------|----------------------------------|---|
| Worst Negative Slack (WNS): 2.675 ns | Worst Hold Slack (WHS): 0.340 ns | Worst Pulse Width Slack (WPWS): 4.500 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: 218 | Total Number of Endpoints: 218 | Total Number of Endpoints: 277 |

All user specified timing constraints are met.

Figure 22: Implementation timing report (post-place & route).



| Name | Slice LUTs (123800) | Slice Registers (267600) | Slice (X3A5) (0) | LUT as Logic (123800) | LUT Flip Flop Pairs (123800) | DSP (740) | Bonded IOB (500) | BUFGCTRL (32) |
|-------------------------|------------------------|-----------------------------|------------------------|--------------------------|---------------------------------|--------------|---------------------|------------------|
| maindsp48a1 | 334 | 275 | 140 | 334 | 51 | 1 | 327 | 1 |
| alreg (mux_reg_opera... | 19 | 18 | 13 | 19 | 0 | 0 | 0 | 0 |
| brreg (mux_reg_opera... | 18 | 36 | 18 | 18 | 0 | 0 | 0 | 0 |
| clreg (mux_reg_opera... | 25 | 48 | 14 | 25 | 24 | 0 | 0 | 0 |
| carriacase (mux_reg... | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| cdv (mux_reg_opera... | 1 | 2 | 2 | 1 | 0 | 0 | 0 | 0 |
| clreg (mux_reg_opera... | 10 | 18 | 5 | 10 | 0 | 0 | 0 | 0 |
| mlreg (mux_reg_opera... | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| opmode (mux_reg_op... | 234 | 8 | 82 | 234 | 4 | 0 | 0 | 0 |
| postadd (post_addr) | 25 | 0 | 30 | 25 | 0 | 0 | 0 | 0 |
| postreg (mux_reg_op... | 1 | 144 | 55 | 1 | 0 | 0 | 0 | 0 |
| preadding (pre_addr) | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 |

Figure 23: Implementation resource utilization (post-place & route).

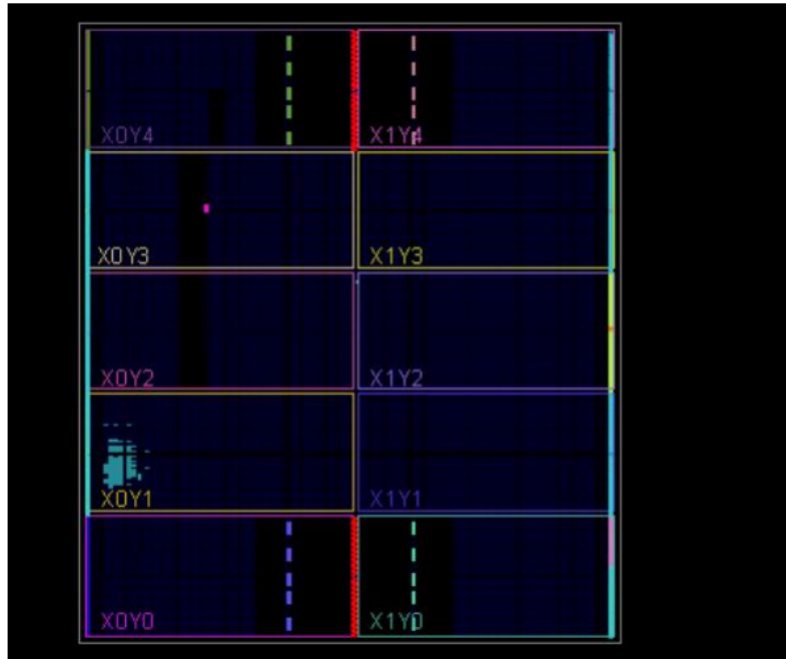


Figure 24: Device view on the target FPGA showing placed DSP48A1 tiles and the implemented design.

0.8 Conclusion

This project demonstrates a modular hardware implementation of high-speed arithmetic functions using the DSP48A1 slice within a Spartan-6 FPGA. By partitioning the design into well-defined modules (preadder, multiplier, postadder, routing/mux blocks, and top-level structural integration), the design achieves clear separation of concerns, easier verification, and good synthesis results. Directed verification confirmed the correct operation of the reset sequence and all four primary data paths. Post-synthesis and implementation reports indicate that the design is synthesizable and fits on the selected target device while leaving room for further optimizations.

Future Work:

- Add parameterized fixed-point support and automated test vectors to cover edge cases.
- Explore pipelining options to increase FMAX and throughput.
- Implement a small driver to interface the design with an external SoC or microcontroller for in-system testing.

0.9 References

1. Xilinx (AMD) — *Spartan-6 FPGA DSP48A1 Slice User Guide (UG389)*. Available from Xilinx/AMD documentation portal.