Design, Verification, and Optimization of a Taylor Series-Based Polynomial Evaluator on Arria 10 FPGA

Authors:
Kunal Malik
Nathan Lind

A Comprehensive Documentation on the
Design, Verification, and Optimization
of a Taylor Series-Based Polynomial Evaluator
Implemented on an Arria 10 FPGA

Arizona State University
SCAI
February 9, 2024

Abstract

This project explores three different methodologies for implementing a Taylor series-based polynomial evaluator: base methodology, pipeline methodology, and shared methodology. Each methodology is evaluated based on timing, resource usage, and power dissipation to understand its performance characteristics. The base methodology represents a straightforward implementation without optimizations, while the pipeline methodology aims to improve throughput through parallel processing. The shared methodology focuses on resource sharing to reduce overall resource utilization. By comparing these methodologies, we identify trade-offs between latency, throughput, power, and area efficiency. This analysis helps in selecting the most suitable design methodology for specific application requirements.

Contents

List of Figures

List of Tables

Introduction

1.1 Background

Polynomial Evaluations play a crucial part in numerous computational and engineering applications, enabling approximation of complex functions via simple polynomial expressions. The method described in the document is instrumental in the scientific computing , digital signal processing, and control systems fields where accuracy and efficiency of numerical computations are paramount[1].

Among all the available methods, Taylor series method is more versatile and gives more mathematical robustness. It approximates a function around a specific point through an infinite series of scaled derivatives, offering a means to achieve arbitrary accuracy, given sufficient computational resources [2].

While implementing polynomial evaluators on hardware platforms one should take into consideration regarding computational speed, resource efficiency and power consumption. With the FPGAs reconfigurable nature and parallel processing capabilities, we have a viable soultion for such computational acceleration. Since FPGAs can be configured for specific computations they outperform general purpose processors in terms of speed and power efficiencyfor specialised applications[3] which is the reason why FPGAs are used for specific computational intensive task in a SOC.

Intel's Arria 10 FPGA stands out by combining high efficiency , adaptability and performance. This device is optimized for wide range of applications ranging from data processing to embedded systems, making it an excellent candidate for deploying sophesticated computational models like the Taylor series Evaluator.

This project seeks to exploit the Arria 10 FPGA's capabilities to develop a Taylor series based Polynomial Evaluator focusing on Design, Verification and Optimization. By comparing Base , pipelined and shared resource implementations, the study aims to identify most efficient approach for FPGA based function approximation contributing valuable insights into both theoretical and practical aspects of hardware acceleration computation.

1.2 Problem Statement

The mathematical function evaluation should be accurate and efficient which is critical requirement in various scientific and engineering applications. Traditional software based function evaluation methods, while versatile often cannot meet the precise performance, latency and power consumption requirements of high speed and embedded computing applications. This challenge is particularly sound in real time processing tasks and in systems where thermal and power constraints limit the use of high performance general purpose processors.

Polynomial approximation methods, such as those based on the Taylor series, offer high speed evaluation of functions by replacing complex computations with simpler polynomial expressions. However direct implementation of such methods on hardware poses several challenges:

- **Resource Efficiency:** Polynomial evaluators, particularly those which are based on higher-order Taylor series expansion, can be resource-intensive, requiring significant computational units for multiplication and addition. The available logic blocks, DSP blocks, and memory are finite and must be judiciously allocated.
- **Latency and Throughput:** For real-time applications, achieving high throughput with low latency is necessary. The inherent serial nature of polynomial computation, especially in higher-order Taylor series, can introduce significant delays.
- **Power Consumption:** High-speed computation often leads to increased power consumption, challenging the deployment of polynomial evaluators in power-sensitive applications.
- **Accuracy and Precision:** Maintaining computational accuracy while minimizing resource utilization requires a delicate balance. The fixed-point arithmetic implementation in FPGAs can introduce errors, which need to be properly handled.

Base Implementation of the Taylor Series Polynomial Evaluator

2.1 Introduction

We will start with base implementation of the Taylor Series-Based Polynomial Evaluator which is foundational in designing other implementations. In this chapter we will delve into base implementation where the focus will be on realizing a straightforward yet effective polynomial on the Arria 10 FPGA platform. This implementation will be a proof of concept demonstrating the viability if FPGA based function approximation using Taylor series.

Our journey through this base implementation is not just about laying down the groundwork but also an exploration of inherent challenges and limitations which comes with hardware based function evaluation. By understanding these foundational aspects, we set the groundwork for subsequent chapters, where we will explore more advanced techniques such as pipelining and resource sharing to enhance performance and efficiency.

As we navigate through this chapter, it's crucial to keep in mind that this base implementation is the cornerstone of our project. It embodies the initial step in our quest to harness the power of FPGAs for polynomial evaluation.

2.2 Design

We will be designing for first five terms of Taylor Series. This design can be extended to more number of terms which will also increase Power consumption, and Resource Utilization. The Taylor series can be given by:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots \tag{2.1}$$

We will use A0, A1, A2, ... for constants.

For our base design the constant will be a 16 bit number. Our base design will consist of series of cascade multiplier and adder to perform this calculation. The coefficients will be carefully encoded in Q2.14 format, laying groundwork for polynomial approximation. A set of registers will be used to capture the input and output ensuring synchronisation of data flow with the clock.

2.2.1 Circuit Diagram

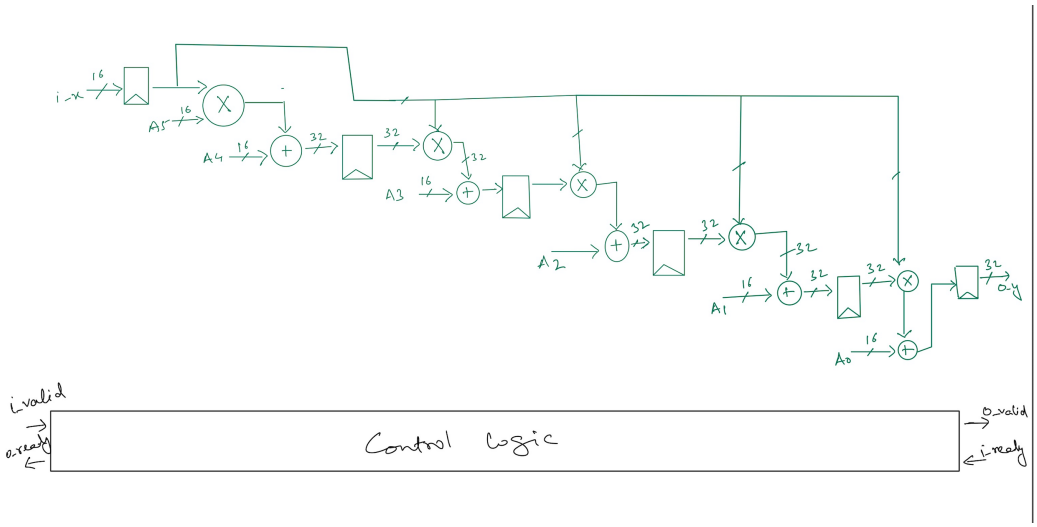

Figure 2.1: High Level Circuit Diagram of Base Implementation.

Using 2.1 a circuit diagram is made. As illustrated in Figure 2.1, the initial stage of our computation involves a 16-bit input and a 16-bit constant value being fed into the first multiplier module. This necessitates the use of a 16 x 16 multiplier to accommodate the bit-width of both inputs. The output from this operation yields a 32-bit result, setting the precedent for subsequent operations.

Following this initial multiplication, the 32-bit result is then directed to an adder module. This adder must also accommodate an additional 16-bit input x x, necessitating the design of a specialized 32p16 adder module. This module is adept at handling one 32-bit and one 16-bit input, producing a cohesive 32-bit output, thereby maintaining consistency in data width throughout the process.

For subsequent multiplication stages, the scenario evolves as we engage with the 32-bit outputs from previous additions. These outputs are then multiplied by a 16-bit input, necessitating the transition to a 32 x 16 multiplier module. This module is specifically tailored to manage the multiplication of a 32-bit operand with a 16-bit operand, ensuring that the integrity and precision of our calculations are upheld throughout the computational sequence. This meticulous approach to module selection and design underpins the robustness and efficiency of our computational architecture.

2.2.2 Verilog Code

We will begin by defining key parameters that set the stage for precision and scalability in our design. The input and output widths, **WIDTHIN** and **WIDTHOUT**, are carefully chosen to balance computational accuracy with resource efficiency. The input is set to a 16-bit width, following the Q2.14 format, which provides sufficient precision for the initial values while conserving FPGA resources. The output, however, extends to 32 bits in the Q7.25 format, accommodating the increased precision required for the accumulated results of the Taylor series expansion. The heart of the module consist of Taylor coefficients, A0 to A5, each encoded in the Q2.14 format. These constants represent the distilled essence of the Taylor series, encapsulated within the binary confines of FPGA logic. They serve as the building blocks for the polynomial approximation, guiding the computation through each successive term of the series. As discussed in previous section, our computational journey starts with mult16x16 module instantiation following a addr32p16 instantiation, following mult32x32 module instantiaiton. The result of every computational module is fed to other module in series according to circuit diagram. We are using handshaking signals for our module to effectively communicate with the external environment. We are using four signals *i_valid*, *i_ready*, *o_valid*, *o_ready*. The function of the control signals are as follows:

- **i_valid:** This signals is driven by our external environment (testbench), it is for communicating with our design to tell it that the input provided by the external environment is a valid input.
- **i_ready:** This signal is also an input to out DUT and hence driven by external environment use to communicate the readniness of our external environment for taking an output from out DUT.
- **o_ready:** This signal is output from our DUT to the external environmnet telling the readiness of out DUT for taking a new input value from the testbench.
- **o_valid:** This signal serves as to tell the testbench that the output generated by the DUT is a valid output and hence this is also driven by our DUT.

For full verilog code please see 6.

2.3 Verification

For verification we prepared a testbench to drive our DUT module. The detailes about the testbench will be discussed in the next section. Since we are using the same testbench for all our designs this will be discussed only once in the next section and will be skipped for all other sections.

2.3.1 Test Bench Setup

Our testbench begins with generation of a 42 MHz clock which oscillates with a period defined by CLK_PERIOD, which is parameter defined in the beginning. After the clock is generated, we are instantiating our DUT in our testbench. We are defining a behavioural function to calculate exponential function directly which will be used to compare our results in future. There is one more function to calculate the error between the expected value and actual value we got from our DUT.

We are using 32 bit 'SEED' to generate random values , SEED can be changed to a different number to get different random values.

Our testbench consist of two processes: i) Producer Process ii) Consumer process The behaviour of these processes are as follows:

- **Producer Process:** Our Producer process begins with initial block where we start out testing process by setting i_valid as low and applying reset for 1 clock period and then releasing the reset.

  We will be generating 50 random inputs for testing purposes. The input value i_x is given after advancing to quarter cycle. Then we advance to quarter cycle two times and we start other check (check for o_ready). Our DUT should not respond to inputs if it claims not to be ready. the to test that is we check for o_ready and apply an input which should get ignored by our DUT.

  Then comes our third check, (check that DUT stalls properly when a valid input is not given to our DUT). This is done for number of testcases / 2 times to ensure our DUT functions properly under these conditions after which advance to next posedge cycle.

- **Consumer Process:** This will test how the DUT responds when giving output to external environment. The test starts with generating same numbers as in Producer Process.

here we check if o_valid is true i.e. output is valid and if that is the case we chaeck the output with our calculated value from the function defined and check the error tolerence. If our value is in the range of 0.045 our test case will pass.

Then we continue our test with checking that our DUT stalls the output properly if our external environment is not ready to take the input from our DUT. This marks the completions of our testing process.

The code for testbench can be found in 6

2.3.2 Results

In the rigorous evaluation of our design within the Questa Sim environment, the meticulous execution of our testbench has yielded a resounding success, as evidenced by the comprehensive passing of all test cases. This accomplishment is not merely a testament to the robustness of our design but also underscores the precision with which our Taylor Series-Based Polynomial Evaluator operates under simulated conditions that mirror real-world complexities. Further affirmation of our design's integrity is encapsulated in the waveform analysis, depicted in 2.2. The waveform, a graphical representation of the dynamic interactions between inputs and outputs over time, offers an insightful glimpse into the operational nuances of our module. It illustrates not only the temporal alignment of handshaking signals—ensuring synchronized data exchange—but also the precise moments of data validity, marked by the o_valid signal, against the backdrop of our meticulously generated clock cycles.

In the rigorous evaluation of our design within the Questa Sim environment, the meticulous execution of our testbench has yielded a resounding success, as evidenced by the comprehensive passing of all test cases. This accomplishment is not merely a testament to the robustness of our design but also underscores the precision with which our Taylor Series-Based Polynomial Evaluator operates under simulated conditions that mirror real-world complexities.

Further affirmation of our design's integrity is encapsulated in the waveform analysis, depicted in Figure 2.2. The waveform, a graphical representation of the dynamic interactions between inputs and outputs over time, offers an insightful glimpse into the operational nuances of our module. It illustrates not only the temporal alignment of handshaking signals—ensuring synchronized data exchange—but also the precise moments of data validity, marked by the `o_valid` signal, against the backdrop of our meticulously generated clock cycles.

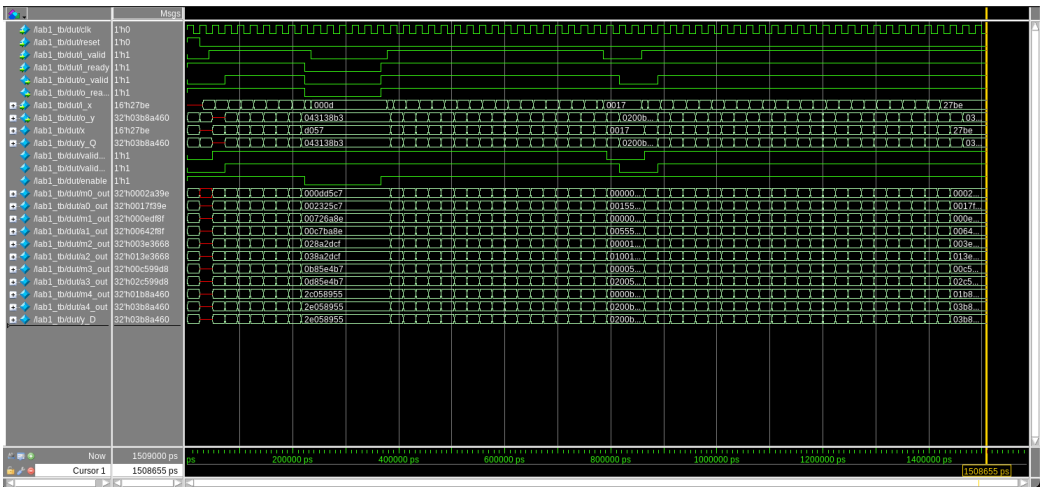

Figure 2.2: Waveform for Base Implementation.

2.4 Compilation and Deployment

The successful compilation of our design on the Arria 10 FPGA using Quartus Prime stands as a significant milestone, achieved without encountering any errors. This accomplishment not only validates the syntactical integrity of our design but also attests to its compatibility with the advanced hardware specifications of the Arria 10 platform. In the forthcoming sections, we will delve into a comprehensive discussion of the outcomes, drawing insights from various reports generated during the compilation process. These discussions aim to illuminate the performance characteristics, resource utilization, and efficiency of our design, providing a holistic understanding of its implications in the realm of FPGA-based computational solutions.

2.4.1 Operating conditions

Table 2.1: Operating Conditions Used

Setting	**Value**
Device power characteristics	Typical
Voltages	
VCCP	0.90 V
VCCPT	1.80 V
VCCA_PLL	1.80 V
VCC	0.90 V
VCCPGM	1.80 V
VCCBAT	1.80 V
VCCERAM	0.90 V
1.8 V I/O Standard	1.8 V
Auto computed junction temperature	54.5 °C
Ambient temperature	50.0 °C
Junction-to-Case thermal resistance	0.10 °C/W
Case-to-Heat Sink thermal resistance	0.10 °C/W
Heat Sink-to-Ambient thermal resistance	1.30 °C/W
Board model used	None

2.4.2 Results

Timing Results

We compiled our design with a clock period of 1ns (1Ghz) at the begining which gave us a significant negative slack of 19.956. After many iterations of changing clock period we got a positive slack of 0.17 with a clock period of 21.4135ns corrosponding to a frequency of 46.7 MHz on worst case operating condition which is Slow 900mV -40C Model. 2.1 describes the operating conditions which are used during compilation.

for our critical path we can see it starts from

2.2 shows our path summary for worst case slack. It can be seen that Data Arrival time is 25.209ns whereas Data Require time is 25.379ns giving us a positive slack of 0.170ns. For our critical path we have the critical path which passes

through the ALMs which implement the first multiplier and adder (Mult0 and Addr0)

- followed by another multiplier (Mult1) split accross two DSP blocks (indicated by locations MPDSP_X100_Y93 and MPDSP_X100_Y94)
- followed by an adder (Addr1) created from ALMs i.e. LUTs and carry chains
- followed by another multiplier (Mult2) split accross two DSP blocks
- followed by another multiplier (Mult3) split accross two DSP blocks
- followed by another multiplier (Mult4) split accross two DSP blocks
- followed by another adder (Addr4) created from logic elements
- and ending in the y_Q[31] register.

More on critical path can be found in worst-case-timing-path report in 6.

Table 2.2: Path Summary

Property	**Value**
From Node	x[6]~_Duplicate6
To Node	y_Q[31]
Launch Clock	clk
Latch Clock	clk
SDC Exception	No SDC Exception on Path
Data Arrival Time	25.209
Data Required Time	25.379
Slack	0.170
Worst-Case Operating Conditions	Slow 900mV -40C Model

Resource Utilisation

The Resource Utilization reports offer a granular review of efficient resource utilization. Our Arria 10 FPGA consist of a total ALMs of 427,200 out of which ALMs needed are 63 out of which 27 ALMs are due to virtual I/Os which we

will not take into account which left us with a toatl ALMs of (63-27) = 36 ALMs which is 1% of total resources. From the Resource Utilization summary 6.

For DSP blocks we have in our Arria 10 FPGA a total of 1518 out of which we are using 6 DSP (which were 8 before dense packing) blocks for our design. The ability to recover DSP Blocks through dense merging, as indicated in the table, exemplifies the FPGA's adeptness at resource optimization, ensuring that each block is utilized to its fullest potential.

More on Resource utilization can be found in 6.

Power Analyzer

The "On-Chip Power Dissipation by Block Type" 6 and "Power Analyzer Summary"2.3 reports offer a clear lens into how energy is expended across different components.

The elements like DSP blocks and combinational cells show relatively modest power usage, at 0.80 mW and 0.15 mW respectively with a total on chip power dissipation of around 3W.

Table 2.3: Power Analyzer Summary

Parameter	**Value**
Power Analyzer Status	Successful - Fri Jan 26 23:31:30 2024
Quartus Prime Version	23.4.0 Build 79 11/22/2023 SC Pro Edition
Revision Name	lab1
Top-level Entity Name	lab1
Family	Arria 10
Device	10AX115N2F45I1SG
Timing Models	Final
Power Models	Final
Device Status	Final
Total On-Chip Power Dissipation	2995.94 mW
Transceiver Standby On-Chip Power Dissipation	0.00 mW
Transceiver Dynamic On-Chip Power Dissipation	0.00 mW
I/O Standby On-Chip Power Dissipation	0.06 mW
I/O Dynamic On-Chip Power Dissipation	0.13 mW
Core Dynamic On-Chip Power Dissipation	3.50 mW
HPS Standby On-Chip Power Dissipation	0.00 mW
HPS Dynamic On-Chip Power Dissipation	0.00 mW
Device Static On-Chip Power Dissipation	2992.24 mW
High Bandwidth Memory Standby On-Chip Power Dissipation	0.00 mW
High Bandwidth Memory Dynamic On-Chip Power Dissipation	0.00 mW
Analog/Digital Converter Standby On-Chip Power Dissipation	0.00 mW
Analog/Digital Converter Dynamic On-Chip Power Dissipation	0.00 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Pipelined Implementation of the Taylor Series Polynomial Evaluator

3.1 Design

For our pipelined design we will Add registers at the output of every computational module (Multiplier & Adder) by which when we are done computing multiplication of first term and move to addition in next clock cycle, we can begin computing multiplication of first term for secomd input. 3.1 describes pipelining process for this implementation design in detail.

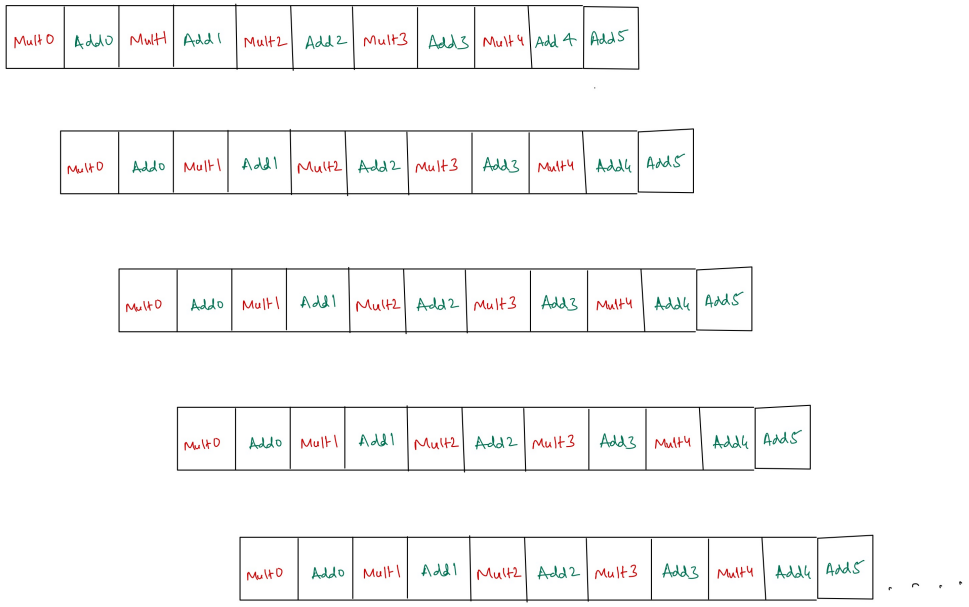

Figure 3.1: Pipeline Working for Pipeline Implementation.

3.1.1 Circuit Diagram

The circuit diagram explains how our actual circuit should look like, extra care should be taken in mind while designing a pipline is that it should be balanced (i.e. while introducing a flop delays, the delays should be equal when more than one input is going in a computational unit). The circuit diagram shows a balnced 10 stage pipeline structure for maximum throughput achievement.

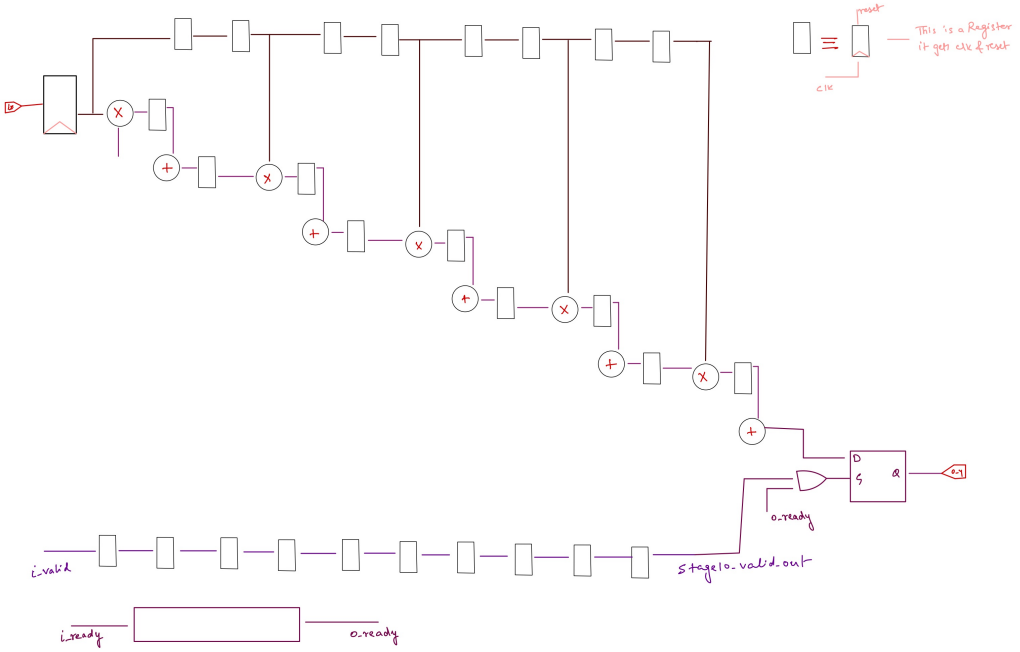

Figure 3.2: Pipeline Circuit Diagram for Pipeline Implementation.

3.1.2 Verilog Code

Our verilog code for pipelining implementation consist of nine different sections out of which seven sections are working part of our code. They are as follows:

- **Signal Declaration:** In this ection we are declaring internal signals which are essential for our design to work. It comparises of additinal signals for input x from pipeline_x1 to pipeline_x2i, pipeline_x2 and so on. There are

various valid signals from stage1_valid_in to stage10_valid_out. there are signals for registering data into different registers in every clock cycle , they are from stage1_data_out to stage10_data_out.

- **Input X Propogation:** In this section we are adding different registers to match the delays to the multiplier form the incoming data signals. Input from our second multiplier has a 2 cycle delay so to match that we are adding 2 register delay from previous multiplier.
- **Module Initializations:** This section consist of essential part of our code where we are instantiating out multiplier and adder module and giving the registered input to every multiplier and adder instantiation.
- **Data Propogation:** The data propogation section consist of addition of registers at the output of every computational module to capture the results used as input to subsequent computational unit in next clokc cycle.
- **Valid Signal Propogation:** Since when propogating data we are checking for valid signal (i.e. input to that computational unit is still valid) therefore we need to propogate the valid signals according to the stages in our pipeline, therefore we are adding registers for capturing valid signals at different stages.
- **Output Management:** The output managemnt unit consist of always block where we are oututting result based on out i_valid and o_ready signals which aligns with our testbench testcases. This ensures a controlled flow of our output.
- **Fixed Point Multiplication & Addition Implementation :** These consist of the foundational units of multiplier and adder which are doing actiual computatins in our design. They are designed carefully to get the output in fixed point arithemetic.

3.2 Verification

3.2.1 Results

The design is verified in questa sim environment, leveraging the same testbench to ensure uniformity in testing parameters. We can see the functional correctness form the waveform 3.3 .The waveform aids in validating the design's compliance

with its intended specifications, ensuring that all functional requirements are met with precision. It highlights the efficiency of data throughput across the pipeline stages, the handling of stall conditions, and the implementation of control logic that governs the pipeline's operation.

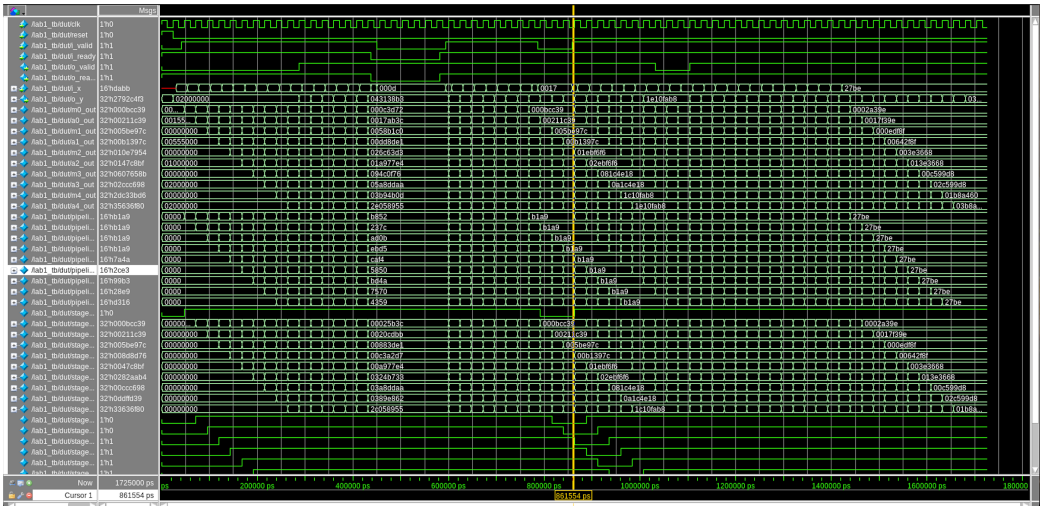

Figure 3.3: Waveform depicting functional correctness for pipeline implementation.

3.3 Compilation and Deployment

The successful compilation of our design on Arria 10 FPGA using Quartus Prime is achieved without encountering any errors. This validates that the design is synthesizable and also attests to its compatibility with the hardware specifications of Arria 10 platform. In the upcoming sections we will discuss about outcomes, drawing insights from various reports generated during compilation process. We will take a deeper look into these report to better understand our design.

3.3.1 Operating Conditions

Table 3.1: Operating Conditions Used

Setting	**Value**
Device power characteristics	Typical
Voltages	
VCCP	0.90 V
VCCPT	1.80 V
VCCA_PLL	1.80 V
VCC	0.90 V
VCCPGM	1.80 V
VCCBAT	1.80 V
VCCERAM	0.90 V
1.8 V I/O Standard	1.8 V
Auto computed junction temperature	54.5 °C
Ambient temperature	50.0 °C
Junction-to-Case thermal resistance	0.10 °C/W
Case-to-Heat Sink thermal resistance	0.10 °C/W
Heat Sink-to-Ambient thermal resistance	1.30 °C/W
Board model used	None

3.3.2 Results

Timing Results

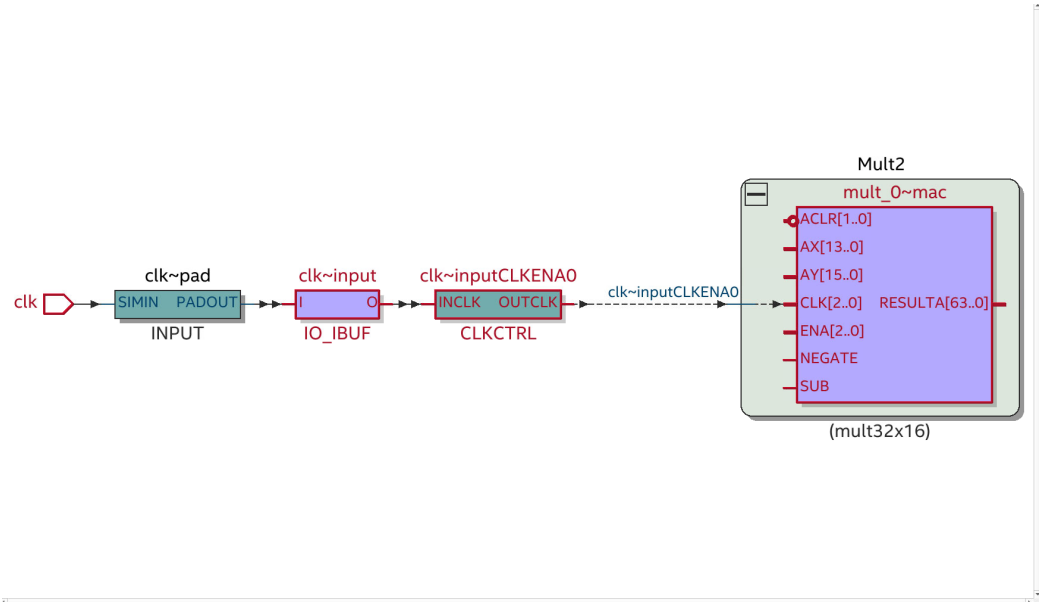

Figure 3.4: Critical path in technology viewer

We have achieved a positive slack of 0.200 ns with a clock time period of 3.053ns corresponding to a frequency of 327.55Mhz which is a significant improvement from our base design which shows how efficient is our pipeline design.

A closer look on worst case path 3.2 shows us the worst case setup path in our design which is **"Mult2—mult_0 mac reg2"**element within the DSP block (MPDSP_X100_Y20_N0), which amounts to about 3.261s increment in our design.

Table 3.2: Data Arrival Path

T (ns)	I (ns)	RF	Ty	F/O	Loc	HS/LP	Elem
0.000	0.000						Launch edge time
0.000	0.000						Clock path
0.000	0.000	R					Clock net delay
0.000	0.000			1	PIN_G38		Clk
0.000	0.000	R		1	PIN_G38		Clk
4.623	4.623						Data path
0.000	0.000	RR	IC	1	IOBUF_X78_Y197_N47		Clk input
0.683	0.683	RR	CELL	1	IOBUF_X78_Y197_N47		Clk output
0.842	0.159	RR	CELL	1	IOBUF_X78_Y197_N47		Clk io_48_lvds
0.842	0.000	RR	IC	2	CLKCTRL_2L_G_I7		Clk CLKENA0
1.362	0.520	RR	CELL	219	CLKCTRL_2L_G_I7		Clk CLKENA0 out
4.623	3.261	RR	IC	16	MPDSP_X100_Y20_N0	HS	Mult2 mac clk[1]
4.623	0.000	RR	CELL	64	MPDSP_X100_Y20_N0	HS	Mult2 mac reg2

More information on timing paths can be found in 6.

The critical path can also be seen in technology map viewer from 3.4

Resource Utilisation

The resource utilization report gives us insights on how efficiently the tool compiled our design. We can see from 6 that a total of 124-27(duo to virtual I/Os) = 97 ALMs are needed for our pipeline design which is greater than base design because pipelined design usually requires more hardware from normal sequential designs. Still it is very low as compared to available ALMs(427, 200) which amounts to still 1%.

Our DSP block utilization is decreased from 6 (in base design) to 4. This time the tool is able to recover 4 DSP blocks via dense packing insted of 2 as in base design.

More on Resource utilization can be found in 6. The 6 shows how resources are distributed by blocks.

Power Analyser

The power Aalyzer results shows us that the core Dynamic On chip power for our pipeline design is 26.31 mW which is lower than the base design(3.50 mW).

The total power for pipeline design is increased a little much amounting to 3022.12 mW

The power by block type shows that DSP blocks now use more power than in base design which is increased from 0.80 mW to 2.14 mW as well as combinational cells from 0.15 to 1.41 mW

The power usage of register cells increased from 0.03 to 1.73mW, which is plausible as pipeline design uses more registers than base design.

Both reports can be accessed from 6.

Shared Resource Implementation of the Taylor Series Polynomial Evaluator

4.1 Design

4.1.1 Block Diagram

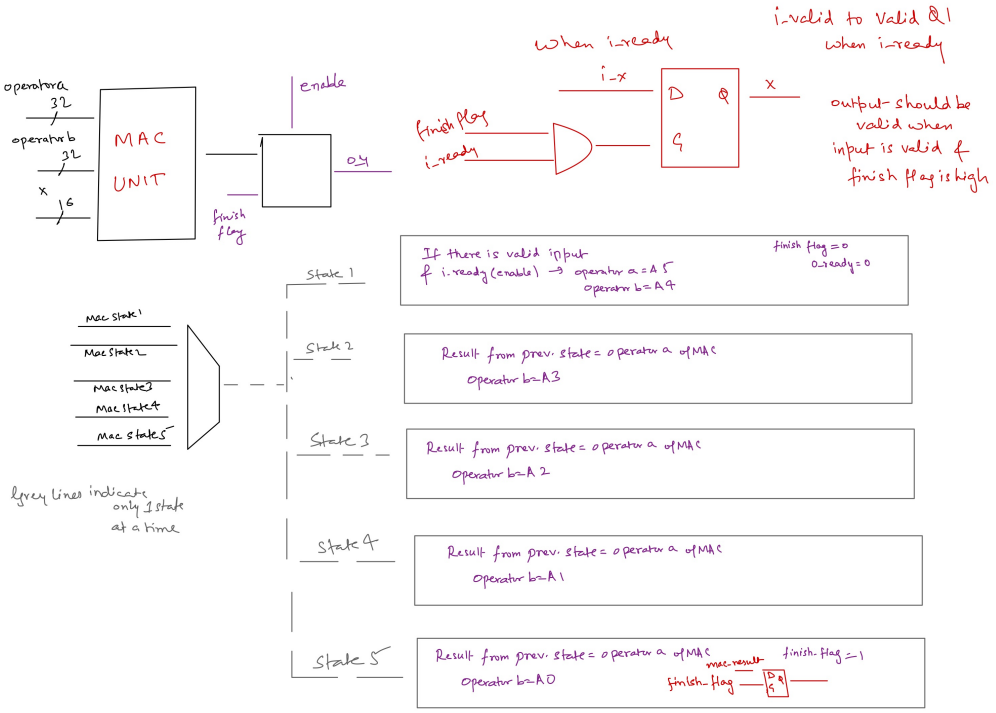

Figure 4.1: High Level Block diagram of design functionality

In the shared resource usage approach for the design, the focus is on maximizing hardware efficiency by minimizing the number of instantiated modules. By adopting this methodology, we will utilize a single multiplier and a single adder module to perform all necessary calculations. This strategy is particularly effective in scenarios where resource constraints are a significant consideration, such

as in FPGA designs where the number of available logic blocks can be a limiting factor.

The shared resource model allows different parts of the design to use the same hardware resources at different times. This is often managed through careful scheduling, ensuring that no two operations that require the same resource are scheduled to occur simultaneously.

We will be using Finite State Machine Logic to efficiently schedule our resources. 4.2 shows a Finite State Machine Diagram which will be used in our design.

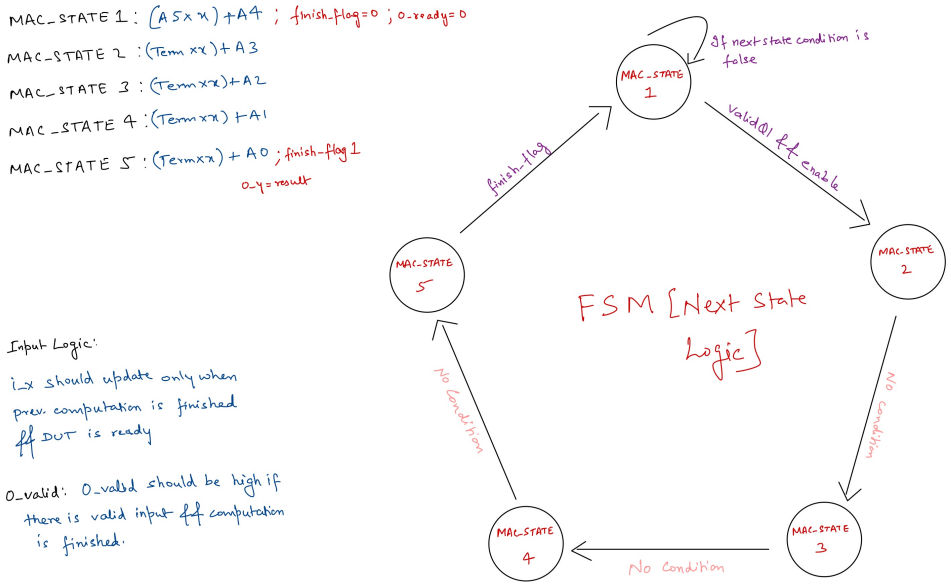

Figure 4.2: Logic For Shared resource Usage Design

4.1.2 Verilog Code

The verilog consist of six working sections:

- **Signal Declaration:** The module starts with internal signal declaration. We

are using parameters to define states and reg to hold current and next states. Also ,we are using wire types to get the output from our Mac unit.

- **MAC Unit Instantiation:** We are instatiating our mac module which will do all the computations
- **State Machine Implementation:** State Machine is implemented to schedule the tasks which out Mac unit will be doing so that the computations happen in a control manner. We are using 5 states for computing each state computes a multiplication followed by addition operation. This can be seen from 4.2
- **Input handling & Synchronization:** In this section we are registering input based on finish flag and o-ready. This is important as this uses handshaking with external environment on when to start the computation.
- **MAC Module Definition:** We are using a MAC module which is instantiating our foundational adder and multiplier modules to do both operation in a single module.
- **Fixed Point Multiplication & Addition Implementation:** In this section we are defining our Multiplier and Adder modules . We are using 32x16 multiplier and 32p32 adder instead of 32p16 adder because we were facing accuracy issues with 32p16 adder which is why we are also using 32 bit constants instead of original 16 bit constants in previous designs.

4.2 Verification

4.2.1 Results

A rigrous verification is done in Questa Sim and wave form is generated to check the correct functionality. Our design passes all the testcases perfectly as can be validated from 4.3 which tells that are testbench requirements are met with precision.

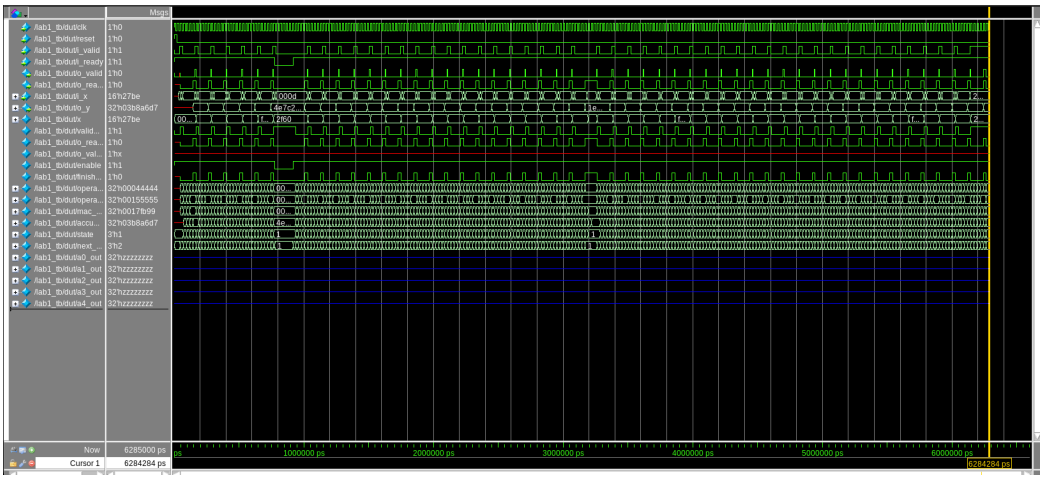

Figure 4.3: Waveform Results for Shared Implementation

4.3 Compilation and Deployment

from any errors. Initially, the design faced some challenges due to multiple assignments of a signal value across different always blocks, leading to the error *"Multiple driving ports to the signal."* This issue was meticulously addressed, ensuring that each signal is driven by a single source, thereby resolving the conflict and allowing the design to compile smoothly. Subsequently, the design was successfully synthesized, generating all essential reports and outputs. Additionally, careful adjustments were made to the clock settings to enhance the timing margin, resulting in a more favorable positive slack. This meticulous approach to resolving synthesis issues and optimizing the clock configuration underscores the robustness and efficiency of the final implementation.

4.3.1 Results

Timing Results

We were able to run the design on a clock frequency of 169.2 MHz with a positive slack of 0.192 ns.

A closer look at the worst case timing path tells us that the worst case path is **mac0—mult_inst—mult_0 DATAOUTA0—clk[0] - mult_0 DATAOUTA0 reg0 - mac0—mult_inst—mult_0 DATAOUTA0—resulta[23] - mac0—mult_inst—mult_0 mac—by[5] - mac0—mult_inst—mult_0 mac—resulta[10] - mac0—add_inst—add_0 57—dataf - mac0—add_inst—add_0 77—cout - mac0—add_inst—add_0 81—cin - mac0—add_inst—add_0 85– - mac0—add_inst—add_0 89—cin - mac0—add_inst—add_0 93—cout - mac0—add_inst—add_0 97– - mac0—add_inst—add_0 101—cout - mac0—add_inst—add_0 105—cin - mac0—add_inst—add_0 1 - mac0—add_inst—add_0 113—cin - mac0—add_inst—add_0 117—sumout**.

The most significant delay occurring in the transition from *mult_0 DATAOUTA0—resulta[23]* to *mac0—mult_inst—mult_0 mac—by[5]*.

We can also see the critical path in the technology map viewer from 4.4

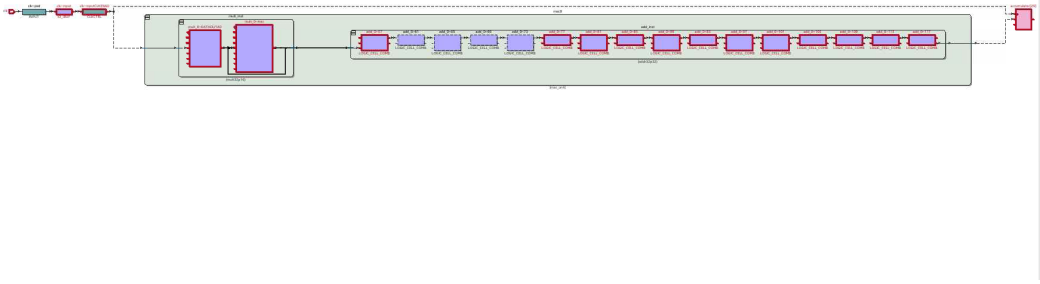

Figure 4.4: Critical Path in technology Viewer

Resource Utilisation

The resource utilization summary of our FPGA design presents a highly efficient use of available resources. With a total of 427,200 Adaptive Logic Modules (ALMs) available on the device, our design utilizes merely 73 ALMs, which is the result of subtracting 27 from 100. This equates to a scant 1% of the total ALMs, underscoring the design's exceptional efficiency and optimization. This level of utilization is indicative of a design that maximizes the potential of the FPGA while minimizing resource consumption, a key attribute in scalable and cost-effective FPGA applications.

Moreover, the design's consumption of DSP blocks stands at just 2, a figure that is notably less than what was observed in both the pipelined and base design iterations. This reduction in DSP block usage not only reflects the design's optimization but also frees up valuable resources for additional functionalities or for scaling the design within the same FPGA. This is reasonable as we are sharing single mutiplier and adder module.

It's important to note that the aforementioned resource metrics pertain to a single instance of the circuit.

Power Analyser

The Power Analyzer report for our FPGA design indicates a core chip dynamic power dissipation of 9.89 mW, with the total on-chip power dissipation reaching 3001.51 mW. This represents a significant increase from the base design's 3.50 mW, suggesting enhancements that have likely contributed to higher dynamic activity within the core logic areas. When compared to the pipelined design, which had a dynamic power dissipation of 26.31 mW, our current design demonstrates a more balanced approach. It implies that while our design has higher power consumption than the base model, it remains substantially more power-efficient than the pipelined version. This balance is indicative of a design that carefully navigates the trade-offs between performance and power consumption, optimizing for efficiency while still maintaining a level of performance that exceeds the base design. Such a strategy is crucial in applications where power efficiency is essential, but performance cannot be compromised, highlighting the nuanced decision-making involved in FPGA design optimization.

For more power dessipiation reports refer to 6

Comparision

5.0.1 Timing

Initially, our design targeted an ambitious clock frequency of 1 GHz, resulting in a substantial negative slack of 19.956 ns, indicating timing violations and a non-viable design under these conditions. Through iterative adjustments to the clock period, we achieved a stable design with a positive slack of 0.17 ns at a reduced clock frequency of 46.7 MHz under the challenging Slow 900mV -40°C Model operating conditions. This frequency represents the design's resilience and adaptability to stringent operating environments, ensuring reliability even in less-than-ideal circumstances.

Remarkably, our pipelined design demonstrated a significant leap in performance, achieving a positive slack of 0.200 ns with a clock period of 3.053 ns, which translates to an operating frequency of 327.55 MHz. This is a substantial improvement over the base design and further optimization allowed the design to operate at 169.2 MHz with a positive slack of 0.192 ns which is reasonable since we are focusing more on area than performance in our shared implementation. These enhancements underscore the efficacy of our pipeline architecture in elevating the design's performance, showcasing its capability to meet higher operational frequencies while maintaining timing integrity, a testament to the design's optimization and efficiency.

5.0.2 Resource utilization

In comparing the resource utilization across the base, pipelined, and shared designs for our Arria 10 FPGA project, it's evident that each design approach offers unique advantages in terms of efficiency and optimization. The base design is remarkably lean, using only 36 Adaptive Logic Modules (ALMs) after accounting for virtual I/Os, which translates to a mere 1% of the FPGA's total resources. Even more impressive is the strategic use of DSP blocks, with the design initially using 8 blocks but effectively reducing this to 6 through dense packing, showing the design's resourcefulness in maximizing hardware efficiency.

On the other hand, the pipelined design, while demanding a slightly higher count of 97 ALMs due to the inherent complexity of pipelining, still maintains a low footprint, utilizing only about 1% of the available ALMs. This design further optimizes DSP block usage, decreasing the count to 4 from the base design's 6, re-

flecting the tool's capability to enhance resource allocation through dense packing. The shared design takes optimization a step further by requiring just 73 ALMs, maintaining the 1% utilization rate while significantly reducing DSP block usage to 2. This drastic decrease highlights the shared design's innovative approach to leveraging single modules for multiple functions, thereby freeing up resources for additional features or scalability within the same FPGA framework. Each design iteration reflects a thoughtful balance between performance needs and resource constraints, illustrating the adaptability and efficiency of FPGA-based designs in handling varying computational demands.

5.0.3 Power Analysis

Moving from the base design through the pipelined version to the shared setup, we've seen a real journey in how power is used across our FPGA projects. The base setup was pretty lean, consuming power at around 3W total, with even the busiest bits like the DSP blocks barely touching the meter. But then, we cranked things up with the pipelined design, and saw the power needs jump a bit, especially with those DSP blocks and registers working overtime, pushing the total to just over 3022 mW. The shared design, though, found a sweet spot. Even with a bit more going on than in the base model, it kept things tighter than the pipelined one, with the total power draw reining in at 3001.51 mW.

5.0.4 Calculations and Results

The maximum number of copies of a circuit that can be accommodated within an FPGA device is given by:

$$\text{Max number of Copies} = \min\left(\frac{\text{Total ALMs}}{\text{Number of ALMs in circuit}}, \frac{\text{Total Number of DSPs}}{\text{Number of DSPs in Circuit}}\right) \tag{5.1}$$

$$\text{Total Throughput for a full device} = \text{Number of Copies} \times \text{Operational Frequency of one copy} \tag{5.2}$$

$$\text{Throughput per Watt} = \frac{\text{Total Throughput}}{\text{Total Power Consumption}} \tag{5.3}$$

Using the above equations we will calculate and fill the table below

Base Design

Max Number of Copies Per Device
Using equation 5.1 the total number of copies will be

$$\text{Max Number of Copies} = \min\left(\frac{427,200}{36}, \frac{1518}{6}\right)$$

$$\text{Max Number of Copies} = \min(11,867,253)$$

$$\text{Max Number of Copies} = 253$$

Total Throughput for Full Device
Using 5.2 total throughput for full device can be calculated as

$$\text{Total Throughput for Full Device} = 253 \times 46.7\text{MHz}$$

$$\text{Total Throughput for Full Device} = 253 \times 46.7\text{MHz}$$

$$\text{Total Throughput for Full Device} = 11.8151 \times 10^9$$

$$\text{Total Throughput for Full Device} = 11.8151\text{GOPS}$$

Throughput per Watt
Using 5.3 we can find throughput per watt for our device

$$\text{Throughput per Watt} = \frac{11.8151 \times 10^9}{253 \times 3.50\text{mW}}$$

$$\text{Throughput per Watt} = \frac{11.8151 \times 10^9}{253 \times 3.50\text{mW}}$$

$$\text{Throughput per Watt} = 13.34\text{GOPS/W}$$

Pipeline Design

Max Number of Copies Per Device
Using equation 5.1 the total number of copies will be

$$\text{Max Number of Copies} = \min\left(\frac{427,200}{97}, \frac{1518}{4}\right)$$

$$\text{Max Number of Copies} = \min(4, 404, 379)$$

$$\text{Max Number of Copies} = 379$$

Total Throughput for Full Device
Using 5.2 total throughput for full device can be calculated as

$$\text{Total Throughput for Full Device} = 379 \times 327.56\text{MHz}$$

$$\text{Total Throughput for Full Device} = 124.15 \times 10^9$$

$$\text{Total Throughput for Full Device} = 124.15\text{GOPS}$$

Throughput per Watt
Using 5.3 we can find throughput per watt for our device

$$\text{Throughput per Watt} = \frac{124.15 \times 10^9}{379 \times 26.31\text{mW}}$$

$$\text{Throughput per Watt} = 12.45\text{GOPS/W}$$

Shared Design

Max Number of Copies Per Device
Using equation 5.1 the total number of copies will be

$$\text{Max Number of Copies} = \min\left(\frac{427,200}{73}, \frac{1518}{2}\right)$$

$$\text{Max Number of Copies} = \min(5852, 759)$$

$$\text{Max Number of Copies} = 759$$

Total Throughput for Full Device
Using 5.2 total throughput for full device can be calculated as

$$\text{Total Throughput for Full Device} = 759 \times 169.2\text{MHz}$$

$$\text{Total Throughput for Full Device} = 128.42 \times 10^9$$

$$\text{Total Throughput for Full Device} = 128.42\text{GOPS}$$

Throughput per Watt
Using 5.3 we can find throughput per watt for our device

$$\text{Throughput per Watt} = \frac{128.42 \times 10^9}{759 \times 9.89\text{mW}}$$

$$\text{Throughput per Watt} = 17.11\text{GOPS/W}$$

Table 5.1: Implementation Results for the 3 Different Implementations of the Studied Circuit

Parameter	**Baseline Circuit**	**Pipelined Circuit**	**Shared HW Circuit**
Resources for one circuit	36 ALMs + 6 DSPs	97 ALMs + 4 DSPs	73 ALMs + 2 DSPs
Operating frequency	46.7 MHz	327.56 MHz	169.2 MHz
Cycles per valid output	1	Initial 10 then 1	5
Critical path	DSP mult-add + 2x DSP mult + LE adder + 6x DSP mult + LE adder	1xDSP-mult	3xDSP-mult + 12xDSP-add
Max. # of copies/device	253	379	759
Max. Throughput for a full device (GOPS)	11.8151	124.15	128.42
Dynamic power of one circuit @ 46.7MHz/327.56MHz/ 169.2MHz	3.50 mW	26.31 mW	9.89 mW
Max. through-put/Watt for a full device GOPS/W	13.34	12.45	17.11

Conclusion

In conclusion, this comprehensive exploration into the design, verification, and optimization of a Taylor Series-Based Polynomial Evaluator on the Arria 10 FPGA platform has illuminated the vast potential and flexibility of FPGA technology in addressing complex computational challenges. Through the meticulous implementation and comparative analysis of three distinct design strategies—base, pipelined, and shared resource implementations—this report has underscored the importance of a nuanced approach to FPGA design, where considerations of resource utilization, operational frequency, power efficiency, and computational throughput are balanced to achieve optimal performance.

The base implementation served as a foundational platform, demonstrating the viability of FPGA-based function approximation using the Taylor series, with a focus on simplicity and proof of concept. The subsequent exploration of a pipelined design revealed significant improvements in operational frequency and throughput, highlighting the benefits of parallel processing capabilities inherent in FPGAs. This approach, however, necessitated a more complex design and increased resource utilization. In contrast, the shared resource implementation presented an innovative solution to maximize hardware efficiency by minimizing the number of instantiated modules, thus offering a compelling strategy for resource-constrained scenarios.

As evidenced by the detailed analysis and results, each design approach offers unique advantages and trade-offs, making them suitable for different application requirements and constraints.

Appendix A: Verilog Code

For the source code of base design:
Base_Design.v.
For source code of Pipeline design:
Pipeline_Design.v
For source code of shared Implementation:
Shared_Design.v
For source code of testbench:
Testbench.v

Appendix B: Results

B.1 Base Design

Timing Reports
Report Timing
Clocks
Setup Summary
Resource Utilization
Resource Usage Summary
Resource Utilization By Block
Power Analysis
Power Summary
Power Utilization by Block

B.2 Pipeline Design

Timing Reports
Report Timing
Clocks
Setup Summary
Resource Utilization
Resource Usage Summary
Resource Utilization By Block
Power Analysis
Power Summary
Power Utilization By Block

B.3 Shared Design

Timing Reports
Report Timing
Clocks
Setup Summary
Resource Utilization
Resource Usage Summary
Resource Utilization By Block

Power Analysis
Power Summary
Power Utilization By Block

Bibliography

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, 1965.

[2] E. W. Weisstein, *Taylor Series*. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/TaylorSeries.html`.

[3] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann, 2007.