Group #:
G12

Final Project

ENSC 350 1257

Project Submitted in Partial Fulfillment of the
Requirements for Ensc 350
Towards a Bachelor Degree in Engineering Science.

Last Name:	Last Name:	Last Name:
Saghafi	Schaufele	Singh
SID: 30145	**SID:** 30145	**SID:** 30139
5814	4255	4671

Table of Contents

1 - Introduction

This project explored eight design candidates for an execution unit consisting of an adding subsystem, a logic subsystem, and a shifting subsystem. Each candidate corresponds to a unique combination of two adder circuit topologies and two shifting topologies, synthesized on two different FPGA devices. Specifically, each topology was implemented on a Cyclone IV E and an Arria II FPGA (models EP4CE115F29C7 and EP2AGX45DF29C6, respectively) to enable timing simulations using Quartus, which provides detailed timing and configuration data for these devices. The two adder topologies were designed based on carry-propagation and conditional carry-propagation principles, while both shifting topologies were based on the barrel-shifting principle. The VHDL entities that form the foundation for all topologies are provided in Appendix A.3.

2 - Experimental Procedures

In this experiment, four execution-unit topologies were implemented in VHDL and synthesized in Quartus for two FPGA devices: (1) a ripple-carry adder (RCA) with a manual barrel shifter, (2) a carry-bypass adder (CBA) with a manual barrel shifter, (3) an RCA with a shifting unit using IEEE shifting functions, and (4) a CBA with a shifting unit using IEEE shifting functions. Each design was analyzed, fitted, and compiled to evaluate hardware synthesis behaviour and verify correct implementation.

Following synthesis, a configuration VHDL file and several script files were generated using a Python script, and a comprehensive testbench was developed for both functional and timing simulations. The testbench read test vectors from a `.tvs` file containing predefined inputs and expected outputs, which were applied to the adders to verify correct operation under various input conditions. The overall experimental flow is illustrated in the figure below.

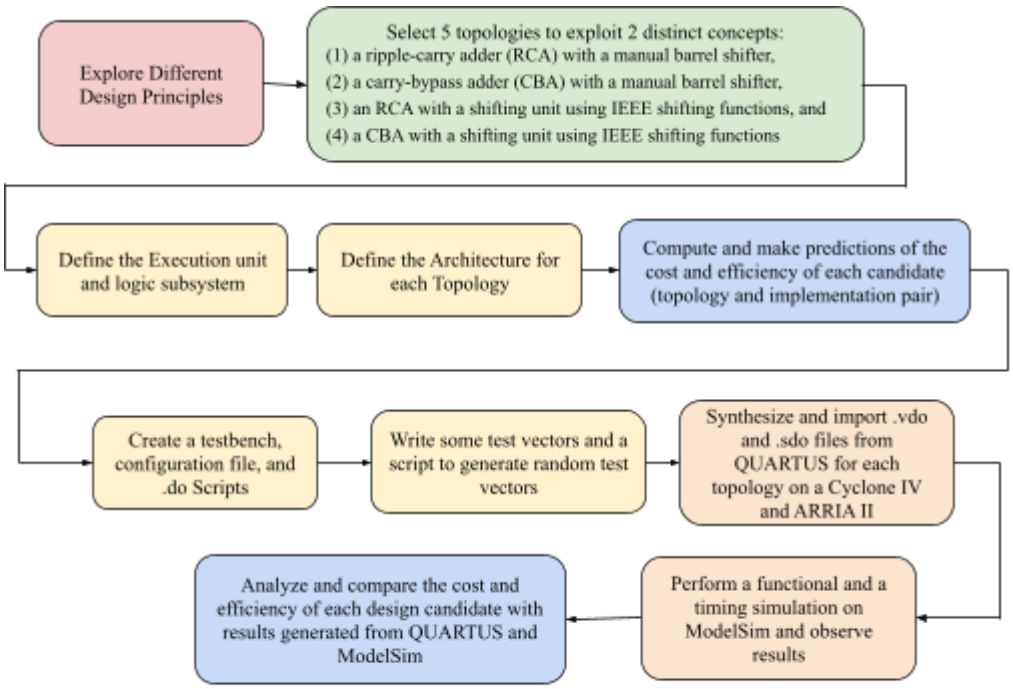

Figure 1: Flow of experimental procedure

3 - Design Candidates

Adding Subsystem Topologies

Ripple-Carry Adder (RCA)

The ripple-carry adder topology is based on the design principle of carry propagation, where each full adder waits for the carry-out from the previous bit before producing its own sum and carry. The circuit shown in Figure 2 illustrates the first two bits of the addition; the pattern extends identically for all 64 bits. This design, implemented on a Cyclone IV, was treated as a baseline to establish functional correctness and cost benchmarks, representing the most straightforward approach.

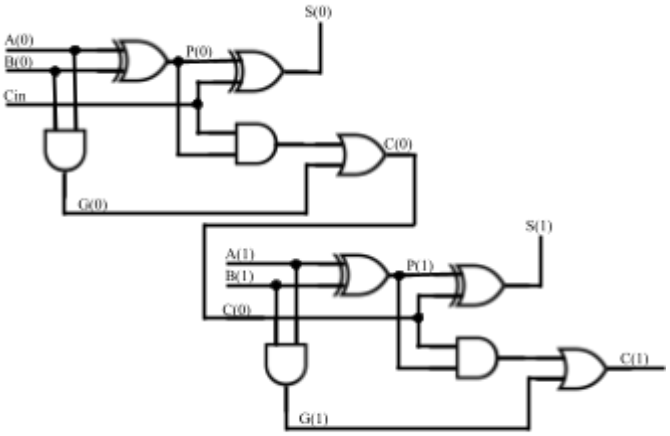

Figure 2: Ripple Adder (case N=2)

Carry-Bypass Adder (CBA)

The Carry-Bypass adder uses a look-ahead carry network to generate fast internal carries. The group propagates a signal from each block to a multiplexer (mux) that determines whether the block can be bypassed. This bypass mechanism shortens the critical path whenever all bits in a block propagate. This design is beneficial for low-cost implementations that aim to enhance the performance of a ripple adder in most use cases.

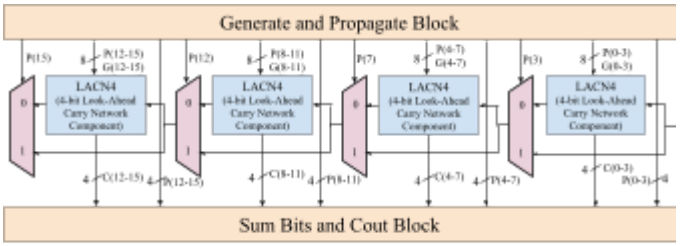

Figure 3: Carry-Bypass Adder (case N=16)

Shifting Subsystem Topologies

The shifting subsystem topologies are both barrel shifters; however, one was implemented manually using signal-assignment multiplexers, while the other relied on IEEE shift functions (shift_left, shift_right). The manual version explicitly exposes each shifting stage, giving complete control over the datapath and making timing behaviour easy to trace. The IEEE-based version hides these structural details and allows Quartus to infer the underlying hardware, potentially producing a more optimized or device-specific implementation. Comparing the two illustrates the contrast between explicit RTL design and synthesis-inferred logic.

Logic Subsystem Topology

The logic subsystem precomputes LUI and the logic operations AND, OR, and XOR. One of these results is selected based on the two-bit LogicFN input signal: ‘00’ corresponds to LUI, ‘01’ to XOR, ‘10’ to OR, and ‘11’ to AND. The logic operations are computed normally, while the LUI channel effectively acts as a pass-through wire, since the immediate value used for the LUI instruction is assumed to have been formed earlier in the unpack-upper-immediate stage before reaching the execution unit.

Design Implementations

Each adder topology was implemented on two FPGA devices using structural VHDL to evaluate how architectural differences affect resource utilization and cost. The Cyclone IV uses 4-input 1-output Logic Elements (LEs), which serve as the device’s fundamental building blocks for combinational and sequential logic. In contrast, the Arria II uses 8-input 2-output Adaptive Logic Modules (ALMs). Both primitive elements act as look-up tables, utilizing the architecture's Boolean expressions to map the circuit structure. Comparing designs across these devices highlights how the same circuit structure is mapped and optimized differently onto FPGA families with different primitive elements.

Candidate Estimations

Cost

The cost of each candidate was split into 4 separate calculations: The adding element, the shifting element, the logic element and the encapsulating execution unit.

The estimated cost for the RCA topology was 130 LEs on the Cyclone IV E and 66 ALMs on the Arria II, calculated using the following formula:

$$Total\ LE/ALMs\ = (LE/ALMs\ for\ N\ =\ 2\ Addition)\ *\ N/2\ +\ Ovfl\ +\ Cout$$

The estimated cost for the CBA topology was 304 LEs on the Cyclone IV E and 192 ALMs on the Arria II, computed using the following formula:

$$Total\ LE/ALMs\ = (LE/ALMs\ for\ LACN4)\ *\ N/4\ +\ (LE/ALMs\ needed\ for\ G\ \&\ P\ \&\ Sum)\ *\ N\ +\ (LE/ALMs\ for\ muxs)\ *\ N/4$$

The estimated cost of the shifting unit was 384 LEs and 192 ALMs, calculated using the following formula:

$$Total\ LE/ALMs\ =\ Log_2(N)\ \times\ LE/ALMs\ for\ N\ muxes$$

Both the Barrel and IEEE_fn architectures use the same general structure, which was confirmed by the post-fitting netlist; thus, we estimate the cost for both to be roughly the same.

The estimated cost for the logic unit was 320 LEs and 160 ALMs, calculated using the following formula:

$$Total\ LE/ALMs\ =\ LE/ALMs\ for\ N\ gates\ \times\ 3\ +\ LE/ALMs\ for\ output\ selection\ muxes$$

The estimated cost for the overarching execution unit was 635 LEs and 310 ALMs, calculated using the following method: 7 N-bit 2-channel muxes + 1 N-bit 4-channel mux + 3 gates for output flags + 64 input NOR gate for the zero flag.

The estimated total cost for each candidate was then calculated using the following formula:

$$Total\ LE/ALMS\ =\ C_{adder}\ +\ C_{shift}\ +\ C_{logic}\ +\ C_{ExU}$$

Timing

The worst-case timing estimation for each candidate was calculated using the following formula:

$$T_{total}\ =\ T_{adder}\ +\ T_{ExU\ Muxes}$$

We chose this method because the adder has the longest propagation delay of all execution unit entities. The logic unit has only 1 level of gate delay, and the shifting unit only has $log_2(N)$ levels of delay, while both the RCA has a worst-case delay of N. Since all 3 modules are computed in parallel, we only consider the longest possible path, which would be through the adding unit.

The worst-case timing delay of the RCA was found to be 66 levels of LEs/ALMs, 64 for the carry network, 1 for the generate-and-propagate block, and 1 for the sum-generation block.

The worst-case timing delay of the CBA was estimated to be 50 levels of LEs on the Cyclone IV E and 34 levels of ALMs on the Arria II, using the following formula:

$$Max\ Delay\ = (Delay\ of\ LACN4)\ *\ N/4\ +\ Delay\ of\ G\ \&\ P\ \&\ Sum\ blocks$$

4 - Testbenches

In this experiment, a single testbench was used to verify both the functional correctness and timing behaviour of the execution unit. The testbench reads a standard test-vector file, Exec64.tvs, which includes worst-case timing vectors for addition and subtraction, logic operations, shifting, set-less-than and set-less-than-unsigned cases, and several randomly generated vectors. Each test vector is stored on a single line in the format

A, B, FuncClass, LogicFN, ShiftFN, AddnSub, ExtWord, Y, Zero, AltB, AltBu,

Where A and B are 64-bit operands in hexadecimal; FuncClass, LogicFN, and ShiftFN are 2-bit input select signals; AddnSub and ExtWord are 1-bit input select signals; Y is the 64-bit output; and Zero, AltB, and AltBu are status flags.

A configuration file was created to test every topology, and a Python program was written to generate several .do scripts to automate both timing and functional simulations for each candidate design.

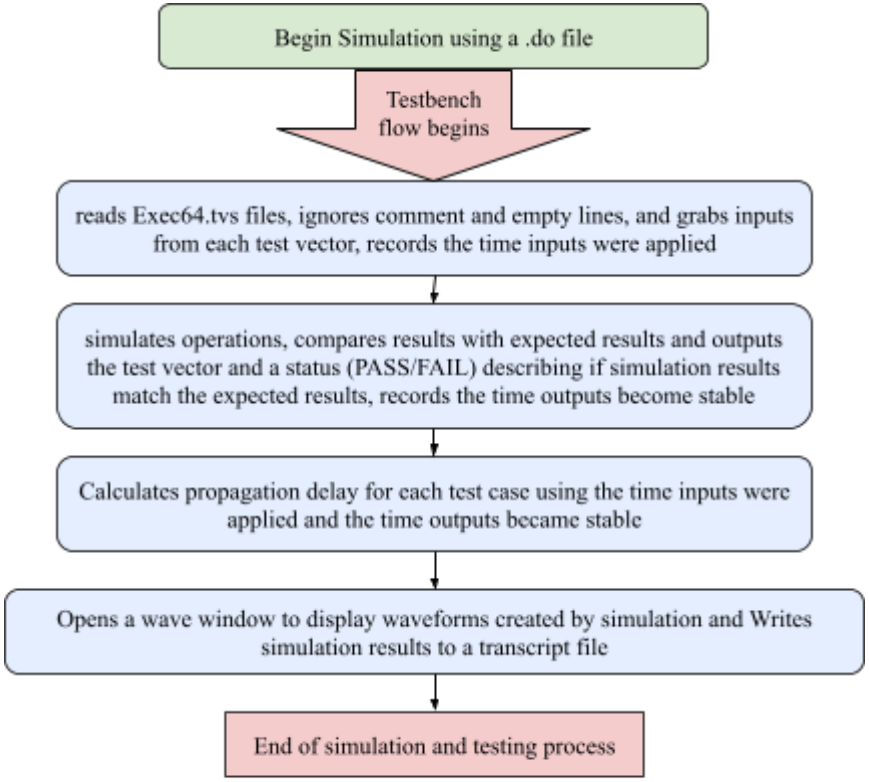

Figure 4: Flow of Simulation

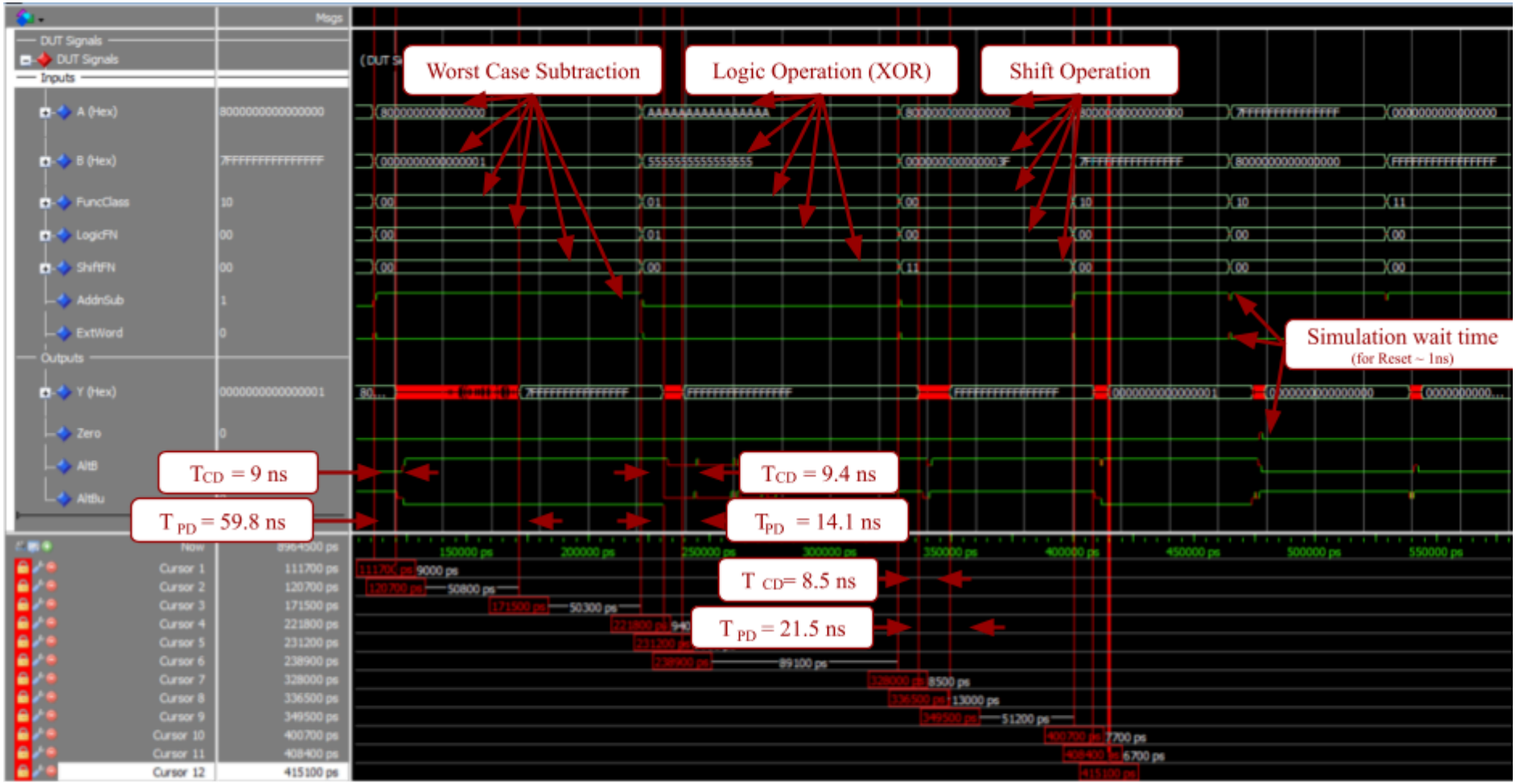

Figure 4: Sample outputs Waveform from the timing simulation of the Execution Unit using RCA and Manual Barrel Shifting on Cyclone IV

5 - Cost Analysis

The costs for the design candidates were calculated using this formula:

$$Cost\ per\ LE/ALM\ =\ Device\ price\ \div\ Total\ Device\ LEs/ALMs$$

This method of calculating the cost factors accounts for the total resources available to the board and the fraction of those resources utilized by the circuit. Resource data was obtained from the Quartus synthesis reports. LEs were selected as the unit of cost, and all ALMs were converted to LEs using a conversion factor. The normalized cost is obtained by dividing each candidate cost by the baseline cost. The boards used to synthesize the architectures are currently obsolete [1][2]. This inflates the price and makes cost calculations inaccurate. To account for these inaccuracies, the Cyclone IV Nano [3] and the ARRIA 10 GX 480 [4] were considered. These devices are comparable to those used in synthesis; however, they have more accurate pricing because they are currently listed on DigiKey, indicating they are still widely used and in production. The following calculations assume that the LEs and ALMs on the available devices cost the same as the original board's elements.

Device	**Total Device LEs/ALMs**	**Device Price (CAD)**	**Cost per LE/ALM (CAD)**	**Normalized Cost in terms of LEs**
Cyclone IV Na	22320	$169.49 [3]	$0.00759184 [3]	1
Arria 10 GX 480	181790	$3937.25 [4]	$0.02165823 [4]	2.85

Design Candidate	Target Device	Resource Usage Predications (LEs/ALMs)	Resources Used (LEs/ALMs)	Predicted Cost (CAD)	Candidate Cost (CAD)	Cost (LEs)	Normalized Cost (W.R.T. Baseline LEs)
1 (Baseline)	Cyclone IV	1469	1149	$11.15	$8.72	1149	1.000
2 (RCA + Barrel Shifter)	ARRIA II	728	744	$15.76	$16.11	2120	1.847
3 (RCA + IEEE FN)	Cyclone IV	1469	1217	$11.15	$9.24	1217	1.059
4 (RCA + IEEE FN)	ARRIA II	728	756	$16.76	$16.37	2155	1.877
5 (CBA+ IEEE FN)	Cyclone IV	1643	1421	$12.47	$10.79	1421	1.237
6 (CBA+ IEEE FN)	ARRIA II	854	901	$18.50	$19.51	2568	2.237
7 (CBA + Barrel Shifter)	Cyclone IV	1643	1329	$12.47	$10.09	1329	1.157
8 (CBA + Barrel Shifter)	ARRIA II	854	883	$18.50	$19.12	2517	2.192

Figure 5: Cost Calculations of Design Candidates

The conversion factor from LEs to ALMs was 2.85, slightly more than double.

6 - Timing Analysis

For timing simulation and analysis, the .vho and .sdo files created by Quartus for each design candidate were used, along with ModelSim's timing capabilities, to simulate and calculate the propagation delays.

From the device datasheets, the worst-case timing delays of a single LE/ALM were determined using the slowest elements available on each device for timing estimation.

For the Cyclone IV E, the worst-case timing delay was found to be 9.94 ns:

- 4.146 ns input delay from pin to internal cells
- 4.374 ns input delay from pin to input register
- 1.420 ns delay from the output register to the output pin

$$Total\ Delay\ (LE) = 4.146 + 4.374 + 1.420 = 9.94\ ns$$

For the Arria II, the worst-case timing delay was 7.65 ns:

- 0.873 ns output enable pin delay
- 0.722 ns delay from output register to output pin
- 2.944 ns input delay from pin to internal cell
- 2.944 ns input delay from pin to input register

$$Total\ Delay\ (ALM) = 0.873 + 0.722 + 2.944 + 2.944 = 7.65\ ns$$

Design Candidate	**Target Device**	**Estimated Levels of Delay**	**Estimated Max Propagation Delay (ns)**	**Max Propagation Delay (ns)**	**Performance** (W.R.T. Baseline Simulation)
1 (Baseline)	Cyclone IV	72	715.68	98.8	1
2 (RCA + Barrel Shifter)	ARRIA II	72	550.8	103.3	0.956
3 (RCA + IEEE FN)	Cyclone IV	72	715.68	102.2	0.967
4 (RCA + IEEE FN)	ARRIA II	72	550.8	96.2	1.027
5 (CBA+ IEEE FN)	Cyclone IV	56	556.64	76.7	1.288
6 (CBA+ IEEE FN)	ARRIA II	40	306	66.0	1.497
7 (CBA + Barrel Shifter)	Cyclone IV	56	556.64	86.3	1.14
8 (CBA + Barrel Shifter)	ARRIA II	40	306	64.3	1.536

Figure 6: Propagation Delay Calculations of Design Candidates

7 - Optimization Analysis

Candidate		N = 16	N = 32	N = 64	N =128
1	**RCA Barrel on Cyclone**	232	572	1228	2516
2	**RCA Barrel on ARRIA**	162	366	735	-
3	**RCA IEEE on Cyclone**	252	558	1283	2656
4	**RCA IEEE on ARRIA**	166	343	733	-
5	**CBA IEEE on Cyclone**	303	657	1457	2874
6	**CBA IEEE on ARRIA**	197	413	862	-
7	**CBA Barrel on Cyclone**	283	669	1379	2810
8	**CBA Barrel on ARRIA**	179	443	839	-

The Cyclone IV E reached a bottleneck at 256 bits, and the Arria II at 128 bits, due to the limited number of I/O pins available on these devices to support wide data paths. The LE/ALM usage also increases sharply with the number of bits, as shown in Figure X. Another notable observation is that ALMs become increasingly efficient at higher bit-widths compared to Cyclone IV LEs, which is reflected in the diverging slopes of the two curves.

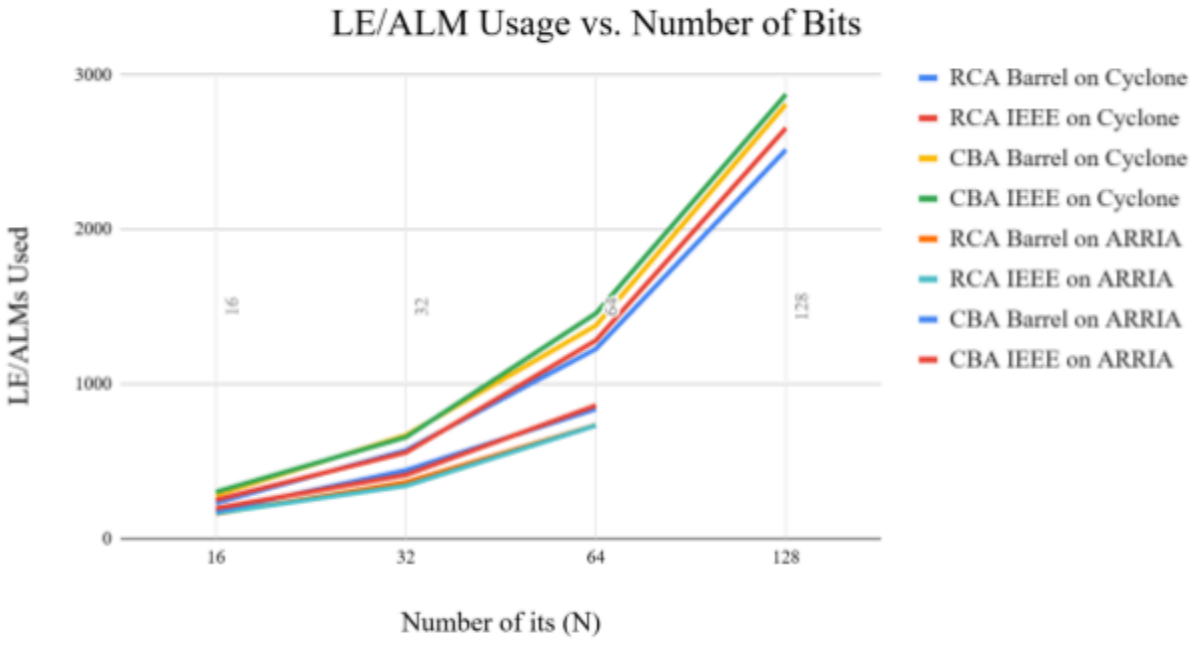

Figure 7: LE/ALM usage for each N

8 - Conclusion

Design Candidate	Target Device	Normalized Cost (W.R.T. Baseline LEs)	Performance (W.R.T. Baseline Simulation)	Performance to Cost Ratio
1 (Baseline)	Cyclone IV	1.000	1	1
2 (RCA + IEEE FN)	ARRIA II	1.847	0.956	0.517
3 (RCA + Barrel Shifter)	Cyclone IV	1.059	0.967	0.913
4 (RCA + Barrel Shifter)	ARRIA II	1.877	1.027	0.830
5 (CBA+ IEEE FN)	Cyclone IV	1.237	1.288	1.04
6 (CBA+ IEEE FN)	ARRIA II	2.237	1.497	0.669
7 (CBA + Barrel Shifter)	Cyclone IV	1.157	1.14	0.985
8 (CBA + Barrel Shifter)	ARRIA II	2.192	1.536	0.701

Figure 8: Cost and Performance Comparison of Design Candidates

This comprehensive exploration of 64-bit execution unit architectures across Cyclone IV and Arria II FPGAs reveals clear optimization pathways tailored to specific design constraints. The analysis demonstrates that the optimal architectural choice is highly dependent on whether the primary goal is maximizing performance-to-cost efficiency, achieving pure performance, or minimizing resource utilization.

For the best balance of performance and cost, the Carry-Bypass Adder with IEEE shift functions implemented on Cyclone IV emerges as the superior choice. This configuration delivers a 28.8% performance improvement over the baseline ripple-carry adder while requiring only 23.7% additional logic elements, yielding the highest performance-to-cost ratio of 1.04. The Cyclone IV platform consistently proves more efficient for balanced designs, with all optimized candidates maintaining performance-to-cost ratios above 0.90, significantly outperforming Arria II implementations in this metric. The Carry-Bypass Adder architecture itself strikes an effective middle ground between the simple but slow ripple-carry approach and more complex, resource-intensive alternatives.

When absolute performance is the paramount concern regardless of cost, the Carry-Bypass Adder with barrel shifter on Arria II provides the highest computational throughput, achieving a 53.6% performance

improvement over baseline. However, this comes at a substantial 119.2% increase in resource utilization, illustrating the diminishing returns of aggressive optimization on high-performance FPGAs. The Arria II platform generally enables higher performance ceilings, with all Arria-based Carry-Bypass Adder designs showing performance ratios exceeding 1.49× baseline, though at significantly higher resource cost and lower efficiency ratios ranging from 0.517 to 0.830.

For severely cost-constrained applications, the ripple-carry adder with barrel shifter on Cyclone IV represents the most resource-efficient option, requiring only 5.9% additional logic elements while maintaining 96.7% of baseline performance. The shift unit implementation choice proves particularly significant for cost optimization, with IEEE shift functions generally requiring fewer resources than barrel shifters, especially at larger operand widths where barrel shifter resource consumption grows more rapidly.

Scalability analysis reveals predictable growth patterns, with Carry-Bypass Adder designs showing better performance scalability than ripple-carry implementations as operand width increases. The Arria II platform consistently requires 30–40% fewer resources than Cyclone IV for equivalent functionality, though this advantage must be evaluated against overall system costs. Practical design decisions should therefore consider not only architectural choices but also platform characteristics, with Cyclone IV favoring efficiency-oriented implementations and Arria II better suited for performance-critical applications where resource constraints are secondary. These findings underscore that effective digital design optimization requires careful matching of architecture to application requirements and platform capabilities, with the Carry-Bypass Adder emerging as the most versatile core arithmetic unit across diverse implementation scenarios.

8 - Appendix

A.1 Directory Structure

(folder) Documentation

- **FP-Report-G12-350-1257.pdf**

  Final project report including cost and efficiency analysis.

- **FP-Summary-G12-350-1257.pdf**

  Summary of tasks completed and overview of cost/efficiency results.

- **ProjectLog-G12-xxxx-350-1257.pdf**

  Each group member's activity log (xxxx = last 4 digits of student ID).

(folder) Images

Contains all images used throughout the report.

(folder) OutputFiles

- **Quartus-Summaries.txt**

  Summary reports from Quartus synthesis for every ExecUnit topology.

- **FS-xxx-xxx-Transcript.txt**

  Functional simulation transcripts for each topology combination

  (xxx = abbreviation of each sub-block topology, e.g., RCA, CBA, LAC, etc.).

- **TS-CYC-xxx-xxx-Transcript.txt**

  Timing simulation transcripts on the **Cyclone IV** device.

  TS-ARRIA-xxx-xxx-Transcript.txt

  Timing simulation transcripts on the **Arria II** device.

- **VHDLSrc_EN_ExecUnit.pdf**

  PDF listing of the ExecUnit VHDL source.

- **VHDLSrc_EN_Logic.pdf**

  PDF listing of the Logic subsystem VHDL source.

- **VHDLSrc_EN_Shift.pdf**

  PDF listing of the Barrel Shifter VHDL source.

- **VHDLSrc_EN_Adder.pdf**

  PDF listing of the Adder entity and all adder architectures used in ExecUnit.

- **VHDLSrc_TB_ExecUnit.pdf**

  PDF listing of the general testbench.

- **VHDLSrc_Config_ExecUnit.pdf**

  PDF listing of all VHDL configurations for executing FS/TS simulations.

(folder) Simulation

(folder) ModelSim / Questa

Contains all auto-generated ModelSim files produced during compilation:

- .vho files
- .sdo delay annotation files

  For every ExecUnit candidate architecture.

(folder) TestVectors

- **Exec64.tvs**

  Test vector file for the 64-bit ExecUnit testbench.

- **gen_testVec.py**

  Python script used to append randomly generated vectors to Exec64.tvs.

- **FP.mpf / FP.cr.mti**

  ModelSim project files.

- **wave.do**

  Waveform setup script used in all FS/TS runs.

- **FS_xxx_xxx.do**

  Functional simulation scripts for each pair of logic/adder/shift topologies.

- **TS_CYC_xxx_xxx.do**

  Timing simulation do-files for Cyclone IV device.

- **TS_ARRIA_xxx_xxx.do**

  Timing simulation do-files for Arria II device.

- **FS_run.do**

  Runs functional simulation for **all** topology combinations.

- **TS_run.do**

  Runs timing simulation for **all** topology combinations.

- **runall.do**

  Runs every functional and timing simulation automatically.

- **TB_ExecUnit.vhd**

  The general VHDL testbench for the execution unit.

- **Config_ExecUnit.vhd**

  All VHDL configuration declarations used for FS/TS simulations.

(folder) SourceCode

- **EN_ExecUnit.vhd**

  Top-level execution unit containing combinational selection logic for

  Logic → Shift → Add/Sub → Output flags.

- **EN_Logic.vhd**

  Logic subsystem (AND, OR, XOR, LUI).

- **EN_Shift.vhd**

  Barrel shifter providing LL, RL, and RA operations.

- **EN_Adder.vhd**

  Adder subsystem including:

  - ripple-carry adder
  - fast ripple
  - conditional sum adder
  - look-ahead carry tree

    Brent–Kung adder

    carry-bypass adder

- **FP.qpf / FP.qsf**

  Quartus project files for synthesis, timing analysis, and Fmax evaluation.

A.2 References

[1]
"EP4CE115F29C7 Altera | Integrated Circuits (ICs) | DigiKey." Accessed: Oct. 12, 2025. [Online]. Available: https://www.digikey.ca/en/products/detail/altera/EP4CE115F29C7/2260452

[2]
"EP2AGX45DF29C6 Altera | Integrated Circuits (ICs) | DigiKey." Accessed: Oct. 12, 2025. [Online]. Available: https://www.digikey.ca/en/products/detail/altera/EP2AGX45DF29C6/2349475

[3]
"P0082 Terasic Inc. | Development Boards, Kits, Programmers | DigiKey." Accessed: Oct. 25, 2025. [Online]. Available: https://www.digikey.ca/en/products/detail/terasic-inc/P0082/2625112

[4]
"IW-G24D-CU0F-4D004G-S008G-NCI," DigiKey Electronics. Accessed: Oct. 25, 2025. [Online]. Available: https://www.digikey.ca/en/products/detail/iwave-global/IW-G24D-CU0F-4D004G-S008G-NCI/15780313

[5]
"Cyclone IV Device Handbook, Volume 3." Dec. 2016.

[6]
"Device Interfaces and Integration for the Arria® II Device Handbook." Accessed: Nov. 15, 2025. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/654011/device-interfaces-and-integration-for-the-arria-ii-device-handbook.html

A.3 VHDL Entities

Adder Entity Definition within EN_Adder.vhd

```
entity EN_Adder is
    generic (N: natural := 16);
    port (
        A, B  : in std_logic_vector (N-1 downto 0);
        S : out std_logic_vector (N-1 downto 0);
        Cin : in std_logic;
        Cout, Ovfl : out std_logic
    );
end EN_Adder;
```

Look-Ahead Carry Generator Block Entity Definition within EN_LACN4.vhd

```
entity EN_LACG4 is
    generic (N: natural := 4);
    port (
        Gin, Pin  : in std_logic_vector (N-1 downto 0);
        Gout, Pout  : out std_logic;
        Cin : in std_logic;
        C : out std_logic_vector (N-1 downto 1)
    );
end EN_LACG4;
```

Shift Unit Definition within EN_Shift.vhd

```
entity EN_Shift is
    Generic ( N : natural := 64 );
    Port (
        A : in std_logic_vector( N-1 downto 0 );
        ShiftCount : in std_logic_vector( 5 downto 0 );
        Y_LL , Y_RL, Y_RA : out std_logic_vector (N-1 downto 0)
        );
end EN_Shift;
```

Logic Unit Definition with EN_Logic.vhd

```
entity EN_Logic is
    Generic ( N : natural := 16 );
    Port (
        A, B : in std_logic_vector(N-1 downto 0);
        LogicFN : in std_logic_vector(1 downto 0);
        Y : out std_logic_vector(N-1 downto 0)
    );
end entity EN_Logic;
```

Execution Unit definition with EN_ExecUnit.vhd

```
Entity EN_ExecUnit is
    Generic ( N : natural := 64 );
    Port ( A, B : in std_logic_vector( N-1 downto 0 );
    FuncClass, LogicFN, ShiftFN : in std_logic_vector( 1 downto 0 );
    AddnSub, ExtWord : in std_logic := '0';
    Y : out std_logic_vector( N-1 downto 0 );
    Zero, AltB, AltBu : out std_logic );
End Entity EN_ExecUnit;
```