2008

Implementation Of Null Conventional Logic In COTS FPGA's

Mohamad Rajgara
Wright State University

Implementation of NULL CONVENTIONAL LOGIC in COTS FPGA's

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering

By

MOHAMAD RAJGARA
B.E. in Electronics, University of Mumbai, India, 2003

2008
Wright State University

WRIGHT STATE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

December 19, 2008

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Mohamad Rajgara ENTITLED Implementation of NULL CONVENTIONAL LOGIC in COTs FPGA BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF Master of Science in Engineering

John M. Emmert, Ph.D
Thesis Director

Kefu Xue, Ph.D
Chair, Dept. of Electrical Engineering

Committee on Final Examination

John M. Emmert Ph.D
Professor, Dept. of Electrical Engineering

Ray.E.Siferd Ph.D
Professor Emeritus, Dept. of Electrical Engineering

Henry Chen Ph.D
Professor Emeritus, Dept. of Electrical Engineering

Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

ABSTRACT

Rajgara, Mohamad M.S.E., Department of Electrical Engineering, Wright State University, 2008
Implementation of NULL CONVENTIONAL LOGIC in COTS FPGA's

The digital world has been dominated by the growth of synchronous digital design techniques for last few decades. Traditional Boolean logic is symbolically incomplete since it has a separate control logic, time in the form of clock signal which be carefully integrated with the logic design, but with digital clock signal already in the GHz range, this integration is further complicated. One approach to address post GHz digital logic design is to use clockless or asynchronous digital design techniques. In the mid '90s Theaseus Logic Inc. proposed a method to design Asynchronous circuits using Null Convention Logic (NCL).

This thesis introduces a technique for mapping NCL circuits and applications onto commercial-off-the-shelf (COTS) field programmable gate arrays (FPGAs). NCL logic was introduced as a four value logic (as opposed to the two valued standard Boolean) then transitioned to a three value logic, and finally to a two value logic which is similar to a two valued Boolean logic but with integrated control logic. We extend this theory to basic functional gates that can be implemented in FPGA functional look-up-tables (LUTs). To demonstrate the techniques we map basic adder circuits then extend that to a more complicated Fast Fourier Transform (FFT) circuit. All blocks were built and implemented on a Xilinx Virtex II pro FPGA. The circuits were tested using Modelsim and implemented using Xilinx Platform Studio.

Table of Contents

1. INTRODUCTION **1**

2. WHY ASYNCHRONOUS CIRCUITS? **3**

2.1 WHAT IS NULL CONVENTIONAL LOGIC? 4
2.2 DERIVING 4 NCL TO 3 NCL 8
2.3 TWO VALUE NULL CONVENTIONAL LOGIC 10
2.4 THRESHOLD GATE 10
2.5 BASIC LOGICAL OPERATORS USING NCL 14
2.6 FPGA – VIRTEX II PRO 17

3. IMPLEMENTING FFT USING NULL CONVENTIONAL LOGIC **19**

3.1RADIX 2 BUTTERFLY ARCHITECTURE 19
3.2 HALF ADDER IMPLEMENTATION IN NCL 21
3.3 FULL ADDER IMPLEMENTATION IN NCL 22
3.4TOP MODULE 23

4. TESTING AND RESULT **27**

5. CONCLUSION & FUTURE WORK **34**

6. REFERENCES **35**

List of Figures

Figure 1: *Successive wavefront transitions between different domains* 6
Figure 2*: Example Combinational Circuit* 6
Figure 3: *Threshold Gate* 11
Figure 4: *Boolean logic half adder* 11
Figure 5: *NULL Convention half adder* 12
Figure 6: *THmn Threshold Gate* 13
Figure 7: *OR Function in NCL* 15
Figure 8: *AND Function in NCL* 16
Figure 9: *XOR Function in NCL* 17
Figure 10: *FFT Butterfly* 19
Figure 11*: butterfly node in NCL* 20
Figure 12: *Optimized HALF ADDER in NCL* 22
Figure 13: *Optimized FULL ADDER in NCL* 23
Figure 14: *Top Module* 23
Figure 15: *Internal Block of the top module* 24
Figure 16: *Asynchronous FFT block* 24
Figure 17: *Internal module of AFFT block* 25
Figure 18: *Input Buffer* 26
Figure 19: *Output Buffer* 26
Figure 20: *FPGA Design Flow* 27
Figure 21: *Output waveform of TH24w2 threshold gate* 28
Figure 22: RTL Schematic of TH24w2 29
Figure 23: *Output waveform of Half Adder* 29
Figure 24: *RTL Schematic of Half Adder* 30
Figure 25: *Output waveform of Full Adder* 30
Figure 26: *RTL Schematic of Full Adder* 31
Figure 27: *Output simulation of a 16 point Asynchronous FFT* 32
Figure 28: *RTL Schematic of 16 point Asynchronous FFT* 33
Figure 29: *system view* 33

List of Tables

Table 1: *Boolean truth table with NULL value added* 5
Table 2: *truth table with INTERMEDIATE value added* 8
Table 3*: Correspondences of 4NCL and 3NCL* 9
Table 4*: truth table with FEEDBACK* 9
Table 5: *Correspondence of the logic* 10
Table 6: *27 NCL Macros* 14
Table 7*: truth table – OR Function* 15
Table 8: *Truth Table – AND Function* 16
Table 9*: truth table - XOR Function* 17
Table 10: *Truth table of HALF ADDER in NCL* 22
Table 11: *Truth table of FULL ADDER in NCL* 23
Table 12: *top module pin out* 24
Table 13: *AFFT block pin out* 25

ACKNOWLEDGEMENTS

I would like to gratefully acknowledge the enthusiastic supervision of Dr Marty Emmert without his guidance this thesis would not have been possible. I really admire his patience and the support that he provided throughout the project. I am also thankful to the Committee members Dr. Ray E. Siferd and Dr. Henry Chen for their valuable feedback. I would also like to thank my friends who were involved directly or indirectly for their support. Finally, I am forever indebted to my parents and my sister for their understanding and encouragement when it was most required.

Dedicated TO Mom Dad and Sister

Acronyms, Definitions and Symbols

This thesis document uses the following set of Acronyms, definitions and Symbols

DI	Delay Insensitive Circuits
SI	Speed Independent Circuits
NCL	Null Conventional Logic
QDI	Quasi Delay Insensitive Circuits
THmn	Threshold Gate
THmnW	Threshold Gate with Weighted inputs
TCR	Threshold combinational Reduction Method
SOP	Sum of Product
X^0	LOW(0) signal of the X in dual rail system
X^1	HIGH(1) signal of the X in dual rail system
FPGA	Field Programmable Logic Array
LUTs	Look Up Tables
MUXF5	Multiplexer F5 in a slices
IP	Intellectual Property
DSP	Digital Signal Processing
PPC405	Power PC 405 RISC Processor from IBM
RAM	Random Access Memory
GPRs	General Purpose Registers
OCM	On Chip memory controller
PLB	Processor Local Bus

OPB	On Chip Peripheral Bus
DCR	Device Control Register
EIC	External interrupt controller
JTAG	Join Test Action Group
FFT	Fast Fourier Transform
DFT	Discrete Fourier Transform
$\boldsymbol{W_N^K}$	Complex Multiplier
R[_]	Real term of a complex equation
I [_]	Imaginary term of a complex equation
?	Exclusive OR function
XOR	Exclusive OR function
OR	OR function
AND	AND function
PD	Primary input DATA from the ADC
SYS_CLK	System Clock
PES / ES	Primary ENABLE signal for input buffer
PEO / EO	Primary ENABLE signal for output buffer
PZ0 / ZERO	Primary signal ZERO for NULL DATA for Asynchronous
PEIA_Rn / EIA_Rn	Primary SELECT signal between DATA and NULL
PO	Primary output DATA from the output buffer.
ADC	Analog to Digital Converter
AFFT	Asynchronous Fast Fourier Transform

1. Introduction

The digital world is dominated by synchronous techniques from many decades due to ease in the design. Also CAD tools for designing synchronous circuits are sophisticated that it makes the designers life easier by making the complete process automated. But with clock speed in GHz the global wire delays is increasing than the gate delay. With increase in wire delay problem related to clock like clock skew, signal integrity, power, etc becomes more serious in synchronous techniques. Hence designers are evaluating new techniques to cover come wire delays. They are looking towards asynchronous techniques and/or self timed circuits for potential solution.

Synchronous circuits are implemented using traditional Boolean logic. Traditional Boolean logic is not symbolically complete. It has time dependence as well as symbolic value dependencies [1][2][3]. Symbolic values dependency depends upon interconnect in logic and their truth table. A time dependency depends on the delays in components to determine validity and invalidity of data value. Hence in mid 90's Theseus Logic Inc. proposed a method to design asynchronous circuits using NULL CONVENTIONAL LOGIC [3][1]which is symbolically complete and completely free from time dependencies. This is general implemented using dual rail signals, quad rail signals, Mutually Exclusive Assertion Groups (MEAGs), [1] since it incorporates data and control information mixed on one signal to avoid dependencies of times.

This thesis describes a technique to Implements NULL CONVENTIONAL LOGIC in commercially available FPGA's. This document describes basic understanding on NULL conventional logic. In chapter 2 it shows the advantages of Asynchronous circuits. It

shows how to make Boolean logic symbolically complete as four value logic. Later it reduces four value to three value logic and then it shows how NCL can be implemented as a two value logic which can be practically feasible to implement. It explain basic implementation of functions like XOR, AND and OR [3][1]. Later it gives a brief overview of Virtex II pro FPGA. Chapter 3 describe the implementation of 16 point Asynchronous FFT using Radix 2 Butterfly architecture [4][5]. It shows detailed optimization method for designing a half adder and full adder implemented using NCL threshold gates [1]. Chapter 4 shows method of testing the NCL implemented logic. It shows result waveform of one of the NCL macros [1]. It shows simulation and synthesis result of Adders. It also shows the embedded system implemented using PPC405.

2. Why Asynchronous Circuits?

In most digital circuit available today are synchronous. Synchronous circuits are based in two most important principles.

- Time i.e. clock signal

In an asynchronous circuit the clock signal is replaced with handshake signals or feedback paths for synchronization, communication between their components. The advantages of asynchronous circuits are:

- Low power consumption due to zero standby power consumption since there is no clock signal
- No clock distribution hence no problems of clock routing with minimum skew
- High operating speed since depends on local latency and not global latency
- Less circuit noise since no clock signal

Traditional Boolean logic are not symbolic complete in expression. They are dependent on control logic i.e. time. The time dependent expression will depend on propagation of delay to determine correctness of data and control dependent factor depends on the interconnection of different gates to generate control signals.

In late 1950's attempts were made to eliminate both the dependencies from the Boolean logic and make it symbolical complete. The attempts were:

- *Delay insensitive (DI) circuits*

DI circuits are circuits which operate correctly with positive bounded but unknown delay in wires as well as gates. Such circuits are extremely robust and expensive to design.

- *Speed independent (SI) circuits*

SI circuits are circuits which operate correctly with assume positive bounded delay in gate and zero delay in wires, but assuming zero delay wires is not realistic and not practically possible into today's semiconductor.

- *Fundamental mode circuits*

It use method of matched delay lines to provide local time reference as done in the synchronous circuits

From the above mentioned methods DI method is the most appropriate approach for designing asynchronous circuits

2.1 What is NULL CONVENTIONAL LOGIC?

NULL CONVENTIONAL LOGIC is a new technique developed for designing asynchronous circuits. NULL mean there is NODATA or spacer between corresponding DATA. It indicates no input or output is present. NCL is theoretically complete and economically viable approach to delay insensitive circuits. NCL is not purely based on DI circuit method but on a class of DI called as quasi delay insensitive (QDI). QDI is designed using Isochronic Forks. The delay in the fanout is assumed to be same [6][7][8].

NCL uses 2 criteria to achieve delay insensitive behavior

- Symbolic completeness
- Completeness of inputs

NCL logic design is implemented using threshold gates with hysteresis. These gates have many inputs and one output. Output from this gate is DATA when numbers of inputs reach or exceed the threshold. The hysteresis behavior is achieved by keeping the output is same state as previous till all the input goes to NO DATA

Making Boolean Logic Symbolically Complete.

Making Boolean logic symbolically complete is done are two steps:

First step is to satisfy input criteria of logic; we assign a value to present Boolean (true or false) assertion domain to express validity / invalidity of data. To retain the sense of Boolean logic the current two values will be expressing data and third value "NULL" will be expressing not a valid data value. Hence when both input are data output is data value (true or false) otherwise it is NULL. Table 1, shows the truth table of basic gate with NULL value added.

AND

	T	F	N
T	T	F	N
F	F	F	N
N	N	N	N

OR

	T	F	N
T	T	T	N
F	T	F	N
N	N	N	N

NOT

T	F
F	T
N	N

Table 1: ***Boolean truth table with NULL value added*** **[3] [9]**

Hence we have a **three value logic**. A transition from null to data value will assert in beginning of valid data and transition from data to null end the valid data (i.e. beginning of invalid data). This truth table enforces the **completeness of input criteria** for data

To the data value (true or false) we add NULL to express for no data. This is referred as NULL CONVENTION. The transition from data to null and via versa is referred as wavefront as shown in Fig. 1. The output transition from *complete DATA* to *complete NULL* is referred as NULL wavefront. Similarly the output transition from *complete NULL* to *complete DATA* is referred as DATA wavefront.

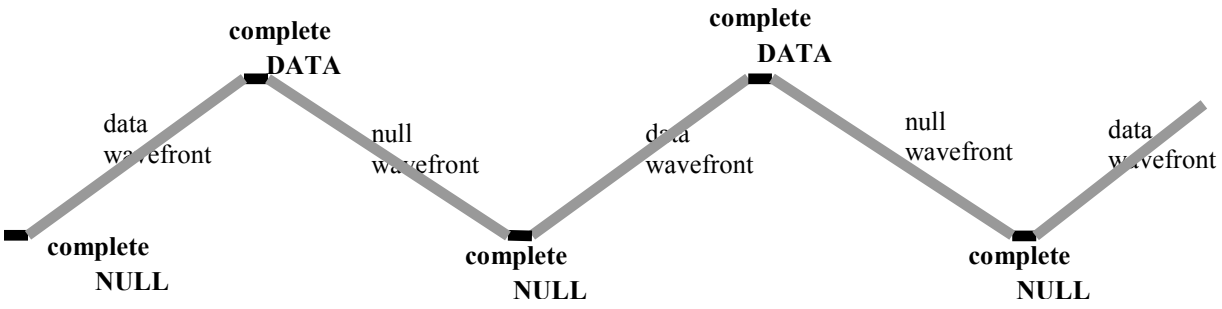


Figure 1: ***Successive wavefront transitions between different domains*** **[9]**

DATA WAVEFRONT

Consider combinational circuit as shown in Figure 2. Each number represents a Boolean gate.

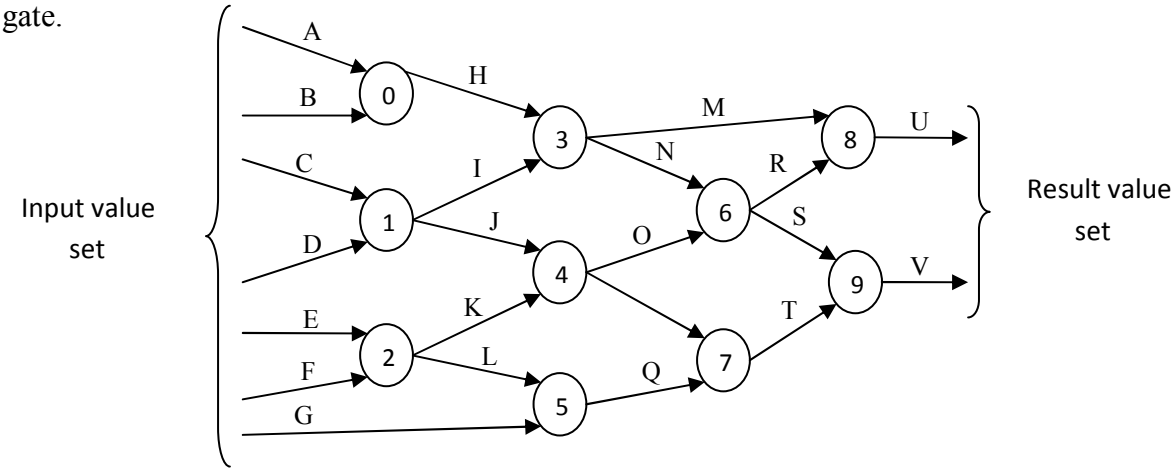


Figure 2*: Example Combinational Circuit* **[3] [9]**

Assume the circuit is in all Null state i.e., all input values, internal values and result values are all Null. If one input value, say A changes to data, gate 0 will still output NULL value since other input is still asserting null. For gate 0 to assert data both input A and B has to assert data. Similarly gate 8 will not assert data until M and R doesn't assert data values. Also for instance if all the input values are data except G, which remain NULL, therefore gate 5 will assert value Null for Q which intend make gate 7 assert value Null for T which make output of gate 9 assert value Null. When G becomes data,

all the inputs set will be data, which will propagate through the circuit and result data set will be complete.

When all the results values become data, it means that complete set of input values are presented to the circuit. This is what meant by completeness of input behavior of each gate which scales up to whole circuit. **The circuit as a whole forces the completeness of the input criteria for data.** To express completeness of the input criteria for data, the circuit has to start from all Null state and after each result data set it should return to all null state before next result data set. Referring to Fig. 2 after a result value set, input E become Null. Hence gate 2, 4, 5, 6, 7, 8 and 9 will assert NULL, which makes the result set values to NULL while there are still data values lingering on the input and internal to the circuit [3].

NULL WAVEFRONT

In *Second Step,* we make Boolean logic symbolically complete by enforcing the completeness of input criteria for NULL as done for DATA. This is achieved by either of the methods mentioned below.

INTERMEDIATE VALUE SOLUTION

In this method we add another value to the three value logic which provides a solution that is purely insensitive to delay and purely symbolically complete. Table 2 show truth table with added value called intermediate value.

	T	F	I	N
T	T	F	I	I
F	F	F	I	I
I	I	I	I	I
N	I	I	I	N

AND

	T	F	I	N
T	T	T	I	I
F	T	F	I	I
I	I	I	I	I
N	I	I	I	N

OR

T	F
F	T
I	I
N	N

NOT

Table 2: ***truth table with INTERMEDIATE value added*** **[3] [9]**

We now have a **four value logic.** As seen from the truth table, that a gate will only assert data value only when both the inputs are data and will only assert a NULL when both inputs are NULL. The result value set asserts a NULL only when all the inputs are NULL. Hence the circuit is completely reset and ready or next set of data. When the result value set asserts data, a complete data set is present at the input. Referring to Fig. 2 again we assume that all the result value set is date which implies all the inputs are data. If say input E goes to NULL then gate 2 will assert INTERMEDIATE, hence the subsequent gates will go into INTERMEDIATE state resulting in result value set to go INTERMEDIATE state until all the inputs goes to NULL state which propagates through the circuit making result value set NULL and ready for new next set of data. Now **the circuit as a whole enforces the completeness of input criteria for both data in relation to NULL and for NULL in relation to data.**

2.2 DERIVING 4 NCL TO 3 NCL

A four value NCL is practically not feasible to implement. Hence we can reduce the 4 value NCL to 3 value NCL. The INTERMEDIATE state explicitly indicates that the partial inputs are changed from the current state. Hence the output should hold to previous value. This means that the performance of INTERMEDIATE state is same as state hold behavior such as FEEDBACK.

4NCL	3NCL
TRUE	TRUE
FALSE	FALSE
INTERMEDIATE	state holding behavior (FEEDBACK)
NULL	NULL

Table 3*: Correspondences of 4NCL and 3NCL* **[9]**

FEEDBACK SOLUTION

Feedback solution makes each gate a state machine with hysteresis of result value assertion. Table 4 shows the truth table for a feedback gate.

R	A	B	RESULT R
N	N	N	N
N	N	T	N
N	N	F	N
N	T	N	N
N	T	T	T
N	T	F	F
N	F	N	N
N	F	T	F
N	F	F	F
F	N	N	N
F	X	X	F
T	N	N	N
T	X	X	T

AND

R	A	B	RESULT R
N	N	N	N
N	N	T	N
N	N	F	N
N	T	N	N
N	T	T	T
N	T	F	T
N	F	N	N
N	F	T	T
N	F	F	F
F	N	N	N
F	X	X	F
T	N	N	N
T	X	X	T

OR

Table 4*: truth table with FEEDBACK*

R is the result variable feed back to the input. When the gate is asserting NULL it will keep asserting NULL until both the inputs are data. At this point it will transition from NULL state to valid Data state. After result asserts data (R = T or F) it will keep asserting the same data until both the inputs value assert NULL at that point it will transition its result value to NULL and enter NULL state. Thus **the feedback gate enforces the completeness of input criteria for both data in relation to NULL and for NULL in relation to data.**

2.3 Two value NULL Conventional Logic

If we want to reduce to 2 NCL from 3 NCL we need either to remove TRUE, FALSE or NULL state. The state hold behavior can still be retained using feedback. If we decide to retain TRUE and FALSE the resulting logic will be Boolean logic. If we decide to differentiate using DATA and NON-DATA the NULL conventions can be retained. Hence the corresponding 2 NCL is shown in table below.

4NCL	3NCL	2NCL	Boolean Logic
TRUE	TRUE	DATA	TRUE
FALSE	FALSE		FALSE
INTERMEDIATE	state holding behavior	state holding behavior	
NULL	NULL	NULL	

Table 5: ***Correspondence of the logic*** **[9]**

As NCL has to be limited to two values, one value is assigned to express NULL and other to express DATA. Two data - true or false for instance, must be expressed with two wires as a dual rail signal. A dual rail signal D consist of 2 wires (D^0 and D^1) which may assume any value from (DATA0, DATA1 and NULL). DATA0 state ($D^0 = 1$, $D^1 = 0$) corresponds to digital logic 0. DATA1 state ($D^0 = 0$, $D^1 = 1$) corresponds to digital logic 1. NULL state ($D^0 = 0$, $D^1 = 0$) corresponds to empty state. The two wires are mutually exclusive so that both cannot assert simultaneously.

2.4 Threshold Gate

NCL is implemented using discrete threshold gates. This gate determines DATA of the output when the threshold numbers of wires are present with data at the input. As Figure

3 shows a 3 threshold / 5inputs. If any 3 or more inputs are present with DATA output will assert DATA.

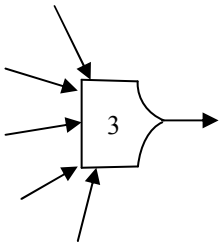
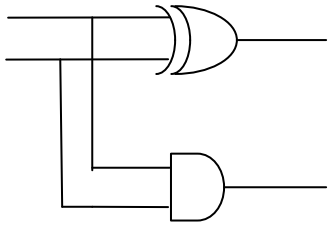


Figure 3: ***Threshold Gate***

The most effective way to understand the two values NCL formed with discrete threshold gate is using a combinational circuit. Fig. 4 shows a conventional Boolean logic half adder and Fig. 5 show a NULL conventional logic half adder circuit.



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 4: ***Boolean logic half adder***

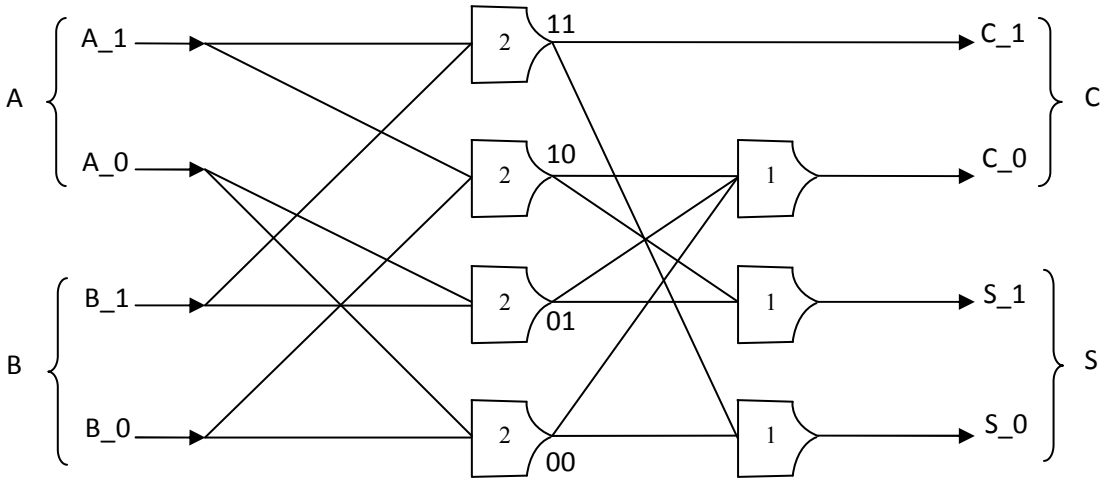


Figure 5: ***NULL Convention half adder*** **[3] [9]**

Boolean logic half adder expresses the 4 possible combinations for input data values with two data value (0, 1) on two wires (A, B). NULL conventional logic expresses the same 4 possible combinations for input data values with one data value (DATA) on four wires (A_1, A_0, B_1, B_0). As stated above, each data value (A, B, C and S) is expressed mutually exclusive into group of two wires (A_1, A_0 and B_1, B_0 and C_1, C_0 and S_1, S_0). Only one wire from each group can assert DATA at a time, so we get a complete set of input values. Threshold 2 gate complete the input criteria of 2 DATA values. Hence only one threshold 2 gate will assert DATA as long as mutually exclusive assertion is maintained for the input group. **The gates, as well as the circuit as a whole, enforce the completeness of input criteria for data.** Similarly to enforce completeness of input criteria for NULL we need feedback loop [9][3].

To provide the hysteresis behavior of the discrete threshold gate the result value is feedback with a weight of one less than the threshold as shown in Fig. 6. This gate is called *THmn gate*, $1 \leq m \leq n$ where 'm' corresponds to the threshold value of the gate and

'n' corresponds to number of input. Example TH34 is a 3 threshold with 4 inputs. Let inputs be A, B,C and D. Another type of threshold gate is referred to as a weighted threshold gate. This gate is called THmnW where W = w_1, w_2 … w_r and $m \geq w_r \geq 1$and $1 \leq R \leq n$. 'W' corresponds to weight of the input value. Example TH34w2 is a 3 threshold with 4 inputs. Let inputs be A, B, C and D. Weight of input A will be 2

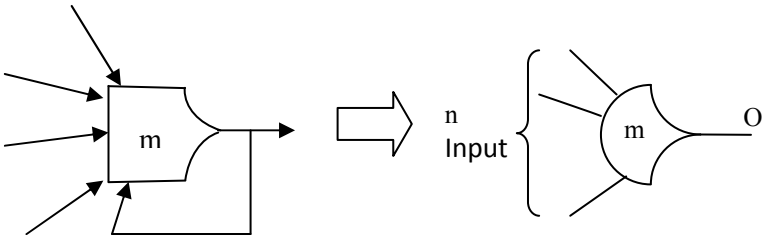


Figure 6: ***THmn Threshold Gate***

Table 6 lists the 27 transistor networks along with its Boolean equation [1]. This transistor network is use to construct NCL circuit. These transistor gates are macros with four or fewer variables. Since NCL has mutually exclusive input variable these four variable functions doesn't implies as four literals which would mean four dual rail signals.

NCL Macros	Boolean Equation	4 input LUT count
TH12	A + B	1
TH22	AB	1
TH13	A + B + C	1
TH23	AB + AC + BC	1
TH33	ABC	1
TH23w2	A + BC	1
TH33w2	AB + AC	1
TH14	A + B + C + D	2
TH24	AB + AC + AD + BC + BD + CD	2
TH34	ABC + ABD + ACD + BCD	2
TH44	ABCD	2
TH24w2	A + BC + BD + CD	2
TH34w2	AB + AC + AD + BCD	2
TH44w2	ABC + ABD + ACD	2
TH34w3	A + BCD	2
TH44w3	AB + AC + AD	2
TH24w22	A + B + CD	2
TH34w22	AB + AC + AD + BC + BD	2
TH44w22	AB + ACD + BCD	2
TH54w22	ABC + ABD	2
TH34w32	A + BC +BD	2
TH54w32	AB + ACD	2
TH44w322	AB + AC + AD + BC	2
TH54w322	AB + AC + BCD	2
THxor0	AB + CD	2
THand0	AB + BC + AD	2
TH24comp	AC + BC + AD + BD	2

Table 6: ***27 NCL Macros*** **[1]**

2.5 Basic Logical Operators Using NCL

The design process of any circuits in NULL conventional logic is implemented using *threshold combinational Reduction (TCR) method* [1]. In this method the equation of the circuit is expressed in Boolean logic. Then optimization criteria included in the design.

The NCL circuit is optimized using sum-of-product (SOP) expression and then mapped in to THmn gates.

OR FUNCTION

The 2 input OR function can be expressed as Z = X + Y. The canonical expression for Z^0 = X^0 Y^0 which can be directly mapped into TH22 gate from table 6. Then canonical expression for Z^1 = X^1Y^1 + X^0Y^1 + X^1Y^0 which can be directly mapped into THand0 gate. Without optimizations the OR function, X^1Y^1, X^0Y^1, X^0 Y^0 and X^1Y^0 each can be mapped into TH22 gate and output of X^1Y^1, X^0Y^1 and X^1Y^0 into TH13 gate. Hence total of four TH22 gate and one TH13 gate is required which would require five, 4 input LUT in an FPGA. But with TCR method one TH22 gate and one THand0 gate is required which requires three, 4 input LUTs [1][2].

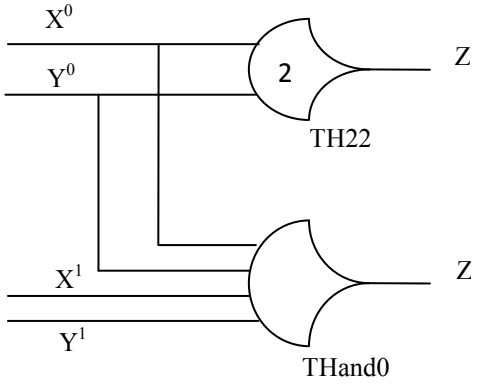


Figure 7: ***OR Function in NCL*** **[1]**

A^0	A^1	B^0	B^1	Z^0	Z^1
D	X	D	X	D	X
D	X	X	D	X	D
X	D	D	X	X	D
X	D	X	D	X	D

Table 7*: truth table – OR Function*

AND FUNCTION

The 2 input AND function can be expressed as Z = X.Y. The canonical expression for Z^0 = $X^0\ Y^0 + X^0Y^1 + X^1Y^0$ which can be directly mapped into THand0 gate from table 6. Then canonical expression for $Z^1 = X^1Y^1$ which can be directly mapped into TH22 gate. Without optimizations the AND function, X^1Y^1, X^0Y^1, $X^0\ Y^0$ and X^1Y^0 each can be mapped into TH22 gate and output of X^1Y^1, X^0Y^1 and X^1Y^0 into TH13 gate. Hence total of four TH22 gate and one TH13 gate is required which would require five, 4 input LUTs in an FPGA. But with TCR method one TH22 gate and one THand0 gate is required which requires three, 4 input LUTs [1][2].

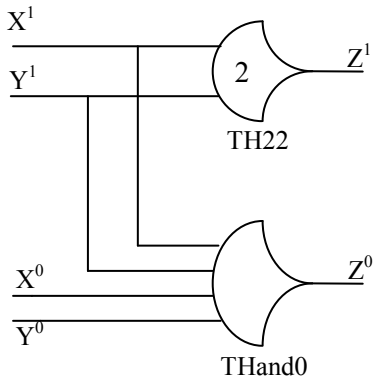


Figure 8: ***AND Function in NCL*** **[1]**

A^0	A^1	B^0	B^1	Z^0	Z^1
D	X	D	X	D	X
D	X	X	D	D	X
X	D	D	X	D	X
X	D	X	D	X	D

Table 8: ***Truth Table – AND Function***

XOR FUNCTION

The 2 input XOR function can be expressed as Z = X ⊕ Y. The canonical expression for Z^0 = $X^0\ Y^0 + X^1Y^1$ and for $Z^1 = X^0Y^1 + X^1Y^0$ which can be directly mapped into THxor0 gate or with addition of *don't care* terms like $X^0\ X^1$ and Y^0Y^1 in both Z^0 and Z^1 the new equation formed are $Z^0 = X^0\ Y^0 + X^1Y^1 + X^0\ X^1 + Y^0Y^1$ and $Z^1 = X^0\ Y^1 + X^1Y^0 + X^0\ X^1 + Y^0Y^1$ which can be directly mapped into TH24comp [1] [2]. Either way it is mapped it will require four 4 input LUTs.

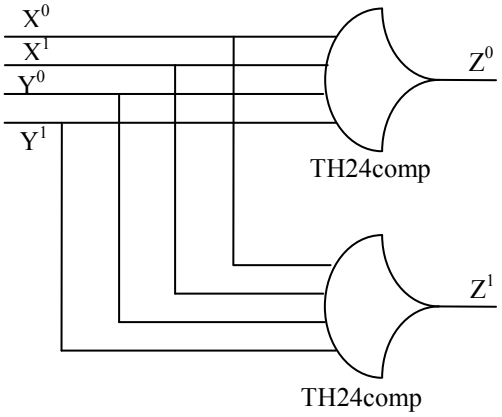


Figure 9: ***XOR Function in NCL [1]***

A^0	A^1	B^0	B^1	Z^0	Z^1
D	X	D	X	D	X
D	X	X	D	X	D
X	D	D	X	X	D
X	D	X	D	D	X

Table 9*: truth table - XOR Function*

2.6 FPGA – Virtex II Pro

Virtex II Pro families of FPGA are used as platform to design IP cores and customized modules. It endues complete solutions for designing telecommunication, wireless, DSP applications. It incorporates embedded POWER PC 405, multi-gigabit transceivers, block RAMs. The POWER PC 405 CU is a RISC processor. It has an on chip cache to reduce design complexity and improve system throughput. It has thirty two 32 bit general purpose registers (GPRs). The Virtex II Pro processor block is mainly divided into four blocks, Embedded *POWER PC 405 RISC CPU core, on chip memory controller (OCM), clock and control interface and CPU – FPGA interface*. The main core block operates at 300+ MHz while it maintains low power consumption. The OCM controllers work as an interface between block RAMs and FPGA. The clock and control interface block provides proper interface between clock/power management, reset, PLB cycle control, and OCM interface. The CPU – FPGA interface provide access between core processor block and general FPGA. It contains Processor Local Bus (PLB) interface. It is a high speed access bus between data and instruction cache. PLB slave interface are available to

user for soft IP cores implementation on FPGA. The Device Control Register (DCR) bus interface in CPU – FPGA block is a 10 bit address use for managing status and configuration registers, and helps in reducing traffic on PLB and improves system performance. The External interrupt controller (EIC) interface in Virtex-II pro provide critical and non critical interface. This interrupt can be controlled by user or soft interrupt controller IP core outside the processor block can be used. It has a JTAG and trace ports for debugging. The core architecture of power PC 405 provides three namely processor local bus (PLB), on chip peripheral bus (OPB) and device control bus (DCR) for interconnecting Processor Blocks, Xilinx soft IP, third party IP and custom logic. In my design, I have used OPB clock to clock my design. Also I have used eight one bit GPRs to control port of my design.

3. Implementing FFT using NULL CONVENTIONAL LOGIC

Fast Fourier Transform (FFT) is an efficient algorithm to implement Discrete Fourier Transform (DFT). It is calculated by decomposing the N point time domain signal into N different time domain signal with each point. Then this signal is converted into N frequency domain spectra which are later converted into single frequency domain.

3.1Radix 2 Butterfly Architecture

To implement FFT there are various algorithms been developed but the most used algorithm is the Radix 2 decimation-in-time (DIT) where the transform length is positive integer power of 2. By using this algorithm a time domain signal (x) is converted to frequency domain signal (X) using processing node. This node is called Butterfly. Each butterfly consist of 2 point DFT which are complex number say (a,b), one of the numbers (say b) is multiplied by $\boldsymbol{W_N^K}$ which is a complex number and then it is added and subtracted with other complex number to obtain 2 new complex number (say A,B). Therefore each butterfly involve one complex multiplication, one complex addition and a complex subtraction. Complex multiplier is designed as a scalar multiplier which has 4 multipliers [5][4].

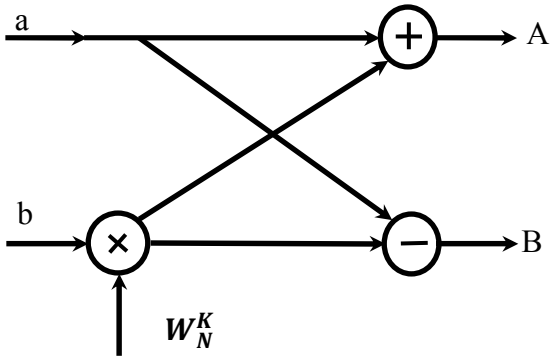


Figure 10: ***FFT Butterfly***

W_N^K $= e^{-j\,2\pi K/N}$ where K = no. of butterfly and N = no of point in FFT, which is in polar form. In rectangular form can be obtained by solving $e^{-j\,2\Pi K/N}$ as $e^{j\theta} = \cos\theta + j\sin\theta$.

Assume W_N^K as Wr + j Wi in rectangular form

Solving butterfly node we get,

$$\mathbb{R}[A] = \mathbb{R}[a] + \mathbb{R}[W]\,\mathbb{R}[b] - \mathbb{I}[W]\,\mathbb{I}[b]$$

$$\mathbb{I}[A] = \mathbb{I}[a] + \mathbb{R}[W]\,\mathbb{I}[b] + \mathbb{R}[b]\,\mathbb{I}[W]$$

$$\mathbb{R}[B] = \mathbb{R}[a] - \mathbb{R}[W]\,\mathbb{R}[b] + \mathbb{I}[W]\,\mathbb{I}[b]$$

$$\mathbb{I}[B] = \mathbb{I}[a] - \mathbb{R}[W]\,\mathbb{I}[b] - \mathbb{R}[b]\,\mathbb{I}[W]$$

To implement a butterfly node in NCL each input and output will have a dual rail. It is shown as follows

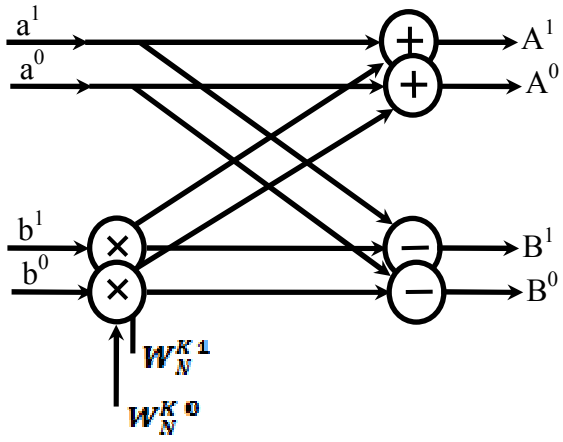


Figure 11*: butterfly node in NCL*

Solving butterfly node we get,

$$\mathbb{R}[A^1] = \mathbb{R}[a^1] + \mathbb{R}[W^1]\,\mathbb{R}[b^1] - \mathbb{I}[W^1]\,\mathbb{I}[b^1]$$

$$\mathbb{I}[A^1] = \mathbb{I}[a^1] + \mathbb{R}[W^1]\,\mathbb{I}[b^1] + \mathbb{R}[b^1]\,\mathbb{I}[W^1]$$

$$\mathbb{R}[B^1] = \mathbb{R}[a^1] - \mathbb{R}[W^1]\,\mathbb{R}[b^1] + \mathbb{I}[W^1]\,\mathbb{I}[b^1]$$

$$\mathbb{I}[B^1] = \mathbb{I}[a^1] - \mathbb{R}[W^1]\,\mathbb{I}[b^1] - \mathbb{R}[b^1]\,\mathbb{I}[W^1]$$

$$\mathbb{R}[A^0] = \mathbb{R}[a^0] + \mathbb{R}[W^0]\ \mathbb{R}[b^0] - \mathbb{I}[W^0]\ \mathbb{I}[b^0]$$

$$\mathbb{I}[A^0] = \mathbb{I}[a^0] + \mathbb{R}[W^0]\ \mathbb{I}[b^0] + \mathbb{R}[b^0]\ \mathbb{I}[W^0]$$

$$\mathbb{R}[B^1] = \mathbb{R}[a^0] - \mathbb{R}[W^0]\ \mathbb{R}[b^0] + \mathbb{I}[W^0]\ \mathbb{I}[b^0]$$

$$\mathbb{I}[B^0] = \mathbb{I}[a^0] - \mathbb{R}[W^0]\ \mathbb{I}[\ b^0] - \mathbb{R}[b^0]\ \mathbb{I}[W^0]$$

For an N point FFT there will be $\log_2 N$ level of computation and each level of computation requires N/2 butterfly processing nodes. Therefore for 16 point FFT we get 4 levels of computations and each level will have 8 butterfly processors node.

3.2 HALF ADDER IMPLEMENTATION IN NCL

The Half Adder is implemented using the NCL gates and it is optimize using TCR method. C^0 assert output data when either A^0 or B^0 assert data which can be implemented using TH12 gate. C^1 assert output data only when A^1 and B^1 assert data. S^1 assert output data when C^0 assert data with either A^1 assert data input value or B^1 assert data input value but not both as same time. Hence we can implement S^1 using TH33w2 gate where C^0 will be input with more weight than others. S^0 asserts output data when C^1 assert data or A^0 and B^0 assert data inputs. This can be implemented using TH23w2 gate where C^1 will be input with more weight than others [1][2].

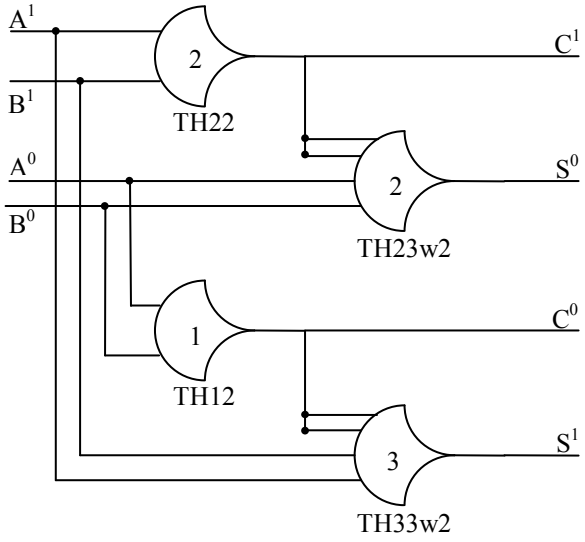


Figure 12: ***Optimized HALF ADDER in NCL*** **[1]**

A0	A1	B0	B1	S0	S1	C0	C1
D	X	D	X	D	X	D	X
D	X	X	D	X	D	D	X
X	D	D	X	X	D	D	X
X	D	X	D	D	X	X	D

Table 10: ***Truth table of HALF ADDER in NCL***

3.3 FULL ADDER IMPLEMENTATION IN NCL

From the truth table optimize equation for $C_o^1 = A^1B^1 + C_i^1A^1 + C_i^1B^1$ and $C_o^0 = A^0B^0 + C_i^0A^0 + C_i^0B^0$. Both this functions can directly be mapped in to TH23 gate. The K – map for Sum output S is based on A, B and C_i along with output from C_o. $S^0 = C_o^1A^0 + C_o^1B^0 + C_o^1\ C_i^0 + A^0B^0C_i^0$ and $S^1 = C_o^0A^1 + C_o^0B^1 + C_o^1\ C_i^0 + A^1B^1\ C_i^0$, both of which can be directly mapped into TH34w2 gates where C_o has more weight than the other input [1][2].

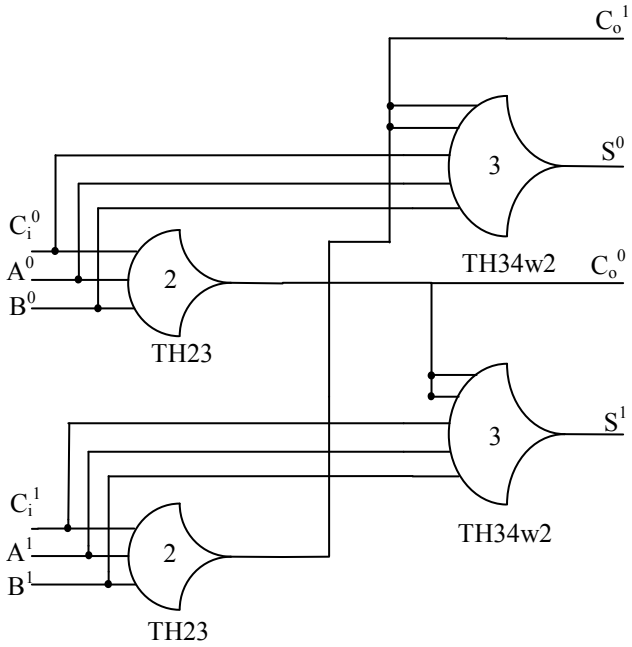


Figure 13: ***Optimized FULL ADDER in NCL*** **[1]**

A^0	A^1	B^0	B^1	C_I^0	C_I^1	S^0	S^1	C_o^0	C_o^1
D	X	D	X	D	X	D	X	D	X
D	X	D	X	X	D	X	D	D	X
D	X	X	D	D	X	X	D	D	X
D	X	X	D	X	D	D	X	X	D
X	D	D	X	D	X	X	D	D	X
X	D	D	X	X	D	D	X	X	D
X	D	X	D	D	X	D	X	X	D
X	D	X	D	X	D	X	D	X	D

Table 11: ***Truth table of FULL ADDER in NCL***

3.4TOP MODULE

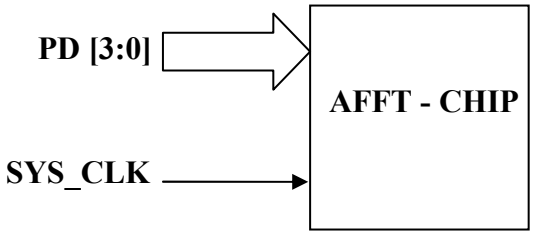


Figure 14: ***Top Module***

PIN	No of Bits	Type of Pins	Operation
PD	4	Input	Data from ADC
SYS_CLK	1	Input	System Clock from the crystal oscillator @105 Mhz

Table 12: ***top module pin out***

The lower 4 bits or the ADC output is connected to the input of the design. The output of the design is displayed on hyper terminal.

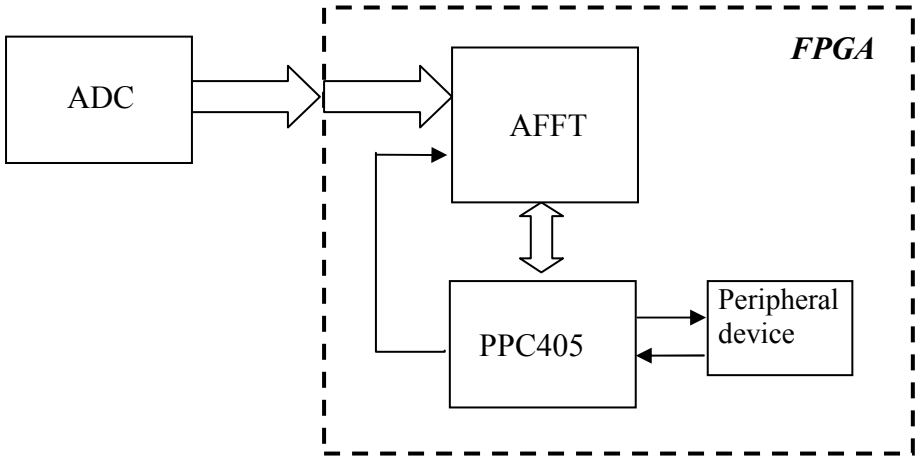


Figure 15: ***Internal Block of the top module***

AFFT BLOCK

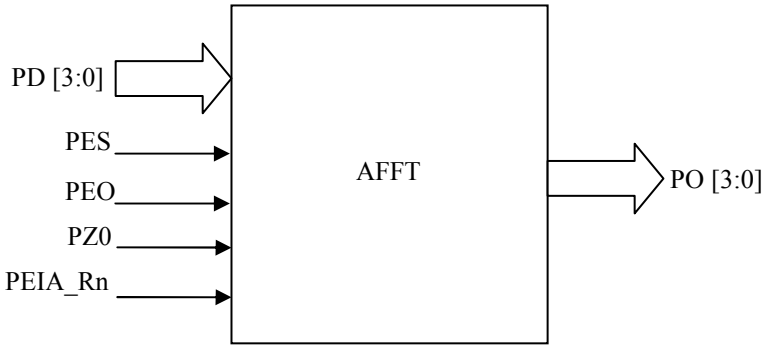


Figure 16: ***Asynchronous FFT block***

PIN	No of Bits	Type of Pins	Operation
PD	4	Input	Data from ADC
PES	1	Input	Enable Input 1 - Data flows from input 0 - No data is taken from the ADC
PEO	1	Input	Enable Output 1 - processed data from AFFT 0 - data shifted from input buffer
PZ0	1	Input	Provide NULL data to the AFFT
PEIA_Rn	1	Input	when 1 - Input DATA (true or false) on data lines 0 - Input NULL on data lines
PO	4	Output	Output data

Table 13: ***AFFT block pin out***

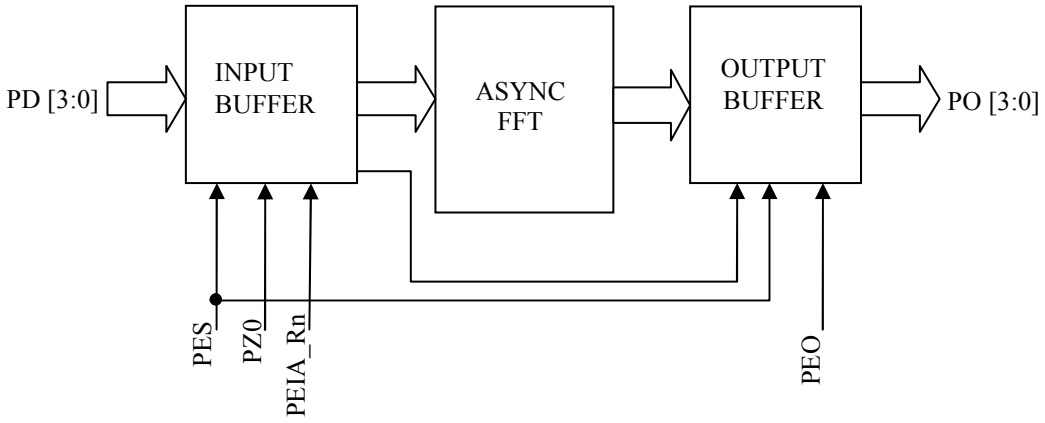


Figure 17: ***Internal module of AFFT block***

AFFT block is a 16 point Asynchronous FFT implemented using NCL gates. PD input is attached to the ADC output. PZ0, PEIA_Rn, PES, PEO and PO are provided by the software controller register from the PPC 405. The butterfly node which is used to implement the asynchronous FFT is constructed using NCL based Full Adders, Half Adders, XOR gates and OR gate. The input and output buffers block are designed to store, read and write inputs and outputs to and from Asynchronous FFT block. Also the inputs from the input buffer can be serially shifted to the output buffer to verify the outputs.

INPUT BUFFER

Input buffer is design as shown below. In this block when ES = '1' new data from the ADC. When ES = '0' previous data is feedback from the register. When EIA_Rn = '1' DATA is sent and when EIA_Rn = '0' NULL (ZERO) is sent over the dual rail. The output Z from the register also is serially shifted to output buffer.

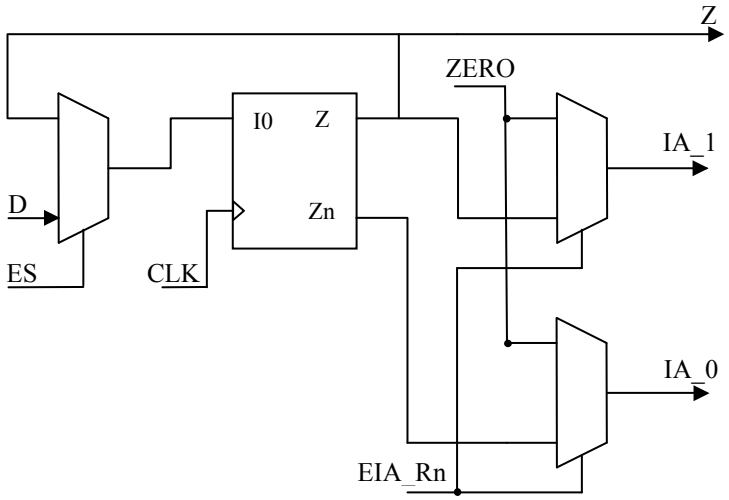


Figure 18: ***Input Buffer***

OUTPUT BUFFER

Output buffer is design as shown below. In this block when ES = '1' data from the input buffer is shifted serial in to output buffer and when ES = '0' and EO = '1' output from the asynchronous FFT block is shifted out. When ES = '0' and EO = '1' feedback loop is created.

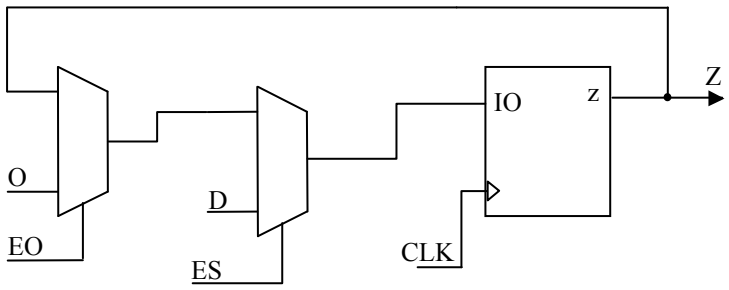


Figure 19: ***Output Buffer***

4. Testing and Result

Testing in general can be defined as a process which identifies how well something works. In hardware or software, testing is used as checkpoints to determine whether the objective of the design is met. The design mentioned in the above chapters is developed and tested adhering to the Xilinx ISE design flow. The tools primarily used are the Xilinx ISE, Modelsim, Xilinx SDK and NC- Sim for simulations, Synthesis and implementation. Figure 20 show the design flow implemented. This shows testing at various stages of the design. Initially behavioral testing was done when design entry or coding part is completed, then the post place and route simulation.

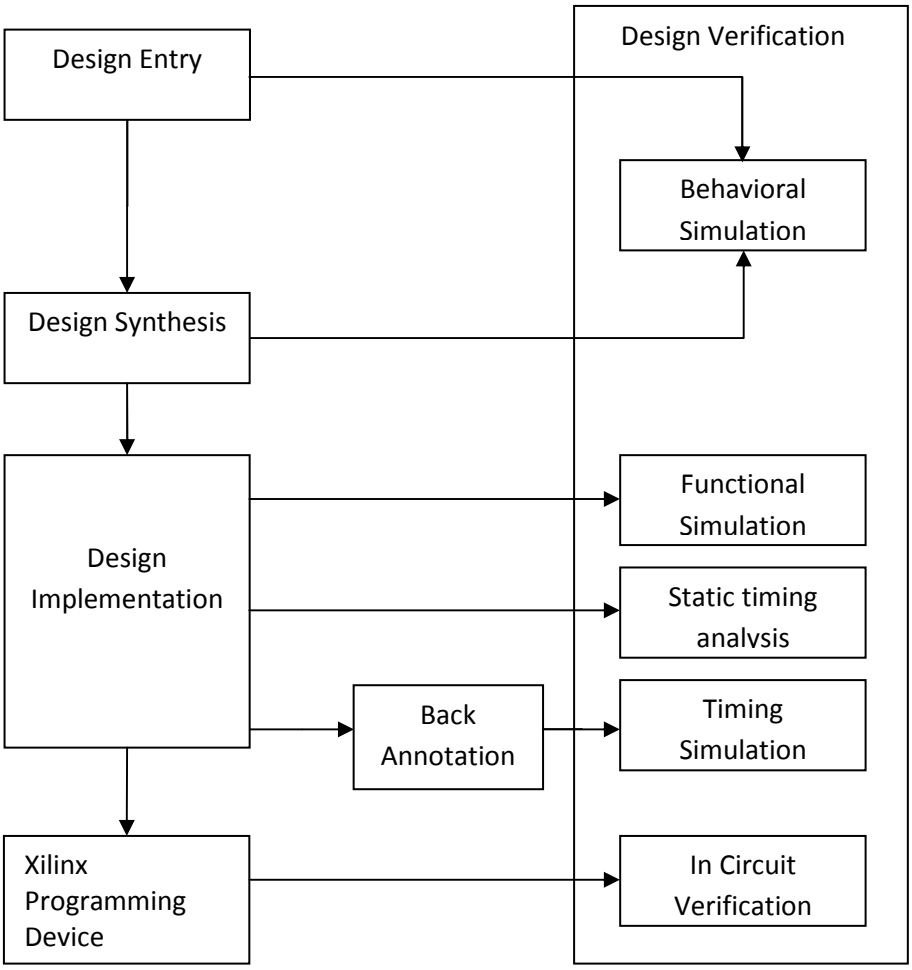


Figure 20: ***FPGA Design Flow***

Each function of 27 NCL macros shown in table 6 is hard wired in to 4 input LUT's. Depending on the number of inputs along with feedback the function are mapped into 1 or 2 LUT's with MUX F5. After implementing each function different combination of DATA and NULL is provided as test inputs and corresponding outputs are verified.

Below is the output trace of gate TH24w2. In this threshold gate there are 4 inputs namely A, B, C and D. The threshold value for this gate is 2 and input A has weight 2.

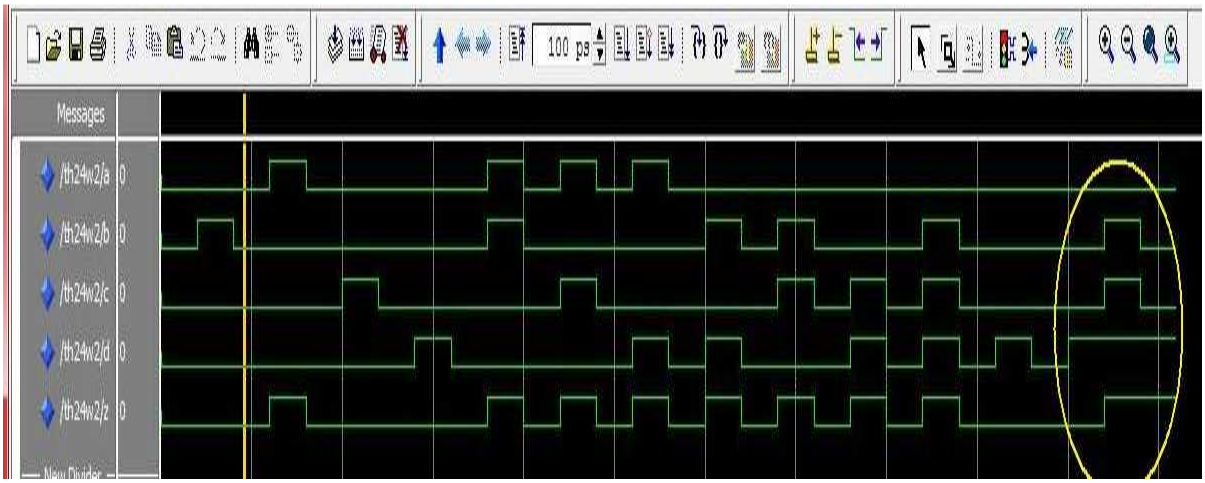


Figure 21: ***Output waveform of TH24w2 threshold gate***

As seen from the figure 21 initially when any one input from A, B, C and D is DATA output is NULL except for input A. As soon as input A goes DATA output goes DATA since it has weight 2. When there is other 2 inputs say BC, BD or CD containing DATA output is DATA. In the highlighted portion initially the output is NULL. Output goes to DATA when 2 inputs are DATA. In the next cycle when there is only one input say D is DATA, output should go to NULL but it remains DATA due to feedback loop. Output would have gone NULL if the previous cycle had all input as NULL. Synthesis of TH24w2 threshold gate is shown in Figure 22. The function is mapped in to two 4 inputs LUT and MUX – F5. As we can see in the figure the output Z is feedback to I0 line of the input.

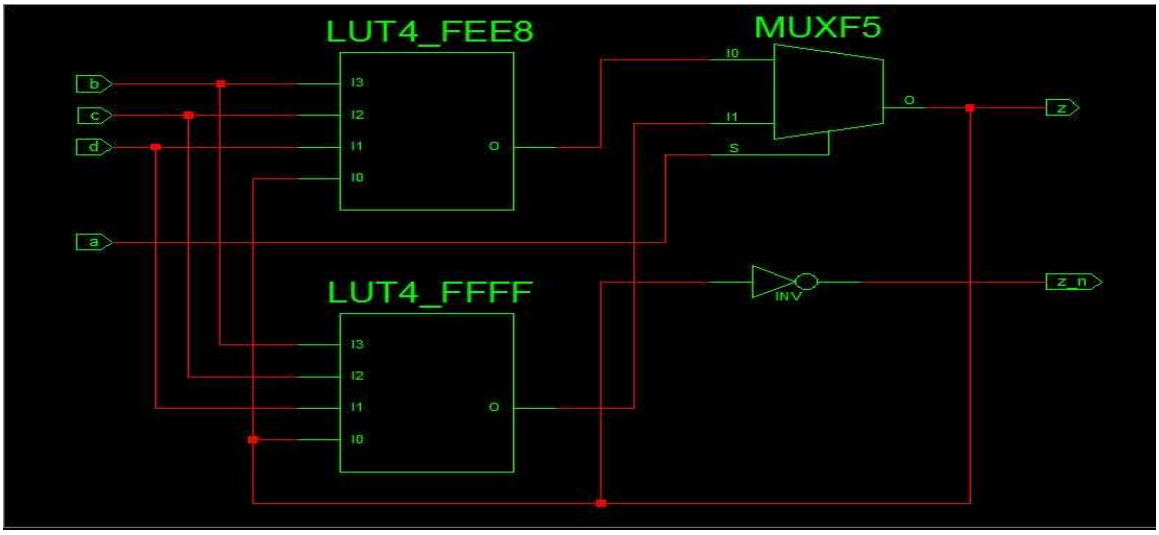


Figure 22: RTL Schematic of TH24w2

Similarly we can simulate remaining NCL threshold gates and the basic boolean gates (XOR, AND and OR) built using the NCL gates.

After simulating all the NCL macros we simulated half adder and full adder. The test bench was generated with all the possible combination of inputs and corresponding outputs were verified. Simulated output of half adder and full adder is as shown in the figure 23 and figure 25 respectively. After simulation it was synthesized and RTL schematic is as shown in the figure 24 and figure 26

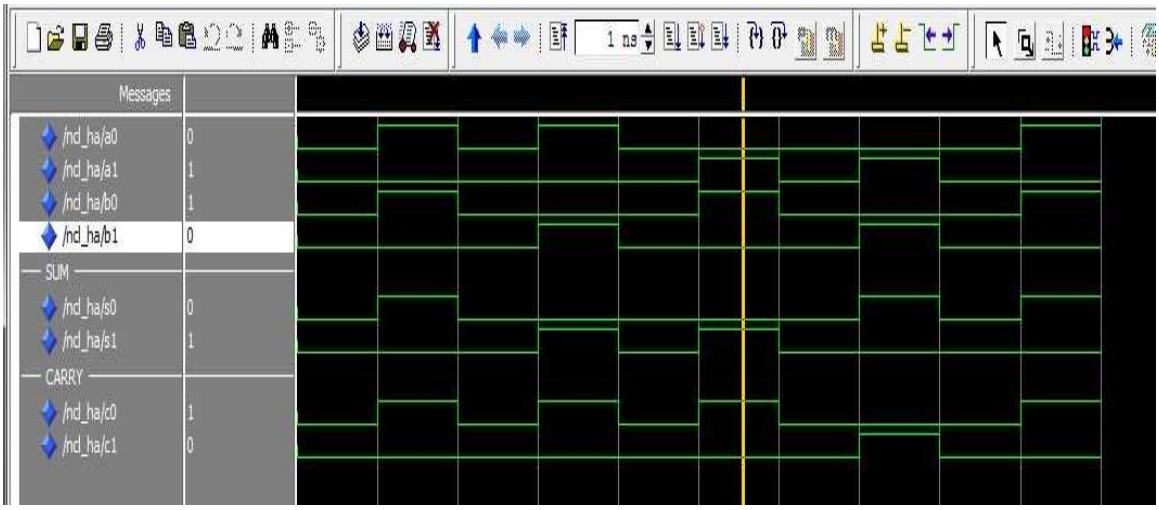


Figure 23: ***Output waveform of Half Adder***

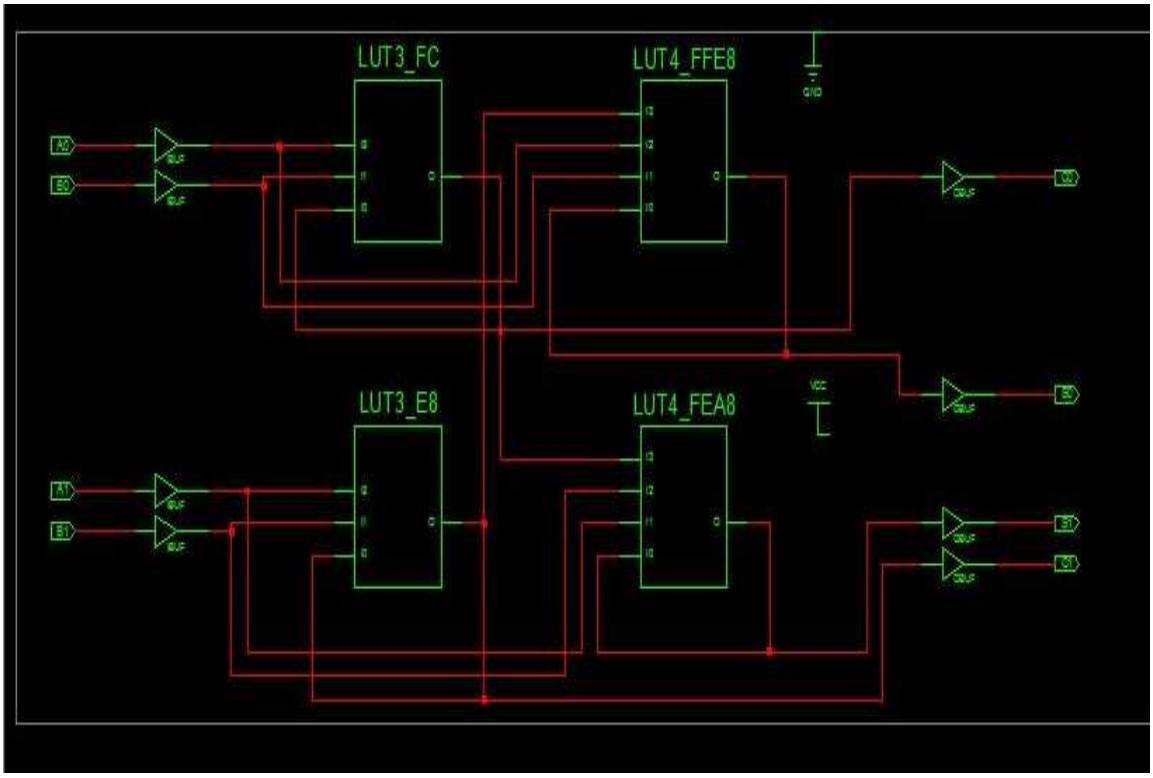


Figure 24: ***RTL Schematic of Half Adder***

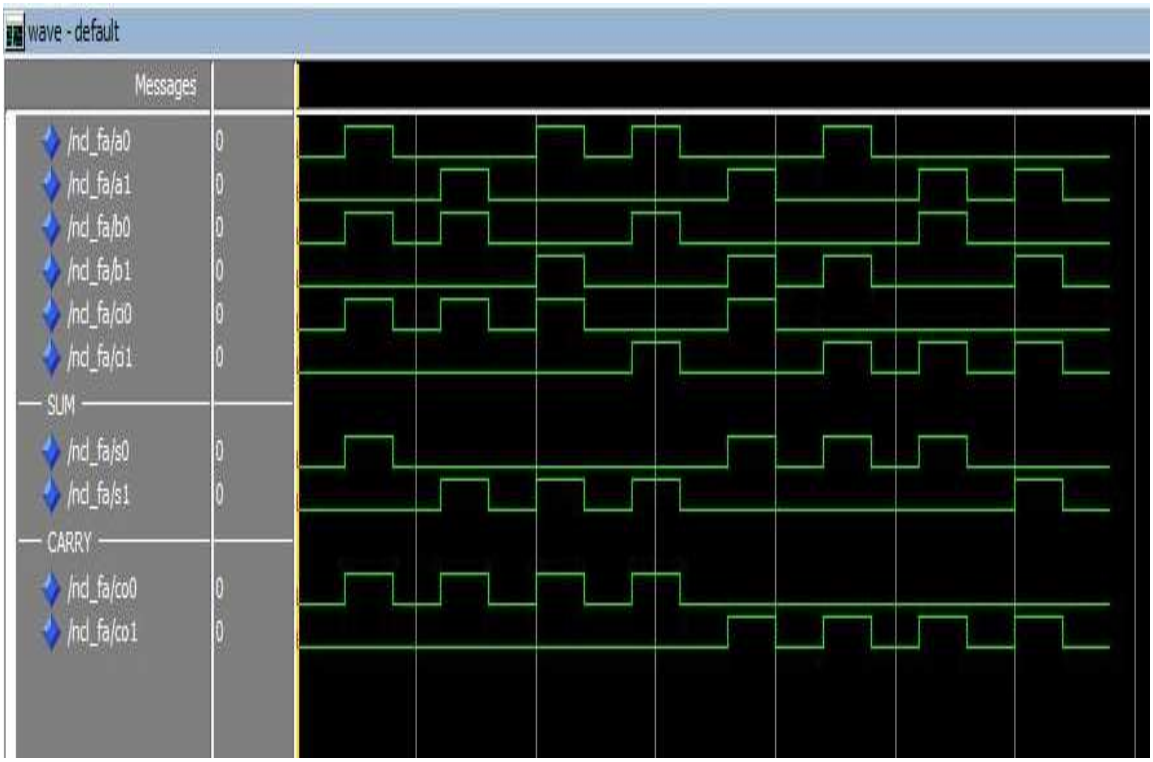


Figure 25: ***Output waveform of Full Adder***

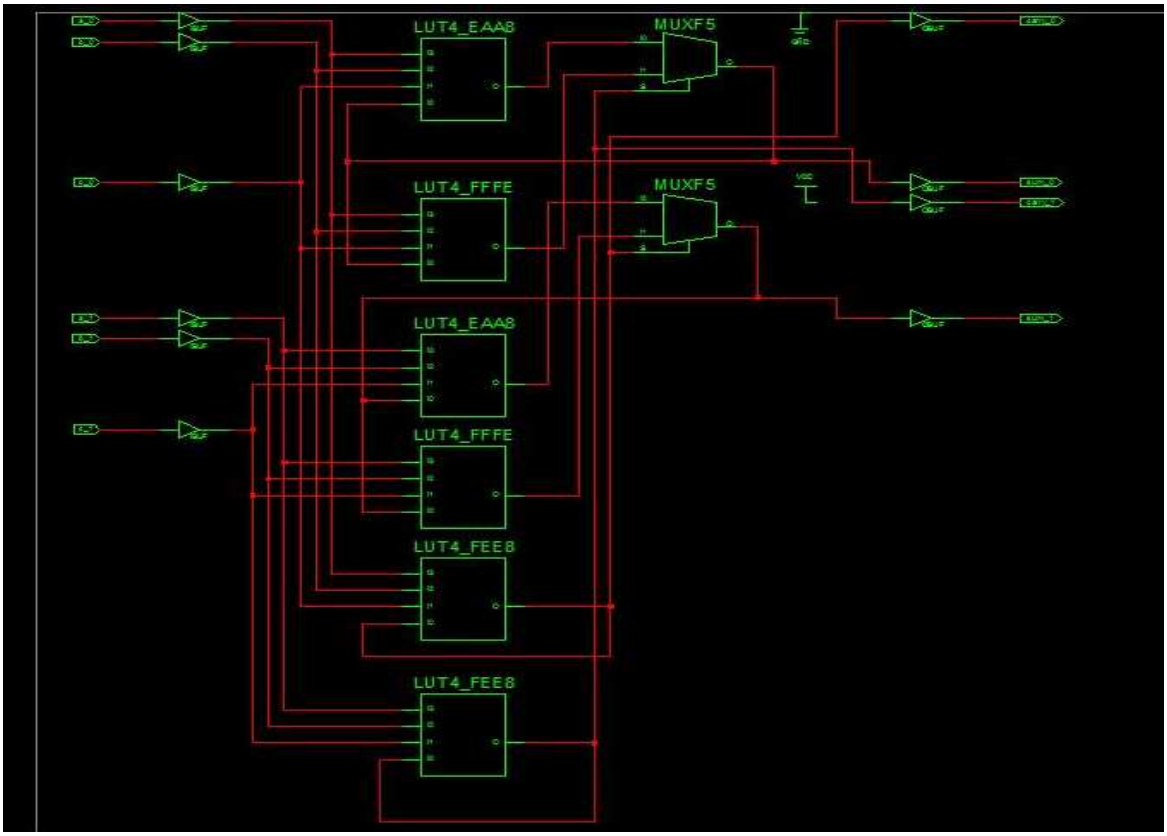
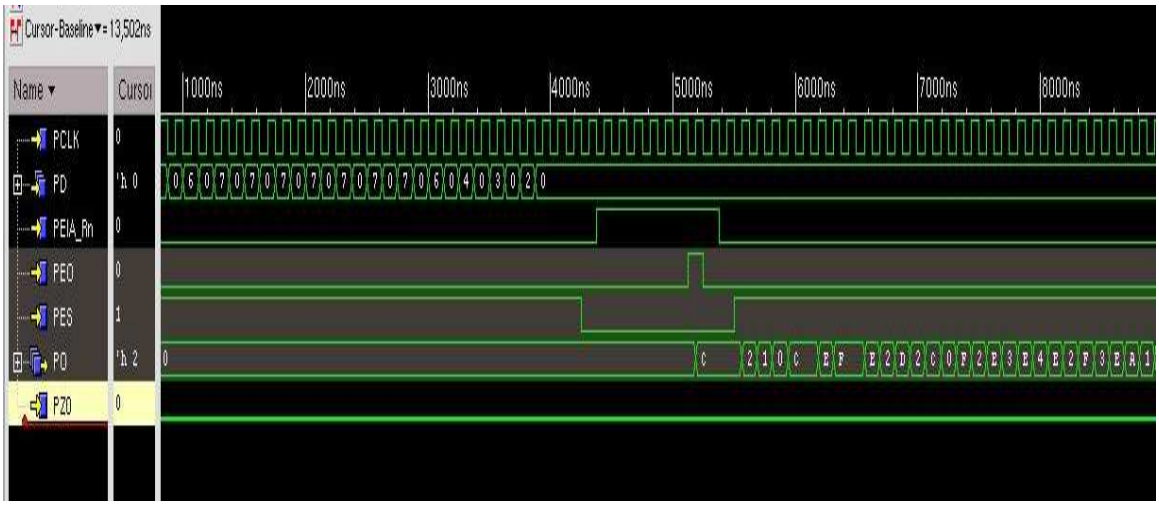


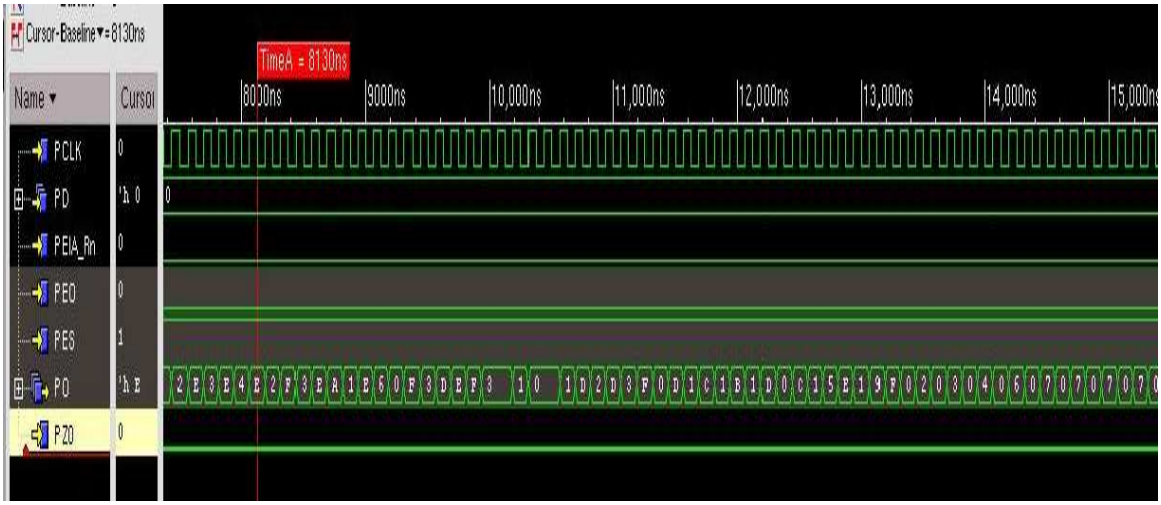
Figure 26: ***RTL Schematic of Full Adder***

Testing sequence for 16 point FFT was done in the following order:

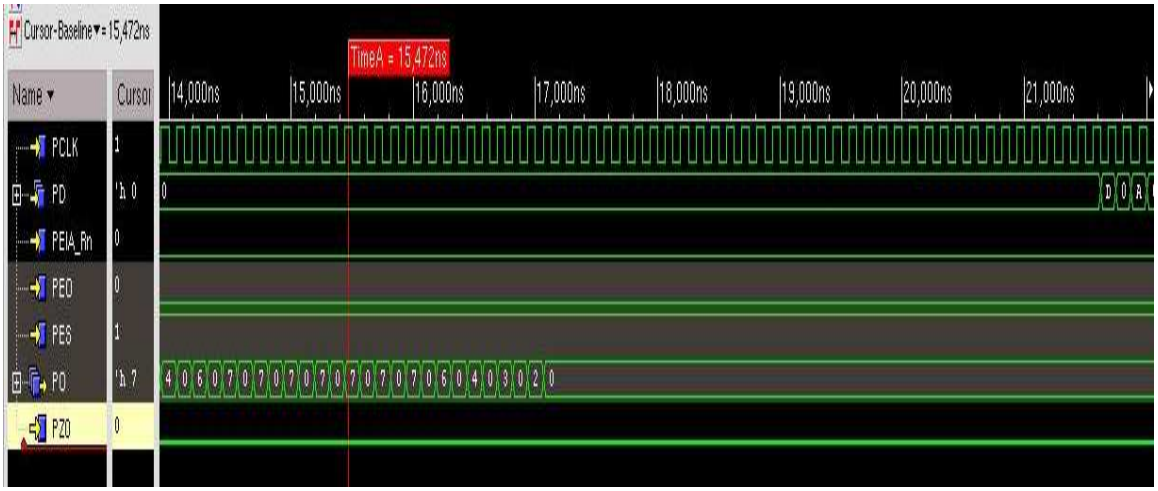
- After few clock PES is enabled. Input from the ADC is taken IN. Data is stored in the input buffer. And PZ0 is always ZERO. Hence NULL is send as input to the Asynchronous FFT
- Later PEIA_Rn is enabled and PES is disabled. In this condition data is taken as input to the Asynchronous FFT.
- When PEO is enabled and PES is disabled, output from the asynchronous FFT Is stored into the output buffer.
- After the output from the FFT is stored PES is enabled to flush out the data stored from the input buffer into the output buffer.
- It was synthesized and RTL schematic is shown in figure 27



(a)



(b)



(c)

Figure 27: ***Output simulation of a 16 point Asynchronous FFT***

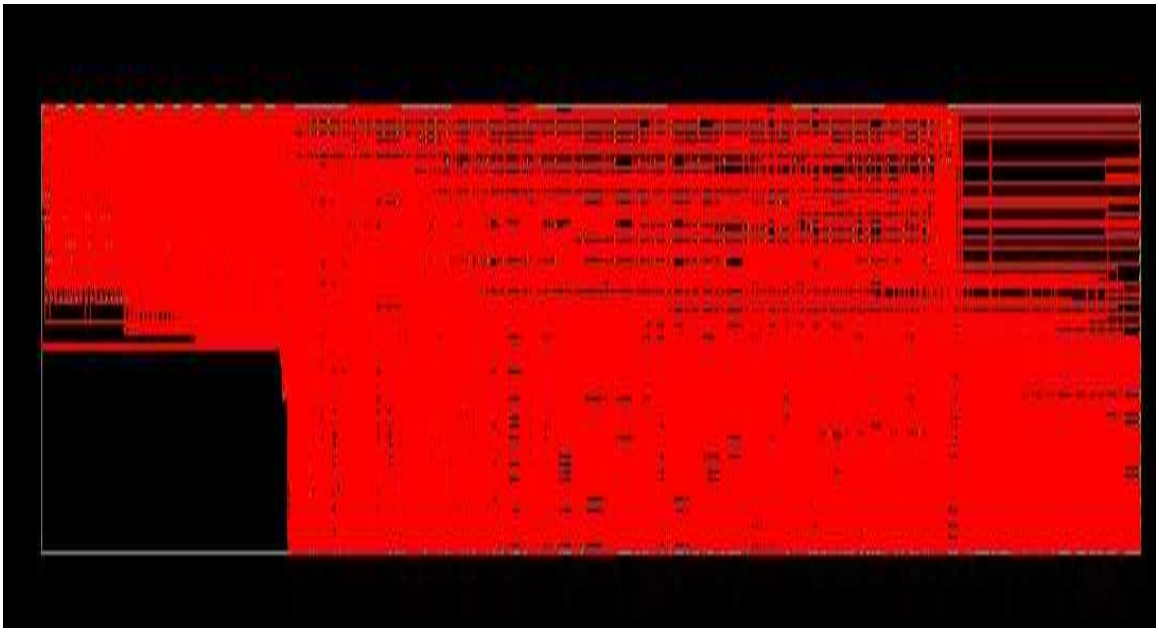


Figure 28: ***RTL Schematic of 16 point Asynchronous FFT***

The complete system implemented using Xilinx Platform Studio.

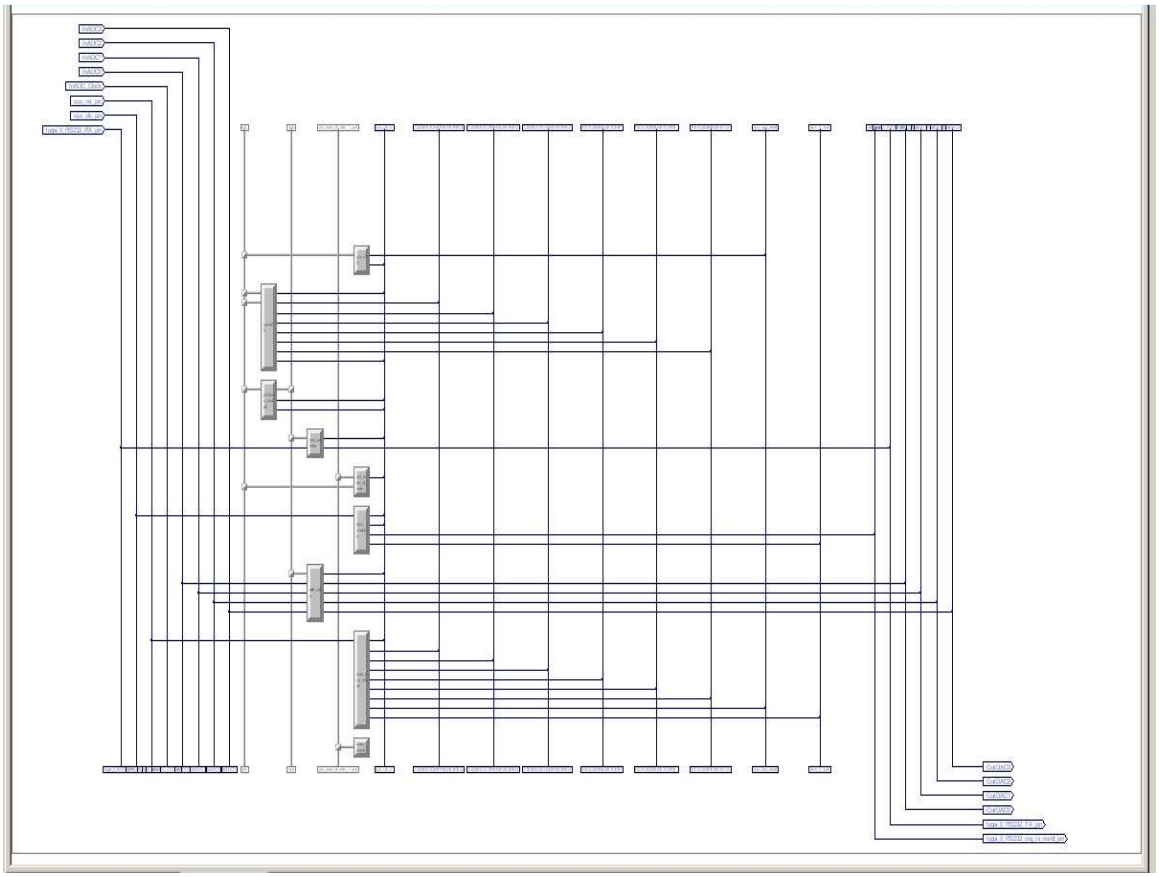


Figure 29: ***system view***

5. Conclusion & Future Work

From this report it can be concluded that asynchronous design can be implemented using NCL. The current commercial available FGPA can be used for implementation any asynchronous design.

Future Work

Future Work for this includes working on developing architecture for Asynchronous FPGA. This architecture will have feedback path. The LUT's and registers will be based on NCL threshold gates. Also with the architecture the simulation and synthesis should also be enhances, so while simulation it would be robust to handle feedback path and it can optimize the asynchronous logic. Also libraries can be developed to implement behavioral style for coding.

6. References

[1] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson and D. Lamb, "Optimization of NULL convention self-timed circuits," vol. 37, pp. 135, 2004.

[2] S. C. Smith. (JUNE 2007, Design of an FPGA logic element for implementing asynchronous NULL convention logic circuits. *15(6),* pp. 672.

[3] K. M. Fant and S. A. Brandt, "NULL convention Logic: A complete and Consistent Logic for Asynchronous Digital Circuits Synthesis." pp. 261, 1996.

[4] A. V. Oppenheim and R. W. Schafer, *Discrete Time Signal Processing.* Englewood Cliffs, New Jersey 07632: Prentice Hall, pp. 879.

[5] S. A. White, "A Simple FFT Butterfly Arithmetic Unit," vol. 28, April 1981.

[6] K. V. Berkel, "Beware of Isochronic fork," vol. 13, June 1992.

[7] J. S. Yuan, "Teaching Asynchronous Design in Digital Integrated Circuits," vol. 47, pp. 397, August 2004.

[8] J. Sparsø and S. Furber. (December 2001, *Principles of Asynchronous Circuit Design - A Systems Perspective.* Available: http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=855

[9] K. M. Fant, Logically Determined Design: Clockless System Design with NULL Convention Logic. ,1st ed.Wiley-Interscience, 2005, pp. 292.