Parallel Addition for Double Base Number System

A Project Report

submitted by

Pravin Kumar Singh (11EC71)

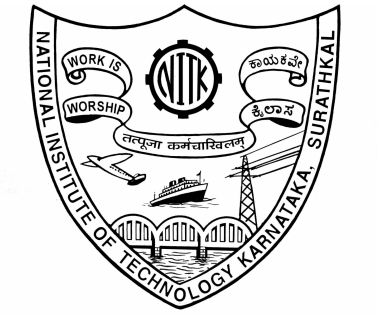
Ankit Shah (11EC86)

under the guidance of

Dr M R Arulalan

in partial fulfilment of the requirements for the course

DSP SYSTEMS AND ARCHITECTURE



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE - 575025

November 8, 2014

ABSTRACT

Double base number system (DBNS) is an alternative number system besides the binary system. Its representation is similar to the radix number system together with two bases, usually be two and three. DBNS preserves the two important properties: redundancy and sparseness. The redundancy is the property accommodating with the parallelism. In this research, we are interested in parallel addition algorithm on DBNS. Our theoretical result shows that parallel addition in DBNS can be performed. An addition algorithm together with the proof of correctness is described in this paper. In generally, we study the generalization form of DBNS addition algorithms in any sizes.

TABLE OF CONTENTS

ABSTRACT **i**

1 Introduction **1**

2 DBNS Representation **2**

2.1 Binary to DBNS Conversion 3

2.2 Algorithm: GREEDY algorithm to convert integers into DBNS 4

Example 4

2.3 Parallel Addition on DBNS in General Form 5

3 A Pseudo Code for FPGA Implementation **7**

4 Simulation Results **8**

4.1 Block Diagram Map 10

4.2 Power Analysis 11

5 Results **12**

6 Conclusions **13**

LIST OF FIGURES

2.1 *DBNS Maps for 15* 2

2.2 *RALUT Structure* 3

2.3 *The n-square and its carry propagation flow* 5

4.1 *Simulation Results for DBNS conversion from a binary number* 8

4.2 *Simulation Results for TOP module for addition of two numbers* 9

4.3 *Block level schematic* 10

4.4 *Power analysis using XPOWER analyser tool* 11

CHAPTER 1
Introduction

Arithmetic computation plays an important role in arithmetic and logic unit (ALU) part of central processing unit (CPU). Binary system is classical used for number representations in ALU in order to do arithmetic operations. This system is simple but hard to perform any parallel arithmetic operations. However, there are many proposed number systems which can speed up the computational time. The most efficiency technique is to apply a signed-digit concept to the representation. The system had first proposed by Avizienis .

Classical technique used for fast computing (especially for parallel computing) is a redundancy property. Redundancy means that a number can have more than one representation in the system. This technique is proved to be able to reduce the carry propagation chain, which can speed up the time complexity of the operations. Using the concept of redundancy, double-base number representation system (DBNS) designed by V.S. Dimitrov, G.A. Jullien and W.C. Miller in 1997 has been proposed in order to reduce the number of 1s digits in the representation. Any number in double-base representation is illustrated by two bases: two and three. Moreover, a number can also have more than one representation. Fundamental arithmetic operations such as addition, subtraction, multiplication, and division can be performed in DBNS.

We introduce a parallel algorithm for DBNS addition operation. We consider the representation of number as a two-dimensional table with respect to the bases two and three. Each table (representation) can be considered as the combination of several square sub-tables. Our technique is to propose an algorithm for performing addition of all sub-tables in the same time. Theoretical result shows that addition can be solved in the constant time complexity.

CHAPTER 2

DBNS Representation

The double base number system is a redundant system that has been first proposed by V.S. Dimitrov, G.A. Jullien and W.C. Miller . The following definition describes the DBNS representation system.

Definition : *Double base number representation system (DBNS) is a number representation system using the exponential form of bases 2 and 3 with digits in* $D = \{0, 1\}$. *For an arbitrary integer m, it can be represented in this system by*

$$m = \sum d_{i,j} 2^i 3^j$$

such that $d_{i,j} \in \{0, 1\}$ *with integers i and j.* It is obviously that any integer can have a representation in this system; any binary representation is also of the form of DBNS by setting j to zero. Since a representation contains two bases, the number is usually illustrated by a table as shown in Example.

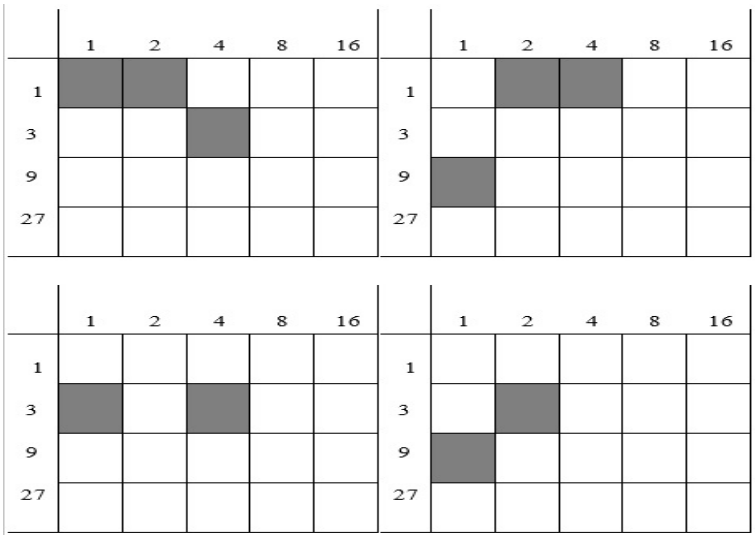


Figure 2.1: *DBNS Maps for 15*

2.1 Binary to DBNS Conversion

We are using modified greedy algorithm to convert binary number into DBNS. This process uses a pseudo floating point look-up table (M bits for the mantissa of the i t3 component), a barrel shifter (for the i b2 component and the floating-point exponent from the table) and a sign corrector to generate the binary 2s complement representation for each digit. Multiple digits are simply accumulated to generate the final binary representation. Reversing of the process is relatively straight forward; the barrel shifter becomes a normalizer, and the look-up table (1 +T rows) becomes another look-up (M 2 rows); the sign corrector will not be used as the inputs for large-bit word purposes will always be positive integers. The new table, now indexed by the mantissa, contains the same output data as the original along with the second base exponent t . This reversal process has serious scalability problems which causes the new table to be exponentially larger than the original. To solve this problem a new memory device was created, known as a Range Addressable Look-Up Table (RALUT). The RALUT is capable of matching ranges of numbers to particular outputs. Physically it is a generic memory device with a modified address decoder mechanism which is an optimized set of parallel comparators with some additional logic to determine the proper memory line to activate.

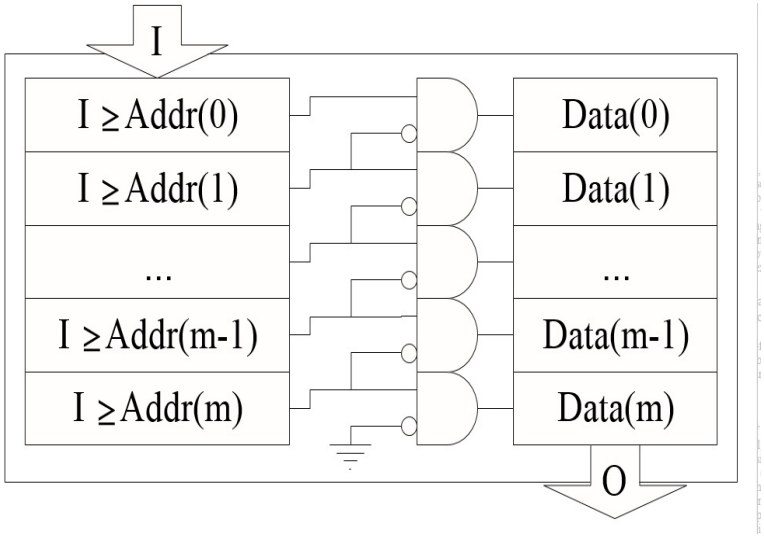


Figure 2.2: *RALUT Structure*

2.2 Algorithm: GREEDY algorithm to convert integers into DBNS

Input: a positive integer x

Output: the sequence of exponents (b_n, t_n) leading to one DBNS representation of x.

1 while $x > 0$ do
2 Find $z = 2^b.3^t$, the largest DBNS representation less than or equal to x
3 Save b and t
4 $x = x - z$

Example

1. $remain = bits - 1 = 4$
2. Normalize x =10111 such that bit 4 (5th bit) is 1; no change, $remain = remain - shift = 4$
3. Lookup 10111 in table such that $remain \geq Exponent$; returns $Mantissa = 10010, Exponent = 011, t = 10$
4. First digit is$2^{remain-Exponent}3^t = 2^1 3^2 = 18$
5. Subtract mantissa from input,

$$x = 10111 - 10010 = 00101$$

6. Normalize x = 00101 such that bit 4 (5th bit) is 1; x =10100 , $remain = remain - shift = 2$
7. Lookup 10100 in table such that remain Exponent ; returns $Mantissa = 10000, Exponent = 000, t = 00$
8. Second digit is $2^{remain-Exponent}3^t = 2^2 3^0 = 4$
9. Subtract mantissa from input,

$$x = 10100 - 10000 = 00100$$

10. Normalize x = 00100 such that bit 4 (5th bit) is 1; x =10000 ,$remain = remain - shift = 0$
11. Lookup 10000 in table such that $remain \geq Exponent$; returns $Mantissa = 10000, Exponent = 000, t = 00$
12. Third digit is $2^{remain-Exponent}3^t = 2^0 3^0 = 1$
13. Subtract *mantissa* from input,

$$x = 10000 - 10000 = 00000$$

14. *remain* is 0, conversion complete with 3 digit DBNS representation $(1, 2), (2, 0), (0, 0)$

2.3 Parallel Addition on DBNS in General Form

In this section, we consider a technique for performing addition in parallel manner. Since DBNS satisfies the redundancy property which causes the proposed parallel algorithm. Our technique is to group a square number of cells together and evaluate the numerical value of each square. Then, calculate the carry-out from the proposed equations and bring the carry-out of the above square to be the carry-in for every n-square simultaneously. Consequently, we get the full Result table representation by combining every n-square together. We show in this section how parallel addition can be done. The following theorem demonstrates that addition can be operated in constant time.

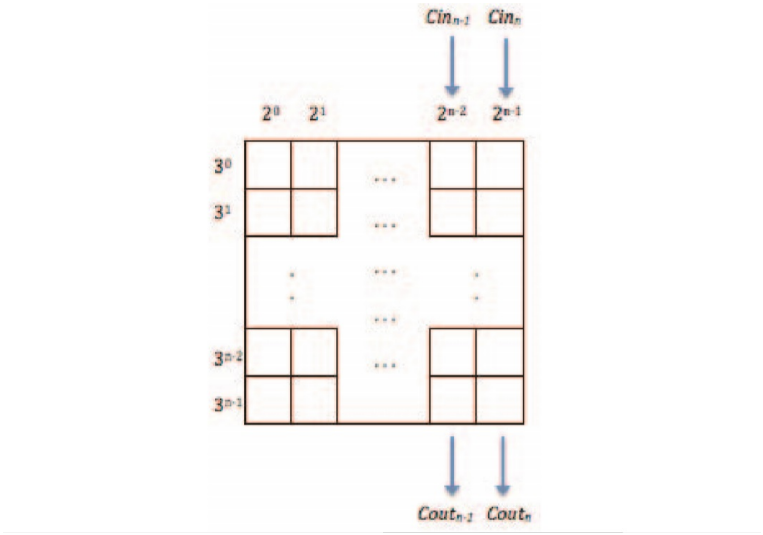


Figure 2.3: *The n-square and its carry propagation flow*

Algorithm: Parallel Addition

Input: $A = \sum_{i=0,j=0}^{cn-1} a_{i,j}2^i3^j$ and

$B = \sum_{i=0,j=0}^{cn-1} b_{i,j}2^i3^j$ and $a_{i,j}b_{i,j} \in \{0,1\}$

Output: $Z = \sum_{i=0,j=0}^{cn-1} z_{i,j}2^i3^j$ and $z_{i,j} \in \{0,1\}$

The inputs and the output are in table representation form. Let i, j be the coordinate of each cell in the original tables and k, l is the coordinate of each n-square related with the full table.

Begin

Step1 Perform addition of A and B, cell-by-cell, in parallel.

$$x_{i,j} = a_{i,j} + b_{i,j}$$

Step2 Evaluate the carry-out numerical value of each n- square simultaneously.

$$Sum_{k,l} = \sum_{i=0,j=0}^{cn-1} x_{i,j} 2^i 3^j$$

$$Cout_{n_{k,l}} = \lfloor \frac{sum_{k,l}}{2^{n-1}3^n} \rfloor$$

$$Cout_{n-1_{k,l}} = \lfloor \frac{sum_{k,l}}{2^{n-1}3^n} \rfloor - 2Cout_{n_{k,l}}$$

Step3 Take all of $Couts$ become $Cins$ of the below n-square simultaneously related with their own k, l coordinate. Then, map the numerical value of each $Result_{k,l}$ to be in table representation form(n-square). At last, we get the last Result table from combining every n-square.

$$Result_{k,l} = Sum_{k,l} + Cin_{k,l} - Cout_{k,l}$$

where $Cout_{k,l} = 2^{n-1}3^n Cout_{n-1_{k,l}} + 2^{n-1}3^n Cout_{n_{k,l}}$

End

CHAPTER 3

A Pseudo Code for FPGA Implementation

1 Take two decimal input

2 Use modified greedy algorithm to convert decimal number into DBNS.It consist of a modified look up table which has very less no of comparison.

3 Save the output in a register of 4 bits.

4 We store the DBNS value in different register of 4 bits which is useful in dividing the 2 dimensional DBNS into groups of 2x2 matrix.

5 We are using a look up table to compute the sum of 2x2 matrix in the first two rows simultaneously and compute C_{in} for the next two rows.

6 Output of the two decimal input is shown in the DBNS.

7 Computations further if done using the DBNS representation would save the time for conversion back to decimal and carrying out the same computation

CHAPTER 4

Simulation Results

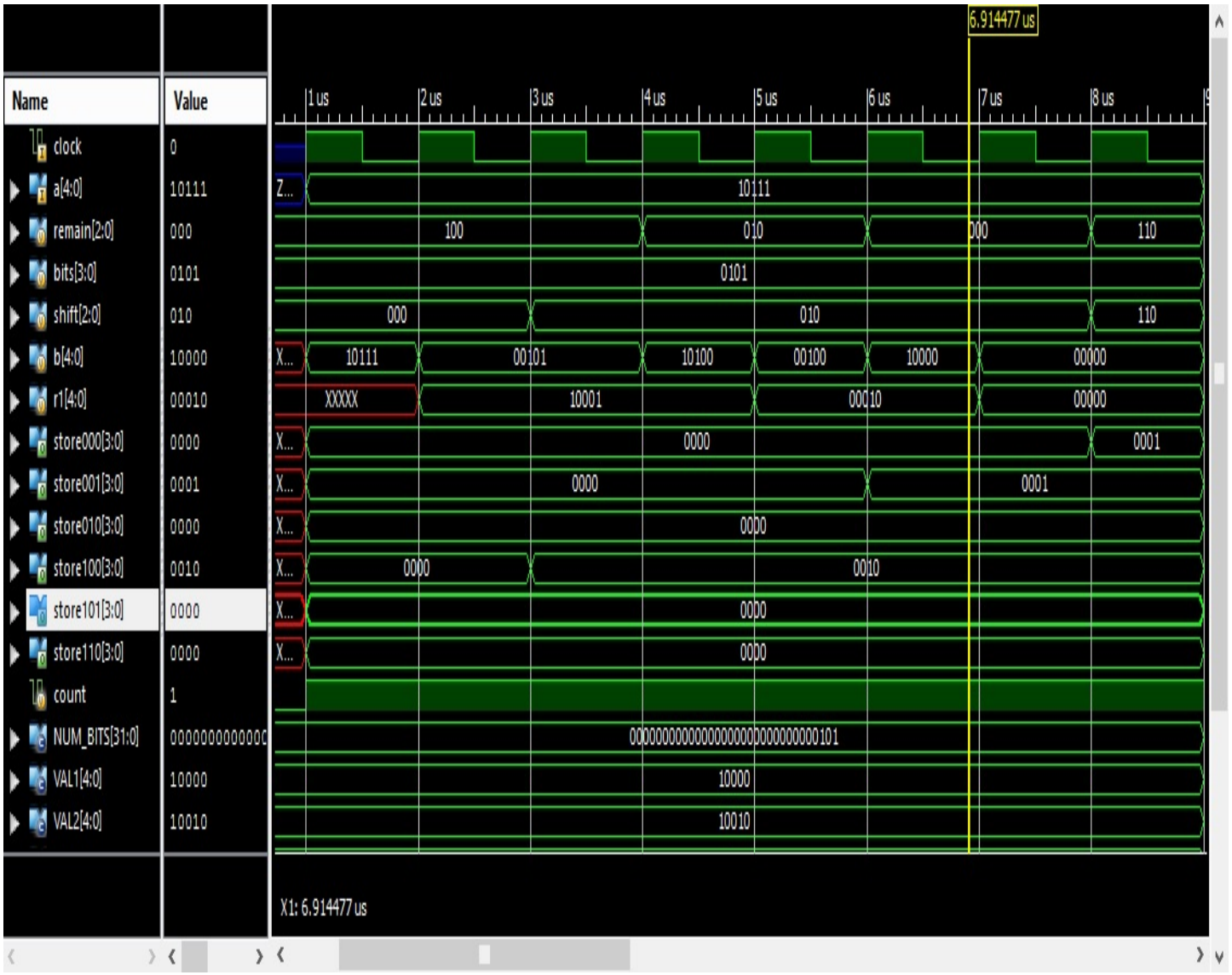


Figure 4.1: *Simulation Results for DBNS conversion from a binary number*

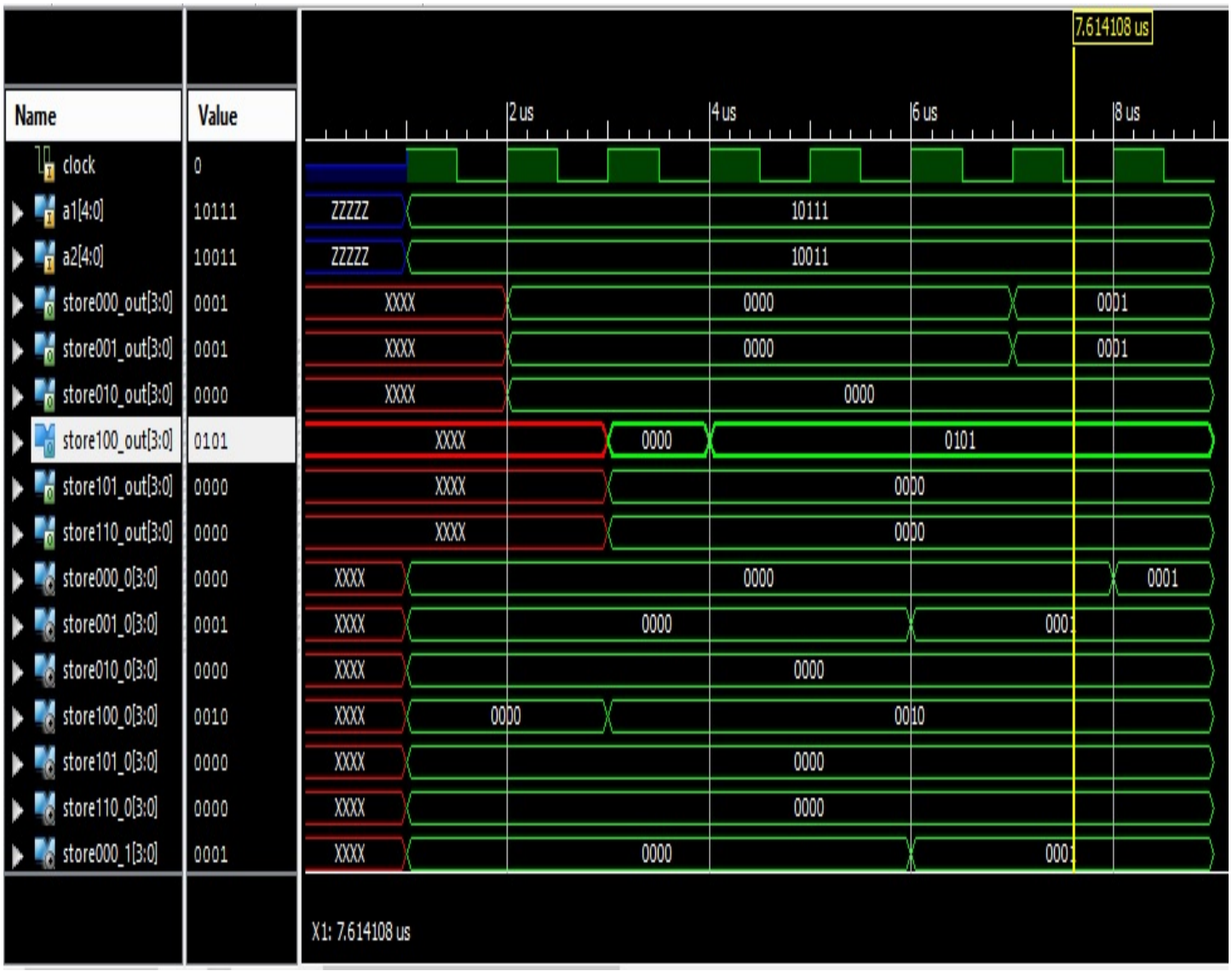


Figure 4.2: *Simulation Results for TOP module for addition of two numbers*

4.1 Block Diagram Map

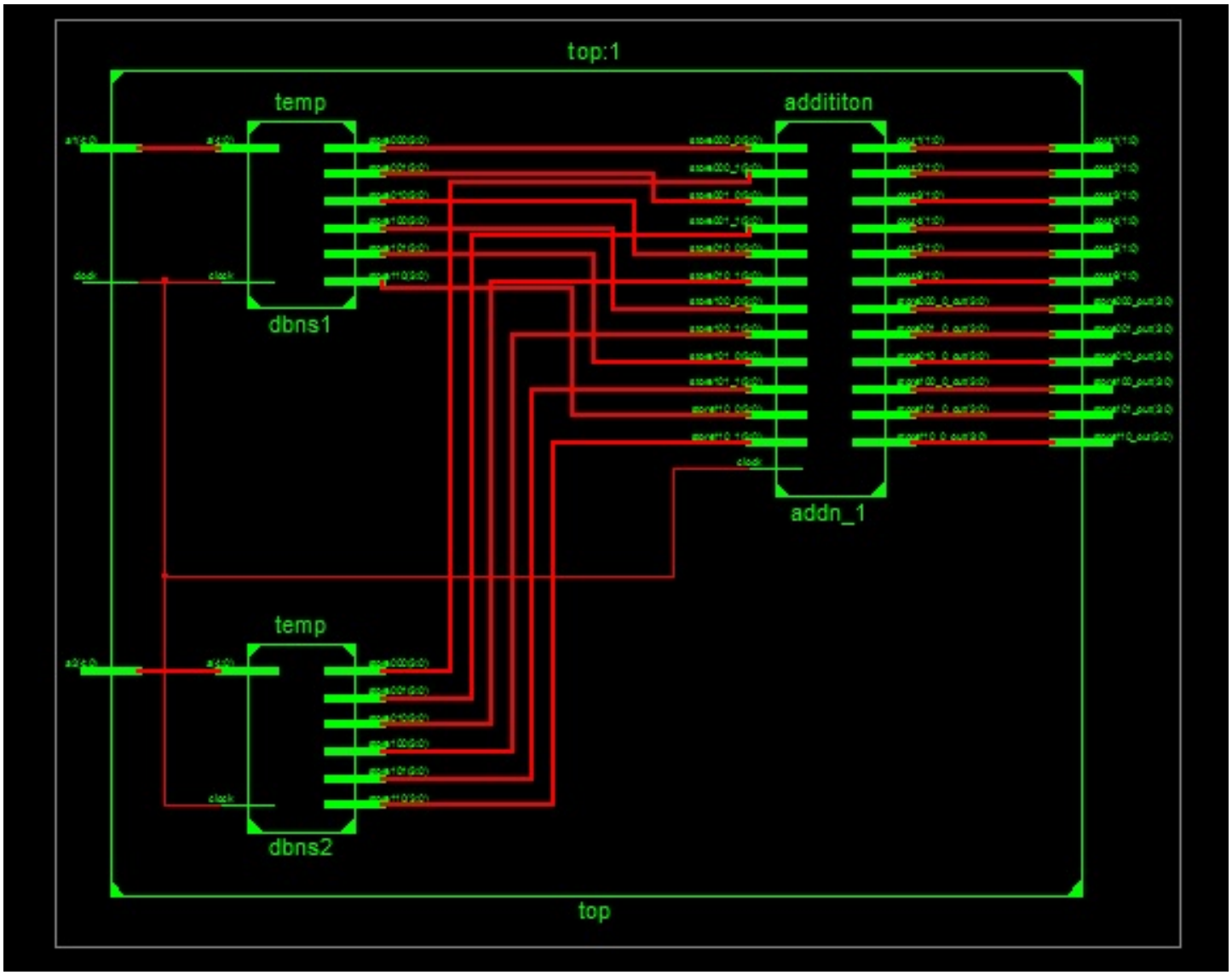


Figure 4.3: *Block level schematic*

4.2 Power Analysis

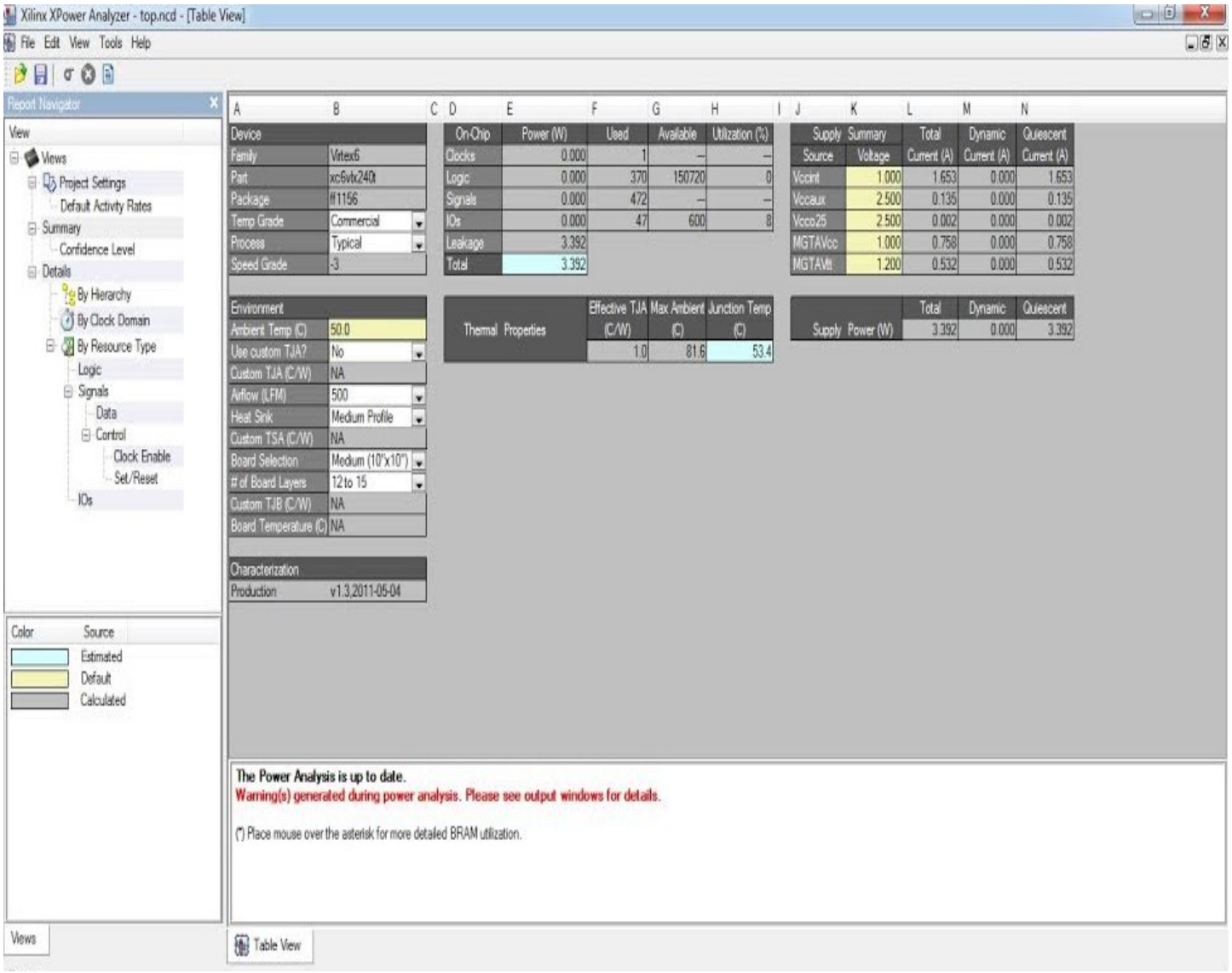


Figure 4.4: *Power analysis using XPOWER analyser tool*

CHAPTER 5

Results

```
Selected Device : 6vlx240tff1156-3


Slice Logic Utilization:
 Number of Slice Registers:             100  out of  301440     0%
 Number of Slice LUTs:                  455  out of  150720     0%
    Number used as Logic:               455  out of  150720     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    460
   Number with an unused Flip Flop:     360  out of    460    78%
   Number with an unused LUT:             5  out of    460     1%
   Number of fully used LUT-FF pairs:    95  out of    460    20%
   Number of unique control sets:        22

IO Utilization:
 Number of IOs:                         47
 Number of bonded IOBs:                 47  out of    600     7%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               1  out of     32     3%

Timing Summary:
---------------------------------------------------
Speed Grade: -3

   Minimum period: 2.784ns (Maximum Frequency: 359.155MHz)
   Minimum input arrival time before clock: 0.650ns
   Maximum output required time after clock: 0.669ns
   Maximum combinational path delay: No path found
```

CHAPTER 6

Conclusions

We implemented an addition algorithm for double base number system in parallel manner. Additions of several smaller n-square representations are able to be performed at the same time. We demonstrated that addition in DBNS can be realized with a constant-time complexity algorithm. Our approach is also suitable for implementing an ALU. We found that this concept can be used for developing other fundamental arithmetic operations in DBNS in the future.

The parallel addition algorithm can be used for improving the speed of operation in DBNS based representation and thus the multiplication can be performed faster using this adder.

REFERENCES

[1] A Hardware Efficient Very Large Bit Word Binary to Double Base Number System Converter for Encryption Applications *by Roberto Muscedere Electrical and Computer Engineering, University of Windsor Ontario, Canada*

[2] Parallel Addition for Double Base Number System *by Wutthipat chalermchatwichien and Athasit Surarerks* at 10th International Joint Conference on Computer Science and Software Engineering