Asmita Haveliya / International Journal of Engineering Research and Applications (IJERA)
ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 1,Jan-Feb 2012, pp.678-684

FPGA IMPLEMENTATION OF A VEDIC CONVOLUTION ALGORITHM

Asmita Haveliya

M.Tech. (Pursuing), Dept. Of Electronics, ASET
Amity University
Lucknow, India

ABSTRACT

In digital signal processing convolution is a fundamental computation that is ubiquitous in many application areas. In order to compute convolution of long sequence, Overlap-Add method (OLA) and Overlap-Save method (OLS) methods are employed. In this paper, block convolution process is proposed using a multiplier architecture based on vertical and crosswise algorithm of Ancient Indian Vedic Mathematics and embedding it in OLA method for reducing calculations.And as the vedic multiplier is been used it is named as Vedic convolution algorithm.The coding is done in VHDL (Very High Speed Integrated Circuits Hardware Description Language) for the FPGA , as it is being increasingly used for variety of computationally intensive applications.Simulation and synthesis is done using Xilinx.

Keywords - **Convolution; Overlap-Add (OLA); Overlap-Save (OLS); Vedic Maths; VHDl .**

I. INTRODUCTION

In this paper, Urdhva-Tiryakbhyam Sutra [7] is first applied to the binary number system and is used to develop digital multiplier architecture. This Sutra also shows the effectiveness of reducing the N×N multiplier [18] structure into an efficient 4×4 multiplier structures. This work presents a systematic design methodology for fast and area efficient digital multiplier based on Vedic mathematics [6]. The basic work proposed in this paper is been explained using the block diagram in Fig 1.

In this paper, the block convolution [21] algorithm is implemented in VHDL (Very High Speed Integrated Circuited Hardware Description Language) [3] and the FPGA synthesis and logic simulation are done using Xilinx ISE design suite 12.1

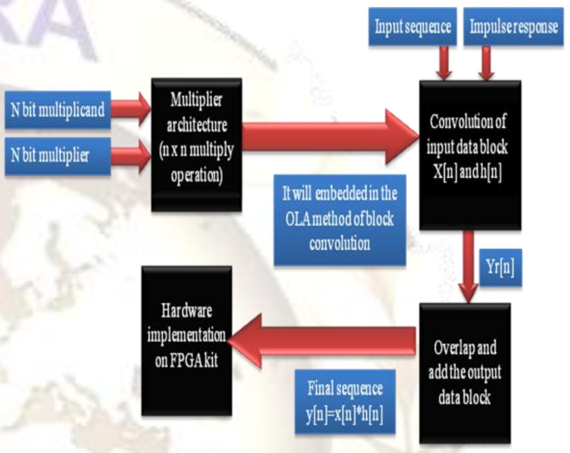

Input sequence
Impulse response
N bit multiplicand
N bit multiplier
Multiplier architecture (n x n multiply operation)
Convolution of input data block X[n] and h[n]
It will embedded in the OLA method of block convolution
Yr[n]
Hardware implementation on FPGA kit
Overlap and add the output data block
Final sequence y[n]=x[n]*h[n]

FIGURE 1: Block diagram of the process followed.

II. CONVOLUTION

Convolution [12] is the mathematical process that relates the output, y(t), of a linear, time-invariant system [4] to its input, x(t), and impulse response, h(t).

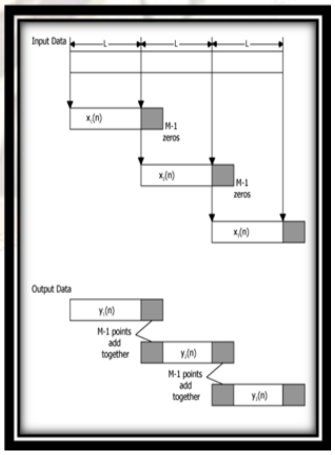


FIGURE 2: Overlap add method.

The overlap-add method [13] (OLA) is an efficient way to evaluate the discrete convolution between a very long signal x[n] with a finite impulse response h[n]**.**The Fig.1 shows the concept of overlap add [11] method by Zero-pad length-*L* blocks by *M*−1 samples. Add successive blocks, overlapped by *M*−1 samples, so that the tails sum to produce the complete linear convolution.

III. URDHVA-TIRYAGBHYAM SUTRA

Urdhva-Tiryagbhyam [19] is the general formula applicable to all cases of multiplication and also in the division of a large number by another large number. It means vertically and crosswise. We discuss multiplication of two, three digit numbers with this method by placing the carried over digits under the first row and proceed.

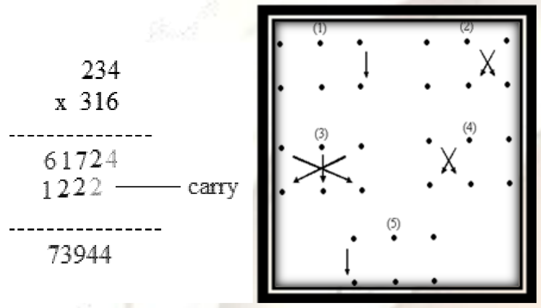


FIGURE 3: General rule for a 3 digit by 3 digit multiplication.

Steps:

i) 4 X 6 = 24: 2, the carried over digit is placed below the second digit.

ii) (3 X 6) + (4 x 1) = 18 + 4 = 22; 2, the carried over digit is placed below third digit.

iii) (2 X 6) + (3 X 1) + (4 X 3) = 12 + 3 + 12 = 27; 2, the carried over digit is placed below fourth digit.

iv) (2 X 1) + (3 X 3) = 2 + 9 = 11; 1, the carried over digit is placed below fifth digit.

v) (2X3) =6.

vi) Respective digits are added.

The basic rule for the multiplication of two numbers of 6 digits is shown using the line drawing as follows. Similarly for any number of digits this multiplication technique of ancient Indian Vedic mathematics [6] can be used.

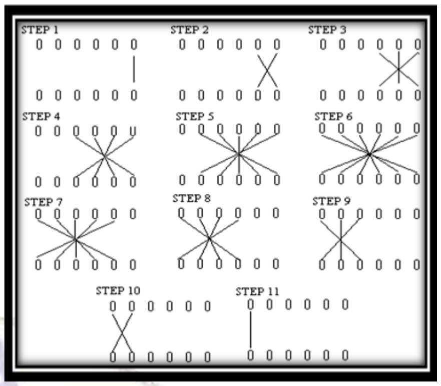
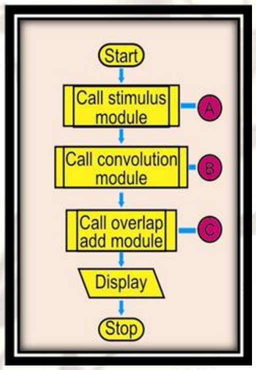


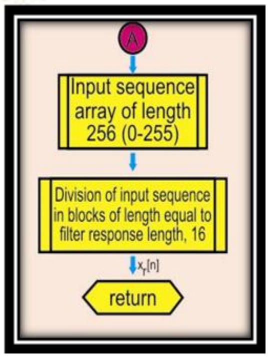
FIGURE 4: General rule for a 6 digit by 6 digit multiplication.

IV. METHEDOLOGY FOLLOWED

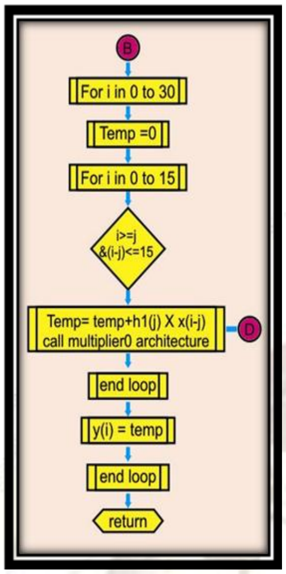
In this proposed paper we have made a convolution with x(n) and h(n) both having 256 samples.And as we are performing block convolution using overlap add method this sample is divided into 16 input data block for OLA method, each having 16 elements. The methedology followed in this proposed work is explained using the flow diagrams below.



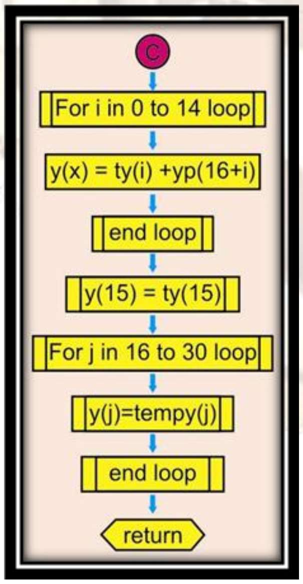
This flow “A” is the stimulus module which is dividing the input sequence of length 256 into 16 input blocks of length 16.


A
Input sequence array of length 256 (0-255)
Division of input sequence in blocks of length equal to filter response length, 16
$x_r[n]$
return

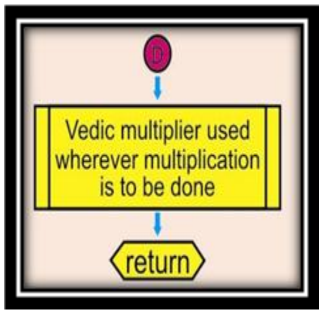
This flow "B" is the convolution module which is performing the convolution of individual block with 16 elements.



The flow "C" is overlap add module which helps in providing efficient area and speed of the proposed architechture as it reduces the complexity of the calculation.



This flow "D" is the main block which reduces the calculation complexity to a very wide extent.This block is Vedic multiplier which is used wherever multiplication is to be done.



V. FPGA

The introduction of field programmable gate arrays (FPGA), has made it feasible to provide hardware for application specific computation design. The changes in designs in FPGA's [20] can be accomplished within a few hours, and thus result in significant savings in cost and design cycle. FPGAs offer speed comparable to dedicated and fixed hardware systems for parallel algorithm.The vedic convolution algorothm proposed in this paper is been simulated and synthesised using the xilinx design suite 12.1 with the device family as Vertex 6 (low power).the summary of the device description of the vertex FPGA used is explained in the table below

TABLE I. SUMMARY OF FPGA FEATURES

Device Family	Vertex 6
Device	XC6VLX75TL
Package	FF484
Speed Grade	-3L

The features of the vertex 6 FPGA used in this proposed work with Xilinx Design Suite 12.1, as described the Xilinx are listed in the table below.

TABLE II. SUMMARY OF VERTEX 6 FEATURES

Features	Virtex-6
Logic Cells	760,000
BlockRAM	38Mb
DSP Slices	2,016
DSP Performance (symmetric FIR)	2,419GMACS
Transceiver Count	72
Transceiver Speed	11.18Gb/s
Total Transceiver Bandwidth (full duplex)	536Gb/s
Memory Interface (DDR3)	1,066Mb/s
PCI Express® Interface	Gen2x8

Agile Mixed Signal (AMS)/XADC	Yes
Configuration AES	Yes
I/O Pins	1,200
I/O Voltage	1.2V, 1.5V, 1.8V, 2.5V
EasyPath Cost Reduction Solution	Yes

VI. RESULTS

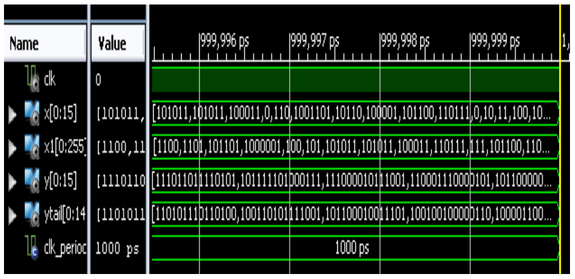
The main point of this paper was to introduce a method for calculating the linear convolution sum of two finite length sequences that is easy to learn and perform.It has been found on embedding Vedic multiplication for OLA, there is a considerable improvement in their performance.The table below shows the synthesis report of the proposed work with the logic resource utilization.

TABLE III. SUMMARY OF SYNTHESIS REPORT

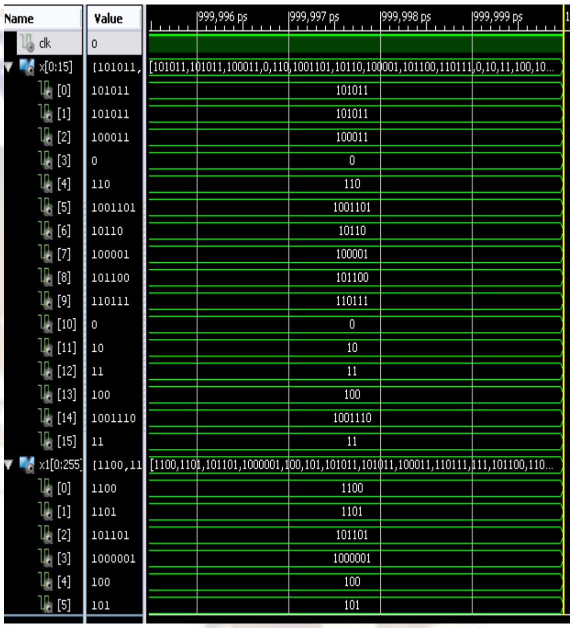
Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	988	93120	1%
Number of Slice LUTs	10799	46560	23%
Number of fully used LUT-FF pairs	408	11379	3%
Number of bonded IOBs	625	240	260%
Number of BUFG/BUFG CTRLs	1	32	3%

The other constraints of the synthesis report are as follows.
Total REAL time to Xst completion: 775.00 secs
Total CPU time to Xst completion: 774.89 secs
Total memory usage is 317328 kilobytes

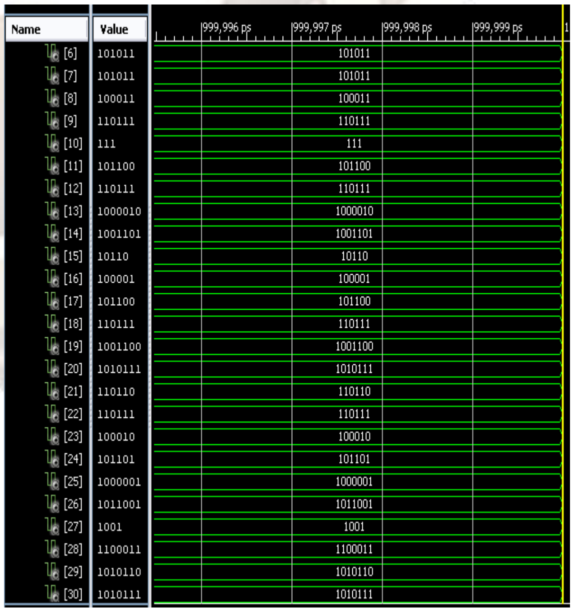
The following are the simulation results of the proposed work.



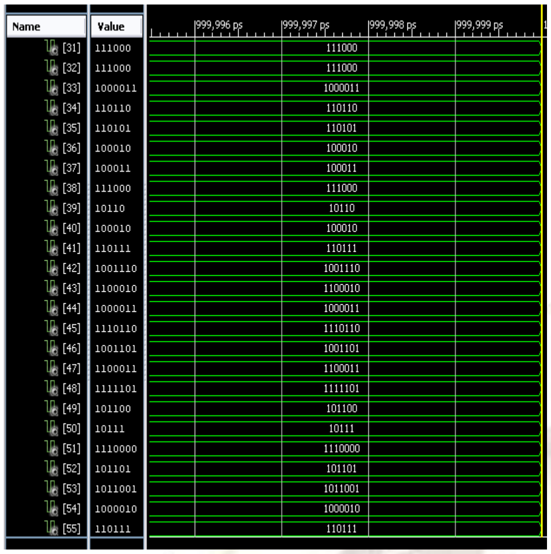
```
Name        Value       |999,996 ps |999,997 ps |999,998 ps |999,999 ps
clk         0
x[0:15]     [101011,    [101011,101011,100011,0,110,1001101,10110,100001,101100,110111,0,10,11,100,10...
x1[0:255]   [1100,11    [1100,1101,101101,1000001,100,101,101011,101011,100011,110111,111,101100,110...
y[0:15]     [1110110    [11101101 1110101,10111110 1000111,11100001011 1001,110001110000101,101100000...
ytail[0:14  [1101011    [1101011101 10100,1001101011 11001,10110001001 1101,1001001000010110,10000 1100...
clk_perioc  1000 ps     1000 ps
```



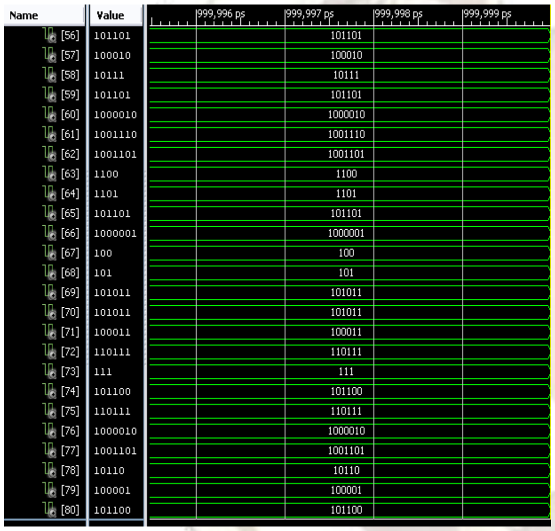
```
Name        Value       |999,996 ps |999,997 ps |999,998 ps |999,999 ps
clk         0
x[0:15]     [101011,    [101011,101011,100011,0,110,1001101,10110,100001,101100,110111,0,10,11,100,10...
  [0]       101011      101011
  [1]       101011      101011
  [2]       100011      100011
  [3]       0           0
  [4]       110         110
  [5]       1001101     1001101
  [6]       10110       10110
  [7]       100001      100001
  [8]       101100      101100
  [9]       110111      110111
  [10]      0           0
  [11]      10          10
  [12]      11          11
  [13]      100         100
  [14]      1001110     1001110
  [15]      11          11
x1[0:255]   [1100,11    [1100,1101,101101,1000001,100,101,101011,101011,100011,110111,111,101100,110...
  [0]       1100        1100
  [1]       1101        1101
  [2]       101101      101101
  [3]       1000001     1000001
  [4]       100         100
  [5]       101         101
```



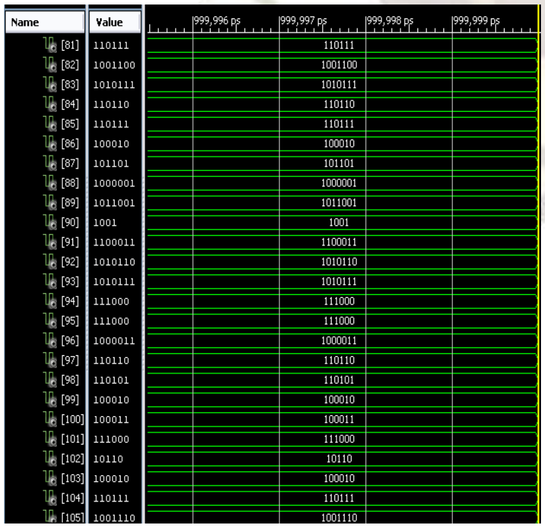
```
Name        Value       |999,996 ps |999,997 ps |999,998 ps |999,999 ps
  [6]       101011      101011
  [7]       101011      101011
  [8]       100011      100011
  [9]       110111      110111
  [10]      111         111
  [11]      101100      101100
  [12]      110111      110111
  [13]      1000010     1000010
  [14]      1001101     1001101
  [15]      10110       10110
  [16]      100001      100001
  [17]      101100      101100
  [18]      110111      110111
  [19]      1001100     1001100
  [20]      1010111     1010111
  [21]      110110      110110
  [22]      110111      110111
  [23]      100010      100010
  [24]      101101      101101
  [25]      1000001     1000001
  [26]      1011001     1011001
  [27]      1001        1001
  [28]      1100011     1100011
  [29]      1010110     1010110
  [30]      1010111     1010111
```



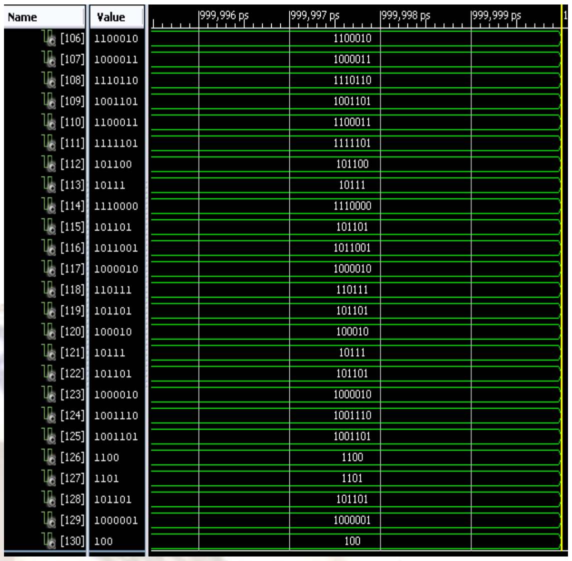
Name	Value
[31]	111000
[32]	111000
[33]	1000011
[34]	110110
[35]	110101
[36]	100010
[37]	100011
[38]	111000
[39]	10110
[40]	100010
[41]	110111
[42]	1001110
[43]	1100010
[44]	1000011
[45]	1110110
[46]	1001101
[47]	1100011
[48]	1111101
[49]	101100
[50]	10111
[51]	1110000
[52]	101101
[53]	1011001
[54]	1000010
[55]	110111



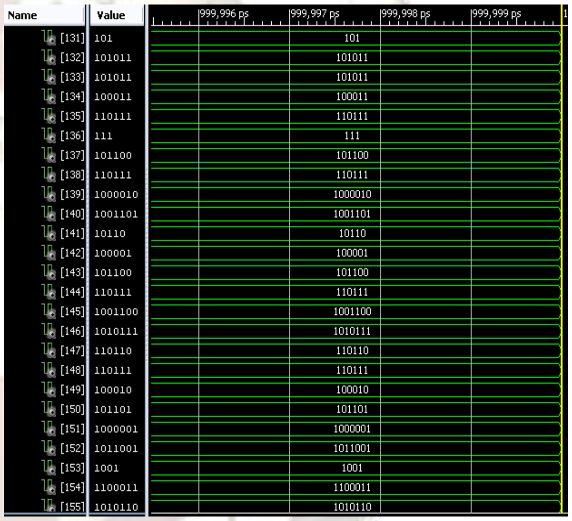
Name	Value
[56]	101101
[57]	100010
[58]	10111
[59]	101101
[60]	1000010
[61]	1001110
[62]	1001101
[63]	1100
[64]	1101
[65]	101101
[66]	1000001
[67]	100
[68]	101
[69]	101011
[70]	101011
[71]	100011
[72]	110111
[73]	111
[74]	101100
[75]	110111
[76]	1000010
[77]	1001101
[78]	10110
[79]	100001
[80]	101100



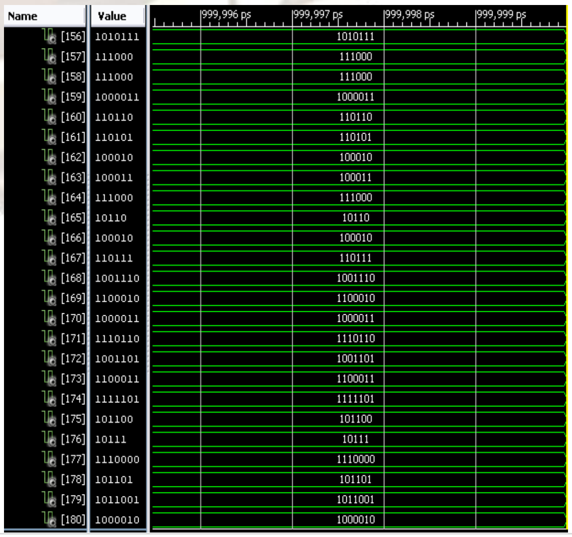
Name	Value
[81]	110111
[82]	1001100
[83]	1010111
[84]	110110
[85]	110111
[86]	100010
[87]	101101
[88]	1000001
[89]	1011001
[90]	1001
[91]	1100011
[92]	1010110
[93]	1010111
[94]	111000
[95]	111000
[96]	1000011
[97]	110110
[98]	110101
[99]	100010
[100]	100011
[101]	111000
[102]	10110
[103]	100010
[104]	110111
[105]	1001110



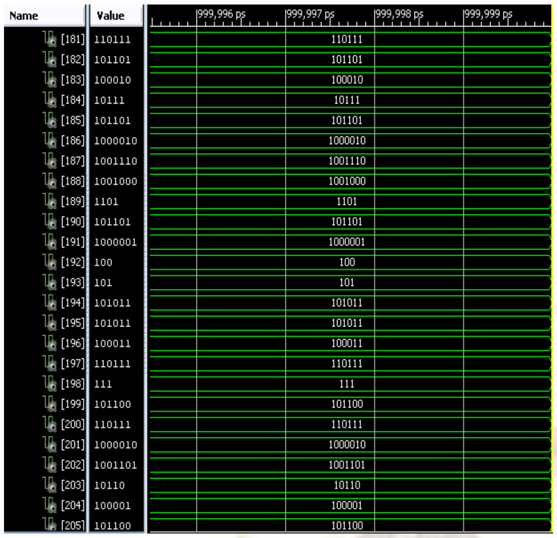
Name	Value
[106]	1100010
[107]	1000011
[108]	1110110
[109]	1001101
[110]	1100011
[111]	1111101
[112]	101100
[113]	10111
[114]	1110000
[115]	101101
[116]	1011001
[117]	1000010
[118]	110111
[119]	101101
[120]	100010
[121]	10111
[122]	101101
[123]	1000010
[124]	1001110
[125]	1001101
[126]	1100
[127]	1101
[128]	101101
[129]	1000001
[130]	100



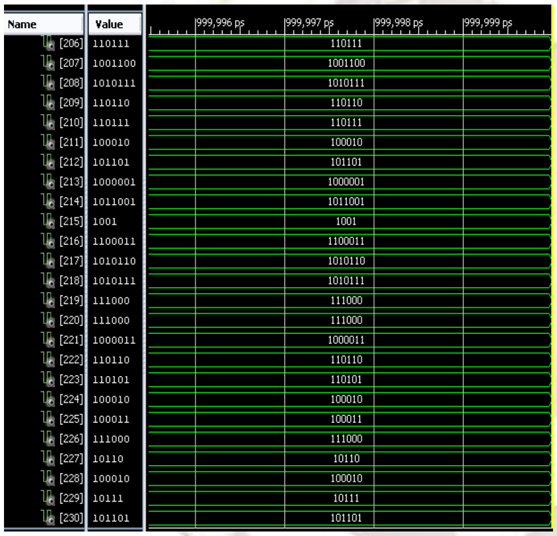
Name	Value
[131]	101
[132]	101011
[133]	101011
[134]	100011
[135]	110111
[136]	111
[137]	101100
[138]	110111
[139]	1000010
[140]	1001101
[141]	10110
[142]	100001
[143]	101100
[144]	110111
[145]	1001100
[146]	1010111
[147]	110110
[148]	110111
[149]	100010
[150]	101101
[151]	1000001
[152]	1011001
[153]	1001
[154]	1100011
[155]	1010110



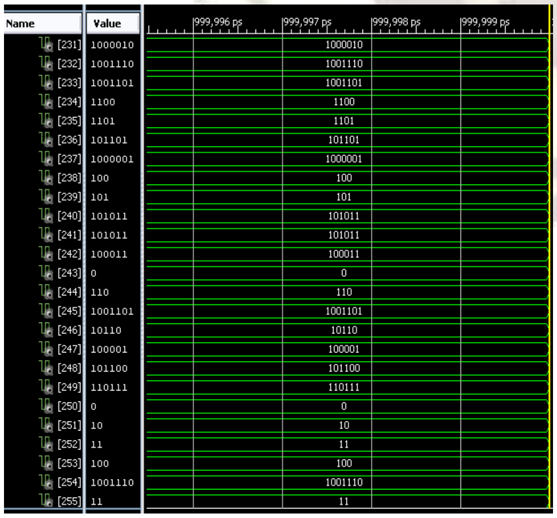
Name	Value
[156]	1010111
[157]	111000
[158]	111000
[159]	1000011
[160]	110110
[161]	110101
[162]	100010
[163]	100011
[164]	111000
[165]	10110
[166]	100010
[167]	110111
[168]	1001110
[169]	1100010
[170]	1000011
[171]	1110110
[172]	1001101
[173]	1100011
[174]	1111101
[175]	101100
[176]	10111
[177]	1110000
[178]	101101
[179]	1011001
[180]	1000010



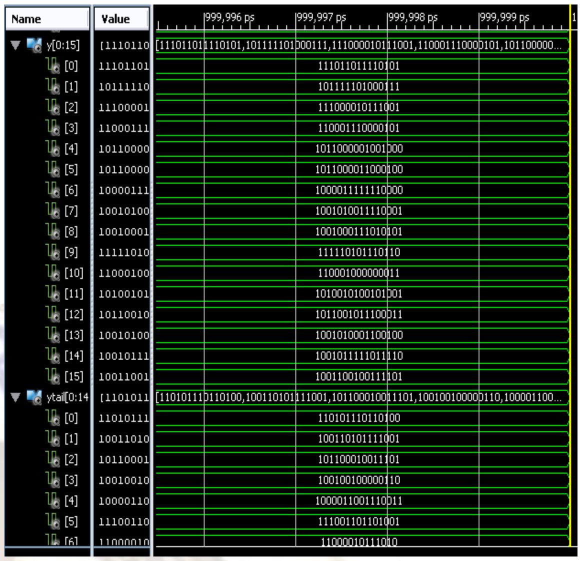
```
Name   Value     999,996 ps  999,997 ps  999,998 ps  999,999 ps
[181]  110111    110111
[182]  101101    101101
[183]  100010    100010
[184]  10111     10111
[185]  101101    101101
[186]  1000010   1000010
[187]  1001110   1001110
[188]  1001000   1001000
[189]  1101      1101
[190]  101101    101101
[191]  1000001   1000001
[192]  100       100
[193]  101       101
[194]  101011    101011
[195]  101011    101011
[196]  100011    100011
[197]  110111    110111
[198]  111       111
[199]  101100    101100
[200]  110111    110111
[201]  1000010   1000010
[202]  1001101   1001101
[203]  10110     10110
[204]  100001    100001
[205]  101100    101100
```



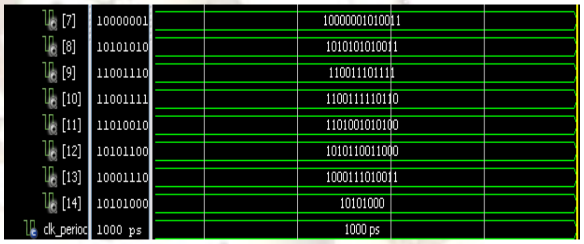
```
Name   Value     999,996 ps  999,997 ps  999,998 ps  999,999 ps
[206]  110111    110111
[207]  1001100   1001100
[208]  1010111   1010111
[209]  110110    110110
[210]  110111    110111
[211]  100010    100010
[212]  101101    101101
[213]  1000001   1000001
[214]  1011001   1011001
[215]  1001      1001
[216]  1100011   1100011
[217]  1010110   1010110
[218]  1010111   1010111
[219]  111000    111000
[220]  111000    111000
[221]  1000011   1000011
[222]  110110    110110
[223]  110101    110101
[224]  100010    100010
[225]  100011    100011
[226]  111000    111000
[227]  10110     10110
[228]  100010    100010
[229]  10111     10111
[230]  101101    101101
```



```
Name   Value     999,996 ps  999,997 ps  999,998 ps  999,999 ps
[231]  1000010   1000010
[232]  1001110   1001110
[233]  1001101   1001101
[234]  1100      1100
[235]  1101      1101
[236]  101101    101101
[237]  1000001   1000001
[238]  100       100
[239]  101       101
[240]  101011    101011
[241]  101011    101011
[242]  100011    100011
[243]  0         0
[244]  110       110
[245]  1001101   1001101
[246]  10110     10110
[247]  100001    100001
[248]  101100    101100
[249]  110111    110111
[250]  0         0
[251]  10        10
[252]  11        11
[253]  100       100
[254]  1001110   1001110
[255]  11        11
```

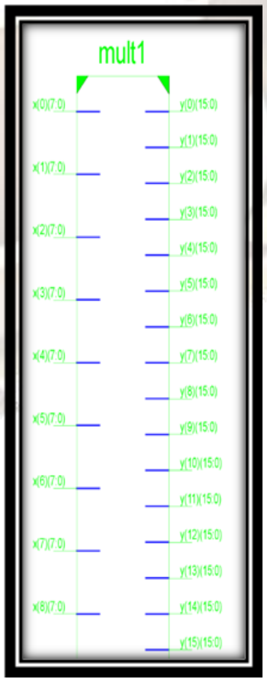


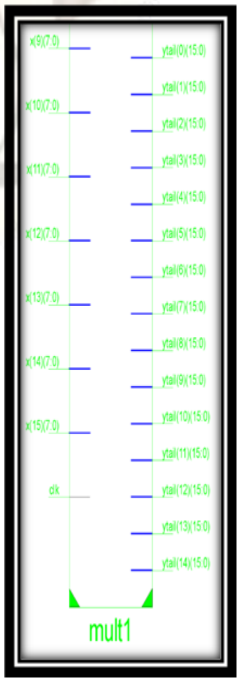
```
Name          Value      999,996 ps  999,997 ps  999,998 ps  999,999 ps
y[0:15]       [1110110   [1110110111110101,1011111101000111,1110000101111001,1100011100000101,101100000...
  [0]         11101101   1110110111110101
  [1]         10111110   1011111101000111
  [2]         11100001   1110000101111001
  [3]         11000111   1100011100000101
  [4]         10110000   1011000001001000
  [5]         10110000   1011000011000100
  [6]         10000111   1000011111110000
  [7]         10010100   1001010011110001
  [8]         10010001   1001000111010101
  [9]         11111010   1111101011110110
  [10]        11000100   1100010000000011
  [11]        10100101   1010010100101001
  [12]        10110010   1011001011100011
  [13]        10010100   1001010001100100
  [14]        10010111   1001011111011110
  [15]        10011001   1001100100111101
ytail[0:14    [1101011   [1101011100110100,1001101011111001,1011000100111101,1001001000000110,100001100...
  [0]         11010111   1101011100110100
  [1]         10011010   1001101011111001
  [2]         10110001   1011000100111101
  [3]         10010010   1001001000000110
  [4]         10000110   1000011001110011
  [5]         11100110   1110011011101001
  [6]         11000010   1100001011101010
```



```
  [7]         10000001   100000010100011
  [8]         10101010   1010101010011
  [9]         11001110   110011101111
  [10]        11001111   1100111110110
  [11]        11010010   1101001010100
  [12]        10101100   1010110011000
  [13]        10001110   1000111010011
  [14]        10101000   10101000
clk_perioc    1000 ps    1000 ps
```

The figure shown below is the RTL view of the proposed Vedic convolution.





REFERENCES

[1] Asmita Haveliya,Kamlesh Kumar Singh," A Novel Approach For High Speed Block Convolution Algorithm (Based on Ancient Indian Vedic Mathematics Approach)" *International Conference on Advanced Computing and ommunication Technologies (ACCT 2011) Copyright © 2011 RG Education Society ISBN: 978-981-08-7932-7*.].

[2] Very High Speed Integrated Circuit Hardware Description Language. URL: http://electrosofts.com/vhdl/.

[3] Alan V. Oppenheim, Ronald W. Schafer with John R. Buck, *Discrete Time Signal Processing*, Second Edition.

[4] Hanuman tharafu .M.C., Jayalaxmi . H., Renuka R.K., Ravishankar .M., "A high speed block convolution using Ancient Indian Vedic Mathematics", *IEEE international conference on computational intelligence and multimedia applications, 2007.*

[5] Jagadguru Swami Sri Bharati Krishna Tirthji Maharaja, "*Vedic Mathematics*", Motilal Banarsidas, Varanasi, India, 1986.

[6] A.P Nicholas, K.R Williams, J. Pickles-*Vertically and Crosswise applications of the Vedic Mathematics Sutra*, Motilal Banarsidass Publishers, Delhi, 2003.

[7] J.G Proakis and D.G Monalkis, *Digital Signal Processing*. Macmillian,1988.

[8] Hanumantharaju M.C and Shashidhara K.S "A Novel Multiplier Architecture for FIR Filter Based on Field Programmable Gate Array's", *IEEE International Conference on Signal and Image Processing, Hubli, Dec 2006.*

[9] Purushottam D. Chidgupkar and Mangesh T. Karad, "The Implementation of Vedic Algorithms in Digital Signal Processing", *Global J. of Engng. Educ., Vol.8, No.2 © 2004 UICEE Published in Australia.*

[10] Madihalli J. Narasimha, "Modified Overlap-Add and Overlap-Save Convolution Algorithms for Real Signals" ,*IEEE Transactions on Signal Process, vol. 13, no. 11, pp. 669-671, Nov. 2006*.

[11] Madihalli J. Narasimha, "Linear Convolution using Skew Cyclic Convolutions", *IEEE Transactions on Signal Process, vol. 14, no. 3, pp. 173-176, Mar. 2007.*

[12] Shogo Muramatsu, Hitoshi Kiya, "An Extended Overlap-Add and Save Methods for Multirate Signal Processing" , *IEEE Transactions on Signal Process, vol. 45, no. 9, pp. 2376-2380, Sep 1997.*

[13] John W. Pierre, "A Novel Method for Calculating Convolution Sum of Two Finite Length Sequences" *IEEE Transactions on Education, vol. 39. no. 1 , pp. 77-80, Feb 1996.*

[14] Jung KapKuk and Nam Ik Cho "Block convolution with arbitrary delays using fast fourier transform" in Proceedings of *IEEE International Symposium on Intelligent Signal Processing and Communication Systems, Dec2005*.

[15] Shripad Kulkarni, "Discrete Fourier Transform (DFT) by using Vedic Mathematics", *report, vedicmathsindia.blogspot.com, 2007*.

[16] Abhijeet Kumar, Dilip Kumar, Siddhi, "Hardware Implementation of 16*16 bit Multiplier and Square using Vedic Mathematics", *Design Engineer, CDAC, Mohali.*

[17] Shamim Akhter, "VHDL Implementation of Fast NXN Multiplier Based on Vedic Mathematics", *Jaypee Institute of Information Technology University, Noida, 201307 UP, INDIA, 2007 IEEE.*

[18] Multi digit multiplication: vertically and crosswise. URL:http://threesixty360.wordpress.com/2008/02/15/multidigit-multiplication- vertically-and-crosswise/

[19] Ahmed Saeed, M. Elbably, G. Abdelfadeel, and M. I. Eladawy, "Efficient FPGA implementation of fft/ifft processor*", international journal of circuits, systems and signal processing, issue 3, volume 3, 2009.*

[20] Convolution. URL:http://mue.music.miami.edu/thesis/jvandekieft/jvchapter2.htm