Turkish Journal of Computer and Mathematics Education Vol.12 No.11 (2021), 4739-4749
Research Article

Vlsi Implementation Of Multiply And Accumulate Unit Using Offset Binary Coding Distributed Arithmetic

Bharathi.M[1], Dr. Yasha Jyothi M Shirur[2]

[1]Research Scholar in BNMIT , VTU and Assistant Professor, Department of ECE, Center for VLSI & Embedded Systems, Sree Vidyanikethan Engineering College, Tirupati, Andhra Pradesh.
[2]Professor of ECE, BNMIT, Bangalore, VTU.
[1]bharathi891@gmail.com, [2]yashamallik@gmail.com

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 10 May 2021

Abstract: In general, Digital Signal processors are designed with Harvard architecture which in turn comprises a special block called Multiply and Accumulate unit (MAC). The speed improvement of any processor can be done by improving the speed of the dedicated Multiply and Accumulate unit. Offset binary coding based Distributed Arithmetic (DA) is a compelling technique that improves the area, delay, and power trade-off in designing of MAC core, and which in-turn adds the benefits to digital signal processor design. Also introduced the mathematical concepts which lead to offset based Distributed arithmetic are shown below. The different optimization techniques of offset based Distributed Arithmetic based MAC core is synthesized and implemented for efficient implementation of inner product generation. Implementation of different Offset based distributed architectures such as LUT based four-term & two-term, single LUT inner product computations are compared with LUT-less based architecture are done. The conclusion drawn from this research work is demonstrated on 16-bit MAC cores using offset binary coding distributed arithmetic architectures using Xilinx ISE 14.7 and verified functionality using simulation results. The design is synthesized to know the area, delay, power and energy issues. The offsent based Distributed Airthmetic is compared with its counterparts. Based on the analysis it is found that LUT-LESS can save the power delay product of 7.33% over LUT based when the worst case margin is considered with 1.754% of area reduction.

Keywords: Offset Binary coding (OBC), Distributed Arithmetic (DA),Look up Table (LUT) Adder based (LUT-LESS)

1. Introduction

System on Chip/System on a chip(SOC) is an IC that integrates most of the blocks on a single chip. Any general SOC architecture includes DSP block, Memory elements, and Input / Output blocks. A dedicated DSP core is used inside the SOC for real-time Computing purposes. Since Dedicated DSP block usually gives good performance efficiency than that of a general-purpose processor. In this increasing technology, DSPs are the fastest Digital signal Processor is an example of Harvard Architecture which fetches data and program instruction parallelly can be suited in many real-time applications such as digital broadcast, video and signal processing, image processing, communication systems & many more. Major DSP manufacturers are Texas, Analog Devices, and Motorola are designing dedicated DSPs for the application intended using Harvard Architecture as shown in figure 1. The MAC, or "Multiply and Accumulate [7]" unit core is a major kernel to perform multiplication operations in DSP systems. Let X, Y are the inputs, the basic MAC operation includes Z = Z + x*y, where Z is an accumulator unit as shown in figure 2. This is the most fundamental operation used in many DSP architectures.The future MAC in DSP needs to perform more computational functions to engage in real-time signal processing operations of the complex applications.

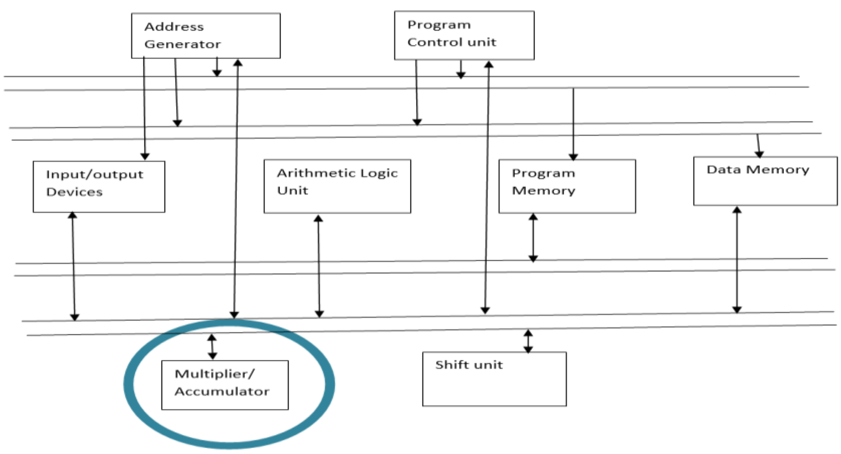


Figure 1:Hardvard Architecture

The MAC, or "Multiply and accumulate" unit core is a major kernel to perform multiplication operations in DSP systems. Let X, Y are the inputs, the basic MAC operation includes Z = Z + x*y, where Z is an accumulator unit as shown in figure 2. This is the most fundamental operation used in many DSP architectures. The future MAC in DSP needs to perform more computational functions to engage in real-time signal processing operations of the complex applications.

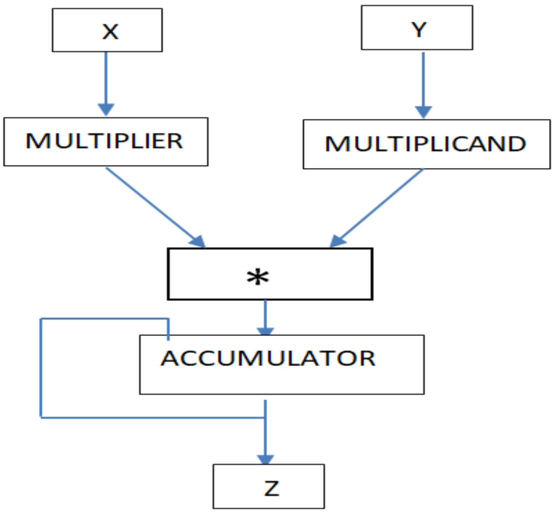


Figure 2: Multiply and Accumulate Core

2. Existing Distributed Arithmetic

In Existing DA,computationInner product between two inputs X &Y can be done using precomputed LUT's. . This can be well suited for both ASIC and FPGA based implementations. The Distributed Arithmetic [5] based MAC core can be expressed using mathematical concepts as shown below.

Algorithm:

Suppose that X is the vector of input samples and X is a constant vector of input coefficient, corresponding to the MAC unit. Vector X and Y each consist of M elements XK and YK. The dot product Z of X and Y can be written as

Consider the following sum of product:

$$Z = \sum_{k=1}^{k} X_k Y_k \quad \ldots\ldots\ldots (1)$$

- Let Y_k be an N-bit scaled two's complement number. In other words,

$$|Y_k| < 1$$
$$Y_k : \{b_{k0}, b_{k1}, b_{k2} \ldots\ldots\ldots, b_{k(n-1)}$$

Where b_{k0} is the sign bit

- b. We can express X_k as

$$Y_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \qquad \ldots\ldots (2)$$

c. Substituting (2) in (1),

$$Z = \sum_{k=1}^{k} X_k \left[-b_{k0} + \sum_{n=1}^{N-1} b_{kn}\, 2^{-n} \right]$$

$$Z = \sum_{k=1}^{k} (b_{kn} * X_k) + \sum_{k=1}^{k} \sum_{n=1}^{N-1} (X_k * b_{kn}) \qquad \ldots (3)$$

$$Z = -\sum_{k=1}^{k}(b_{k0} * X_k) + \sum_{k=1}^{k}[(X_k * b_{k1})2^{-1} + (X_k * b_{k2})2^{-2} + \cdots + (X_k * b_{k(N-1)})2^{-(N-1)}]$$

$$\begin{aligned} Z = &-[b_{10} * X_1 + b_{20} * X_2 + b_{k0} * X_k] \\ &+\left[(b_{11} * X_1)2^{-1} + (b_{12} * X_1)2^{-2} + \cdots + \left(b_{1(N-1)} * X_1\right)2^{-(N-1)}\right] \\ &+\left[(b_{21} * X_2)2^{-1} + (b_{22} * X_2)2^{-2} + \cdots + \left(b_{2(N-1)} * X_2\right)2^{-(N-1)}\right] \\ &\ldots\ldots \\ &+\left[(b_{k1} * X_k)2^{-1} + (b_{k2} * X_k)2^{-2} + \cdots + \left(b_{k(N-1)} * X_k\right)2^{-(N-1)}\right] \end{aligned}$$

$$\begin{aligned} Z = &-[b_{10} * X_1 + b_{20} * X_2 + b_{k0} * AX_k] \\ &+[(b_{11} * X_1) + (b_{21} * X_2) + \cdots + (b_{k1} * X)]2^{-1} \\ &+[(b_{12} * X_1) + (b_{22} * X_2) + \cdots + (b_{k2} * X)]2^{-1} \\ &\ldots\ldots\ldots \\ &+\left[\left(b_{1(N-1)} * X_1\right) + \left(b_{2(N-1)} * X_2\right) + \cdots + \left(b_{k(N-1)} * X_k\right)\right]2^{-(N-1)} \end{aligned}$$

$$Z = -\sum_{k=1}^{k}(b_{k0}) * X_k + \sum_{n=1}^{N-1}[b_{1n} * X_k + b_{2n} * X_2 + \cdots + b_{kn} * AX_k]2^{-n}$$

$$Z = -\sum_{k=1}^{k} X_k * (b_{k0}) + \sum_{n=1}^{N-1}[\sum_{k=1}^{k} X_k * b_{kn}]2^{-n} \qquad \ldots.(4)$$

Consider the equation (4) rewritten as:

$$Z = \sum_{n=1}^{N-1}\left[\sum_{k=1}^{k} X_k\, b\right]2^{-n} + \sum_{k=1}^{k} X_k\,(-b_{k0})$$

- $\left[\sum_{k=1}^{k} X_k\, b_{kn}\right]$ has only 2^{k} possible values
- $\left[\sum_{k=1}^{k} X_k\, b_{kn}\right] hasonly\ 2^{k} possiblevalues$
  - With the sign bit as an input,we can store it in a ROM of size=2*2^{k}

To realize the inner product computation, the conventional DA uses a LUT-based architecture as shown in Figure 3 .It includes 3 blocks mainly

1)Input Data Section
2)LUT section
3)Accumulator Section

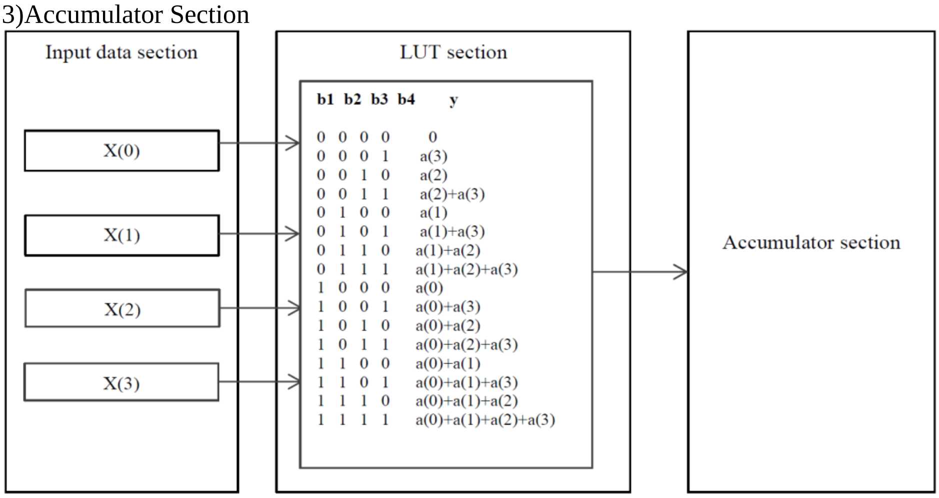


Figure 3: Conventional Distributed Arithmetic MAC core

In the data section, the bits of inputs are ({X0, X1,⋯, Xi}) are applied to create LUT addresses. The contents in LUT follow accumulator which in turn includes adder and register fork rising from to N-1as appeared in Equation (3). Several updating shifters within the accumulator can take place with previous output is to create progressive scaling with powers of two. After N cycles, compared to the bit-width of input vector X, the ultimate esteem of yield Z can be a final result as the result of the accumulation.

The two limitations of using this DA are:

1) This bit-serial multiplication design of LUT based DA gets to be a bottleneck when achieving the result for each clock cycle.

2) Another issue with LUT-based DA is that its LUT measure (2K word) develops exponentially as K increments. As the number of inputs are growing further tends to increase the LUT entries. LUT Based DA speeds up the duplication preparation by pre-computing all conceivable values and putting away them in a LUT Section.

3. Method

Offset Binary Coding method is based on a modified two's-complement representation of the values and reduces the size of LUT by half. The OBC can be further extended, reducing the memory size in steps by factor of two from 2K to K in theory. However, this requires additional hardware in terms of adders and multiplexers, thus increasing the delay.

Offset Binary Coding Algorithm:

.

$$Y_k = \frac{1}{2}[x_k - (-x_k)]$$

$$Y_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \qquad \ldots(5)$$

Equation (5) is converted into 2’s complement

$$-Y_k = -\overline{b_{k0}} + \sum_{n=1}^{N-1} \overline{b_{kn}}\, 2^{-n} + 2^{-(N-1)}$$

$$Y_k = \frac{1}{2}\left[-\left(b_{k0} - \overline{b_{k0}}\right) + \sum_{n=1}^{N-1}\left(b_{kn} - \overline{b_{kn}}\right) 2^{-n} - 2^{-(N-1)}\right]$$

- Define: Offset code

$$c_{kn} = \begin{cases} b_{kn} - \overline{b_{kn}} & ,n \neq 0 \\ -(b_{kn} - \overline{b_{kn}} & ,n = 0 \end{cases} where c_{kn} \in \{-1,1\}$$

- Finally

$$Y_k = \frac{1}{2}\left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)}\right]$$

Using the new x_k we have

$$Y_k = \frac{1}{2}\left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)}\right]$$

- Substitute the new x_k in

$$Z = \sum_{k=1}^{k} X_k Y_k$$

$$Z = \frac{1}{2}\sum_{k=1}^{k} X_k \left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)}\right]$$

$$Z = \frac{1}{2}\sum_{k=1}^{k}\sum_{n=0}^{N-1} X_k\, c_{kn} 2^{-n} - \frac{1}{2}\sum_{k=1}^{k} X_k\, 2^{-(N-1)}$$

$$\boxed{-\frac{1}{2}\sum_{k=1}^{k} X_k\, 2^{-(N-1)}}$$

$$Z = \sum_{n=0}^{N-1} \frac{1}{2} \sum_{k=1}^{k} X_k \, c_{kn} 2^{-n} \qquad \dots (6)$$

If we let

$$Q(c_{1n} c_{2n} c_{3n} \dots c_{kn}) = \frac{1}{2} \sum_{k=1}^{k} X_k \, c_{kn}$$

$$\boxed{Q(0) = \frac{1}{2} \sum_{k=1}^{k} X_k \qquad \text{Constant}}$$

$$y = \sum_{n=0}^{N-1} Q(c_{1n} \, c_{2n} c_{3n} \dots c_{kn}) 2^{-n} + 2^{-(N-1)} Q(0) \qquad \dots\dots (7)$$

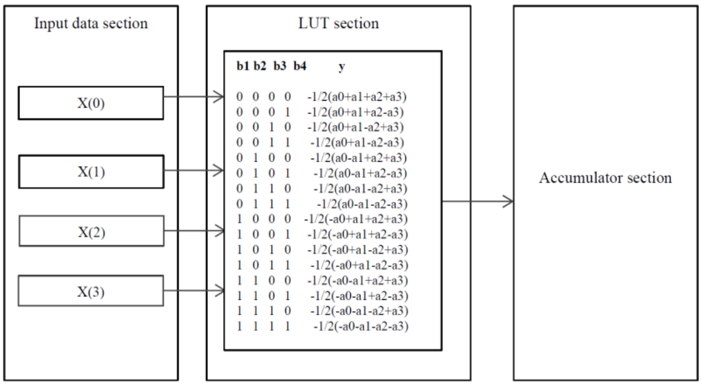


Figure 4: Offset Binary Coding Distributed Arithmetic MAC core

It can be seen from the figure3 that Distributed Arithmetic, LUT section with N inputs have 2N entries which takes different magnitude values, where as in figure 4, Offset Binary Coding architecture take the magnitude values with a sign which are still consistent with the statements as DA architecture.

Let us have a look at how the OBC architecture works. The values stored in the LUT section are shown in the figure. For (0111) the output of LUT is -1/2(a0-a1-a2-a3) and for (1000) the output is -1/2(-a0+a1+a2+a3) . It can be noticed that the upper half of the LUT is the same as the lower half but with the sign reversed thereby the size can still reduce by half. When N clock cycles' accumulation is done, the architecture will give the final result for OBC computation.

Various Techniques:

The different optimization techniques of offset based Distributed Arithmetic based MAC core are synthesized and implemented using Xilinx ISE P5.8f.

Two LUT OBC:

For the given N term, the number of LUT entries are 2N (Single LUT). In Two LUT, each of LUT 2N/2requires half compared with Single LUT but it requires an extra adder as shown in below figure.

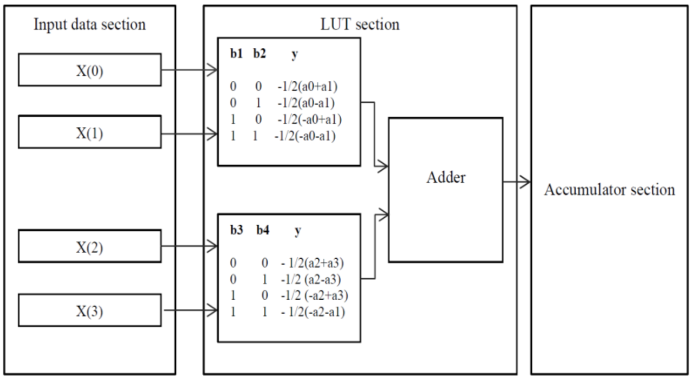


Figure 5: Offset Binary Coding Distributed Arithmetic MAC core using two LUT's

Four LUT OBC:

For example, for N = 16 the LUT in the baseline implementation requires 65,536 (216) rows.

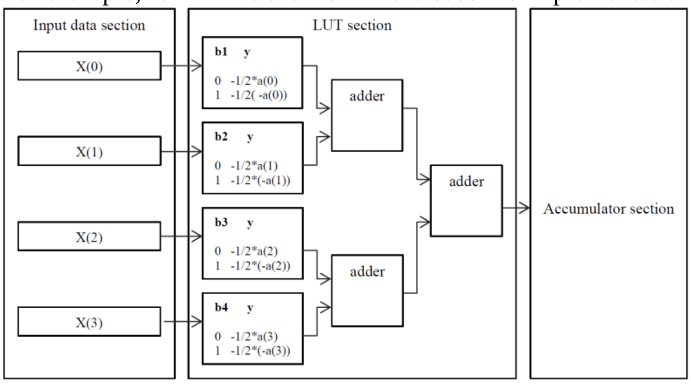
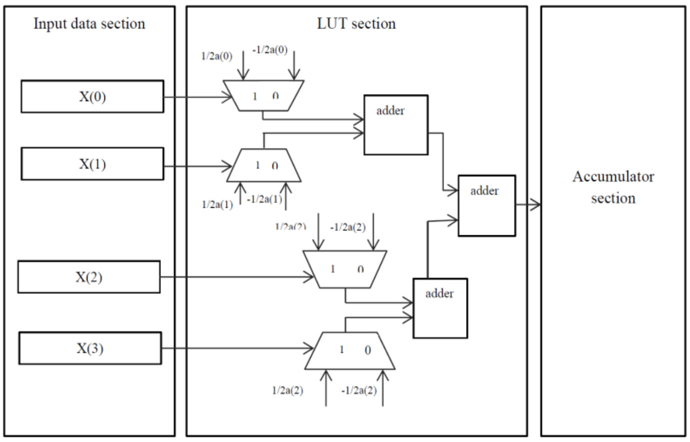


Figure 6: Offset Binary Coding Distributed Arithmetic MAC core with Four LUT's

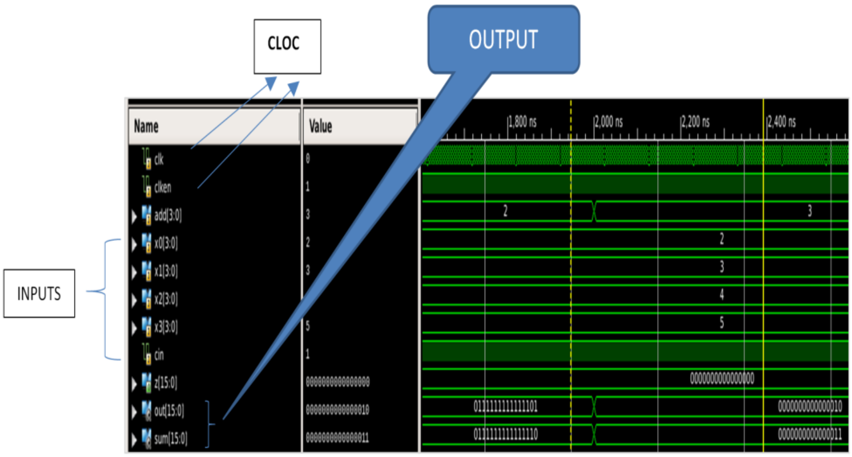
With 2-bank splitting the implementation requires two LUTs each with 256 (28) rows, which is still prohibitively large. Thus, for four LUT of N, the coefficients can be split into four banks.

LUT-LESS OBC (ADDER based OBC):

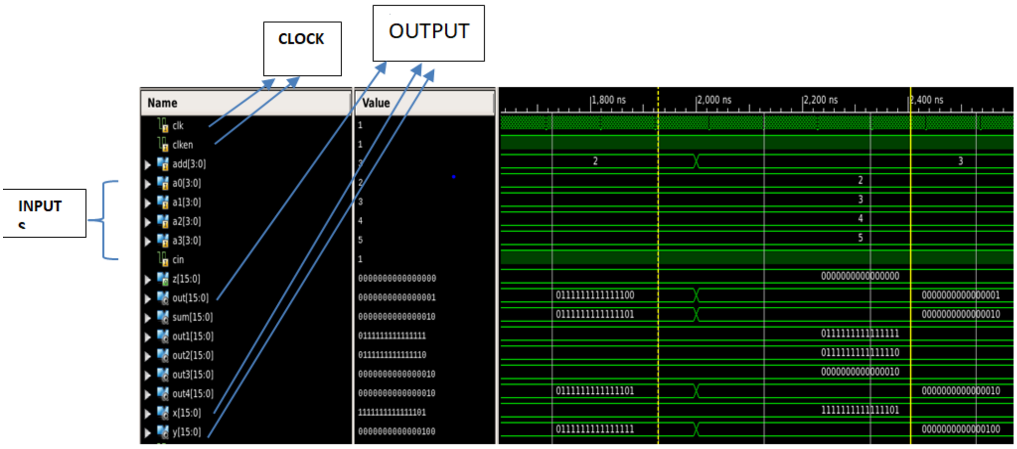


4. Experimental Results and Evaluation

The simulation and synthesis of the above architectures are done using Xilinx ISE P5.8f. The results are shown below:

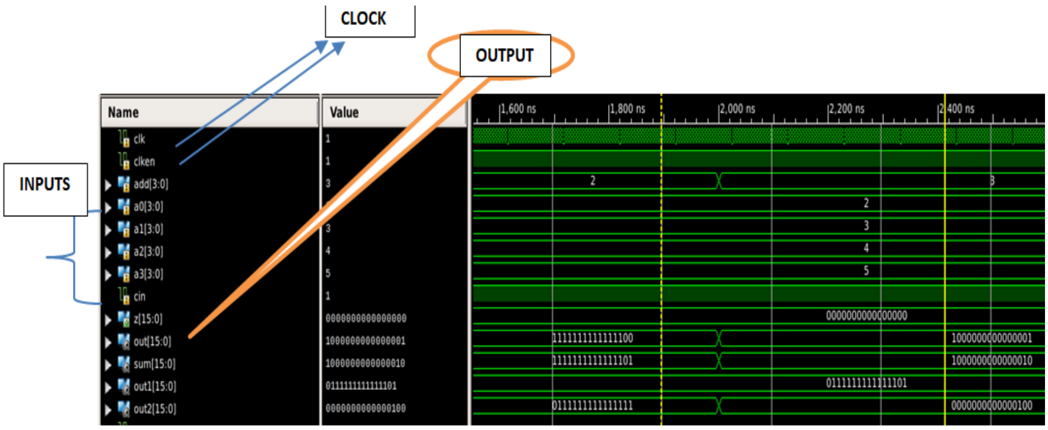


Simulation 1: OBC DA-based implementation of single LUT for inner-product computation
Inputs = a0,a1,a2,a3 = 2,3,4,5
ADDR = 0011
Out =-(a0+a1-a2-a3)/2; = 2(0010)
sum = Out+cin = 2+1 = 3(0011)
Clk , clken =1 then z =0 else accumulation can be done

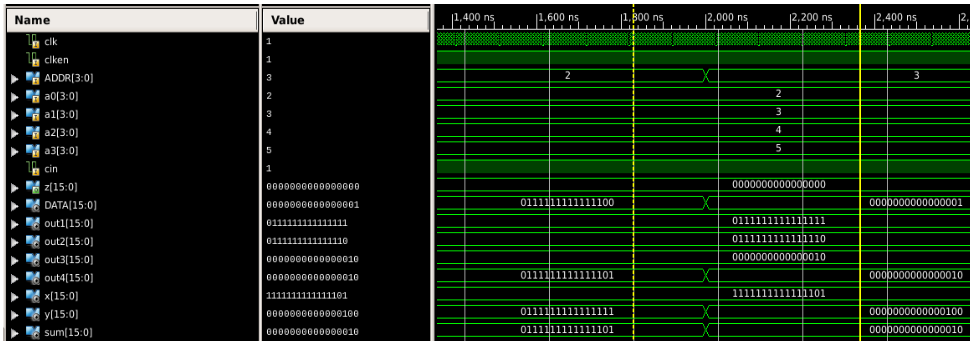


Simulation 2: OBC DA-based implementation of a four-term LUT inner-product computation.

Inputs = a0,a1,a2,a3 = 2,3,4,5
ADDR = 0011
Out1 = -1/2(a0) =-1(0111)
Out2 =-1/2(a1) = -2(1110)
Out3 = -1/2(-a2) =2(0010)
Out4 = -1/2(-a3) =2(0010)
X = Out1+Out2 = -3(1101)
Y = Out3+Out4 = 4(0100)
Out = X+Y = -3+4 =1(0001)
sum = Out+cin = 1+1 = 2(0010)
Clk , clken =1 then z =0 else accumation can be done

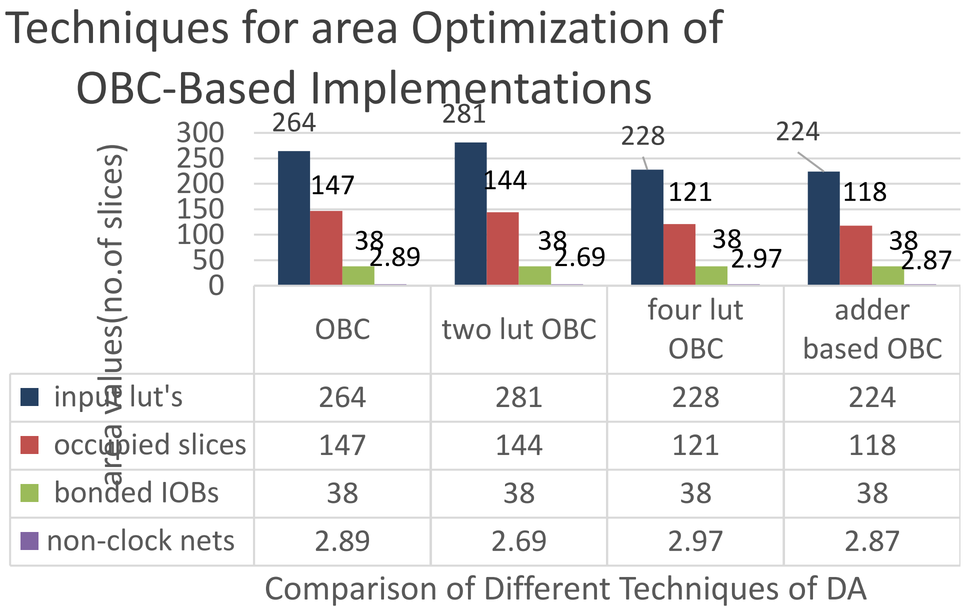


Simulation 3: OBC DA-based implementation of a two-term LUT inner-product computation.
Inputs = a0,a1,a2,a3 = 2,3,4,5
ADDR = 0011
Out1 = -(a0+a1) /2=-3(1101)
Out2 = -(-a2-a3) = 4(0100)
Out = Out1+Out2= -3+4 = 1(0001)
sum = Out+cin = 1+1 = 2(0010)
Clk , clken =1 then z =0 else accumation can be done

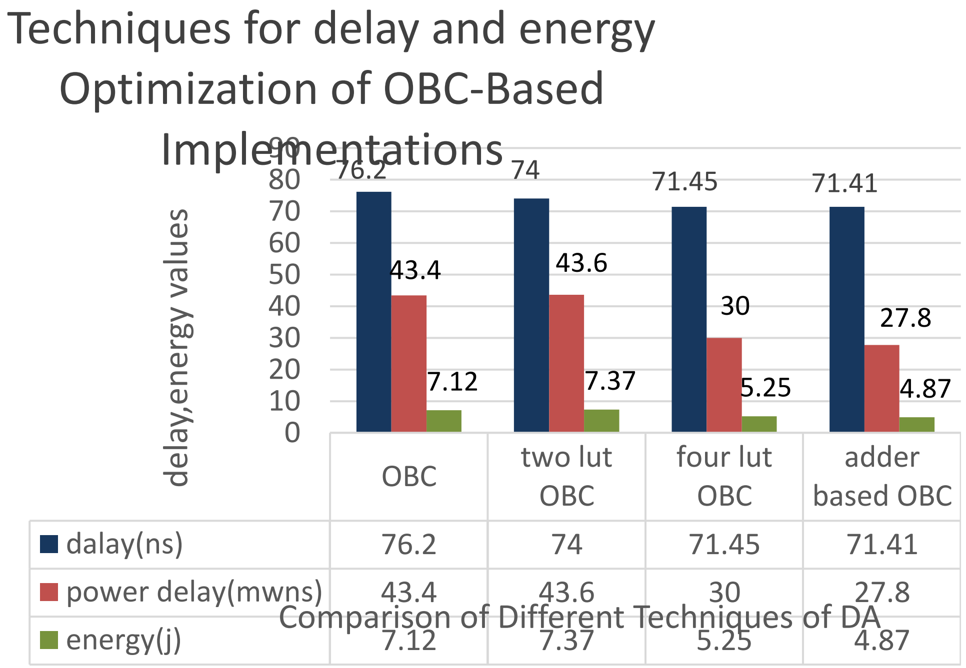


Simulation 4: OBC DA-based implementation of a LUT-LESS(Adder based) inner-product computation.
Inputs = a0,a1,a2,a3 = 2,3,4,5
ADDR = 0011
Out1 = -1/2(a0) =-1(0111)
Out2 =-1/2(a1) = -2(1110)
Out3 = -1/2(-a2) =2(0010)
Out4 = -1/2(-a3) =2(0010)
X = Out1+Out2 = -3(1101)
Y = Out3+Out4 = 4(0100)
Out = X+Y = -3+4 =1(0001)
sum = Out+cin = 1+1 = 2(0010)
Clk , clken =1 then z =0 else accumation can be done

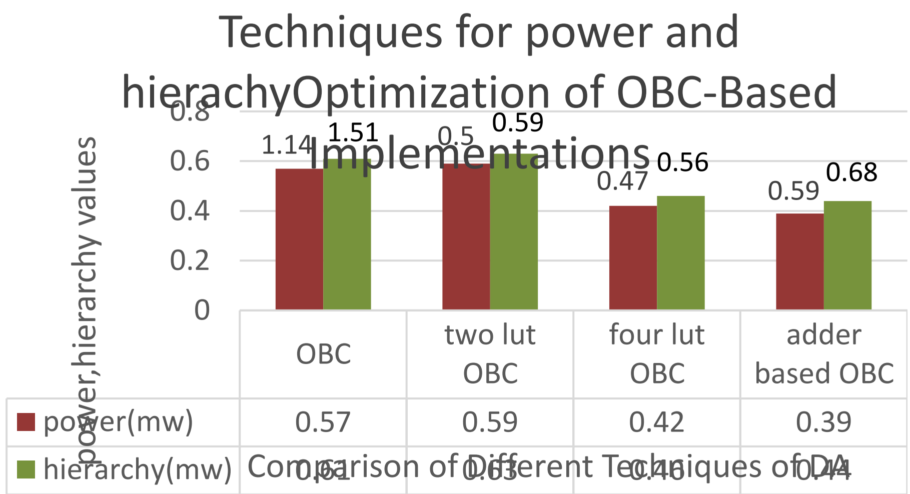
Performance Analysis of Area



Here in OBC-DA architectures, comparison is done with four bank, two bank and adder based(LUT-Less)architectures and compared. Among them adder based consumes less area among all types of other architectures. From the above chart adder based reduces area of 1.754% slices compared with conventional Single LUT based OBC.



From the above chart adder based OBC-DA has 71.41 ns where as conventional OBC has a delay of 76.2ns.so that delay consumption is decreased.



From the above chart OBC-DA consumes increase in power of 7.14% compared with conventional OBC-DA but the power delay product is saved by 7.333%.

5. Results

MAC is the most essential block which can be seen in most DSP Applications [2]. Offset binary coding based Distributed arithmetic plays a key role in implementing DSP functions in ASIC and FPGA devices. The proposed design relies on LUT based and LUT-Less based. In LUT based designs, partitioning the size of LUT leads to a trade-off between area and speed performance. LUT-Less implementation requires several cycles with adders to compute k bits of input data. The architectures are modelled in Verilog HDL and verified using Xilinx ISE. As there is a huge demand for DSP applications, in calculating the pre-computed SOP, the proposed Offset binary coding discussed can be used in high-speed DIP and DSP applications. From the charts, it is observed that LUT-less based design has a less critical path over LUT based MAC core using OBC based DA. This work includes analysis of the area, delay, power, power-delay, and energy-delay products of LUT-Less based and LUT based four-term, two-term, and conventional OBC based DA MAC architectures. Finally, the power delay product of LUT-less is saved by 7.333% compared with conventional DA.

6. Future work

Researchers have many choices of flexibility in designing the desired LUT implementation also able to change the parameters for implementation. Also, low power techniques can be added to still reduce the power.

7. Acknowledgments

Acknowledgments: I would be grateful to thank Yasha Jyothi M Shirur for constantly supporting in my research work

8. Conflicts of interest

The authors are declaring no conflict of interest.

References

1. Mahesh Mehandale, Mohit Sharma, and Pramod Kumar Meher, DA-Based Circuits for Inner-Product Computation, pp:77-103.
2. JiafengXie, jianjun He and guanzheng Tan, FPGA realization of FIR filters for high speed and medium speed by using modified distributed arithmetic architectures, June 2010, pp: 365-370.
3. Wayne P.Burleson, Louis L.Scharf, A VLSI design methodology for distributed arithmetic, June 1991
4. Sonali Mehta, Balwindersingh and Dilip Kumar, performance Analysis of Floating Point MAC Unit, September 2013.
5. R.Prakash Rao, N.Dhanunjaya Rao , K.Naveen and P.Ramya, Implementation of the Standard Floating point MAC Using IEEE 754 Floating point adder, Feb 2018.
6. Yadagiri Karri, Rajesh Misra, Implementation of 32 Bit Floating point MAC Unit to Feed Weighted Inputs to Neural Networks, April 2015.
7. Mohamed Asan Basiri M, Noor Mahammad Sk, An Efficient Hardware-Based Higher Radix Floating Point MAC Design, November 2014.
8. Dhananjaya A, Deepali Koppad, Design Of High Speed Floating Point MAC Using Vedic Multiplier And Parallel Prefix Adder, June 2013.