Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 12, December 2022

AN ANALYTICAL STUDY ON HARDWARE EFFICIENT PARALLEL PREFIX ADDER DESIGN AND IMPLEMENTATION USING XILINX ISE

D.Rajani, Dept of Electronics and Communication Engineering, Sree Venkateswara College Of Engineering, Nellore (Dt), Andhra Pradesh, India.
E.Venkateswarlu, Dept of Electronics and Communication Engineering, Sree Venkateswara College Of Engineering, Nellore (Dt), Andhra Pradesh, India.
V.Venkata Sai Karthik, Dept of Electronics and Communication Engineering, Sree Venkateswara College Of Engineering, Nellore (Dt), Andhra Pradesh, India.
V. Mahesh Kumar, Dept of Computer Science and Engineering, Sree Venkateswara College Of Engineering, Nellore (Dt), Andhra Pradesh, India.

ABSTRACT
The fastest PPA belongs to a type of parallel prefix adders called kogge-stone adders, which do multi-bit addition. Because of their complex circuitry, these adders are difficult to install. Modified parallel prefix adders are developed for 16-bit and 32-bit parallel prefix kogge-stone adders in order to streamline the hardware of the standard adders. In this project, a customised 64-bit parallel prefix kogge-stone adder with less hardware complexity than a standard 64-bit parallel prefix kogge-stone adder is designed and simulated using XILINX ISE 14.7.

1. INTRODUCTION
Every input in a prefix problem influences every output whose magnitude is equal to or greater, and every output whose magnitude is equal to or lower. The associative attribute of the prefix- operator allows for the execution of each operation in any order. In particular, group variables can be created by combining procedures to partially and simultaneously solve the prefix problem for sets of input bits. When group variable sequences are evaluated once again at higher levels, intermediate group variable levels are created, where the group variable reflects the prefix outcome of bits at level l. The last level's group variables must include all bits up to 0 and consequently reflect the solutions to the prefix problem. The most often used redundant arithmetic adders, carry-save adders, are crucial for the effective implementation of multi operand addition circuits. Due to the lack of carry-propagation channels and the simplicity of their construction, they are highly quick, but there is little room for further optimisation. The same is true for signed-digit adders, which employ an alternative redundant number representation. However, in order to proceed with processing the addition results, they typically need to be transformed into an irredundant integer representation. A carry-propagate adder is used for this process. The parallel prefix adder (PPA) is thought to be the most efficient adder for adding two multi-bit values..Circuit complexity and PPA speed are crucial factors when it comes to effective hardware implementation, hence numerous PPA variants with diverse parameter characteristics have been created in recent years. Three stages—preprocessing stage, prefix computation stage, and final processing stage—are used to compute the mechanism of PPA[1]. The well-known parallel prefix tree adders include Sklansky, Han-Carlson, Brent-Kung, and Kogge-Stone. The literature revealed that, when compared to other adders, the Kogge-stone adder is the quickest. In terms of worst-case latency, Ripple-Carry, Carry-Look-Ahead, Carry Select, and Kogge-Stone are shown to have the highest adder priorities. This is because there are so many "Reduced stages."The easiest to implement of all tree adders, the Kogge-Stone adder also has one of the shortest critical paths. The Kogge Stone adder's implementation has the disadvantages of taking up a lot of space and requiring more intricate connection routing (Fan-Out).

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 12, December 2022

2.LITERATURE SURVEY

To mix binary data and calculate the sum, adders employ several logic gate configurations. The subgrouping of the adders is based on how well they can take and combine the digits. In microprocessors, DSPs, mobile devices, and other high speed applications, parallel addition is performed using parallel-prefix adders. The performance is improved by factors like latency and power thanks to the Parallel Prefix Adders' decrease in logic complexity and delay. The Parallel- Prefix adders are therefore the appropriate component in the high speed arithmetic circuits. The carry chain is the main issue when utilising RCA for binary addition. The length of the carry chain rises as the input operand width does as well. When the carry takes the longest route feasible from the Least Significant Bit (LSB) to the Most Significant Bit (MSB), this is the worst case scenario that can happen. We are thinking about using a carry look ahead adder to propagate the carry ahead or lessen the RCA delay. Basically, propagates and generates are the two operations this kind of adder uses. The intricacy of the carry rises as the adder bit width increases. Therefore, designing higher bit CLA becomes complex. There is a concept known as the Parallel Prefix Approach that may be used to quickly and simply compute the carry in the preceding.

EXISTING METHOD PARALLEL PREFIX ADDER

The parallel prefix adder (PPA) is a multi-bit carry-propagate adder that combines two multi-bit values in parallel. PPA accelerate adding by extending the produced and propagated logic of the carry look-ahead adder [4]. The pre-processing stage, prefix computation stage, and final processing stage comprise the three steps of the fundamental schematic structure of the different PPA, perspective architectures (Fig. 2.1)[5]. Let's take a closer look at each stage.

OPERATION OF PARALLEL PREFIX ADDER

At the pre-processing stage, carry-generatedescribedbythefollowingcorrespondinglogicalequations:

$g_i=A_i\&B_i$;i=0,1,2,...,n−1 (1.1)

$h_i=A_i\hat{}B_i$;i=0,1,2,...,n−1 (1.2)

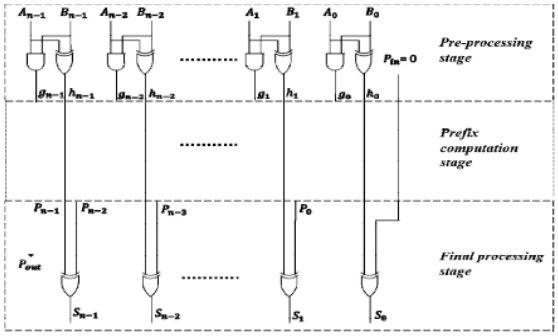

Fig1: Architecture of parallel pre fixadder

3. PROPOSED METHOD

The implementation of today's apps is accelerating at an unstoppable rate.

The modules that support those should be replaced when the bus interface protocols and the width of the peripherals change. Since whatever implementation is used must also support legacy devices, it should be expanded to include next-generation applications and devices. Although the design is good for the applications, the industry does not support it if the implemented design lacks these features. In this project, the modified 32 bit parallel prefix adder is expanded to 64 bit. because the majority of

processors used today have 64 bits. The 32 bit adder requires two cycles to operate on that processor/application, increasing the overall duration and lowering system performance.

MODIFIED 64-BITPARALLEL PREFIX KOGGE-STONEADDER

When compared to a 32 bit parallel prefix adder, the modified 64 bit parallel prefix kogge stone increases. The updated 64-bit parallel prefix adder's block structure is illustrated in Fig. 3.1 below; it has six levels, labelled level L1, level L2, level L3, level L4, level L5, and level L6, and the operation is carried out in three stages: preprocessing, prefixing, and final processing.

OPERATION OF MODIFIED 64-BIT PARALLEL PREFIX KOGGE-STONE ADDER

Three phases make up the improved 64-bit parallel prefix adder's operation. This adder's two inputs are denoted by the symbols A0,A1,A2.....A63 and B0,B1,B2.....B63, respectively. Preprocessor stage is the initial step. In the preprocessing stage, carries are generated and propagated while adding two adjacent bits from two 64-bit inputs, namely (A0,B0),(A1,B1),....(A63,B63). Prefix computation stage is the second stage. Six stages of addition are performed in the prefix computation step. 63 black cells and 1 grey cell make up Level L1.A grey cell performs the addition of bits A1 and B1, and all subsequent additions of other bits are completed by black cells. In the level L2 two grey cells and thirty black cells are grouped together to form two schematic nodes. In the first grey cell, the bits (A2,B2) and the carry from the addition of (A0,B0) in level L1 are added. In the second grey cell, the bits (A3,B3) and the carry from the earlier addition of the bits (A1,B1) in level L1 are added.Similar to this, the bits (A6,B6), (A7,B7), (A10,B10), and (A,B).....(A,B), (A,B), (A,B), (A,B), adder is used to accommodate 32-bit apps, and (A,B) are inserted in black cells alongside prior 64-bit applications. Here, the number of blocks is less than in the standard kogge-stone adder and one stage group carry generated in level L1's earlier stages, respectively.

Four schematic nodes are created at level L3, and addition is performed by skipping four bits. Four grey cells and 28 black cells make up level L3. The carry produced from (A2,B2) at level L2 is combined with the bits (A4,B4) in the first grey cell.The bits (A5,B5), (A6,B6), and (A7,B7) are combined with the carry from level L2's (A3,B3) and (A2,B2) to fill the remaining grey cells. Similar to this, carry created from earlier bits in level L2 is inserted in 20 black cells together with (A12,B12), (A13,B13), (A14,B13), (A15,B15), (A20,B20), (A21,B21), (A22,B22), (A23,B23), and (A60,B60), (A61,B61), (A62,B62), and (A63,B63).

8 schematic node groups of level L4 are constructed. By omitting eight bits, the addition is completed. There are 24 black cells and 8 grey cells at level L4. The carries produced from earlier bits are added to the bits (A8,B8), (A9,B9), (A10,B10), (A11,B11), (A12,B12), (A13,B1 3), (A14,B14), and (A15,B15) at level L3.Similar group carries created from earlier bits in level L3 are inserted in black cells together with (A24,B24)....(A31,B31), (A40,B40), (A47,B47), (A56,B56), and (A63,B63).

Groups with up to 16 nodes are established at level L5. The increase is carried out by skipping 16 bits in the level L5's 16 grey and 16 black cells. The carry produced from earlier bits in level L4 are inserted in sixteen grey cells together with the bits (A16,B16), (A17,B17),.....(A31,B31). In sixteen black cells, the bits (A48,B48), (A49,B49),... (A63,B63), along with carries produced from the previous bits in level L4 in the appropriate places, are added. When 32 bits are grouped, level L6 is created. There are 32 grey cells in it. In level L5, carries produced from earlier bits are combined with the bits (A32,B32), (A33,B33), and (A63,B63) in 32 grey cells.

The sums created by the bits skipped at each step are transmitted to the final processing stage in the third stage, where they are added to the carry P0, P1, and P63 using an XOR gate-based circle. S0, S1, and so on through S63 are the outputs of XOR gates.In this manner, a modified 64-bit parallel prefix

adder is used to create the total and carry.

IMPLEMENTATION OF64-BIT PARALLEL PREFIX KOGGE STONE ADDER

Modified 8 bit parallel prefix kogge stone adder implementation makes it simple to comprehend modified 64 bit parallel prefix kogge stone adder implementation. Three layers make up the modified 8 bit parallel prefix kogge stone adder. The modified 8 bit parallel prefix kogge stone adder circuit diagram is equivalent to 1/8th of the modified 64 bit parallel prefix adder circuit schematic. The modified 64-bit parallel prefix kogge stone adder consists of 64 group carries, numbered C0 to C63, which are run concurrently in the prefix stage of the adder's second stage utilising the sums and carries produced in the preprocessing stage of the adder's first stage. All of these group carries are then added in the final processing stage. The modified 8 bit parallel prefix kogge stone adder is used to describe the carries execution from C0 to C8. Fig. 4.1 displays the internal circuit diagram of the modified 8 bit parallel prefix kogge stone adder.

The white cells in the preprocessing step are arranged in 8 columns, numbered 0 to 7.Take two 8-bit inputs a0, a1, a2, a 3, a 4, a 5, a 6, and a 7 into consideration. Sums and carries are created during the preprocessing stage in the same

A0xb0=g0 in column 0, where a0xor b0=h0,

In column 1, a1xorb1=h1, a1xb1=g1, in column 2, a2xorb2=h2, a2xb2=g3, in column 4, a4xorb4=g4, in column 5, a5xorb5=g5, in column 6, a6xorb6=g6, and in column 7, a7xorb7=h7, a7xb7=g7.

Stage2: prefix calculation stage

In the second stage group carries are generated, the group carries are

$C_0=g_0$

$C_1=g_1+h_1xC_0=g_1+h_1g_0C_2=g_2+h_2xC_1=g_2+h_2(g_1+h_1xg_0)=g_2+h_2g_1+h_1xh_2xg_0C_3=g_3+h_3xC_2=g_3+h_3(g_2+h_2g_1+h_1h_2g_0)=g_3+h_3g_2+h_3h_2g_1+h_3h_1\ h_2g_0C_4=g_4+h_4xC_3=g_4+h_4(g_3+h_3g_2+h_3h_2g_1+h_3h_1h_2g_0)=g_4+h_4g_3+h_4h_3g_2+h_4h_3h_2g1+h_4h_3h_2h_1g_0C_5=g_5+h_5xC_4=g_5+h_5(g_4+h_4g_3+h_4h_3g_2+h_4h_3h_2g1+h_4h_3h_2h_1g_0)$

$=g_5 + g_4\ h_5+ h_5h_4g_3+ h_5h_4h_3g_2+ h_5h_4h_3h_2g1+$

$h_5h_4h_3h_2h_1g_0C_6=g_6+h_6xC_5=g_6+h_6(g_5+g_4h_5+h_5h_4g_3+h_5h_4h_3g_2+h_5h_4h_3h_2g1+h_5h_4h_3h_2h_1g_0)=g_6+h_6g_5+h_6g_4h_5+h_6h_5h_4g_3+h_6h_5h_4h_3g_2+ h_6h_5h_4h_3h_2g1+$

$h_6h_5h_4h_3h_2h_1g_0C_7=g_7+h_7xC_6=g_7+h_7(g_6+h_6g_5+h_6g_4h_5+h_6h_5h_4g_3+h_6h_5h_4h_3g_2+h_6h_5h_4h_3h_2g1+h_6h_5h_4h_3h_2h_1g_0)$

$=g_7+h_7g_6+h_7h_6g_5+h_7h_6g_4h_5+h_7\quad h_6h_5h_4g_3+ h_7h_6h_5h_4h_3g_2+\ h_7h_6h_5h_4h_3h_2g1+h7h6h5h4h3h2h1g0$

These carries generated parallel in three levels level1, level2, level3 using7 columns.

Level1 results: The results of level1 are denoted as G[i:K] where I denotes the present column and k Denotes the previous column from where carry is

$G[5:4]=g_5+h_5g_4$ and $H[5:4]=H[5:0]=h_5Xh_4G[6:5]=g_6+h_6g_5$ and $H[6:5]=H[6:0]=h_6xh_5G[7:6]=g_7+h_7g_6$andH[7:6]=H[7:0]=$h_7xh_6$

Hence the group carries Co and C1 are generated inthelevel1and are denoted as G[0:0] and G[1:0]which are called as prefixes and the final sums are also generated which are added to these prefixes in final stage.

Level2 results: the results of level2 are also denoted as G [i:K] but the execution starts from 2nd column.

$G[2:0]=G[2:1]+H[2:1]g_0=g_2+h_2g_1+h_1h_2g_0=C_3=G[2:0]$andH[2:0]=H[2:1]G[3:1]=G[3:2]+H[3:2]Xg[1:0]=$g_3+h_3g_2+(h_3xh_2)(g_1+h_1xg_0)$

$=g_4+h_4g_3+h_4h_3g_2+h_4h_3h_2g1+h_4h_3h_2h_1g_0=C4=G[3:0]$andH[3:1]=H[3:2]=H[3:0]G[6:4]=G[6:5]+H[6:5]xG[4:3]=$g_6+h_6g_5+(h_6h_5)(g_4+h_4xg_3)=$

$g_6+h_6g_5+h_6h_5g_4+h_6h_5g_3$ and

H[6:4]=H[6:5]h_4=h_6xh_5xh_4G[7:5]=G[7:6]+H[7:6]xG[5:4]=g_7+h_7xg_6+(h_7xh_6)(g_5+h_5xg_4
)=g_7+h_7xg_6+h_7xh_6xg_5+h_7xh_6xh_5xg_4 andH[7:5]=H[7:6]xh_5=h_7xh_6xh_5
Thusinlevel2thegroupcarriesC3andC4aregenerated which are denoted as G[2:0] and G[3:0]theseare also called prefixes.

Level 3 results: In level 3 the execution starts fromthe4thcolumntakingthecarryof2ndcolumngenerated in the level 2. Thus for the 4th bit executionthe carry is generated at 2nd bit itself which results inlessdelay.
G[4:2]=G[4:3]+H[4:3]xG[2:0]=g_4+h_4xg_3+(h_4xh_3)(g_2+h_2xg_1+h
takenforexecution.Itusestheg$_i$generated inpreprocessingstage.
andh$_i$
values
$_1$xh_2xg_0)=g_4+$h_4$$g_3$+$h_4$$h_3$$g_2$+$h_4$$h_3$$h_2$g1+$h_4$$h_3$$h_2$$h_1$$g_0$=
C4=G[4:0]
G[0:0]=C=gand H[0:0]=h
G[5:3]=G[5:4]+H[5:4]xG[3:0]= g_5+h_5xg_4+($h_5$$h_4$)(
0 0 0
G[1:0]=g+hg=C1 and H[1:0]=h
g_4+$h_4$$g_3$+$h_4$$h_3$$g_2$+$h_4$$h_3$$h_2$g1+$h_4$$h_3$$h_2$$h_1$$g_0$)=$g_5$+$g_4$$h_5$+$h_5$$h_4$$g_3$+
1 10 1
G[2:1]=g+hgand H[2:1]=H[2:0]=hxh
$h_5$$h_4$$h_3$$g_2$+$h_5$$h_4$$h_3$$h_2$g1+$h_5$$h_4$$h_3$$h_2$$h_1$$g_0$=C5=G[5:0]
2 21 1 2
G[3:2]= g_3+$h_3$$g_2$and H[3:2]=H[3:0]=$h_3xh_2$G[4:3]=$g_4$+$h_4$$g_3$andH[4:3]=H[4:0]=$h_4xh_3$
G[6:2]=G[6:4]+H[6:4]xG[2:0]=g_6+h_6xg_5+h_6xh_5xg_4+h_6xh_5x
g_3+(h_6xh_5xh_4)x(g_2+h_2xg_1+h_1xh_2xg_0)=g_6+$h_6$$g_5$+$h_6$$g_4$$h_5$+$h_6$$h_5$$h_4$$g_3$+ $h_6$$h_5$$h_4$$h_3$$g_2$+ $h_6$$h_5$$h_4$$h_3$$h_2$g1+
$h_6$$h_5$$h_4$$h_3$$h_2$$h_1$$g_0$ =C6=G[6:0]G[7:3]=G[7:5]+H[7:5]xG[3:0]=g_7+h_7xg_6+h_7xh_6xg_5+h_7xh_6x
h_5xg_4+(h_7xh_6xh_5)x(g_4+$h_4$$g_3$+$h_4$$h_3$$g_2$+$h_4$$h_3$$h_2$g1+$h_4$$h_3$$h_2$$h_1$$g_0$)=
g_7+$h_7$$g_6$+$h_7$$h_6$$g_5$+$h_7$$h_6$$g_4$$h_5$+$h_7$$h_6$$h_5$$h_4$$g_3$+ $h_7$$h_6$$h_5$$h_4$$h_3$$g_2$+hhhhhhg1+hhhhhhhg=c7=G7:0
generated from P_{16} to P_{31}, in level 6 remaining 32 aregenerated from P_{32} to P_{63} and these all are added infinal processing stage and 64 outputs S_0,S_1 to S_{63} areobtained.
Theimplementationofmodified64-bit
parallelprefixadderisdoneusing XilinxVerilog
7 6 54 32 7654 32 10
Thus in level 3 the carries C_4,C_5,C_6,C_7 are generated anddenoted as G[4:0],G[5:0],G[6:0], G[7:0].
Stage3:Finalprocessingstage
G[0:0],G[1:0],G[2:0],G[3:0],G[4:0],G[5:0],G[6:0],G[
7:0]obtainedfromsecondstagearecalledasprefixesdenotedasP_0,P_1,P_2,P_3,P_4,P_5,P_6,P_7respectivelyandthesea readdedtofinalsumsgenerated in level 1 of second stage in the third stagei.e,final processingstage.
InthefinalprocessingstageainputcarryP_{in}=0denotedasP_{-1}isgivenandthefinaloutputsarecalculated asshownbelow
S_0=H[0:0] XOR $P_{-1}$$S_1$=H[1:0] XOR $P_0$$S_2$=H[2:0] XOR $P_1$$S_3$=H[3:0] XOR $P_2$$S_4$=H[4:0] XOR
$P_3$$S_5$=H[5:0] XOR $P_4$$S_6$=H[6:0] XOR $P_5$$S_7$= H[7:0] XOR P_6S8=P7=POUT
Thusthefinalresultsareobtained.
In the case of modified 64 bit, the same procedure applies since the levels of the bits and the number of columns will both rise. Up to 8 columns operate similarly to modified 8 bit ppa prefixes P0 to P7 are generated using 3 levels, remaining 8 carries starting from P8 to P15 are generated in level 4, and the

final 16 carries are HDL software. The modified 64 bit ppa is made up of 64 columns ranging from 0 to 63, 6 levels from level 1 to level 6, and 64 carries from C0 to C63.

4. RESULT ANALYSIS

Before the design could be tested for accuracy by walking practical simulation or downloading into the prototype board, it needed to be synthesised and put to use.

a. For cell-based designs, carry-skip, carry-select, and prefix adders have been compared qualitatively and quantitatively.
b. This design is an alternative to carry-save adders with the option of having a constant or variable binary number as the third operand.
c. The suggested method performs on par with a multi-operand conventional adder.
d. Because the new logic is integrated into a standard adder, there is no longer a requirement for specialised adder units to carry out the operation.
e. Despite its high power and area consumption, the Kogge-Stone adder performs well in terms of speed.
f. In order to examine the impact and potential performance increase of the adder, it will be applied in real-world application designs like decimal math and multiplier units.

Fig 2: RTL schematic window

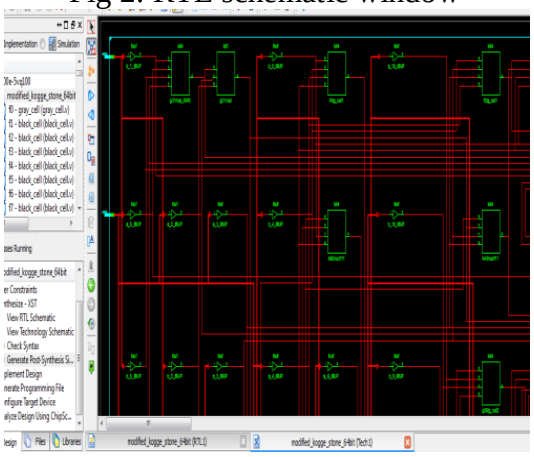

Fig3; Technology schematic window

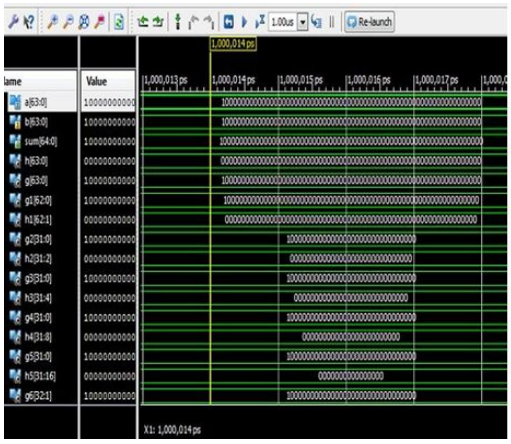

Fig 4: simulation window

CONCLUSION

We explore parallel prefix adders, which are highly favoured in arithmetic-logic components of contemporary processors including digital signal processors and microprocessors. This research looks at the fastest parallel prefix adder, the conventional 64-bit kogge-stone adder. The hardware complexity of the design makes it difficult to implement. A modified 64-bit parallel prefix Kogge-stone adder is created in this project.

Starting at level 2, the pairs of columns are grouped starting with level 2 (i.e., two pairs of columns are grouped in level 2 and carry propagated only from those groups)., While in the standard 64 bit parallel prefix adder no such grouping is done and carries are generated and propagated from every column, which requires extra gates to generate and propagate these carries from every column, level 3 groups four pairs of columns, level 5 groups eight pairs of columns, and level 6 groups 16 pairs of columns, all of which are grouped and prefixes are calculated. Thus, the hardware complexity of the standard 64 bit kogge stone adder was reduced by the modified 64 bit parallel prefix kogge adder. Verification is done on the simulation outcomes of a modified 64-bit parallel Kogge-Stone adder.

REFERENCES

[1] Geeta Rani, Sachin Kumar.―Delay Analysis of Parallel-Prefix Adders‖. International Journal of Science and Research (IJSR), ISSN: 2319-7064,Impact Factor(2012):3.358.Volume3Issue6,June, 2014.pp.2339.

[2] Sunil.M,Ankith.R.D,Manjunatha.G.DandPremananda.B.S.Designandimplementationoffasterparallelprefix KoggeStoneadder.InternationalJournalofElectricalandElectronic Engeering&Telecommunications2014.ISSN2319 – 2518.Vol.3,No.1,January2014.pp. 116.

[3] AungMyoSan,AlexeyN.Yakunin–Reducing the Hardware Complexity of a Parallel Prefix Adder‖ IEEE2018pp.1350

[4] Athira.T.S, Divya.R, Karthik.M, Manikandan.A.DesignofKogge-Stoneforfastaddition.Proceedings of 34th IRF International Conference,26th February 2017, Bengaluru, India. ISBN: 978-93-86291-639.pp. 27-28.

[5] CH.Sudha,Rani,CH.Ramesh.DesignandImplementation of High Performance Parallel PrefixAdders.InternationalJournalofInnovativeResearchinComputerandCommunicationEngineering. AnISO3297:2007CertifiedOrganization. Vol.2, Issue 9, September 2014. pp.5900.

[6] VibhutiDave.High-speedmultioperandaddition utilization flag bits. Submitted inpartialfulfillmentoftherequirementsforthedegreeofDoctorofPhilosophyinComputerEngineering,Chicago.Illinois. May2007. pp. 38-39.

[7] RaghumanoharAdusumilli,VinodkumarK –DESIGNANDIMPLEMENTATIONOFAHIGH SPEED64BITKOGGE-STONEADDERUSING VERILOG HDL‖ International journal of electricaland electronic engineering & Telecommunications(IJEETC) ISSN 2319_2518, vol.41,January 2015pp.15-16.