FLOATING POINT ADDITION USING LOW POWER CSA

DILJITH MURALY1, SUNIL JACOB2, MR. K PADMAKUMAR3
1M.Tech, VLSI and Embedded Systems, 2, 3 Associate Professor, Electronics and Communication
SCMS School of Engineering & Technology, Karukutty, Ernakulam1,2
Scientist sf, fcd/avn, VSSC, ISRO, Trivandrum3
1Dilvichu143@gmail.com, 2suniljacob01@gmail.com

ABSTRACT
The floating-point representation covers a wider range of values compared to a fixed-point representation. Due to the widespread use and inherent complexity of floating-point addition, much effort has been devoted to its speedup via algorithmic and circuit techniques. Even simple floating-point operations consume a large amount of computational resources. Floating points achieve a high domain - from very small numbers close to zero to very high numbers, sometimes even higher than the number of atoms in the universe. This assures accuracy to a higher level. Selection of suitable adder for floating point numbers is of major importance. Power dissipation is one of the most important design objectives in integrated circuits, especially in software that performs large scale mathematical calculations. In computing, floating point describes a method of representing an approximation of a real number. The term floating point refers to the fact that the radix point (decimal point, or, more commonly in computers, binary point) can "float"; that is, it can be placed anywhere relative to the significant digits of the number. The speed of floating-point operations, commonly referred to in performance measurements as FLOPS, is an important characteristic of a computer system, especially in software that performs large-scale mathematical calculations. Floating-point representation the most common solution basically represents real’s in scientific notation. Scientific notation represents numbers as a base number and an exponent. Floating point describes a system for representing numbers that would be too large or too small to be represented as integers. Numbers are in general represented approximately to a fixed number of significant digits and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form: Significant digits × base exponent

INTRODUCTION
Floating-point solves a number of representation problems. Floating-point employs a sort of "sliding window" of precision appropriate to the scale of the number. This allows it to represent numbers from 1,000,000,000,000 to 0.0000000000000001 with ease. The advantage of floating-point representation over fixed-point (and integer) representation is that it can support a much wider range of values. Floating-point representation is that of an exponent. Floating point describes a system for representing numbers that would be too large or too small to be represented as integers. Numbers are in general represented approximately to a fixed number of significant digits and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form: Significant digits × base exponent

FLOATING POINT REPRESENTATION OVERVIEW
In this project the Floating point unit used is,

FLOATING POINT ADDITION

The 16 bit micro controller is based on the existing architecture of VIKRAM 1601 processor. VIKRAM 1601 is a 16-bit processor which supports floating point operations. Since these floating point operations are implemented in micro code it takes longer time for computation. So to improve the computational speed of the processor, it was suggested to realize as hardwired unit which supports all the floating point computations. Thus with this implementation the execution time will be drastically reduced so that enough timing margin is there.

[1] INTRODUCTION
Floating-point solves a number of representation problems. Floating-point employs a sort of "sliding window" of precision appropriate to the scale of the number. This allows it to represent numbers from 1,000,000,000,000 to 0.0000000000000001 with ease. The advantage of floating-point representation over fixed-point (and integer) representation is that it can support a much wider range of values. Floating-point representation is that of an exponent. Floating point describes a system for representing numbers that would be too large or too small to be represented as integers. Numbers are in general represented approximately to a fixed number of significant digits and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form: Significant digits × base exponent

The term floating point refers to the fact that the radix point (decimal point, or, more commonly in computers, binary point) can "float"; that is, it can be placed anywhere relative to the significant digits of the number. The speed of floating-point operations, commonly referred to in performance measurements as FLOPS, is an important characteristic of a computer system, especially in software that performs large-scale mathematical calculations. In computing, floating point describes a method of representing an approximation of a real number in a way that can support a wide range of values. Floating-point values are not uniformly spaced. Small values, close to zero, can be represented with much higher resolution (e.g. one femtometre) than large ones because a greater scale (e.g. light years) must be selected for encoding significantly larger values. That is, floating-point numbers cannot represent point coordinates with atomic accuracy at galactic distances, only close to the origin. In this project different types of adders are compared for the floating point addition. Carry Select Adders [1] are used for the addition in this project due to the less power consumption and area. The result is implemented in modelsim by using VHDL.

In this paper Floating point representation and different categories of floating point numbers and advantages are discussed in chapter 2. The comparison of different adders is in chapter 3. And a brief explanation about CSA is in chapter 4.

[2] FLOATING POINT ADDITION

In this project the Floating point unit used is,

[2.1] Floating point unit
The 16 bit micro controller is based on the existing architecture of VIKRAM 1601 processor. VIKRAM 1601 is a 16-bit processor which supports floating point operations. Since these floating point operations are implemented in micro code it takes longer time for computation. So to improve the computational speed of the processor, it was suggested to realize as hardwired unit which supports all the floating point computations. Thus with this implementation the execution time will be drastically reduced so that enough timing margin is there.
VIKRAM 1601 can support fixed point architecture as 16 bit unsigned fraction, 16 bit unsigned integer, 32 bit unsigned integer and floating point architecture as 32 bit real and 48 bit extended real operation. This paper mainly discussed about the 32 bit real addition operations.

[2.2] Floating point format

VIKRAM 1601 architecture supports two types of floating point numbers, viz 32 bit real and 48 bit extended real number. Both formats use biased exponents with bias value of 128. In VIKRAM 1601 processor all the floating point operations are carried out with normalized numbers. The final result after the floating point operation is always rounded to nearest even number technique. The value V of a floating point number with magnitude M and biased exponent E is given by:

\[ V = M \times 2^{E-128} \]

In the above floating point format, the exponent +127 is used to represent infinity. In the closed arithmetic environment, infinity is necessarily excluded. Therefore, in the event of computational overflow the exponent of the result is saturated to the value +126 and the AE (arithmetic error) flag of the status register is set to indicate arithmetic error. During an underflow condition, the exponent of result is \(<= -128\). In this condition the result is approximated to floating zero and the AE flag is not affected.

[2.3] Rounding

The accuracy of results obtained in floating point arithmetic unit is limited even if the intermediate results calculated in the arithmetic unit are accurate. The number of computed digits may exceed the total number of digits allowed by the format, and we have to dispose of the extra digits before the final results are stored in user-accessible register or in the memory. When selecting a round-off scheme we need to consider the following factors:

- Accuracy of the results (numerical consideration)
- Cost of implementation

The entire rounding schemes present round to nearest even number technique. This rounding scheme requires three bits. First bit is the LSB of the word (L), the next bit is considered as the round bit (R) and the last one is the sticky bit (S) which is the logical OR of all the remaining bits to be truncated. As per this rounding rule a 1 has to be added to the LSB of the data to be preserved if and only if R and S (or L) is 1. With this rounding scheme the total bias from the original value will be very less compared to other rounding schemes.

[2.4] Overflow and Underflow

Overflow occurs when the computed results happens to be outside the represent able range of the data type. In the present architecture followed an overflow occurs when the final exponent attains a value greater than 254. Whenever a data attains a value above this range a corrective action is provided by setting the result to the maximum value which in turn depends on the sign of the result. For example, if we are multiplying two numbers with opposite signs then in the case of overflow during computation, the result would be set to the maximum – ve value which in our case, will be 800000FEh. If two positive numbers are being multiplied then in overflow condition the result would be clamped to 7FFFFFeH. In addition to this clamping of the result an indication of this is set by setting arithmetic error (AE) in the status register of the processor. Underflow occurs when the computed result is very small to be represented but the data format. This happens when the exponent of the result is less than -. In such a condition the corrective action is to set the result to zero and carry on with the next instruction. In this case AE register will not be set.

[2.5] Floating point addition

The floating point addition is done using normalized input values. The floating point addition hardware is more complex than their arithmetic circuits [3]. This is due to different data flow paths that occur with respect to different set of operands.

[3] FLOATING POINT ADDITION

Floating point addition is done using normalized input values. The floating point addition hardware is more complex than their arithmetic circuits [3]. This is due to different data flow paths that occur with respect to different set of operands.

Addition of floating point numbers involves the pre-alignment, addition, normalization and rounding of significant as well as exponent evaluation. Significant pre-alignment is a pre-requisite for addition. In floating point additions, the exponent of the larger number is chosen as the tentative exponent of the result. Exponent equalization of the smaller floating point number to that of the larger number demands the shifting of the significant of the smaller number through an appropriate number of bit positions. The absolute value of the difference between the exponents of the numbers decides the magnitude of alignment shift. Addition of significant is essentially a signed magnitude addition, the result of which operation is also represented in signed-magnitude form. Signed-magnitude addition of significant can lead to the generation of a carry out from the...
MSB position of the significant or the generation of leading zeros or even a zero result. Normalization shifts are essential to restore the result of the signed-magnitude significant addition into standard form. Rounding of normalized significant is the last step in the whole addition process. Rounding demands a conditional incrementing of the normalized significant. The operation of rounding, by itself can lead to the generation of a carry out from the MSB position of the normalized significant. That means, the rounded significant need be subjected to a correction shifting in certain situations [4].

[3.1] Addition algorithm
Assuming that, the operands are floating point. The standard proposed for this architecture, performing floating point addition. Result= (X+Y) = X_m x 2^X + Y_m x 2^Y
Involves the following steps:
Align the binary point:
• Initial result exponent: larger of the X_E, Y_E
• Compute the exponent difference: Y_E - X_E
• If X_E < Y_E, shift X_M right by the abs(difference)
• If X_E > Y_E, shift Y_M right by the abs (difference)
• After addition check whether the result is in the normalized shift left form, otherwise do the normalization.
Normalization steps:
• Left shift the result, if the mantissa is of the form 0.0001XXXX….. till it becomes 0.1XXXX (Decrement the exponent)
• Right shift the result, if the mantissa is of the form 10.1XXXX…. till it becomes 1.0XXXX (increment the exponent)
• Check the final exponent and look whether any overflow or underflow has occurred during this normalization phase.
• Round the significant back to 24 bit and see whether any exponent adjustment is required to account for normalization after rounding. (round to nearest even is used for rounding)
• If the mantissa has become zero, then set the exponent to zero by a special step to generate proper zero.

[4] COMPARISON OF ADDERS
Existing different types of adders are:

[4.1] Ripple Carry Adder
The ripple carry adder is constructed by cascading full adders (FA) blocks in series. One full adder is responsible for the addition of two binary digits at any stage of the ripple carry. The carryout of one stage is fed directly to the carry-in of the next stage. This configuration is called ripple carry adder, since the carry bit—ripples from one stage to the other. A number of full adders may be added to the ripple carry adder or ripple carry adders of different sizes may be cascaded in order to accommodate binary vector strings of larger sizes. For an n-bit parallel adder, it requires n computational elements (FA). A 4-bit ripple-carry adder composed of four full adders. The augends’ bits of x are added to the addend bits of y respectfully of their binary position. Each bit addition creates a sum and a carry out. The carry out is then transmitted to the carry in of the next higher-order bit. The final result creates a sum of four bits plus a carry out (c4).
Advantages
• This is a simple adder
• Provides easy connections with neighboring circuits
• Can be used to add unrestricted bit length numbers
Disadvantages
• Propagation delay is very high as carry ripple through each stage and the sum will be generated only after previous carry is obtained.
• Delay increases linearly with the bit length.
• Not very efficient when large bit numbers are used.

[4.2] Carry Look Ahead Adder
The carry look-ahead adder solves the delay problem in ripple carry adder by calculating the carry signals in advance, based on the input signals. The result is a reduced carry propagation time. Carry look-ahead adder’s structure can be divided into three parts: the propagate/generate generator, the sum generator and the carry generator
The Propagate P and generate G in a full-adder, is given as:
P_i = A_i \oplus B_i \tag{1}
G_i = A_i B_i \tag{2}
Propagate and generate signals depend only on the input bits and thus will be valid after one gate delay. The new expressions for the output sum and the carryout are given by
\[ S_i = P_i \oplus C_{i-1} \]  

(3)

\[ G_{i+1} = G_i + P_i \]  

(4)

These equations show that a carry signal will be generated in two cases: If both bits \( A_i \) and \( B_i \) are 1 and If either \( A_i \) or \( B_i \) is 1 and the carry-in \( C_i \) is 1.

### 4.3 Carry Skip Adder / Carry Bypass Adder

A carry-skip adder consists of a simple ripple carry-adder with a special speed up carry chain called a skip chain. This chain defines the distribution of ripple carry blocks, which compose the skip adder. A carry skip chain is added in parallel to each adder block. The carry need not be passed through the entire adder block in all cases. In certain the carry can be bypassed to next stages via carry skip chain.

**Advantages**
- Improved speed for wider-bit adders
- Area overhead due to bypass circuitry
- Increased capacitance.

**Disadvantages**
- Harder to implement than other designs

### 4.4 Manchester Adder

A Manchester carry adder consists of cascaded stages of Manchester propagation cells. The optimum amount of cascaded stages may be calculated for a technology by simulation. For a 16 bit adder example a 4-bit adder made up of four static stage cells is chosen in order to reduce the number of series-propagate transistors, which greatly improves speed. In the case of a four-bit adder, the maximum number of transistors that are in series with the gate, when all propagate signals and \( C_i \) is true, is only five. In addition to the cascaded Manchester propagation cells the adder requires carry propagation and carry generation logic, also called a PG generator. Finally to complete the design four XNOR blocks each of which produces the SUM at each particular stage is required.

**Advantages**
- Low delay; comparable to CSA.
- Faster than CRA, CLA.
- Small area; smaller than everything except CRA

**Disadvantages**
- Area overhead issues.

### 4.5 Carry select adder

It is composed of two four-bit ripple carry adders per section. Both sum and carry bits are calculated for the two alternatives of the input carry, —01 and —11. The carry out of each section determines the carry in of the next section, which then selects the appropriate ripple carry adder. The least significant block or block 0 is same as that of the ripple carry adder. For higher sections, each block is divided into two sections. For one section we assume \( C_i \) to be zero and for the other section \( C_i \) is assumed as one[2].

**Advantages**
- Propagation delay is reduced
- Sum bits are generated in parallel

**Disadvantages**
- Area overhead issues.

The CSLA is used in many computational systems to alleviate the problem of carry propagation delay by independently generating multiple carries and then select a carry to generate the sum. However, the CSLA is not area efficient because it uses multiple pairs of Ripple Carry Adders (RCA) to generate partial sum and carry by considering carry input and, then the final sum and carry are selected by the multiplexers (mux).
[5] PROPOSED DESIGN
The proposed design consists of a modified Carry select adder with a BEC. This paper also discuss about the area evaluation of the modified CSA.

[5.1] Modified CSA
The basic idea of this work is to use Binary to Excess-1 Converter (BEC) instead of RCA with Cin = 1 in the regular CSLA to achieve lower area and power consumption. The main advantage of this BEC logic comes from the lesser number of logic gates than the n-bit Full Adder (FA) structure[5].

The major problems associated with normal CSA is that increased area consumption and also the power consumption rate. The efficiency and speed of the design is very less as compared to the normal design. Also the delay factor plays a major role. A complete solution to all the above said problems can be given by using a BEC

[5.1.1] BEC
As stated above the main idea of this work is to use BEC instead of the RCA with in order to reduce the area and power consumption of the regular CSLA. To replace the n-bit RCA, an n-bit BEC is required. A structure and the function table of a 4-b BEC are shown in Fig. 2 and Table, respectively. Fig. 2 illustrates how the basic function of the CSLA is obtained by using the 4-bit BEC together with the mux. One input of the 8:4 mux gets as it input (B3, B2, B1, and B0) and another input of the mux is the BEC output. This produces the two possible partial results in parallel and the mux is used to select either the BEC output or the direct inputs according to the control signal Cin. The importance of the BEC logic stems from the large silicon area reduction when the CSLA with large number of bits are designed. The Boolean expressions of the 4-bit BEC is listed as (note the functional symbols NOT, & XOR)

![Fig 2.CSA Concept using BEC](image)

[5.2] Floating point addition using CSA
The existing system, output of first RCA given as the selection line of mux. If the carry output that is the mux selection line is 0 RCA1 output will select otherwise RCA2 will select. But this selection process cause many disadvantages. That is area to accommodate RAC is very high and it also increases the logic complexity.

In order to reduce the logic complexity RAC is replaced by BEC. The working of a single module of BEC is shown in figure 2. As that function, if the selection line of mux is 0 that is carry is 0 then the BEC output is considered otherwise RCA is considered. That is the proposed design will reduce the circuit complexity due to the usage of two RCA and power consumption also reduced.

[6] SIMULATION RESULT
Table 2 exhibits the simulation results of both the CSLA structures in terms of delay, area and power. The area indicates the total cell area of the design and the total power is sum of the leakage power, internal power and
switching power. The percentage reduction in the cell area, total power, power-delay product and the area–delay product as function of the bit size are shown in Fig. 4 (a). Also plotted is the percentage delay overhead in Fig. 4 (b). It is clear that the area of the 32b proposed SQRT CSLA is reduced by 16.7%. The total power consumed shows a similar trend of increasing reduction in power consumption 13.63%, with the bit size. Interestingly, the delay overhead also exhibits a similarly decreasing trend with bit size. Delay overhead for the 32-b is decreased by 6.7%. The power-delay product of the proposed 32-b is reduced by 12.28%. Similarly the area-delay product of the proposed design for 32-b reduced by 11%.

Fig. 4. Comparison graph

[6.1] Area comparison
Fig 5 Shows, area comparison of existing and proposed design. The result shows that, For Existing system, the total gate count for the system is 11,155 and For Modified system, Total gate count for the system is 10,461. From the above two simulation results it is clear that the area is reduced for the modified system by a considerable amount.

Table 2 Comparison of Simulation Result

[6.2] Power consumption comparison
Fig 6 shows, the power consumption comparison of existing and proposed design. The For Existing system, Estimated Power consumption: 109W and For Modified system, Estimated Power consumption: 97 W
CONCLUSION
Floating point addition requires precise calculations and it demands calculations with at most accuracy. Different from fixed point addition floating point addition is mostly preferred in day to day calculations that involve modern computers microprocessors etc. In more sensitive areas like military applications, space research centers advanced computing areas floating point representation has already made its mark. Different methods of carry select adders are compared; from the comparison it is found that the modified carry select adder using Binary to Excess One converter has reduced area, power consumption. But at the same time there is a slight increase in total delay. The reduced number of gates of this work offers the great advantage in the reduction of area and also the total power. The compared results show that the modified SQRT CSLA has a slightly larger delay (only 3.76%), but the area and power of the 64-b modified SQRT CSLA are significantly reduced by 17.4% and 15.4% respectively. The power-delay product and also the area-delay product of the proposed design show a decrease for 16-, 32-, and 64-b sizes which indicates the success of the method and not a mere tradeoffs of delay for power and area. The modified CSLA architecture is there-fore, low area, low power, simple and efficient for VLSI hardware implementation. It would be interesting to test the design of the modified 128-b SQRT CSLA. The modified CSLA architecture is therefore, low area, low power, simple and efficient for VLSI hardware implementation.

ACKNOWLEDGMENT
The authors would like to thank for the support of Department Electronics and Communication Engineering SSET, Karukutty, Ernakulum, India.

REFERENCES
[2]. “Power-Delay Product Minimization in High-Performance 64-bit Carry-Select Adders” AmauryNève, Member, IEEE, Helmut Schettler, Thomas Ludwig, Member, IEEE, and Denis Flandre, Senior Member, IEEE, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 12, NO. 3, MARCH 2004.