ECE 511 Project: Fast Low-Energy Approximate Adder for Error-Tolerant System

Hao Xuan
University of Alberta
hxuan@ualberta

***Abstract*—Approximate computing has already raised interest in various of areas due to its energy efficiency, high performance, and acceptable loss in accuracy. It is used by many systems and applications which tolerate minor input imperfection for different improvements. In this report, we review a few existing approximate arithmetic circuit designs. Then, algorithm-level improvements and corresponding simulations are provided.**

***Index Terms*—Adder, Approximate Computing, Error-Tolerance, Error Rate**

I. INTRODUCTION

For a long time, errors had been seen as defects of the system that should be avoided. However, nowadays, applications like multimedia processing, wireless communication, and data mining are capable of dealing imperfect inputs [1]. There are several reasons for this. Because of the human perceptual limitations, we cannot tell the difference of having a small amount of error in video, audio or image processing. Also, there are many existing applications that have error tolerant feature built inside such as signal processing units. By applying the approximate arithmetic circuit, a system can potentially reduce power dispensing like the low part OR adder (LOA) [2], improve circuit speed like the Speculative carry select adder (SCSA) [3], or achieve a relative high accuracy while been energy efficient and fast like the Carry skip adder (CSA) [4].

In this report, we will focus on reviewing the achievements made in approximate adder areas. Then we will propose a high-speed low-error approximate adder design. Also, a combination of transistor-level calculation and system-level simulation is provided to demonstrate the accuracy and performance the proposed adder is achieved.

II. REVIEW

In this section, three approximate adder designs will be reviewed. The first is the Error-Tolerant Adder [5]. It is a segmented adder based approximate adder. It contains sum generators and carry generators which work in a parallel structure to speed up the performance. The output of a carry generator will feed into the next sum generator which means the maximum length of the carry propagation chain is two block length. And with a proper choice of the block size, it will provide a decent accuracy with faster speed and less energy compare the conventional adder. However, this can be further improved by the Error-Tolerant Adder type IV (ETA IV) [5]. The improved version combines segmented adder with carry select feature. Every block of the ETA IV contains two parts. The first part is a normal parallel part contains a sum generator and a carry generator which has a 0 carry in signal. And the second part is a carry select structure where two carry generators with 0 and 1 carry in signal are connected to a multiplexer. And the output carry signal depends on the carry out signal from the first part. In this way, the most significant bit will now have longer carry propagation chain, thus higher accuracy. Also, the carry select structure with two parallel carry generators will reduce the circuit delay.

Next, we have the lower-part-OR adder (LOA) [6]. It is a simple structured approximate full adder design. The adder is divided into two parts. On the left, there is a conventional full adder, and on the right, there will be a OR-gate-based approximate adder where every output bit S_i will be produced only by $A_i\ OR\ B_i$. Normally, we will use the XOR gate to operate input bits before carry bit. However, since for the OR-gate based approximate adder, we do not have the carry propagation chain, OR gate will produce a closer result. For example, the correct answer for 1001 + 0101 would be 1110. With XOR gate, the output is going to be 1100 while 1101 with OR gate which is closer to the correct result. Moreover, OR gate will perform faster and more power efficient than XOR gate.

Finally, we have the Accuracy-configurable approximate adder (ACAA) [7]. It is a special structured approximate adder which consists of normal approximate adder part and error detection and correction part (EDC). The approximate adder part is a segmented adder with overlapped structure. The error detection circuit will use AND gates to check the carry propagation and generate corresponding error signal. The error signals will then go into the incrementor which performs the error correction function and produce error free output. However, going through one EDC will not give the completely correct result. Therefore, pipelined structure is introduced to potentially connect one and more EDC system to the adder for better accuracy output. This will give the adder ability to change output accuracy during runtime, switching between better accuracy mode and power efficient mode.

III. IMPROVE

In this section, a few improvement from the existing design is proposed. The basic concept of the approximate adder design is just to break the carry propagation chain to reduce the power consumption and speed up the computation process. However, it would be essential to find the proper length of the carry propagation chain which will result in an acceptable accuracy.

Table I shows the probability of length that a carry will propagate. As shown in the table, for 16-bit addition, the probability that a carry will only propagate for 4 bits is around 95%. That means if we can allow carry to propagate for at least 4 bits, a minimum accuracy of 95% can be achieved while the accuracy would be more complicate when we go further.

Table I. Probability of carry propagation length

Number of bit carry will propagate	1	2	3	4
Probability of one carry will propagate the length	0.53126	0.24999	0.11718	0.05468
Number of bit carry will propagate	5	6	7	8
Probability of one carry will propagate the length	0.02538	0.01171	0.00537	0.00244

The ETA type II seems to be a great starting point since it has a very standard segmented adder structure and overall accuracy is acceptable. We will build our base line model quite similar to ETAIIM as shown in Figure 1.

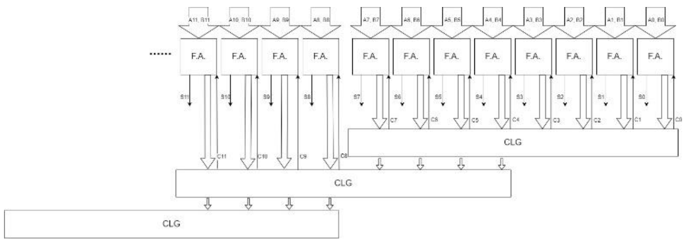


Figure 1: Block diagram of the proposed baseline adder

Furthermore, from experiment, we know that the accuracy of an approximate adder will be greatly increased if the more significant bits have a higher accuracy. Therefore, we will double the carry propagation chain length just for the left most block, so it can get extra carry propagation.

In theory, with longer carry propagation chain length, the circuit delay will be higher. The carry-lookahead structure will perfectly fix the speed problem while causes power consumption to increase since the carry-lookahead is a relatively complex structure. However, we will utilize the benefit of the simple-structured LOA and integrate it into the proposed adder. As shown in Figure 3, the final proposed adder will have OR-based sub adder at lower bits, and baseline adder with carry extension at higher bits.

However, there will be situations where the adder size cannot be perfectly divided by the block size. In that case, the left most block will have the size of $adder\ length\ \%\ block\ length$, where % stands for mod operation, in order to let the most significant bit to have more than two block length of carry propagation chain length, shown in Figure 2.

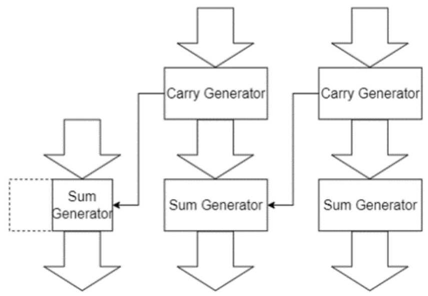


Figure 2: Alternate implementation of the left most block

Finally, it is very important to choose the correct block size for both the carry generators and the OR sub adder. We will use the empirical approach which will be explained in the next section.

IV. STIMULATION

Some definitions of the terminologies that will be used later:

- **Error**: $Error = |S_T - S_P|$, where S_T is the true result and S_P is the output produced by the approximate adder.
- **Accuracy**: $Accuracy = 1 - \frac{Error}{S_T}$
- **Minimum acceptable accuracy**: The minimum acceptable accuracy (MAA) is the minimum accuracy that a system can take. The result should produce a higher accuracy in order to allow the system or application to perform normally.

We will take 32-bit approximate adder for example and simulate for 100000 samples per group. And the probability of the approximate adder result reaching MMA will be recorded. The input range is from 0 to $2^{32} - 1$. The simulation is conducted using Python program [8].

Table II. Simulation result for sub adder size 0

Generator block size / MMA	2 bits	3 bits	4 bits	6 bits	8 bits
100%	0.2152	0.6247	0.8601	0.9776	0.9959
99%	0.7300	0.9416	0.9978	0.9994	0.9999
98%	0.7864	0.9494	0.9994	0.9999	1
97%	0.7983	0.9698	0.9997	0.9999	1
96%	0.8178	0.9825	0.9999	1	1
95%	0.8366	0.9897	0.9999	1	1
94%	0.8512	0.9921	0.9999	1	1
93%	0.8635	0.9941	1	1	1
92%	0.8763	0.9956	1	1	1
91%	0.8825	0.9966	1	1	1
90%	0.8887	0.9977	1	1	1

Due to this special design, choosing different block size can lead to very different accuracy. As an example of 32-bit addition, if we choose block size of 3 bits, the left most bit will

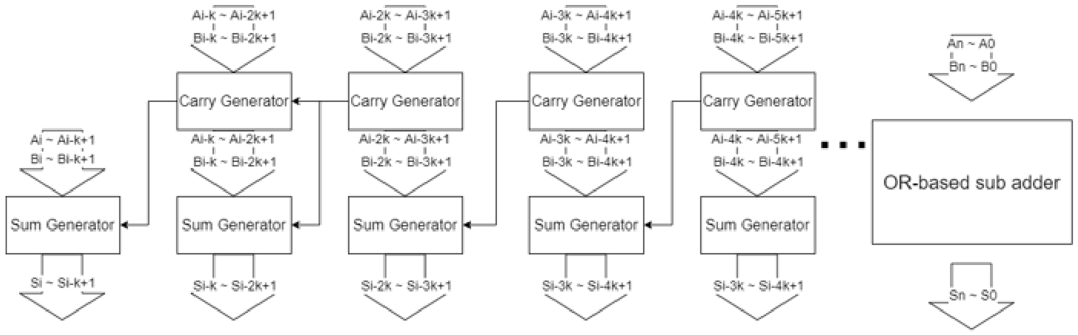


Figure 3: the improved version of the baseline adder

receive 8 bits carry propagation since $32 \ mod \ 3 = 2$. However, we choose block size of 4 bits, the left most bit will have 12 bits carry propagation.

As we can see from the simulation result above, when we have the block size of 4 bits, the adder can produce an accuracy of 99.78% when MMA is 99%. Therefore, we decide to choose block size 4 and 6 as our testing case below.

Table III. Simulation result for block size 4

Sub adder size / MMA	8 bits	12 bits	16 bits	24 bits
100%	0.0898	0.0296	0.0010	0.0008
99%	0.9976	0.9976	0.9975	0.9904
98%	0.9995	0.9995	0.9993	0.9975
97%	0.9997	0.9997	0.9996	0.9990
96%	0.9998	0.9998	0.9998	0.9995
95%	0.9999	0.9999	0.9999	0.9996
94%	0.9999	0.9999	0.9999	0.9997
93%	0.9999	0.9999	0.9999	0.9998
92%	0.9999	0.9999	0.9999	0.9998
91%	0.9999	0.9999	0.9999	0.9999
90%	0.9999	0.9999	0.9999	0.9999

Table IV. Simulation result for block size 6

Sub adder size / MMA	6 bits	12 bits	16 bits	24 bits
100%	0.1751	0.0312	0.0049	0.0012
99%	0.9994	0.9993	0.9990	0.9908
98%	0.9999	0.9998	0.9996	0.9976
97%	0.9999	0.9999	0.9998	0.9989
96%	0.9999	0.9999	0.9998	0.9993
95%	1	0.9999	0.9999	0.9996
94%	1	0.1	0.9999	0.9998
93%	1	0.1	0.9999	0.9998
92%	1	0.1	0.9999	0.9998
91%	1	0.1	0.9999	0.9999
90%	1	0.1	0.9999	0.9999

As shown in the Table III & IV, the proposed adder shows a very high accuracy when MMA is below 100%. And this verify that if we increase the propagation chain length at more significant bits, we can have a relatively high accuracy while low bits are not accurate. Adding the OR-gate sub adder greatly affect the accuracy of complete correct result since the OR-gate sub adder can almost ensure to have an incorrect result. However, it affects little on accuracy below 100% MMA, the accuracy of sub adder size 16 bits is almost identical to the adder without OR sub adder. Therefore, the proposed adder is perfectly suitable for error-tolerant applications or systems. If the system already can accept input which is 99% correct, the proposed adder will provide both power efficient and fast performance.

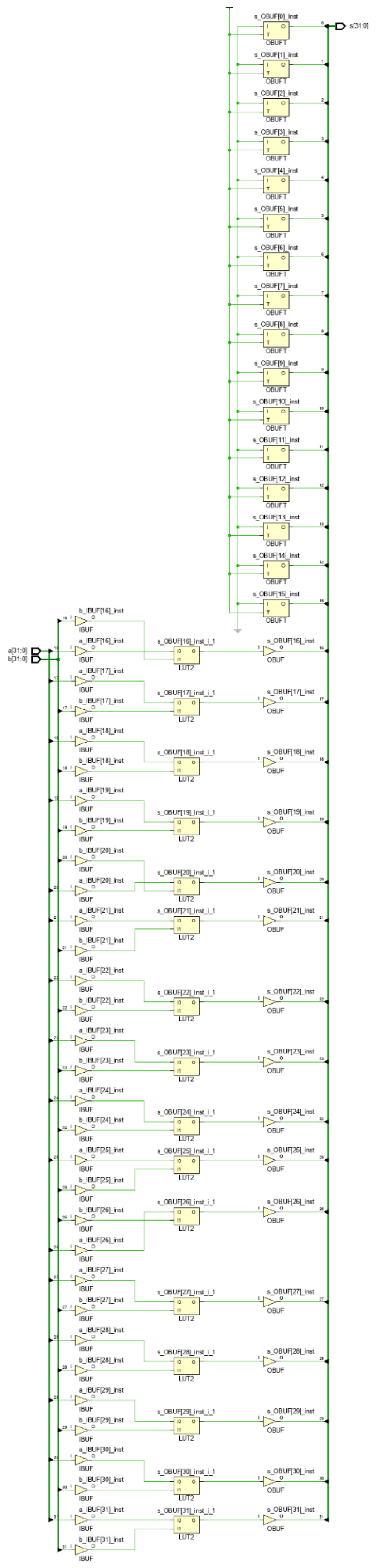
The 32-bit adder with block size 4 and sub adder size 16 is chosen for VHDL implementation since it holds a decent 99.75% accuracy at 99% MMA and has a reasonable block size and sub adder size. The result is in the appendix.

Theoretically, the computation speed of the approximate adder is constant. The sub adder only needs 1 gate delay to produce the output since it has all parallel OR gates structure. The carry generator will take 1 parallel and 2 sequential gate delay to produce the output. And the sum generator having a CLA structure will need 5 gate delay to produce the output. The critical path occurs at the left most block where the carry will propagate for 2 block size. It will have a delay of $1 + 2 + 2 + 5 = 10$ gate delay. Therefore, the worst-case delay should be 10 gate delay.

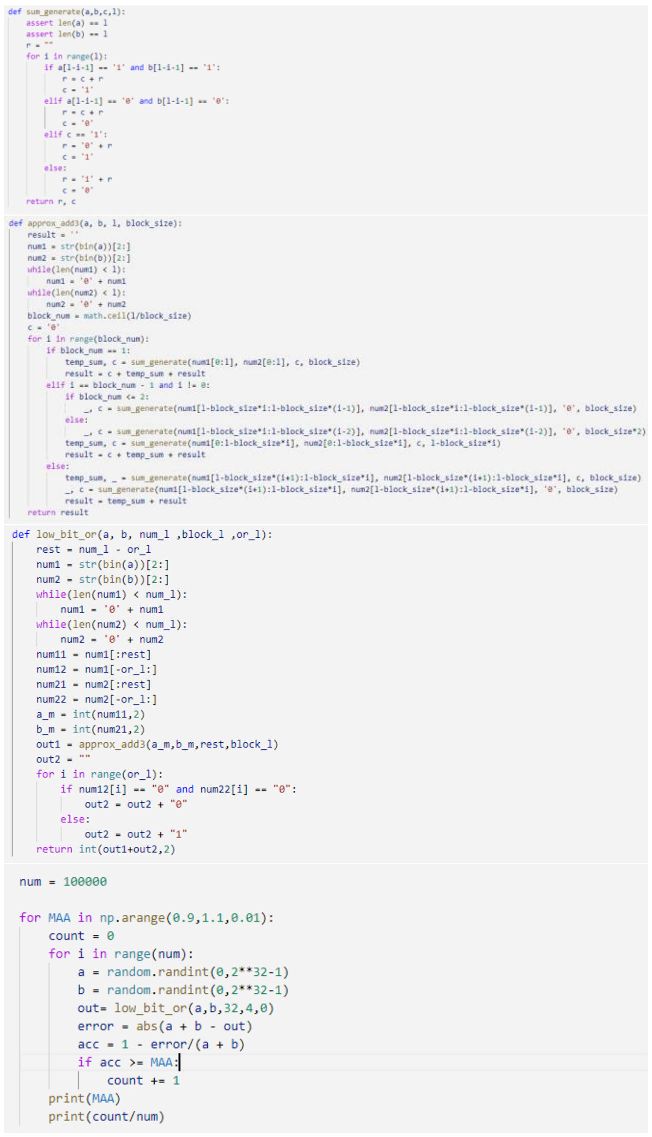
V.SUMMARY

In this project, we review some of the existed design of the approximate adder. Also, we propose a new approximate design improved on others' basis. Our proposed adder is a high-speed power-efficient approximate adder and designed for error-tolerant applications and systems. It can deliver a promising performance when the system does not require complete correct inputs. It is power saving since we combine the OR-gate based adder which has a very simple structure into our design. Also, CLA structure ensures the adder performs at a very high speed. Future work will further investigate the idea of increasing the accuracy of more significant bits for better overall accuracy and lowing the accuracy of less significant bits for power saving.

APPENDIX



VHDL implementation



```
def sum_generate(a,b,c,l):
    assert len(a) == l
    assert len(b) == l
    r = ""
    for i in range(l):
        if a[l-i-1] == '1' and b[l-i-1] == '1':
            r = c + r
            c = '1'
        elif a[l-i-1] == '0' and b[l-i-1] == '0':
            r = c + r
            c = '0'
        elif c == '1':
            r = '0' + r
            c = '1'
        else:
            r = '1' + r
            c = '0'
    return r, c

def approx_add3(a, b, l, block_size):
    result = ''
    num1 = str(bin(a))[2:]
    num2 = str(bin(b))[2:]
    while(len(num1) < l):
        num1 = '0' + num1
    while(len(num2) < l):
        num2 = '0' + num2
    block_num = math.ceil(l/block_size)
    c = '0'
    for i in range(block_num):
        if block_num == 1:
            temp_sum, c = sum_generate(num1[0:l], num2[0:l], c, block_size)
            result = c + temp_sum + result
        elif i == block_num - 1 and i != 0:
            if block_num <= 2:
                _, c = sum_generate(num1[l-block_size*i:l-block_size*(i-1)], num2[l-block_size*i:l-block_size*(i-1)], '0', block_size)
            else:
                _, c = sum_generate(num1[l-block_size*i:l-block_size*(i-2)], num2[l-block_size*i:l-block_size*(i-2)], '0', block_size*2)
            temp_sum, c = sum_generate(num1[0:l-block_size*i], num2[0:l-block_size*i], c, l-block_size*i)
            result = c + temp_sum + result
        else:
            temp_sum, _ = sum_generate(num1[l-block_size*(i+1):l-block_size*i], num2[l-block_size*(i+1):l-block_size*i], c, block_size)
            _, c = sum_generate(num1[l-block_size*(i+1):l-block_size*i], num2[l-block_size*(i+1):l-block_size*i], '0', block_size)
            result = temp_sum + result
    return result

def low_bit_or(a, b, num_l ,block_l ,or_l):
    rest = num_l - or_l
    num1 = str(bin(a))[2:]
    num2 = str(bin(b))[2:]
    while(len(num1) < num_l):
        num1 = '0' + num1
    while(len(num2) < num_l):
        num2 = '0' + num2
    num11 = num1[:rest]
    num12 = num1[-or_l:]
    num21 = num2[:rest]
    num22 = num2[-or_l:]
    a_m = int(num11,2)
    b_m = int(num21,2)
    out1 = approx_add3(a_m,b_m,rest,block_l)
    out2 = ""
    for i in range(or_l):
        if num12[i] == "0" and num22[i] == "0":
            out2 = out2 + "0"
        else:
            out2 = out2 + "1"
    return int(out1+out2,2)

num = 100000

for MAA in np.arange(0.9,1.1,0.01):
    count = 0
    for i in range(num):
        a = random.randint(0,2**32-1)
        b = random.randint(0,2**32-1)
        out= low_bit_or(a,b,32,4,0)
        error = abs(a + b - out)
        acc = 1 - error/(a + b)
        if acc >= MAA:
            count += 1
    print(MAA)
    print(count/num)
```

Python Code for simulation

All Python and VHDL code can also be found in [8].

References

1. Han J, Orshansky M (2013) Approximate computing: an emerging paradigm for energy efficient design. In: European test symposium. IEEE, Piscataway, pp 1–6
2. Mahdiani HR, Ahmadi A, Fakhraie SM, Lucas C (2010) Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. IEEE Trans Circuits Syst 57(4):850–862
3. Du K, Varman P, Mohanram K (2012) High performance reliable variable latency carry select addition. In: Proceedings of the conference on design, automation and test in Europe. EDA Consortium, San Jose, pp 1257–1262
4. Kim Y, Zhang Y, Li P (2013) An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems. In: Proceedings of the international conference on computer-aided design. IEEE, Piscataway, pp 130–137
5. Zhu N, Goh WL, Yeo, KS (2009) An enhanced low-power high-speed adder for error-tolerant application. In: Proceedings of the international symposium on integrated circuits. IEEE, Piscataway, pp 69–72
6. Mahdiani HR, Ahmadi A, Fakhraie SM, Lucas C (2010) Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. IEEE Trans Circuits Syst 57(4):850–862
7. Kahng AB, Kang S (2012) Accuracy-configurable adder for approximate arithmetic designs. In: Design automation conference. IEEE, Piscataway, pp 820–825
8. https://github.com/lovettxh/ECE511