CARRY-SELECT ADDER

Batch – 2

Abstract: Adders are basic building blocks of any processor or data path application. For the design of high-performance processing units high speed adders with low power consumption is a requirement. Carry Select Adder (CSA) is known to be one of the fastest adders used in many data processing applications. In this project, we present a new CSLA architecture using NAND gates alone. It employs multiplexers and full-adders for calculation of sum and carry of two 16-bit numbers.

INTRODUCTION:

Adding two numbers using a calculator, we get the answer in a few milli-seconds, but the addition of two five-digit numbers for instance is not so facile. It has many procedures like computing the sum of each digit, computing the carry and propagating the carry to the next digit. How the calculator performs such clumsy tasks in a very short time? What is mechanism happening behind the calculator for performing this task? What is the name of the circuit used for performing this clumsy task? Which circuit calculates the sum of two numbers most swiftly and efficiently? This project aims on giving the most suitable answer for all such question.

Firstly, the circuit which is used for performing this strenuous task is known as an adder. An adder is a digital circuit that performs addition of numbers. In a more technical way, it is a digital circuit which adds up the amplitude of two propagating waves. In many computers and other kinds of processors adders are used in the arithmetic logic units or ALU. They are also used in other parts of the processor, where they are used to calculate addresses, table indices, increment and decrement operators and similar operations. Although adders can be constructed for many number representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement or ones' complement is being used to represent negative numbers, it is trivial to modify an adder into an adder–subtractor. Other signed number representations require more logic around the basic adder. On the basis of the number of bits they operate on adders can be classified into two categories, which are, the adders which operate non single bit and the adders which operate on multiple bits.

There are two basic adders that operate on single bit which are the half adders and full adder. The half adder adds two single binary digits A and B. It has two outputs, sum (S) and carry (C). The carry signal represents an overflow into the next digit of a multi-digit addition. The simplest half-adder design, incorporates an XOR gate for Sum and an AND gate for Carry. The input variables of a half adder are called the augend and addend bits. The output variables are the sum and carry. A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full-adder adds three one-bit numbers, often written as A, B, and $C_{in.}$ Here, A and B are the operands, and Cin is a bit carried in from the previous less-significant stage. A full adder can also be constructed from two half adders by connecting A and B to the input of one half-adder, then taking its sum-output S as one of the inputs to the second half adder and C_{in} as its other input, and finally the carry outputs from the two half-adders are connected to an OR gate. The sum-output from the second half adder is the final sum output (S) of the full adder and the output from the OR gate is the final carry output (C_{out}).

Here, usually a cascade of the adders which operate in single bit is used to make the adders which operate in multiple bits, where, the first bit i.e., the least significant bit is usually operated using a half adder and all the other bits are evaluated using full adders, because there are no carries generated before evaluating the last bit of the number.

Looking into the adders that operate on multiple bits there are four major topologies of adders namely,

- Ripple carry adder
- Carry look-ahead adder
- Carry skip adder
- Carry select adder

A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage. In a ripple carry adder the sum and carry out bits of any half adder stage is not valid until the carry in of that stage occurs. Propagation delays inside the logic circuitry is the reason behind this. A structure of multiple full adders is cascaded in a manner to gives the results of the addition of an n bit binary sequence. This adder includes cascaded full adders in its structure so, the carry will be generated at every full adder stage in a ripple-carry adder circuit. These carry output at each full adder stage is forwarded to its next full adder and there applied as a carry input to it. This process continues up to its last full adder stage. So, each carry output bit is rippled to the next stage of a full adder. By this reason, it is named as "RIPPLE CARRY ADDER". The most important feature of it is to add the input bit sequences whether the sequence is 4 bit or 5 bit or any. One of the most important point to be considered in this carry adder is the final output is known only after the carry outputs are generated by each full adder stage and forwarded to its next stage. So, there will be a delay to get the result with using of this carry adder.

A carry-Lookahead adder is a fast parallel adder as it reduces the propagation delay by more complex hardware, hence it is costlier. In this design, the carry logic over fixed groups of bits of the adder is reduced to two-level logic, which is nothing but a transformation of the ripple carry design. This method makes use of logic gates so as to look at the lower order bits of the augend and addend to see whether a higher order carry is to be generated or not. In this adder, the carry input at any stage of the adder is independent of the carry bits generated at the independent stages. Here the output of any stage is dependent only on the bits which are added in the previous stages and the carry input provided at the beginning stage. Hence, the circuit at any stage does not have to wait for the generation of carry-bit from the previous stage and carry bit can be evaluated at any instant of time. High-speed Carry Look-ahead Adders are used as implemented as IC's. Hence, it is easy to embed the adder in circuits. By combining two or more adders' calculations of higher bit Boolean functions can be done easily. Here the increase in the number of gates is also moderate when used for higher bits. For this Adder there is a trade-off between area and speed. When used for higher bit calculations, it provides high speed but the complexity of the circuit is also increased thereby increasing the area occupied by the circuit. This adder is usually implemented as 4-bit modules which are cascaded together when used for higher calculations. This adder is costlier compared to other adders.

A carry-skip adder (also known as a carry-bypass adder) is an adder implementation that improves on the delay of a ripple-carry adder with little effort compared to other adders. The improvement of the worst-case delay is achieved by using several carry-skip adders to form a block-carry-skip adder. The n-bit-carry-skip adder consists of a n-bit-carry-ripple-chain, a n-input AND-gate and one multiplexer. The n-bit-carry-skip adder consists of a n-bit-carry-ripple-chain, a n-input AND-gate and one multiplexer. Each propagate bit p_i, that is provided by the carry-ripple-chain is connected to the n-input AND-gate. The resulting bit is used as the select bit of a multiplexer that switches either the last carry-bit c_n or the carry-in c_0 to the carry-out signal c_{out}.

Carry Select Adder (CSLA) is one of the fastest adders use in many data-processing processors to perform fast arithmetic functions. From the structure of the CSLA, it is clear that there is scope for reducing the area and power consumption in the CSLA. This work uses a simple and efficient gate-level modification to significantly reduce the area and power of the CSLA. A carry-select adder is a particular way to implement an adder, which is a logic element that computes the (n+1)-bit sum of two n-bit numbers. The carry-select adder is simple but rather fast. The carry-select adder generally consists of ripple carry adders and a multiplexer. Adding two n-bit numbers with a carry-select adder is done with two adders (therefore two ripple carry adders), in order to perform the calculation twice, one time with the assumption of the carry-in being zero and the other assuming it will be one. After the two results are calculated, the correct sum, as well as the correct carry-out, is then selected with the multiplexer once the correct carry-in is known. The number of bits in each carry select block can be uniform, or variable. In the uniform case, the optimal delay occurs for a block size of $[\sqrt{n}]$. When variable, the block size should have a delay, from addition inputs A and B to the carry out, equal to that of the multiplexer chain leading into it, so that the carry out is calculated just in time. The $O(\sqrt{n})$ delay is derived from uniform sizing, where the ideal number of full-adder elements per block is equal to the square root of the number of bits being added, since that will yield an equal number of MUX delays. In this circuit we have used 7 ripple-carry adder's (4 bit) and 15 multiplexers. The multiplexers are used to select the one of the carries from one of the two ripple carry adders each time. CSLA is found to be efficient on the basis of time required for calculation, because, already the carry and sum will be calculated and we are going to select the output using the multiplexer. The conditions for a logic circuit to be efficient are to occupy Less Area, to consume Less Power and to have very less time delays. All the above conditions except the constraint on area are satisfied by CSLA. Hence, it is considered to be one of the high efficiency adders. This circuit can further be optimized by using BEC (Binary to Excess-1 Converter) instead of ripple-carry adders. The carry select adder has various application in a variety of fields as follows:

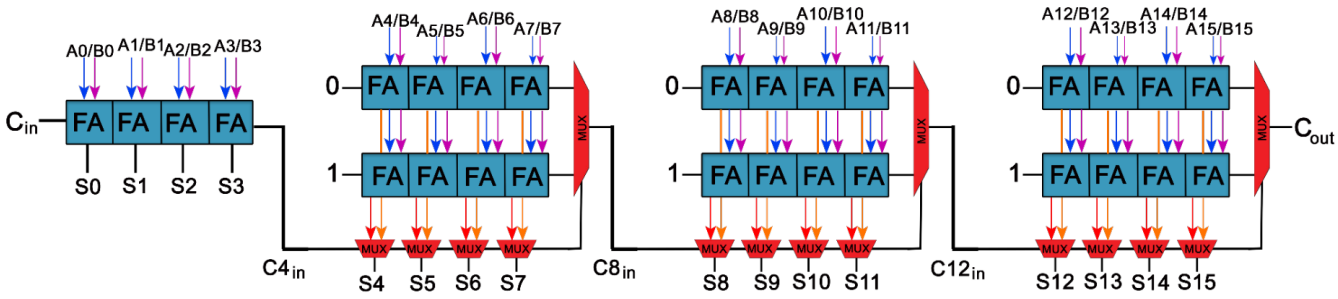
Image processing: In image processing with interpolation, an output of the gamma circuit and the input data are input to an adder circuit so as to obtain the added and averaged values at a predetermined ratio.

Arithmetic logic units: Carry select adder is used in arithmetic logic units to perform addition and multiplication in a less amount of time.

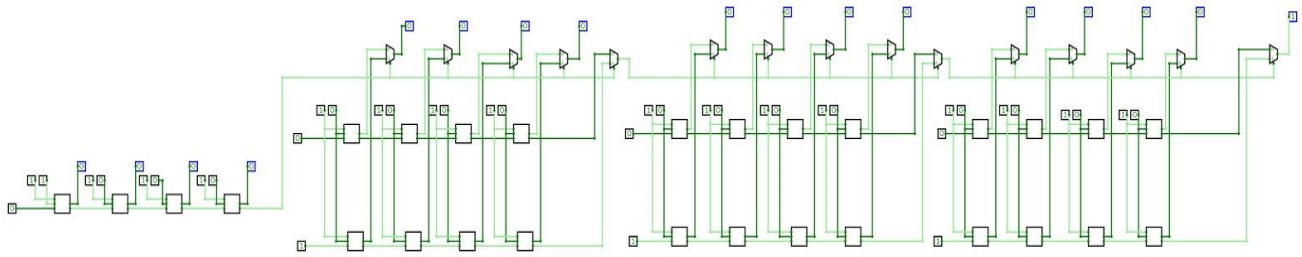
Advanced microprocessor design: In microprocessor design, the adder is used for the conversion mechanism in calculating the physical address using the offset address and segment address.

High speed multiplication: In multiplier, each bit of the Product P is obtained by a summation of bits AiBj using an array of single bit adders. The bits AiBj are formed using AND gates.

CHIP DIAGRAM:



CIRCUIT DIAGRAM



https://circuitverse.org/users/66575/projects/clsa

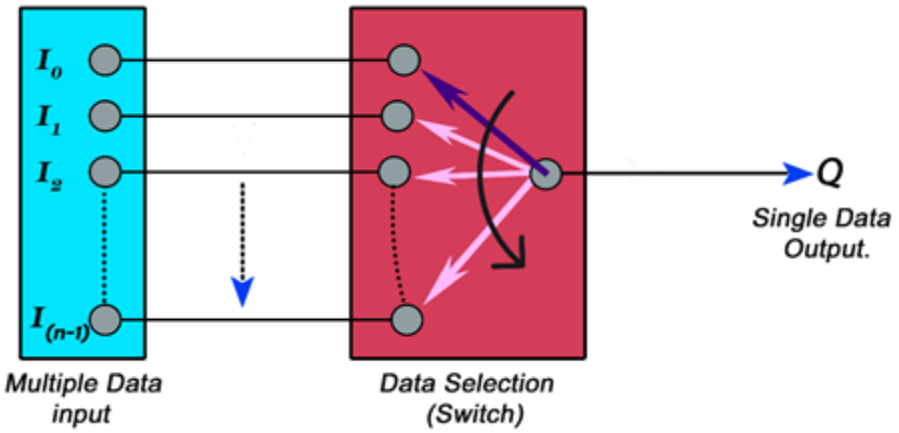
WORKING AND IMPLEMENTATION

Multiplexer

The term ‘multiplexing’ describes the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds. The device used for multiplexing is called Multiplexer.

It is shortly called MUX. It is a combinational logic circuit designed to switch one of several input lines to a single common output line by the application of a control signal, called selection lines. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called “channels” one at a time to the output.

Basic working of MUX:



The rotary switch is a mechanical device whose input is selected by a rotating shaft. The rotary switch is a manual switch that you can use to select individual data or signal lines simply by turning its inputs ‘ON’ or ‘OFF’. That is the reason why MUX is also known as ‘DATA SELECTORS’.

It is generally used to reduce the number of logic gates required in a circuit design. The selection of input is done by the selection line or control lines, according to the binary condition of these control inputs (either ‘HIGH’ or ‘LOW’). Number of input lines in a Mux = 2n where, n is the number of selection lines. So, a 2:1 MUX means it has 2 input lines, one selection line and one output line.

2: 1 MUX

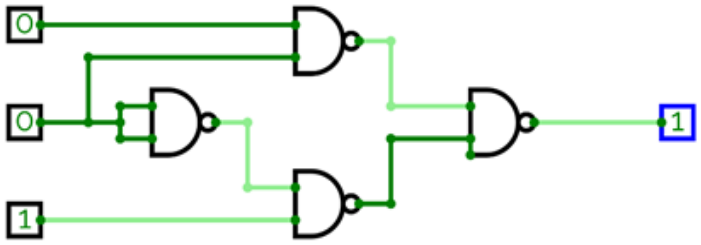
Truth table

S_0	D_1	D_0	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	0	1

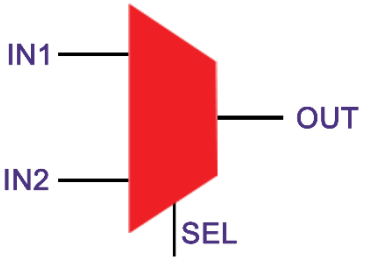
K -map

S_0 \ D_1D_0	00	01	11	10
0	0	1	1	0
1	0	0	1	1

Chip diagram



schematic diagram



This is a 2-1 MUX constructed from NAND gates. Hence, the equation of MUX from k – map is,

$$f = S_0'.D_0 + S_0.D_1$$

When data of selection line input is LOW, then I_0 is selected and comes as output, while the other input is blocked. When data of selection line input is HIGH, then I_1 is selected and comes as output, while the other input is blocked.

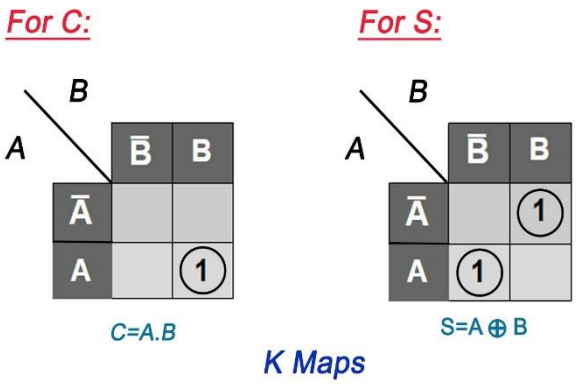
HALF ADDER:

It is a combinational logic circuit, that can add 2 bits. The input variables are two, one-bit numbers and outputs are sum and carry.

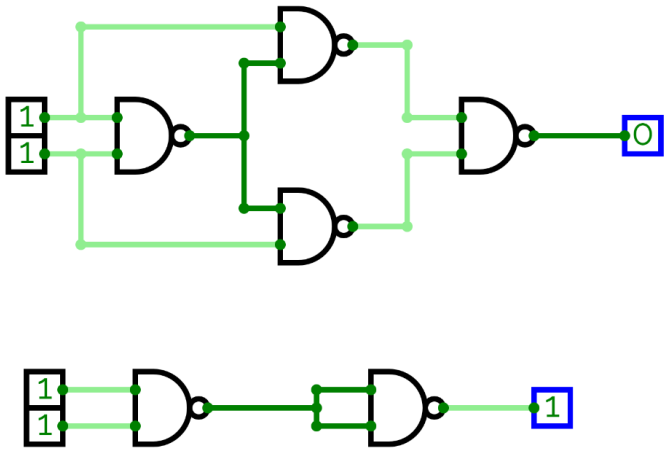
Truth-Table:

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

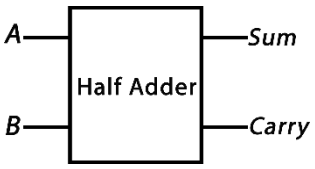
K – map



Chip diagram



schematic diagram



FULL ADDER

Full adder is a logical circuit that adds two input bits and a Carry-in bit and gives Carry-out bit and a sum-bit as output. The Sum-out of a full adder is the XOR of input bits A, B and Carry-in bit.

Truth table

Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K - map

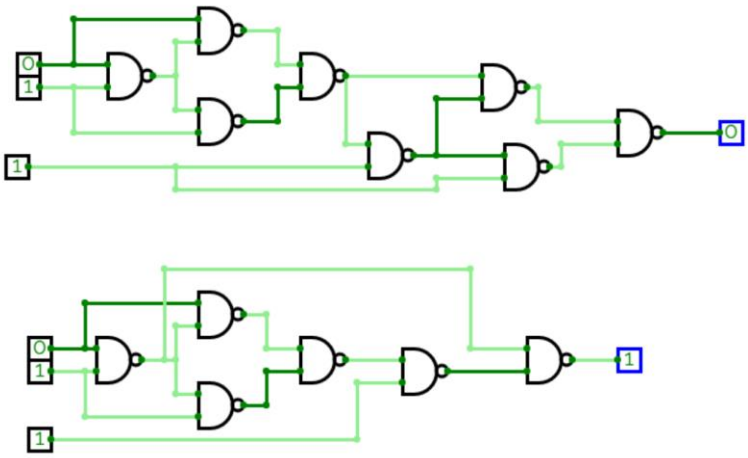
For s:

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

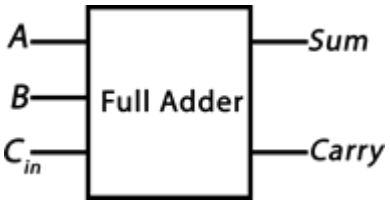
For c:

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Chip diagram



schematic diagram



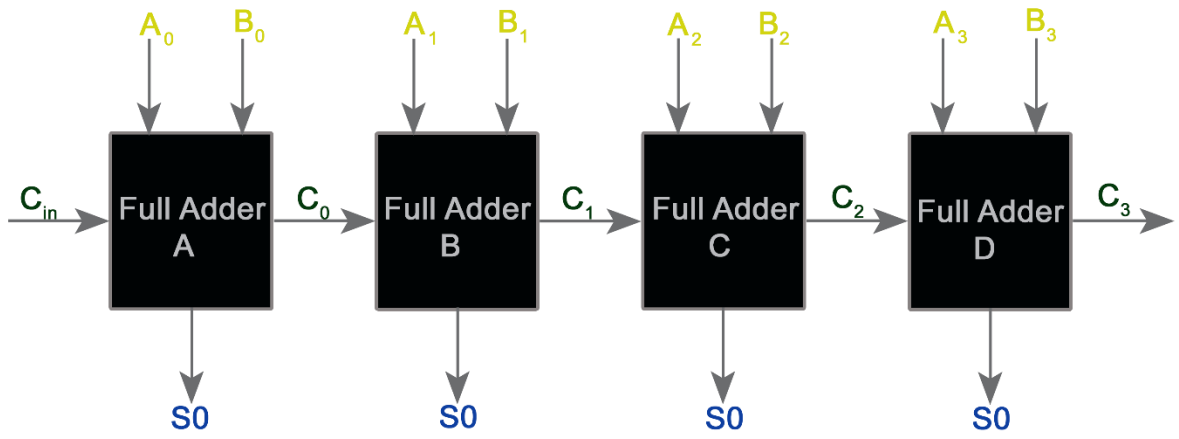
RIPPLE CARRY ADDER:

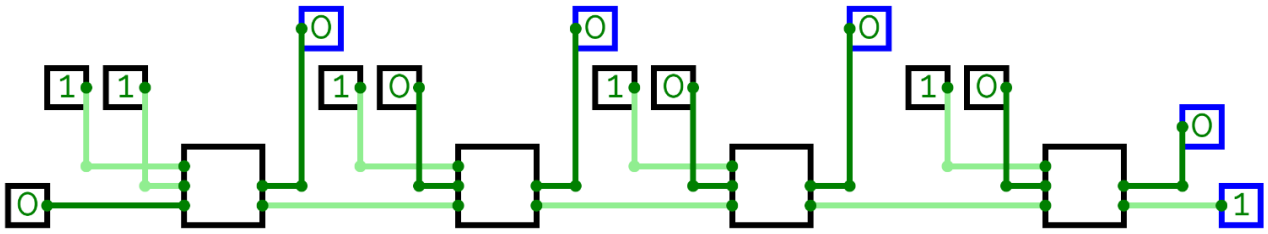
Consists of Full adder and Half adder.

It is constructed by cascading multiple full adders in parallel to add an n-bit number.

For an n- bit parallel adder, there must be ‘n’ number of full adder circuits. The carry-out of each full adder is the carry-in of the succeeding next most significant full adder. Since, each carry bit gets rippled into the next stage, it is called a ripple carry adder. In a ripple carry adder, the sum and carry-out bits of any half adder stage is not valid until the carry-in of that stage occurs. The reason behind this is, the propagation delays inside the logic circuitry. Propagation delay is time elapsed between the application of an input and occurrence of the corresponding output. Carry propagation delay is the time elapsed between the application of the carry in signal and the occurrence of the carry-out (C-out) signal.

Chip diagram





Sum-out S_0 and carry out C-out of the Full Adder 1 is valid only after the propagation delay of Full Adder 1. In the same way, Sum-out S_3 of the Full Adder 4 is valid only after the joint propagation delays of Full Adder 1 to Full Adder 4. Hence, the final result of the ripple carry adder is valid only after the joint propagation delays of all full adder circuits inside it.

COMPUTATION OF SUM-BIT AND CARRY-BIT:

STAGE 1:

Calculation of S_0

$S_0 = A_0 + B_0 + C_{in}$

Calculation of C_0

$C_0 = A_0 B_0 + B_0 C_{in} + C_{in} A_0$

STAGE 2:

When C_0 is fed as input to the full adder B, it activates the full adder B.
Full adder B computes the sum bit and carry bit as-

Calculation of S_1

$S_1 = A_1 \oplus B_1 \oplus C_0$

Calculation of C_1

$C_1 = A_1 B_1 + B_1 C_0 + C_0 A_1$

STAGE 3:

When C_2 is fed as input to the full adder C, it activates the full adder C.
Full adder C computes the sum bit and carry bit as-

Calculation of S_2

$S_2 = A_2 \oplus B_2 \oplus C_2$

Calculation of C_2

$C_2 = A_2 B_2 + B_2 C_2 + C_2 A_2$

STAGE 4:

When C_3 is fed as input to the full adder D, it activates the full adder D.
Full adder D computes the sum bit and carry bit as –

Calculation of S_3

$S_3 = A_3 \oplus B_3 \oplus C_2$

Calculation of C3

$C_3 = A_3 B_3 + B_3 C_2 + C_2 A_3$

CARRY SELECT ADDER:

The carry select adder uses multiple narrow adders to create fast wide adders. It breaks the addition problem into smaller groups. It is one of the fast-type of adder. The adder consists of two independent units. Each unit implements the addition operation in parallel.

The adder based on this approach is Carry Select Adder (CSLA). It is generally used in data processing processors to perform fast arithmetic functions.

It consists of two parallel adders (ripple carry adder) and a multiplexer.

For two given numbers A(n-bit) and B(n-bit), we perform addition twice.

- With carry in = '0'
- With carry in = '1'

Once the correct carry in is known, the correct sum is selected by a mux. For a multi-bit adder, the number of bits in each carry select block can be either uniform or variable.

Here, we add two 16-bit numbers. So, we use 4 blocks. Each block consists of 2 ripple-carry adders and four 2:1 MUX es. Except the first block, which has one ripple carry adder and no MUX. The first block computes the sum of first 4 bits of two 16-bit numbers based on the Carry-in given. It then generates Carry-out and four bits of sum of two bits (S_3, S_2, S_1, S_0).

The generated Carry-out goes as a selection line for 2:1 MUX es of the second block. The second block has 2:1 MUX for selecting the sum of two 16-bit numbers, generated by two ripple-carry adders, based on the Carry-out of the first block. Since, two ripple-carry adders are there, two Carry-out will be generated. The Carry-out of the first block acts as a selection line for this 2:1 MUX to select the required carry. Now, the required Carry-out and Sum-out (Next four bits – S_7, S_6, S_5, S_4) is selected.

This Carry-out goes as a selection line for 2:1 MUX es in the third block. So, the required Sum-out (Next four bits - S_{11}, S_{10}, S_9, S_8) and Carry-out are generated.

This Carry-out goes to the fourth block as the selection line for 2:1 MUX es. So, the required Sum-out (Next four bit – S_{15}, S_{14}, S_{13}, S_{12}) and Carry-out (Final Carry-out) is selected.

Thus, a Sum and Carry-out is generated. This is how a Carry Select Adder works.

Time Propagation Delay of Carry select adder:

A carry select adder is an arithmetic combinational logic circuit which adds two N-bit binary numbers and outputs their N-bit binary sum and a 1-bit carry. This is not different from a ripple carry adder in function, but in design the carry select adder does not propagate the carry through as many full adders as the ripple carry adder does. This means that the time to add two numbers is shorter in a CSLA.

It avoids propagating the carry from bit to bit in sequence. If we have two adders in parallel: one with a carry input of 0, the other with a carry input of 1, then we could use the actual carry input generated to select between the outputs of the two parallel adders. This means all adders are performing their calculations in parallel. Having two adders for each result bit is quite wasteful so we could configure the N-bit adder to use 2*N/M-1 M-bit ripple carry adders in parallel. The adder for the least significant bits will always have a carry input of 0 so no parallel addition is needed in this case.

FOR RIPPLE CARRY ADDER:

The computation time is given by: $T = k1 * n$

Where k1 is the delay through one adder cell and n is the number of bits.

For Carry Select Adder:

If we divide the adder into blocks, each with 2 parallel paths, then the completion time T becomes,

$T = (K1 * n/2) + k2$

Where k2 is time needed by the multiplexer of the next block to select the actual output carry.

Suppose the n-bit adder is divided into 'M' blocks, and that each block contains 'P' adder cells in series,

The Completion time T for overall carry output signal is composed of two parts.

i) The propagation delay through the first block

ii) The propagation delay through the mux block

$T = Pk1 + (M-1) * k2$

Let $n = M * P$

Then, $T = (n/m) * k1 + (M-1) * k2$

On differentiating with respect to M, we get

M = sqrt(n*k1/k2)

GENERAL POINTS----------

Carry Select Adder (CSLA) is one of the fastest adders use in many data-processing processors to perform fast arithmetic functions. From the structure of the CSLA, it is clear that there is scope for reducing the area and power consumption in the CSLA. This work uses a simple and efficient gate-level modification to significantly reduce the area and power of the CSLA.

LOW-POWER, area-efficient, and high-performance VLSI systems are increasingly used in portable and mobile devices, multistranded wireless receivers, and biomedical instrumentation. An adder is the main component of an arithmetic unit. A complex digital signal processing (DSP) system involves several adders. An efficient adder design essentially improves the performance of a complex DSP system. A ripple carry adder (RCA) uses a simple design, but carry propagation delay (CPD) is the main concern in this adder.

Hack ALU - CODE:

HAnand

HDL code:

```
CHIP HAnand {
    IN a, b;    // 1-bit inputs
    OUT sum,    // Right bit of a + b
        carry;  // Left bit of a + b

    PARTS:
    Nand(a=a, b=b, out=ab);
    Nand(a=a, b=ab, out=aab);
    Nand(a=b, b=ab, out=abb);
    Nand(a=aab, b=abb, out=sum);
    Nand(a=ab, b=ab, out=carry);
}
```

Compare file:

a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Test file:

```
load HAnand.hdl,
output-file
HAnand.out,
compare-to HAnand.cmp,
output-list a%B3.1.3
b%B3.1.3 sum%B3.1.3
carry%B3.1.3;

set a 0,
set b 0,
eval,
output;

set a 0,
set b 1,
eval,
output;

set a 1,
set b 0,
eval,
output;

set a 1,
set b 1,
eval,
output;
```

FAnand

HDL - code

```
CHIP FAnand {
    IN a, b, c;  // 1-bit inputs
    OUT sum,     // Right bit of a + b + c
        carry;   // Left bit of a + b + cbc

    PARTS:
    Nand(a=a, b=b, out=ab);
    Nand(a=a, b=ab, out=aab);
    Nand(a=b, b=ab, out=abb);
    Nand(a=aab, b=abb, out=axb);
    Nand(a=axb, b=c, out=axbc);
    Nand(a=axbc, b=axb, out=sum1);
    Nand(a=axbc, b=c, out=sum2);
    Nand(a=sum1, b=sum2, out=sum);
    Nand(a=ab,b=axbc, out=carry);
}
```

Compare file

a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Test file

```
load FAnand.hdl,
output-file FAnand.out,
compare-to FAnand.cmp,
output-list a%B3.1.3
b%B3.1.3 c%B3.1.3
sum%B3.1.3 carry%B3.1.3;

set a 0,
set b 0,
set c 0,
eval,
output;

set c 1,
eval,
output;

set b 1,
set c 0,
eval,
output;

set c 1,
eval,
output;

set a 1,
set b 0,
set c 0,
eval,
output;

set c 1,
eval,
output;

set b 1,
set c 0,
eval,
output;

set c 1,
eval,
output;
```

Mux

HDL - code

```
CHIP Mux {
    IN a, b, sel;
    OUT out;

    PARTS:
    Nand(a=sel,b=sel,out=notsel);
    Nand(a=a,b=notsel,out=out1);
    Nand(a=b,b=sel,out=out2);
    Nand(a=out1,b=out2,out=out);
}
```

Compare file

a	b	sel	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

```
load Mux.hdl,
output-file Mux.out,
compare-to Mux.cmp,
output-list a%B3.1.3
b%B3.1.3 sel%B3.1.3
out%B3.1.3;

set a 0,
set b 0,
set sel 0,
eval,
output;

set sel 1,
eval,
output;

set a 0,
set b 1,
set sel 0,
eval,
output;

set sel 1,
eval,
output;

set a 1,
set b 0,
set sel 0,
eval,
output;

set sel 1,
eval,
output;


set a 1,
set b 1,
set sel 0,
eval,
output;

set sel 1,
eval,
output;
```

RippleCarryAdder4Bit

HDL - code

```
CHIP RippleCarryAdder4Bit {
    IN a[4], b[4],c;
    OUT out[4],cout;

    PARTS:
    FAnand(a=a[0],b=b[0],c=c,sum=out[0],carry=carry1);
    FAnand(a=a[1],b=b[1],c=carry1,sum=out[1],carry=carry2);
    FAnand(a=a[2],b=b[2],c=carry2,sum=out[2],carry=carry3);
    FAnand(a=a[3],b=b[3],c=carry3,sum=out[3],carry=cout);
}
```

Compare file

a	b	c	out	cout
1	1	0	2	0
1	1	1	3	0
15	0	0	15	0
15	0	1	0	1

```
load RippleCarryAdder4Bit.hdl,
output-file
RippleCarryAdder4Bit.out,
compare-to
RippleCarryAdder4Bit.cmp,
output-list a%D2.2.2 b%D2.2.2
c%D2.1.2 out%D2.2.2
cout%B3.1.3;

set a 1,
set b 1,
set c 0,
eval,
output;

set a 1,
set b 1,
set c 1,
eval,
output;

set a 15,
set b 0,
set c 0,
eval,
output;

set a 15,
set b 0,
set c 1,
eval,
output;
```

Block

HDL - code

```
CHIP block {
    IN a[4], b[4],c;
    OUT out[4],cout;

    PARTS:
    FAnand(a=a[0],b=b[0],c=false,sum=out11,carry=carry11);
    FAnand(a=a[1],b=b[1],c=carry11,sum=out12,carry=carry12);
    FAnand(a=a[2],b=b[2],c=carry12,sum=out13,carry=carry13);
    FAnand(a=a[3],b=b[3],c=carry13,sum=out14,carry=cout1);

    FAnand(a=a[0],b=b[0],c=true,sum=out21,carry=carry21);
    FAnand(a=a[1],b=b[1],c=carry21,sum=out22,carry=carry22);
    FAnand(a=a[2],b=b[2],c=carry22,sum=out23,carry=carry23);
    FAnand(a=a[3],b=b[3],c=carry23,sum=out24,carry=cout2);

    Mux(a=out11,b=out21,sel=c,out=out[0]);
    Mux(a=out12,b=out22,sel=c,out=out[1]);
    Mux(a=out13,b=out23,sel=c,out=out[2]);
    Mux(a=out14,b=out24,sel=c,out=out[3]);

    Mux(a=cout1,b=cout2,sel=c,out=cout);
}
```

Compare file

a	b	c	out	cout
2	5	0	7	0
2	5	1	8	0
5	5	0	10	0
5	5	1	11	0
15	0	0	15	0
15	0	1	0	1
15	15	0	14	1
15	15	1	15	1

Test script

```
load block.hdl,
output-file block.out,
compare-to block.cmp,
output-list a%D1.2.1 b%D1.2.1 c%B3.1.3
out%D1.2.1 cout%B3.1.3;

set a 2,
set b 5,
set c 0,
eval,
output;

set c 1,
eval,
output;


set a 5,
set c 0,
eval,
output;

set c 1,
eval,
output;
```

CSLA (decimal format)

HDL -code

```
CHIP CSLA {
    IN a[16], b[16];
    OUT out[16],cout;

    PARTS:
    RippleCarryAdder4Bit(a=a[0..3],b=b[0..3],out=out[0..3],cout=c1);
    block(a=a[4..7],b=b[4..7],c=c1,out=out[4..7],cout=c2);
    block(a=a[8..11],b=b[8..11],c=c2,out=out[8..11],cout=c3);
    block(a=a[12..15],b=b[12..15],c=c3,out=out[12..15],cout=cout);
}
```

Compare file

a	b	out	cout
13	11	24	0
2468	4628	7096	0
25855	25855	13826	0
32767	1	32768	0

Test file

```
load CSLA.hdl,
output-file CSLA.out,
compare-to CSLA.cmp,
output-list a%D1.5.1 b%D1.5.1
out%D1.5.1 cout%B3.1.3;

set a 13,
set b 11,
eval,
output;

set a 2468,
set b 4628,
eval,
output;


set a 25855,
set b 25855,
eval,
output;

set a 32767,
set b 1,
eval,
output;
```

CSLA (binary format)

Test file

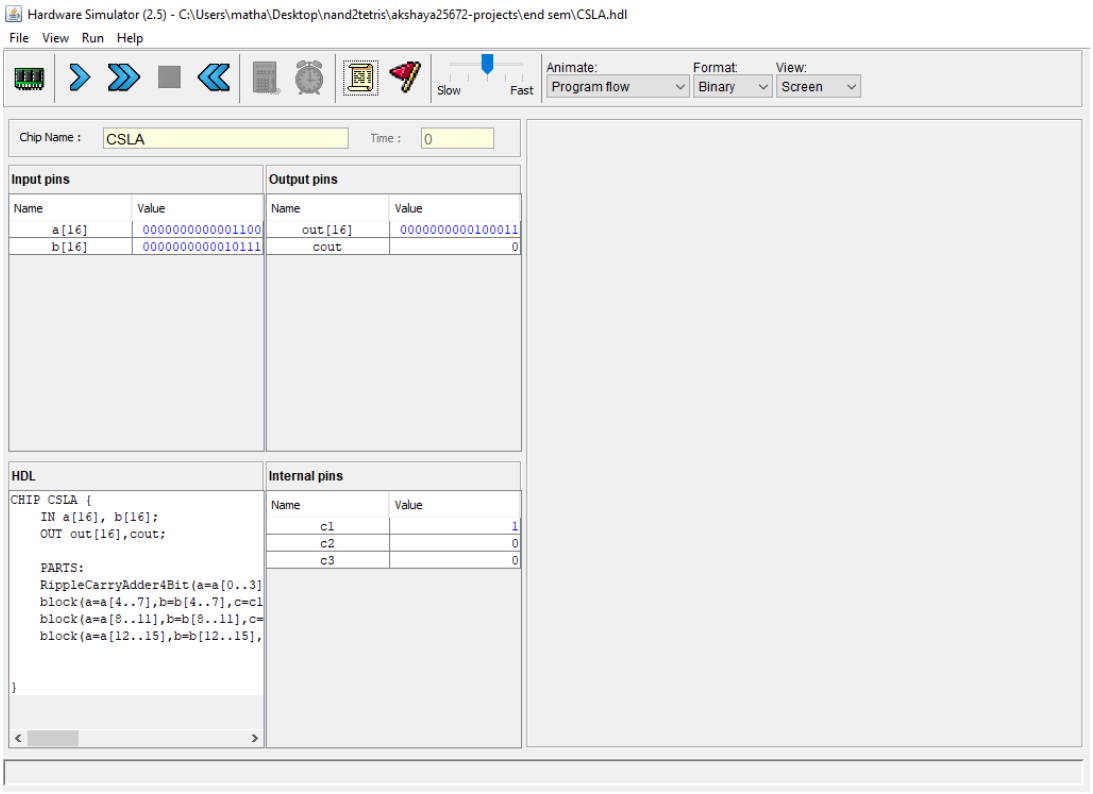
```
load CSLA.hdl,
output-file CSLA1.out,
compare-to CSLA1.cmp,
output-list a%B3.16.3 b%B3.16.3 out%B3.16.3
cout%B3.1.3;

set a %B0000000000000000,
set b %B0101010101010101,
eval,
output;

set a %B1010101010101010,
set b %B0101010101010101,
eval,
output;

set a %B0011001100110011,
set b %B1100110011001100,
eval,
output;


set a %B1111111111111111,
set b %B0000000000000001,
eval,
output;
```

Compare file

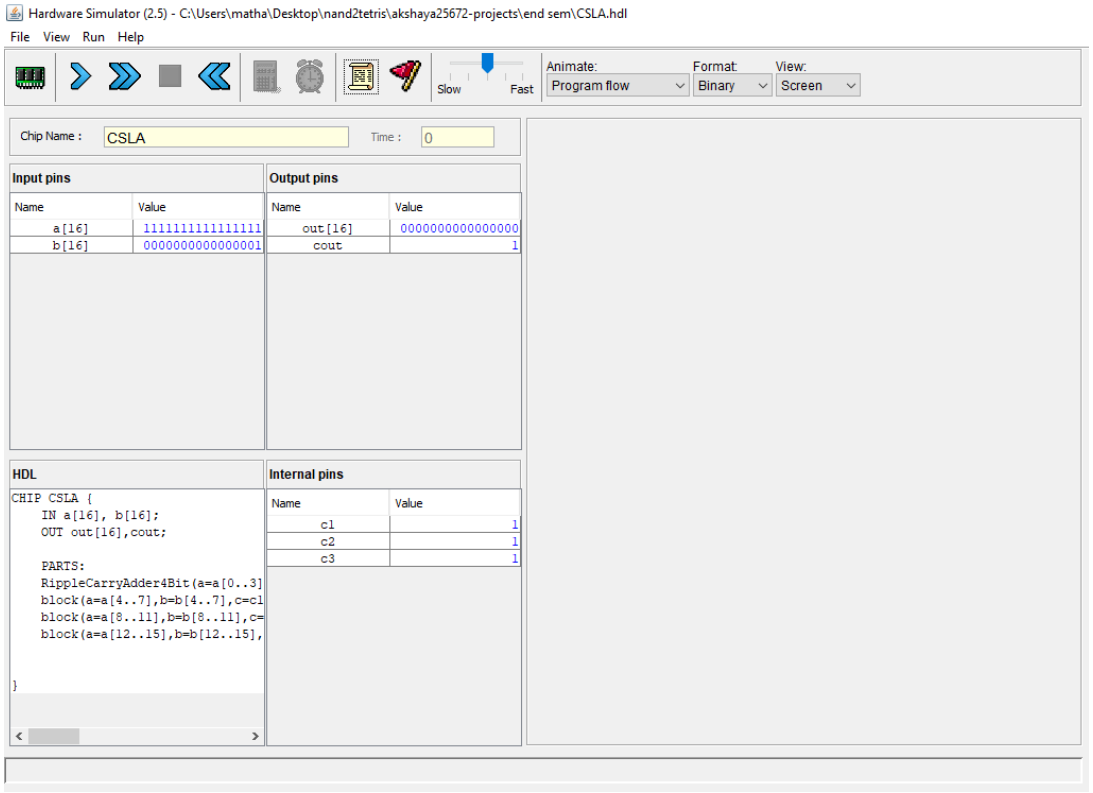
a	b	out	cout
0000000000000000	0101010101010101	0101010101010101	0
1010101010101010	0101010101010101	1111111111111111	0
0011001100110011	1100110011001100	1111111111111111	0
1111111111111111	0000000000000001	0000000000000000	1

RESULT:

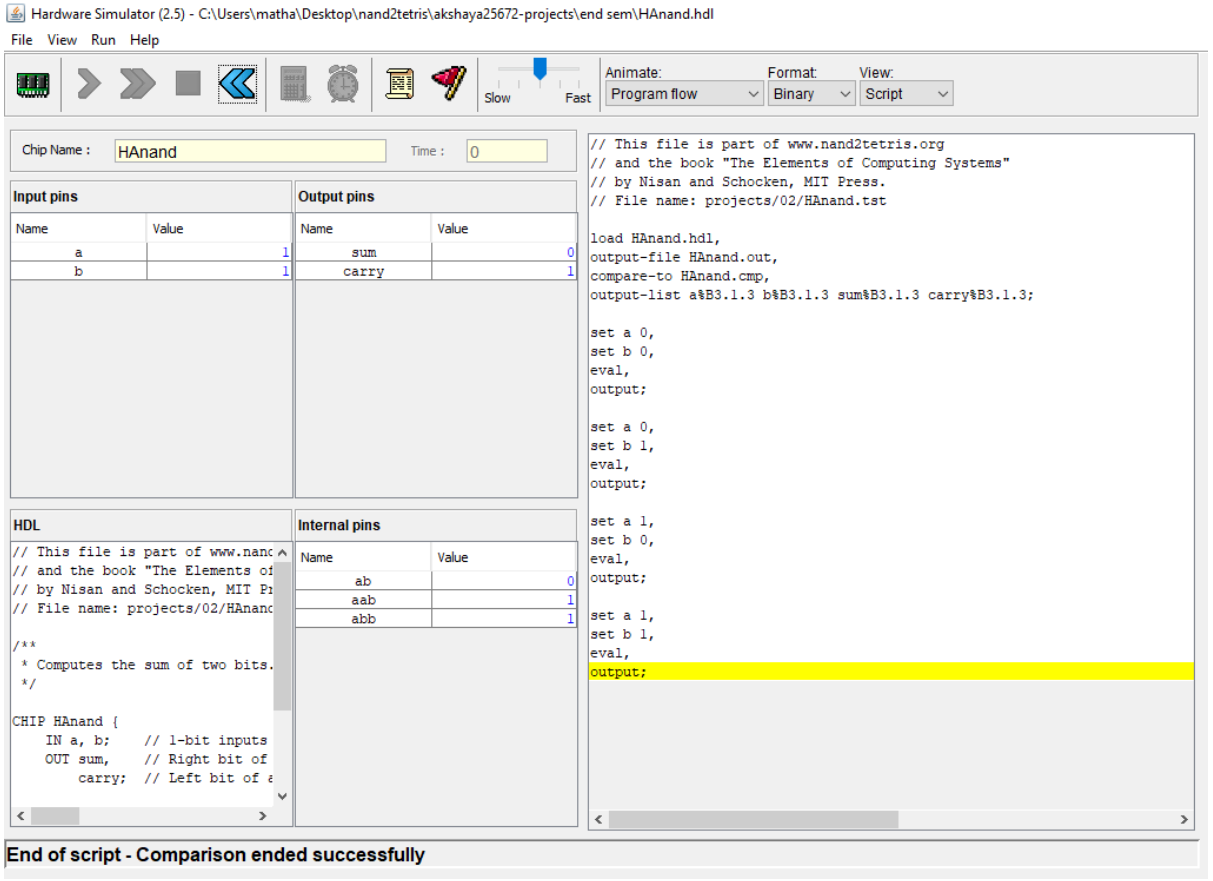
The carry select adder is used to add two n-bit numbers. Let us say one number be a[16] = 0000000000001100 and other number be b[16] = 0000000000010111.

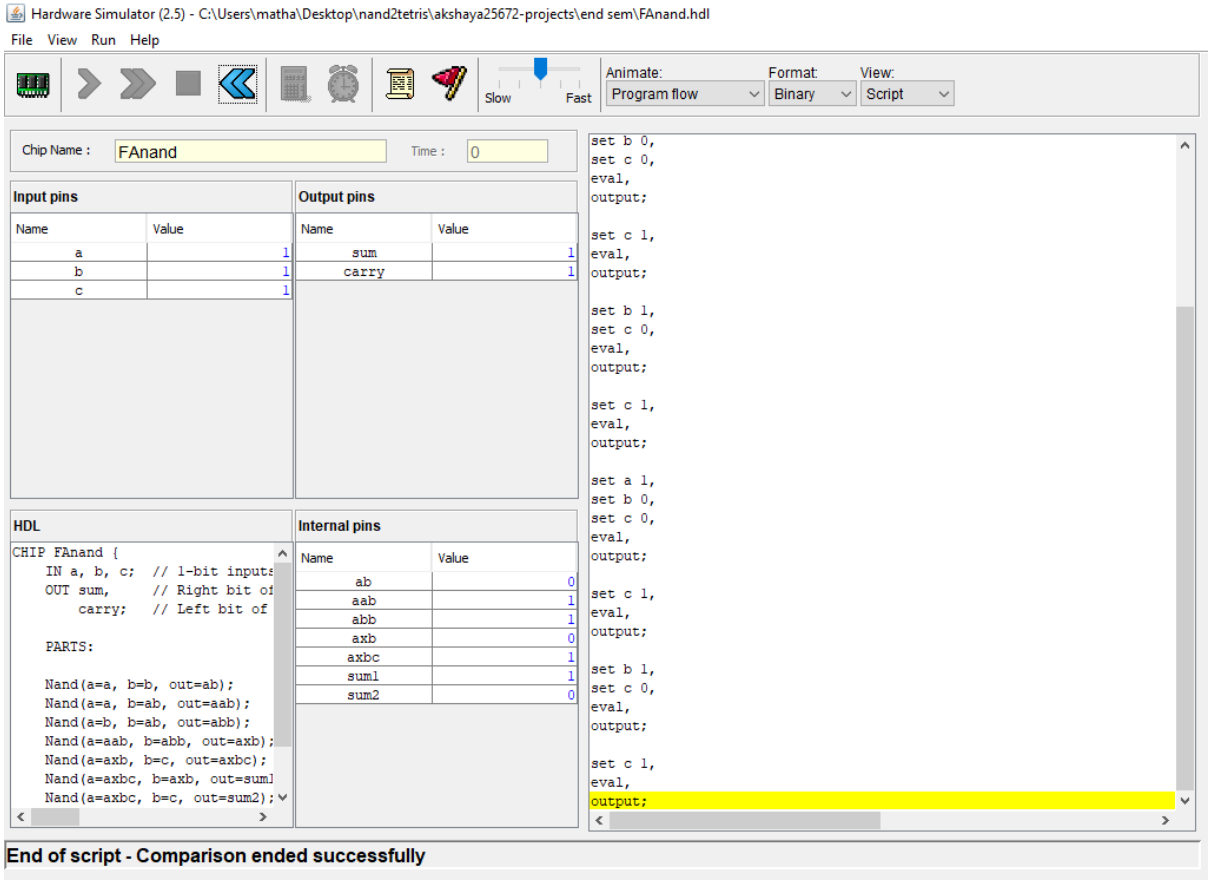


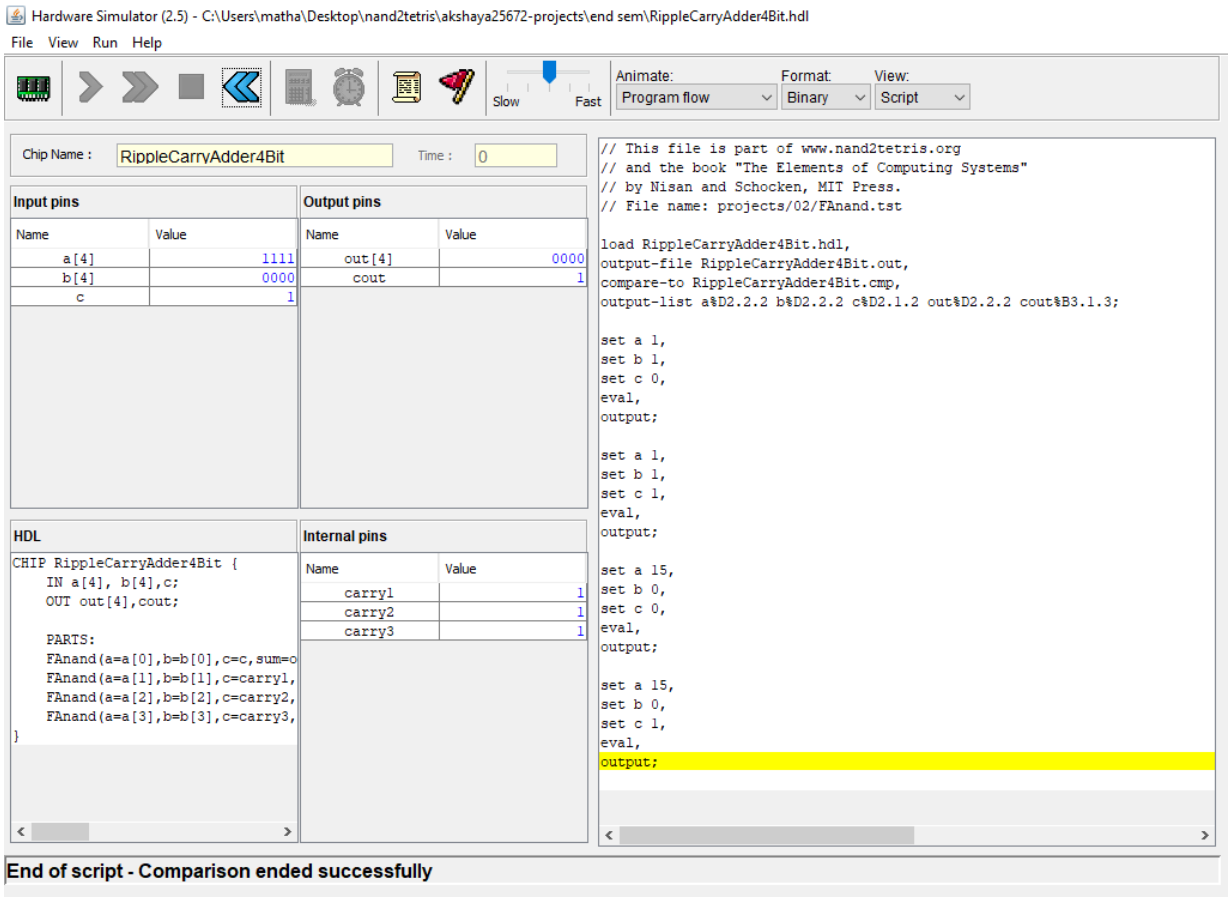
For another instance we can consider a[16]=1111111111111111 and b[16] = 0000000000000001

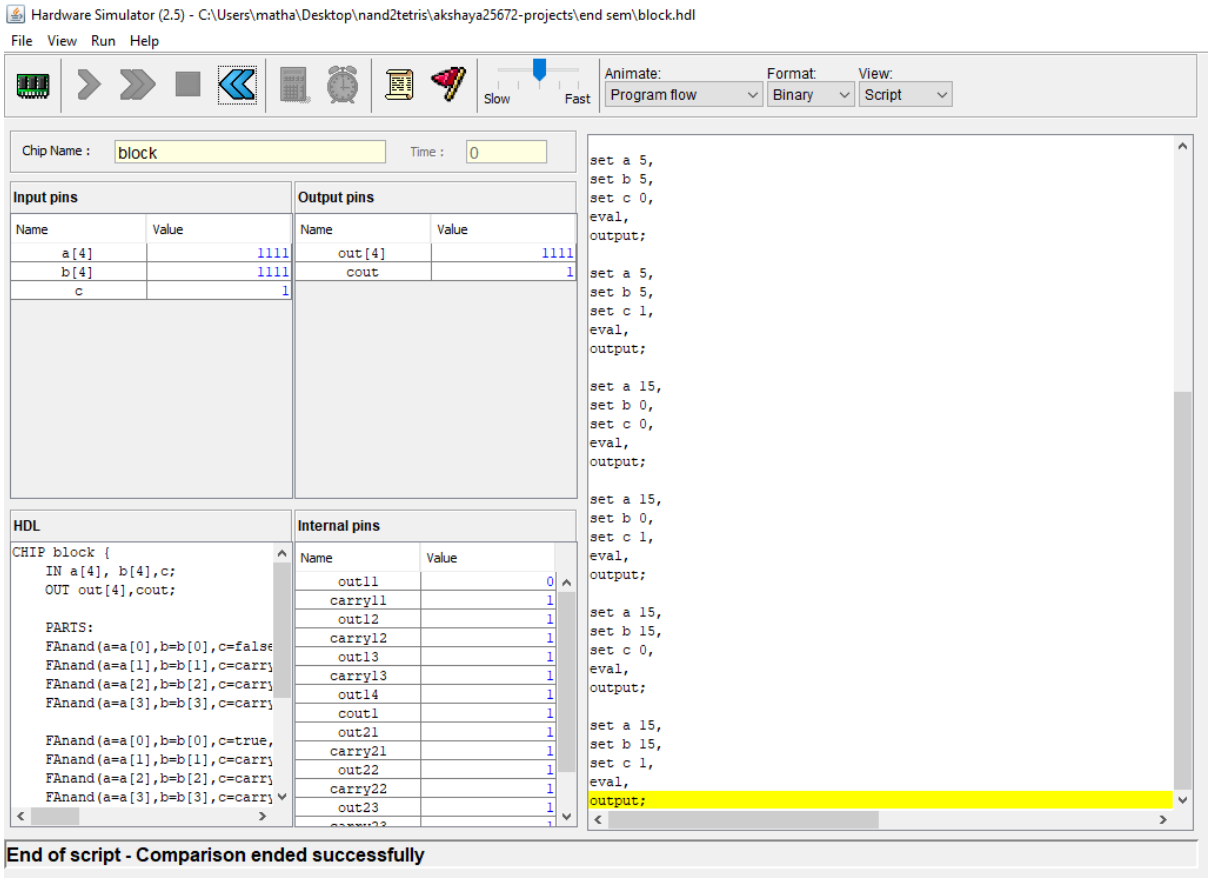


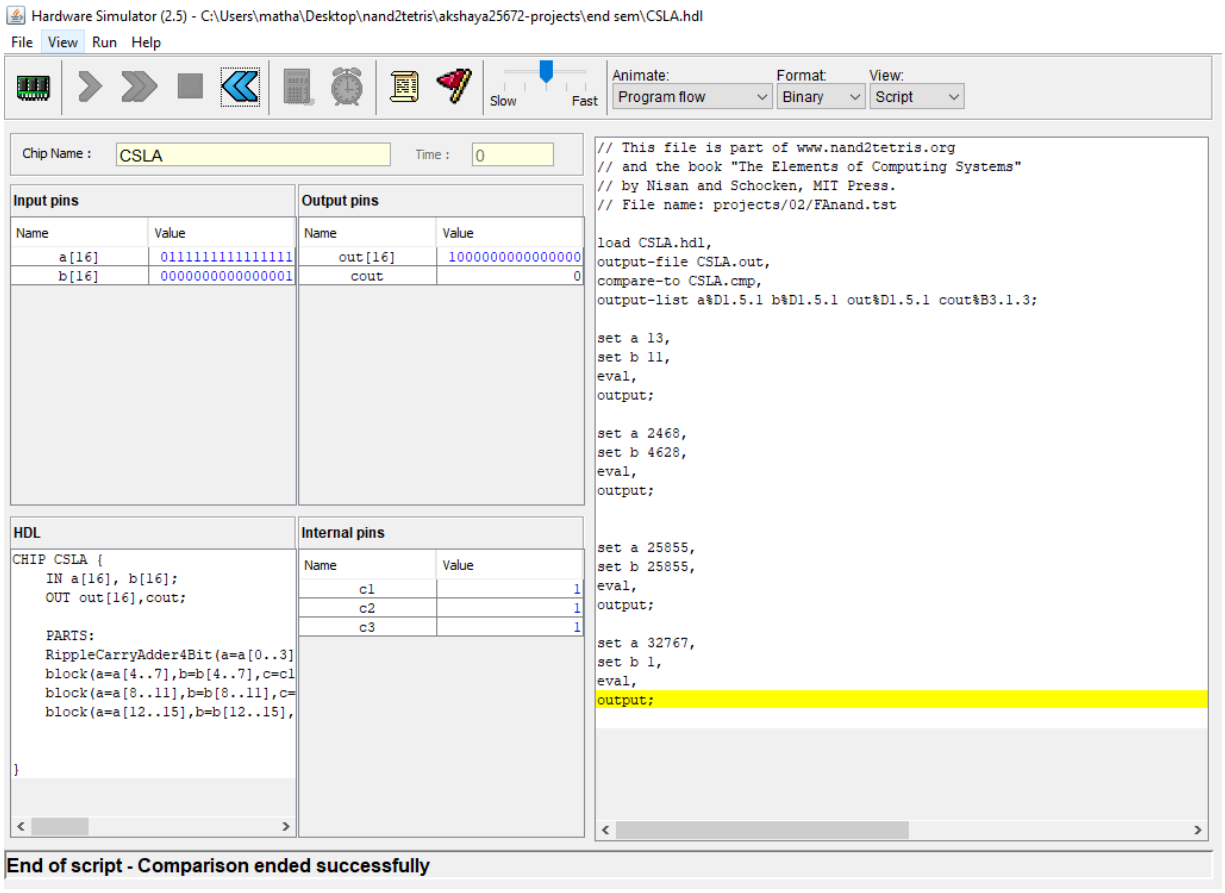
Now on loading all the above test scripts in hardware simulator we get the following outputs.

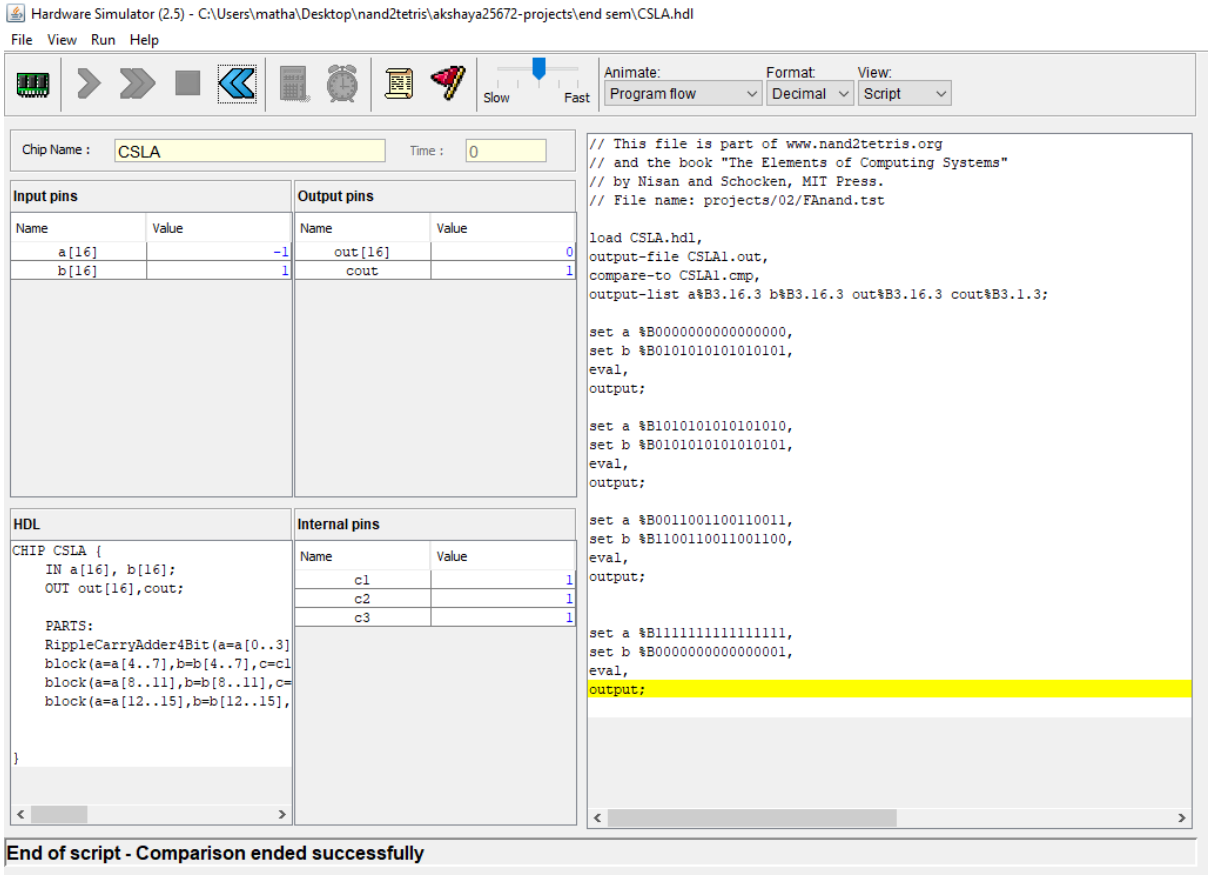





Hardware Simulator (2.5) - C:\Users\matha\Desktop\nand2tetris\akshaya25672-projects\end sem\RippleCarryAdder4Bit.hdl
File View Run Help
Slow
Fast
Animate:
Program flow
Format:
Binary
View:
Script
Chip Name :
RippleCarryAdder4Bit
Time :
0
Input pins
Name
Value
a[4]
1111
b[4]
0000
c
1
Output pins
Name
Value
out[4]
0000
cout
1
HDL
CHIP RippleCarryAdder4Bit {
IN a[4], b[4],c;
OUT out[4],cout;
PARTS:
FAnand(a=a[0],b=b[0],c=c,sum=o
FAnand(a=a[1],b=b[1],c=carry1,
FAnand(a=a[2],b=b[2],c=carry2,
FAnand(a=a[3],b=b[3],c=carry3,
}
Internal pins
Name
Value
carry1
1
carry2
1
carry3
1
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/02/FAnand.tst
load RippleCarryAdder4Bit.hdl,
output-file RippleCarryAdder4Bit.out,
compare-to RippleCarryAdder4Bit.cmp,
output-list a%D2.2.2 b%D2.2.2 c%D2.1.2 out%D2.2.2 cout%B3.1.3;
set a 1,
set b 1,
set c 0,
eval,
output;
set a 1,
set b 1,
set c 1,
eval,
output;
set a 15,
set b 0,
set c 0,
eval,
output;
set a 15,
set b 0,
set c 1,
eval,
output;
End of script - Comparison ended successfully


Hardware Simulator (2.5) - C:\Users\matha\Desktop\nand2tetris\akshaya25672-projects\end sem\block.hdl
File View Run Help
Slow
Fast
Animate:
Program flow
Format:
Binary
View:
Script
Chip Name :
block
Time :
0
Input pins
Name
Value
a[4]
1111
b[4]
1111
c
1
Output pins
Name
Value
out[4]
1111
cout
1
HDL
CHIP block {
IN a[4], b[4],c;
OUT out[4],cout;
PARTS:
FAnand(a=a[0],b=b[0],c=false
FAnand(a=a[1],b=b[1],c=carry
FAnand(a=a[2],b=b[2],c=carry
FAnand(a=a[3],b=b[3],c=carry
FAnand(a=a[0],b=b[0],c=true,
FAnand(a=a[1],b=b[1],c=carry
FAnand(a=a[2],b=b[2],c=carry
FAnand(a=a[3],b=b[3],c=carry
Internal pins
Name
Value
out11
0
carry11
1
out12
1
carry12
1
out13
1
carry13
1
out14
1
cout1
1
out21
1
carry21
1
out22
1
carry22
1
out23
1
set a 5,
set b 5,
set c 0,
eval,
output;
set a 5,
set b 5,
set c 1,
eval,
output;
set a 15,
set b 0,
set c 0,
eval,
output;
set a 15,
set b 0,
set c 1,
eval,
output;
set a 15,
set b 15,
set c 0,
eval,
output;
set a 15,
set b 15,
set c 1,
eval,
output;
End of script - Comparison ended successfully


Hardware Simulator (2.5) - C:\Users\matha\Desktop\nand2tetris\akshaya25672-projects\end sem\CSLA.hdl
File View Run Help
Slow
Fast
Animate:
Program flow
Format:
Binary
View:
Script
Chip Name :
CSLA
Time :
0
Input pins
Name
Value
a[16]
0111111111111111
b[16]
0000000000000001
Output pins
Name
Value
out[16]
1000000000000000
cout
0
HDL
CHIP CSLA {
IN a[16], b[16];
OUT out[16],cout;
PARTS:
RippleCarryAdder4Bit(a=a[0..3]
block(a=a[4..7],b=b[4..7],c=cl
block(a=a[8..11],b=b[8..11],c=
block(a=a[12..15],b=b[12..15],
}
Internal pins
Name
Value
c1
1
c2
1
c3
1
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/02/FAnand.tst
load CSLA.hdl,
output-file CSLA.out,
compare-to CSLA.cmp,
output-list a%D1.5.1 b%D1.5.1 out%D1.5.1 cout%B3.1.3;
set a 13,
set b 11,
eval,
output;
set a 2468,
set b 4628,
eval,
output;
set a 25855,
set b 25855,
eval,
output;
set a 32767,
set b 1,
eval,
output;
End of script - Comparison ended successfully


Hardware Simulator (2.5) - C:\Users\matha\Desktop\nand2tetris\akshaya25672-projects\end sem\CSLA.hdl
File View Run Help
Slow
Fast
Animate:
Program flow
Format:
Decimal
View:
Script
Chip Name :
CSLA
Time :
0
Input pins
Name
Value
a[16]
-1
b[16]
1
Output pins
Name
Value
out[16]
0
cout
1
HDL
CHIP CSLA {
IN a[16], b[16];
OUT out[16],cout;
PARTS:
RippleCarryAdder4Bit(a=a[0..3]
block(a=a[4..7],b=b[4..7],c=cl
block(a=a[8..11],b=b[8..11],c=
block(a=a[12..15],b=b[12..15],
}
Internal pins
Name
Value
c1
1
c2
1
c3
1
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/02/FAnand.tst
load CSLA.hdl,
output-file CSLA1.out,
compare-to CSLA1.cmp,
output-list a%B3.16.3 b%B3.16.3 out%B3.16.3 cout%B3.1.3;
set a %B0000000000000000,
set b %B0101010101010101,
eval,
output;
set a %B1010101010101010,
set b %B0101010101010101,
eval,
output;
set a %B0011001100110011,
set b %B1100110011001100,
eval,
output;
set a %B1111111111111111,
set b %B0000000000000001,
eval,
output;
End of script - Comparison ended successfully

CONCLUSION:

The carry select adder is used to add two n-bit numbers. We have added two 16-bit numbers and generated one sum and carry as output. The computation time taken by Carry Select Adder is less than that of the ripple carry adder. The computation time of Ripple Carry Adder is the product of delay through one adder cell and the number of bits the number that is to be added is. In our case, we add two 16-bit numbers. So, the computation delay of ripple carry adder is 16 times the delay through each adder cell. The computation time of Carry Select Adder is the sum of time needed for the multiplexer of the next block to select the actual output carry and the product of delay through each adder cell and half the number of bits of the numbers that is to be added. In our case, as we take a 16-bit number, the total computation time will be sum of time needed by the multiplexer of the next block to select the actual output carry and 8 times the delay through each adder cell. From the above two observations, we can say that the computation time taken by Carry Select Adder is less than any other adder. Thus, we have a logical circuit which can add two n-bit numbers with a less computation time.

Carry select adders are the most frequently used adders in computational circuits as they are practically faster than its traditional counterparts. Since it uses pre-computation logic and calculates sum and carry for each stage simultaneously through deployment of parallel computation, the delay for the carry propagation is reduced. It occupies low area, consumes low power, simple and efficient. Thus, Carry select adder is more useful and used for various purposes.

APPLICATIONS OF CARRY SELECT ADDER:

Arithmetic Logic Units

In computing, an arithmetic logic unit (ALU) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. It is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed; the ALU's output is the result of the performed operation. In many designs, the ALU also has status inputs or outputs, or both, which convey information about a previous operation or the current operation, respectively, between the ALU and external status registers. We know that the CSLA is efficient than other adders in terms of computational delay and power consumption due to which CSLA is preferred over other adders in ALU.

High Speed Multiplications

Multiplication involves two basic operations: the generation of partial products and their accumulation. Consequently, there are two ways to speed up multiplication: reduce the number of partial products or accelerate their accumulation. Clearly, a smaller number of partial products also reduces the complexity, and, as a result, reduces the time needed to accumulate the partial products. High-speed multipliers can be classified into three general types. The first generates all partial products in parallel, and then uses a fast multi-operand adder for their accumulation. This is known as a parallel multiplier. The second, known as a high-speed sequential multiplier, generates the partial products sequentially and adds each newly generated product to the previously accumulated partial product. The third is made up of an array of identical cells that generate new partial products and accumulate them simultaneously. Thus, there are no separate circuits for partial product generation and for their accumulation. This is known as an array multiplier, and it tends to have a reduced execution time, at the expense of increased hardware complexity. High-speed multiplication needs a large arrays to be evaluated using the exponential operations which sequentially need large partial sum and partial carry registers. Thus, CSLA being both fast and consumes less power it is used in high-speed multiplications chips.

Finite Impulse Response

In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying). The impulse response (that is, the output in response to a Kronecker delta input) of an Nth-order discrete-time FIR filter lasts exactly N + 1 samples (from first nonzero element through last nonzero element) before it then settles to zero. FIR filters can be discrete-time or continuous-time, and digital or analogue. FIR filtering is one of the mostly used operations in DSP. There is necessity for a reduction in power therefore FIR is implemented with CSLA for efficiency for high-speed digital signal processing. While comparing FIR implementation with CSLA and other adders CSLA is proved to be efficient in terms of both time delay and power consumption.

Advanced Microprocessor Design

A microprocessor is an electronic component that is used by a computer to do its work. It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together. Some microprocessors in the 20th century required several chips. Microprocessors help to do everything from controlling elevators to searching the Web. Everything a computer does is described by instructions of computer programs, and microprocessors carry out these instructions many millions of times a second. Microprocessors were invented in the 1970s for use in embedded systems. The majority are still used that way, in such things as mobile phones, cars, military weapons, and home appliances. Some microprocessors are microcontrollers, so small and inexpensive that they are used to control very simple products like flashlights and greeting cards that play music when opened. A few especially powerful microprocessors are used in personal computers. Addition being the fundamental operation for any digital system or any control system and adder being used for different operations extensively and hence adder becomes a primary component with efficiency due to which CSLA is preferred over other adders.

Digital Signal Processing

Digital signal processing (DSP) is the use of digital processing, such as by computers or more specialized digital signal processors, to perform a wide variety of signal processing operations. The digital signals processed in this manner are a sequence of numbers that represent samples of a continuous variable in a domain such as time, space, or frequency. In digital electronics, a digital signal is represented as a pulse train, which is typically generated by the switching of a transistor. Digital signal processing and analogue signal processing are subfields of signal processing. DSP applications include audio and speech processing, sonar, radar and other sensor array processing, spectral density estimation, statistical signal processing, digital image processing, data compression, video coding, audio coding, image compression, signal processing for telecommunications, control systems, biomedical engineering, and seismology, among others.DSP can involve linear or nonlinear operations. Nonlinear signal processing is closely related to nonlinear system identification and can be implemented in the time, frequency, and spatio-temporal domains. The application of digital computation to signal processing allows for many advantages over analogue processing in many applications, such as error detection and correction in transmission as well as data compression. Digital signal processing is also fundamental to digital technology, such as digital telecommunication and wireless communications. DSP is applicable to both streaming data and static (stored) data. As we have already seen for the FIR, for the DSP to work efficiently there is a need for less time delay and less power consumption and hence carry select adder is used.

Digital Image Processing

Digital image processing is one of the extensively used techniques in real life application. Digital image processing is the use of a digital computer to process digital images through an algorithm. As a subcategory or field of digital signal processing, digital image processing has many advantages over analogue image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and distortion during processing. Since images are defined over two dimensions (perhaps more) digital image processing may be modelled in the form of multidimensional systems. The generation and development of digital image processing are mainly affected by three factors: first, the development of computers; second, the development of mathematics (especially the creation and improvement of discrete mathematics theory); third, the demand for a wide range of applications in environment, agriculture, military, industry and medical science has increased. The use of carry select adders in Image addition shows reduced propagation delay and fast addition.

REFERENCE:

- http://nand2tetris-questions-and-answers-forum.32033.n3.nabble.com/Errata-Bugs-and-Such-f32642.html
- The elements of computing systems building a modern computer from first principles by Nisan, NoamSchocken, Shimon (z-lib.org)
- https://en.wikipedia.org/wiki/Carry-select_adder
- https://tams.informatik.uni-hamburg.de/applets/hades/webdemos/20-arithmetic/20-carryselect/adder_carryselect.html
- http://www.aoki.ecei.tohoku.ac.jp/arith/mg/algorithm.html#fsa_csl

---

SUBMITTED BY: [Team_2]

#Akshaya.J – 25672
#Ameen Ashadhullah – 36662
#Bhoomika.M – 12929
#Ghaayathri Devi.K – 30132
#Gokul.R – 20368