American Journal of Circuits, Systems and Signal Processing
Vol. 1, No. 3, 2015, pp. 69-85
http://www.aiscience.org/journal/ajcssp

High Level Modelling and Relative Comparison of Different Full Adder Structures

Negin Mahani*

Computer Engineering Faculty, Shahid Bahonar University, Zarand High Education Centre, Kerman, Iran

Abstract

Comparing different full adders is very significant for VLSI design as they are essential components in almost all digital circuits. The science has yield the benchmarking work laborious as often distinct implementation techniques and technologies have been used in the design. Additionally, the design characteristics which are selected for performance analysis are not consistent. This paper shows the results of comparing four adder structures by implementing them all with the same technology and the same level of abstraction.

Keywords

Carry-Select Adder, Carry-Look-Ahead Adder, Carry-Skip Adder, Brent-Kung Adder

Received: May 19, 2015 / Accepted: June 21, 2015 / Published online: July 13, 2015

1. Introduction

A one-bit full adder is a very important leaf cell in the design of Application Specific Integrated Circuits. This paper gives guidelines for high level modelling in Verilog. We have modelled four different full adders in high level with Verilog and then we have made some comparison on them based on their area, delay and etc. We also have simulated and synthesized the adders. So in this paper we have introduced some basic concepts of description levels of HDLs like Verilog and also we have talked about simulation and synthesis.

Here we have modelled different full adders like Carry-Look-Ahead adder, Carry-Skip adder, Carry-Select adder and Brent-Kung adder using high level descriptions. This document consists of some parts as they are described below: The structures of the full adders are described in section 2, while the implementation, modelling and simulation methodology is explained in section 3. Next in section 4, the results are summarized. Finally, conclusions are drawn in section 5.

2. Descriptions of Four Different Full Adders

Here are the structures of different full adders such as: Carry-Look-Ahead adder, Carry-Select adder, Brent-Kung adder and also Carry-Skip adder.

A. CARRY-LOOKAHEAD ADDER (CLA)

CLA can generate all the carries in parallel. As following by applying the equations in Fig.1 part (a) recursively, all the C_{i+1}s can be generated based on G_i, P_i and C_0.

$C_1 = G_0 + C_0P_0$

$C_2 = G_1 + C_1P_1$

$= G_1 + G_0P_1 + C_0P_0 P_1$

$C_3 = G_2 + C_2P_2$

$= G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0 P_1P_2.....$

$C_{i+1} = G_i + G_{i-1}P_i + G_{i-2}P_{i-1}P_i + \ldots + G_0P_1P_2\ldots P_i + C_0P_0 P_1P_2\ldots P_i.....$

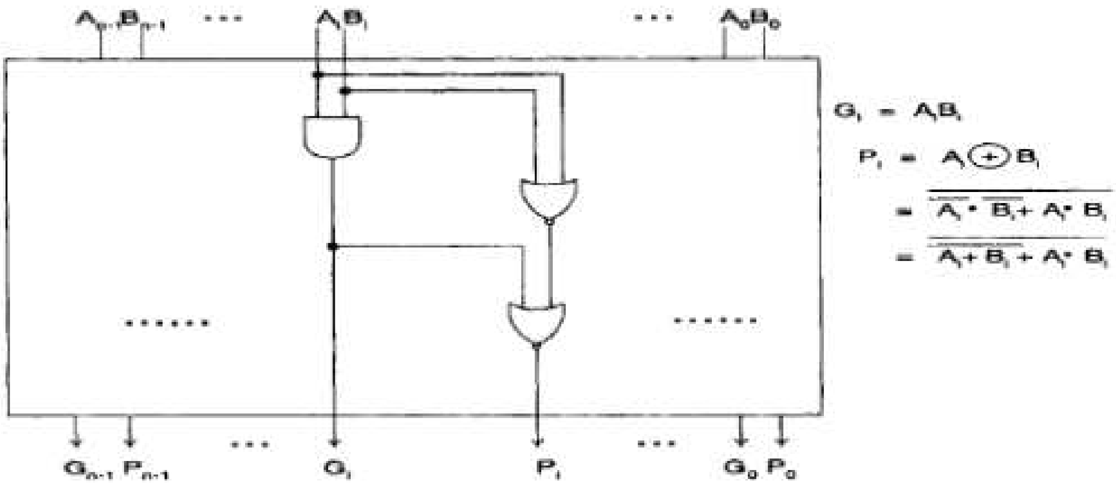
$C_n = G_{n-1} + G_{n-2}P_{n-1} + \ldots + G_0P_1P_2\ldots P_{n-1} + C_0P_0 P_1P_2\ldots P_{n-1}.$

* Corresponding author
E-mail address: Negin.Mahani@uk.ac.ir

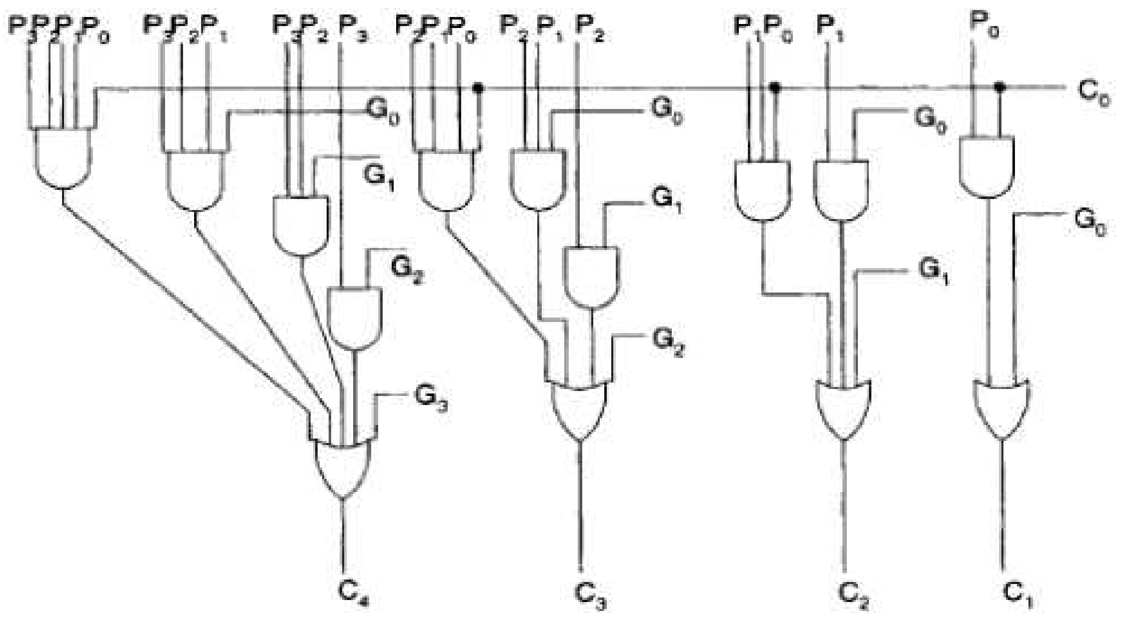
The circuit to generate C_{i+l}s is called Carry-Look-Ahead unit shown in part (b) of Fig.1.The final sum S can be computed once C_is, i = 1, 2. . . N, are known. Part(c) in Fig.1 shows the structure of summation. One can see that in Equation above, the number of terms in the OR function is as big as n + 1, and in the last term the number of valuables in the AND function is also n + 1. Fan-in and fan-out parameters could be a problem in Carry-Look-Ahead adder, as the number of bits (n) increases. On the other hand, sequential generation of each Ci+l is too slow. Instead, block Carry-Look-Ahead adder (BCLA) can be adopted in which groups of carries are generated in parallel [1][6].

B. CARRY-SELECT ADDER

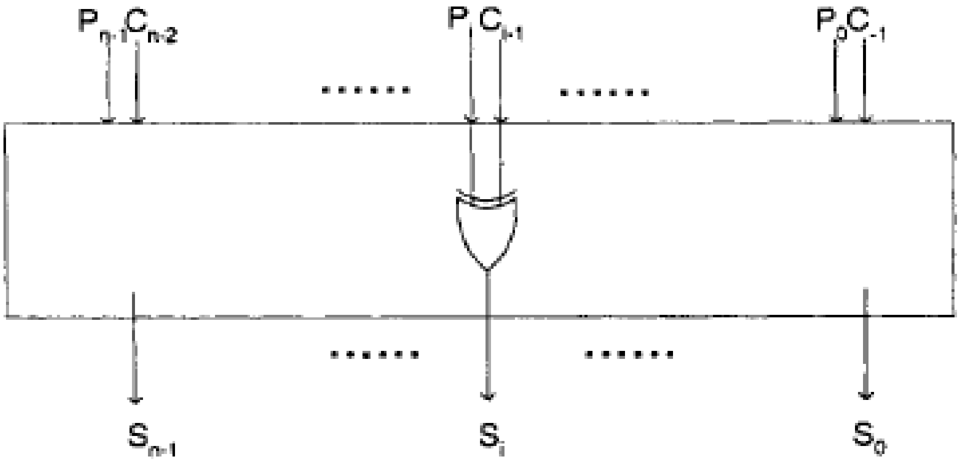
A Carry-Select adder is divided into parts that each of which accomplish two additions in concurrent, one assuming a carry-in of zero, the other a carry-in of one, except for the least significant. The 16-bit carry-select adder of Fig.2 is divided into sectors of lengths 1, 2, 3, 4, and 6. The 4-bit sector of Fig.2 (b) illustrates the common principle. Within the sector, there are two 4-bit ripple- carry adders take the same data inputs but different carry-ins. The upper adder has a carry-in of zero; the lower adder a carry-in of one. The real carry-in from the prior sector chooses one of the two adders. Comparing to a Ripple-Carry adder instead of having to ripple through four full adders, the carry now only has to pass through an individual multiplexer [7].



(a) Carry Generate/Propagate Unit

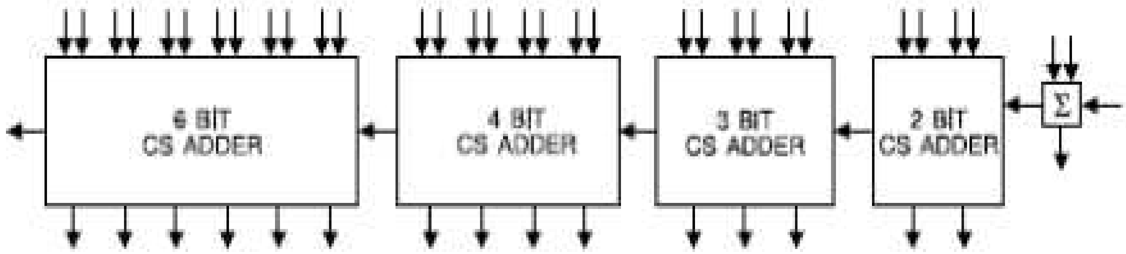


(b) Carry-Lookahead Unit

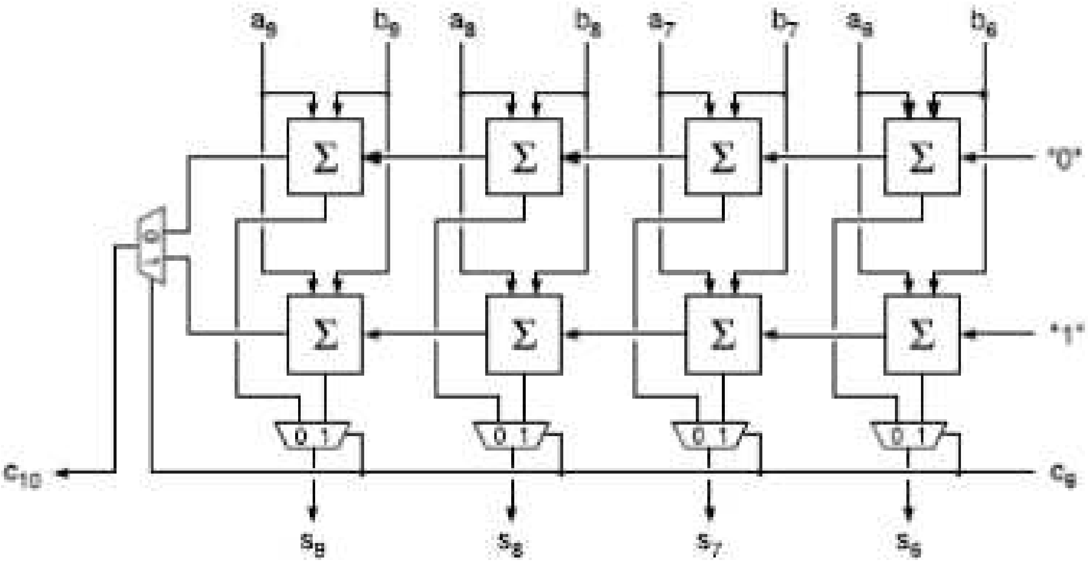


(c) Summation Unit

Fig. 1. Carry-Look-Ahead adder [10].



(a). 16-bit Carry-Select adder



(b). 4-bit Sector (Schematic)

Fig. 2. Carry-Select adder.

C. BRENT-KUNG ADDER

The following picture shows a Brent-Kung full adder (Fig.3.)[11]. The Brent-Kung adder has a gate level depth of $O(log_2(n))$. Adding two n-bit numbers is performed as adding in parallel the lower halves of the addends, the lower halves with input carry 0, and the upper halves with input carry 1, then selecting the appropriate upper half sum based on the output carry from the lower half sum. Thus the gate level depth of the n-bit adder is equal to the depth of the half-width adder plus the depth of a mux. The following is the formula that is the basis of this adder:

$$(g, p) = (g_m, p_m) . (g_1, p_1) = (g_m + p_m . g_1, p_m . p_1) \quad [2]$$

D. CARRY-SKIP ADDER

To speed-up operation, propagation is skipped to position i without waiting for rippling. Operation time varies according to operands as in carry-complete addition. To implement Carry-Skip adder, stages are divided into blocks (Fig.4.)[3].

Carry-Skip logic is added to each block to detect when carry-in the block can be passed directly to the next block. Define carry transfer: $t_i = a_i + b_i$, carry skipping can be detected for a block size of m as follows (carry propagates through all stages):

$$T_j \bullet T_{j+1} \ldots T_{j+m-1} = 1 \; (= (a_j + b_j) \bullet (a_{j+1} + b_{j+1}) \ldots)$$

This method takes into account both propagated and generated carries. Block size in carry-skip adder is very important (Fig.5.) Worst case operation time takes place when:

– Carry is generated in the first block

– Carry skips intermediate stages

– Carry is killed in the last block [4][8].

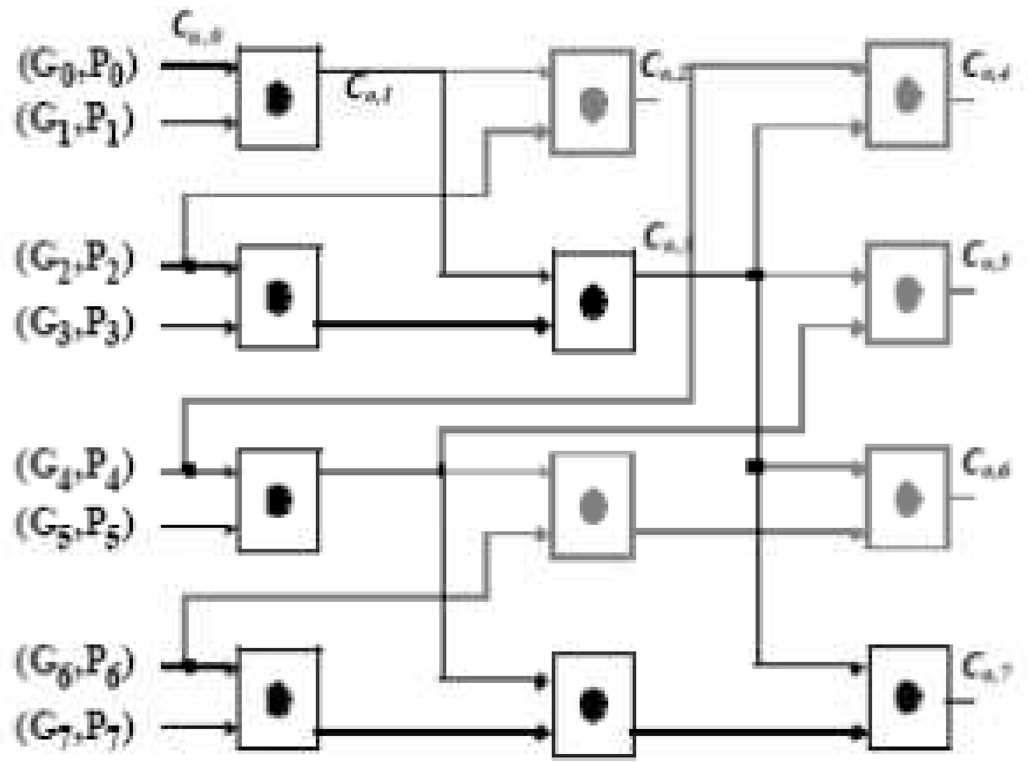


Fig. 3. Brent-Kung adder [3].

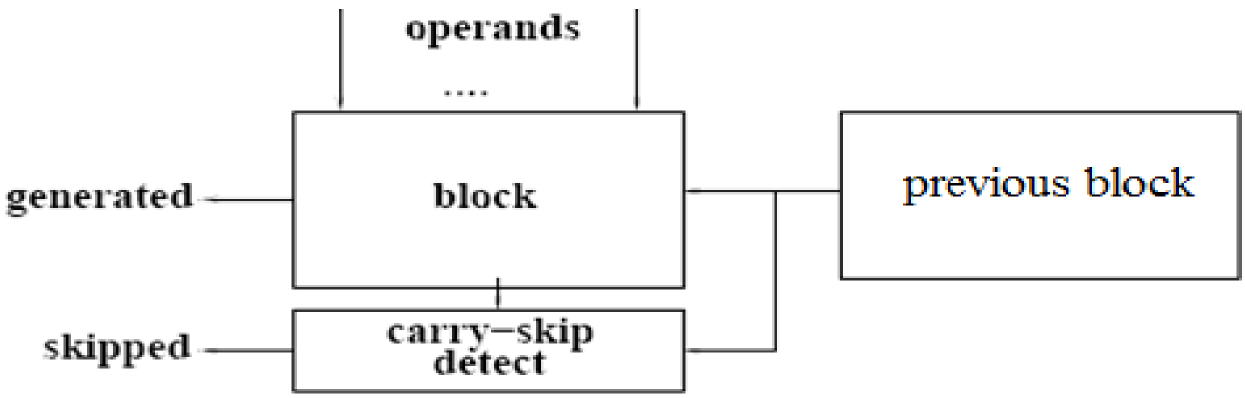


Fig. 4. Carry skipping.

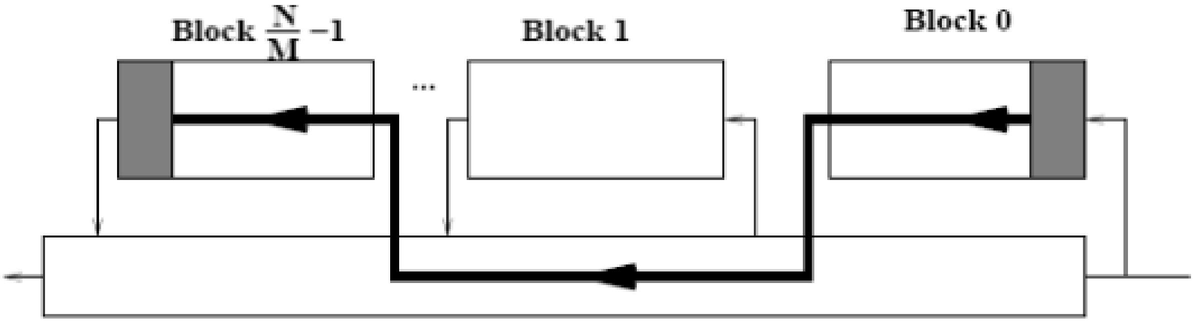


Fig. 5. Carry-Skip logic.

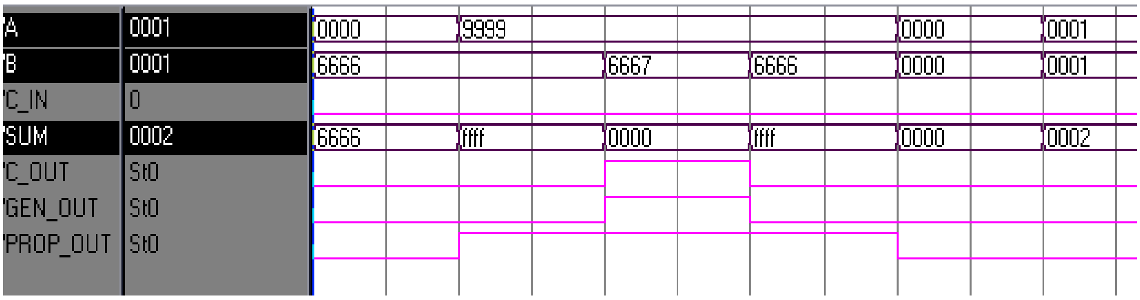


Fig. 6. Simulation result of Carry-Look-Ahead adder.

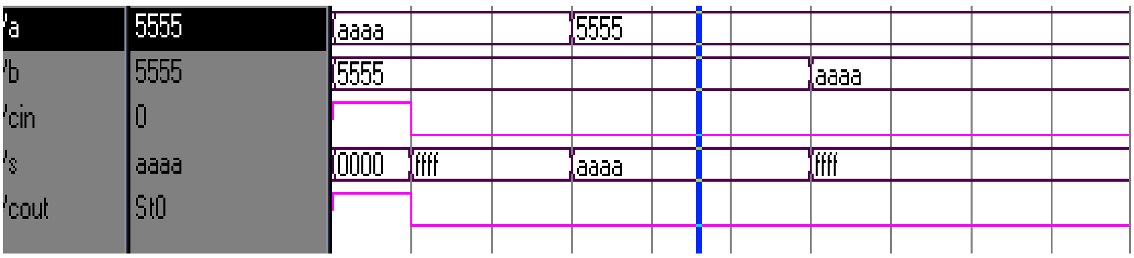


Fig. 7. Simulation result of Carry-Select adder.

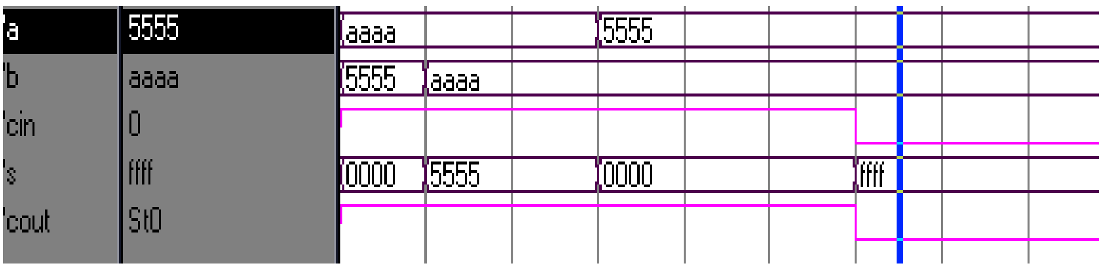


Fig. 8. Simulation result of Brent-Kung adder.

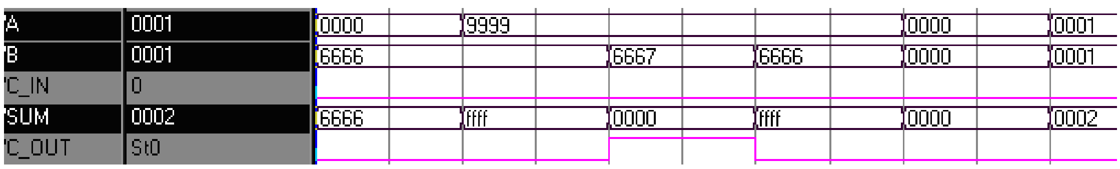


Fig. 9. Simulation result of Carry-Skip adder.

3. Methodology

We have implemented our modules in Verilog. The modules are named as belonged to a special kind of adder such as: Carry-Look-Ahead adder, Carry-Skip adder, Carry-Select adder and Brent-Kung adder using high level descriptions. Then by scripting we have simulated them and after all we have used Synplify and Leonardo to synthesize Brent-Kung adder. We have made optimization based on area/delay or both of them. At last we have made comparisons on them. In this work we have used scripting for simulation with Modelsim (appendix1). The results of simulation and synthesis have come in the result section. Interesting observations could be gotten of them. Appendix1also has some parts of report files of the synthesis.

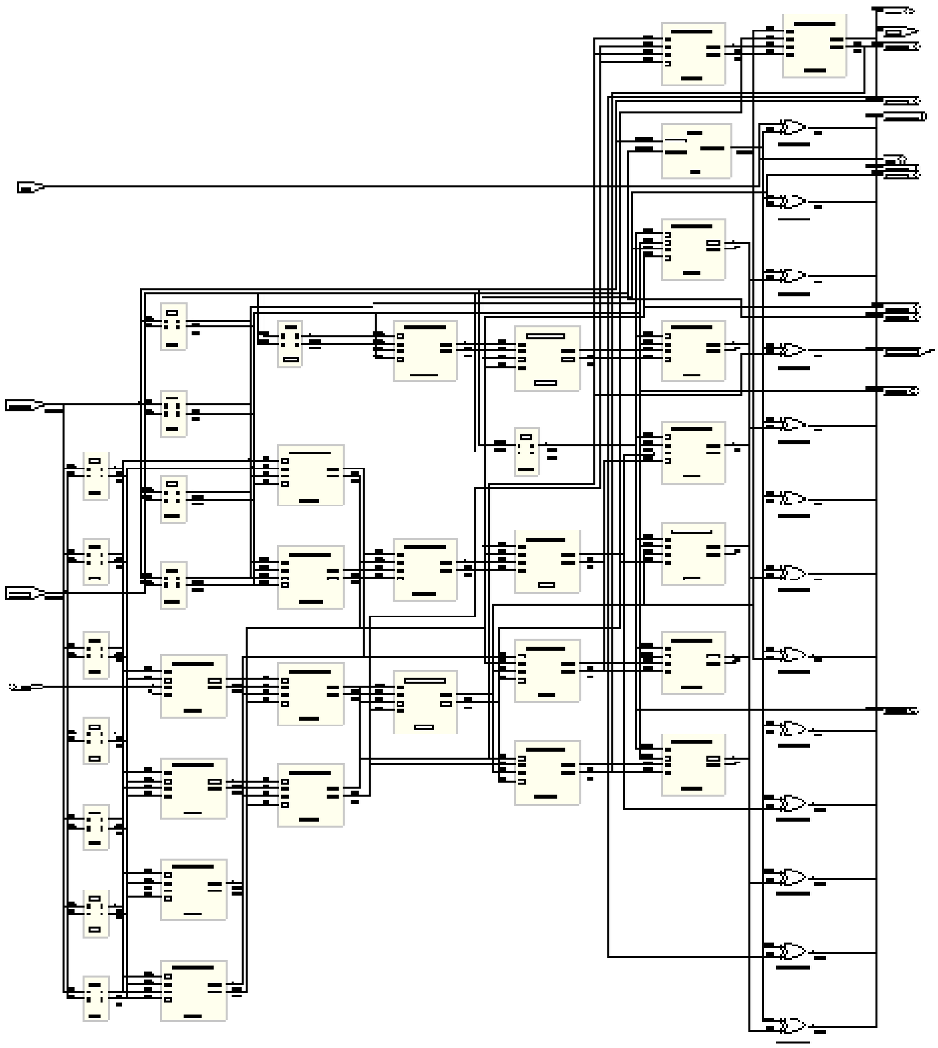


Fig. 10. Brent-Kung adder synthesis result by Synplify RTL view.

4. Results of Simulations and Synthesis Processes

The results of the simulations and synthesis are shown in Fig.6 to Fig.13. The result of Modelsim simulation of Carry-Look-Ahead adder is depicted in Fig. 6. Resulted wave forms of Modelsim simulation of Carry-Select, Brent-Kung and Carry- Skip adders are illustrated in Fig.7, Fig.8 and Fig.9 respectively.

Brent-Kung adder synthesis result by Synplify RTL view is in Fig.10. Brent-Kung adder synthesis result by Synplify technology view is in Fig.11. Brent-Kung adder synthesis result by Leonardo is in Fig.12. Brent-Kung adder synthesis result _critical path_ is in Fig.13.

5. Summary and Conclusion

In this paper we have showed how to implement codes in high level and also we have got familiar with modularity coding. We have used Modelsim for simulation and verification and also we have shown that we can use scripting for simulations. Most of the adders discussed in this paper are applicable to general purpose designs, with a few exceptions. The first exception is the Carry-Skip adder, which is the slowest adder for all bit sizes. It has also larger area requirements and higher active capacitance than the Ripple adder or the Transmission Gate adder for all bit sizes. It may always be replaced with either one of these adder structures. The second adder structure which can be always replaced is the Carry-Look-Ahead adder. But based on previous works for 8 to 32 bit circuits the Conditional-Sum adder has better results than the Carry-Look-Ahead adder. For the 4-bit adder this adder structure can be replaced with the Transmission-Gate adder or with the Ripple adder, because both of these structures have better results [5].

Brent-Kung adder requires 2(log N −1) stages. In Brent-Kung adder using binary tree for carry propagation leads to logarithmic delay. Its area is twice as large as ripple adder and layout of the cells is very compact. Once carry signals are ready, sum bits derived in constant time. It is good for wide adders. Carry-Look-Ahead adder calculates carries in advance. Limited fan-in for NAND gate is less than or equal to 5. It is impractical for Ci with i >4 and need 2-level CLA with block size k. CLA compared to Ripple-Carry adder is Faster, but delay is still linear and has larger area. The limitation is that it cannot go beyond 4 bits of look-ahead and large p, g fan-out slows down carry generation. Carry-Select adder calculates two cases simultaneously. Sum computed in one step after the intermediate carry signals are ready. Area overhead is about an additional carry path and a multiplexer (not the whole adder) and about 30% more than a Ripple-Carry. Its delay is sub-linear [9]. According to the simulation and synthesis results, the adder topology which has the best compromise between area, delay and power dissipation is carry look-ahead adder and it is suitable for high performance and low-power circuits. So the fastest adder is Carry-Select with the penalty of area. Carry-Skip adder improves on the delay of a Ripple-Carry adder with little effort compared to other adders. Carry-Select adder is one of the fastest adders to perform arithmetic operations. From the structure of CSL adder there is a scope for reducing the area and delay.

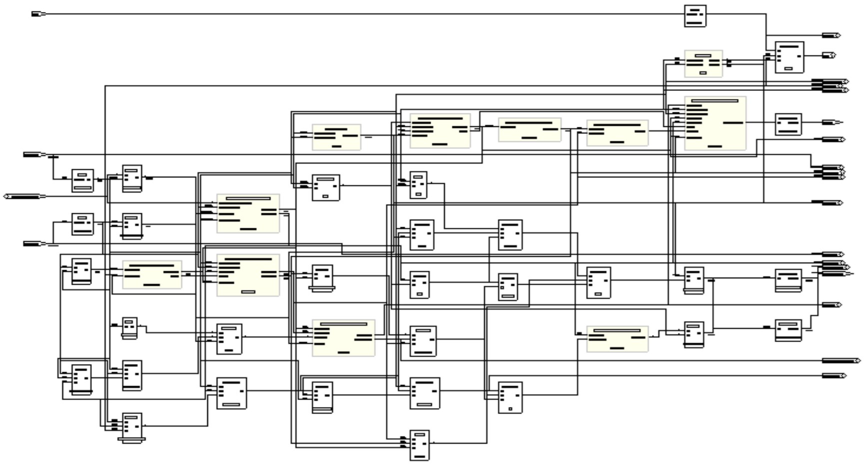


Fig. 11. Brent-Kung adder synthesis result by Synplify technology view.

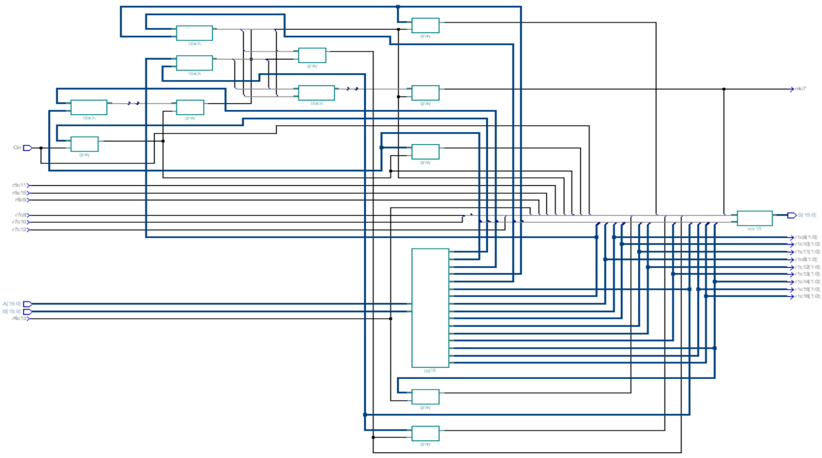


Fig. 12. Brent-Kung adder synthesis result by Leonardo.

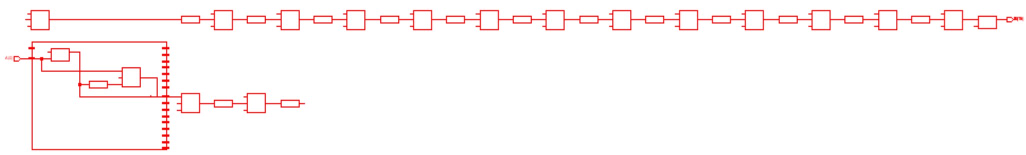


Fig. 13. Brent-Kung adder synthesis result _critical path.

Appendix1

Sunplify synthesis reports of Brent-Kung adder

Brent-Kung plg file of synplify

@P: Worst Slack : -6.187

@P: System - Estimated Frequency : 61.8 MHz

@P: System - Requested Frequency : 100.0 MHz

@P: System - Estimated Period : 16.187

@P: System - Requested Period : 10.000

@P: System - Slack : -6.187

@P: Total Area : 103.0

Rev8(brent_kung.htm)

Worst Path Information

Path information for path number 1:

Requested Period: 10.000

= Required time: 10.000

- Propagation time: 16.187

= Slack (critical) : -6.187

Number of logic level(s): 12

Starting point: B[15:0] / B[0]

Ending point: Sum[15:0] / Sum[11]

The start point is clocked by System [rising]

The end point is clocked by System [rising]

Total path delay (propagation time + setup) of 16.187 is 2.867(17.7%) logic and 13.320(82.3%) route.

Brent_kung.areasrr

Report for cell Brent_Kung_ADDER.verilog

Cell usage:

cell	count	area	count*area
IB33	33	0.0	0.0
OB33PH	17	0.0	0.0
...			
TOTAL	132		103.0

Leonardo synthesis reports of Brent-Kung adder

Auto optimization area

Cell: Brent_Kung16 View: INTERFACE Library: work

****

Cell	Library	References		Total Area
AN21	CUB	15 x	1	9 gates
AND2	CUB	16 x	1	10 gates
EN1	CUB	15 x	1	12 gates
EO1	CUB	1 x	1	1 gates
IN2	CUB	32 x	0	10 gates
OA21	CUB	16 x	1	12 gates
ON21	CUB	1 x	1	1 gates

Number of ports : 50

Number of nets : 129

Number of instances : 96

Number of references to this view : 0

Total accumulated area :

Number of gates : 54

Number of accumulated instances : 96

Delay

Critical Path Report

Critical path #1, (path slack = 1.8):

NAME	GATE	ARRIVAL		LOAD
A(0)/		0.00	0.00 up	0.11
ipg16_ix1/Q	AND2	0.29	0.29 up	0.11
ipg16_ix359/Q	IN2	0.13	0.41 dn	0.05
ipg16_ix7/Q	OA21	0.40	0.81 dn	0.14

ix431/Q	AN21	0.29	1.10 up	0.14
ix3/Q	IN2	0.17	1.27 dn	0.06
ix452/Q	AN21	0.30	1.57 up	0.14
ix7/Q	IN2	0.17	1.74 dn	0.06
ix426/Q	AN21	0.30	2.04 up	0.14
ix11/Q	IN2	0.17	2.21 dn	0.06
ix457/Q	AN21	0.32	2.53 up	0.16
ix15/Q	IN2	0.17	2.70 dn	0.06
ix421/Q	AN21	0.30	3.01 up	0.14
ix19/Q	IN2	0.17	3.17 dn	0.06
ix466/Q	AN21	0.32	3.49 up	0.16
ix23/Q	IN2	0.17	3.67 dn	0.06
ix416/Q	AN21	0.30	3.97 up	0.14
ix27/Q	IN2	0.17	4.14 dn	0.06
ix475/Q	AN21	0.32	4.46 up	0.16
ix31/Q	IN2	0.17	4.63 dn	0.06
ix411/Q	AN21	0.32	4.95 up	0.16
ix35/Q	IN2	0.17	5.13 dn	0.06
ix484/Q	AN21	0.30	5.43 up	0.14
ix39/Q	IN2	0.17	5.60 dn	0.06
ix406/Q	AN21	0.30	5.90 up	0.14
ix43/Q	IN2	0.17	6.07 dn	0.06
ix493/Q	AN21	0.32	6.39 up	0.16
ix47/Q	IN2	0.17	6.56 dn	0.06
ix401/Q	AN21	0.32	6.88 up	0.16
ix51/Q	IN2	0.17	7.05 dn	0.06
ix502/Q	AN21	0.30	7.36 up	0.14
ix55/Q	IN2	0.17	7.53 dn	0.06
ix396/Q	AN21	0.30	7.83 up	0.14
ix95/Q	EN1	0.33	8.16 up	0.02
S(15)/		0.00	8.16 up	0.00

data arrival time 8.16

data required time (default specified) 10.00

----

data required time 10.00

data arrival time 8.16

----

slack 1.84

Delayoptimization

delay

data required time 10.00

data arrival time 7.99

----

slack 2.01

area

Cell	Library	References		Total Area
AN21	CUB	31 x	1	19 gates
AND2	CUB	16 x	1	10 gates
EN1	CUB	15 x	1	12 gates
EO1	CUB	1 x	1	1 gates
IN2	CUB	48 x	0	15 gates
ON21	CUB	1 x	1	1 gates

Number of ports : 50

Number of nets : 145

Number of instances : 112

Number of references to this view : 0

Total accumulated area :

Number of gates : 57

Number of accumulated instances : 112

area optimization

area

Cell	Library	References		Total Area
AN21	CUB	15 x	1	9 gates
AND2	CUB	16 x	1	10 gates
EN1	CUB	15 x	1	12 gates
EO1	CUB	1 x	1	1 gates
IN2	CUB	31 x	0	10 gates
OA21	CUB	16 x	1	12 gates
ON21	CUB	1 x	1	1 gates

```
 Number of ports :                    50
 Number of nets :                    128
 Number of instances :                 95
 Number of references to this view :     0

Total accumulated area :
 Number of gates :                    54
 Number of accumulated instances :      95
Delay
------------------------------------------------------------------
data required time                          10.00
data arrival time                           8.16
                                   ----------
slack                                 1.84
------------------------------------------------------------------
```

One sample script of these four adders

Brent-Kung adder

.tcl file

```
puts {
  Compile and Simulate script for divider
  Provided by Mahdi N. Bojnordi
}

cd ../sim/pre_syn

set library_file_list {
  design_library {../../model/Verilog/Brent_Kung16.v}
  test_library   {../../test/ Brent_Kung16_test.v }
}

set top_level  work.brent_test

set wave_patterns {
  /*
}
set wave_radices {
  hexadecimal {A,B,Cin,Sum,Cout}
}

set PrefMain(font) {
  Courier 10 roman normal
}

# compilation
vlib work
foreach {library file_list} $library_file_list {
  foreach file $file_list {
    if [regexp {.vhdl?$} $file] {
      vcom -93 $file
    } else {
      vlog $file
    }
  }
}

# simulation
eval vsim $top_level

# If waves are required
if [llength $wave_patterns] {
  noview wave
  foreach pattern $wave_patterns {
    add wave $pattern
  }
  configure wave -signalnamewidth 1
  foreach {radix signals} $wave_radices {
    foreach signal $signals {
      catch {property wave -radix $radix $signal}
    }
  }
}

# run the simulation
run -all
```

```
puts {
 job finished.
}
```

Verilog source codes for adders

****

Carry-Look-Ahead adder with test bench

```
module cla_16(a, b, c_in, sum, c_out, gen_out, prop_out);

input [15:0] a, b;      // numbers to add
input c_in;             // carry in
output [15:0] sum;      // sum
output c_out;           // carry of 16-bit addition
output gen_out;         // generate of 16-bit addition
output prop_out;        // propagate of 16-bit addition

wire [3:0] part_carry;  // calculated carry_out of
                        //each 4-bit part
wire [3:0] part_gen;    // generate of each 4-bit part
wire [3:0] part_prop;   // propagate of each 4-bit part
        // Make instances to calculate generates and
propagates.
        // First 4 instances calculate generate and propagate
        // for each 4-bit part.  Last instance calculates
        // generate and propagate for all 16 bits.

gen_prop gp0( (a[3:0] & b[3:0]), (a[3:0] | b[3:0]),part_gen[0],
part_prop[0]);

gen_prop gp1( (a[7:4] & b[7:4]), (a[7:4] | b[7:4]),part_gen[1],
part_prop[1]);

gen_prop gp2( (a[11:8] & b[11:8]), (a[11:8] |
b[11:8]),part_gen[2], part_prop[2]);

gen_prop gp3( (a[15:12] & b[15:12]), (a[15:12] |
b[15:12]),part_gen[3], part_prop[3]);

gen_prop gp(part_gen, part_prop, gen_out, prop_out);

// Make instances to calculate carries for each 4-bit part
carry c( part_gen, part_prop, c_in, part_carry );
assign c_out = part_carry[3];

// make 4-bit adders to do additions
assign sum[3:0] = a[3:0] + b[3:0] + c_in;
assign sum[7:4] = a[7:4] + b[7:4] + part_carry[0];
assign sum[11:8] = a[11:8] + b[11:8] + part_carry[1];
assign sum[15:12] = a[15:12] + b[15:12] + part_carry[2];
endmodule

module carry(gens_in, props_in, c_in, carries);
input [3:0] gens_in;  // generate for each of 4 parts
input [3:0] props_in; // propagate for each of 4 parts
input c_in;           // carry in
 output [3:0] carries;  // carry out for each of 4 parts

function [3:0] get_carries;
input [3:0] gens_in, props_in;
input c_in;
reg [3:0] carries;
integer i;

begin
  for (i = 0; i <= 3; i = i + 1)
    if (i == 0)
      carries[i] = gens_in[i] | props_in[i] & c_in;
    else
      carries[i] = gens_in[i] | props_in[i] &
                           carries[i-1];
  get_carries = carries;
end
endfunction

assign carries = get_carries(gens_in, props_in, c_in);
endmodule
```

```
module gen_prop(gens_in, props_in, gen_out, prop_out);

input [3:0] gens_in;  // generate for each of 4 parts
input [3:0] props_in; // propagate for each of 4 parts
output gen_out, prop_out;

function [1:0] get_gen_and_prop;
input [3:0] gens_in, props_in;
reg prop, gen;
integer i;

begin
  for (i = 0; i <= 3; i = i + 1) begin
    if (i == 0) begin
      gen = gens_in[i];
      prop = props_in[i];
    end else begin
      gen = gens_in[i] | props_in[i] & gen;
      prop = props_in[i] & prop;
    end
  end

get_gen_and_prop = {gen, prop};
end
endfunction

assign {gen_out, prop_out} = get_gen_and_prop(gens_in,
props_in);
endmodule

//////simulation
module cla_test;

reg [15:0] A, B;       // numbers to add
reg C_IN;              // carry in
trireg [15:0] SUM;     // sum
trireg C_OUT;          // carry of 16-bit addition
trireg GEN_OUT;        // generate of 16-bit addition
trireg PROP_OUT;       // propagate of 16-bit addition

cla_16 adder1 (A, B, C_IN, SUM, C_OUT, GEN_OUT,
PROP_OUT);

initial
begin
   $monitor("%0d SUM = %b A = %b B = %b C_IN = %b
C_OUT = %b, GEN_OUT = %b, PROP_OUT = %b",
                 $time, SUM, A, B, C_IN, C_OUT, GEN_OUT,
PROP_OUT);

   A = 16'b0000000000000000;
   B = 16'b0110011001100110;
   C_IN = 0;
   #10 A = 16'b1001100110011001;
   #10  B = 16'b0110011001100111;
   #10 C_IN = 0; B = 16'b0110011001100110;
   #10 A = 16'b0000000000000000; B =
16'b0000000000000000;
   #10 C_IN = 0; A = 16'b0000000000000001; B =
16'b0000000000000001;
   #10 $finish;
end
endmodule
```

Brent-Kung adder with test bench

```
module black (pg, pg0, pgo);

   input [1:0] pg, pg0;
   output [1:0] pgo;

   assign pgo[1] = pg[1] & pg0[1];
   assign pgo[0] = (pg0[0] & pg[1]) | pg[0];

endmodule
///////////////////////////////////////////////////
module gray (pg, pg0, pgo);
```

```
    input [1:0] pg;
    input pg0;
    output pgo;

    assign pgo = (pg0 & pg[1]) | pg[0];

endmodule
////////////////////////////////////////////
module xor16 (A, B, S);

    input [15:0] A, B;
    output [15:0] S;

    assign S = A ^ B;

endmodule
////////////////////////////////////////////////////
module pg16 (A, B, pg15, pg14, pg13, pg12, pg11, pg10, pg9,
pg8, pg7, pg6, pg5, pg4, pg3, pg2, pg1, pg0);
    input [15:0] A, B;
    output [1:0] pg15, pg14, pg13, pg12, pg11, pg10, pg9, pg8,
pg7, pg6, pg5, pg4, pg3, pg2, pg1, pg0;

    assign pg15 = {(A[15] ^ B[15]), (A[15] & B[15])};
    assign pg14 = {(A[14] ^ B[14]), (A[14] & B[14])};
    assign pg13 = {(A[13] ^ B[13]), (A[13] & B[13])};
    assign pg12 = {(A[12] ^ B[12]), (A[12] & B[12])};
    assign pg11 = {(A[11] ^ B[11]), (A[11] & B[11])};
    assign pg10 = {(A[10] ^ B[10]), (A[10] & B[10])};
    assign pg9 = {(A[9] ^ B[9]), (A[9] & B[9])};
    assign pg8 = {(A[8] ^ B[8]), (A[8] & B[8])};
    assign pg7 = {(A[7] ^ B[7]), (A[7] & B[7])};
    assign pg6 = {(A[6] ^ B[6]), (A[6] & B[6])};
    assign pg5 = {(A[5] ^ B[5]), (A[5] & B[5])};
    assign pg4 = {(A[4] ^ B[4]), (A[4] & B[4])};
    assign pg3 = {(A[3] ^ B[3]), (A[3] & B[3])};
    assign pg2 = {(A[2] ^ B[2]), (A[2] & B[2])};
    assign pg1 = {(A[1] ^ B[1]), (A[1] & B[1])};
    assign pg0 = {(A[0] ^ B[0]), (A[0] & B[0])};

endmodule
/////////////////////////////////////////////////
module Brent_Kung16 (A, B, Cin, S, Cout);
    input [15:0] A, B;
    input Cin;
    output [15:0] S;
    output Cout;

   // First generate the propigate and generate signals for each
bit
    wire [1:0] r1c16, r1c15, r1c14, r1c13, r1c12, r1c11, r1c10,
r1c9;
    wire [1:0] r1c8, r1c7, r1c6, r1c5, r1c4, r1c3, r1c2, r1c1;

    pg16
ipg16(.A(A), .B(B), .pg15(r1c16),.pg14(r1c15),.pg13(r1c14),
       .pg12(r1c13),.pg11(r1c12),.pg10(r1c11),.pg9(r1c10),.pg
8(r1c9),
       .pg7(r1c8),.pg6(r1c7),.pg5(r1c6),.pg4(r1c5),.pg3(r1c4),
       .pg2(r1c3),.pg1(r1c2),.pg0(r1c1));

    // First row
    wire [1:0] r2c15, r2c13, r2c11, r2c9, r2c7, r2c5, r2c3;
    wire r2c1;

    black ir1c15(.pg(r1c15), .pg0(r1c14), .pgo(r2c15));
    black ir1c13(.pg(r1c13), .pg0(r1c12), .pgo(r2c13));
    black ir1c11(.pg(r1c11), .pg0(r1c10), .pgo(r2c11));
    black ir1c9(.pg(r1c9), .pg0(r1c8), .pgo(r2c9));
    black ir1c7(.pg(r1c7), .pg0(r1c6), .pgo(r2c7));
    black ir1c5(.pg(r1c5), .pg0(r1c4), .pgo(r2c5));
    black ir1c3(.pg(r1c3), .pg0(r1c2), .pgo(r2c3));
    gray ir1c1(.pg(r1c1), .pg0(Cin), .pgo(r2c1));

    // Second row
    wire [1:0] r3c15, r3c11, r3c7;
```

```
wire r3c3;

black ir2c15(.pg(r2c15), .pg0(r2c13), .pgo(r3c15));
black ir2c11(.pg(r2c11), .pg0(r2c9), .pgo(r3c11));
black ir2c7(.pg(r2c7), .pg0(r2c5), .pgo(r3c7));
gray ir2c3(.pg(r2c3), .pg0(r2c1), .pgo(r3c3));

// Third row
wire [1:0] r4c15;
wire r4c7;

black ir3c15(.pg(r3c15), .pg0(r3c11), .pgo(r4c15));
gray ir3c7(.pg(r3c7), .pg0(r3c3), .pgo(r4c7));

// Fourth row
wire r5c15, r5c11;

gray ir4c15(.pg(r4c15), .pg0(r4c7), .pgo(r5c15));
gray ir6c11(.pg(r3c11), .pg0(r4c7), .pgo(r5c11));

// Fifth row
wire r6c13, r6c9, r6c5;

gray ir5c13(.pg(r2c13), .pg0(r5c11), .pgo(r6c13));
gray ir5c9(.pg(r2c9), .pg0(r4c7), .pgo(r6c9));
gray ir5c5(.pg(r2c5), .pg0(r3c3), .pgo(r6c5));

// Sixth row
wire r7c14, r7c12, r7c10, r7c8, r7c6, r7c4, r7c2;

gray ir6c14(.pg(r1c14), .pg0(r6c13), .pgo(r7c14));
gray ir6c12(.pg(r1c12), .pg0(r5c11), .pgo(r7c12));
gray ir6c10(.pg(r1c10), .pg0(r6c9), .pgo(r7c10));
gray ir6c8(.pg(r1c8), .pg0(r4c7), .pgo(r7c8));
gray ir6c6(.pg(r1c6), .pg0(r6c5), .pgo(r7c6));
gray ir6c4(.pg(r1c4), .pg0(r3c3), .pgo(r7c4));
gray ir6c2(.pg(r1c2), .pg0(r2c1), .pgo(r7c2));

  // Finaly produce the sum
  xor16
ixor16(.A({r5c15,r7c14,r6c13,r7c12,r5c11,r7c10,r6c9,r7c8,r4c7,r7c6,

r6c5,r7c4,r3c3,r7c2,r2c1,Cin}),  .B({r1c16[1],r1c15[1],r1c14[1],

r1c13[1],r1c12[1],r1c11[1],r1c10[1],r1c9[1],r1c8[1],r1c7[1],r1c6[1],
    r1c5[1],r1c4[1],r1c3[1],r1c2[1],r1c1[1]}), .S(S));

  // Generate Cout
  gray gcout(.pg(r1c16), .pg0(r5c15), .pgo(Cout));

endmodule
//////////////////////////////////////////////////////////////////////////////
module brent_test;

    reg [15:0] a,b;     // numbers to add
    reg cin;    // carry in
    wire[15:0] s;       //sum
    wire cout;    //cout

    Brent_Kung16 bk16(a, b, cin, s, cout);

    initial
      begin
        a=16'b1010101010101010;
        b=16'b0101010101010101;
        cin=1'b1;

        #10 b=16'b1010101010101010;
        #20 a=16'b0101010101010101;
        #30 cin=1'b0;
      end
endmodule[8]
```

Carry-Select adder

```
module fulladder(s,cout,a,b,cin);
  input a,b,cin;
  output s,cout;

  assign s = ((a ^ b) ^ cin);
  assign cout = ((a & b)| (cin & (a ^ b)));

endmodule
//////////////////////////////////////////////////////
 module adder4(s, cout, a, b, cin);
    input [3:0] a;
    input [3:0] b;
    input cin;
    output [3:0] s;
    output cout;

   fulladder fulladder0(s[0],cout0, a[0], b[0], cin);
   fulladder fulladder1(s[1],cout1, a[1], b[1], cout0);
   fulladder fulladder2(s[2],cout2, a[2], b[2], cout1);
   fulladder fulladder3(s[3],cout, a[3], b[3], cout2);

 endmodule
//////////////////////////////////////////////////////////
 module csa (s, cout, a, b,  cin, c0, c1 );

            input [15:0] a;
            input [15:0] b;
            input cin,c0,c1;

            output [15:0] s;
            output cout ;

            wire  [8:0] x0 ;
            wire  [8:0] x1;
            wire  [4:0] w1;
            wire  [4:0] w2;
            wire  [3:0] w3;
            wire  [3:0] w4;
            wire  [4:0] w7;
            wire  [4:0] w8;
            wire  [4:0] w9;
            wire  [4:0] w10;
            wire  [4:0] w11;
            wire  [8:0] w12;
            wire  cout1 ;
            wire  w5 ;
            wire  w6 ;


  adder4   a1(s[3:0], cout1, a[3:0], b[3:0], cin);
  adder4   a2(w1[3:0], w1[4], a[7:4], b[7:4], c0);
  adder4   a3(w2[3:0], w2[4], a[7:4], b[7:4], c1);
  adder4   a4(w3[3:0], w5, a[11:8], b[11:8], c0);
  adder4   a5(w4[3:0], w6, a[11:8], b[11:8], c1);
  adder4   a6(w7[3:0], w7[4], a[15:12], b[15:12], c0);
  adder4   a7(w8[3:0], w8[4], a[15:12], b[15:12], c1);

assign  w9[4:0]=   cout1   ?  {w2[4:0]} : (~ cout1)  ?
{w1[4:0]} :  5'b00000;
assign  w10[4:0]=   w6   ?  {w8[4:0]} : (~ w6)  ?  {w7[4:0]} :
5'b00000;
assign  w11[4:0]=   w5   ?  {w8[4:0]} : (~ w5)  ?  {w7[4:0]} :
5'b00000;

  assign x0 [8:4] = w10 [4:0] ;
  assign x0 [3:0] = w3 [3:0] ;

  assign x1[8:4] = w11 [4:0] ;
  assign x1[3:0] = w4 [3:0] ;


 assign  w12[8:0]=   w9[4]   ?  {big_in_1} : (~ w9[4])  ?  {x0} :
9'b0_0000_0000;

  assign s[7:4] = w9[3:0];
  assign s[15:8] = w12[7:0];
```

```
assign cout = w12[8];

endmodule

 module test_csa;

        reg [15:0] a;
        reg [15:0] b;
        reg cin,c0,c1;
        wire [15:0] s;
        wire cout;


        csa csa1 (s, cout, a, b, cin, c0,c1);

        initial
        begin
          a=16'b1010101010101010;
     b=16'b0101010101010101;
     cin=1'b1;


          c0 = 1'b0;
          c1 = 1'b1;

          #10 b=16'b1010101010101010;
     #20 a=16'b0101010101010101;
     #30 cin=1'b0;

        end

    endmodule[9]
```

Carry-Skip adder

```
module padder(carry,sum,po,a,b,c);

  output carry;
  output sum;
  output po;

  input a;
  input b;
  input c;

  assign sum=a^b^c;
  assign po=a|b;
  assign carry=a&b|c&po;

endmodule
////////////////////////////////////////////////
module csblock(cout,Sum,A,B,cin);

  output cout;
  output [3:0] Sum;

  input  [3:0] A;
  input  [3:0] B;
  input  cin;

  wire   [3:0] P,C;

padder a0 (C[0],Sum[0],P[0],A[0],B[0],cin);
padder a1 (C[1],Sum[1],P[1],A[1],B[1],C[0]);
padder a2 (C[2],Sum[2],P[2],A[2],B[2],C[1]);
padder a3 (C[3],Sum[3],P[3],A[3],B[3],C[2]);

assign cout=C[3]|(cin&P[0]&P[1]&P[2]&P[3]);
endmodule
///////////////////////////////////////////////////////////
module csa(cout,Sum,A,B,cin);

  output cout;
  output [15:0] Sum;
  input  [15:0] A;
  input  [15:0] B;
```

```
  input  cin;

  wire   [3:0] carries;

csblock b0 (carries[0],Sum[3:0],A[3:0],B[3:0],cin);
csblock b1 (carries[1],Sum[7:4],A[7:4],B[7:4],carries[0]);
csblock b2 (carries[2],Sum[11:8],A[11:8],B[11:8],carries[1]);
csblock b3 (cout,Sum[15:12],A[15:12],B[15:12],carries[2]);

endmodule
//////////////////////////////////////////////////////////////
 module csa_test;

         reg [15:0] A;
        reg [15:0] B;
        reg cin;
        wire [15:0] Sum;
        wire cout;

       csa csa1 (cout,Sum,A,B,cin);

         initial
          begin
                a=16'b1010101010101010;
           b=16'b0101010101010101;
           cin=1'b1;


                    c0 = 1'b0;
                    c1 = 1'b1;
                    #10 b=16'b1010101010101010;
                 #20 a=16'b0101010101010101;
                 #30 cin=1'b0;
                   end
          endmodule[10]
```

References

[1] B. Gilchrist et al., “Fast Carry Logic for Digital Computers,” IRE Trans. EC-4, pp. 133-136, 1955.

[2] B. Gilchrist, J. H. Pomerene and S. Y. Wong, “Fast Carry Logic for Digital Computers”, IRE Trans. Elec. Comp., Vol. EC-4, no 4, pp. 133-136, 1955.

[3] J. F. Kruy, “A Fast Conditional Sum Adder Using Carry Bypass Logic,” AFIPS Con5 Proceedings, Vol. 27, FJCC, pp. 695-703, 1965

[4] M. Lehman, and N. Burla, “Skip Techniques for High-speed Carry-Propagation in Binary Arithmetic Units,” IRE Trans. EC-10, No. 4, pp. 691-698, 1961.

[5] H. Ling, “High-speed Binary Parallel Adder,” IEEE Trans. Comp., EC-15, No.5, pp. 799-802. 1966.

[6] Mi Lu John ,“Arithmetic and logic in computer systems” Wiley & Sons, Ap - Computers - 246 Pages Pp 61-63, 2005

[7] Sarika A. Parate, Prof- R. N. Mandavgane , “Review of delay and power efficient Carry-Select adder using pipeline”, International Engineering Journal For Research & Development, Volume 2 Issue 1, pp. 16-22, 2012.

[8] Chan, P.K. Schlag, M.D.F.; Thomborson, C.D.; Oklobdzija, V.G.,” Delay optimization of carry-skip adders and block carry-lookahead adders” Computer Arithmetic, 1991. Proceedings., 10th IEEE Symposium, pp 154 - 164 Grenoble 1991

[9] EE 5324–VLSI Design II Kia Bazargan University of Minnesota Part II: Adders Pp 68, retrieved from http://www.ece.umn.edu/users/kia/Courses/EE5324/index.html adders. pdf, 2014.

[10] High Speed Adder, retrieved from : http://l3.elfak.ni.ac.rs/viewvc/int_sab_two_level_cl/doc/background/HighSpeedAdder.pdf?revision=1.1, 2014.

[11] Brent-Kung Adder, retrieved from :http://en. academic. ru /dic.nsf/enwiki/1292234, 2014.