**Digital Signal Design Lab**

Lab CEL-442

Lab Journal: 7



Name: Omar

Class: BCE-06

Enrollment No: 01-132182-024

**Lab # 7**

**Carry Select Adder (CSA)**

**Objective** This lab is aimed at exploring the fast adder’s domain by designing a Carry Select Adder.

**Introduction**

**Carry Select Adder**

* Carry Select Adder is used in many data processing processors to perform fast arithmetic operations.
* Generally, it consists of Ripple Carry Adder and a multiplexer.
* It uses multiple narrow adders to create fast wide adders.
* n-bit carry select adder comprises different blocks with either uniform or variable block size.
* The main idea is to pre-compute the sum and carry for each block with carry\_in=0 as well as carry\_in=1. therefore, two adders are required for each block except first 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 carry-select adder generally consists of two ripple carry adders and a mux. 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 being zero and the other assuming one.

After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known. The number of bits in each carry select block can be uniform, or variable.

**Task 1:**

`timescale 1ns / 1ps

module CSA\_4\_bit(s,cout,a,b,cin);

output s,cout;

input a,b,cin;

assign s=(a^b)^cin;

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

endmodule

module mux(y,d0,d1,s);

output y;

input d0,d1,s;

assign y=s?d1:d0;

endmodule

module csa(s,cout,a,b,cin);

output[4:0]s;

output cout;

input [4:0] a,b;

input cin;

wire[3:0] c0,c1,s0,s1;

wire cin1;

CSA\_4\_bit L0(s[0],cin1,a[0],b[0],cin);

CSA\_4\_bit L00(s0[0],c0[0],a[1],b[1],0);

CSA\_4\_bit L01(s0[1],c0[1],a[2],b[2],c0[0]);

CSA\_4\_bit L02(s0[2],c0[2],a[3],b[3],c0[1]);

CSA\_4\_bit L03(s0[3],c0[3],a[4],b[4],c0[2]);

CSA\_4\_bit L10(s1[0],c1[0],a[1],b[1],1);

CSA\_4\_bit L11(s1[1],c1[1],a[2],b[2],c1[0]);

CSA\_4\_bit L12(s1[2],c1[2],a[3],b[3],c1[1]);

CSA\_4\_bit L13(s1[3],c1[3],a[4],b[4],c1[2]);

mux m0(s[1],s0[0],s1[0],cin1);

mux m1(s[2],s0[1],s1[1],cin1);

mux m2(s[3],s0[2],s1[2],cin1);

mux m3(s[4],s0[3],s1[3],cin1);

mux m4(cout,c0[3],c1[3],cin1);

endmodule

**Testbench**

`timescale 1ns / 1ps

module csa\_test;

// Inputs

reg [4:0] a;

reg [4:0] b;

reg cin;

// Outputs

wire [4:0] s;

wire cout;

// Instantiate the Unit Under Test (UUT)

csa uut (

.s(s),

.cout(cout),

.a(a),

.b(b),

.cin(cin)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

cin = 0;

// Wait 100 ns for global reset to finish

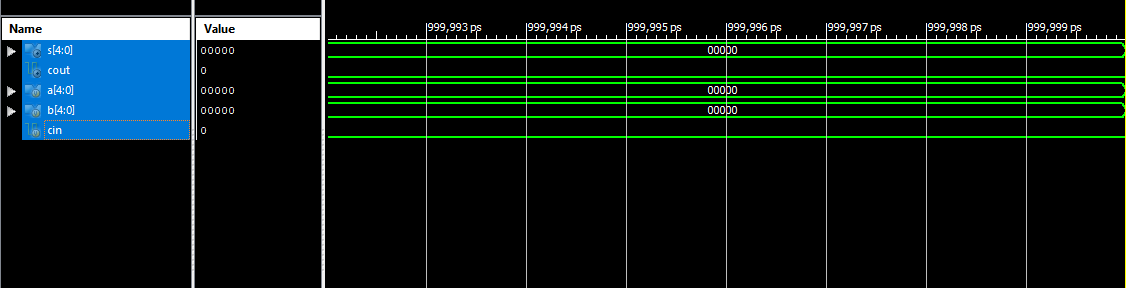
#100;

// Add stimulus here

end

endmodule

**OUTPUT:**

****

**Task 2**

`timescale 1ns / 1ps

module CSA\_4\_bit(s,cout,a,b,cin);

output s,cout;

input a,b,cin;

assign s=(a^b)^cin;

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

endmodule

module mux(y,d0,d1,s);

output y;

input d0,d1,s;

assign y=s?d1:d0;

endmodule

module csa(s,cout,a,b,cin);

output[8:0]s;

output cout;

input [8:0] a,b;

input cin;

wire[7:0] c0,c1,s0,s1;

wire cin1;

CSA\_4\_bit L0(s[0],cin1,a[0],b[0],cin);

CSA\_4\_bit L00(s0[0],c0[0],a[1],b[1],0);

CSA\_4\_bit L01(s0[1],c0[1],a[2],b[2],c0[0]);

CSA\_4\_bit L02(s0[2],c0[2],a[3],b[3],c0[1]);

CSA\_4\_bit L03(s0[3],c0[3],a[4],b[4],c0[2]);

CSA\_4\_bit L04(s0[4],c0[4],a[5],b[5],c0[3]);

CSA\_4\_bit L05(s0[5],c0[5],a[6],b[6],c0[4]);

CSA\_4\_bit L06(s0[6],c0[6],a[7],b[7],c0[5]);

CSA\_4\_bit L07(s0[7],c0[7],a[8],b[8],c0[6]);

CSA\_4\_bit L10(s1[0],c1[0],a[1],b[1],1);

CSA\_4\_bit L11(s1[1],c1[1],a[2],b[2],c1[0]);

CSA\_4\_bit L12(s1[2],c1[2],a[3],b[3],c1[1]);

CSA\_4\_bit L13(s1[3],c1[3],a[4],b[4],c1[2]);

CSA\_4\_bit L14(s1[4],c1[4],a[5],b[5],c1[3]);

CSA\_4\_bit L15(s1[5],c1[5],a[6],b[6],c1[4]);

CSA\_4\_bit L16(s1[6],c1[6],a[7],b[7],c1[5]);

CSA\_4\_bit L17(s1[7],c1[7],a[8],b[8],c1[6]);

mux m0(s[1],s0[0],s1[0],cin1);

mux m1(s[2],s0[1],s1[1],cin1);

mux m2(s[3],s0[2],s1[2],cin1);

mux m3(s[4],s0[3],s1[3],cin1);

mux m5(s[5],s0[4],s1[4],cin1);

mux m6(s[6],s0[5],s1[5],cin1);

mux m7(s[7],s0[6],s1[6],cin1);

mux m8(s[8],s0[7],s1[7],cin1);

mux m4(cout,c0[7],c1[7],cin1);

endmodule

**Test bench**

`timescale 1ns / 1ps

module CAS\_8bit\_test;

// Inputs

reg [8:0] a;

reg [8:0] b;

reg cin;

// Outputs

wire [8:0] s;

wire cout;

// Instantiate the Unit Under Test (UUT)

csa uut (

.s(s),

.cout(cout),

.a(a),

.b(b),

.cin(cin)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

cin = 0;

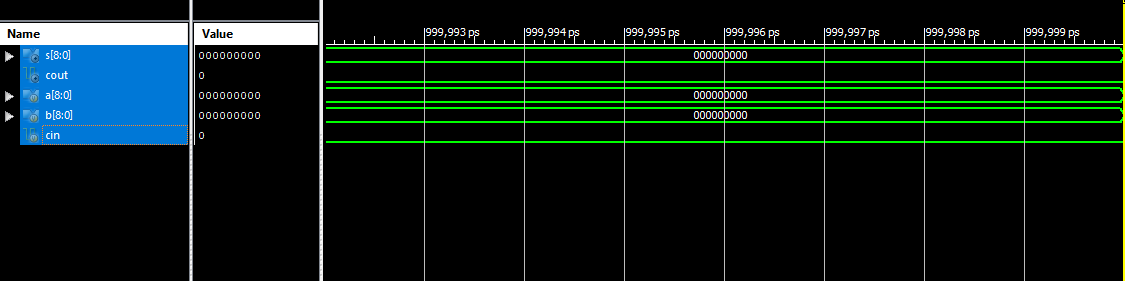
// Wait 100 ns for global reset to finish

#100; // Add stimulus here

end

endmodule

**OUTPUT**



**Task 3**

`timescale 1ns / 1ps

module CSA\_4\_bit(s,cout,a,b,cin);

output s,cout;

input a,b,cin;

assign s=(a^b)^cin;

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

endmodule

module mux(y,d0,d1,s);

output y;

input d0,d1,s;

assign y=s?d1:d0;

endmodule

module csa(s,cout,a,b,cin);

output[15:0]s;

output cout;

input [15:0] a,b;

input cin;

wire[14:0] c0,c1,s0,s1;

wire cin1;

CSA\_4\_bit L0(s[0],cin1,a[0],b[0],cin);

CSA\_4\_bit L00(s0[0],c0[0],a[1],b[1],0);

CSA\_4\_bit L01(s0[1],c0[1],a[2],b[2],c0[0]);

CSA\_4\_bit L02(s0[2],c0[2],a[3],b[3],c0[1]);

CSA\_4\_bit L03(s0[3],c0[3],a[4],b[4],c0[2]);

CSA\_4\_bit L04(s0[4],c0[4],a[5],b[5],c0[3]);

CSA\_4\_bit L05(s0[5],c0[5],a[6],b[6],c0[4]);

CSA\_4\_bit L06(s0[6],c0[6],a[7],b[7],c0[5]);

CSA\_4\_bit L07(s0[7],c0[7],a[8],b[8],c0[6]);

CSA\_4\_bit L08(s0[8],c0[8],a[9],b[9],c0[7]);

CSA\_4\_bit L09(s0[9],c0[9],a[10],b[10],c0[8]);

CSA\_4\_bit L010(s0[10],c0[10],a[11],b[11],c0[9]);

CSA\_4\_bit L011(s0[11],c0[11],a[12],b[12],c0[10]);

CSA\_4\_bit L012(s0[12],c0[12],a[13],b[13],c0[11]);

CSA\_4\_bit L013(s0[13],c0[13],a[14],b[14],c0[12]);

CSA\_4\_bit L014(s0[14],c0[14],a[15],b[15],c0[13]);

CSA\_4\_bit L10(s1[0],c1[0],a[1],b[1],1);

CSA\_4\_bit L11(s1[1],c1[1],a[2],b[2],c1[0]);

CSA\_4\_bit L12(s1[2],c1[2],a[3],b[3],c1[1]);

CSA\_4\_bit L13(s1[3],c1[3],a[4],b[4],c1[2]);

CSA\_4\_bit L14(s1[4],c1[4],a[5],b[5],c1[3]);

CSA\_4\_bit L15(s1[5],c1[5],a[6],b[6],c1[4]);

CSA\_4\_bit L16(s1[6],c1[6],a[7],b[7],c1[5]);

CSA\_4\_bit L17(s1[7],c1[7],a[8],b[8],c1[6]);

CSA\_4\_bit L18(s1[8],c1[8],a[9],b[9],c1[7]);

CSA\_4\_bit L19(s1[9],c1[9],a[10],b[10],c1[8]);

CSA\_4\_bit L20(s1[10],c1[10],a[11],b[11],c1[9]);

CSA\_4\_bit L21(s1[11],c1[11],a[12],b[12],c1[10]);

CSA\_4\_bit L22(s1[12],c1[12],a[13],b[13],c1[11]);

CSA\_4\_bit L23(s1[13],c1[13],a[14],b[14],c1[12]);

CSA\_4\_bit L24(s1[14],c1[14],a[15],b[15],c1[13]);

mux m0(s[1],s0[0],s1[0],cin1);

mux m1(s[2],s0[1],s1[1],cin1);

mux m2(s[3],s0[2],s1[2],cin1);

mux m3(s[4],s0[3],s1[3],cin1);

mux m5(s[5],s0[4],s1[4],cin1);

mux m6(s[6],s0[5],s1[5],cin1);

mux m7(s[7],s0[6],s1[6],cin1);

mux m8(s[8],s0[7],s1[7],cin1);

mux m9(s[9],s0[8],s1[8],cin1);

mux m10(s[10],s0[9],s1[9],cin1);

mux m11(s[11],s0[10],s1[10],cin1);

mux m12(s[12],s0[11],s1[11],cin1);

mux m13(s[13],s0[12],s1[12],cin1);

mux m14(s[14],s0[13],s1[13],cin1);

mux m15(s[15],s0[14],s1[14],cin1);

mux m4(cout,c0[14],c1[14],cin1);

endmodule

**OUTPUT :**

`timescale 1ns / 1ps

module CSA\_16bit\_test;

// Inputs

reg [15:0] a;

reg [15:0] b;

reg cin;

// Outputs

wire [15:0] s;

wire cout;

// Instantiate the Unit Under Test (UUT)

csa uut (

.s(s),

.cout(cout),

.a(a),

.b(b),

.cin(cin)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

cin = 0;

// Wait 100 ns for global reset to finish

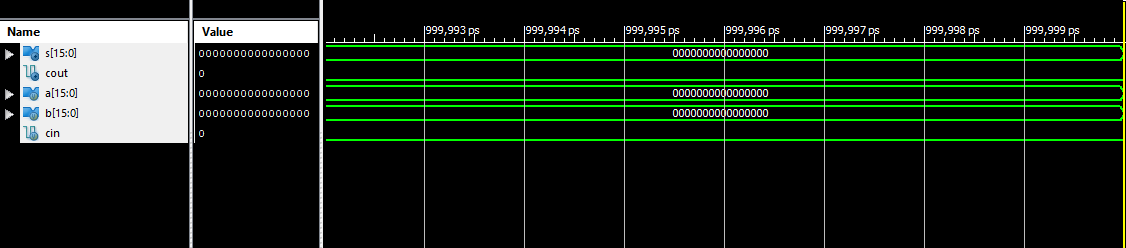
#100;

// Add stimulus here

end

endmodule

**OUTPUT**



**Conclusion: -**

In this lab we learned about exploring the fast adder’s domain by designing a Carry Select Adder.