



Instituto Politécnico Nacional  
Escuela Superior de Cómputo



## Arquitectura de Computadoras

### **“Sumador”**

Alumno:

Malagón Baeza Alan Adrian

Profesor:

Alemán Arce Miguel Ángel

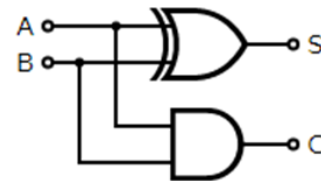
Grupo: 5CV1

# Introducción

El medio sumador, sumador completo, ripple carry adder y carry lookahead son circuitos lógicos utilizados en el diseño de sistemas digitales para realizar operaciones de suma en binario.

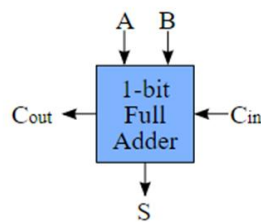
El medio sumador es un circuito lógico que puede sumar dos bits y generar un bit de resultado y un bit de acarreo.

Half Adder Truth Table			
A	B	Carry	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



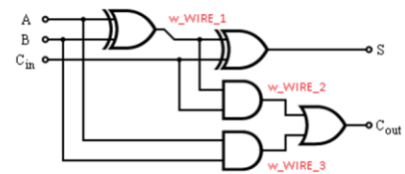
Half-Adder Schematic - From Wikipedia

El sumador completo es un circuito que puede sumar dos bits y un bit de acarreo previo para generar un bit de resultado y un bit de acarreo posterior.



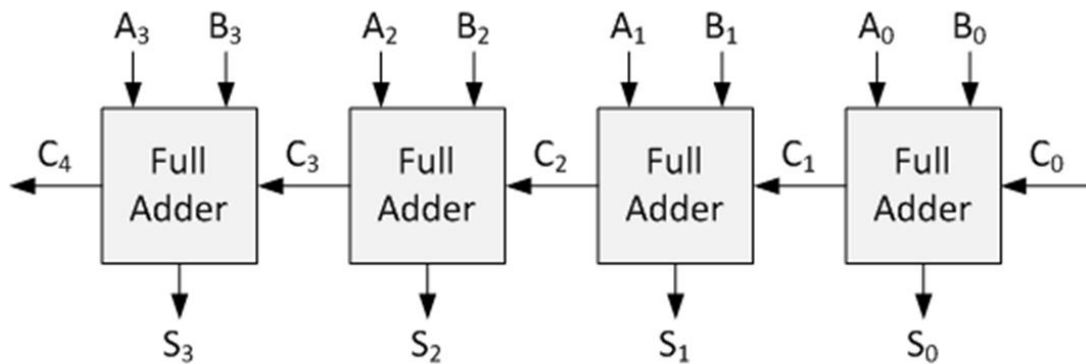
1-bit Full-Adder Block - From Wikipedia

Full Adder Truth Table				
A	B	C <sub>in</sub>	C <sub>out</sub>	Sum
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



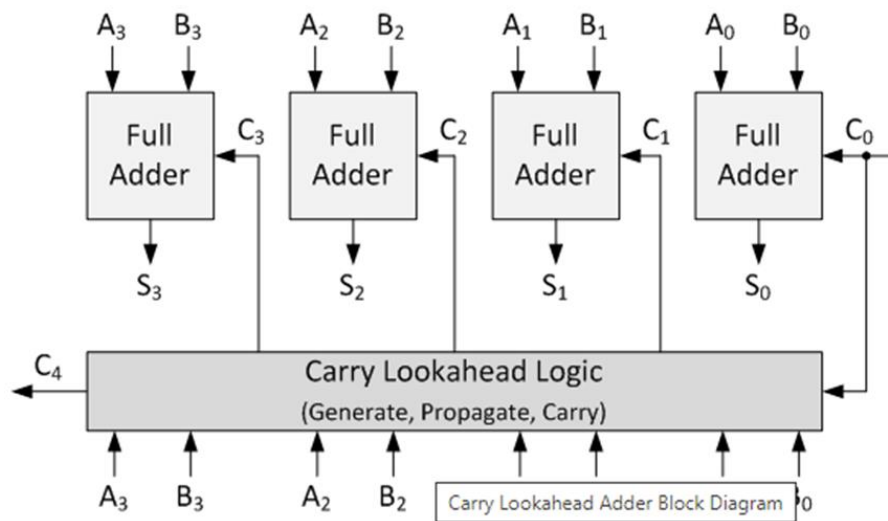
1-bit Full-Adder Detailed Schematic - Modified From Wikipedia

El ripple carry adder es un circuito que utiliza sumadores completos en cascada para sumar números de varios bits.



**Ripple Carry Adder (4-bit) Block Diagram**

El carry lookahead es un circuito que mejora el tiempo de propagación del acarreo en el ripple carry adder.



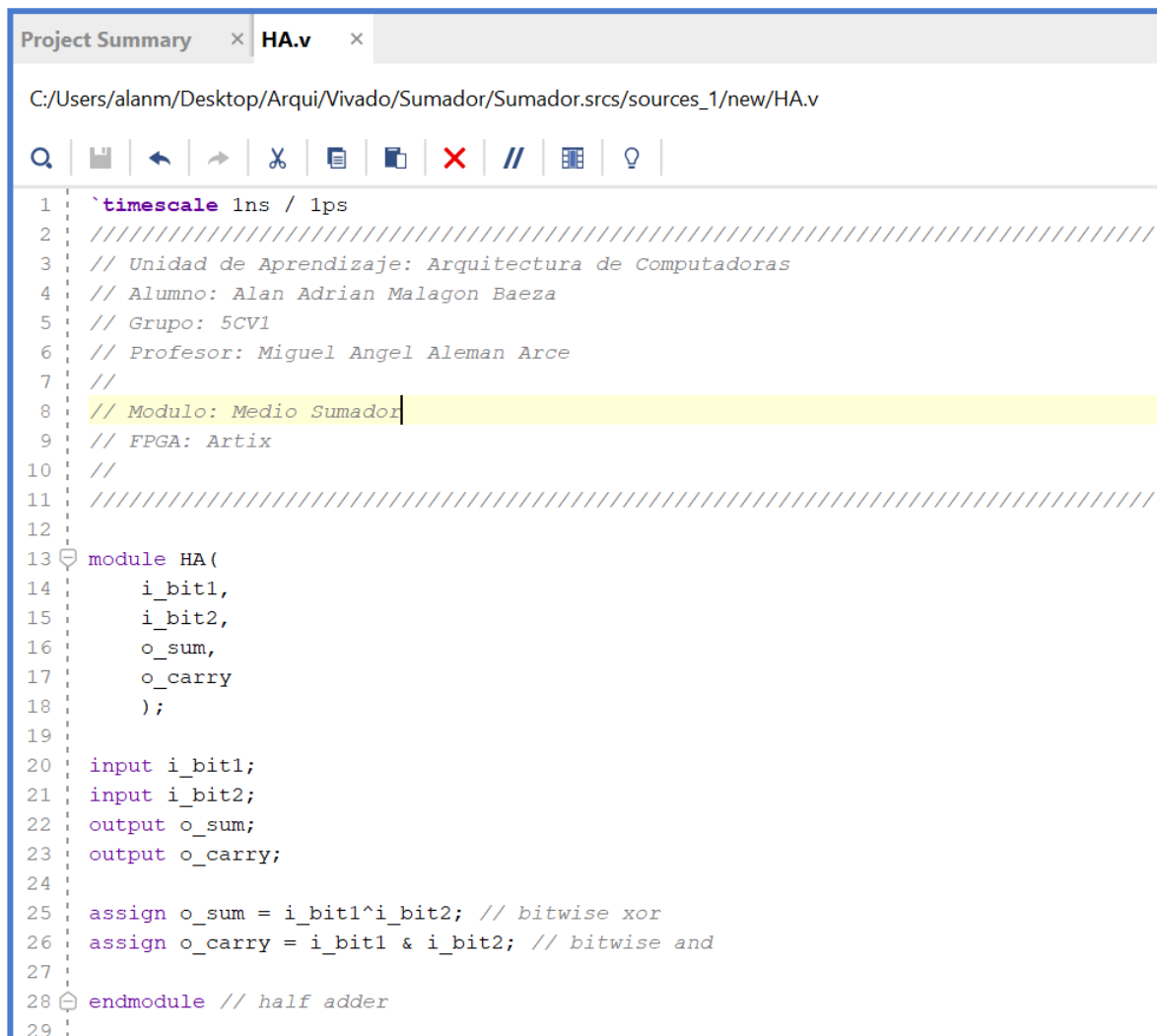
**Carry Lookahead Adder 4-bit Block Diagram**

En Verilog, se puede implementar cada uno de estos circuitos mediante la descripción de hardware. El código Verilog para el medio sumador puede ser simple y directo, mientras que el código para el sumador completo, ripple carry adder y carry lookahead puede ser más complejo debido a la necesidad de conectar varios bloques de hardware. Sin embargo, Verilog proporciona una sintaxis clara y estructurada para describir estos circuitos y permite una simulación y verificación exhaustiva antes de la implementación física en hardware.

# Desarrollo

## Medio Sumador

### Código Verilog



The image shows a screenshot of a Verilog code editor. The top bar has tabs for 'Project Summary' and 'HA.v'. The file path is 'C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srscs/sources\_1/new/HA.v'. The code is a Verilog module for a half adder, named 'HA'. It includes a timescale of 1ns / 1ps and several comments in Spanish. The module has two inputs, 'i\_bit1' and 'i\_bit2', and two outputs, 'o\_sum' and 'o\_carry'. The logic is implemented using bitwise XOR and AND operations. The code is as follows:

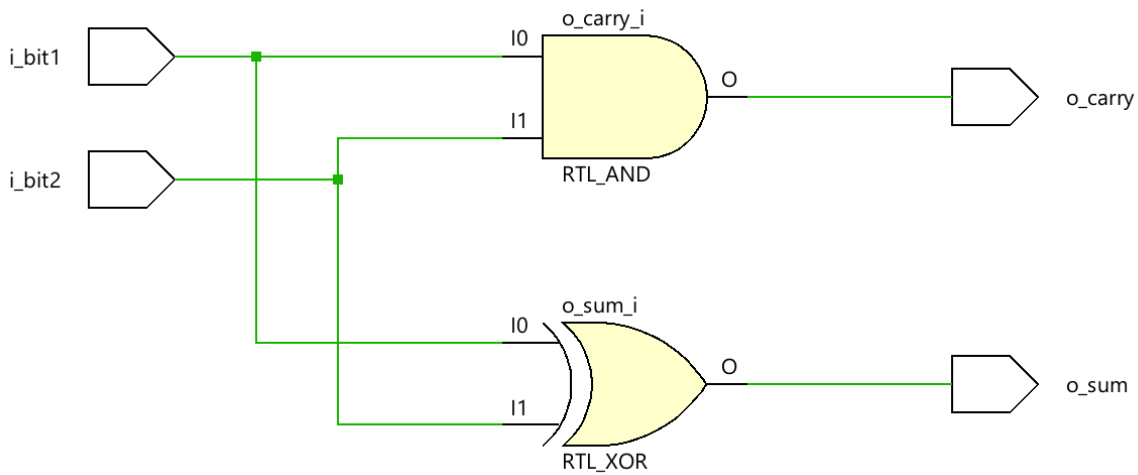
```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: Medio Sumador
9  // FPGA: Artix
10 //
11 ///////////////////////////////////////////////////////////////////
12
13 module HA(
14     i_bit1,
15     i_bit2,
16     o_sum,
17     o_carry
18 );
19
20 input i_bit1;
21 input i_bit2;
22 output o_sum;
23 output o_carry;
24
25 assign o_sum = i_bit1^i_bit2; // bitwise xor
26 assign o_carry = i_bit1 & i_bit2; // bitwise and
27
28 endmodule // half adder
29
```

## Código Verilog Testbench

```
HA.v x HA_tb_behav.wcfg* x HA_tb.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srscs/sim_1/new/HA_tb.v

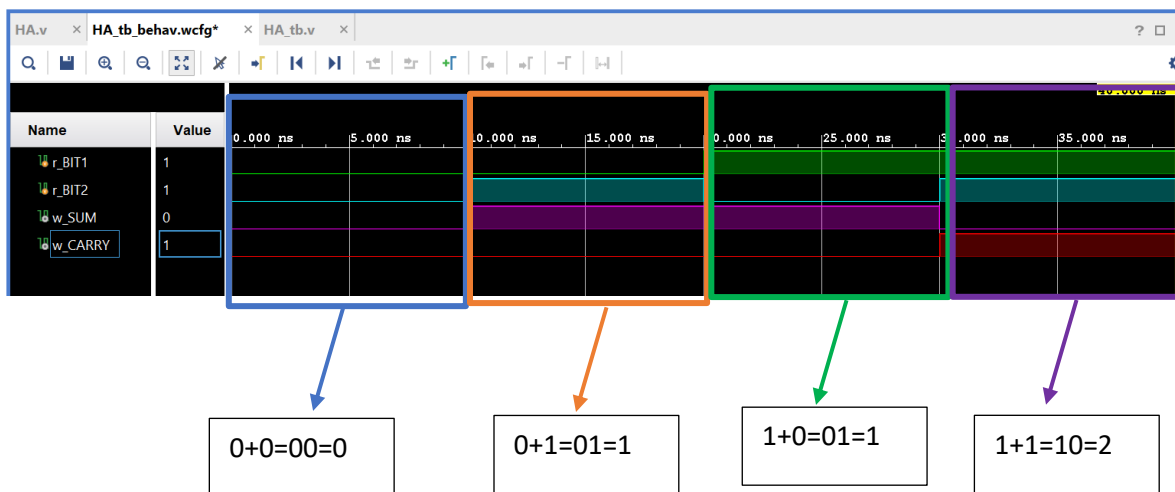
1 | `timescale 1ns / 1ps
2 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 5CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Modulo: Medio Sumador
9 | // FPGA: Artix
10 | //
11 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12 |
13 | module HA_tb;
14 |     reg r_BIT1 = 0;
15 |     reg r_BIT2 = 0;
16 |     wire w_SUM;
17 |     wire w_CARRY;
18 |
19 |     HA DUT (.i_bit1(r_BIT1), .i_bit2(r_BIT2), .o_sum(w_SUM), .o_carry(w_CARRY) );
20 |
21 |     initial
22 |     begin
23 |         r_BIT1 = 1'b0;
24 |         r_BIT2 = 1'b0;
25 |         #10;
26 |         r_BIT1 = 1'b0;
27 |         r_BIT2 = 1'b1;
28 |         #10;
29 |         r_BIT1 = 1'b1;
30 |         r_BIT2 = 1'b0;
31 |         #10;
32 |         r_BIT1 = 1'b1;
33 |         r_BIT2 = 1'b1;
34 |         #10;
35 |         $finish;
36 |     end
37 | endmodule // HA_tb
```

## Implementación RTL en Vivado 2022.2



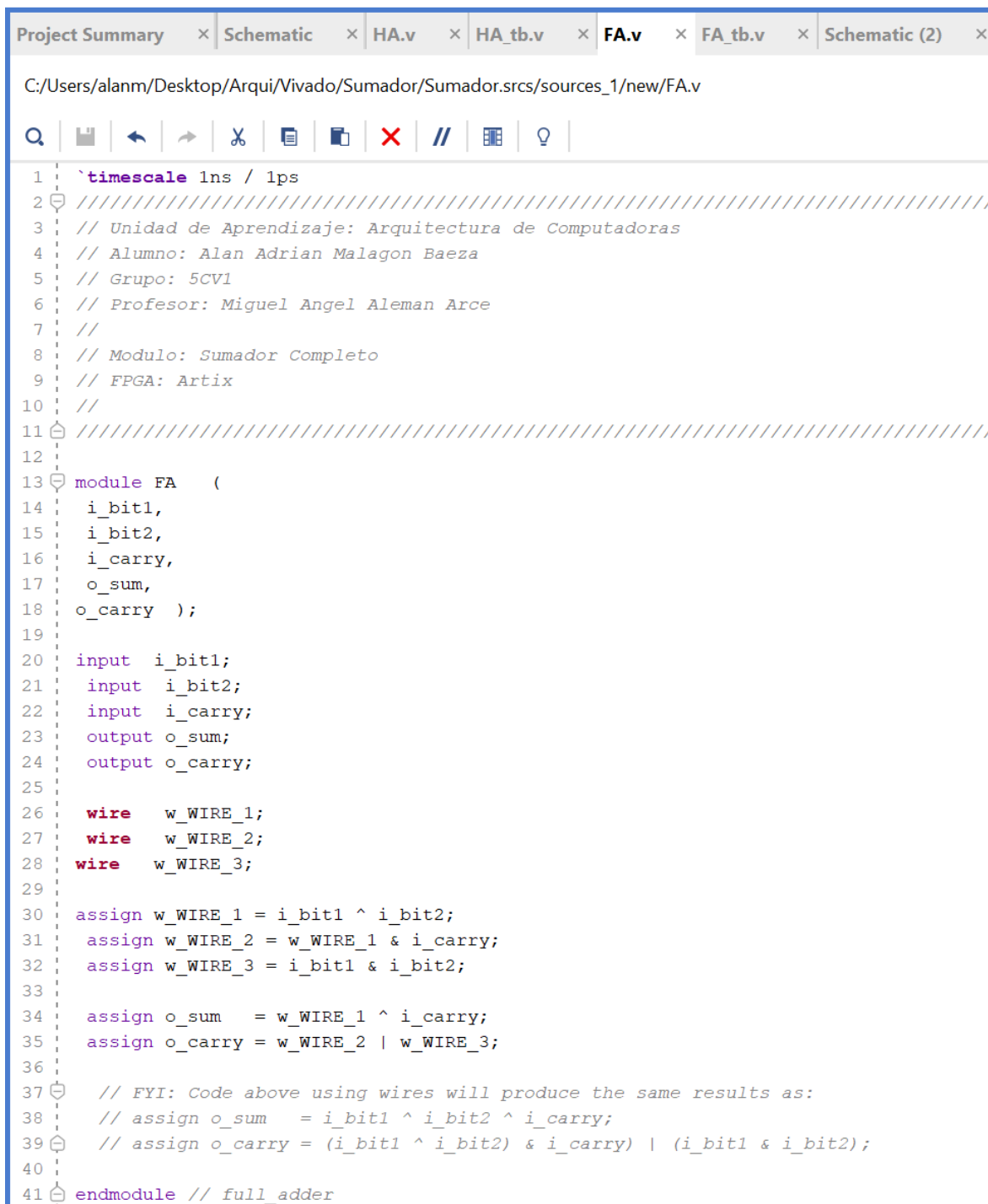
Podemos observar cómo utilizó una compuerta AND y una XOR para la implementación del medio sumador.

### Resultado de la simulación:



## Sumador Completo

### Código Verilog



The image shows a screenshot of a Verilog code editor. The top of the window has a tab bar with several tabs: 'Project Summary', 'Schematic', 'HA.v', 'HA\_tb.v', 'FA.v' (which is the active tab), 'FA\_tb.v', and 'Schematic (2)'. Below the tab bar, the file path is displayed: 'C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srscs/sources\_1/new/FA.v'. A toolbar with various icons for editing and viewing is located below the path. The main area of the editor contains Verilog code for a full adder module named 'FA'. The code includes comments in Spanish, input/output declarations, wire declarations, and logic assignments. Line numbers 1 through 41 are visible on the left side of the code.

```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: Sumador Completo
9  // FPGA: Artix
10 //
11 ///////////////////////////////////////////////////////////////////
12
13 module FA (
14     i_bit1,
15     i_bit2,
16     i_carry,
17     o_sum,
18     o_carry );
19
20 input  i_bit1;
21 input  i_bit2;
22 input  i_carry;
23 output o_sum;
24 output o_carry;
25
26 wire   w_WIRE_1;
27 wire   w_WIRE_2;
28 wire   w_WIRE_3;
29
30 assign w_WIRE_1 = i_bit1 ^ i_bit2;
31 assign w_WIRE_2 = w_WIRE_1 & i_carry;
32 assign w_WIRE_3 = i_bit1 & i_bit2;
33
34 assign o_sum    = w_WIRE_1 ^ i_carry;
35 assign o_carry  = w_WIRE_2 | w_WIRE_3;
36
37 // FYI: Code above using wires will produce the same results as:
38 // assign o_sum    = i_bit1 ^ i_bit2 ^ i_carry;
39 // assign o_carry  = (i_bit1 ^ i_bit2) & i_carry | (i_bit1 & i_bit2);
40
41 endmodule // full_adder
```

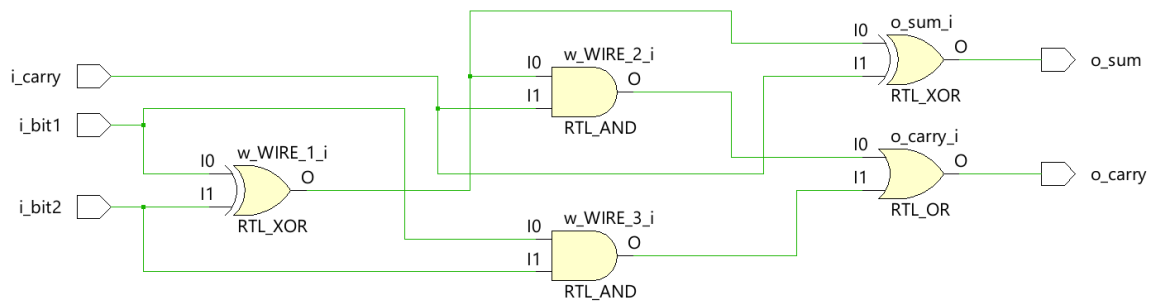
## Código Verilog Testbench

```
Project Summary x FA_tb.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srscs/sim_1/new/FA_tb.v

1 `timescale 1ns / 1ps
2 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 // Unidad de Aprendizaje: Arquitectura de Computadoras
4 // Alumno: Alan Adrian Malagon Baeza
5 // Grupo: 5CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Modulo: Sumador Completo
9 // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 module FA_tb;
14     reg r_BIT1 = 0;
15     reg r_BIT2 = 0;
16     reg i_CARRY = 0;
17     wire w_SUM;
18     wire w_CARRY;
19
20     FA DUT (.i_bit1(r_BIT1), .i_bit2(r_BIT2), .i_carry(i_CARRY), .o_sum(w_SUM), .o_carry(w_CARRY) );
21
22     initial
23     begin
24         r_BIT1 = 1'b0;
25         r_BIT2 = 1'b0;
26         i_CARRY = 1'b0;
27         #10;
28         r_BIT1 = 1'b0;
29         r_BIT2 = 1'b1;
30         i_CARRY = 1'b1;
31         #10;
32         r_BIT1 = 1'b1;
33         r_BIT2 = 1'b0;
34         i_CARRY = 1'b0;
35         #10;
36         r_BIT1 = 1'b1;
37         r_BIT2 = 1'b1;
38         i_CARRY = 1'b1;
39         #10;
40         $finish;
41     end
42 endmodule // FA_tb
```

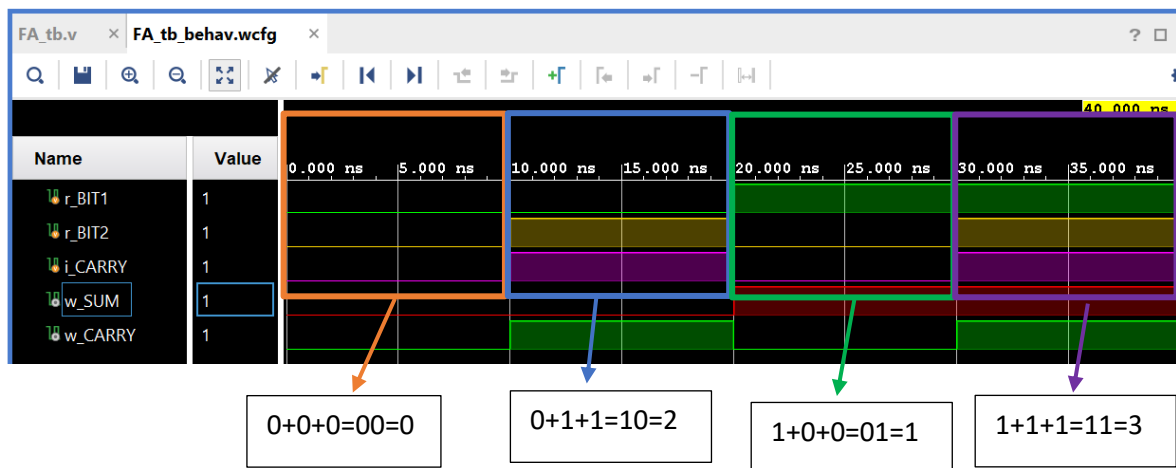


## Implementación RTL en Vivado 2022.2



Podemos observar cómo utilizó compuertas AND, XOR y una compuerta OR para la implementación del medio sumador.

### Resultado de la simulación:



## Ripple Carry Adder

### Código Verilog

```
FA_tb.v x ripple_carry_adder_tb.v x ripple_carry_adder.v x FA.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srscs/sources_1/new/ripple_carry_adder.v

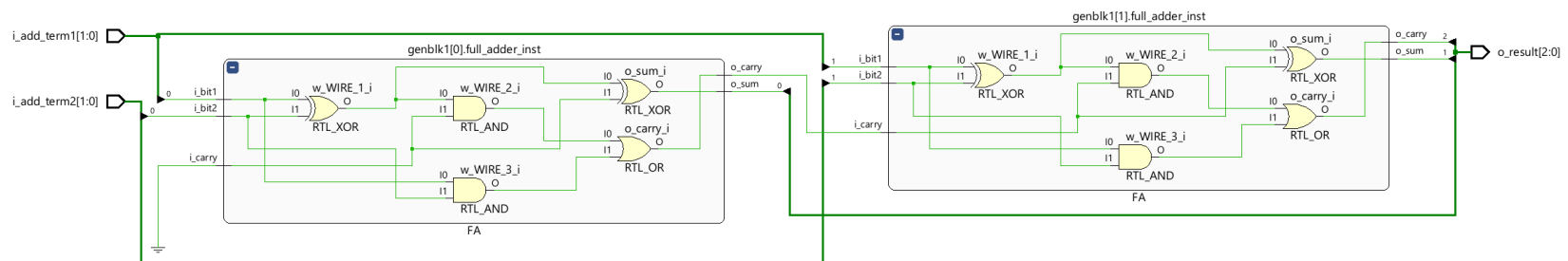
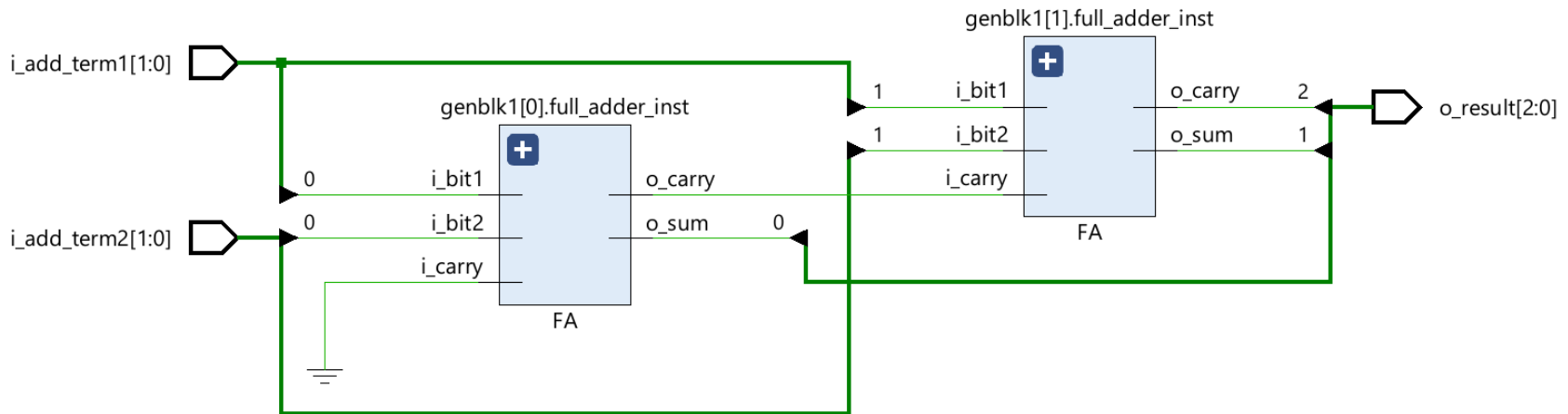
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: Ripple Carry Adder
9  // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 module ripple_carry_adder
14     #(parameter WIDTH=2)
15     (
16         input [WIDTH-1:0] i_add_term1,
17         input [WIDTH-1:0] i_add_term2,
18         output [WIDTH:0] o_result
19     );
20
21     wire [WIDTH:0] w_CARRY;
22     wire [WIDTH-1:0] w_SUM;
23
24     // No carry input on first full adder
25     assign w_CARRY[0] = 1'b0;
26
27     genvar ii;
28     generate
29         for (ii=0; ii<WIDTH; ii=ii+1)
30             begin
31                 FA full_adder_inst
32                 (
33                     .i_bit1(i_add_term1[ii]),
34                     .i_bit2(i_add_term2[ii]),
35                     .i_carry(w_CARRY[ii]),
36                     .o_sum(w_SUM[ii]),
37                     .o_carry(w_CARRY[ii+1])
38                 );
39             end
40     endgenerate
41
42     assign o_result = {w_CARRY[WIDTH], w_SUM}; // Verilog Concatenation
43
44 endmodule // ripple_carry_adder
```

## Código Verilog Testbench

```
FA_tb.v x ripple_carry_adder_tb.v x ripple_carry_adder.v x FA.v x Untitled 1 x
C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srscs/sim_1/new/ripple_carry_adder_tb.v

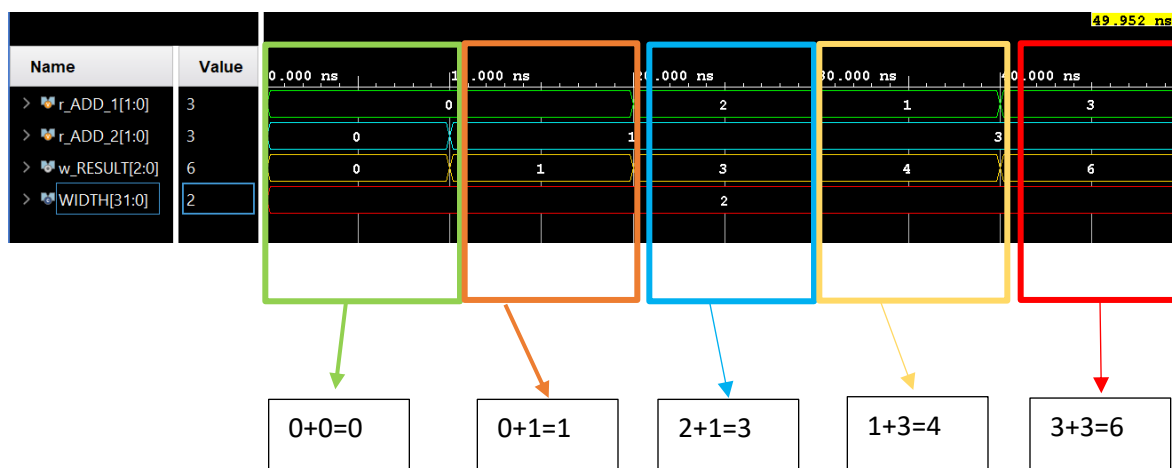
1 | `timescale 1ns / 1ps
2 | //////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 5CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Modulo: Ripple Carry Adder
9 | // FPGA: Artix
10 | //
11 | //////////////////////////////////////////////////
12 |
13 | module ripple_carry_adder_tb ();
14 |     parameter WIDTH = 2;
15 |     reg [WIDTH-1:0] r_ADD_1 = 0;
16 |     reg [WIDTH-1:0] r_ADD_2 = 0;
17 |     wire [WIDTH:0] w_RESULT;
18 |
19 |     ripple_carry_adder #(.WIDTH(WIDTH)) ripple_carry_inst
20 |     (
21 |         .i_add_term1(r_ADD_1),
22 |         .i_add_term2(r_ADD_2),
23 |         .o_result(w_RESULT)
24 |     );
25 |
26 |     initial
27 |     begin
28 |         #10;
29 |         r_ADD_1 = 2'b00;
30 |         r_ADD_2 = 2'b01;
31 |         #10;
32 |         r_ADD_1 = 2'b10;
33 |         r_ADD_2 = 2'b01;
34 |         #10;
35 |         r_ADD_1 = 2'b01;
36 |         r_ADD_2 = 2'b11;
37 |         #10;
38 |         r_ADD_1 = 2'b11;
39 |         r_ADD_2 = 2'b11;
40 |         #10;
41 |         $finish;
42 |     end
43 |
44 | endmodule // ripple_carry_adder_tb
..
```

## Implementación RTL en Vivado 2022.2



Podemos observar cómo utilizó módulos del sumador completo para la implementación del ripple carry adder.

## Resultado de la simulación:



# Carry Lookahead Adder 4 Bit

## Código Verilog

```
Project Summary x Schematic x carry_lookahead_adder_4_bit.v x carry_lookahead_adder_4_bit_tb.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srscs/sources_1/new/carry_lookahead_adder_4_bit.v

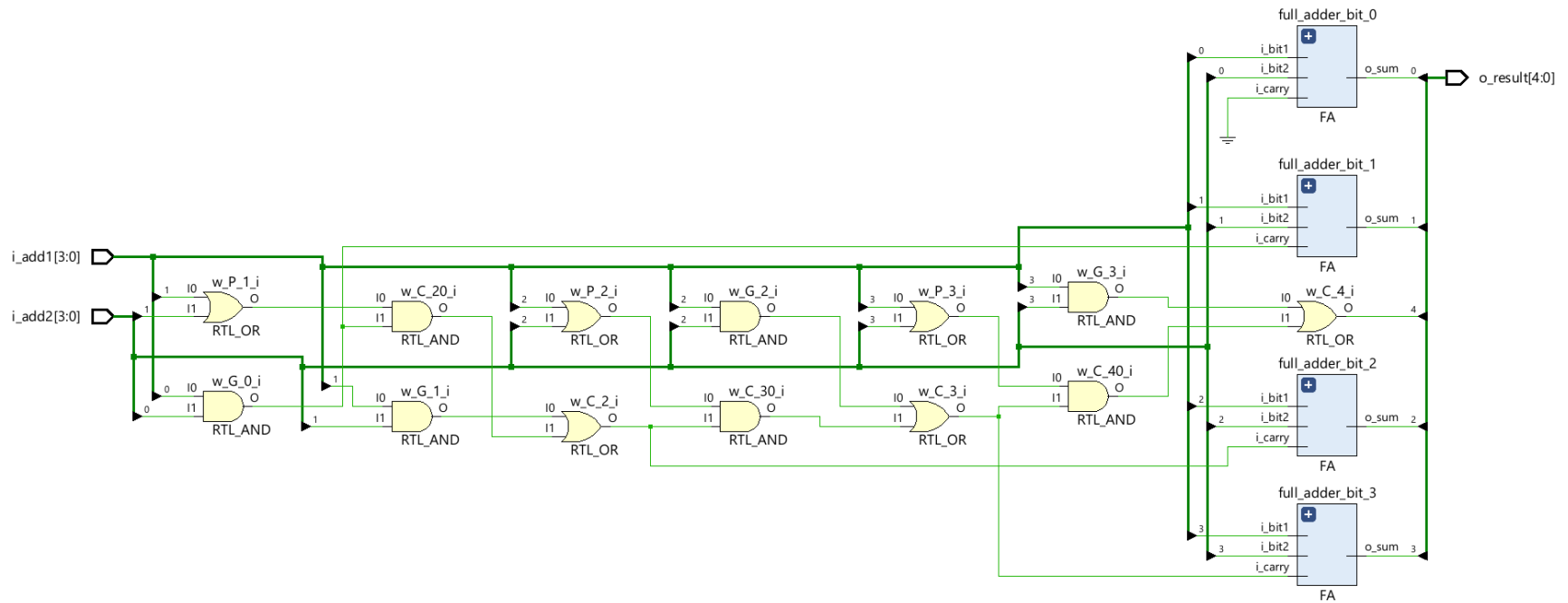
1  `timescale 1ns / 1ps
2  //////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: Carry Lookahead 4 Bit
9  // FPGA: Artix
10 //
11 //////////////////////////////////////
12
13 module carry_lookahead_adder_4_bit (
14     input [3:0] i_add1,
15     input [3:0] i_add2,
16     output [4:0] o_result );
17
18     wire [4:0] w_C;
19     wire [3:0] w_G, w_P, w_SUM;
20
21     FA full_adder_bit_0 ( .i_bit1(i_add1[0]), .i_bit2(i_add2[0]), .i_carry(w_C[0]), .o_sum(w_SUM[0]), .o_carry());
22
23     FA full_adder_bit_1( .i_bit1(i_add1[1]), .i_bit2(i_add2[1]),
24         .i_carry(w_C[1]), .o_sum(w_SUM[1]), .o_carry());
25
26     FA full_adder_bit_2( .i_bit1(i_add1[2]), .i_bit2(i_add2[2]),
27         .i_carry(w_C[2]), .o_sum(w_SUM[2]), .o_carry());
28
29     FA full_adder_bit_3( .i_bit1(i_add1[3]),
30         .i_bit2(i_add2[3]), .i_carry(w_C[3]),
31         .o_sum(w_SUM[3]), .o_carry());
32 // Create the Generate (G) Terms: Gi=Ai*Bi
33 assign w_G[0] = i_add1[0] & i_add2[0];
34 assign w_G[1] = i_add1[1] & i_add2[1];
35 assign w_G[2] = i_add1[2] & i_add2[2];
36 assign w_G[3] = i_add1[3] & i_add2[3];
37
38 // Create the Propagate Terms: Pi=Ai+Bi
39 assign w_P[0] = i_add1[0] | i_add2[0];
40 assign w_P[1] = i_add1[1] | i_add2[1];
41 assign w_P[2] = i_add1[2] | i_add2[2];
42 assign w_P[3] = i_add1[3] | i_add2[3];
43
44 // Create the Carry Terms:
45 assign w_C[0] = 1'b0; // no carry input
46 assign w_C[1] = w_G[0] | (w_P[0] & w_C[0]);
47 assign w_C[2] = w_G[1] | (w_P[1] & w_C[1]);
48 assign w_C[3] = w_G[2] | (w_P[2] & w_C[2]);
49 assign w_C[4] = w_G[3] | (w_P[3] & w_C[3]);
50
51 assign o_result = {w_C[4], w_SUM}; // Verilog Concatenation
52
53 endmodule // carry_lookahead_adder_4_bit
54
```

## Código Verilog Testbench

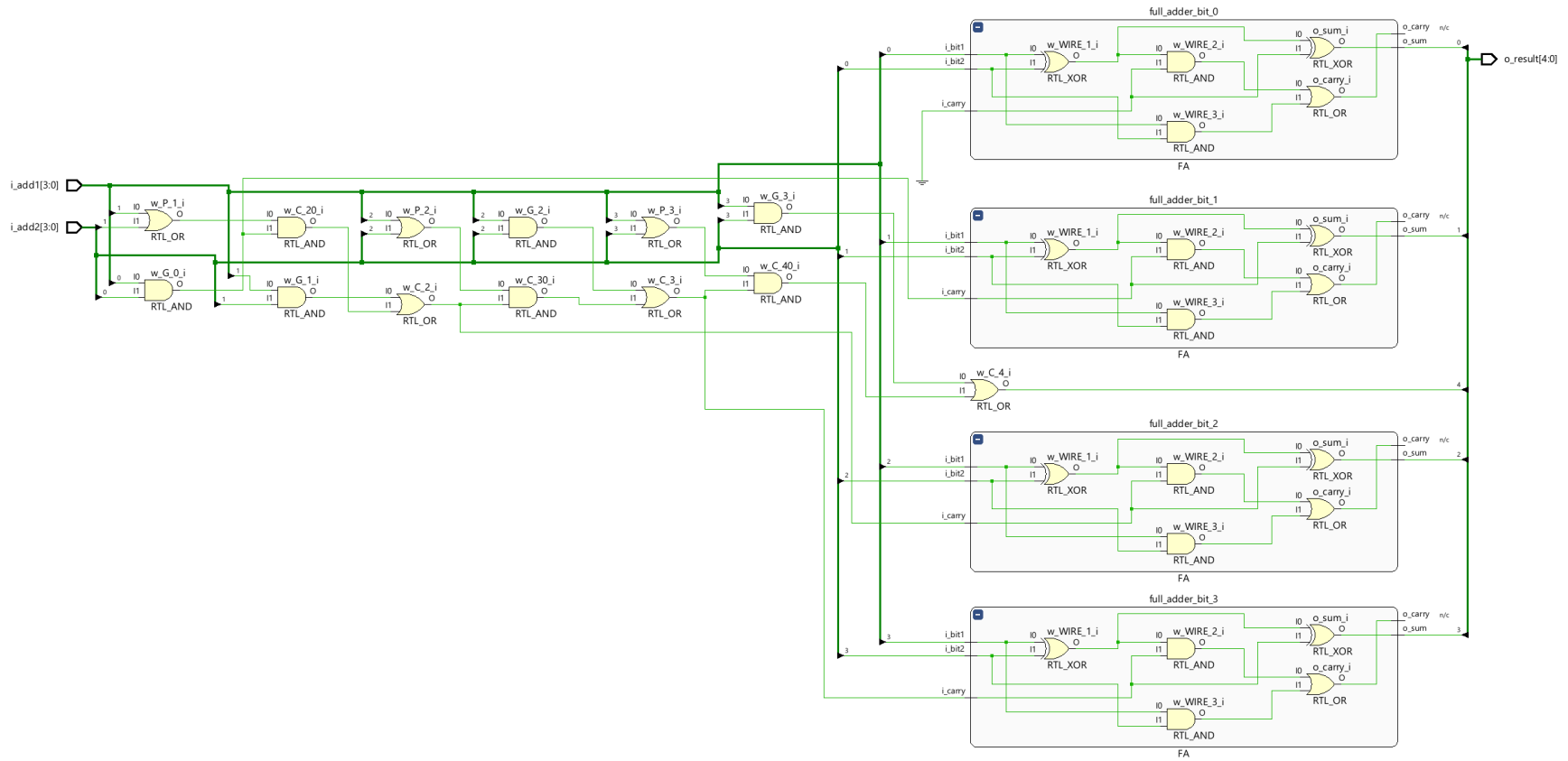
```
Project Summary x Schematic x carry_lookahead_adder_4_bit.v x carry_lookahead_adder_4_bit_tb.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srcs/sim_1/new/carry_lookahead_adder_4_bit_tb.v

1 | `timescale 1ns / 1ps
2 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 5CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Modulo: Carry Lookahead 4 Bit
9 | // FPGA: Artix
10 | //
11 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12 |
13 | module carry_lookahead_adder_4_bit_tb ();
14 |     reg [3:0] r_ADD_1 = 0;
15 |     reg [3:0] r_ADD_2 = 0;
16 |     wire [4:0] w_RESULT;
17 |
18 |     carry_lookahead_adder_4_bit carry_lookahead_inst(
19 |         .i_add1(r_ADD_1), .i_add2(r_ADD_2), .o_result(w_RESULT));
20 |
21 |     initial
22 |     begin
23 |         #10;
24 |         r_ADD_1 = 3'b000;
25 |         r_ADD_2 = 3'b001;
26 |         #10;
27 |         r_ADD_1 = 3'b010;
28 |         r_ADD_2 = 3'b010;
29 |         #10;
30 |         r_ADD_1 = 3'b101;
31 |         r_ADD_2 = 3'b110;
32 |         #10;
33 |         r_ADD_1 = 3'b111;
34 |         r_ADD_2 = 3'b111;
35 |         #10;
36 |         $finish;
37 |     end
38 | endmodule // carry_lookahead_adder_4_bit_tb
39 |
```

## Implementación RTL en Vivado 2022.2

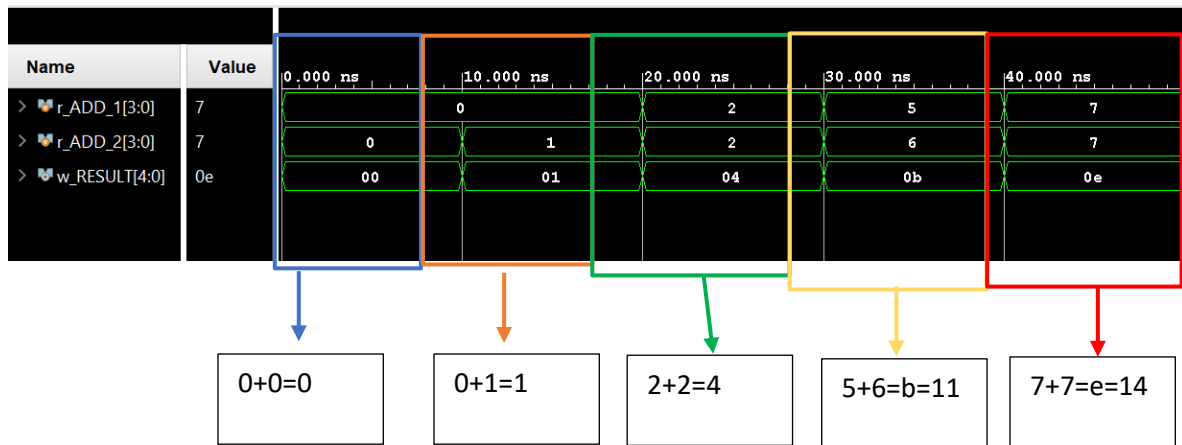






Podemos observar cómo utilizó módulos del sumador completo para la implementación del carry lookahed adder 4 bit.

## Resultado de la simulación:



## Carry Lookahead Adder Parametrizable

### Código Verilog

```
carry_lookahead_adder_tb.v x Schematic x carry_lookahead_adder.v x Schematic (2) x Schematic (3)
C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srscs/sources_1/new/carry_lookahead_adder.v

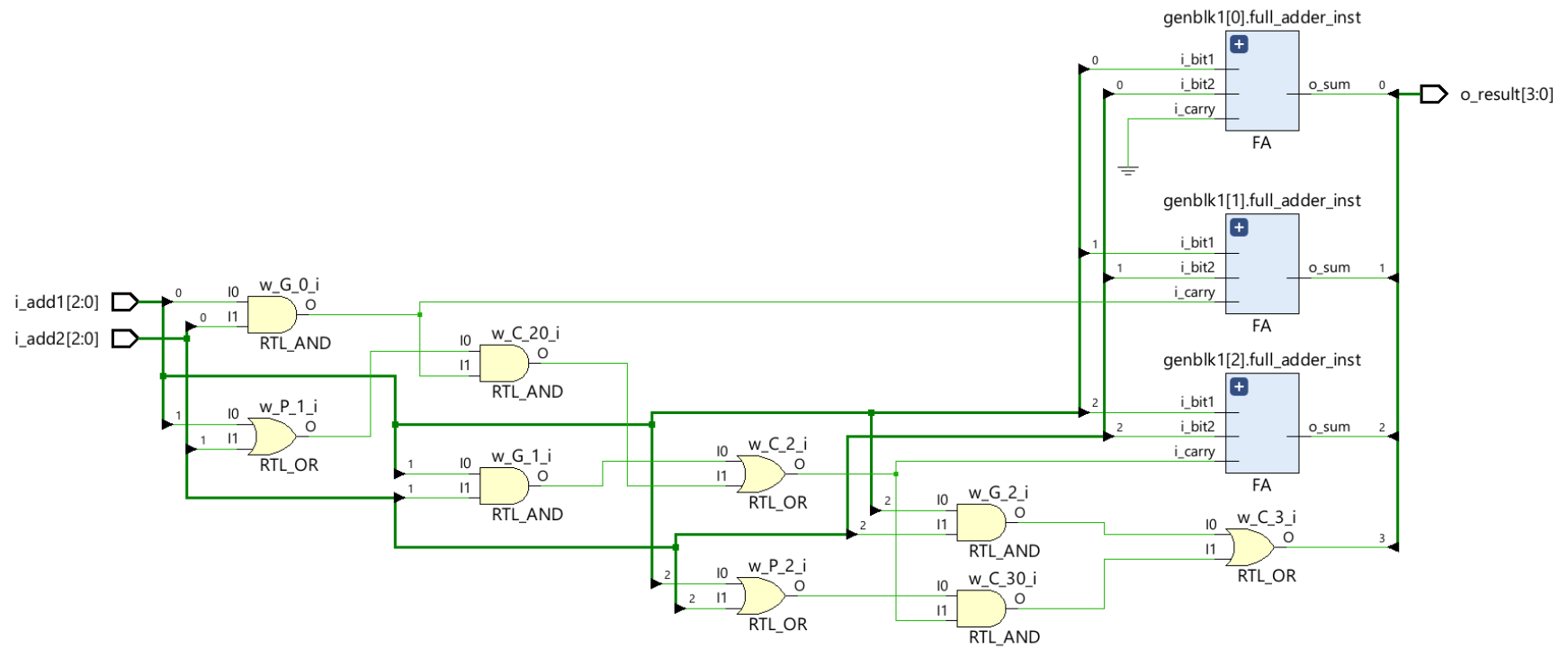
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: Carry Lookahead
9  // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 module carry_lookahead_adder
14     #(parameter WIDTH=3) (
15         input [WIDTH-1:0] i_add1,
16         input [WIDTH-1:0] i_add2,
17         output [WIDTH:0] o_result);
18
19     wire [WIDTH:0] w_C;
20     wire [WIDTH-1:0] w_G, w_P, w_SUM;
21
22     // Create the Full Adders
23     genvar ii;
24     generate
25     for (ii=0; ii<WIDTH; ii=ii+1)
26     begin
27         FA full_adder_inst (.i_bit1(i_add1[ii]), .i_bit2(i_add2[ii]),
28             .i_carry(w_C[ii]), .o_sum(w_SUM[ii]), .o_carry());
29     end
30 endgenerate
31
32 // Create the Generate (G) Terms: Gi=Ai*Bi
33 // Create the Propagate Terms: Pi=Ai+Bi
34 // Create the Carry Terms:
35 genvar jj;
36 generate
37 for (jj=0; jj<WIDTH; jj=jj+1)
38 begin
39     assign w_G[jj] = i_add1[jj] & i_add2[jj];
40     assign w_P[jj] = i_add1[jj] | i_add2[jj];
41     assign w_C[jj+1] = w_G[jj] | (w_P[jj] & w_C[jj]);
42 end
43 endgenerate
44
45 assign w_C[0] = 1'b0; // no carry input on first adder
46 assign o_result = {w_C[WIDTH], w_SUM}; // Verilog Concatenation
47 endmodule // carry_lookahead_adder
```

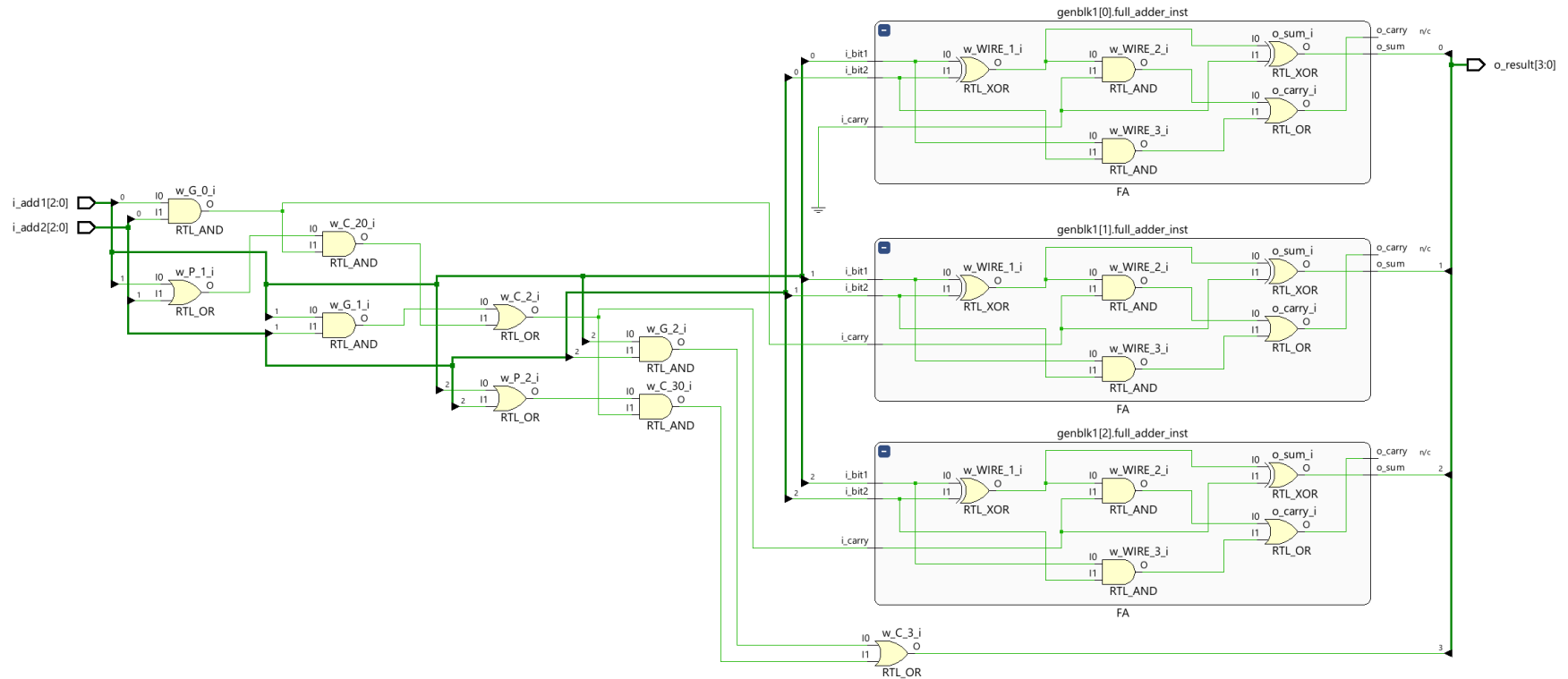
## Código Verilog Testbench

```
carry_lookahead_adder_tb.v x carry_lookahead_adder.v x Untitled 3 x
C:/Users/alanm/Desktop/Arqui/Vivado/Sumador/Sumador.srscs/sim_1/new/carry_lookahead_adder_tb.v

5 // Grupo: 5CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Modulo: Carry Lookahead
9 // FPGA: Artix
10 //
11 //////////////////////////////////////
12
13 module carry_lookahead_adder_tb ();
14     parameter WIDTH = 3;
15     reg [WIDTH-1:0] r_ADD_1 = 0;
16     reg [WIDTH-1:0] r_ADD_2 = 0;
17     wire [WIDTH:0] w_RESULT;
18
19     carry_lookahead_adder #(.WIDTH(WIDTH)) carry_lookahead_inst(
20         .i_add1(r_ADD_1), .i_add2(r_ADD_2), .o_result(w_RESULT));
21
22     initial
23     begin
24         #10;
25         r_ADD_1 = 3'b000;
26         r_ADD_2 = 3'b001;
27         #10;
28         r_ADD_1 = 3'b010;
29         r_ADD_2 = 3'b010;
30         #10;
31         r_ADD_1 = 3'b101;
32         r_ADD_2 = 3'b110;
33         #10;
34         r_ADD_1 = 3'b111;
35         r_ADD_2 = 3'b111;
36         #10;
37         $finish;
38     end
39 endmodule // carry_lookahead_adder_tb
```

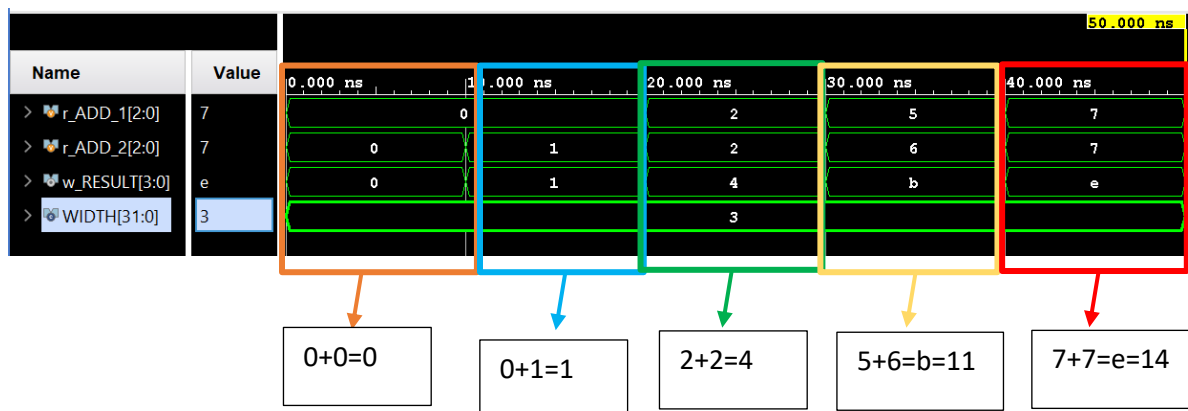
## Implementación RTL en Vivado 2022.2





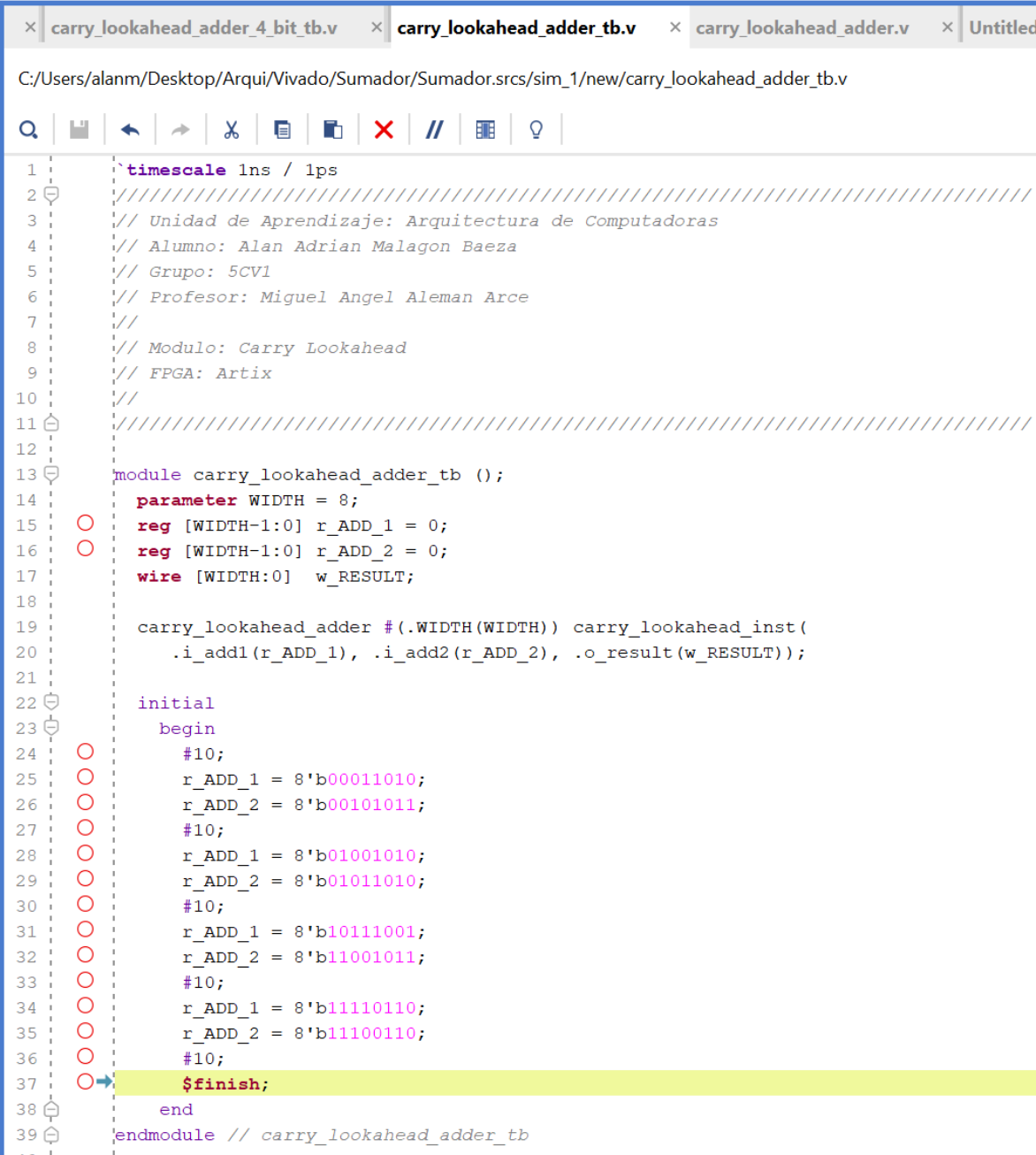
Podemos observar cómo utilizó compuertas AND, OR y módulos del sumador completo para la implementación del carry lookahead parametrizable.

## Resultado de la simulación:



## Carry Lookahead Adder Parametrizable 8 Bits

### Código Verilog Testbench

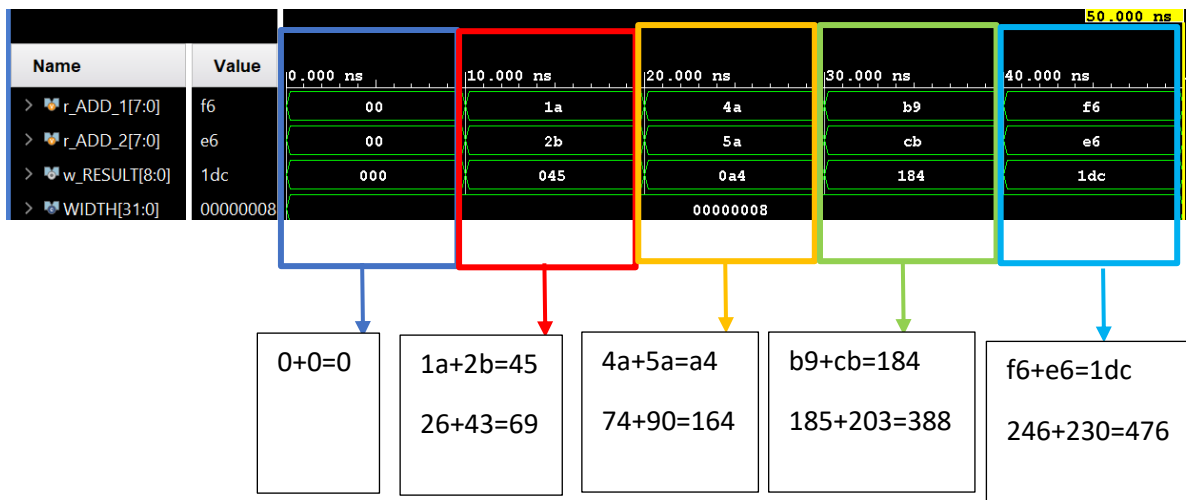


```
1 timescale 1ns / 1ps
2 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 // Unidad de Aprendizaje: Arquitectura de Computadoras
4 // Alumno: Alan Adrian Malagon Baeza
5 // Grupo: 5CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Modulo: Carry Lookahead
9 // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 module carry_lookahead_adder_tb ();
14     parameter WIDTH = 8;
15     reg [WIDTH-1:0] r_ADD_1 = 0;
16     reg [WIDTH-1:0] r_ADD_2 = 0;
17     wire [WIDTH:0] w_RESULT;
18
19     carry_lookahead_adder #(.WIDTH(WIDTH)) carry_lookahead_inst(
20         .i_add1(r_ADD_1), .i_add2(r_ADD_2), .o_result(w_RESULT));
21
22     initial
23     begin
24         #10;
25         r_ADD_1 = 8'b00011010;
26         r_ADD_2 = 8'b00101011;
27         #10;
28         r_ADD_1 = 8'b01001010;
29         r_ADD_2 = 8'b01011010;
30         #10;
31         r_ADD_1 = 8'b10111001;
32         r_ADD_2 = 8'b11001011;
33         #10;
34         r_ADD_1 = 8'b11110110;
35         r_ADD_2 = 8'b11100110;
36         #10;
37         $finish;
38     end
39 endmodule // carry_lookahead_adder_tb
```

Test	r_ADD_1	r_ADD_2
1	00011010	00101011
2	01001010	01011010
3	10111001	11001011
4	11110110	11100110



## Resultado de la simulación:



## Conclusión

En la electrónica digital, los sumadores son circuitos que realizan operaciones de suma de bits. Existen varios tipos de sumadores, entre los cuales se encuentran el medio sumador, sumador completo, ripple carry adder y carry lookahead.

El medio sumador es el circuito más básico de suma de bits y puede sumar dos bits y generar un bit de acarreo (carry). El sumador completo es un circuito más complejo que puede sumar dos bits y el acarreo de la suma anterior, y generar un bit de acarreo para la siguiente suma.

El ripple carry adder es un circuito que suma números binarios de manera secuencial, comenzando desde el bit menos significativo hasta el bit más significativo. A medida que se realiza la suma, el acarreo generado se propaga hacia el bit más significativo. Este tipo de sumador es fácil de implementar pero tiene un tiempo de propagación largo debido a que cada etapa depende del acarreo generado en la etapa anterior.

El carry lookahead es un tipo de sumador que se utiliza para acelerar el proceso de suma. En lugar de propagar el acarreo a través de todas las etapas, el carry lookahead calcula los acarreos de cada etapa de manera simultánea, lo que reduce el tiempo de propagación y mejora la velocidad de la suma.

En conclusión, los sumadores son circuitos esenciales en la electrónica digital y existen varios tipos que se utilizan en diferentes situaciones. El medio sumador es el circuito más básico, el sumador completo es un circuito más complejo, el ripple carry adder es fácil de implementar pero tiene un tiempo de propagación largo, mientras que el carry lookahead se utiliza para acelerar el proceso de suma.

## Referencia

1. Brock J. LaMeres, Introduction to Logic Circuits & Logic Design with Verilog, Springer, 1st Edition, USA, 2017.
2. Harris, D., & Harris, S. (2013). Digital design and computer architecture (2nd ed.). Morgan Kaufmann Publishers.
3. Palnitkar, S. (2003). Verilog HDL: A guide to digital design and synthesis (2nd ed.). Prentice Hall.
4. Weste, N. H. E., & Harris, D. (2011). CMOS VLSI design: A circuits and systems perspective (4th ed.). Addison-Wesley.