Universidade Federal de São Carlos - UFSCar Ciência da Computação

Laboratório de Circuitos Digitais

Experimento 4

Turma - A

Caio Vinicius Barbosa Santos, 726503 Mayk Tulio Bezerra, 727953 Prof. Fredy João Valente

São Carlos 6 de Novembro de 2018

1. Introdução

Uma Unidade Lógica e Aritmética (ULA) é um dispositivo que realiza operações lógicas e aritméticas sobre números representados em circuitos lógicos. Tipicamente, uma ULA recebe dois operandos como entrada, e uma entrada auxiliar de controle que permite especificar qual operação deverá ser realizada. Dessa forma, a ULA realiza as principais operações lógicas e aritméticas de algum dispositivo eletrônico, como soma, subtração, divisão, operações lógicas como AND, OR, XOR, etc. A ULA é uma peça fundamental da UCP (Unidade Central de Processamento), um símbolo esquemático representando-a é exibido a seguir.

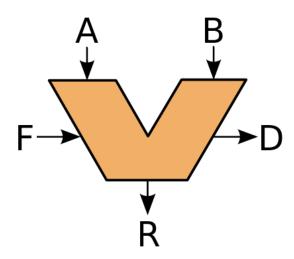


Imagem 0 - Representação da Unidade Lógica Aritmética

Na imagem é possível observar um esquema simplificado representando uma ULA, onde A e B são os operandos, F é a entrada da unidade de controle, D é a saída de status e R a saída gerada pela operação realizada entre A e B.

Neste experimento será realizada a construção de uma ULA utilizando de uma linguagem de descrição de hardware Verilog, todas operações serão simuladas no software Quartus II, e para a primeira etapa com operandos de 4 bits um deploy na placa DE1 da Altera será realizado.

2. Descrição da execução do experimento

2.1 Etapa 1

A primeira etapa deste experimento consiste na construção de uma ULA que contenha as operações especificadas na Figura 1, onde receberá 3 entradas, 2 operandos de 4 bits cada, e uma terceira entrada de 3 bits especificando qual operação será realizada entre eles, o código desenvolvido em Verilog para esta etapa é exibido na Figura 2.

Operation	Mode	Select	Inputs	Output	
	S2	S1	S0	F	
Clear	0	0	0	F = 0 0 0 0	
B - A	0	0	1	F = B - A	
A - B	0	1	0	F = A - B	
ADD	0	1	. 1	F = A + B	
XOR	1	0	0	F = A XOR B	
OR	1	0	1	F = A OR B	
AND	1	1	0	F = A • B	
Preset	1	1	1	F = 1111	

Figura 1. Modelo da ULA de 3 Bits de operações a ser implementado

```
module Exp_4(SW1, SW2, KEY, out, HEX0);
         input [3:0]SW1;
         input [3:0]SW2;
         input [2:0]KEY;
        output reg [3:0]out;
        output reg [0:6]HEX0;
        always@(*)
         case(KEY)
            3'b000:
                    out = 4'b0000;
            3'b001: out = SW2 - SW1;
            3'b010: out = SW1 - SW2;
            3'b011: out = SW1 + SW2;
            3'b100: out = SW1 ^ SW2;
             3'b101: out = SW1 | SW2;
             3'b110: out = SW1 & SW2;
            3'b111: out = 4'b1111;
         endcase
        always@(*)
        case(out)
            4'b0000:
                        HEX0 = 7'b0000001;
            4'b0001:
                        HEX0 = 7'b1001111;
                        HEX0 = 7'b0010010;
            4'b0010:
            4'b0011:
                        HEX0 = 7'b0000110;
            4'b0100:
                        HEX0 = 7'b1001100;
            4'b0101:
                        HEX0 = 7'b0100100;
                        HEX0 = 7'b0100000;
            4'b0110:
             4'b0111:
                        HEX0 = 7'b0001101;
                        HEX0 = 7'b0000000:
            4'b1000:
            4'b1001:
                        HEX0 = 7'b0000100;
         endcase
     endmodule
35
```

Figura 2. Código compilado para a ULA de 4 Bits

Após a construção do algoritmo, o mesmo foi compilado (Figura 3) e uma simulação para cada operação foi realizada como forma de verificar se o código estava efetuando o que se era esperado. Ao confirmar-se através das simulações que a ULA estava correta, um deploy foi realizado na placa DE1 da Altera, os resultados da simulação e da exibição na placa são mostrados na Seção 3 deste relatório.

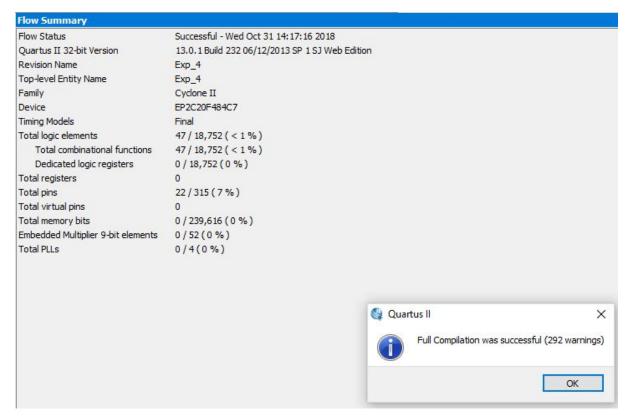


Figura 3. Compilação do código da ULA de 4 Bits

2.2 Etapa 2

A segunda etapa deste experimento consiste na construção de uma ULA de 32 bits, neste caso os dois operandos de entradas possuem 32 bits e a saída gerada era baseado na tabela de instruções (Figura 4).

Opcode	Operation	Function		
000XX	ALU_OUT <= A	Pass A		
001XX	ALU_OUT <= A + B	Add		
010XX	ALU_OUT <= A-B	Subtract		
011XX	ALU_OUT <= A AND B	Logical AND		
100XX	ALU_OUT <= A OR B	Logical OR		
101XX	ALU_OUT <= A + 1	Increment A		
110XX	ALU_OUT <= A-1	Decrement A		
111XX	ALU_OUT <= B	Pass B		
XXX00	Y <= ALU_OUT	Pass ALU_OUT		
XXX01	Y<= SHL(ALU_OUT)	Shift Left		
XXX10	Y<=SHR(ALU_OUT)	Shift Right (unsigned-zero fill)		
XXX11	Y<= 0	Pass 0's		

Figura 4. ULA 3Bits + 2Bits de operações a ser implementada

Assim como na primeira etapa, foi construído um código em Verilog representando as operações contidas na Figura 4, o código é exibido a seguir:

```
module Exp 4 32b(SW1, SW2, KEY, out);
 2
 3
        input [31:0]SW1;
        input [31:0]SW2;
 4
        input [4:0]KEY;
        output reg [31:0]out;
 6
 7
8
       reg [31:0]outi;
9
10
11
12
    always@(*)
13
14 = case(KEY[4:2])
15
           //Pass out
16
           3'b000: outi = SW1;
           3'b001: outi = SW2 + SW1; // A+B
17
           3'b010: outi = SW1 - SW2; // A-B
3'b011: outi = SW1 & SW2; // A & B
18
19
20
           3'b100: outi = SW1 | SW2; // A OR B
          3'b101: outi = SW1 + 1; // A + 1
21
                                      // A - 1
22
          3'b110: outi = SW1 - 1;
                                    // B
23
          3'bll1: outi = SW2;
24
        endcase
   always@(*)
25
26
    case (KEY[1:0])
27
           2'b00: out = outi;
28
           2'b01: out = outi << 1;
29
           2'b10: out = outi >> 1;
30
           2'bl1: out = 0;
31
         endcase
32
33
      endmodule
34
```

Figura 5. Código da ULA de 32 Bits

Uma simulação do código criado foi realizada com as operações especificadas na Figura 4, a simulação é exibida na próxima seção deste relatório.

3. Avaliação dos resultados dos experimentos

3.1. Etapa 1



Figura 6. Operação 111 com 8 e 12 ambos em 4 Bits - PRESET

	Name	Value at 0 ps	0 ps 0 ps	40.0 ns	80.0 ns	120.0 ns	16
	,		3				
out	HEX0[0]	B 0					
out	HEX0[1]	B 0					
out	HEX0[2]	B 0					
out	HEX0[3]	В 0					
out	HEX0[4]	В 0					
out	HEX0[5]	В 0					
out	HEX0[6]	B 1					
in	KEY[0]	В 0					
in	KEY[1]	B 1					
in	KEY[2]	B 1					
in	SW1[0]	B 0					
in_	SW1[1]	B 1					
<u></u>	SW1[2]	B 1					
in	SW1[3]	В 0					
in_	SW2[0]	B 1					
in	SW2[1]	В 0					
in	SW2[2]	В 0					
in	SW2[3]	В 0					
out	out[0]	В 0					
out	out[1]	В 0					
out	out[2]	В 0					
out	out[3]	В 0					

Figura 7. Operação 110 (AND) com 0110 e 0001 ambos em 4 Bits, Saída = 0000.

	1200	Value at	0 ps	40.	0 ns	80	0 ns	120	.0 ns	160	.0 ns	
	Name	0 ps	0 ps									
245	HEXO[0]	B 0										
out	HEX0[1]	B 0										
out	HEX0[2]	B 0				1					Ш	
out -	HEX0[3]	B 0										
out	HEX0[4]	B 0										
eut E	HEX0[5]	B 0										
out S	HEX0[6]	B 1										Ī
13	KEY[0]	B 0										
13	KEY[1]	В 0										
5	KEY[2]	B 0										L
5	SW1[0]	B 0				li						
13	SW1[1]	B 0										
is.	SW1[2]	В 0										
5	SW1[3]	В 0										1
5	SW2[0]	B 0										
5-	SW2[1]	В 0										-
5	SW2[2]	В 0										
5	SW2[3]	В 0										
tuc	out[0]	В 0										-
out	out[1]	В 0										-
out	out[2]	В 0										-
out	out[3]	В 0										-

Figura 8. Operação 000 (CLEAR) com 0000 e 0000 ambos em 4 Bits, Saída = 0000.

	Name	Value at 0 ps	0 ps	40.0 ns	80.0 ns	120.0 ns	160.0 ns
out	HEX0[0]	80	3				
		50					
out	HEX0[1]	80					
out	HEX0[2]	B 1					
out	HEX0[3]	80					
out	HEX0[4]	B 0					
out	HEX0[5]	B 1					
out	HEX0[6]	80					
15	KEY[0]	B 1					
in_	KEY[1]	80					
15	KEY[2]	В 0					
in	SW1[0]	B 1					
in_	SW1[1]	B 0					
in	SW1[2]	В0					
in_	SW1[3]	В 0					
in	SW2[0]	B 1					
in_	SW2[1]	B 1					
15	SW2[2]	В0					
in_	SW2[3]	B 0					
out	out[0]	B 0					
out	out[1]	B 1					
out	out[2]	80					
out	out[3]	80					

Figura 9. Operação 001 (SW2 - SW1) com SW1 = 0001 e SW2 = 0011, Saída = 0010.

	Name	Value at 0 ps	0 ps 0 ps	40.0 ns	80.0 ns	120.0 ns	160.0 ns
out	HEX0[0]	ВО					
out	HEX0[1]	B 1					
out	HEX0[2]	B 0					
out	HEX0[3]	В 0					
out	HEX0[4]	B 1					
out	HEX0[5]	В 0					
out	HEX0[6]	В 0					
in	KEY[0]	во					
in	KEY[1]	B 1					
in	KEY[2]	В 0					
in	SW1[0]	В 0					
is	SW1[1]	В 0					
in_	SW1[2]	В 0					
in	SW1[3]	B 1					
in	SW2[0]	B 1					
is-	SW2[1]	B 1					
in	SW2[2]	B 0					
15	SW2[3]	В 0					
out	out[0]	B 1					
out	out[1]	B 0					
out	out[2]	B 1					
out	out[3]	В0					

Figura 10. Operação 010 (SW1 - SW2) com SW1 = 1000 e SW2 = 0011, Saída = 0101.

	Name	Value at	0 ps	40.0 ns	80.0 ns	120.0 ns	160.0
	Name	0 ps	0 ps				
out	HEXO[0]	B 0					
out -	HEX0[1]	B 0					
out	HEX0[2]	B 0					
out 	HEX0[3]	B 0					
out -	HEX0[4]	B 0					
out	HEX0[5]	B 0					
out	HEX0[6]	B 0					
in	KEY[0]	B 1					
in	KEY[1]	B 1					
is-	KEY[2]	B 0					
in	SW1[0]	B 1					
in	SW1[1]	B 1					
in	SW1[2]	B 1					
in	SW1[3]	В 0					
in_	SW2[0]	B 1					
in	SW2[1]	B 0					
in_	SW2[2]	B 0					
in	SW2[3]	B 0					
out	out[0]	B 0					
out	out[1]	В 0					
out	out[2]	B 0					
out	out[3]	B 1					

Figura 11. Operação 011 (ADD) com SW1 = 0111 e SW2 = 0001, Saída = 1000.

	Name	Value at 0 ps	0 ps 0 ps	40.0 ns	80.0 ns	120.0 ns	160.
out	HEX0[0]	во					
out	HEX0[1]	во					
out	HEX0[2]	во					
out	HEX0[3]	В 0					
out	HEX0[4]	В 0					
out	HEXO[5]	В 0					
out	HEX0[6]	B 1					
in	KEY[0]	В 0					
in	KEY[1]	В 0					
in	KEY[2]	B 1					
in	SW1[0]	B 1					
in	SW1[1]	В 0					
in	SW1[2]	B 1					
in_	SW1[3]	В 0					
in	SW2[0]	B 1					
in	SW2[1]	В 0					
in	SW2[2]	B 1					
in	SW2[3]	B 0					
out	out[0]	В 0					
out	out[1]	В 0					
out	out[2]	В 0					
out	out[3]	B 0					

Figura 12. Operação 100 (XOR) com SW1 = 0101 e SW2 = 0101, Saída = 0000.

		Value at	0 ps	40.0 ns	80.0 ns	120.0 ns	160
	Name	0 ps	0 ps	50			
out	HEX0[0]	В 0					
out -	HEX0[1]	В 0					
out	HEX0[2]	В 0					
out -	HEX0[3]	В 0					
out	HEX0[4]	B 1					
out	HEX0[5]	B 1					
out	HEX0[6]	B 0					
in	KEY[0]	B 1					
in	KEY[1]	B 0					
in	KEY[2]	B 1					
in_	SW1[0]	B 0					
in	SW1[1]	B 1					
in	SW1[2]	В 0					
in_	SW1[3]	B 0					
in_	SW2[0]	B 1					
in_	SW2[1]	B 0					
in	SW2[2]	В 0					
in	SW2[3]	B 0					
out	out[0]	B 1					
out	out[1]	B 1					
out	out[2]	В 0					
out	out[3]	В 0					

Figura 13. Operação 101 (OR) com SW1 = 0010 e SW2 = 0001, Saída = 0011.

3.1.1 Resultados Etapa 1 - Deploy na placa DE1 Altera.

Para o deploy na placa, foram utilizadas as KEYS de 0 à 3 para representar a operação desejadas. Nas KEYS, quando pressionadas representam o valor lógico 0 e quando não pressionadas o valor lógico 1.

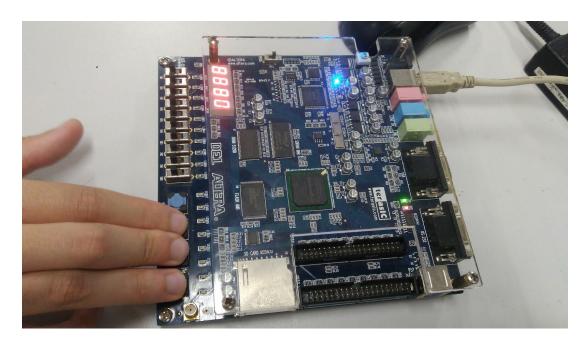


Imagem 1 - Operação CLEAR.

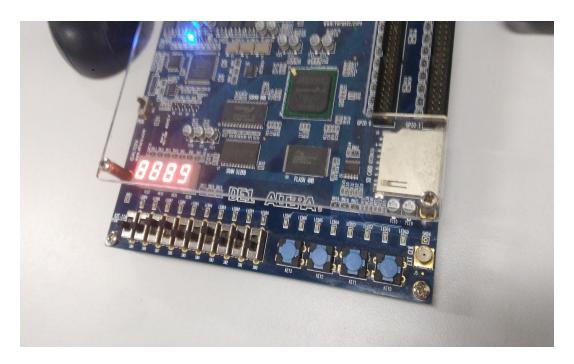


Imagem 2 - Operação SET

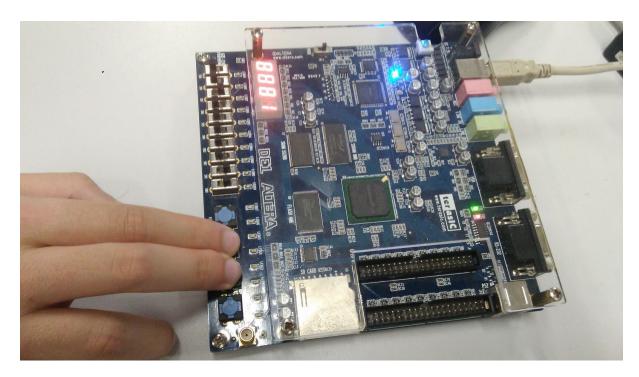


Imagem 3 - Operação 001 (SW2 - SW1) com SW1 = 0010 e SW2 = 0011, Saída = 0001.

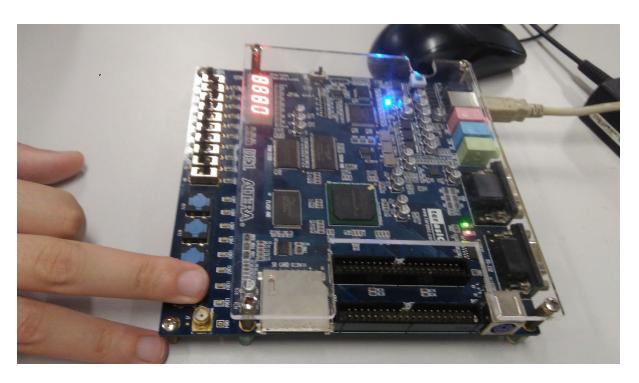


Imagem 4 - Operação 110 (AND) com 0110 e 0001 ambos em 4 Bits, Saída = 0000.

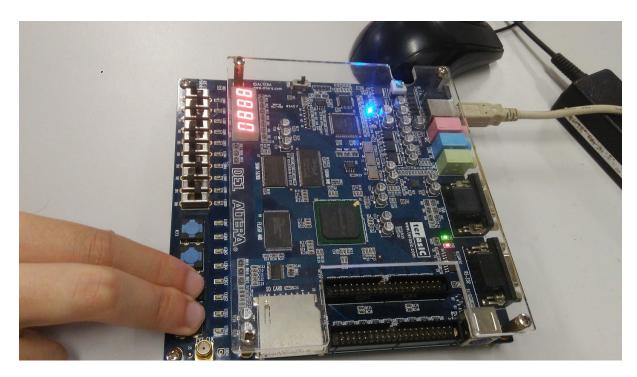


Imagem 5 -. Operação 100 (XOR) com SW1 = 0101 e SW2 = 0101, Saída = 0000.

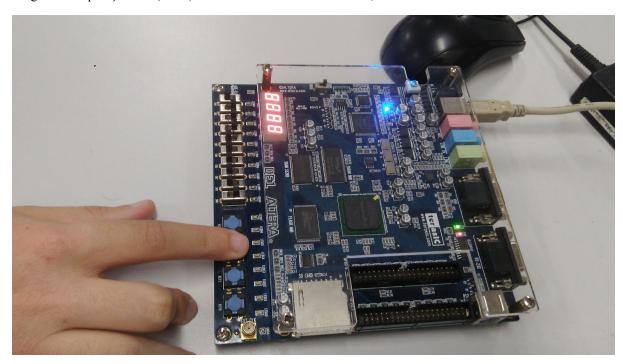


Imagem 6 -. Operação 011 (ADD) com SW1 = 0111 e SW2 = 0001, Saída = 1000.

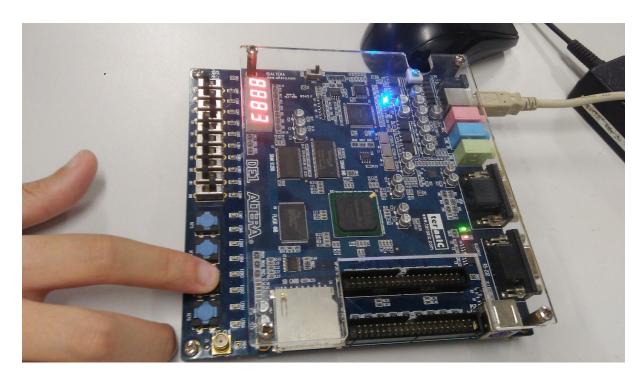


Imagem 7 -. Operação 101 (OR) com SW1 = 0011 e SW2 = 0001, Saída = 0011.

3.2. Etapa 2

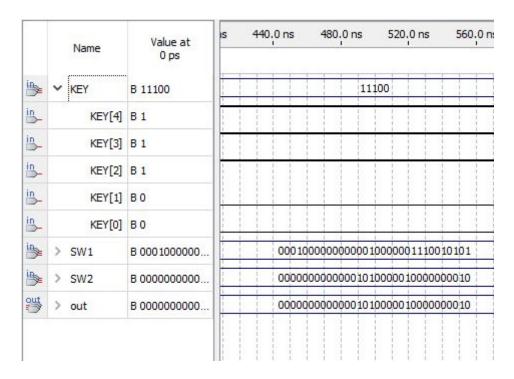


Figura 14. Operação 11100 com 268501909 e 328706 ambos em 32 Bits - PASS B e PASS ALL_OUT

	Name	Value at 0 ps	440.0 ns	480.0 ns	520.0 ns	560.0 ns
in the second	> KEY	B 00000		000	000	
is.	> SW1	В 000 1000000	00010	00000000000	00000011100	10101
<u>i</u>	> SW2	В 0000000000	00000	00000000101	00000100000	00010
34	> out	B 000 1000000	00010	000000000001	00000011100	10101

Figura 15. Operação 00000 com 268501909 e 328706 ambos em 32 Bits - PASS A e PASS ALL_OUT

	Name	Value at 0 ps	40.0 ns 480.0 ns 520.0 ns 560.a
<u>i</u>	✓ KEY	B 00001	00001
in	KEY[4]	B 0	
in_	KEY[3]	B 0	
in_	KEY[2]	B 0	
in	KEY[1]	B 0	
in	KEY[0]	B 1	
ing.	> SW1	B 0001000000	0001000000000010000001110010101
in_	> SW2	B 0000000000	000000000000000000000000000000000000000
**	> out	B 0010000000	00,100,000,000,100,000,000,110,010,1010

Figura 16. Operação 00001 com 268501909 e 328706 ambos em 32 Bits - PASS A e SHIFT RIGHT

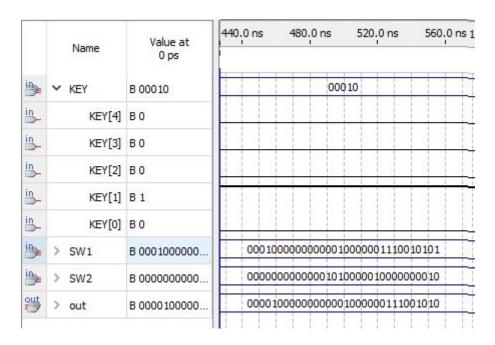


Figura 17. Operação 00010 com 268501909 e 328706 ambos em 32 Bits - PASS A e SHIFT LEFT

	Name	Value at 0 ps	440.0 ns 480.0 ns 520.0 ns 560.
	> KEY	B 00100	00100
	> SW1	B 0001000000	0001000000000010000001110010101
	> SW2	В 0000000000	0000000000001010000010000000010
**	> out	B 0001000000	00010000000001100000011110010111

Figura 18. Operação 00100 com 268501909 e 328706 ambos em 32 Bits - SOMA de SW1,SW2 e PASS ALL_OUT

	Name	Value at 0 ps	440.0 ns	480.0 ns	520.0 ns	560,0 ns
1	✓ KEY	B 01011		010)11	
in_	KEY[4]	В0				
in_	KEY[3]	B 1				
in	KEY[2]	В0				
in	KEY[1]	B 1				
in	KEY[0]	B 1				
<u>i</u>	> SW1	B 0001000000	00010	00000000001	00000011100	10101
	> SW2	В 0000000000	00000	00000000101	00000100000	00010
**	> out	В 0000000000	00000	0000000000	000000000000	00000

Figura 19. Operação 01011 com 268501909 e 328706 ambos em 32 Bits - Subtração entre SW1,SW2 e PASS 0's

	Name	Value at 0 ps	100.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 n
	✓ KEY	B 01100			011	.00	
in	KEY[4]	В0	1 1				
in	KEY[3]	B 1					
in	KEY[2]	B 1					
in	KEY[1]	В0					
in	KEY[0]	В0					
	> SW1	B 0001000000		00010	00000000001	00000011100	10101
	> SW2	В 000000000		00000	00000000101	00000 1000000	00010
**	> out	В 0000000000		00000	000000000001	00000000000	00000

Figura 20. Operação 01100 com 268501909 e 328706 ambos em 32 Bits - LOGICAL AND entre SW1, SW2 e PASS ALL_OUT

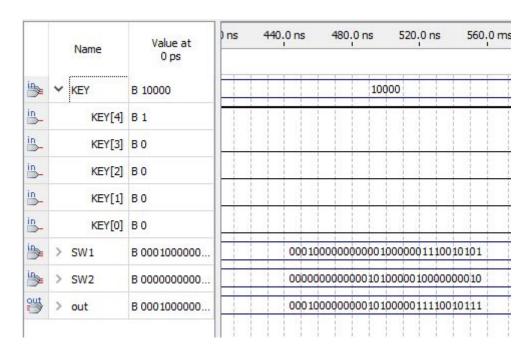


Figura 21. Operação 10000 com 268501909 e 328706 ambos em 32 Bits - LOGICAL OR entre SW1 e SW2 e PASS ALL_OUT

	Name	Value at 0 ps	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 n
 	✓ KEY	B 10100			101	.00	
in_	KEY[4] B 1					
in	KEY[3] B 0					
in_	KEY[2] B 1					
in	KEY[1] B 0					
in	KEY[0	D] B 0					
<u></u>	> SW1	B 0001000000		00010	00000000000	00000011100	0101
in	> SW2	В 0000000000		00000	000000000101	000001000000	0010
**	> out	B 000 1000000		00010	00000000001	00000011100	0110

Figura 22. Operação 10100 com 268501909 e 328706 ambos em 32 Bits - INCREMENT + 1 SW1 e PASS ALL_OUT

	Name	Value at 0 ps	1S	440.0 ns	480.0 ns	520,0 ns	560,0 n) n
B	✓ KEY	B 11000				11000	
in	KEY[4]	B 1	T				
in_	KEY[3]	B 1	T				
in_	KEY[2]	В0	111				
in	KEY[1]	B 0					
in_	KEY[0]	В0					
is.	> SW1	B 0001000000		000100	00000000	0100000011100	10101
in	> SW2	В 0000000000		000000	000000001	0 100000 100000	00010
**	> out	B 000 1000000		000100	00000000	0100000011100	10100

Figura 23. Operação 11000 com 268501909 e 328706 ambos em 32 Bits - DECREMENT SW1 e PASS ALL_OUT

4. Análise crítica e discussão

Através dos estudos realizados é possível entender como uma Unidade Lógica Aritmética trabalha dentro de uma CPU, também é possível perceber o quanto uma linguagem de descrição de hardware facilita o trabalho na construção de circuitos digitais, pois como a ULA trabalha com operações lógicas e aritméticas, simplesmente foi necessário estabelecer quais seriam os operandos e definir a operação entre estes, através do Verilog isto se deu apenas com 1 linha de código, se o operador fosse de soma, e operandos fossem definidos como SW1 e SW2, apenas é necessário fazer a saída receber SW1 + SW2, pensando isso em diagrama de blocos, traria um trabalho muito mais custoso, e para a exibição na placa dos resultados, os experimentos anteriores foram o suficientes para facilitar todo procedimento.