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.

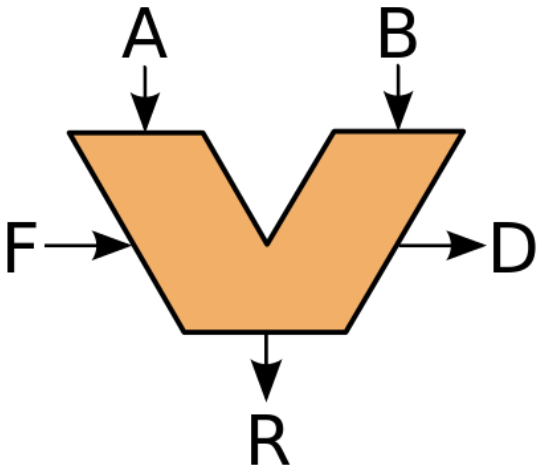


Imagem 0 - Representação da Unidade Lógica Aritmética

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.

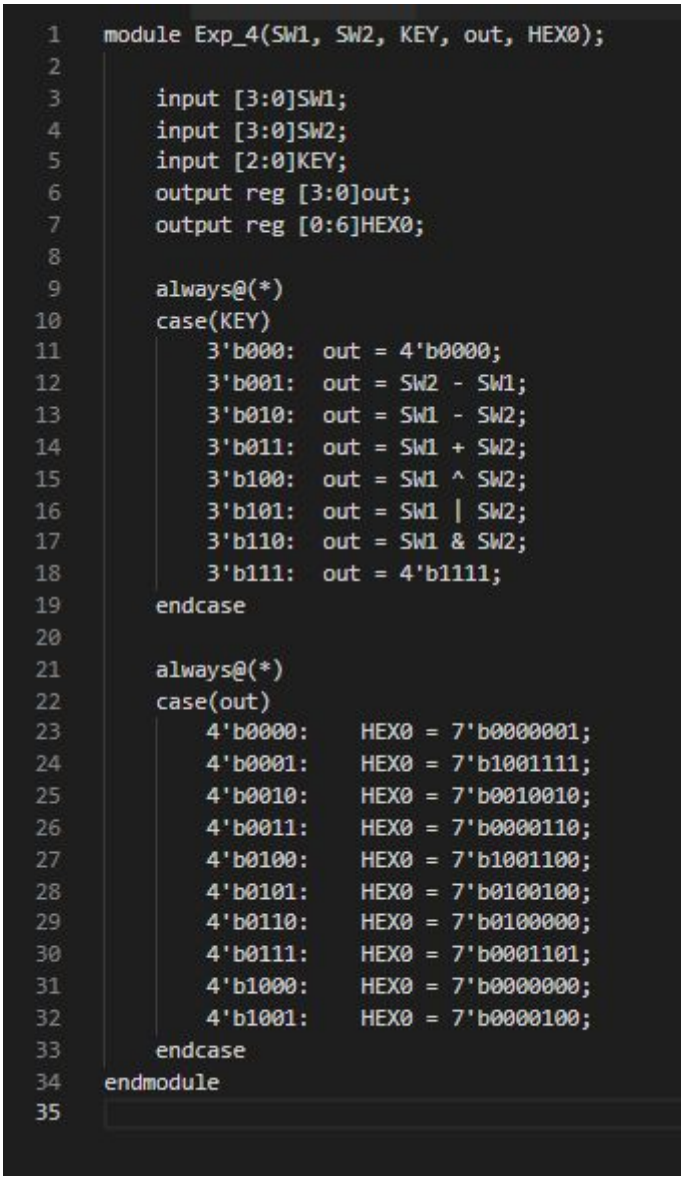
2. Descrição da execução do experimento

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;
        4'b0010:    HEX0 = 7'b0010010;
        4'b0011:    HEX0 = 7'b0000110;
        4'b0100:    HEX0 = 7'b1001100;
        4'b0101:    HEX0 = 7'b0100100;
        4'b0110:    HEX0 = 7'b0100000;
        4'b0111:    HEX0 = 7'b0001101;
        4'b1000:    HEX0 = 7'b0000000;
        4'b1001:    HEX0 = 7'b0000100;
    endcase
endmodule

```

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.

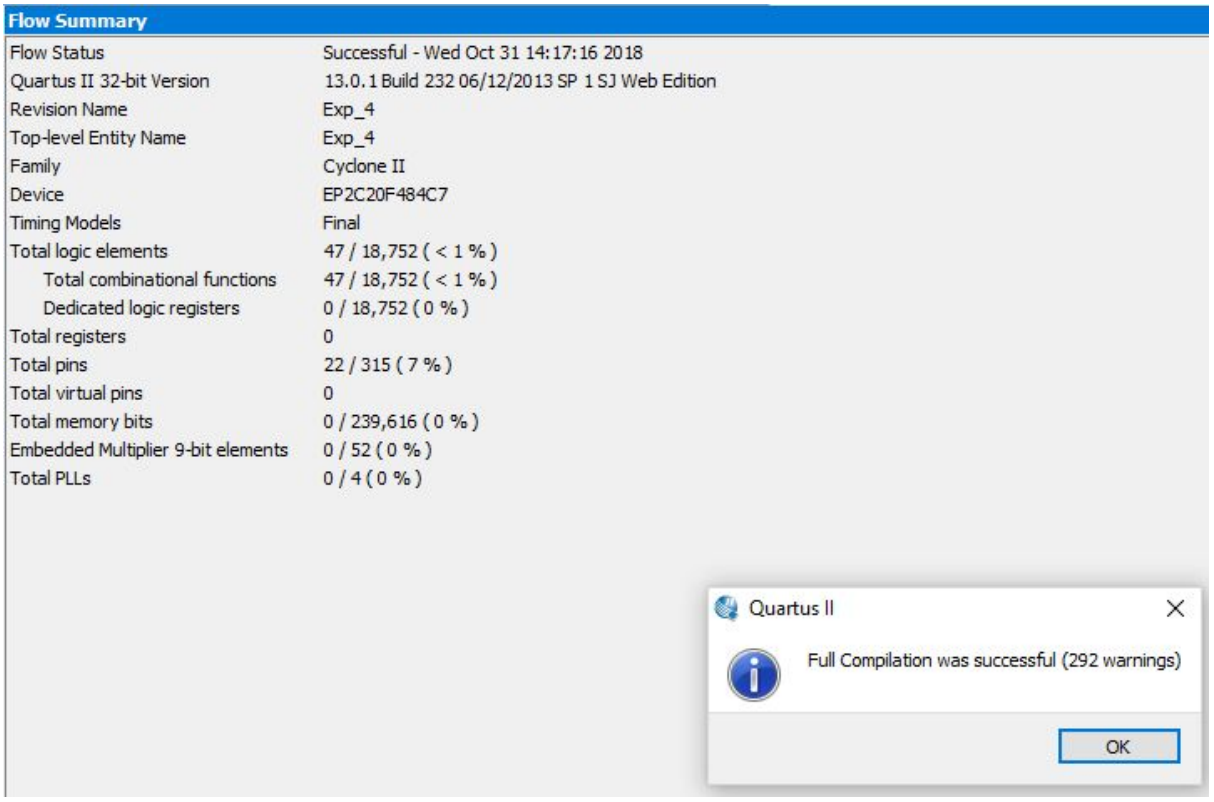


Figura 3. Compilação do código da ULA de 4 Bits

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);

   input [31:0]SW1;
   input [31:0]SW2;
   input [4:0]KEY;
   output reg [31:0]out;

  reg [31:0]outi;



always@(*)

   case(KEY[4:2])
      //Pass out
      3'b000:  outi = SW1;           //  A
      3'b001:  outi = SW2 + SW1; //  A+B
      3'b010:  outi = SW1 - SW2; // A-B
      3'b011:  outi = SW1 & SW2; // A & B
      3'b100:  outi = SW1 | SW2; // A OR B
      3'b101:  outi = SW1 + 1;      //  A + 1
      3'b110:  outi = SW1 - 1;      //  A - 1
      3'b111:  outi = SW2;        // B
   endcase
always@(*)
   case(KEY[1:0])
      2'b00: out = outi;
      2'b01: out = outi << 1;
      2'b10: out = outi >> 1;
      2'b11: out = 0;
   endcase

endmodule
```

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

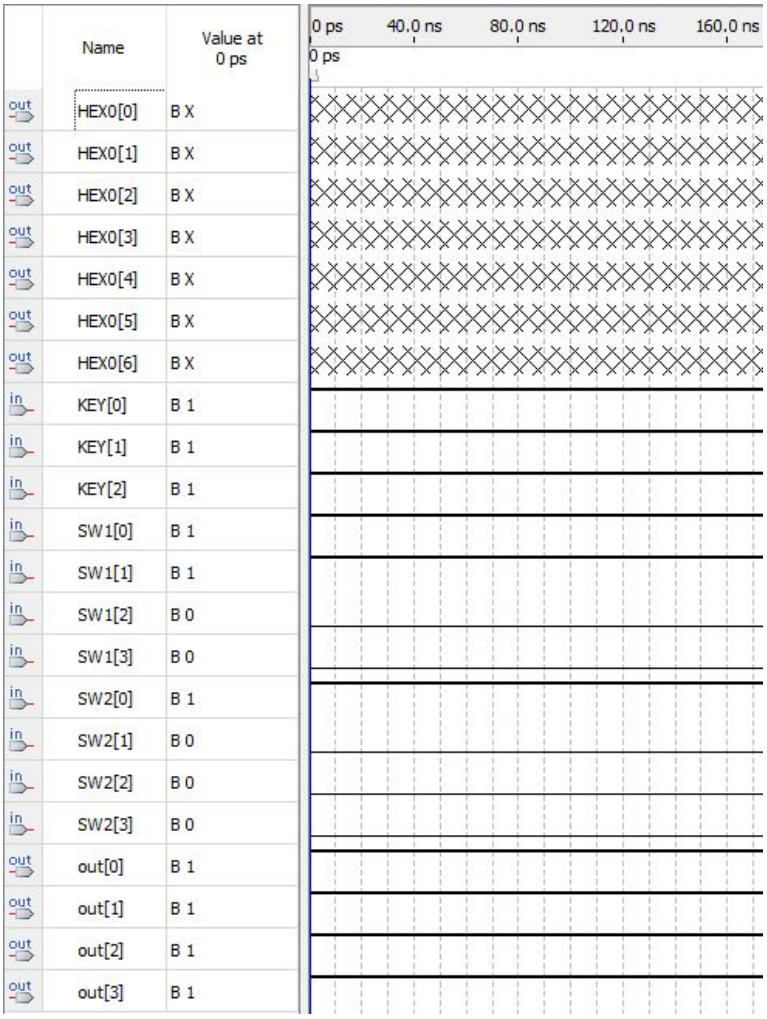


Figura 6. Operação 111 com 8 e 12 ambos em 4 Bits - PRESET

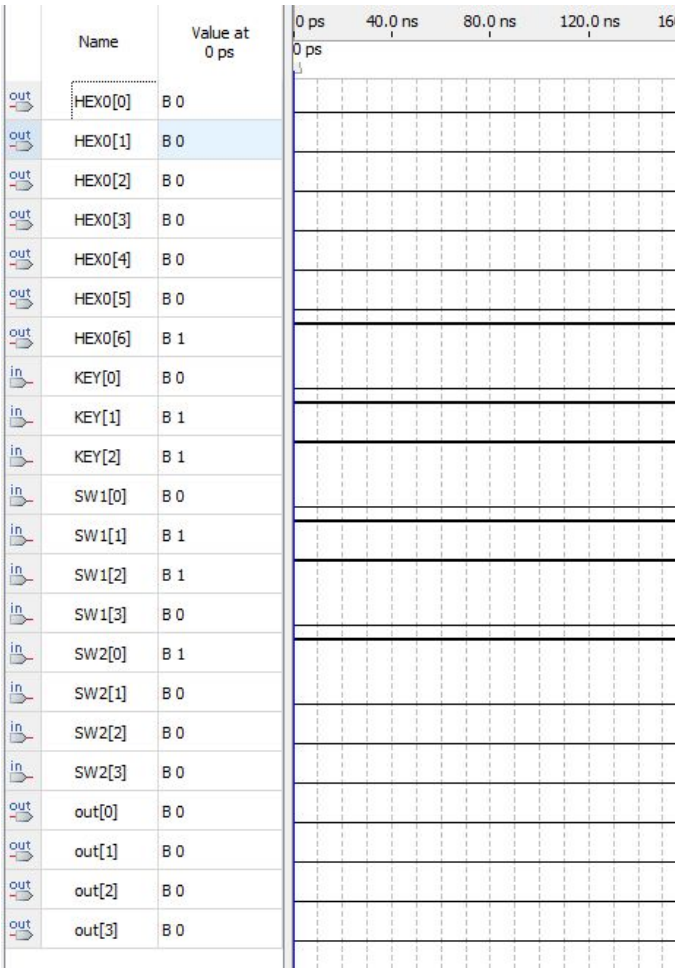


Figura 7. Operação 110 (AND) com 0110 e 0001 ambos em 4 Bits, Saída = 0000.

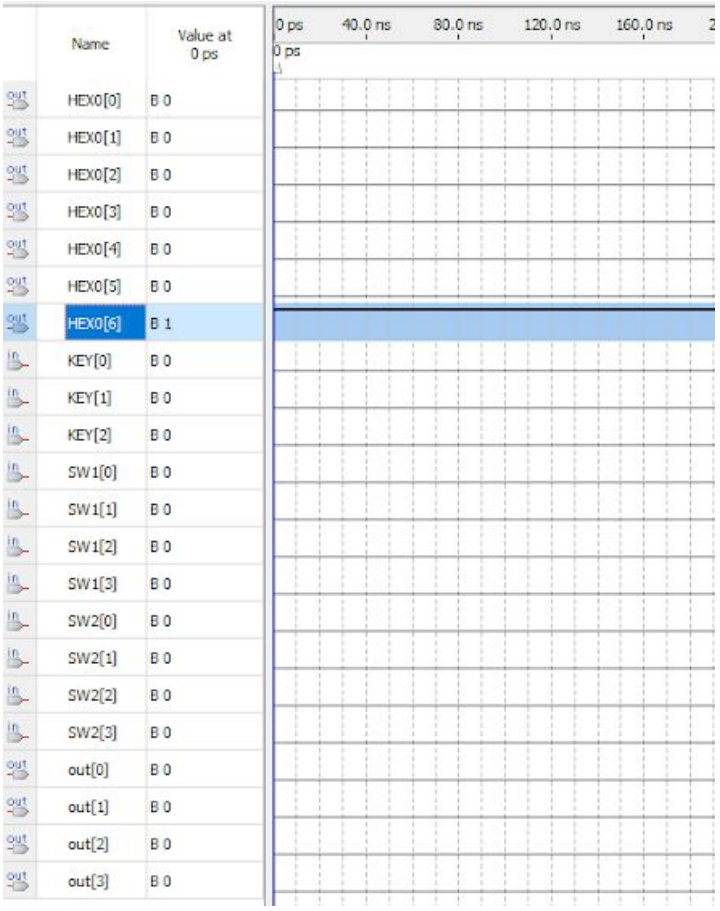


Figura 8. Operação 000 (CLEAR) com 0000 e 0000 ambos em 4 Bits, Saída = 0000.

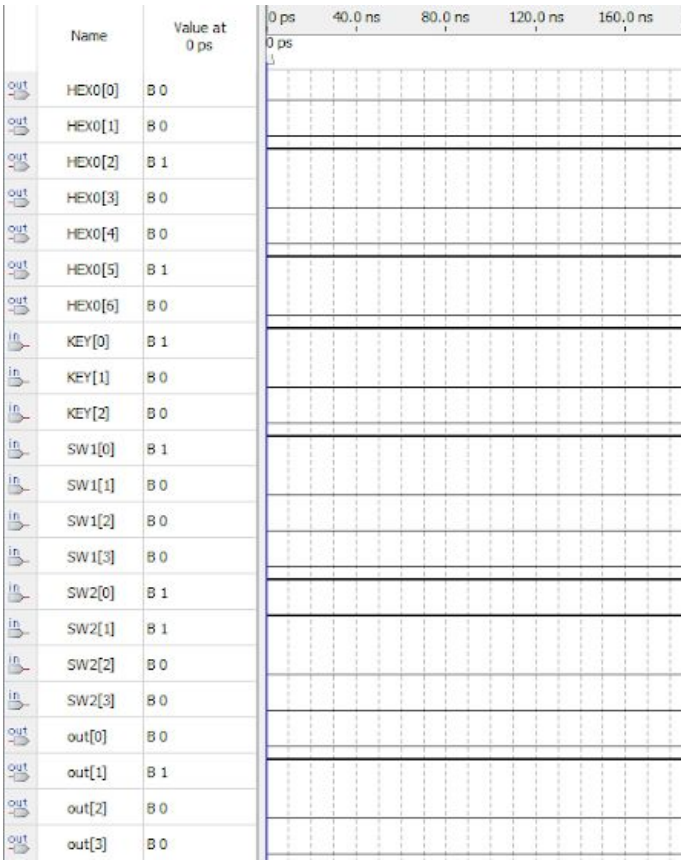


Figura 9. Operação 001 (SW2 - SW1) com SW1 = 0001 e SW2 = 0011, Saída = 0010.

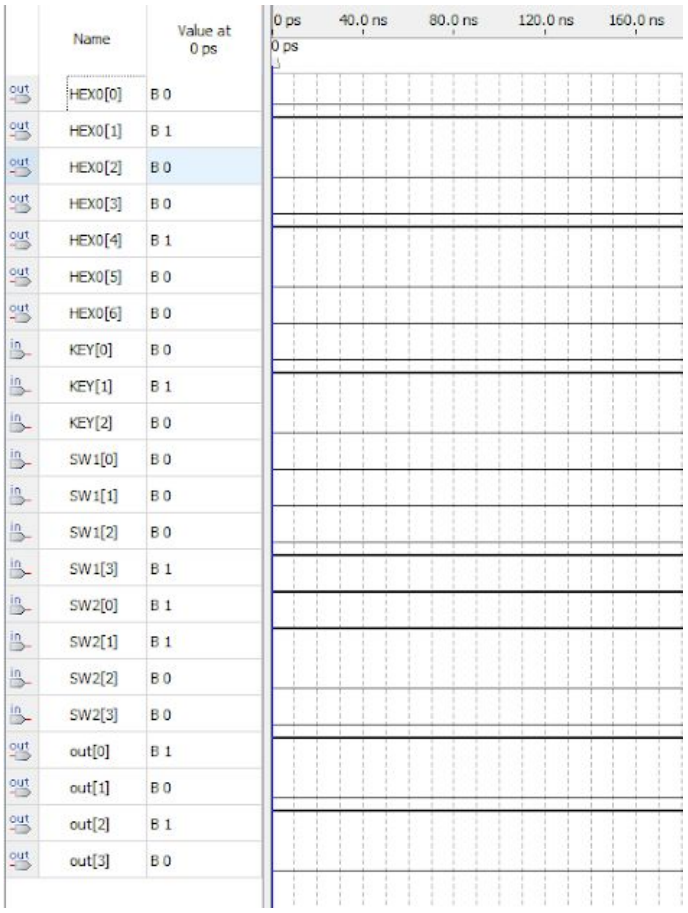


Figura 10. Operação 010 (SW1 - SW2) com SW1 = 1000 e SW2 = 0011, Saída = 0101.

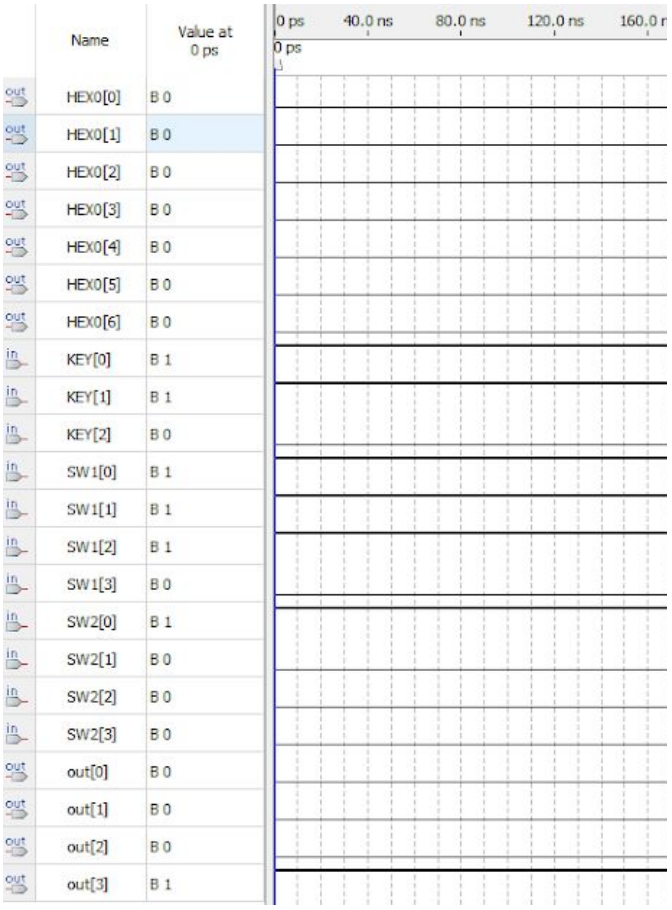


Figura 11. Operação 011 (ADD) com SW1 = 0111 e SW2 = 0001, Saída = 1000.

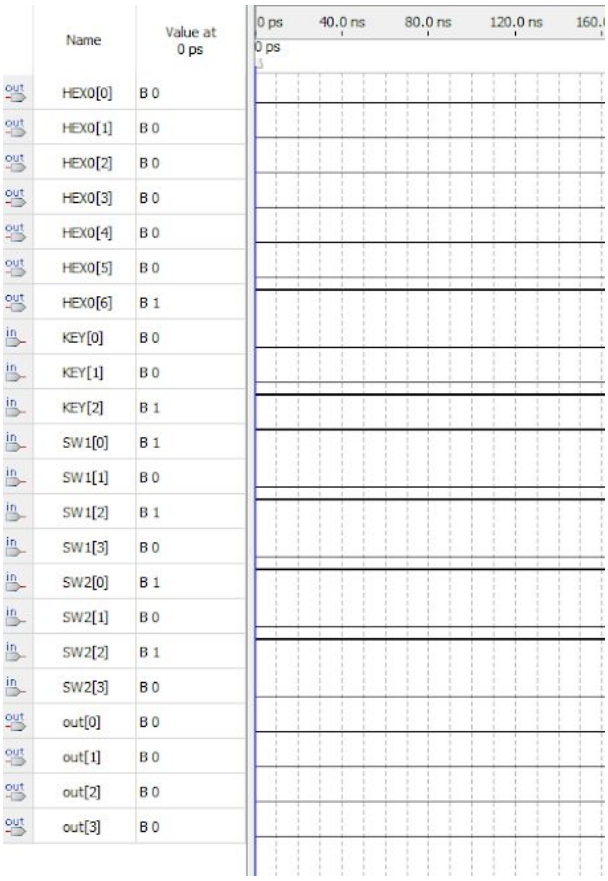


Figura 12. Operação 100 (XOR) com SW1 = 0101 e SW2 = 0101, Saída = 0000.

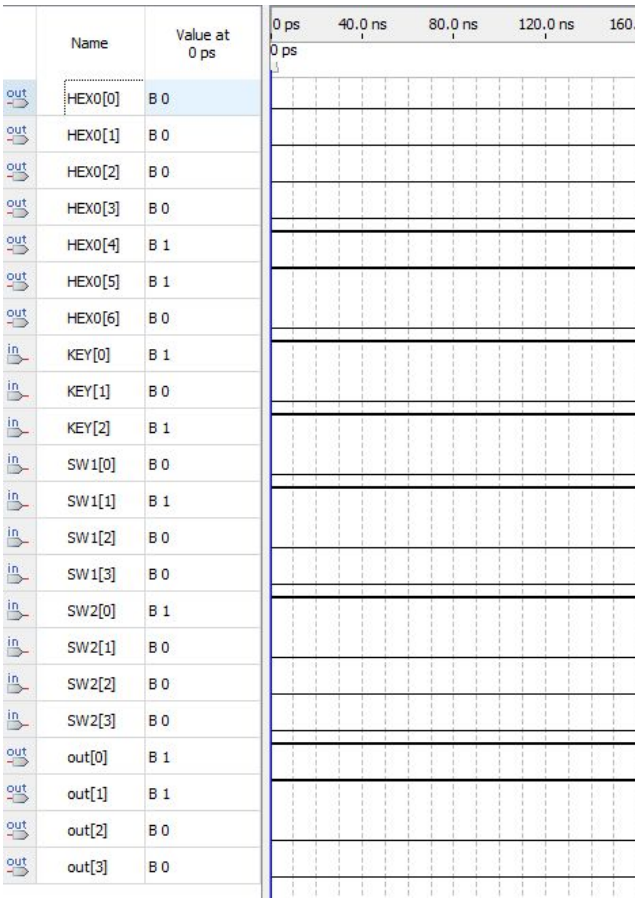


Figura 13. Operação 101 (OR) com SW1 = 0010 e SW2 = 0001, Saída = 0011.

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.

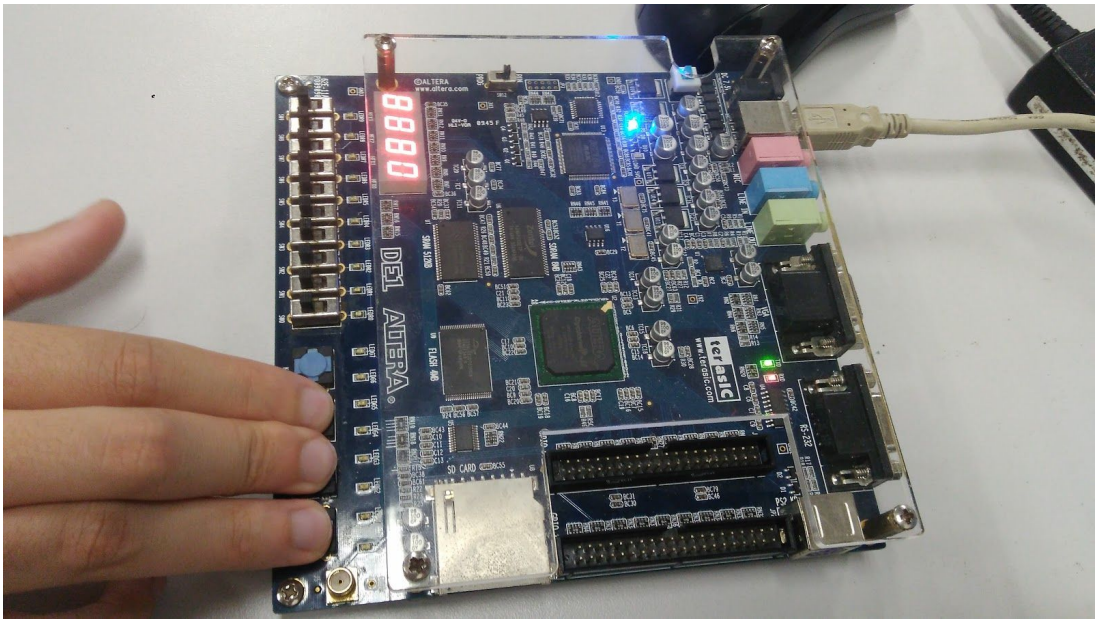


Imagem 1 - Operação CLEAR.

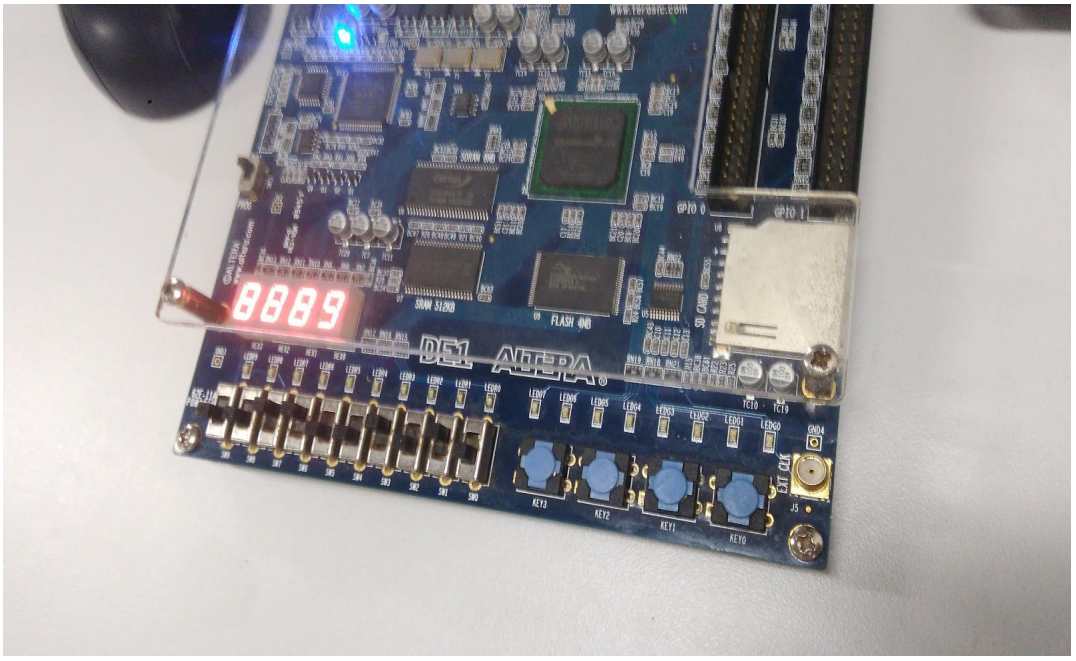


Imagem 2 - Operação SET

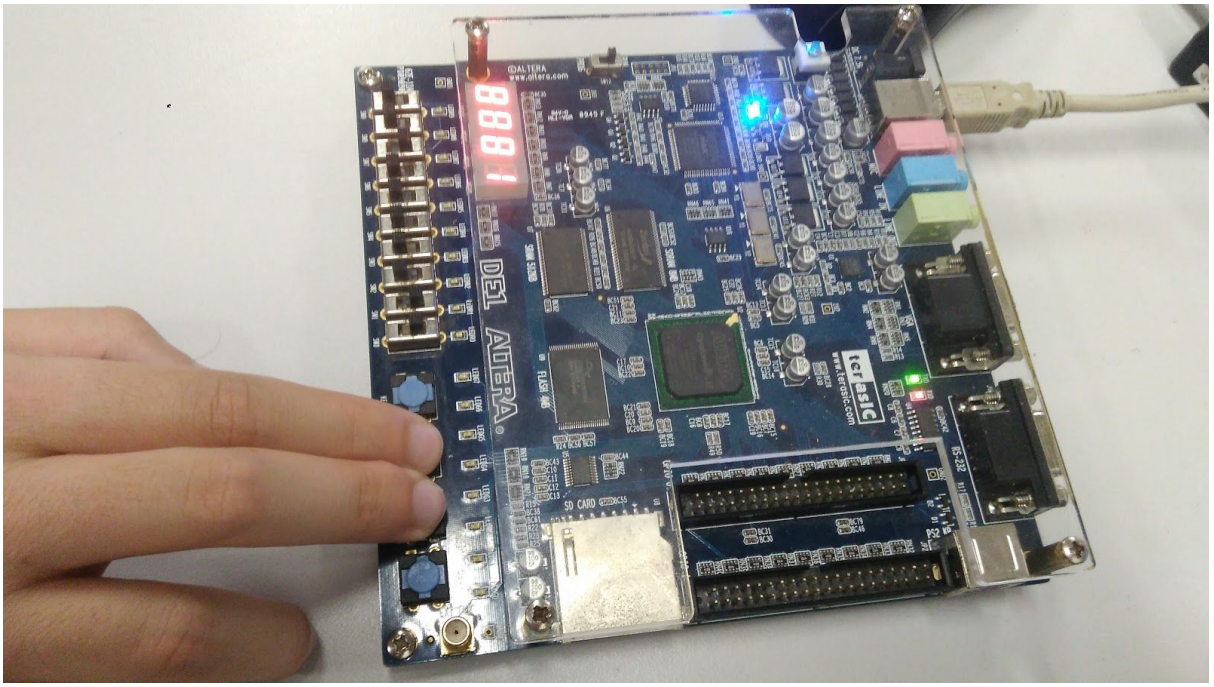


Imagem 3 - Operação 001 (SW2 - SW1) com SW1 = 0010 e SW2 = 0011, Saída = 0001.

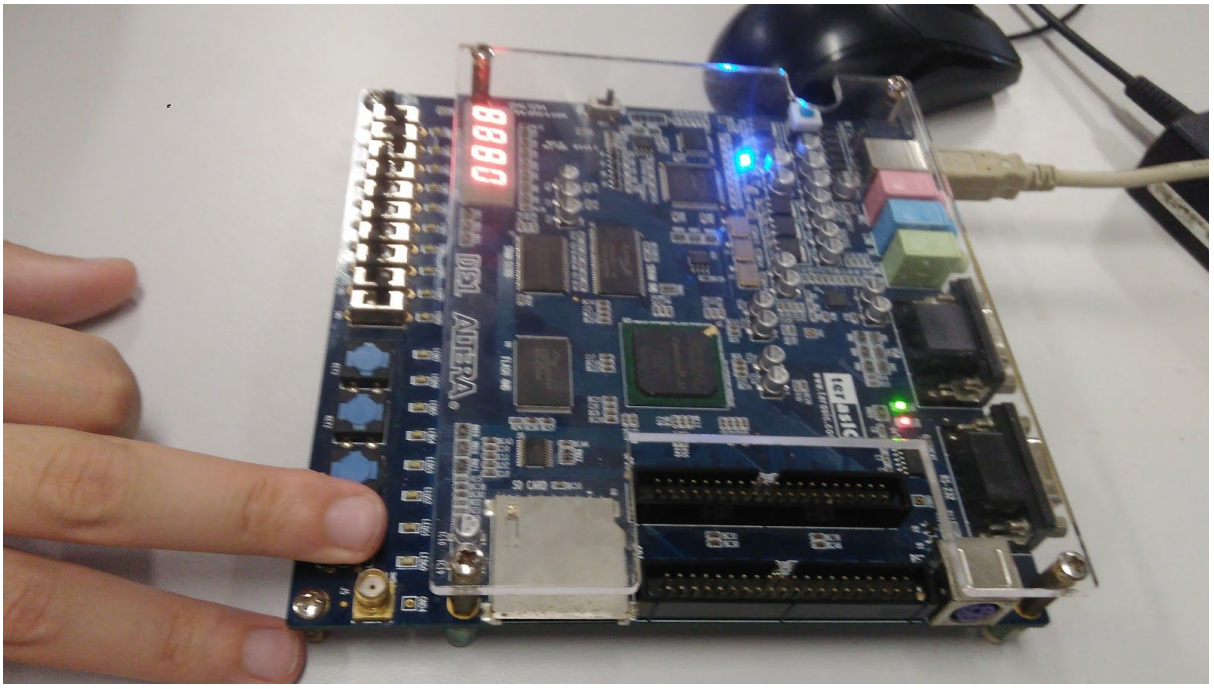


Imagem 4 - Operação 110 (AND) com 0110 e 0001 ambos em 4 Bits, Saída = 0000.

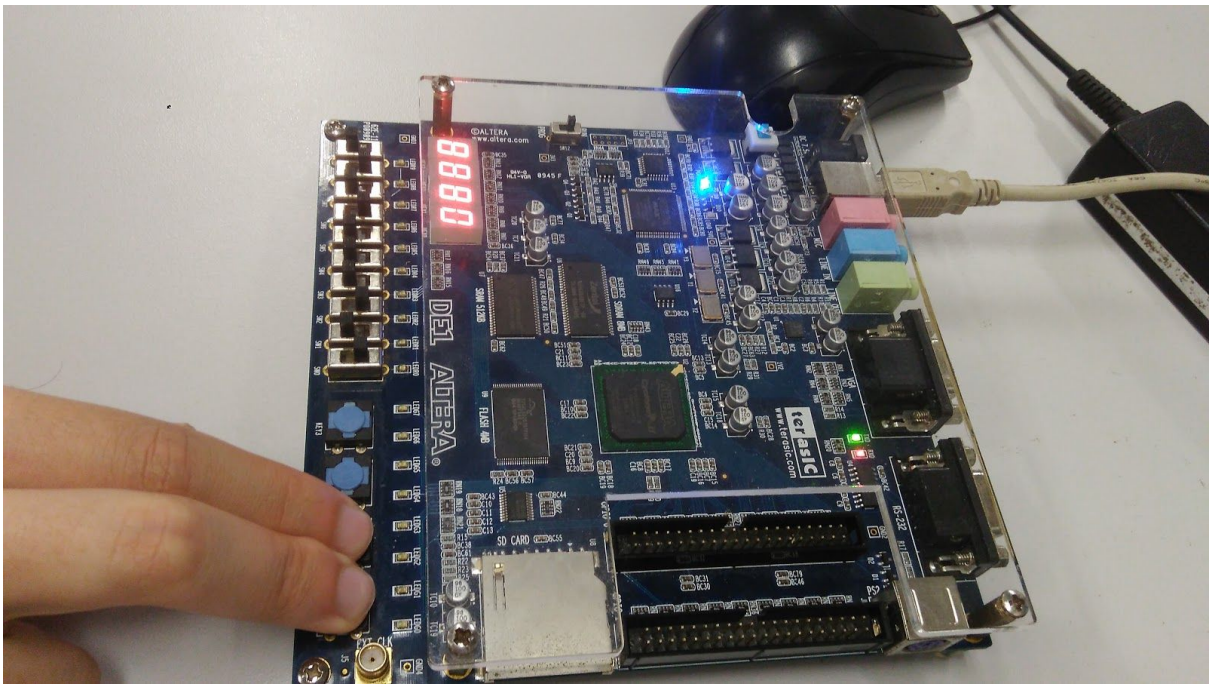


Imagem 5 -. Operação 100 (XOR) com SW1 = 0101 e SW2 = 0101, Saída = 0000.

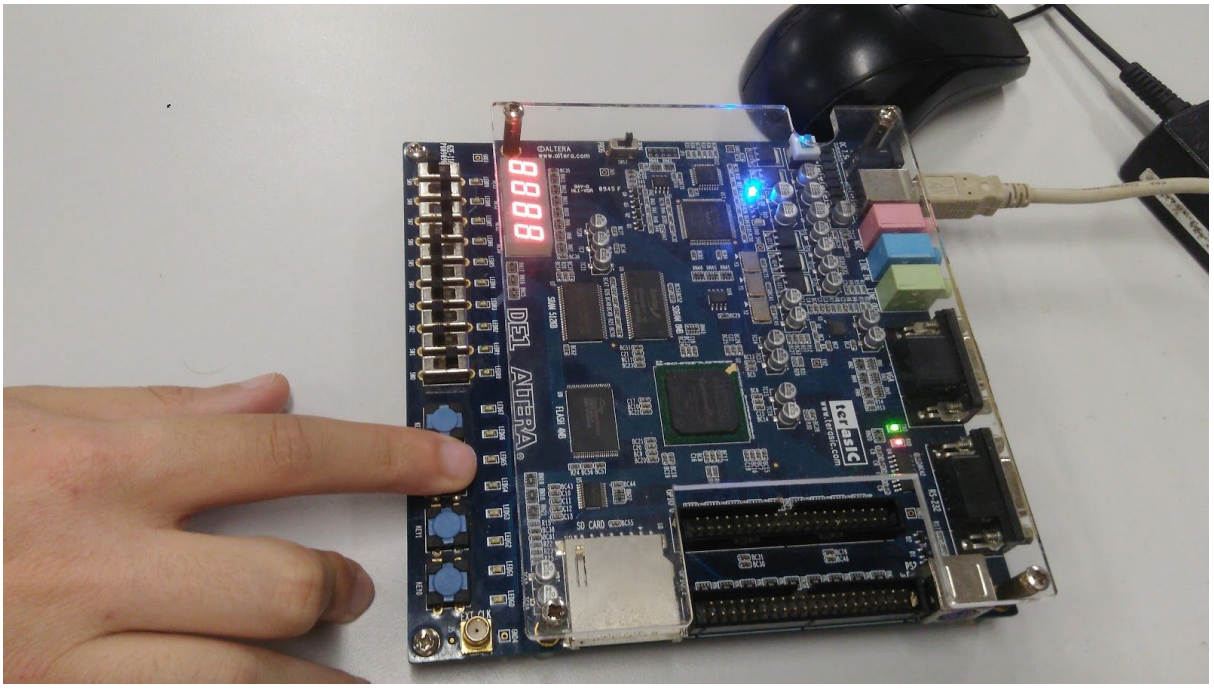


Imagem 6 -. Operação 011 (ADD) com SW1 = 0111 e SW2 = 0001, Saída = 1000.

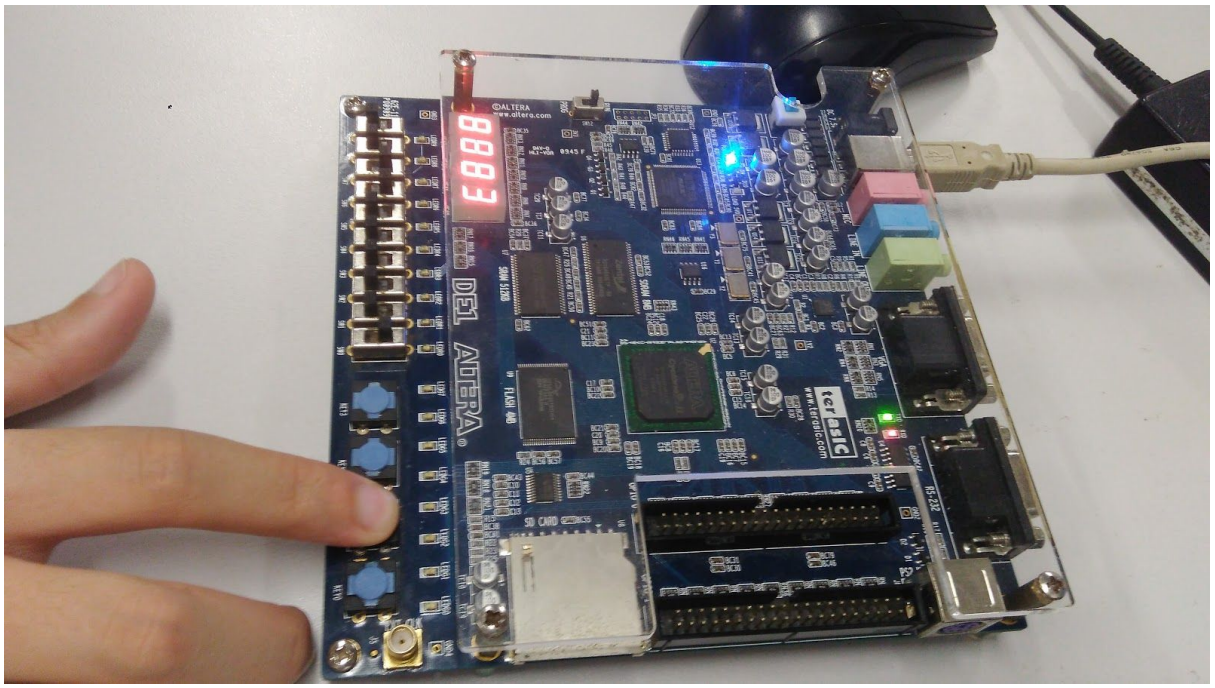


Imagem 7 -. Operação 101 (OR) com SW1 = 0011 e SW2 = 0001, Saída = 0011.

3.2. Etapa 2

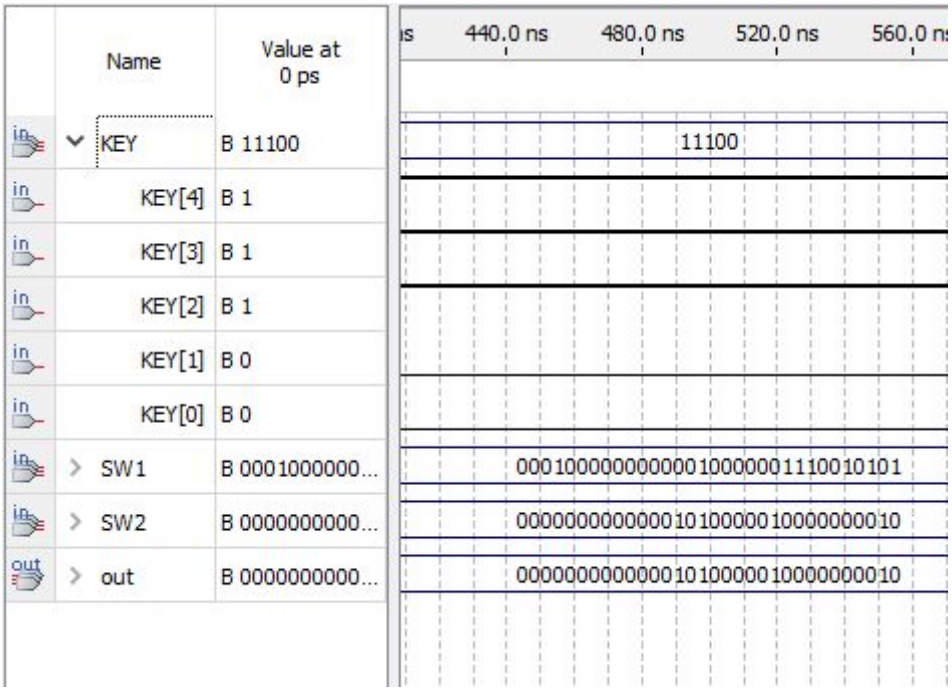


Figura 14. Operação 11100 com 268501909 e 328706 ambos em 32 Bits - PASS B e PASS ALL_OUT

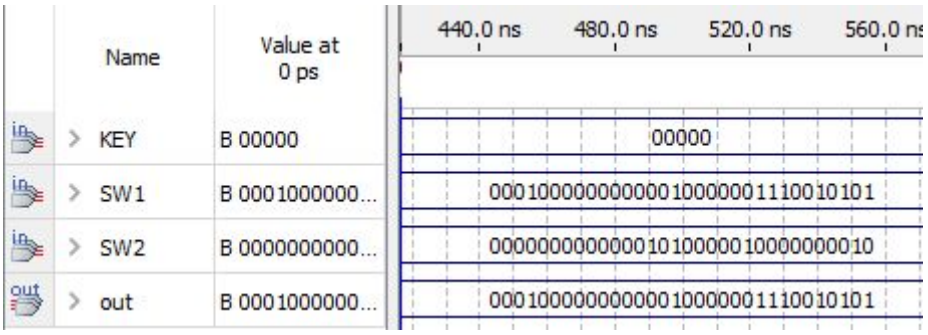


Figura 15. Operação 00000 com 268501909 e 328706 ambos em 32 Bits - PASS A e PASS ALL_OUT

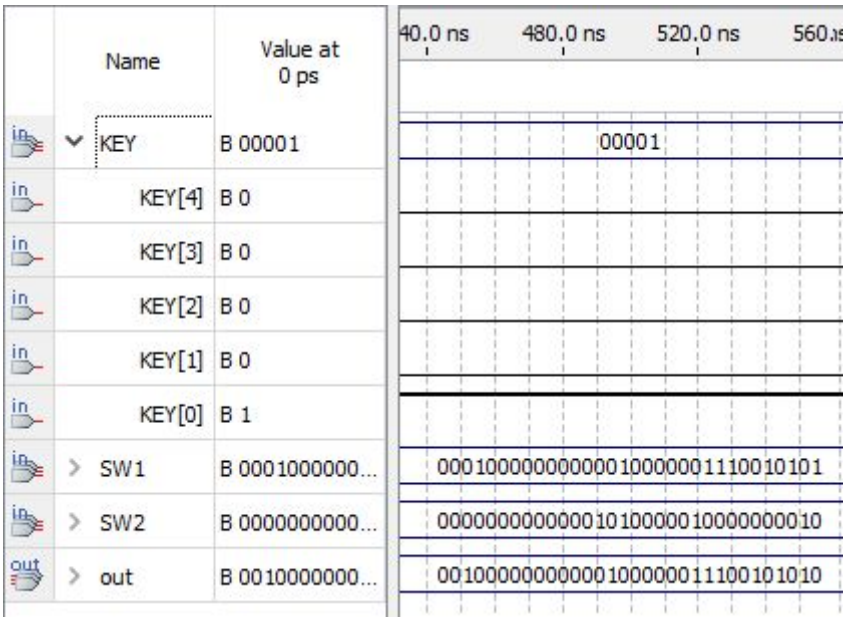


Figura 16. Operação 00001 com 268501909 e 328706 ambos em 32 Bits - PASS A e SHIFT RIGHT

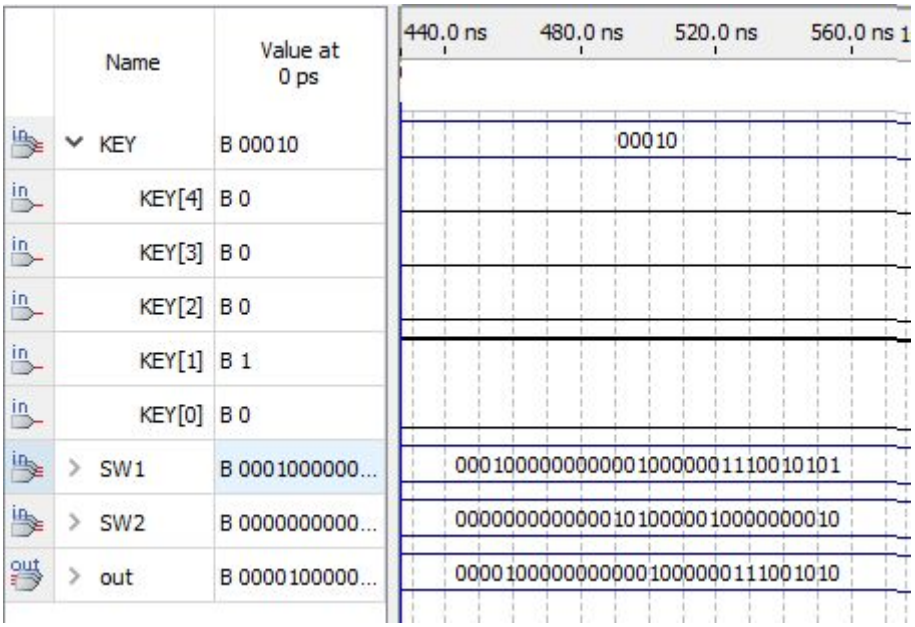


Figura 17. Operação 00010 com 268501909 e 328706 ambos em 32 Bits - PASS A e SHIFT LEFT

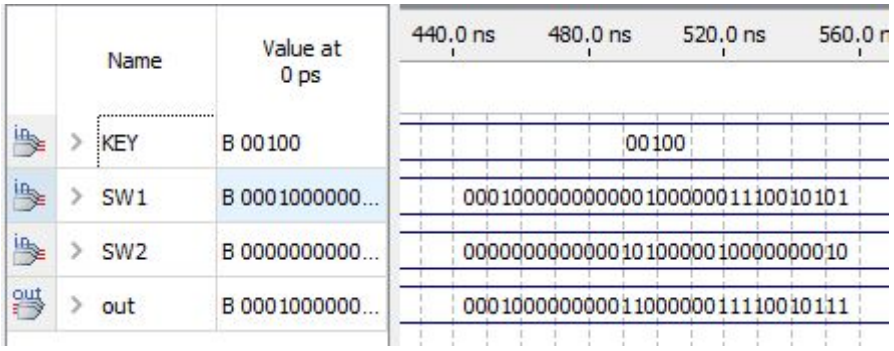


Figura 18. Operação 00100 com 268501909 e 328706 ambos em 32 Bits - SOMA de SW1,SW2 e PASS ALL_OUT

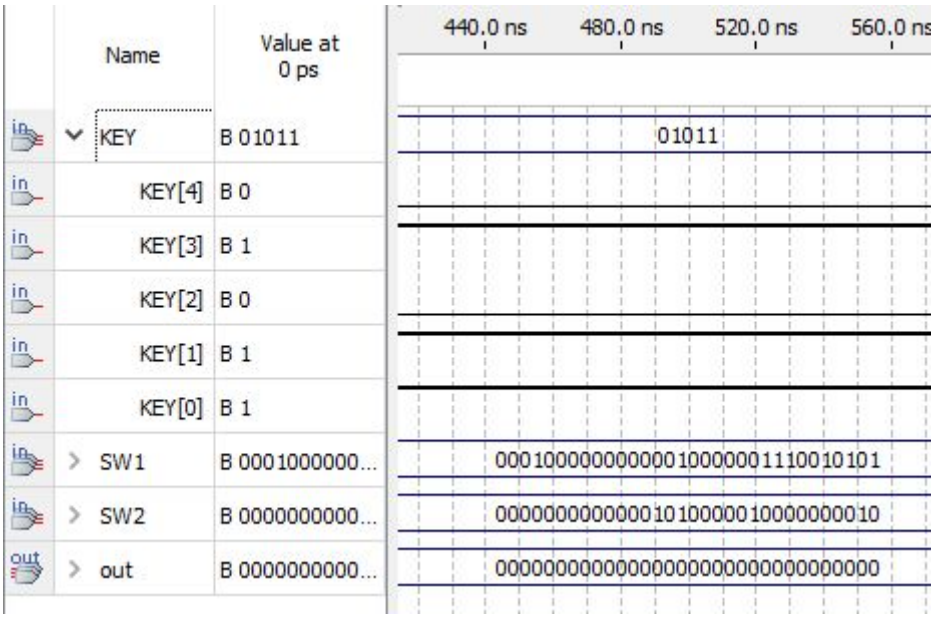


Figura 19. Operação 01011 com 268501909 e 328706 ambos em 32 Bits - Subtração entre SW1,SW2 e PASS 0's

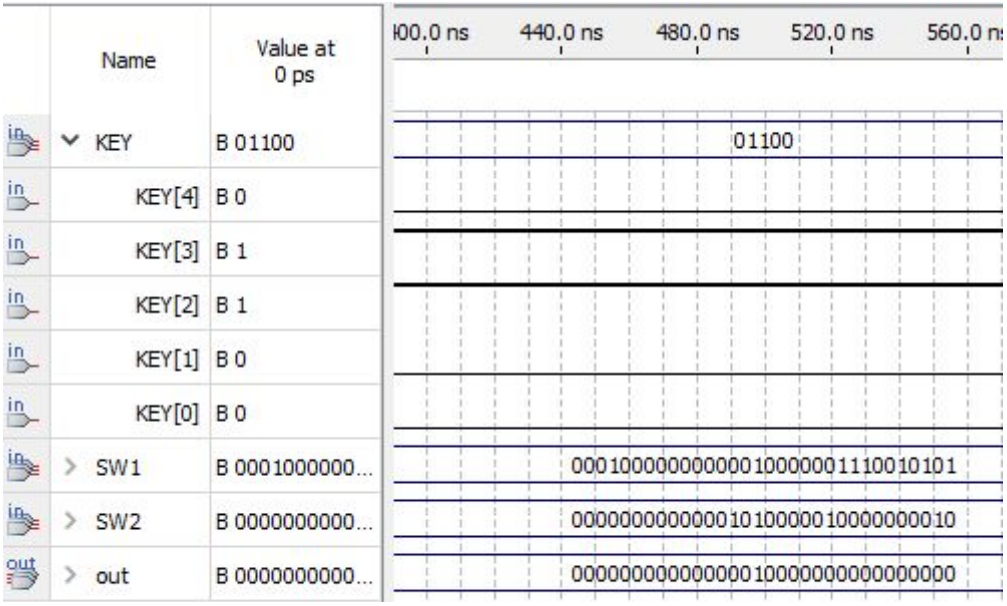


Figura 20. Operação 01100 com 268501909 e 328706 ambos em 32 Bits - LOGICAL AND entre SW1, SW2 e PASS ALL_OUT

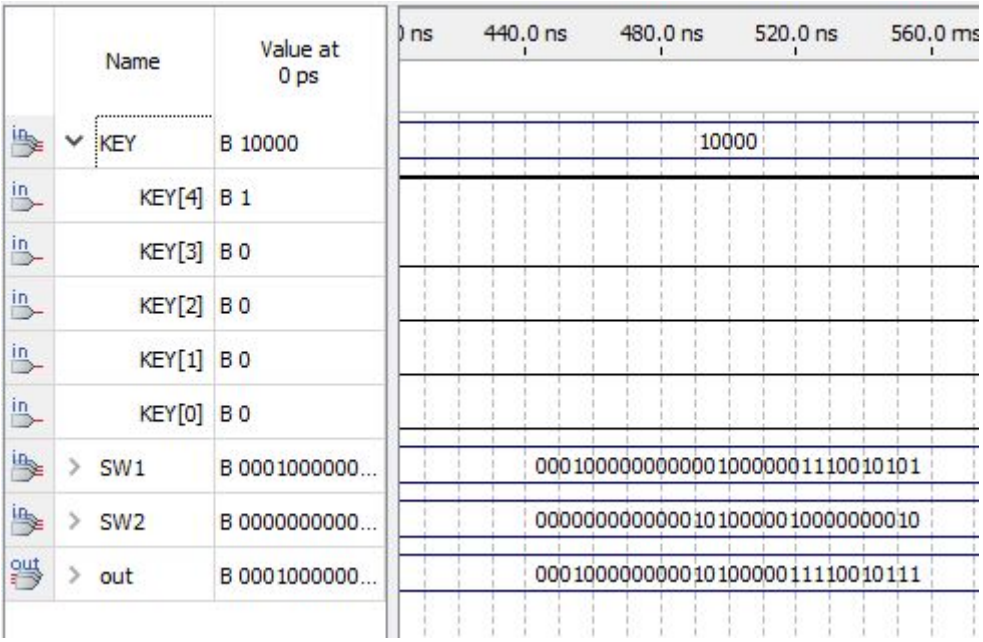


Figura 21. Operação 10000 com 268501909 e 328706 ambos em 32 Bits - LOGICAL OR entre SW1 e SW2 e PASS ALL_OUT

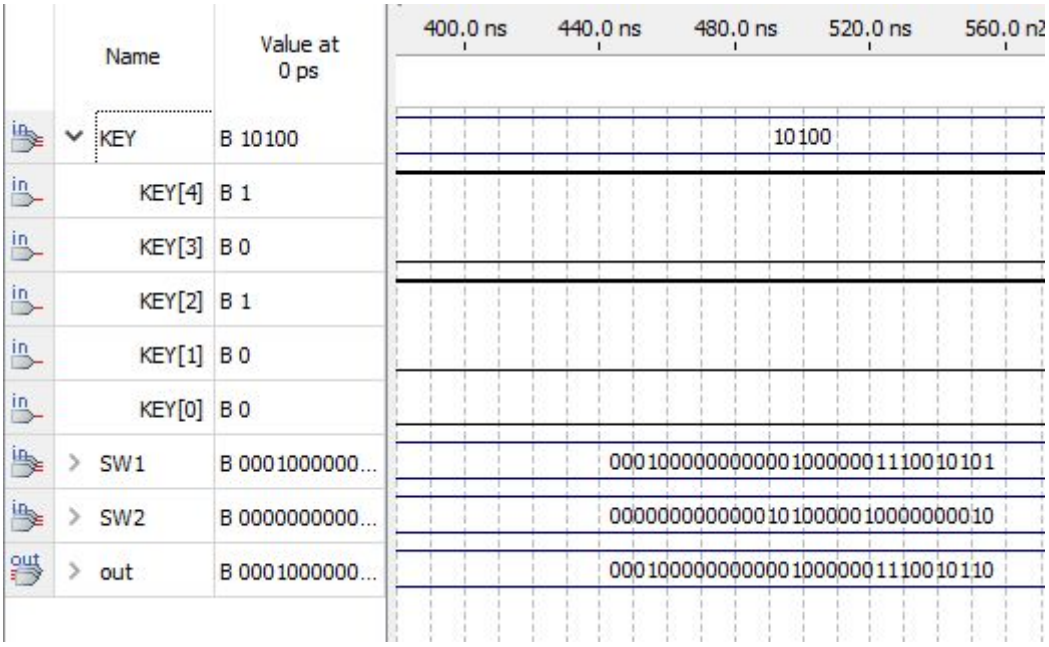


Figura 22. Operação 10100 com 268501909 e 328706 ambos em 32 Bits - INCREMENT + 1 SW1 e PASS ALL_OUT

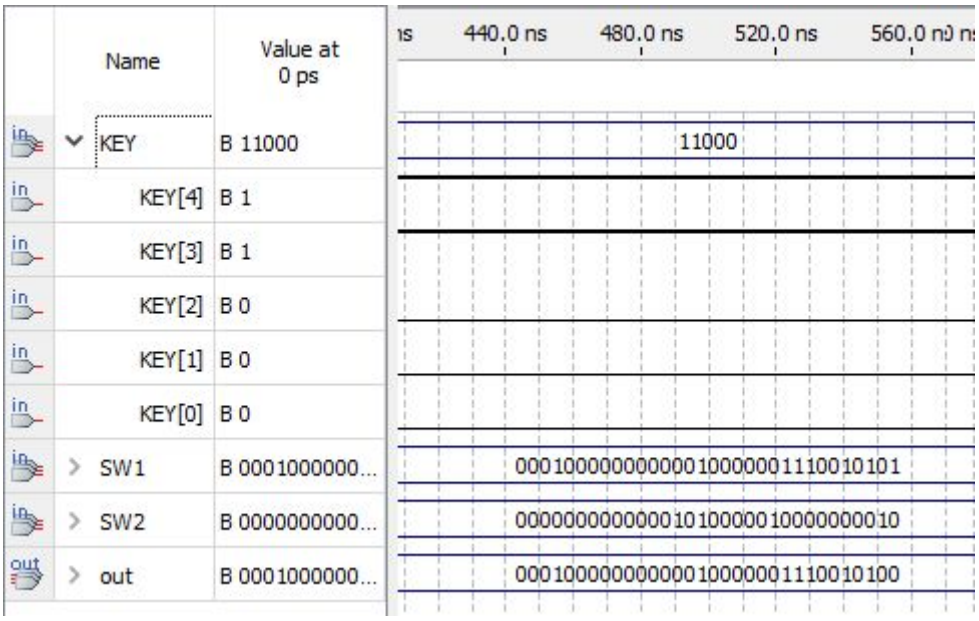


Figura 23. Operação 11000 com 268501909 e 328706 ambos em 32 Bits - DECREMENT SW1 e PASS ALL_OUT

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.