EEL480 - Sistemas Digitais

Laboratório 1 - Implementação de ALU com interface

Universidade Federal
do Rio de Janeiro

- Henrique Chaves - DRE 119025571
  - henriquechaves@poli.ufrj.br
- Pedro Boechat - DRE 119065050
  - pedroboechat@poli.ufrj.br

Rio de Janeiro - RJ
07 de Janeiro de 2022 (2021.2)

Conteúdo

1 Introdução

ALU, sigla para *Arithmetic Logic Unit* é um circuito digital responsável por performar diversas operações aritméticas e lógicas. Para performar a operação, uma ALU recebe como entrada operandos e um código indicando qual operação será realizada (soma, subtração, multiplicação, AND, OR, etc).

A ALU pode ser implementada utilizando uma FPGA (*Field Programmable Gate Arrays*), um dispositivo lógico programável que suporta a implementação de circuitos digitais. Existem diferentes FPGAs no mercado, mas em geral as FPGAs podem ter switches, botões, LEDs e displays (7 segmentos ou LCD). Em sua essência, uma FPGA precisa ter portas lógicas e elementos de memória organizados em formato matricial.

A linguagem de descrição VHDL (*Very High Speed Integrated Circuit Description Language*) permite descrever o comportamento e a estrutura da FPGA através de código. Com ela, podemos programar uma ALU para ser executada na FPGA.

É possível utilizar um simulador, como o Quartus II Lite, para testar o código VHDL e garantir que o circuito sintetizado está se comportando conforme o esperado antes de sintetizar o código para uma FPGA de verdade.

Ter uma FPGA pode ser custoso, por isso existem plataformas on-line onde é possível sintetizar códigos em FPGAs a distância, como no caso da LabsLand. Nessa plataforma, é disponibilizada uma placa FPGA do modelo *Altera Terasic DE-2115* que executa o código implementado e possibilita controlar remotamente a interface de controle do dispositivo. O usuário ainda consegue ver o comportamento dos elementos visuais da placa, como LEDs e displays, pois há uma câmera filmando a mesma.

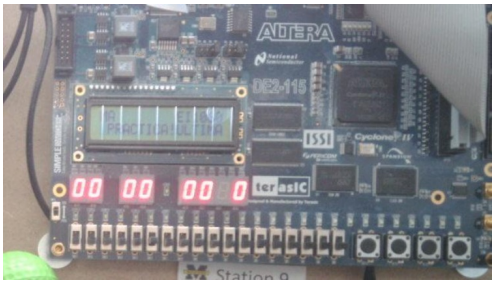

Figura 1: FPGA disponibilizada na plataforma LabsLand

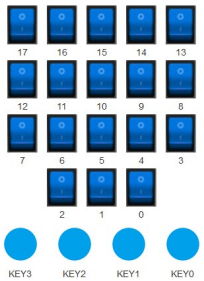

Figura 2: Interface de controle da FPGA na plataforma LabsLand

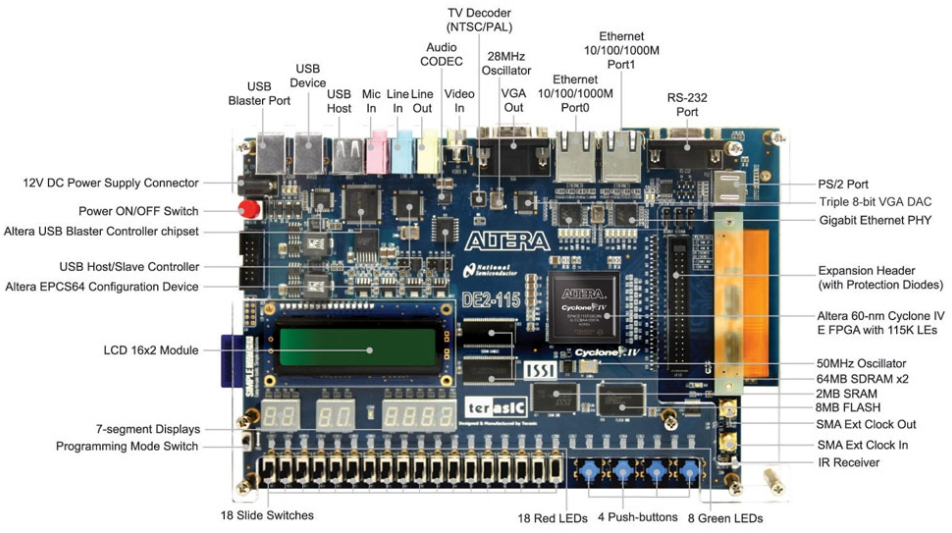

Figura 3: Descrição da placa FPGA Altera Terasic DE-2115

2 Objetivos

Os objetivos desse trabalho são: Desenvolver uma ALU que opere sobre dois números de 4 bits e realize um total de oito operações, e produzir um sistema de interface com o usuário para o teste da ALU.

Além disso, o sistema deverá realizar a carga dos dois operando simultaneamente por meio das chaves da placa. A cada 1 segundo o sistema deverá alterar a operação corrente, exibindo o valor destes operandos e o resultado da operação correspondente no display de 7 segmentos da FPGA.

3 Descrição da Implementação

3.1 Design do sistema

Seguindo os requisitos do projeto, o sistema criado para a experiência 1 opera em dois estados, SET e RESET. Para iniciá-lo é necessário o colocar em estado RESET, pressionando o botão KEY0 (porta V_BT(0)), para assim selecionar os números A e B, de 4 bits cada, com os switches 7, 6, 5 e 4 (porta V_SW(7 DOWNTO 4)) para A e 3, 2, 1 e 0 (porta V_SW(3 DOWNTO 0) para B. Os números serão mostrados instantaneamente em representação decimal nos displays de 7 segmentos (portas G_HEX7 e G_HEX6 para A e G_HEX5 e G_HEX4 para B) e em binário nos 8 primeiros LEDs vermelhos da esquerda para a direita (porta G_LEDR(17 DOWNTO 14) para A e G_LEDR(13 DOWNTO 10) para B). Não há até então nenhuma operação entre os valores, sendo o valor de resultado constante em 0 nos displays de 7 segmentos (porta G_HEX3 e G_HEX2) e nos últimos quatro LEDs vermelhos à direita (porta G_LEDR(3 DOWNTO 0). Após concluída a seleção de A e B, se pressiona o botão KEY1 (porta V_BT(1)), para mudar o estado do sistema para SET. Foi implementado um sistema de clock na ALU para controlar as operações realizadas, que reduz o clock de entrada do sistema (porta G_CLOCK_50) de 50 Mhz para aproximadamente 1 Hz. Dessa forma, a cada 1 segundo a operação muda entre as 8 possíveis. A operação atual é exibida em representação decimal no display de 7 segmentos (porta G_HEX0) e nos 3 últimos LEDs verdes à direita (porta G_LEDG(2 DOWNTO 0)). No estado SET, o resultado da operação atual para A e B passam a ser exibidos nos indicadores de resultado. Para alterar os valores de A ou B, é necessário voltar ao estado RESET.

Elemento	**Código**	**Descrição**
Switch 0	V_SW(0)	Controlar o bit 0 do operando B
Switch 1	V_SW(1)	Controlar o bit 1 do operando B
Switch 2	V_SW(2)	Controlar o bit 2 do operando B
Switch 3	V_SW(3)	Controlar o bit 3 do operando B
Switch 4	V_SW(4)	Controlar o bit 0 do operando A
Switch 5	V_SW(5)	Controlar o bit 1 do operando A
Switch 6	V_SW(6)	Controlar o bit 2 do operando A
Switch 7	V_SW(7)	Controlar o bit 3 do operando A
KEY0	V_BT(0)	Botão RESET
KEY1	V_BT(1)	Botão SET

Tabela 1: Elementos de interface de controle da FPGA na LabsLand

Código	Descrição
G_LEDR(0)	Bit 0 do resultado da operação
G_LEDR(1)	Bit 1 do resultado da operação
G_LEDR(2)	Bit 2 do resultado da operação
G_LEDR(3)	Bit 3 do resultado da operação
G_LEDR(10)	Bit 0 do operando B
G_LEDR(11)	Bit 1 do operando B
G_LEDR(12)	Bit 2 do operando B
G_LEDR(13)	Bit 3 do do operando B
G_LEDR(14)	Bit 0 do operando A
G_LEDR(15)	Bit 1 do operando A
G_LEDR(16)	Bit 2 do operando A
G_LEDR(17)	Bit 3 do do operando A
G_LEDG(0)	Bit 0 do operador
G_LEDG(1)	Bit 1 do operador
G_LEDG(2)	Bit 2 do operador

Tabela 2: Elementos visuais de controle da FPGA na LabsLand

3.2 Programação do sistema

O sistema é dividido em 6 arquivos VHDL. São esses:

- `FA.vhd`
- `FS.vhd`
- `MULT.vhd`
- `ALU.vhd`
- `DEC7SEG.vhd`
- `LAB1.vhd`

Foram utilizados 3 módulos da biblioteca IEEE. São esses:

- IEEE.NUMERIC_STD.ALL
  Para usar operadores lógicos.
- IEEE.STD_LOGIC_1164.ALL
  Para usar tipagens de lógica padrão.
- IEEE.STD_LOGIC_UNSIGNED.ALL
  Para realizar operações aritiméticas com vetores lógicos.

3.2.1 FA.vhd

Nesse arquivo é implementada um Full Adder. É definida a entidade "FA" com as seguintes portas:

- A (IN) : STD_LOGIC
  Entrada do número A.
- B (IN) : STD_LOGIC
  Entrada do número B.
- CIN (IN) : STD_LOGIC
  Entrada do carry in.
- S (OUT): STD_LOGIC
  Saída do resultado da soma.
- COUT (OUT): STD_LOGIC
  Saída do carry out da soma.

Em seguida, é definida sua arquitetura "BEHAVIORAL". Nessa simples implementação do componente, é retornado o resultado da soma na porta S e o carry out da soma na porta COUT.

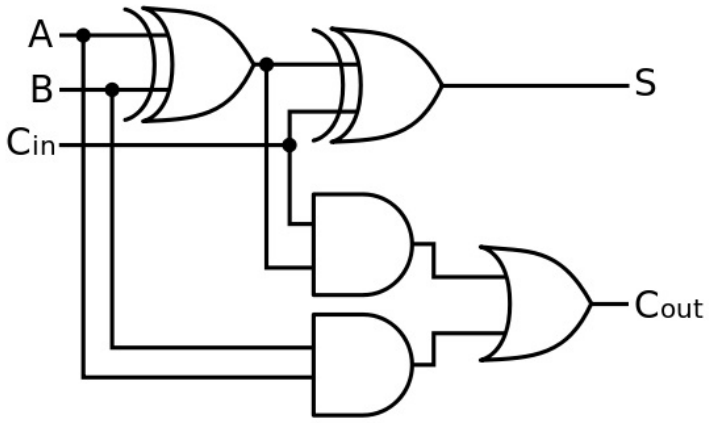

Figura 4: Circuito do Full Adder

3.2.2 FS.vhd

Nesse arquivo é implementada um Full Subtractor. É definida a entidade "FS" com as seguintes portas:

- A (IN) : STD_LOGIC
  Entrada do número A.
- B (IN) : STD_LOGIC
  Entrada do número B.
- BIN (IN) : STD_LOGIC
  Entrada do borrow in.
- D (OUT): STD_LOGIC
  Saída do resultado da diferença.
- BOUT (OUT): STD_LOGIC
  Saída do borrow out da diferença.

Em seguida, é definida sua arquitetura "BEHAVIORAL". Nessa simples implementação do componente, é retornado o resultado da diferença na porta D e o borrow out da soma na porta BOUT.

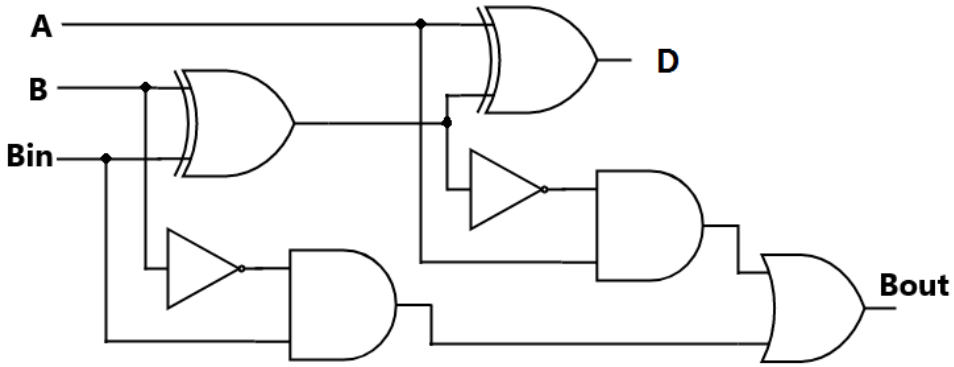

Figura 5: Circuito do Full Subtractor

3.2.3 MULT.vhd

Nesse arquivo é implementada um 4x4 Multiplier. É definida a entidade "MULT" com as seguintes portas:

- A (IN) : STD_LOGIC_VECTOR(3 DOWNTO 0)
  Entrada do número A.
- B (IN) : STD_LOGIC_VECTOR(3 DOWNTO 0)
  Entrada do número B.
- POUT (OUT): STD_LOGIC_VECTOR(3 DOWNTO 0)
  Saída do resultado da multiplicação.

Em seguida, é definida sua arquitetura "BEHAVIORAL". São definidos 6 sinais para auxiliar a implementação do circuito, sendo 4 do tipo STD_LOGIC e 2 do tipo STD_LOGIC_VECTOR. O componente recebe os dois vetores lógicos contendo os números A e B e retorna o resultado do produto na porta POUT.

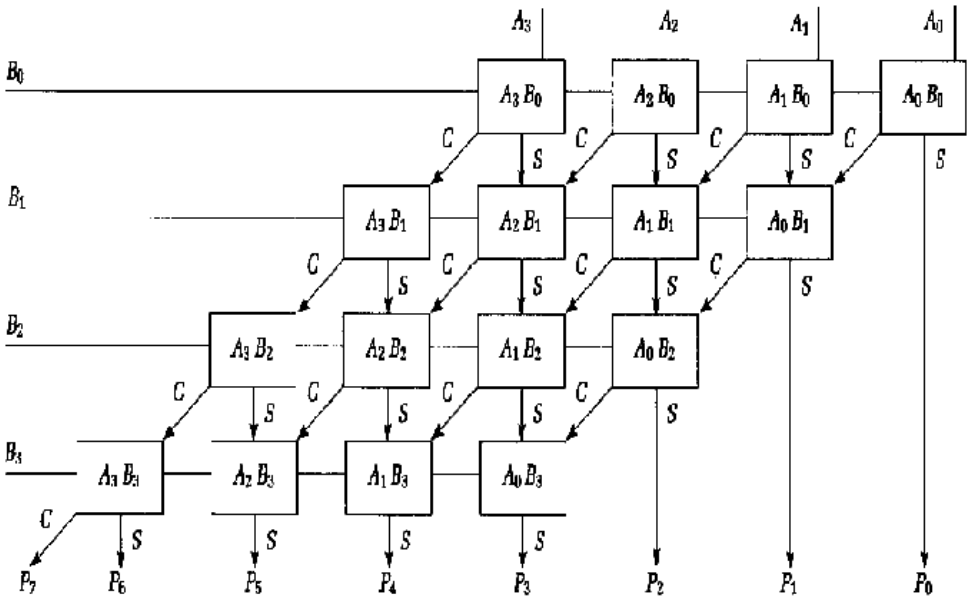

Figura 6: Circuito do 4x4 Multiplier

3.2.4 ALU.vhd

Nesse arquivo é implementada uma ALU. É definida a entidade "ALU" com as seguintes portas:

- A (IN) : STD_LOGIC_VECTOR(3 DOWNTO 0)
  Entrada do número A.
- B (IN) : STD_LOGIC_VECTOR(3 DOWNTO 0)
  Entrada do número B.
- CLK (IN) : STD_LOGIC
  Entrada do sinal de clock.
- OPT (OUT): STD_LOGIC_VECTOR(2 DOWNTO 0)
  Saída da opção atual de operação.
- OUTPUT (OUT): STD_LOGIC_VECTOR(3 DOWNTO 0)
  Saída do resultado da ALU.

Em seguida, é definida sua arquitetura "BEHAVIORAL". São instanciados os componentes do Full Adder, do Full Subtractor e do 4x4 Multiplier. Também são definidos 14 sinais, sendo 2 vetores logicos unsigned para a redução do clock, 5 STD_LOGIC para o Full Adder, 5 STD_LOGIC para o Full Subtractor, 1 STD_LOGIC_VECTOR para o 4x4 Multiplier e 1 STD_LOGIC_VECTOR para o resultado da ALU. A arquitetura começa com o *port mapping* de 4 FAs, 4 FSs e 1 MULT. Para ambos os FAs e FSs, cada um recebe um bit do número A, e são conectador pelos seus carries e borrows, de forma que se obtem o resultado completo da adição e da subtração em seus respectivos sinais. O MULT recebe os vetores lógicos completos que representam os números A e B e retorna o vetor lógico do resultado da multiplicação em um sinal dedicado a ele. Após isso, é iniciado um processo, que começa pelo redutor de clock. Como o sinal do clock do LabsLand é de 50MHz e queremos que as operações da ALU mudem apenas a cada 1 segundo, a cada *rising edge* do clock, se adiciona 1 à um sinal auxiliar e, quando o contador atinge o valor de 50331648, se aumenta o sinal da opção de operação em 1 e o contador é resetado, reduzindo o clock para 0.993Hz. Depois da redução do clock, o sinal da opção de operação passa por um *case*, que verifica o seu valor e atribui ao sinal do resultado o valor da operação atual. Por fim, é retornada a opção da operação atual na porta OPT e o resultado da operação na porta OUTPUT.

3.2.5 DEC7SEG.vhd

Nesse arquivo é implementado um decodificador para o display de 7 segmentos, em BCD. É definida a entidade "DEC7SEG" com as seguintes portas:

- INPUT (IN) : STD_LOGIC_VECTOR(3 DOWNTO 0)

  Entrada do número a ser decodificado para o display de 7 segmentos, em BCD.

- OUTPUT (IN) : STD_LOGIC_VECTOR(13 DOWNTO 0)

  Saída do valor decodificado.

Em seguida, é definida sua arquitetura "BEHAVIORAL". Nela, o valor de entrada do número passa por um *case* e em seguida é retornado valor decodificado para o display de 7 segmentos, em BCD, na porta OUTPUT.

3.2.6 LAB1.vhd

Nesse arquivo é implementada a *top-level entity*, que é a interface do sistema com o LabsLand. É definida a entidade "LAB1" com as seguintes portas:

- G_CLOCK_5 (IN) : STD_LOGIC

  Entrada do sinal de clock.

- V_SW (IN) : STD_LOGIC_VECTOR(7 DOWNTO 0)

  Entrada dos switches responsáveis pelos números A e B.

- V_BT (IN) : STD_LOGIC_VECTOR(1 DOWNTO 0)

  Entrada dos botões responsáveis por alterar os estados de SET e RESET do sistema.

- G_LEDR (OUT) : STD_LOGIC_VECTOR(17 DOWNTO 0)

  Saída dos LEDs vermelhos, responsáveis por representar a forma binária dos números A e B e do resultado.

- G_LEDG (OUT) : STD_LOGIC_VECTOR(2 DOWNTO 0)

  Saída dos LEDs verdes, responsáveis por representar a forma binária da opção de operação.

- G_HEX7 (OUT) : STD_LOGIC_VECTOR(6 DOWNTO 0)

  Saída do decodificador para display de 7 segmentos responsável por representar a dezena da representação decimal do número A.

- G_HEX6 (OUT) : STD_LOGIC_VECTOR(6 DOWNTO 0)

  Saída do decodificador para display de 7 segmentos responsável por representar a unidade da representação decimal do número A.

- G_HEX5 (OUT) : STD_LOGIC_VECTOR(6 DOWNTO 0)

  Saída do decodificador para display de 7 segmentos responsável por representar a dezena da representação decimal do número B.

- G_HEX4 (OUT) : STD_LOGIC_VECTOR(6 DOWNTO 0)

  Saída do decodificador para display de 7 segmentos responsável por representar a unidade da representação decimal do número B.

- G_HEX3 (OUT) : STD_LOGIC_VECTOR(6 DOWNTO 0)

  Saída do decodificador para display de 7 segmentos responsável por representar a dezena da representação decimal do resultado da operação.

- G_HEX2 (OUT) : STD_LOGIC_VECTOR(6 DOWNTO 0)

  Saída do decodificador para display de 7 segmentos responsável por representar a unidade da representação decimal do resultado da operação.

- G_HEX0 (OUT) : STD_LOGIC_VECTOR(6 DOWNTO 0)

  Saída do decodificador para display de 7 segmentos responsável por representar a opção de operação de forma decimal.

Em seguida, é definida sua arquitetura "BEHAVIORAL". São instanciados os componentes da ALU e do decodificador para o display de 7 segmentos. Também são definidos 11 sinais, sendo 10 vetores logicos para o resultado da ALU, a opção de operação atual da ALU e para o *input* e *output* dos valores no decodificador para display de 7 segmentos e 1 STD_LOGIC_VECTOR para guardar o estado atual do sistema. A arquitetura começa com o *port mapping* de 1 ALU e 4 DEC7SEGs. Os DEC7SEGs serão usados para os sinais dos números A e B, do resultado da operação e da opção de operação. Após isso, é iniciado um processo, que começa verificando se algum dos botões de SET ou RESET foi pressionado. Caso tenha, ele define o sinal de estado para o estado correto. Depois disso é verificado o estado do sistema. Caso esteja no estado RESET, os valores dos sinais dos números A e B são atualizados, o valor do sinal do resultado da operação é definido para 0 e os LEDs vermelhos e displays de 7 segmentos são atualizados com os valores atuais dos sinais. Caso esteja no estado SET, apenas o valor do sinal do resultado da operação é atualizado, para o valor do resultado da ALU e apenas os LEDs vermelhos e displays de 7 segmentos do resultado da operação são atualizados. Por fim, independente do estado do sistema, é atualizado o valor dos LEDs verdes e do display de 7 segmentos que representam a opção de operação da ALU.

4 Resultados

Com o objetivo de validar o resultado de todas as funções implementadas na ALU, foram feitas simulações no software *Quartus Prime 20.1*, gerando as respectivas funções de onda para cada elemento.

Pequenas alterações no código original foram necessárias, pois a simulação trabalha com intervalos de tempo pequenos (até $100\mu s$), e a frequência do CLOCK estava definida como 1Hz. Portanto, aumentamos essa frequência para 100MHz, para conseguir se ter várias oscilações no intervalo de $1\mu s$.

Forma realizadas duas simulações, setando um par de operandos (A, B). Na primeira simulação, A = 4 (0100) e B = 2 (0010). Na segunda simulação, A = 5 (0101) e B = 3 (0011). Os resultados da primeira simulação podem ser observados na imagem 7 e as descrições na tabela 3. Enquanto os resultados da segunda simulação podem ser consultados na imagem 8 e na tabela 4

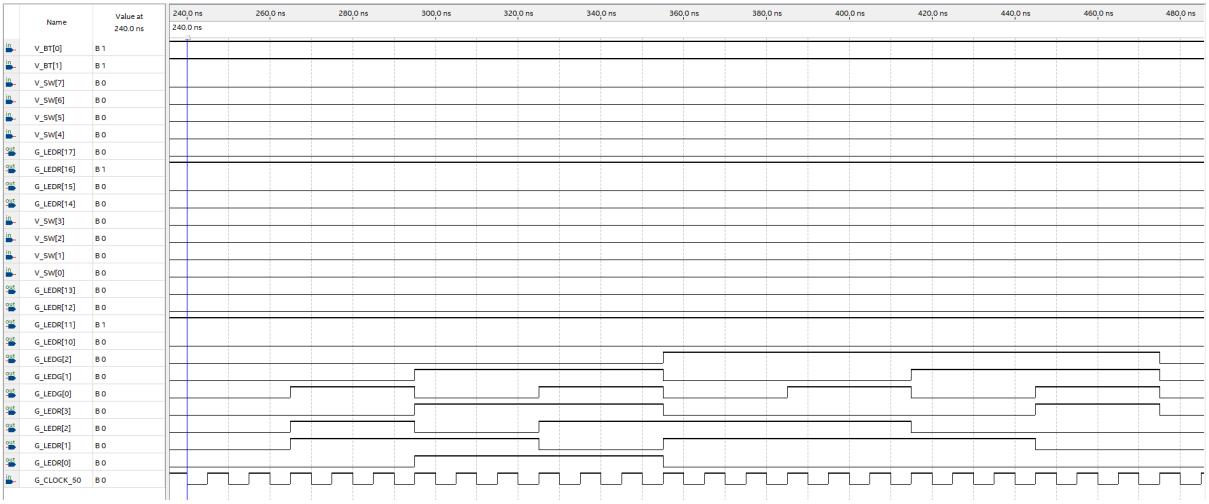

Figura 7: Simulação com A = 4 (0100) e B = 2 (0010)

Operação (descrição)	**Operação (binário)**	**Resultado esperado (binário)**	**Resultado obtido (binário)**
AND	000	0000	0000
OR	001	0110	0110
NOT A	010	1011	1011
NOT B	011	1101	1101
XOR	100	0110	0110
SOMA	101	0110	0110
SUBTRAÇÃO	110	0010	0010
MULTIPLICAÇÃO	111	1000	1000

Tabela 3: Resultados esperados e obtidos na primeira simulação, onde A = 4 (0100) e B = 2 (0010)

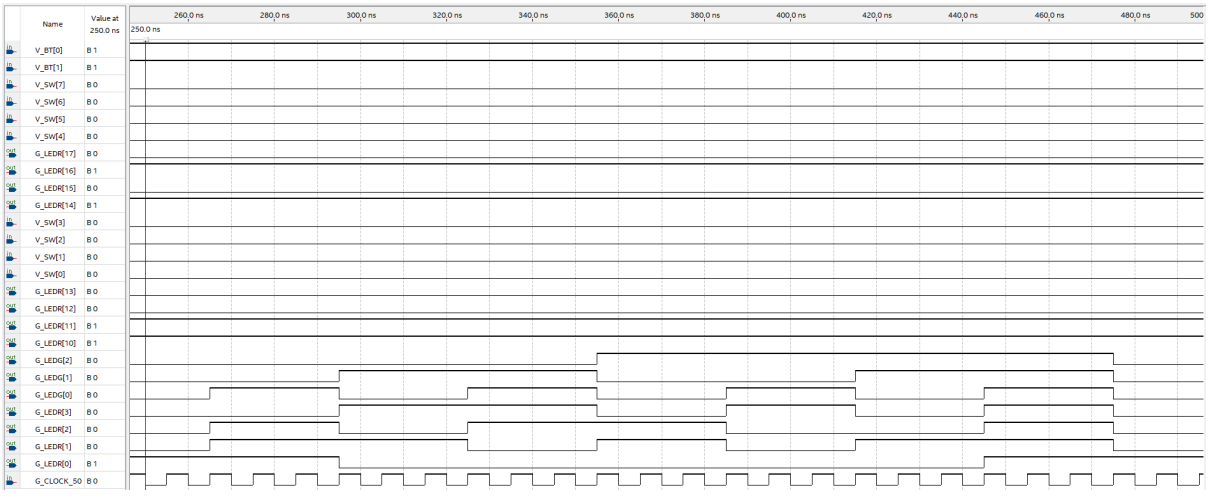

Figura 8: Simulação com A = 5 (0101) e B = 3 (0011)

Operação (descrição)	Operação (binário)	Resultado esperado (binário)	Resultado obtido (binário)
AND	000	0001	0001
OR	001	0111	0111
NOT A	010	1010	1010
NOT B	011	1100	1100
XOR	100	0110	0110
SOMA	101	1000	1000
SUBTRAÇÃO	110	0010	0010
MULTIPLICAÇÃO	111	1111	1111

Tabela 4: Resultados esperados e obtidos na segunda simulação, onde A = 5 (0101) e B = 3 (0011)

Todos os resultados obtidos deram iguais aos esperados, evidenciando a funcionalidade adequada da ALU implementada. As descrições dos códigos na esquerda das imagens de simulação podem ser consultados nas tabelas 1 e 2.

Além das simulações, a ALU foi executada em uma FPGA pela plataforma LabsLand, onde também se confirmou o funcionamento adequado dos elementos visuais, como no caso dos displays de 7 segmentos (dois para o operando A, dois para o operando B, dois para o resultado e um para o operador), e também os LEDs vermelhos e verdes, como na simulação.

5 Conclusão

A partir do trabalho realizado, foi adquirido um conhecimento essencial sobre circuitos digitais e programação dos mesmos, através do uso da linguagem VHDL para programar FPGA, implementando uma ALU.

Todas as funções implementadas na ALU desempenharam corretamente, não tendo nenhum erro de resultado ou operação durante os testes e simulações. Portanto, conclui-se que todas as funcionalidades da ALU, além das interfaces, tiveram implementações corretas pelos alunos.

O código fonte do projeto se encontra em `https://github.com/henchaves/eel480-sd`.

Lista de Tabelas

Lista de Figuras