UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO

RELATÓRIO Somador Vetorial

GRUPO: Nicholas Costa (122051214)
Fábio Sales (122088960)
Beatriz Lima (123254085)
Filipe A. Barbosa(125100347)
Daniely Soares (125085115)

Rio de Janeiro
2026

1. Introdução

Com o aumento da quantidade de dados processados atualmente, a computação escalar tradicional, baseada no modelo SISD (Single Instruction, Single Data), passou a apresentar limitações de desempenho, principalmente em aplicações que realizam muitas operações aritméticas simples. Como alternativa, arquiteturas modernas utilizam o modelo SIMD (Single Instruction, Multiple Data), no qual uma única instrução é aplicada a vários dados ao mesmo tempo, aproveitando melhor o hardware disponível sem depender apenas do aumento da frequência de clock.

Nos processadores x86-64, esse conceito é implementado por meio da extensão AVX-512, que permite realizar operações paralelas sobre dados empacotados em registradores de 512 bits. Inspirado nesse modelo — e também nas extensões vetoriais do RISC-V — este trabalho apresenta a implementação de um somador/subtrator vetorial de 32 bits, capaz de operar com números inteiros de 4, 8, 16 e 32 bits.

O foco não é reproduzir todo o paralelismo físico do AVX-512, mas sim mostrar, de forma simplificada, como um mesmo circuito aritmético pode ser reutilizado para diferentes tamanhos de dados, exemplificando na prática os princípios básicos da computação vetorial.

2. Metodologia

A metodologia adotada neste trabalho baseia-se nos princípios da Arquitetura Vetorial, conforme apresentados na literatura clássica de Patterson & Hennessy. Diferente da computação escalar tradicional, onde um único valor é processado por vez, a abordagem vetorial trata conjuntos de dados como uma única entidade lógica, permitindo um melhor aproveitamento do hardware disponível.

Do ponto de vista conceitual, o projeto explora principalmente duas ideias. A primeira é o Strip Mining, técnica utilizada para adaptar algoritmos vetoriais a registradores de tamanho limitado, garantindo que o processamento possa ser realizado mesmo quando o tamanho dos dados é maior que a largura física do hardware.

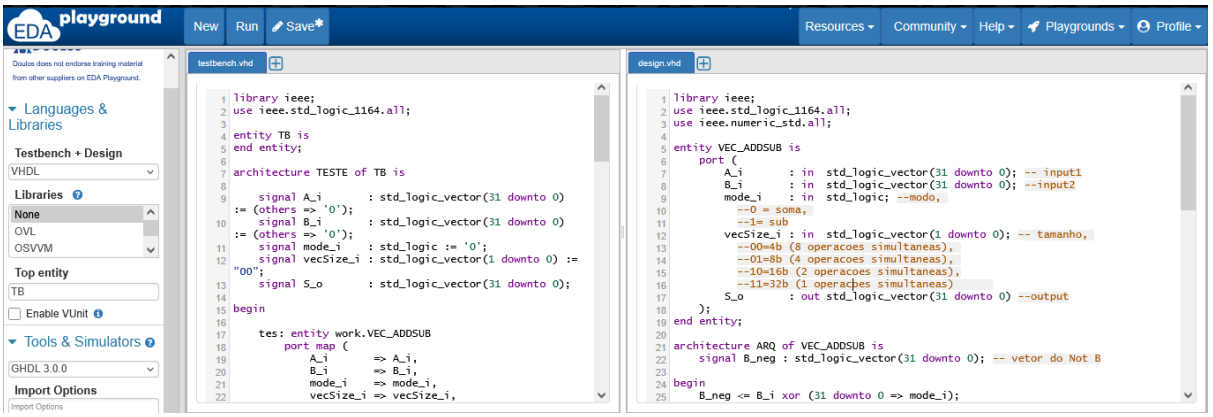
A segunda é o Subword Parallelism, no qual uma palavra de 32 bits é dividida em sub-elementos menores, possibilitando que um único somador execute múltiplas operações de menor precisão de forma independente.

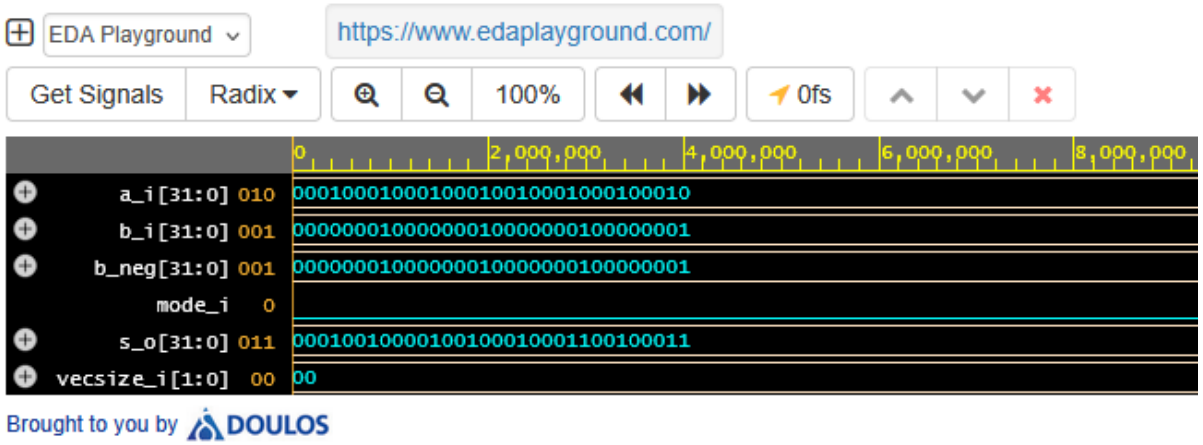
Esses conceitos foram aplicados no nível de hardware por meio de uma descrição em VHDL, onde o comportamento vetorial é representado pela segmentação lógica da palavra e pelo controle explícito do sinal de carry entre os sub-blocos. Dessa forma, mesmo utilizando um circuito simples, foi possível exemplificar o funcionamento básico de operações vetoriais e sua relação com arquiteturas modernas.

3. Implementação Prática

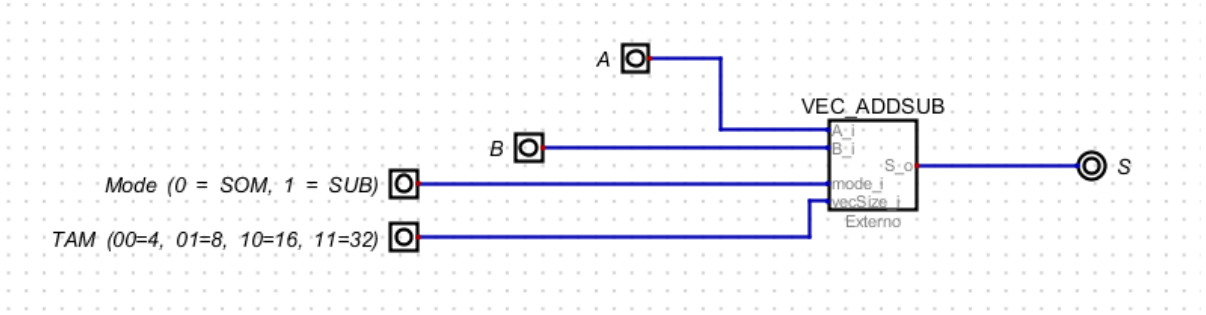
A implementação iniciou-se com a escrita e prototipagem do código em VHDL utilizando a ferramenta EDAPlayground. Essa escolha foi motivada principalmente pela possibilidade de observar o comportamento interno do circuito ao longo do tempo, permitindo a análise detalhada, bit a bit, de sinais intermediários como o carry e os operandos modificados durante a operação.

Esse nível de visualização foi essencial para compreender como a propagação do carry se comporta quando a palavra de 32 bits é dividida em sub-blocos menores, algo que não se apresenta de forma tão explícita no software Digital.





Após a consolidação do código, o circuito foi implementado no Digital, atendendo à exigência da atividade de disponibilizar um ambiente de testes sem o uso de testbenches tradicionais. Essa etapa permitiu a interação direta com os sinais de entrada e facilitou a demonstração do funcionamento do módulo.



Externo

Básico | Opções

Rótulo: VEC_ADDSUB

Largura: 5

Entradas: A_i:32,B_i:32,mode_i,vecSize_i:2

Saídas: S_o:32

Código de programa:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity VEC_ADDSUB is
6      port (
7          A_i       : in  std_logic_vector(31 downto 0); -- input1
8          B_i       : in  std_logic_vector(31 downto 0); --input2
9          mode_i    : in  std_logic; --modo,
10                   --0 = soma,
11           --1= sub
12         vecSize_i : in  std_logic_vector(1 downto 0); -- tamanho,
13                   --00=4b (8 operacoes simultaneas),
14                   --01=8b (4 operacoes simultaneas),
15                   --10=16b (2 operacoes simultaneas),
16                   --11=32b (1 operacoes simultaneas)
17         S_o       : out std_logic_vector(31 downto 0) --output
18     );
19 end entity;
20
21 architecture ARQ of VEC_ADDSUB is
22     signal B_neg : std_logic_vector(31 downto 0); -- vetor do Not B
23
24 begin
25     B_neg <= B_i xor (31 downto 0 => mode_i);
26         -- se for subtracao, pegaremos o complemento de 2 do B_i:
27                 --(A - B = A + (Not B))
28
29 process (A_i, B_neg, mode_i, vecSize_i)
30     variable c   : std_logic;
```

Aplicação: GHDL Verificar

3.1 Funcionamento do Código VHDL

O módulo implementa uma unidade vetorial de soma e subtração baseada em um somador completo bit a bit, percorrendo sequencialmente os 32 bits por meio de um laço for. Apesar da descrição sequencial, a lógica implementada representa corretamente o comportamento de uma operação vetorial segmentada.

A subtração é realizada por meio do complemento de dois, utilizando a identidade: $A-B=A+(\sim B+1)$

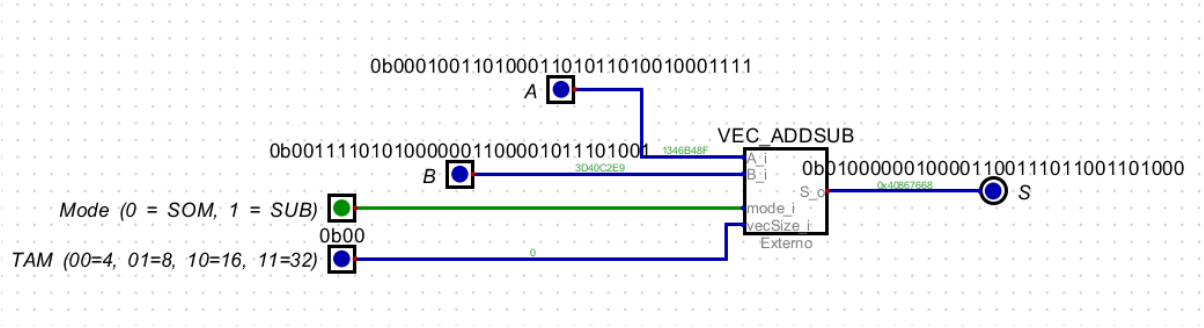
O vetor intermediário B_neg é obtido aplicando uma operação XOR entre B_i e o sinal mode_i. Quando mode_i = 1, todos os bits de B_i são invertidos. O valor +1 necessário para o complemento de dois é inserido inicializando o carry com o próprio mode_i.

O aspecto central do projeto está no controle do carry. O carry é reinicializado sempre que um novo subvetor se inicia, conforme o valor de vecSize_i. Essa estratégia impede que o carry se propague entre blocos distintos, garantindo que cada subpalavra seja processada de forma independente, exatamente como ocorre em instruções vetoriais AVX que operam sobre múltiplos elementos empacotados.

Assim, embora o hardware não execute operações em paralelo fisicamente, o circuito modela corretamente a semântica de uma soma/subtração vetorial, respeitando os limites de cada elemento do vetor.

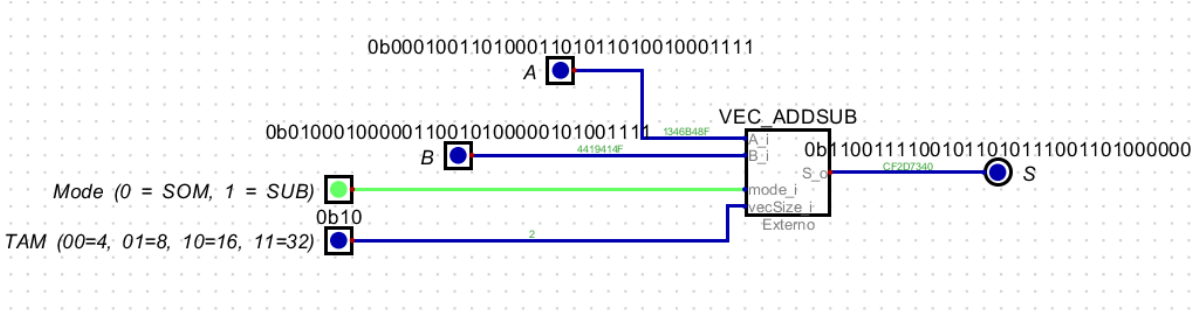
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity VEC_ADDSUB is
   port (
      A_i      : in  std_logic_vector(31 downto 0); -- input1
      B_i      : in  std_logic_vector(31 downto 0); --input2
      mode_i    : in  std_logic; --modo,
        --0 = soma,
        --1= sub
           vecSize_i : in   std_logic_vector(1 downto 0); --
tamanho,
        --00=4b (8 operacoes simultaneas),
        --01=8b (4 operacoes simultaneas),
        --10=16b (2 operacoes simultaneas),
        --11=32b (1 operacoes simultaneas)
      S_o      : out std_logic_vector(31 downto 0) --output
   );
end entity;

architecture ARQ of VEC_ADDSUB is
    signal B_neg : std_logic_vector(31 downto 0); -- vetor
do Not B

begin
   B_neg <= B_i xor (31 downto 0 => mode_i);
        -- se for subtracao, pegaremos o complemento de 2
do B_i:
        --(A - B = A + (Not B))

process (A_i, B_neg, mode_i, vecSize_i)
   variable c   : std_logic;
begin
   -- Inicialização obrigatória do carry
   c := mode_i;

   for i in 0 to 31 loop

      if (i = 0) or
        (vecSize_i = "00" and i mod 4 = 0) or
        (vecSize_i = "01" and i mod 8 = 0) or
        (vecSize_i = "10" and i mod 16 = 0) or
        (vecSize_i = "11" and i = 0) then
         c := mode_i;
      end if;

      -- Soma do bit
      S_o(i) <= A_i(i) xor B_neg(i) xor c;

      c := (A_i(i) and B_neg(i)) or
           (A_i(i) and c) or
           (B_neg(i) and c);
   end loop;
end process;

end ARQ;
```

4. Resultados Experimentais



- (vecSize = 00 (4 bits), mode = 0 (Soma))

Subvetor	A (4 Bits)	B (4 bits)	Soma (4 bits)
1	0001	0011	0100
2	0011	1101	0000
3	0100	0100	1000
4	0110	0001	0111
5	1011	0100	1111
6	0100	0010	0110
7	1000	1110	0110
8	1111	1001	1000



- (vecSize = 10 (16 bits), mode = 1(Subtração))

Subvetor	A (16 Bits)	B (16 bits)	Subtração(16 bits)
1	0001001101000110	0100010000011001	1100111100101101
2	1011010010001111	0100000101001111	0111001101000000

5. Conclusão

Este trabalho apresentou o projeto de um somador/subtrator vetorial de 32 bits, capaz de operar sobre subvetores de 4, 8, 16 e 32 bits, alinhado aos princípios das extensões vetoriais modernas, como o AVX. A implementação em VHDL demonstra que, mesmo com uma descrição sequencial, é possível representar corretamente o comportamento lógico de operações SIMD por meio do controle adequado do carry e da segmentação dos dados.

Conclui-se que a arquitetura vetorial, mesmo em versões simplificadas, é fundamental para a compreensão de sistemas orientados a desempenho e constitui uma base sólida para arquiteturas de alto throughput utilizadas em aplicações modernas de processamento de dados.