A logo with blue and green squares

Description automatically generatedUniversidade de Brasília – UNB

Curso: Engenharia de Redes de Comunicação  
Disciplina: Laboratório de Sistemas Digitais   
Turma: 08

Relatório da Disciplina Laboratório de Sistemas Digitais  
Tema: Experimento 05 – Operações Aritméticas e Testbenches

Aluno: Pedro Henrique Dias Avelar  
Matrícula: 241037112  
Professor: Eduardo Paiva

# Lista de Referências

[Figura 1:Representação gráfica do cascateamento de somadores completos - Fonte: Material de Roteiro para o Experimento 5, Depto de Engenharia Elétrica da UNB 4](#_Toc187189292)

[Figura 2: 0000+0000=00000; 0001+0001=00010; 0010+0010=00100; 0011+0011=00110 5](#_Toc187189293)

[Figura 3:0100+0100=01000; 0101+0101=01010; 0110+0110=01100; 0111+0111=01110 6](#_Toc187189294)

[Figura 4:1000+1000=10000; 1001+1001=10010; 1010+1010=10100; 1011+1011=10110 6](#_Toc187189295)

[Figura 5:1100+1100=11000; 1101+1101=11010; 1110+1110=11100; 1111+1111=11110 6](#_Toc187189296)

[Figura 6: Resultados: 01111; 10000; 10001; 10010; 10011; 10100; 10101; 10110 8](#_Toc187189297)

[Figura 7:Resultados: 10111; 11000; 11001; 11010; 11011; 11100; 11101; 11110 8](#_Toc187189298)

[Figura 8: Resultado da simulação para o top module 11](#_Toc187189299)

[Figura 9: Log da simulação 11](#_Toc187189300)

[Tabela 1: Tabela verdade para o somador completo 3](#_Toc187189302)

[Tabela 2: Tabela verdade para a simulação da atividade 01 7](#_Toc187189303)

[Tabela 3: Tabela verdade para a simulação da atividade 02 8](#_Toc187189304)

[Código 1: Somador completo do experimento 02 4](#_Toc187189312)

[Código 2: Ligação entre os 4 somadores completos 5](#_Toc187189313)

[Código 3:Implantação do somador de palavras de 4 bits com o uso da biblioteca STD\_LOGIC\_ARITH 7](#_Toc187189314)

[Código 4: Código para o testbench 9](#_Toc187189315)

[Código 5: Código para o top module 10](#_Toc187189316)

# Introdução

O presente experimento tem os seguintes objetivos:

* Implementar um somador de palavras binárias usando somadores completos em cascata.
* Usar o pacote STD\_LOGIC\_ARITH
* Desenvolver um *testbench* para simulação de circuitos em VHDL

# Atividade 01

Escreva em VHDL e simule no ModelSim um somador de palavras de 4 bits usando apenas somadores completos (desenvolvidos no experimento 2). A nova entidade deve ter como entrada dois vetores A e B (de 4 bits cada) e um vetor de saída S (de 5 bits). Inclua os somadores completos desenvolvidos anteriormente como componentes. Sua arquitetura principal deve fazer **apenas** conexões entre os elementos.

O somador completo é um circuito que recebe 3 bits de entrada – os dois primeiros bits iremos nomear como A e B e são os bits que serão somados e o terceiro bit é o “vem um” ou, em inglês, carry-in . A saída consiste nos bits S e no “vai um” ou carry-out . A lógica para montar a tabela verdade é simples: Se os 3 bits de entrada forem zero, a soma S e o “vai um” serão zero. Se apenas um dos bits de entrada for 1, então a soma será um e o “vai um” será zero. Se dois dos bits de entrada for 1, temos que, na base binária, 1+1 = 10, ou seja, a soma S será zero e o “vai um” será um. Se os três bits forem 1, então temos que 1+1=10; 10+1 = 11; ou seja, tanto a soma quanto o “vai um” serão 1 neste caso. Assim, temos então a seguinte tabela verdade:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B |  | S |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Tabela 1: Tabela verdade para o somador completo

Esta tabela verdade pode ser obtida com as equações do experimento 2:

Podemos usar em cascata N somadores completos para fazer a soma de palavras de N bits:

A diagram of a number of objects

Description automatically generated with medium confidence

Figura 1:Representação gráfica do cascateamento de somadores completos - Fonte: Material de Roteiro para o Experimento 5, Depto de Engenharia Elétrica da UNB

O somador de palavras de 4 bits foi implantado no ModelSim com os seguintes códigos:

Código 1: Somador completo do experimento 02

-- Experimento 02 - Questão 01

-- Aluno: Pedro Henrique Dias Avelar 241037112

-- Turma 08

-- Data: 02/11/2024

-- Funções lógicas do somador completo:

-- S = A xor B xor Cin

-- Cout = AB or ACin or BCin

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** SOMADOR\_COMPLETO **IS**

**PORT** **(**A**,**B**,**CIN**:** **IN** STD\_LOGIC**;** --ENTRADAS

S**,**COUT**:** **OUT** STD\_LOGIC**);** --SAÍDAS

**END** SOMADOR\_COMPLETO**;**

**ARCHITECTURE** ARC\_SOMADOR\_COMPLETO **OF** SOMADOR\_COMPLETO **IS**

**BEGIN**

S **<=** A **XOR** B **XOR** CIN**;** --S = A xor B xor Cin

COUT **<=** **(**A **AND** B**)** **OR** **(**A **AND** CIN**)** **OR** **(**B **AND** CIN**);** --Cout = AB or ACin or BCin

**END** ARC\_SOMADOR\_COMPLETO**;**

-- Experimento 05 - Questão 01

-- Aluno: Pedro Henrique Dias Avelar 241037112

-- Turma 08

-- Data: 11/12/2024

--SOMADOR DE PALAVRAS DE 4 BITS

--ENTRADA: A E B (4 BITS)

--SAÍDA: S (5 BITS)

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** E05Q01 **IS**

**PORT** **(**A**,**B**:** **IN** STD\_LOGIC\_VECTOR **(**3 **DOWNTO** 0**);**

S**:** **OUT** STD\_LOGIC\_VECTOR **(**4 **DOWNTO** 0**));**

**END** E05Q01**;**

**ARCHITECTURE** ARC\_E05Q01 **OF** E05Q01 **IS**

--SOMADOR COMPLETO DO EXPERIMENTO 02 QUESTÃO 01

**COMPONENT** SOMADOR\_COMPLETO **is**

**PORT** **(**A**,**B**,**CIN**:** **IN** STD\_LOGIC**;** --ENTRADAS

S**,**COUT**:** **OUT** STD\_LOGIC**);** --SAÍDAS

**END** **COMPONENT;**

**SIGNAL** C1**,**C2**,**C3**:** STD\_LOGIC**;**

**BEGIN**

SC0**:** SOMADOR\_COMPLETO **PORT** **MAP** **(**A**(**0**),**B**(**0**),**'0'**,**S**(**0**),**C1**);**

SC1**:** SOMADOR\_COMPLETO **PORT** **MAP** **(**A**(**1**),**B**(**1**),**C1**,**S**(**1**),**C2**);**

SC2**:** SOMADOR\_COMPLETO **PORT** **MAP** **(**A**(**2**),**B**(**2**),**C2**,**S**(**2**),**C3**);**

SC3**:** SOMADOR\_COMPLETO **PORT** **MAP** **(**A**(**3**),**B**(**3**),**C3**,**S**(**3**),**S**(**4**));**

**END** ARC\_E05Q01**;**

Código 2: Ligação entre os 4 somadores completos

O primeiro somador assume o “carry-in” como zero; já o último somador assume seu “carry-out” como o último digito da soma.

Para simplificar a simulação, visto que na atividade 3 será realizado o teste para todas as combinações de valores dos vetores A e B, a mesma foi realizada considerando os vetores A e B iguais. Desta forma a tabela verdade apresentará apenas 16 resultados:

A screenshot of a computer program

Description automatically generated

Figura 2: 0000+0000=00000; 0001+0001=00010; 0010+0010=00100; 0011+0011=00110

A screenshot of a computer

Description automatically generated

Figura 3:0100+0100=01000; 0101+0101=01010; 0110+0110=01100; 0111+0111=01110

A screenshot of a computer program

Description automatically generated

Figura 4:1000+1000=10000; 1001+1001=10010; 1010+1010=10100; 1011+1011=10110

A screenshot of a computer program

Description automatically generated

Figura 5:1100+1100=11000; 1101+1101=11010; 1110+1110=11100; 1111+1111=11110

Montando então a tabela verdade então temos que:

|  |  |  |  |
| --- | --- | --- | --- |
| A/B | A/B (decimal) | S | S(decimal) |
| 0000 | 0 | 00000 | 0 |
| 0001 | 1 | 00010 | 2 |
| 0010 | 2 | 00100 | 4 |
| 0011 | 3 | 00110 | 6 |
| 0100 | 4 | 01000 | 8 |
| 0101 | 5 | 01010 | 10 |
| 0110 | 6 | 01100 | 12 |
| 0111 | 7 | 01110 | 14 |
| 1000 | 8 | 10000 | 16 |
| 1001 | 9 | 10010 | 18 |
| 1010 | 10 | 10100 | 20 |
| 1011 | 11 | 10110 | 22 |
| 1100 | 12 | 11000 | 24 |
| 1101 | 13 | 11010 | 26 |
| 1110 | 14 | 11100 | 28 |
| 1111 | 15 | 11110 | 30 |

Tabela 2: Tabela verdade para a simulação da atividade 01

# Atividade 02

Escreva em VHDL e simule no ModelSim um somador de palavras de 4 bits usando o operador ‘+’ do pacote STD\_LOGIC\_ARITH. A nova entidade deve ter como entrada dois vetores A e B (de 4 bits cada) e um vetor de saída S (de 5 bits). **Dica**: use o comando ‘unsigned(.)’ para converter uma variável do tipo STD\_LOGIC\_VECTOR em UNSIGNED para então usar o operador ‘+’.

O uso da biblioteca STD\_LOGIC\_ARITH simplifica bastante a implantação do somador. Com ela, o somador pode ser implantado com uma simples operação matemática de soma:

-- Experimento 05 - Questão 02

-- Aluno: Pedro Henrique Dias Avelar 241037112

-- Turma 08

-- Data: 11/12/2024

--SOMADOR DE PALAVRAS DE 4 BITS

--ENTRADA: A E B (4 BITS)

--SAÍDA: S (5 BITS)

--USAR O OPERADOR '+' DA BIBLIOTECA STD\_LOGIC\_ARITH

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**USE** IEEE**.**STD\_LOGIC\_ARITH**.ALL;**

**ENTITY** E05Q02 **IS**

**PORT** **(**A**,**B**:** **IN** STD\_LOGIC\_VECTOR **(**3 **DOWNTO** 0**);**

S**:** **OUT** STD\_LOGIC\_VECTOR **(**4 **DOWNTO** 0**));**

**END** E05Q02**;**

**ARCHITECTURE** E05Q02\_ARC **OF** E05Q02 **IS**

**BEGIN**

S **<=** **(**'0' **&** UNSIGNED**(**A**))** **+** **(**'0' **&** UNSIGNED**(**B**));**

**END** E05Q02\_ARC**;**

Código 3:Implantação do somador de palavras de 4 bits com o uso da biblioteca STD\_LOGIC\_ARITH

Para o devido funcionamento do código acima, é necessário concatenar um zero por meio do operador &, de modo que os dois operandos da soma possuam o mesmo número de bits que o vetor S.

Novamente, para simplificar a simulação, desta vez o vetor B foi considerado como tendo o valor constante “1111” (15). Assim a tabela verdade novamente apresentará 16 resultados:

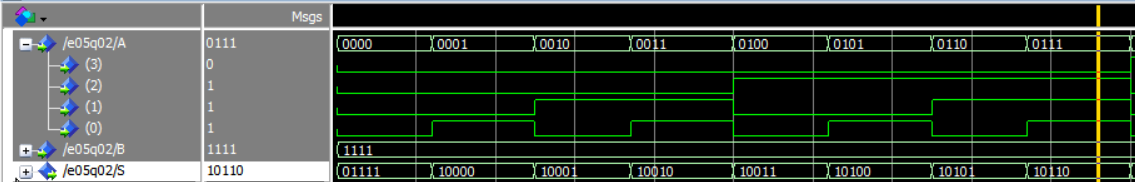


Figura 6: Resultados: 01111; 10000; 10001; 10010; 10011; 10100; 10101; 10110

A screenshot of a computer

Description automatically generated

Figura 7:Resultados: 10111; 11000; 11001; 11010; 11011; 11100; 11101; 11110

Montando a tabela verdade temos que:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | A (decimal) | B | B (decimal) | S | S (decimal) |
| 0000 | 0 | 1111 | 15 | 01111 | 15 |
| 0001 | 1 | 1111 | 15 | 10000 | 16 |
| 0010 | 2 | 1111 | 15 | 10001 | 17 |
| 0011 | 3 | 1111 | 15 | 10010 | 18 |
| 0100 | 4 | 1111 | 15 | 10011 | 19 |
| 0101 | 5 | 1111 | 15 | 10100 | 20 |
| 0110 | 6 | 1111 | 15 | 10101 | 21 |
| 0111 | 7 | 1111 | 15 | 10110 | 22 |
| 1000 | 8 | 1111 | 15 | 10111 | 23 |
| 1001 | 9 | 1111 | 15 | 11000 | 24 |
| 1010 | 10 | 1111 | 15 | 11001 | 25 |
| 1011 | 11 | 1111 | 15 | 11010 | 26 |
| 1100 | 12 | 1111 | 15 | 11011 | 27 |
| 1101 | 13 | 1111 | 15 | 11100 | 28 |
| 1110 | 14 | 1111 | 15 | 11101 | 29 |
| 1111 | 15 | 1111 | 15 | 11110 | 30 |

Tabela 3: Tabela verdade para a simulação da atividade 02

# Atividade 03

Escreva em VHDL e simule no ModelSim um *testbench* para simular e testar o somador de palavras de 4 bits desenvolvido para o item 1. Esse *testbench* deve gerar todas as 256 combinações possíveis de valores para A e B, aguardando 500 nanosegundos entre combinações, e, para cada combinação, comparar a saída do somador do item 1 (utilizado aqui como *Device Under Test*) com a saída do somador do item 2 (utilizado aqui como *golden model*). Para cada combinação em que as saídas não concordarem, deve ser impressa uma mensagem de erro. **Atenção**: o testbench deve ser capaz de imprimir mensagens em caso de erro, mas é esperado que os dois somadores obtenham sempre a mesma saída.

A terceira atividade envolve a criação de dois módulos adicionais. O testbench nos permite realizar a simulação com maior controle, podendo utilizar variáveis e estruturas de repetição. Já o top module serve para fazer a ligação entre os códigos das atividades 01 e 02 com o testbench.

Código 4: Código para o testbench

-- Experimento 05 - Questão 03

-- Aluno: Pedro Henrique Dias Avelar 241037112

-- Turma 08

-- Data: 16/12/2024

--Testbench

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**USE** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;**

**USE** IEEE**.**NUMERIC\_STD**.ALL;**

**ENTITY** TESTBENCH **IS**

**PORT** **(**S\_DUT**,** S\_GM**:** **IN** STD\_LOGIC\_VECTOR **(**4 **DOWNTO** 0**);**

AT**,** BT**:** **OUT** STD\_LOGIC\_VECTOR **(**3 **DOWNTO** 0**));**

**END** TESTBENCH**;**

**ARCHITECTURE** TESTBENCH\_ARC **OF** TESTBENCH **IS**

**BEGIN**

**PROCESS**

**VARIABLE** CONT\_A**:** UNSIGNED **(**3 **DOWNTO** 0**);**

**VARIABLE** CONT\_B**:** UNSIGNED **(**3 **DOWNTO** 0**);**

**BEGIN**

**REPORT** "INICIANDO TESTE..."**SEVERITY** NOTE**;**

CONT\_A **:=** "0000"**;**

CONT\_B **:=** "0000"**;**

**FOR** I **IN** 0 **TO** 15 **LOOP**

AT **<=** STD\_LOGIC\_VECTOR**(**CONT\_A**);**

**FOR** J **IN** 0 **TO** 15 **LOOP**

BT **<=** STD\_LOGIC\_VECTOR**(**CONT\_B**);**

**WAIT** **FOR** 500 ns**;**

**ASSERT** **(**S\_DUT **=** S\_GM**)** **REPORT** "FALHOU: I = " **&** INTEGER'**IMAGE(**I**)** **&** " J = " **&** INTEGER'**IMAGE(**J**)** **SEVERITY** ERROR**;**

CONT\_B **:=** CONT\_B **+** "1"**;**

**END** **LOOP;**

CONT\_A **:=** CONT\_A **+** "1"**;**

**END** **LOOP;**

**REPORT** "TESTE FINALIZADO!" **SEVERITY** NOTE**;**

**WAIT;**

**END** **PROCESS;**

**END** TESTBENCH\_ARC**;**

Usando as duas variáveis CONT\_A e CONT\_B, junto com duas estruturas de repetição FOR aninhadas, podemos facilmente gerar todas as 256 combinações possíveis para os vetores de entrada A e B. A cada iteração, o *testbench* irá fazer um ASSERT para verificar se a soma do *device under test* (módulo da atividade 01) é igual a soma do *golden module* (módulo da atividade 2), imprimindo uma mensagem de erro em caso de divergência. Em cada iteração há uma pausa de 500 nanosegundos realizada por meio do comando WAIT.

Código 5: Código para o top module

-- Experimento 05 - Questão 03

-- Aluno: Pedro Henrique Dias Avelar 241037112

-- Turma 08

-- Data: 11/12/2024

--Top Module

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**USE** IEEE**.**STD\_LOGIC\_ARITH**.ALL;**

**ENTITY** TOPMODULE **IS**

**END** TOPMODULE**;**

**ARCHITECTURE** TOPMODULE\_ARC **OF** TOPMODULE **IS**

**COMPONENT** E05Q01 **IS**

**PORT** **(**A**,**B**:** **IN** STD\_LOGIC\_VECTOR **(**3 **DOWNTO** 0**);**

S**:** **OUT** STD\_LOGIC\_VECTOR **(**4 **DOWNTO** 0**));**

**END** **COMPONENT;**

**COMPONENT** E05Q02 **IS**

**PORT** **(**A**,**B**:** **IN** STD\_LOGIC\_VECTOR **(**3 **DOWNTO** 0**);**

S**:** **OUT** STD\_LOGIC\_VECTOR **(**4 **DOWNTO** 0**));**

**END** **COMPONENT;**

**COMPONENT** TESTBENCH **IS**

**PORT** **(**S\_DUT**,** S\_GM**:** **IN** STD\_LOGIC\_VECTOR **(**4 **DOWNTO** 0**);**

AT**,** BT**:** **OUT** STD\_LOGIC\_VECTOR **(**3 **DOWNTO** 0**));**

**END** **COMPONENT;**

**SIGNAL** A\_SIGNAL**,** B\_SIGNAL**:** STD\_LOGIC\_VECTOR **(**3 **DOWNTO** 0**);**

**SIGNAL** SOM\_DUT**,** SOM\_GM**:** STD\_LOGIC\_VECTOR **(**4 **DOWNTO** 0**);**

**BEGIN**

SOMADOR\_DUT**:** E05Q01 **PORT** **MAP(**A\_SIGNAL**,** B\_SIGNAL**,** SOM\_DUT**);**

SOMADOR\_GML**:** E05Q02 **PORT** **MAP(**A\_SIGNAL**,** B\_SIGNAL**,** SOM\_GM**);**

SOMADOR\_TST**:** TESTBENCH **PORT** **MAP(**SOM\_DUT**,** SOM\_GM**,** A\_SIGNAL**,** B\_SIGNAL**);**

**END** TOPMODULE\_ARC**;**

O top module usa o código das atividades 01 e 02, além do testbench, como componentes. Utiliza-se os mesmos sinais (A\_SIGNAL e B\_SIGNAL) para os 3 componentes, de modo que a entrada de ambos os somadores seja a mesma gerada pelo testbench. Da mesma forma, os sinais de saída SOM\_DUT e SOM\_GM são conectados ao testbench para que se possa fazer o ASSERT. Para rodar a simulação, foi selecionada a opção SIMULATE – RUN – RUN ALL para o top module.

A screenshot of a computer

Description automatically generated

Figura 8: Resultado da simulação para o top module

A screenshot of a computer program

Description automatically generated

Figura 9: Log da simulação

Como esperado, embora previsto no testbench, como não houve divergência entre a soma calculada pelo device under test e a soma calculada pelo golden module, não houve nenhuma impressão de mensagem de erro; houve impressão apenas das mensagens de início e fim do teste.