UNIVERSIDADE DE BRASÍLIA

Adrianne Alves da Silva - 160047595
Ana Paula Gomes de Matos - 160023629

TURMA "F"
PRÁTICA DE ELETRÔNICA DIGITAL - 119466
Experimento 02:Circuitos Lógicos Combinacionais

Brasília - DF
07 de abril de 2017

UNIVERSIDADE DE BRASÍLIA

PRÁTICA DE ELETRÔNICA DIGITAL - 119466

Relatório referente ao segundo experimento da disciplina Prática de Eletrônica Digital, ministrada pela professora Lourdes Mattos Brasil, realizado no dia 31 de março de 2017.

Alunas:

Adrianne Alves da Silva

Ana Paula Gomes de Matos

Brasília – DF

SUMÁRIO

1 INTRODUÇÃO

1.1 Circuitos Lógicos Combinacionais

O conhecimento das portas lógicas fundamentais, assim como as suas derivações, abrem portas para o desenvolvimento da eletrônica. A partir do momento em que é possível combinar a lógica de várias delas a fim de alcançar a resolução de um problema, obtém-se os circuitos lógicos combinacionais. Assim, se obtemos a conexão de portas lógicas para uma saída especificada pela combinação das entradas, sem que estas sejam armazenadas, o circuito obtido obedece à lógica combinacional. [1]

Esse tipo de abordagem é poderosa no que diz respeito à eletrônica digital, pois é a partir dela que se constrói circuitos lógicos úteis. Dessa maneira, identificada uma situação, consegue-se relacioná-la através de uma lógica booleana que apresenta um comportamento determinado para a saída, de acordo às entradas processadas. A principal maneira de expressar essa conexão é através da tabela-verdade que ilustra o problema, isto após identificar quantas e quais são as entradas e saídas, assim como, o que representaria a entrada '1' e '0' na questão. [2]

Em geral, um circuito lógico combinacional é aquele que possui saídas dependentes única e exclusivamente dos valores pré-estabelecidos em sua entrada, assim, seu fluxo é composto por um problema, tabela verdade, equação lógica e o circuito propriamente dito.

1.2 Tabela verdade e expressão lógica

A tabela verdade é usada para determinar valores lógicos de proposições compostas a partir de proposições simples com valores iniciais já conhecidos, é ainda um meio de estabelecer as correspondências existentes entre os valores de uma função e os das variáveis. [3]

Ela pode ser obtida das seguintes formas:

1) Análise do problema proposto: este método é o mais subjetivo, pois vai depender integralmente o problema apresentado, não havendo portanto um padrão pré-estabelecido.

2) Por meio de uma função booleana:
   a) monta-se a coluna completa de todas as possíveis combinações das variáveis de entrada de acordo com a expressão $2^{\square} + 1$, onde **n** é a quantidade de variáveis;
   b) monta-se colunas complementares equivalentes à quantidade de "parcelas" da equação booleana;
   c) monta-se uma última coluna correspondente aos valores de saída;
   d) por fim, deve-se completar a tabela obedecendo às restrições de cada "parcela".

3) Diretamente do circuito: deve-se realizar os procedimentos **a** e **c** do item anterior, após isto, deve-se analisar linha por linha da tabela e identificar o resultado de saída.

Por intermédio da tabela verdade também é possível identificar a equação booleana do circuito, para melhor compreensão, observe o exemplo a seguir:

Tabela Verdade

	A	B	C	y
1	0	0	0	0
2	0	0	1	0
3	0	1	0	1
4	0	1	1	0
5	1	0	0	1
6	1	0	1	1
7	1	1	0	0
8	1	1	1	1

a) deve-se considerar somente as linhas cujo valor de **y** são iguais a **1**;
b) faz-se a aplicação da função AND entre as variáveis **A**, **B** e **C**, caso o resultado dê "0" deve-se aplicar também a função NOT, exemplo:

$$\text{linha } 3 \rightarrow A=0,\ B=1 \text{ e } C=0 \rightarrow \overline{A}\cdot B\cdot \overline{C}$$
$$\text{linha } 5 \rightarrow A=1,\ B=0 \text{ e } C=0 \rightarrow A\cdot \overline{B}\cdot \overline{C}$$
$$\text{linha } 6 \rightarrow A=1,\ B=0 \text{ e } C=1 \rightarrow A\cdot \overline{B}\cdot C$$
$$\text{linha } 8 \rightarrow A=1,\ B=1 \text{ e } C=1 \rightarrow A\cdot B\cdot C$$

c) o último passo é aplicar a função OR entre cada expressão obtida anteriormente:

$$y = A\cdot B\cdot C + A\cdot B\cdot C + A\cdot B\cdot C + A\cdot B\cdot C$$

Após isto é possível simplificar as equações como é apresentado no tópico seguinte

1.3 Simplificação de funções lógicas

Existem diferentes formas para simplificação de equações lógicas, dentre elas vale destacar o mapa de Karnaugh e o método de fatoração, aqui será abordado apenas o segundo método, pois o primeiro torna-se complicado de usar caso o sistema apresente mais que sete variáveis.

A simplificação de funções lógicas é importante, pois diminui o grau de dificuldade presente no processo de montagem e no custo do circuito. Para que os passos sejam facilmente executados são necessários conhecimentos prévios de álgebra booleana, suas propriedades, equivalência, etc., nesta seção será abordado apenas as principais propriedades.

Quadro 01: Propriedades da álgebra booleana

Propriedade	Complemento	Adição	Multiplicação
Identidade	$\overline{\bar{A}} = A$	A + 0 = A A + 1 = 1 A + A = A A + $\bar{A}$ = 1	A . 0 = 0 A . 1 = A A . A = A A . $\bar{A}$ = 0
Comutativa		A + B = B + A	A . B = B . A
Associativa		A+(B+C) = (A+B)+C = A+B+C	A.(B.C) = (A.B).C = A.B.C
Distributiva		A+(B.C) = (A+B) . (A+C)	A.(B+C) = A.B + A.C

Vale destacar também as seguintes propriedades:

Obs.: $\bar{A}$ = **A'**

A + (A.B) = A
A . (A+B) = A
A + $\bar{A}$.B = A + B
(A + B) . (A + C) = A + B.C
(A.B)' = A' + B'
(A + B)' = A'. B'

Para a simplificação das expressões lógicas, faz-se necessário apenas a aplicação das propriedades apresentadas, estes procedimentos facilitam a implementação no código em VHDL.

1.4 Objetivo

Capacitar o aluno no que diz respeito à aplicação real de circuitos lógicos combinacionais, mostrando-o os resultados devido à tomada de decisões em um projeto.

1.5 Justificativa

Nem sempre um problema deixa explícito sua equação lógica, logo, conhecer técnicas que auxiliam na obtenção da mesma facilitam na identificação da tabela verdade, implementação em um *software* e principalmente, na certificação de que o circuito está realmente correto.

Entender a relação das variáveis de entradas com os valores da saída de um circuito, é de extrema relevância quando há interesse em predeterminar o comportamento de um projeto e certificar de que o mesmo atinge seu objetivo, para isto, é crucial a compreensão da prática e teoria que

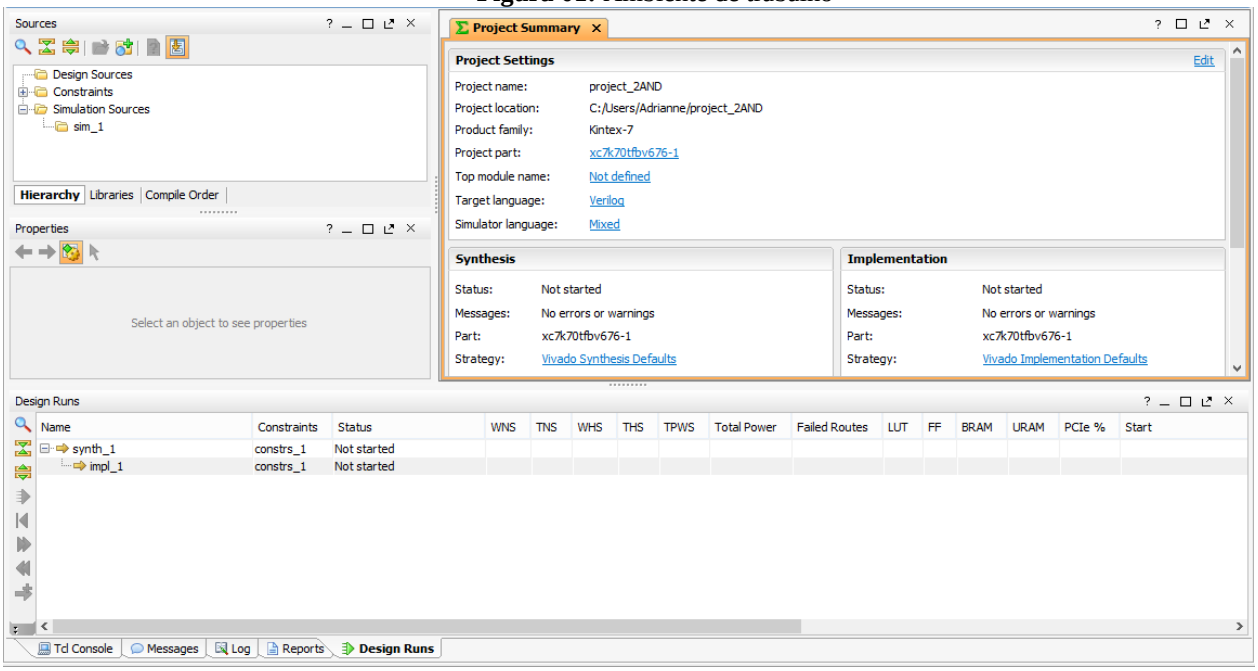
rodeiam temáticas ligadas a circuitos lógicos combinacionais, e isto pode ser claramente observado na execução do experimento dois.

2 EXPERIMENTOS

2.1 Ambiente de simulação Vivado

Como todo projeto, inicia-se criando um novo arquivo no software de preferência, nesse caso, o Vivado. Após definir as portas de entrada e saída, nomeadas as entradas como A,B e C e Saída com S, foi necessário adicionar a placa Basys 3 ao projeto, assim como o arquivo em VHDL no qual seria realizada a implementação.Com as demais configurações básicas, obteve-se o ambiente descrito abaixo.

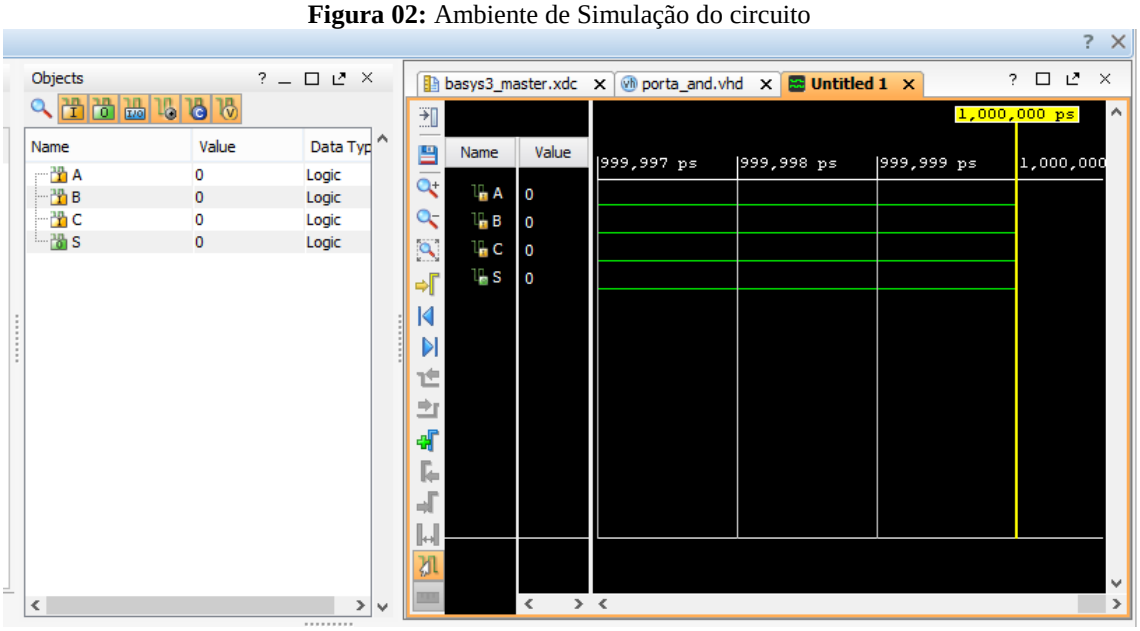
Figura 01: Ambiente de trabalho



Em sequência, implementou-se para cada atividade o código correspondente, através da expressão booleana adequada na linguagem VHDL, e mesmo que não fossem realizados testes físicos em FPGA’s, os pinos de entrada e saída foram ligados à placa Basys 3. Isso foi feito, adequadamente, por meio da retirada de comentários de linhas específicas do arquivo *basys3.xdc*. Assim, os circuitos implementados foram sintetizados, gerando os esquemáticos que tornam a tarefa de checagem de pequenos circuitos, trivial.

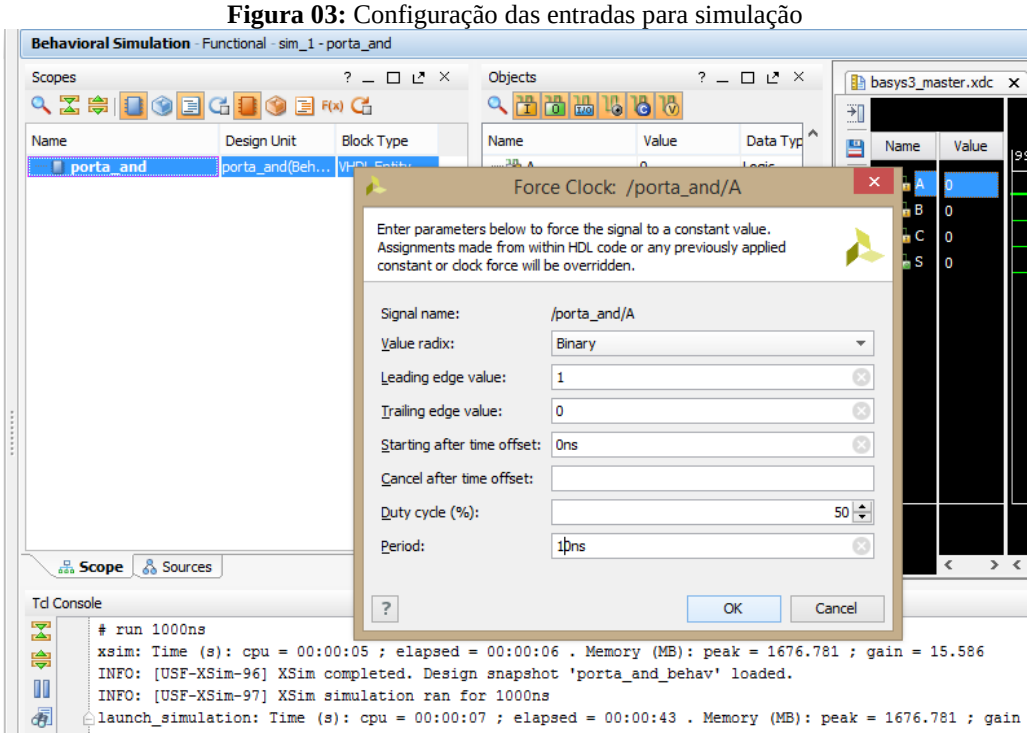
O passo seguinte, talvez o mais importante para análise comportamental do circuito, foi fazer a simulação do seu funcionamento, por meio da run simulation que gera o ambiente de simulação presente na figura 2.

Figura 02: Ambiente de Simulação do circuito



Nesse espaço é possível configurar a simulação para tornar essa atividade a mais produtiva possível. O primeiro passo consiste em definir os valores de entrada (‘0’ ou ‘1’), assim como o período do sinal para cada entrada. É necessário forçar o *clock* de modo que o pino de uma entrada possua o dobro do período da outra , com a finalidade de gerar ondas mais fáceis de serem analisadas.

Figura 03: Configuração das entradas para simulação



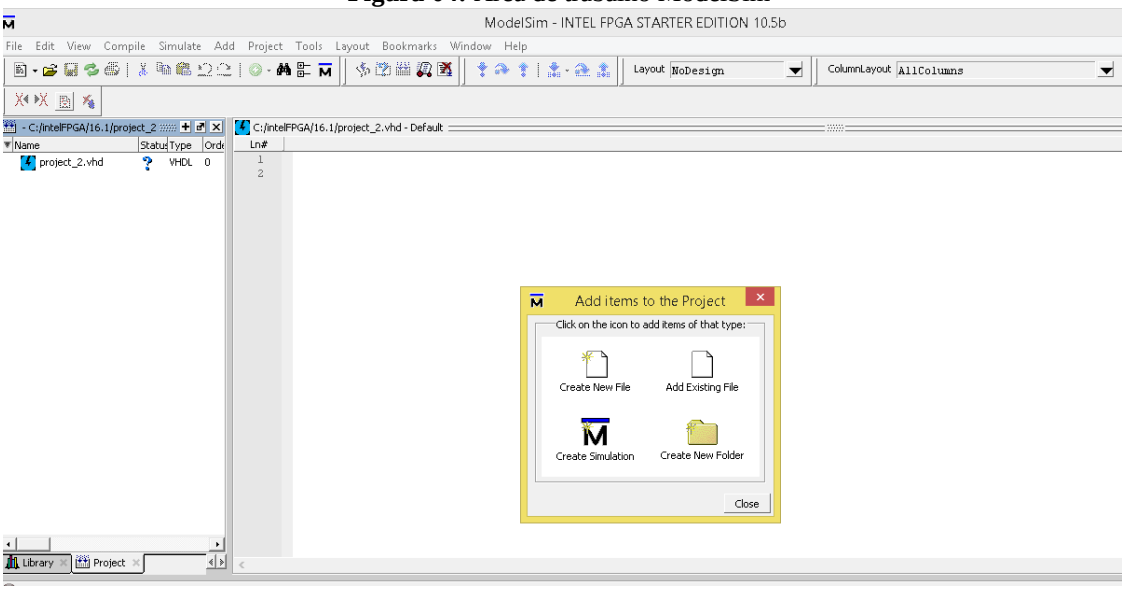
A primeira configuração realizada é a modificação do argumento *value* para o tipo binário informando entradas diferenciadas para os argumento *Leading edge value* e *Trailing*

edge value, neste experimento, adotou-se o padrão de 1 e 0 para todas as entradas.Após toda essa configuração basta executar a simulação.

2.2 Ambiente de simulação ModelSim

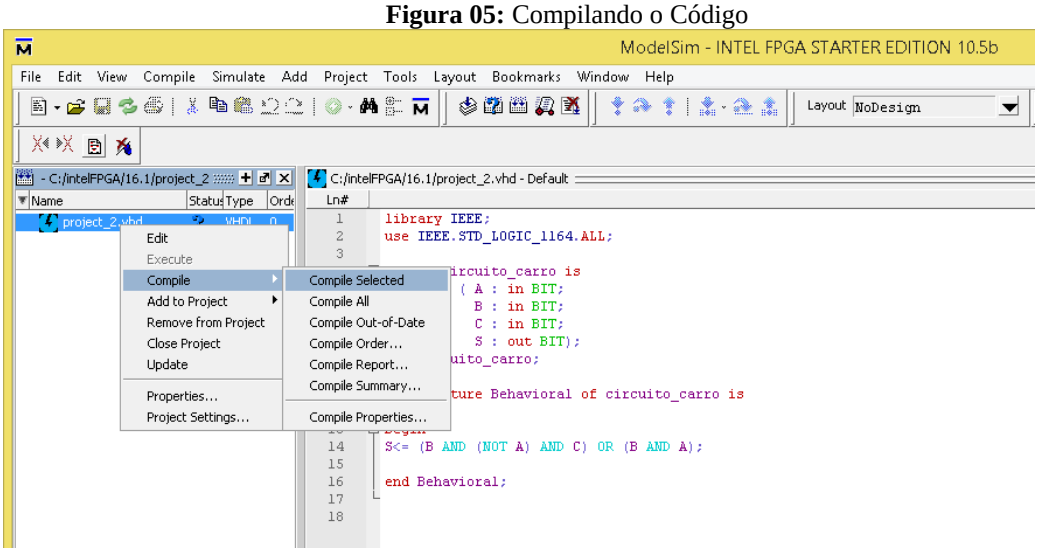
O modelSim, semelhantemente ao Vivado, iniciamos criando um novo projeto através do caminho *File - > New Project* por meio do qual é possível criar o arquivo VHDL , tal que apresenta a seguinte àrea de trabalho (figura 4).

Figura 04: Área de trabalho ModelSim



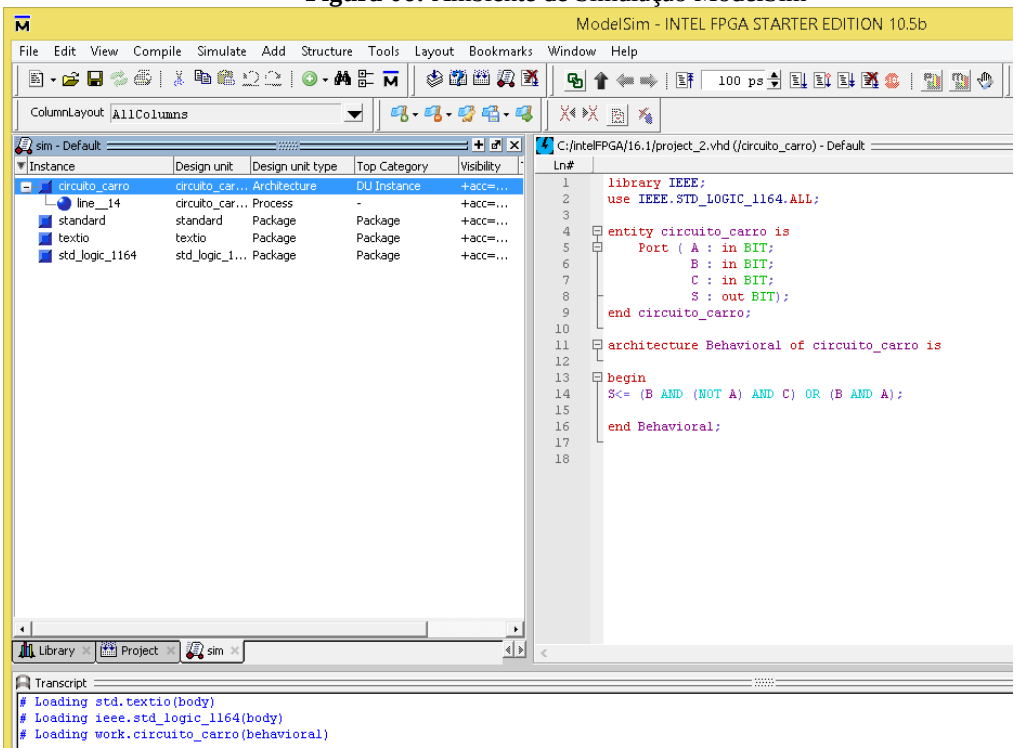
Após criado o arquivo VHDL, basta com um clique sobre o nome, abri-lo e implementá-lo, salvando com o atalho *crt+s* .É necessário então, compilar o seu código, seguindo o caminho ilustrado na figura a seguir.

Figura 05: Compilando o Código



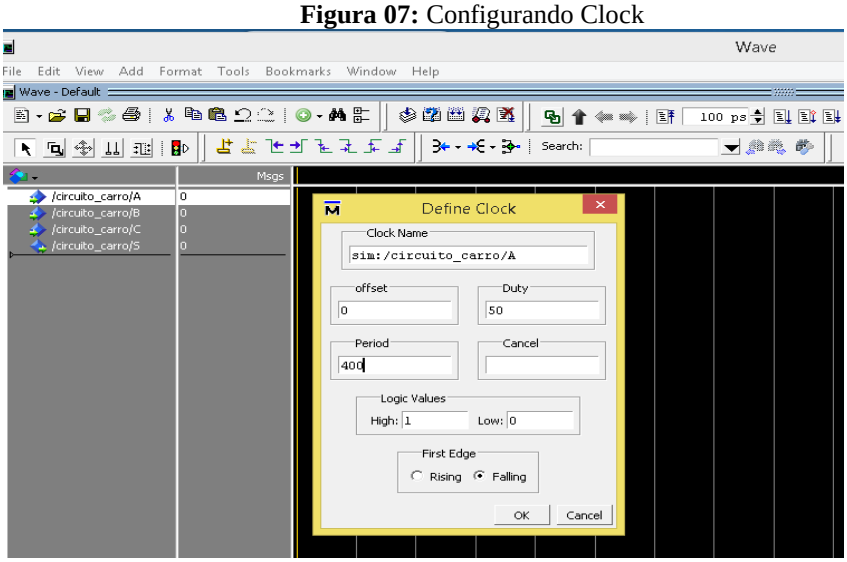
Após compilado, se não houver nada de errado, é possível realizar, finalmente, a simulação através do caminho *simulate - > start simulation*, escolhe-se o arquivo implementado, e enfim, alcança o seguinte ambiente de simulação:

Figura 06: Ambiente de Simulação ModelSim



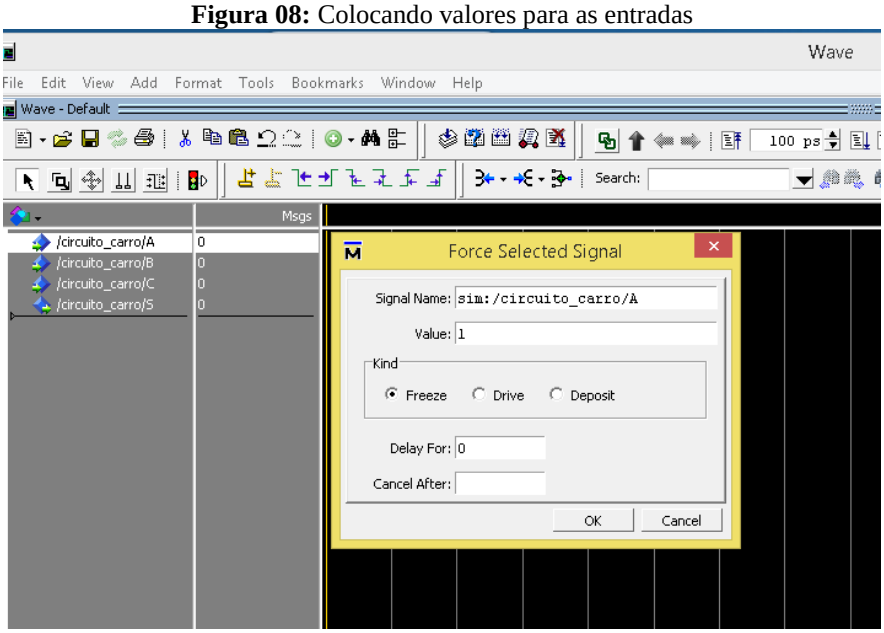
Basta clicar agora com o botão direito sobre o seu projeto e *add wave*, para então gerar a onda referente ao seu projeto. Na tela que se abrir, basta clicar com o botão direito sobre cada uma das entradas e ir em *clock (*figura 7*)*, para configurar o período entre as ondas e depois em cada entrada modificar o valor através de *force (figura 8)* .Feito isso, basta rodar a simulação e analisar as ondas resultantes.

Figura 07: Configurando Clock



Para o *clock* adota-se para uma entrada, o dobro da outra, para que as ondas se tornem mais fáceis de serem visualizadas, além disso, é preciso marcar a opção *Falling*.

Figura 08: Colocando valores para as entradas



2.3 Execução do experimento

Esse experimento consistiu em três diferentes projetos que relacionam a lógica combinacional, a simplificação por meio das técnicas do mapa de Karnaugh e a implementação propriamente dita em VHDL e simulação em diferentes ambientes profissionais (Vivado, ModelSim e QUCS).

2.3.1 Simplificação de circuitos

O uso de técnicas para a simplificação de expressões lógicas influem diretamente sob o número de portas lógicas utilizadas, e consequentemente, na quantidade de material e recurso financeiro gasto com o mesmo. Além disso, é possível que essa simplificação reduza a dificuldade em encontrar uma porta lógica específica, como por exemplo, uma porta *AND* com três pinos de entrada, como propõe o primeiro circuito a ser analisado. Considera-se no mesmo, que por ambiguidade de escrita seja tarefa do leitor tornar um circuito com uma porta de três entradas em um circuito com portas AND de 2 entradas, para isto, basta obter uma nova expressão lógica, conforme apresenta o código a seguir.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity porta_and is
        Port ( A : in BIT;
        B : in BIT;
        C : in BIT;
        S : out BIT);
end porta_and;
```

architecture Behavioral of porta_and is

begin

S <= A AND B AND C;

end Behavioral;

Observa-se que foi realizada a troca de uma porta de duas entradas por duas simples com duas entradas cada, cujo esquemático, está apresentado ao final deste arquivo. Mas seu funcionamento é expresso segundo a tabela verdade (tabela 1) que se assemelha à tabela verdade de uma única porta *AND*.

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Se rodarmos a simulação, percebemos que possui o mesmo funcionamento de uma porta AND de três pinos (figura 5).

Figura 09: Formas de Onda do circuito com duas portas AND - Vivado

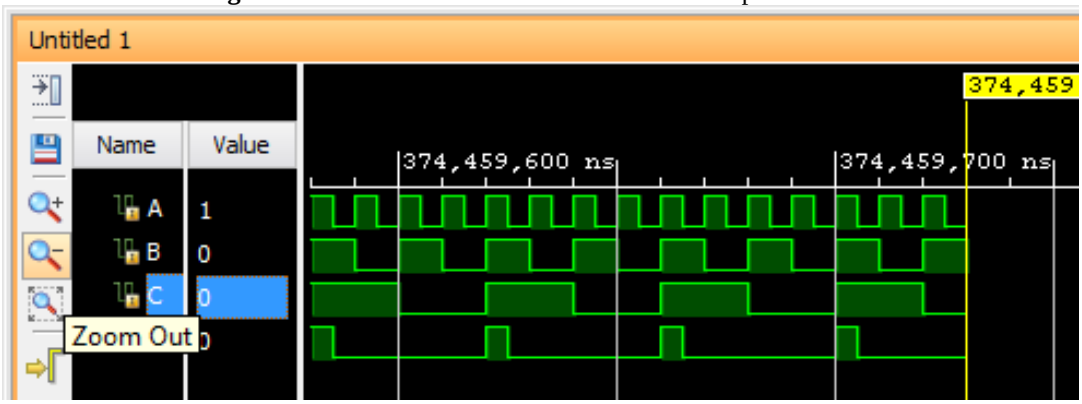
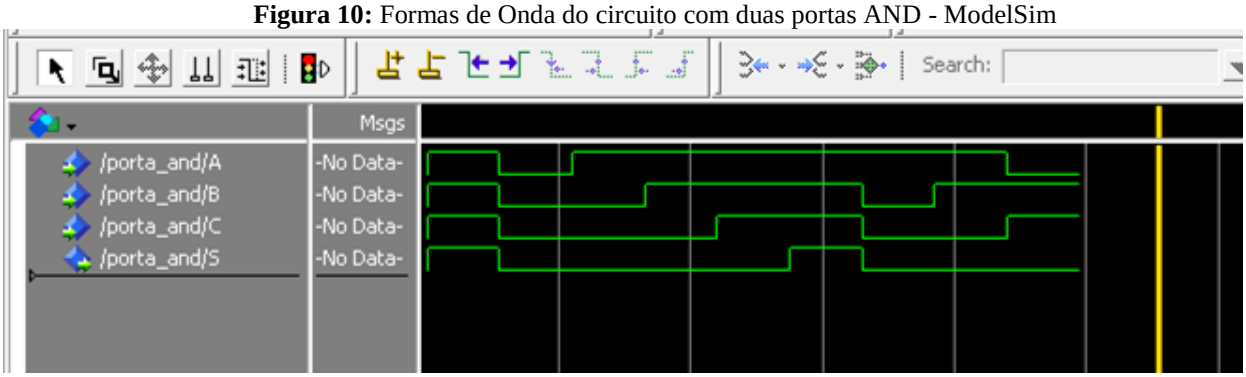


Figura 10: Formas de Onda do circuito com duas portas AND - ModelSim



2.3.2 Alarme de carro

O problema propõe, nesse caso, um circuito que satisfaça o projeto de produção do alarme de um carro, levando em consideração o estado da porta do motorista (aberta ou fechada), a ignição(ligada ou desligada) e o estado dos faróis (acesos ou apagados). São colocadas diversas regras para que o alarme seja acionado. Em primeiro contato, percebe-se que o foco deste projeto é a ativação ou não do alarme, que por si só representa uma relação booleana (‘0’ - desligado e ‘1’ ligado).

É possível definir então que o estado do alarme representa a saída do problema e possui valores binários, ligado ou desligado que podem ser escritos, equivalentemente como ‘1’ para ligado e ‘0’ para desligado. Semelhantemente percebe-se que as condições para a saída estão envoltas nos três diferentes casos ou variantes, a porta, a ignição e os faróis. Assim sendo, há 3 diferentes entradas no problema, referentes aos sinais enviados pelos sensores, e a relação descrita através das condições para que o alarme seja ligado são expressas na tabela verdade (figura 06), sendo ‘1’ para quando alguma variável está ativada (ligada, desligada, aberta) e ‘0’ para a situação contrária.

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1

1	1	1	1

Analisando as linhas cujo resultado para a ativação do alarme é positivo conseguimos a seguinte representação para a questão:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL

entity circuito_carro is
    Port ( A : in BIT;
           B : in BIT;
           C : in BIT;
           S : out BIT);
end circuito_carro;

architecture Behavioral of circuito_carro is
begin

S<= (NOT (A) AND B AND C) OR (A AND B AND (NOT(C)))OR (A AND B AND C);

end Behavioral;
```

Entretanto, esse tipo de abordagem é pouco eficiente, pois engloba um total de oito portas lógicas, sem contar com as inversoras, enquanto que, utilizando as técnicas de simplificação, é possível escrever o circuito com apenas quatro portas, reduzindo o custo com material pela metade, conforme se apresenta abaixo.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity circuito_carro is
    Port ( A : in BIT;
           B : in BIT;
           C : in BIT;
           S : out BIT);
end circuito_carro;

architecture Behavioral of circuito_carro is
```

begin

S<= (B AND (NOT A) AND C) OR (B AND A);

end Behavioral;

O esquemático de ambos os casos constam ao fim deste relatório, entretanto, é possível notar que os resultados de ambos são semelhantes, e obedecem à tabela verdade apresentada pelo problema (figura 05 e 06).

Figura 11: Formas de Onda do circuito alarme de carro (expressão sem simplificação)-Vivado

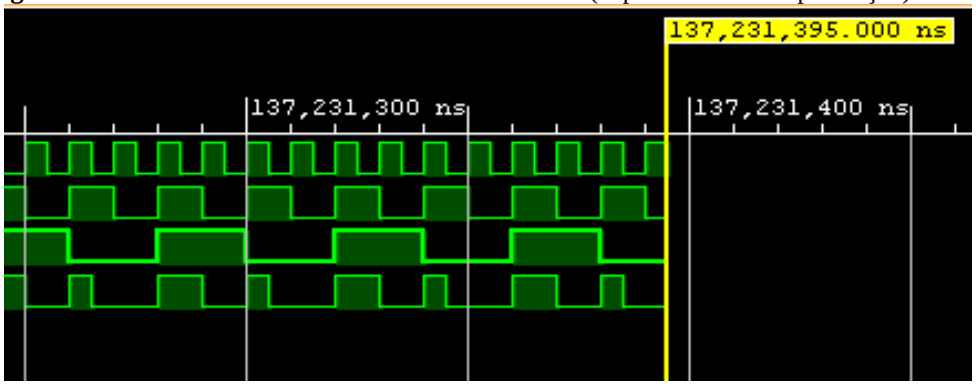


Figura 12: Formas de Onda do circuito alarme de carro (expressão simplificada)-Vivado

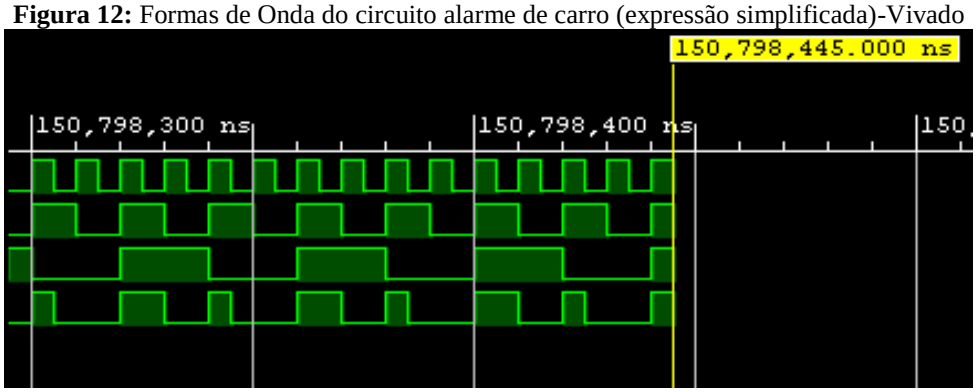
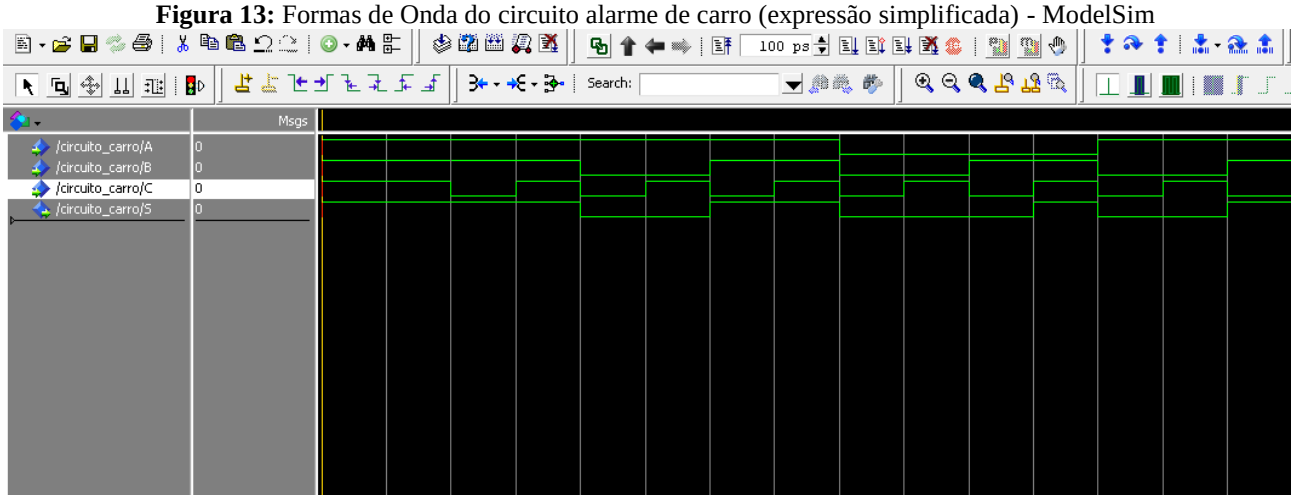


Figura 13: Formas de Onda do circuito alarme de carro (expressão simplificada) - ModelSim



2.3.3 Detector de Paridade

Um detector de paridade nada mais é que um circuito que analisando um conjunto x de bits retorna se a quantidade de bits ‘1’ foi par ou ímpar. Nesse sentido, é possível observar claramente que as entradas são os números x de bits, no caso deste projeto, quatro bits de entrada, sendo eles binários, e a saída, única deve retornar ‘1’ caso o número de ‘1’ seja ímpar e ‘0’ se for par. A análise deste problema reflete a tabela verdade a seguir, sendo as entradas A,B,C e D e saída S.

A	B	C	D	S
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Analisando as linhas da tabela cujo resultado é positivo (bit ‘1’), é possível obter a seguinte expressão implementada em VHDL :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity circuito_digitos is
        Port ( A : in BIT;
               B : in BIT;
               C : in BIT;
               D : in BIT;
               S : out BIT);
        end circuito_digitos;

architecture Behavioral of circuito_digitos is

begin

S<= ( NOT(A) AND NOT(B) AND(C XOR D))OR ( NOT(C) AND NOT (D) AND(A XOR B)) OR ( C AND D AND(A XOR B)) OR (A AND B AND (C XOR D));

 end Behavioral;
```

Observa-se que essa expressão necessita de diversas portas lógicas, entretanto, está já é uma versão simplificada da encontrada anteriormente, visto que passagens com sintaxe < (not(A) and B) or (not(B) and A)> , característica da porta lógica XNOR foram devidamente substituídas para economia de recursos . Por fim, obteve-se as seguintes ondas:

Figura 14: Formas de Onda do circuito alarme de carro (expressão simplificada)-Vivado

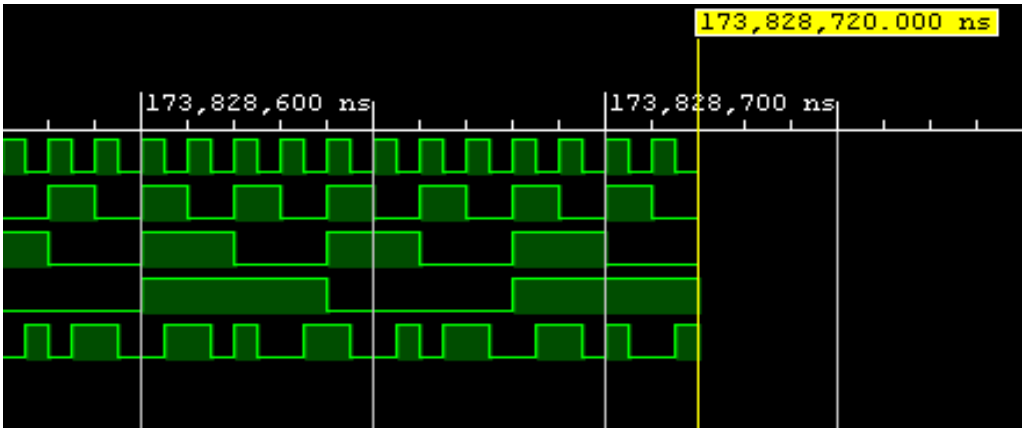
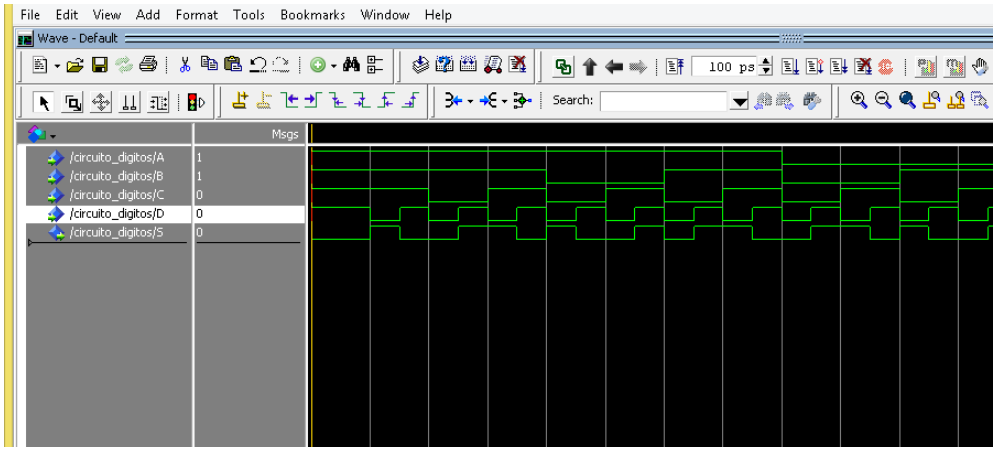


Figura 15: Formas de Onda do circuito alarme de carro (expressão simplificada)-ModelSim



Se analisarmos as ondas para cada linha da tabela, perceberemos que os resultados apresentados são coerentes, assim, o esquemático segue nas últimas páginas deste documento.

3 DISCUSSÃO

De início, foram dados dois problemas os quais foram analisados, identificadas suas variáveis de entrada e saída, levando em consideração quantas variáveis o problema demandava, quais eram elas e o que cada uma representava naquela situação específica, em seguida foi extraída a tabela verdade de ambos os casos, posterior a isso foi identificada a expressão lógica embutida no problema, no entanto, nas duas situações foram encontradas expressões muitos extensas que geram circuitos maiores e consequentemente mais caros. Contudo, por meio de manipulações algébricas embasadas nas propriedades da álgebra booleana, foi possível simplificas tais expressões sem prejudicar os valores de saída do circuito.

4 CONCLUSÕES

Diante do que foi apresentado, nota-se que todas as atividades solicitadas no experimento foram executadas com resultados satisfatórios, ou seja, foi possível simular os projetos 1 e 2 no software Vivado, o primeiro faz referência a um alarme de automóvel e o segundo a um detector de paridade, desta forma, nota-se que o objetivo do trabalho foi alcançado, contudo, com algumas limitações. Como exemplo, pode-se citar a dificuldade em encontrar materiais em língua portuguesa a respeito dos *softwares* usados para simular os circuitos; aprender a usar uma ferramenta nova, sem nenhuma orientação em um período de uma semana requer muito tempo e dedicação, sendo que os estudantes têm outras atividades em paralelo. O ideal seria a solicitação de menos detalhes nos relatórios ou até mesmo a ampliação do prazo para entrega, contudo, nota-se que a primeira opção é a mais viável, já que o cronograma da disciplina está bem apertado.

REFERÊNCIAS BIBLIOGRÁFICAS

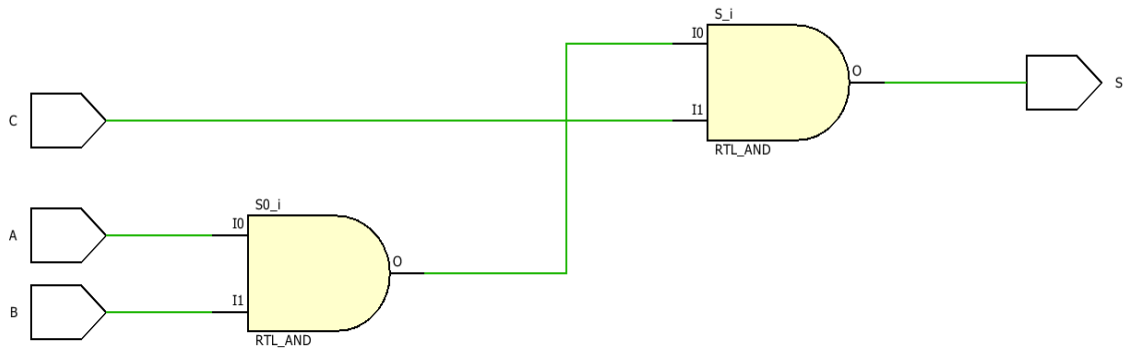
[1] FLOYD, Thomas. Álgebra booleana e simplificação lógica. In _____. **Sistemas digitais: fundamentos e aplicações**. Bookman Editora, 2009. Cap. 4, p. 200.

[2] NULL, Linda; LOBUR, Julia. **Princípios básicos de arquitetura e organização de computadores**. Bookman Editora, 2009. p. 155 - 156.

[3] AMARAL, Valder Moreira. **Eletrônica Digital**. São Paulo: Fundação Padre Anchieta, 2011. p. 41-42. 4 v.

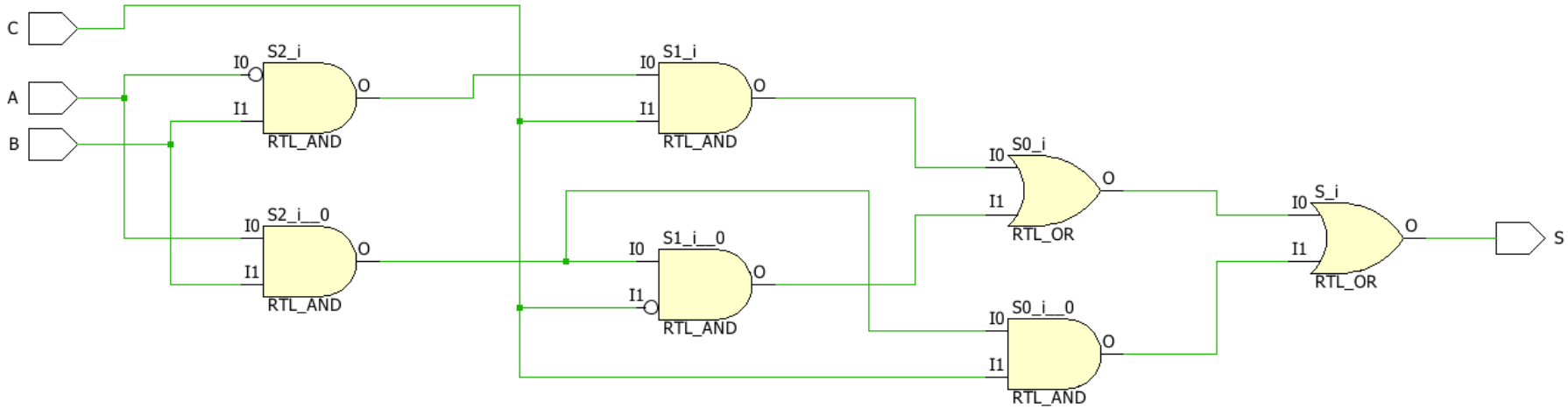
DIAGRAMAS ESQUEMÁTICOS

Esquemático 01: Diagrama esquemático simplificação de circuitos



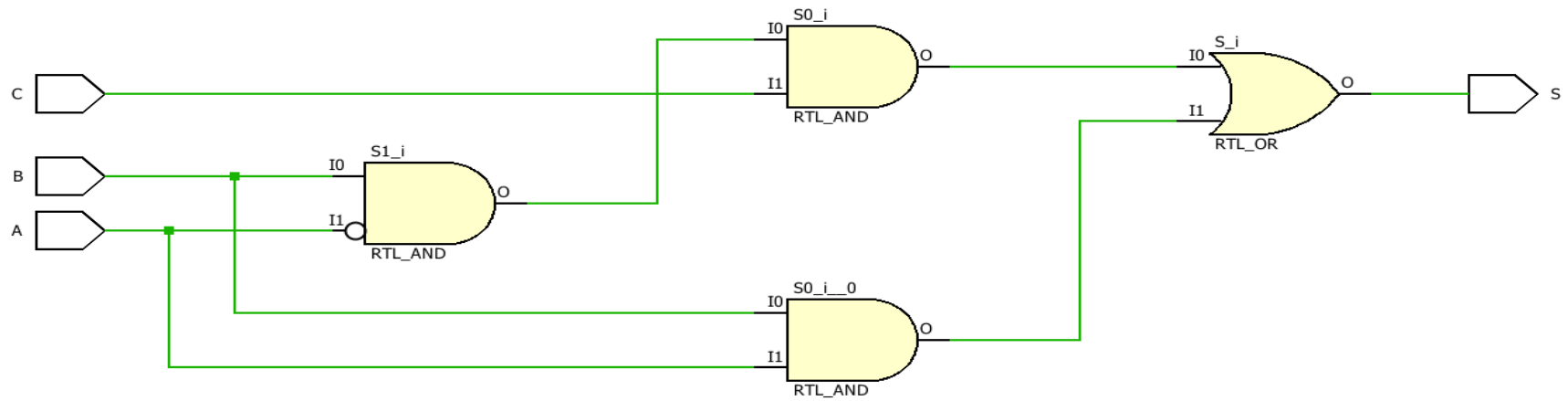
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Circuitos Combinacionais	1.0	31/03/2017	**Revisado por:**	Ana Paula Gomes de Matos

Esquemático 02: Diagrama esquemático do alarme de carro (não simplificado)



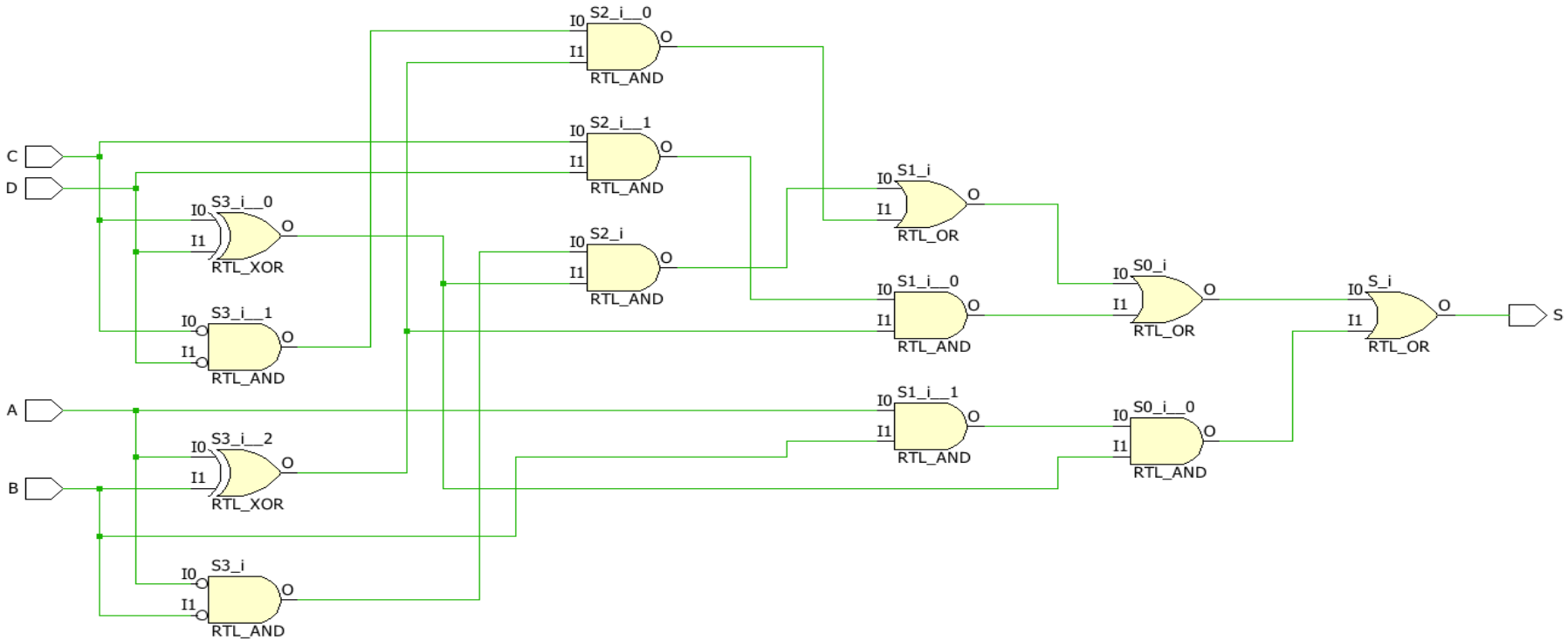
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Circuitos Combinacionais	1.0	31/03/2017	**Revisado por:**	Ana Paula Gomes de Matos

Esquemático 03: Diagrama esquemático do alarme de carro (simplificado)



UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Circuitos Combinacionais	1.0	31/03/2017	**Revisado por:**	Ana Paula Gomes de Matos

Esquemático 04: Diagrama esquemático do circuito de paridade



UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Circuitos Combinacionais	1.0	31/03/2017	**Revisado por:**	Ana Paula Gomes de Matos