UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
Curso: Bacharelado em Sistemas de Informação
Disciplina: Introdução aos Sistemas Lógicos

Nome: João Marcos Ribeiro Tolentino
Matrícula: 2021049536

Trabalho Prático 1: Processamento de Dados de Sensores Binários para o Tesla Autopilot

1) Implementação do circuito:

Foi criado o módulo “tesla_autopilot” implementado em Verilog que visa calcular o resto da divisão por 5 de valores fornecidos em binários de 4 bits. Nesse sentido, foi feito utilizando bloco always para avaliar qualquer sinal de entrada quando a variável de entrada definida for alterada. Desse modo, foram usadas estruturas “case” que mapearam os valores de forma clara para cada entrada respectiva de seu resto. Dessa maneira, esse modelo proporciona um código enxuto, simples e com um tempo de resposta curto para atender as necessidades que o módulo precisa possuir.

A abordagem foi feita de forma que atenda a lógica combinacional assegurando que a saída (que no caso é o resto por 5) seja atualizada de maneira automática sem atrasos quando ocorrer a mudança nos sinais da entrada. Em relação a eficiência da implementação, o código ficou com um bom desempenho, simples e determinístico. Assim, foi definido o resto para cada caso da divisão por 5, implementando com baixa complexidade e com fácil entendimento do código. No caso, é retornado o resto da divisão por 5 em um formato de 3 bits.

O caso de teste definido por “testbench”, visa simular o módulo “tesla_autopilot” para verificar o comportamento do módulo de cada valor por 5 com a saída de 3 bits. Desse modo, foram cobertos todos os exemplos descritos na documentação e mais alguns exemplos adicionais para garantir um código assertivo. Ademais, foi configurado o uso de um arquivo de forma de onda para fazer uma análise gráfica.

2) Forma canônica (Σm):

Forma canônica das saídas no resto da divisão por 5:

Rest: 0 - Σm(0, 5, 10, 15)
Rest: 1 - Σm(1, 6, 11)
Rest: 2 - Σm(2, 7, 12)
Rest: 3 - Σm(3, 8, 13)
Rest: 4 - Σm(4, 9, 14)

3) **Forma simplificada como P**roduto de Somas (PoS)**:**

Entrada: sensor_input = [a, b, c, d]

Sendo representados como:

a: item 3 b: item 2
c: item 1 d: item 0

Saída: representada por Rest como: [y2, y1, y0] cada valor representa o parte do resto

y2: 0100, 1001, 1110
y1: 0010, 0011, 0111, 1000, 1100, 1101
y0: 0001, 0011, 0110, 1000, 1011, 1101

Após análise do mapa de karnaugh segue na forma de PoS:
y0 = (C’ + D’ + B) (D’ + A’ + B’) (B’ + C + D’) (A’ + B + D) (A + B + C) (A + B’ + C’ + D)
y1 = (C’+ A’)(C+ A)(C’+D+ B’)(D’+ A’ + B)
y2= (D’+ B’)(D+ A’)(C+ A’)(C+ B’)(D+ B)(C’+ A+ B)

4) **Diagrama do circuito gerado:**

Diagrama do circuito gerado de acordo com os mintermos (analisados a partir do mapa de karnaugh):

Diagrama de Y0:

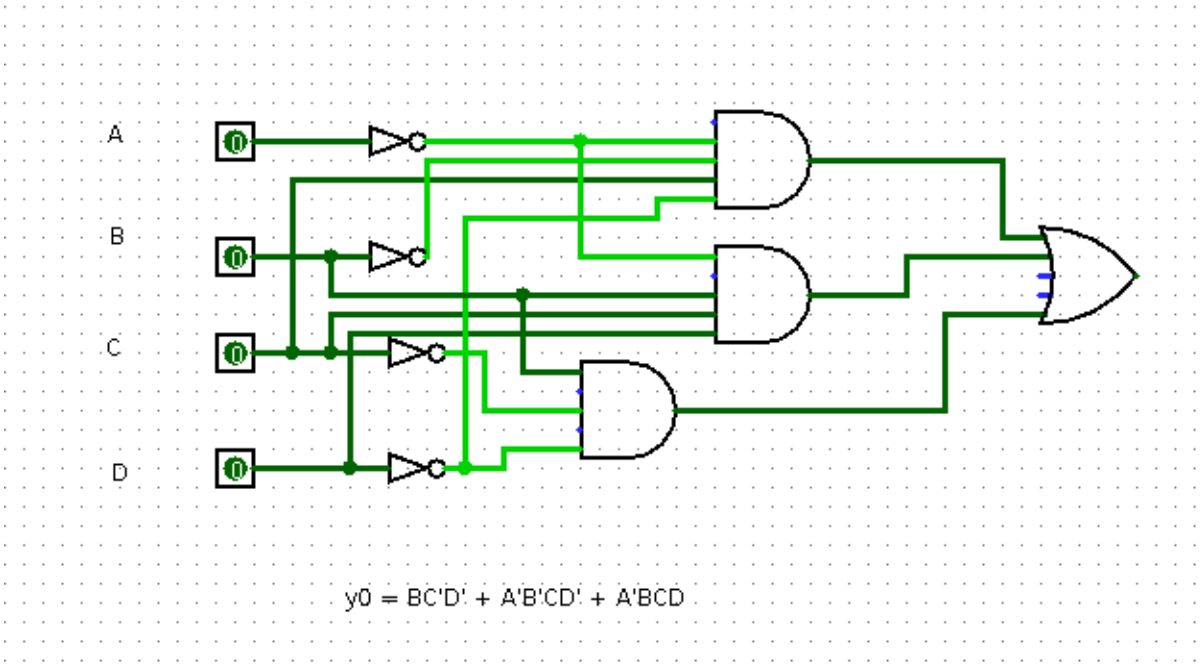


Diagrama de Y1:

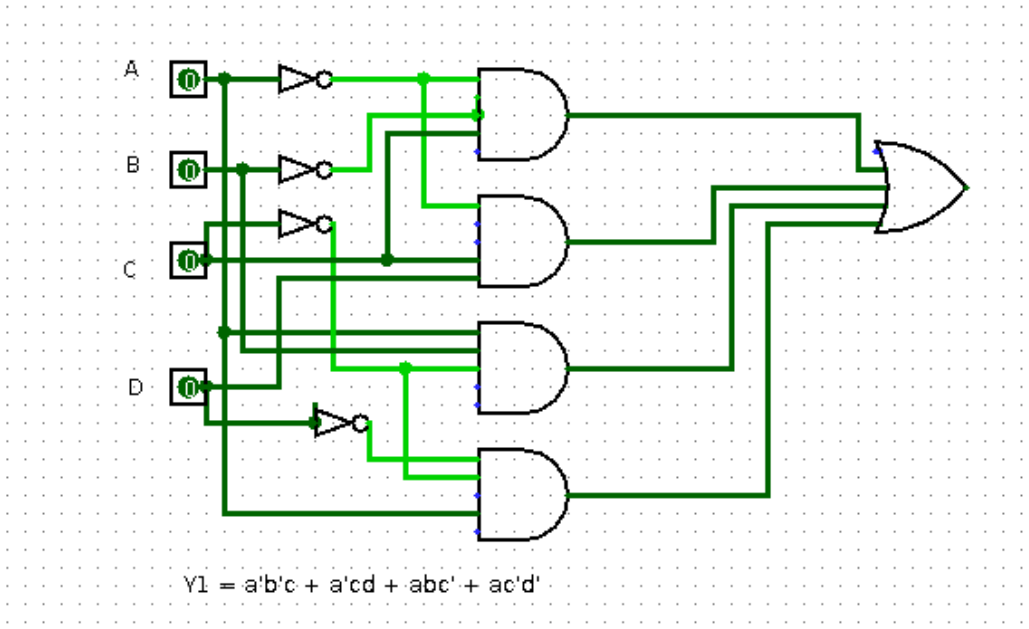
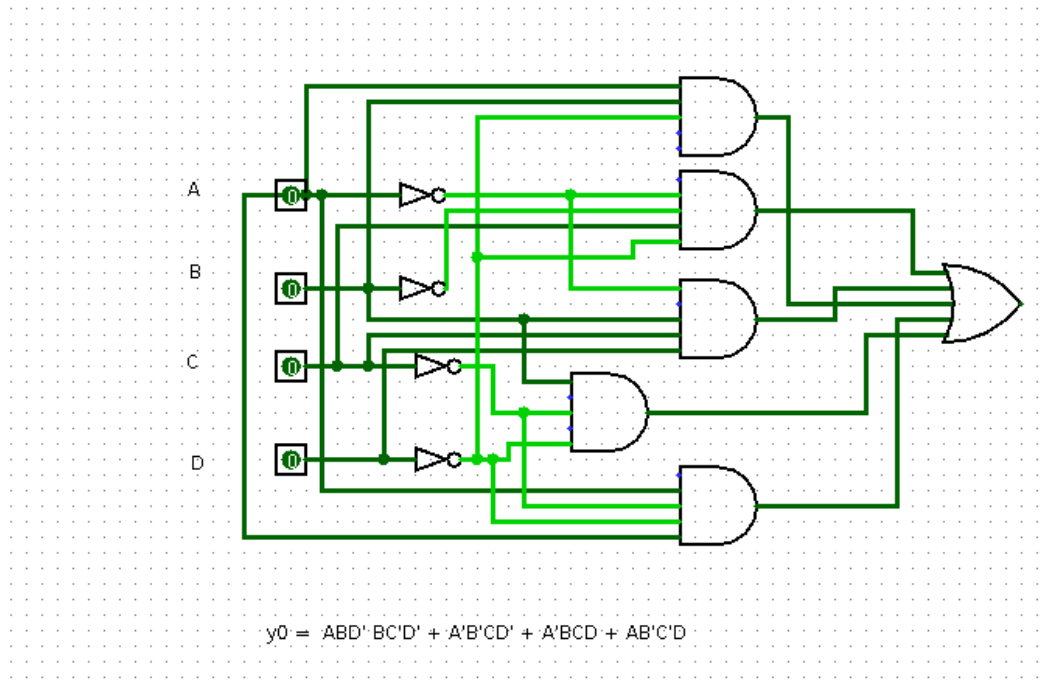


Diagrama de Y2:



5) **Mapa de karnaugh (de cada saída):**

De acordo com a tabela verdade construída no tópico 3:
y2: 0100, 1001, 1110
y1: 0010, 0011, 0111, 1000, 1100, 1101.
y0: 0001, 0011, 0110, 1000, 1011, 1101.

Mapa de karnaugh para y0:

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	0	0	1
11	0	1	0	0
10	1	0	1	0

Mapa de karnaugh para y1:

CD \ AB	00	01	11	10
00	0	0	1	1
01	0	0	1	0
11	1	1	0	0
10	1	0	0	0

Mapa de karnaugh para y2:

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	0	0	1
10	0	1	0	0

Conforme o mapa de karnaugh segue a Forma simplificada:

y2= a'bcd' + ab'cd + a'b'c'd
y1 = ca'b' + cda' + abc' + ac'd'
y0 = abd' + bc'd' + a'b'cd' + a'bcd + ab'c'd

6) Tabela verdade

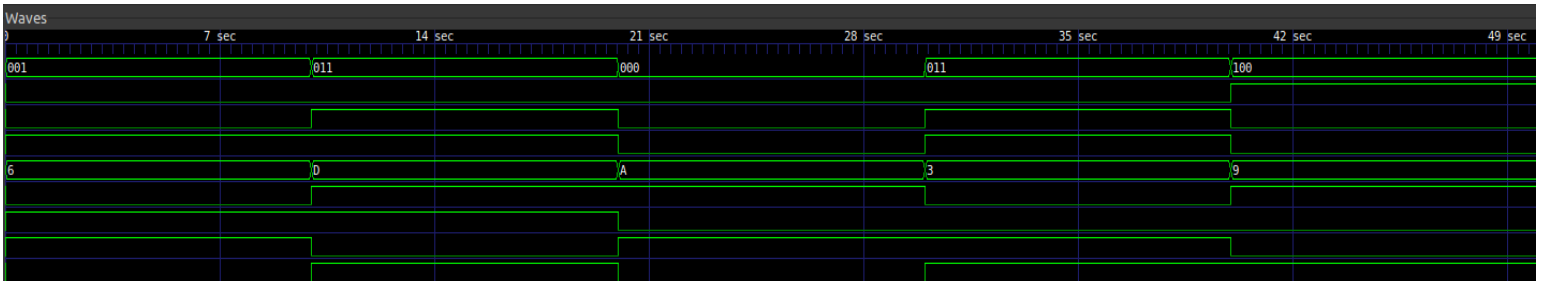
Entrada (a b c d)	Resto (y2 y1 y0)
0000	000
0001	001
0010	010
0011	011
0100	100
0101	000
0110	001
0111	010
1000	011
1001	100

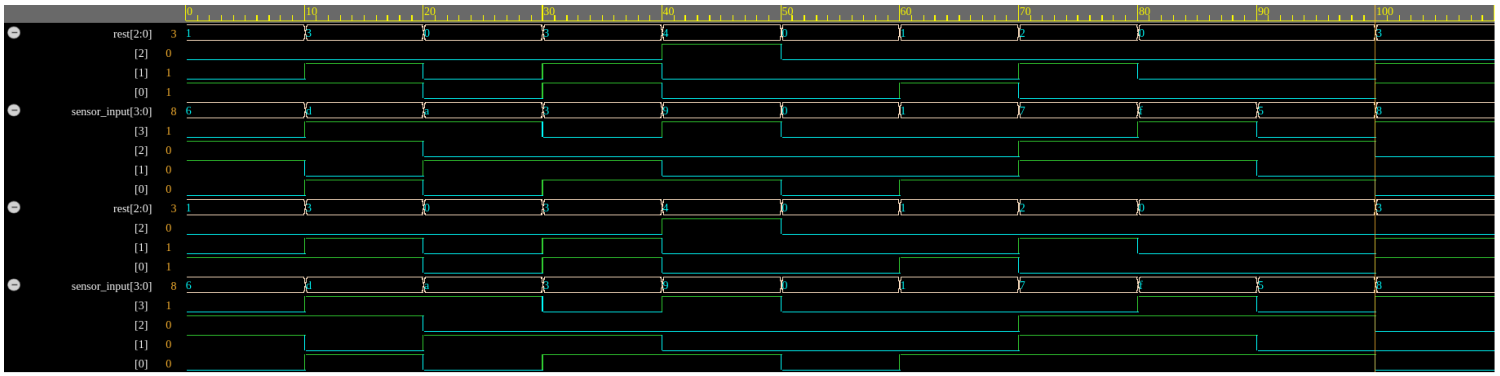
1010	000
1011	001
1100	010
1101	011
1110	100
1111	000

y2: 0100, 1001, 1110
y1: 0010, 0011, 0111, 1000, 1100, 1101
y0: 0001, 0011, 0110, 1000, 1011, 1101

y2= a’bcd’ + ab’cd + a’b’c’d
y1 = ca’b’ + cda’ + abc’ + ac’d’
y0 = abd’ + bc’d’ + a’b’cd’ + a’bcd + ab’c’d

7) Visualização das formas de onda do caso de teste:





Languages & Libraries

Testbench + Design: SystemVerilog/Verilog

UVM / OVM: None

Other Libraries: None, OVL, SVUnit

Enable TL-Verilog
Enable Easier UVM
Enable VUnit

Tools & Simulators

Icarus Verilog 12.0

Compile Options: -Wall -g2012

Run Options: Run Options

Use **run.bash** shell script
Open **EPWave** after run
Show output file after run
Download files after run

Examples

using EDA Playground
VHDL
Verilog/SystemVerilog
UVM

testbench.sv

```
1  module testbench;
2
3      reg [3:0] sensor_input;     // Entrada com 4 bits
4      wire [2:0] rest;            // Saida com 3 bits
5
6      // Instancia o modulo tesla_autopilot
7      tesla_autopilot dut (
8          .sensor_input(sensor_input),
9          .rest(rest)
10     );
11
12     initial begin
13         // Formas de onda em depuracao
14         $dumpfile("test.vcd");
15         $dumpvars(0, testbench);
16
17         // Testa os sensores fornecidos (com diretiva de atraso temporal)
18         sensor_input = 4'b0110; #10; // Sensor 1: 0110 - 6 % 5 = 1
19         $display("Input: %b, Rest: %b", sensor_input, rest);
20
21         sensor_input = 4'b1101; #10; // Sensor 2: 1101 - 13 % 5 = 3
22         $display("Input: %b, Rest: %b", sensor_input, rest);
23
24         sensor_input = 4'b1010; #10; // Sensor 3: 1010 - 10 % 5 = 0
```

Log | Share

```
[2024-11-17 20:31:24 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile test.vcd opened for output.
Input: 0110, Rest: 001
Input: 1101, Rest: 011
Input: 1010, Rest: 000
Input: 0011, Rest: 011
Input: 1001, Rest: 100
Input: 0000, Rest: 000
Input: 0001, Rest: 001
Input: 0111, Rest: 010
Input: 1111, Rest: 000
Input: 0101, Rest: 000
Input: 1000, Rest: 011
testbench.sv:52: $finish called at 110 (1s)
Done
```

Log da simulação

Simulação feita com: testbench.sv
Foram adicionados casos adicionais para garantir maior robustez e efetividade no caso de teste.