Aluno do Embarcatech_37 no IFMA

Nome: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Residência Profissional em FPGA – Turma DELTA - 2025.2

Atividade – Referente ao capítulo 01 da unidade 07

Tema do Capítulo – Linguagem SystemVerilog: Introdução

Prazo dia 05/10/2025 as 23:59

Objetivo: Desenvolver um código utilizando SystemVerilog que represente um somador e subtrator completo.

Enunciado: Desenvolva um código utilizando a plataforma EDA Playground(https://www.edaplayground.com/) em SystemVerilog que represente um somador e subtrator completo. A aplicação deverá ser composta por quatro entradas (M, A, B, Te) e duas saídas (S, Ts), sendo que M representa a variável de controle, onde M = 0 será a seleção para a soma, M = 1 será a seleção para subtração, Te representa o transporte de entrada e Ts representa o transporte de saída.

Faça a compilação, simulação e visualização das formas de onda do projeto utilizando o EDA Playground e verifique os resultados da simulação no GTKWave.

A tabela 1 mostra os valores de entrada e saída do circuito subtrator completo.

M	A	B	Te	S	Ts
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	1	1

A Figura 1 mostra o circuito digital do subtrator completo com as indicações das entradas “M”, “A”, “B”, “Te” e as saídas “S” e “Ts”

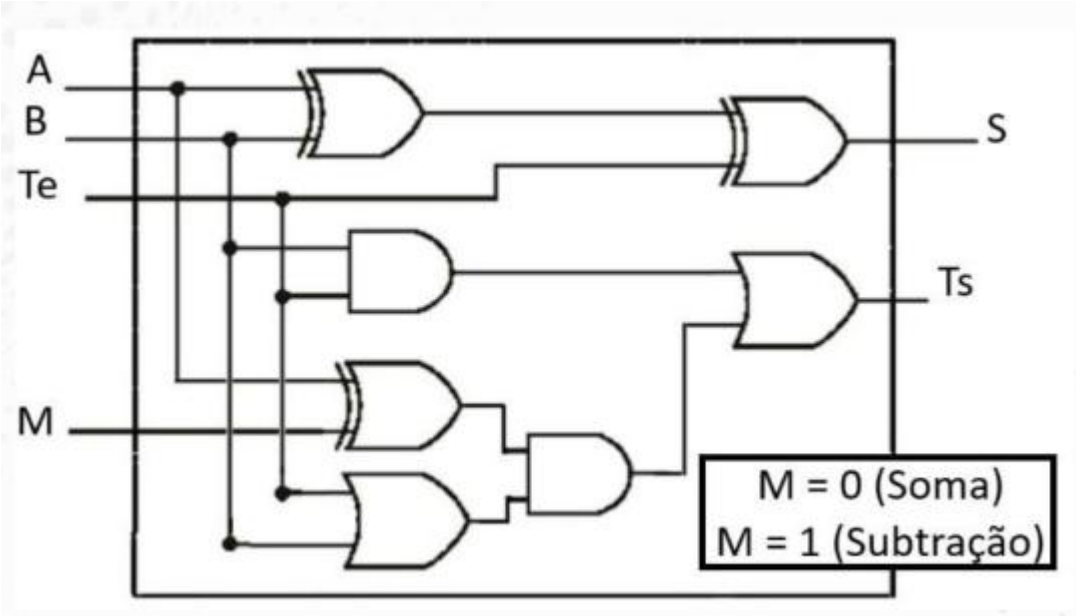


Figura 1. Circuito digital do somador e subtrator completo

Fonte: Adaptado de Eletrônica Digital – Aula 08 para Faculdade (2023), Slideshare. https://pt.slideshare.net/slideshow/eletronica-digital-aula-08-para-faculdade/271949497#45

Instruções:

Seguir os passos descritos no ebook:

1) No terminal da plataforma EDA Playground escreva o código “design.sv” e “testbech.sv”.;
2) Faça a salve e excute os arquivos (“RUN”)
3) Por fim, clique em cada combinação de entradas e saídas na área da forma de onda e será apresentada a combinação das entradas e saídas para análise do projeto e identificação de eventuais inconsistências.

Materiais e Conceitos Envolvidos:

• Computador configurado, contendo:

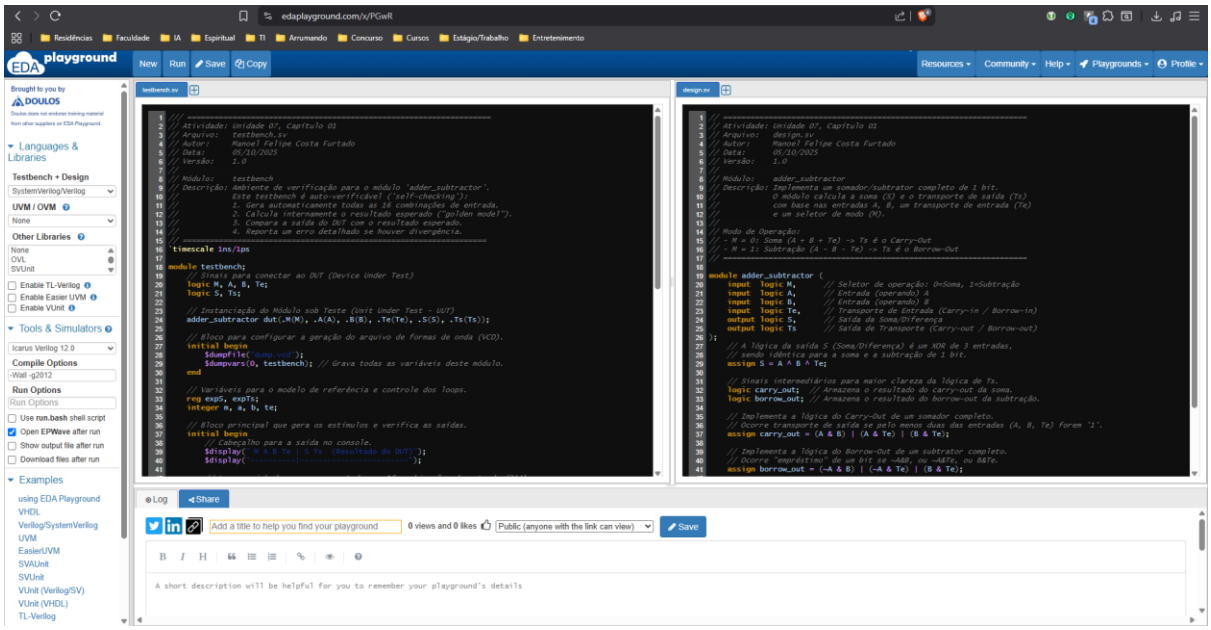
EDA Playground com as seguintes configurações

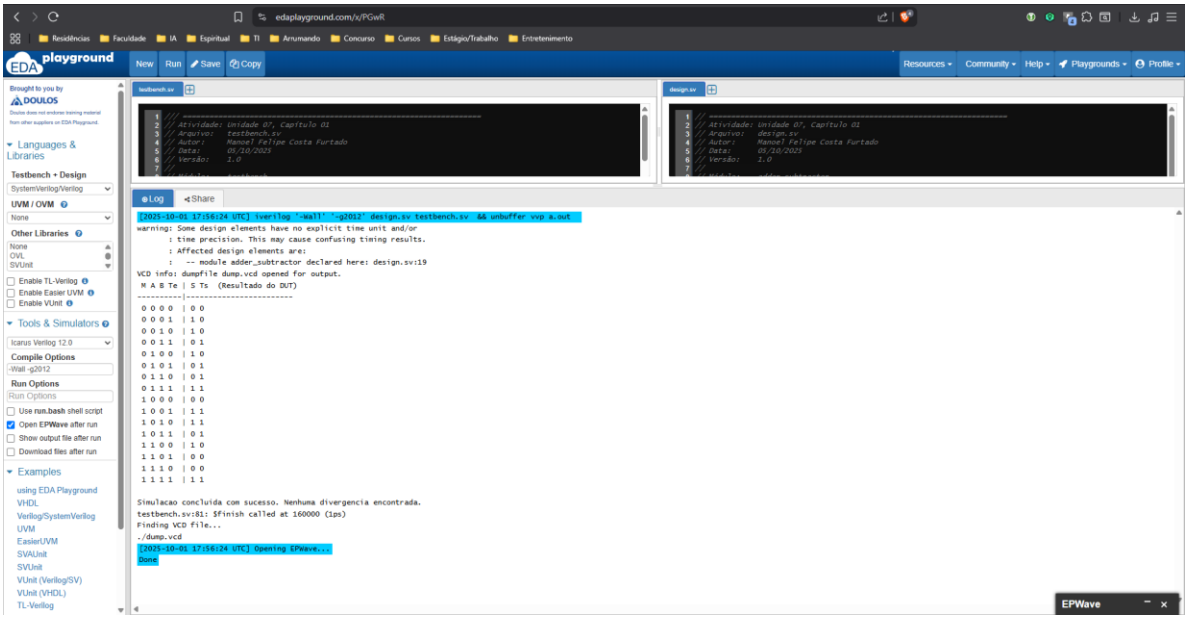
- SystemVerilog/Verilog
- Icarus Verilog 12.0
- Open EPWAve after run

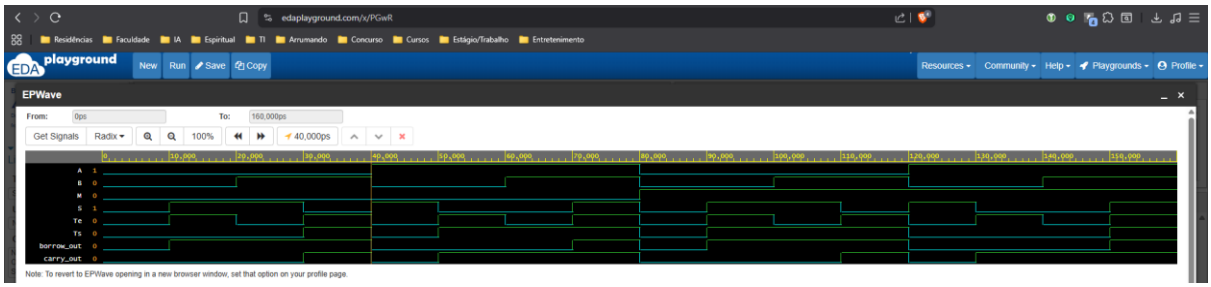
Solução

- GitHub: Segunda_Fase_FPGA/Unidade_07/Cap_01
- Link do Código Completo: Unid_07_Cap_01.zip
- Link do Código na EDA Playground: https://www.edaplayground.com/x/PGwR

1) No terminal da plataforma EDA Playground escreva o código “design.sv” e “testbech.sv”. E Faça a salve e excute os arquivos (“RUN”)

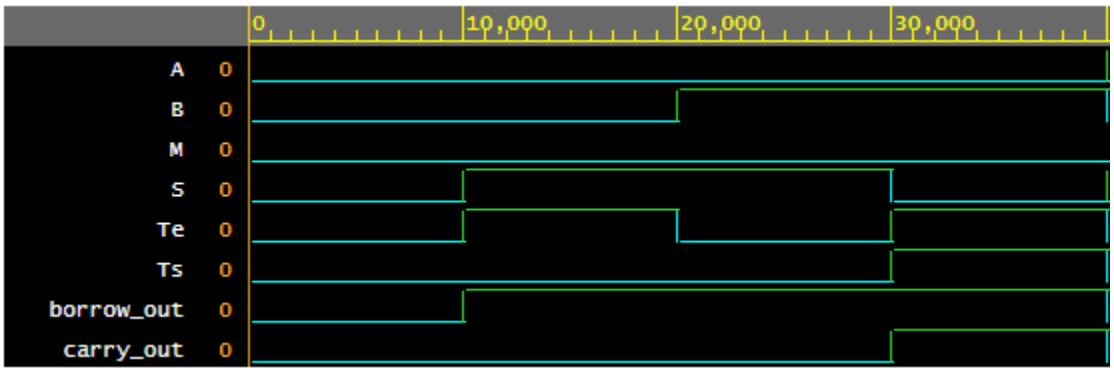


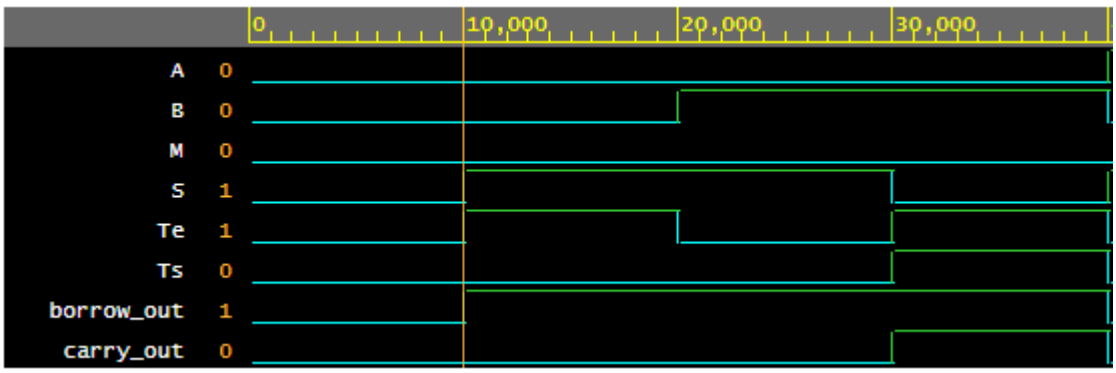


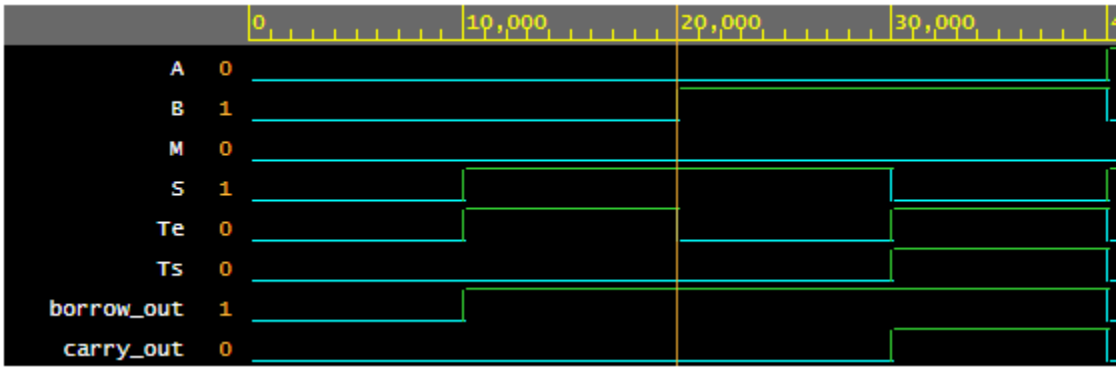


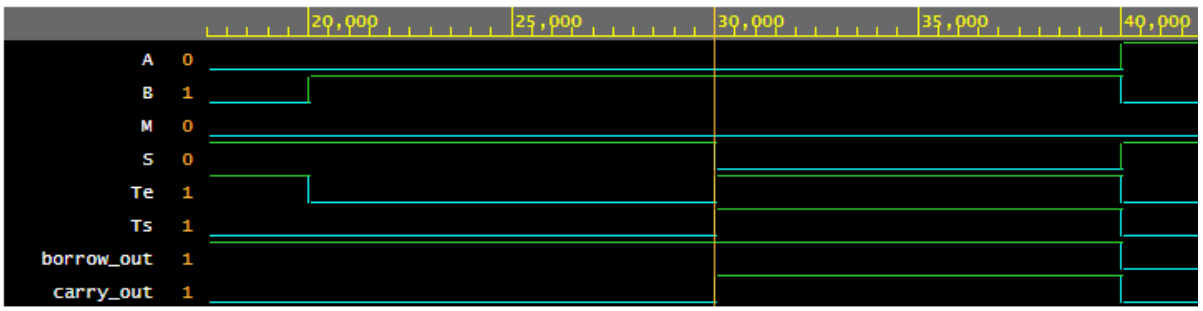
2) Combinações de entradas e saídas na área da forma de onda e será apresentada a combinação das entradas e saídas para análise do projeto e identificação de eventuais inconsistências.

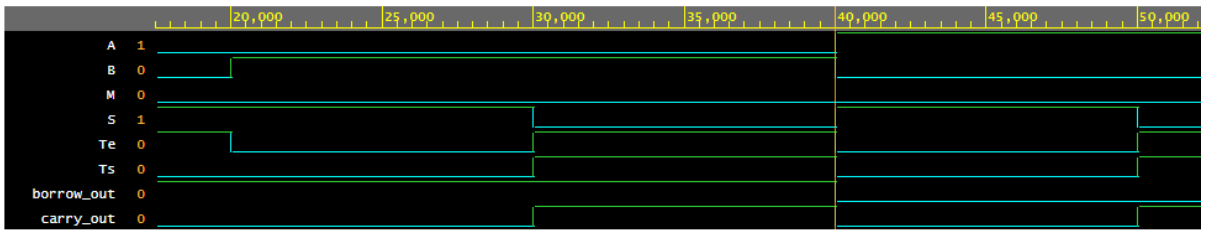
As 16 (2^4) Condições da Tabela Verdade:

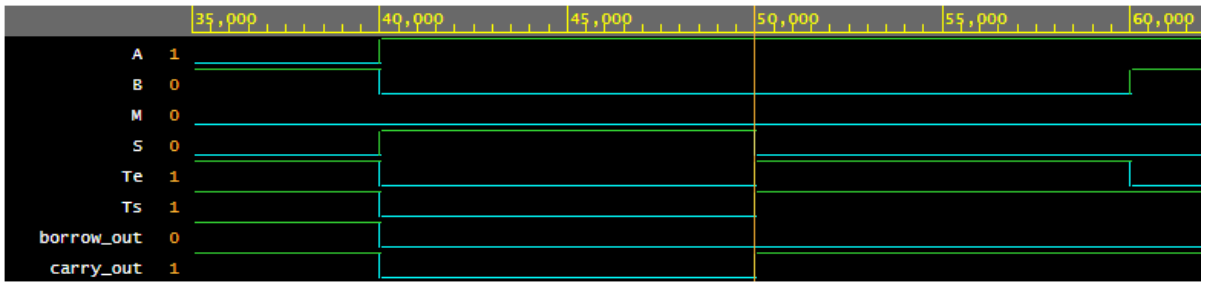


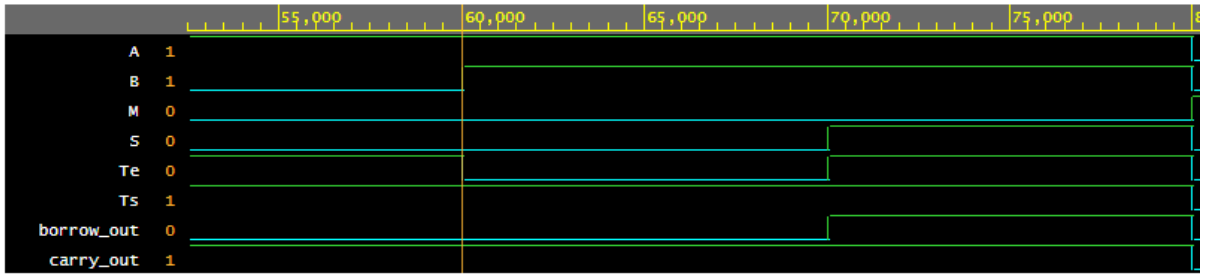

0
10,000
20,000
30,000
A 0
B 0
M 0
S 1
Te 1
Ts 0
borrow_out 1
carry_out 0

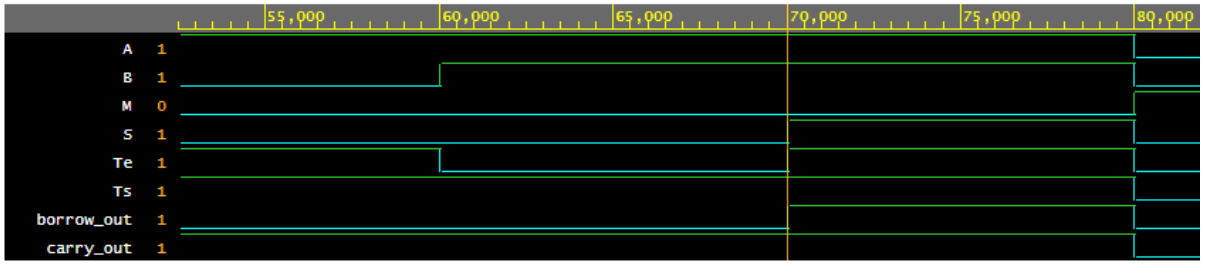

0
10,000
20,000
30,000
A 0
B 1
M 0
S 1
Te 0
Ts 0
borrow_out 1
carry_out 0

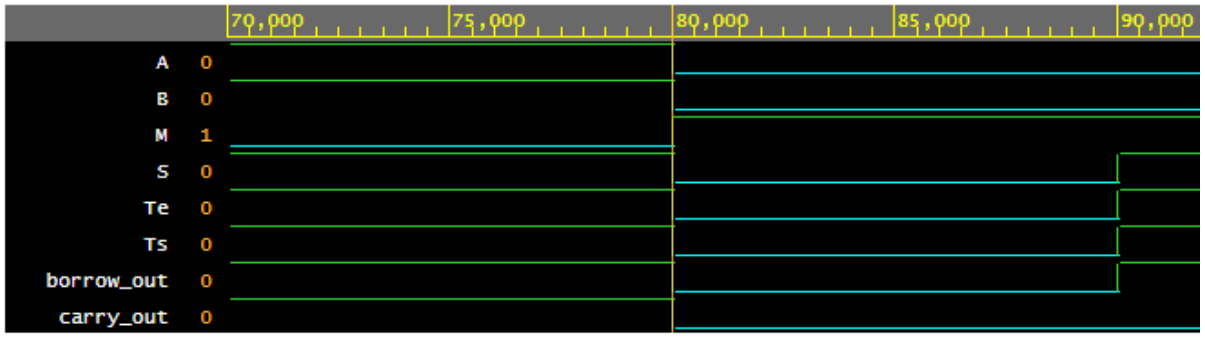

20,000
25,000
30,000
35,000
40,000
A 0
B 1
M 0
S 0
Te 1
Ts 1
borrow_out 1
carry_out 1

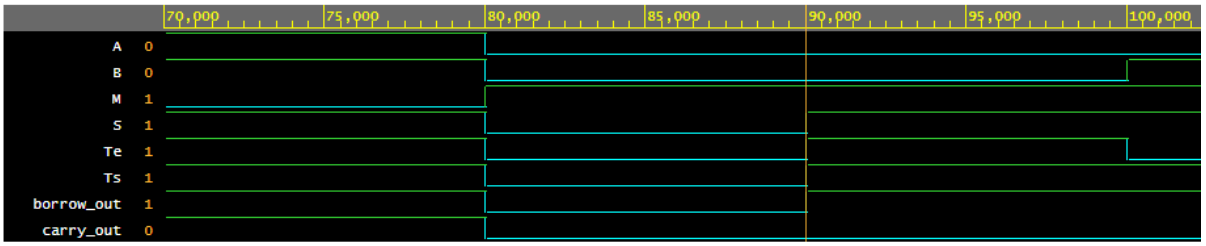

20,000
25,000
30,000
35,000
40,000
45,000
50,000
A 1
B 0
M 0
S 1
Te 0
Ts 0
borrow_out 0
carry_out 0

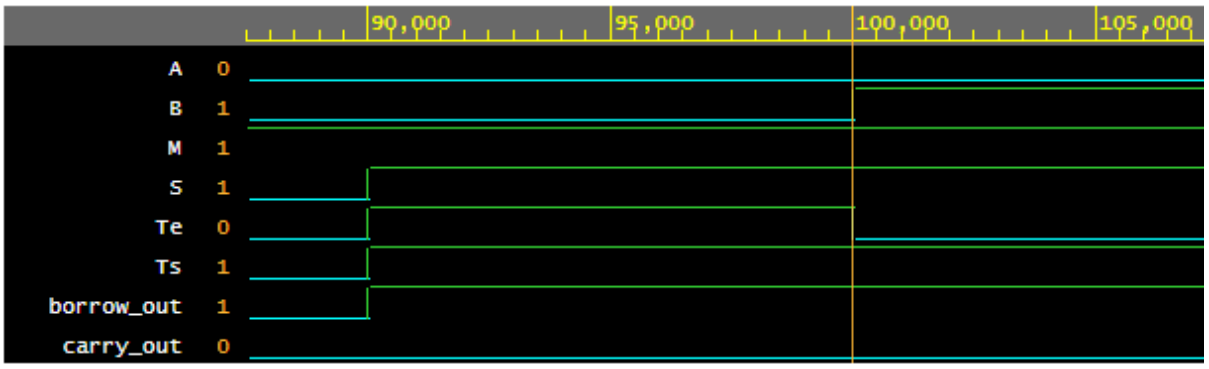

35,000
40,000
45,000
50,000
55,000
60,000
A 1
B 0
M 0
S 0
Te 1
Ts 1
borrow_out 0
carry_out 1

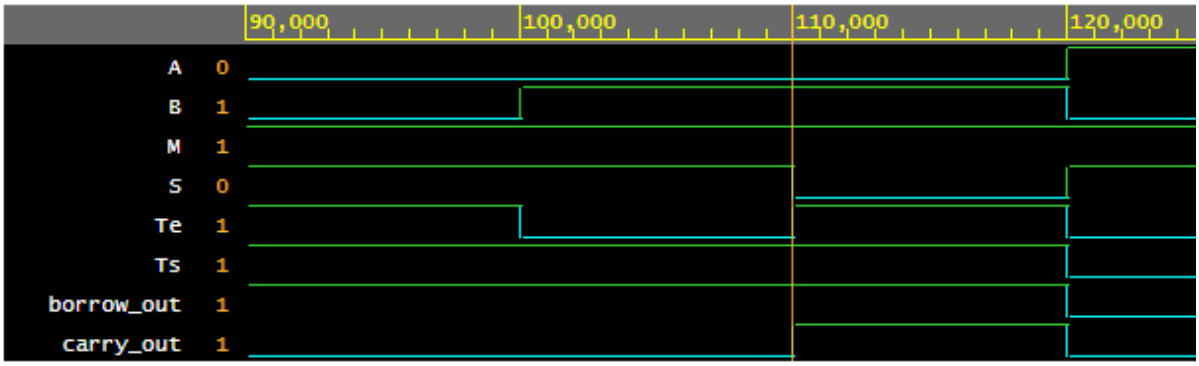

55,000
60,000
65,000
70,000
75,000
A 1
B 1
M 0
S 0
Te 0
Ts 1
borrow_out 0
carry_out 1

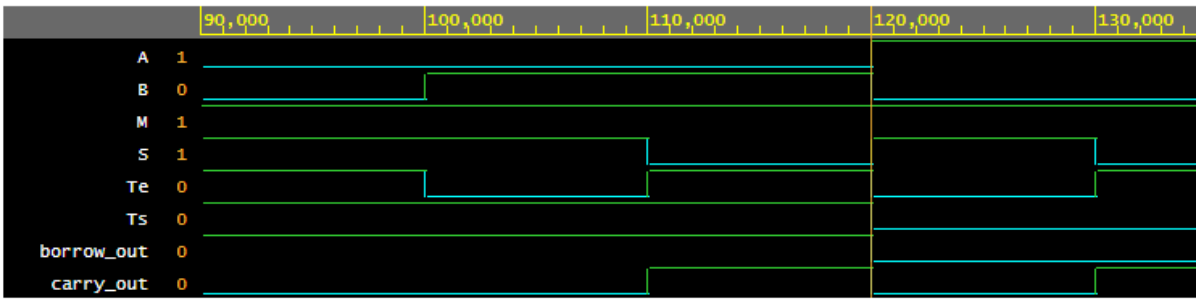

55,000
60,000
65,000
70,000
75,000
80,000
A 1
B 1
M 0
S 1
Te 1
Ts 1
borrow_out 1
carry_out 1

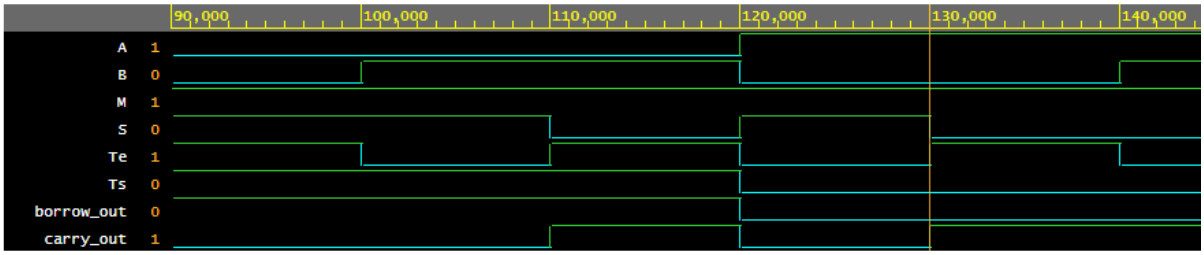

70,000
75,000
80,000
85,000
90,000
A 0
B 0
M 1
S 0
Te 0
Ts 0
borrow_out 0
carry_out 0

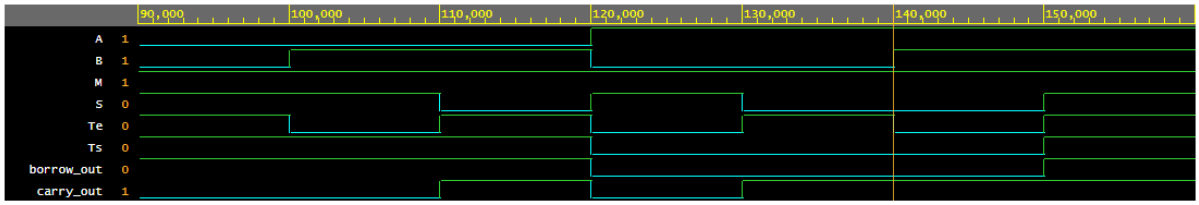

70,000
75,000
80,000
85,000
90,000
95,000
100,000
A 0
B 0
M 1
S 1
Te 1
Ts 1
borrow_out 1
carry_out 0

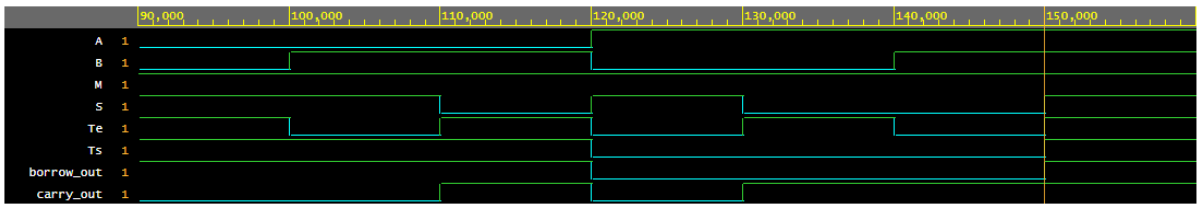

90,000
95,000
100,000
105,000
A 0
B 1
M 1
S 1
Te 0
Ts 1
borrow_out 1
carry_out 0


90,000
100,000
110,000
120,000
A 0
B 1
M 1
S 0
Te 1
Ts 1
borrow_out 1
carry_out 1


90,000
100,000
110,000
120,000
130,000
A 1
B 0
M 1
S 1
Te 0
Ts 0
borrow_out 0
carry_out 0


90,000
100,000
110,000
120,000
130,000
140,000
A 1
B 0
M 1
S 0
Te 1
Ts 0
borrow_out 0
carry_out 1


90,000
100,000
110,000
120,000
130,000
140,000
150,000
A 1
B 1
M 1
S 0
Te 0
Ts 0
borrow_out 0
carry_out 1


90,000
100,000
110,000
120,000
130,000
140,000
150,000
A 1
B 1
M 1
S 1
Te 1
Ts 1
borrow_out 1
carry_out 1

3) Código design.sv

```
// ================================================================
// Atividade: Unidade 07, Capítulo 01
// Arquivo:   design.sv
// Autor:     Manoel Felipe Costa Furtado
// Data:      05/10/2025
// Versão:    1.0
// Módulo:    adder_subtractor
// Descrição: Implementa um somador/subtrator completo de 1 bit.
//            O módulo calcula a soma (S) e o transporte de saída (Ts)
//            com base nas entradas A, B, um transporte de entrada (Te)
//            e um seletor de modo (M).
// Modo de Operação:
// - M = 0: Soma (A + B + Te) -> Ts é o Carry-Out
// - M = 1: Subtração (A - B - Te) -> Ts é o Borrow-Out
// ================================================================

module adder_subtractor (
    input  logic M,      // Seletor de operação: 0=Soma, 1=Subtração
    input  logic A,      // Entrada (operando) A
    input  logic B,      // Entrada (operando) B
    input  logic Te,     // Transporte de Entrada (Carry-in / Borrow-in)
    output logic S,      // Saída da Soma/Diferença
    output logic Ts      // Saída de Transporte (Carry-out / Borrow-out)
);
    // A lógica da saída S (Soma/Diferença) é um XOR de 3 entradas,
    // sendo idêntica para a soma e a subtração de 1 bit.
    assign S = A ^ B ^ Te;

    // Sinais intermediários para maior clareza da lógica de Ts.
    logic carry_out;  // Armazena o resultado do carry-out da soma.
    logic borrow_out; // Armazena o resultado do borrow-out da subtração.

    // Implementa a lógica do Carry-Out de um somador completo.
    // Ocorre transporte de saída se pelo menos duas das entradas (A, B, Te) forem '1'.
    assign carry_out = (A & B) | (A & Te) | (B & Te);

    // Implementa a lógica do Borrow-Out de um subtrator completo.
    // Ocorre "empréstimo" de um bit se ~A&B, ou ~A&Te, ou B&Te.
    assign borrow_out = (~A & B) | (~A & Te) | (B & Te);

    // Multiplexador 2x1: Seleciona a saída Ts com base no modo M.
    // Se M for 0 (soma), a saída é carry_out.
    // Se M for 1 (subtração), a saída é borrow_out.
    assign Ts = (M == 0) ? carry_out : borrow_out;

endmodule
```

4) Código testbench.sv

```
// ================================================================
// Atividade: Unidade 07, Capítulo 01
// Arquivo:   testbench.sv
// Autor:     Manoel Felipe Costa Furtado
// Data:      05/10/2025
// Versão:    1.0
//
// Módulo:    testbench
// Descrição: Ambiente de verificação para o módulo 'adder_subtractor'.
//            Este testbench é auto-verificável ('self-checking'):
//            1. Gera automaticamente todas as 16 combinações de entrada.
//            2. Calcula internamente o resultado esperado ("golden model").
//            3. Compara a saída do DUT com o resultado esperado.
//            4. Reporta um erro detalhado se houver divergência.
// ================================================================
`timescale 1ns/1ps

module testbench;
    // Sinais para conectar ao DUT (Device Under Test)
    logic M, A, B, Te;
    logic S, Ts;

    // Instanciação do Módulo sob Teste (Unit Under Test - UUT)
    adder_subtractor dut(.M(M), .A(A), .B(B), .Te(Te), .S(S), .Ts(Ts));

    // Bloco para configurar a geração do arquivo de formas de onda (VCD).
    initial begin
        $dumpfile("dump.vcd");
        $dumpvars(0, testbench); // Grava todas as variáveis deste módulo.
    end

    // Variáveis para o modelo de referência e controle dos loops.
    reg expS, expTs;
    integer m, a, b, te;

    // Bloco principal que gera os estímulos e verifica as saídas.
    initial begin
        // Cabeçalho para a saída no console.
        $display(" M A B Te | S Ts  (Resultado do DUT)");
        $display("----------|-----------------------");

        // Loops aninhados para varrer todas as 16 combinações de entrada (2^4).
        for (int m=0; m<2; m++) begin
            for (int a=0; a<2; a++) begin
                for (int b=0; b<2; b++) begin
                    for (int te=0; te<2; te++) begin
```

```
                    // 1. Aplica os valores de entrada ao DUT.
                    M = m;
                    A = a; B = b; Te = te;

                    // Aguarda 1ns para a propagação dos sinais no DUT antes de verificar.
                    #1;

                    // 2. Calcula o resultado esperado (modelo de referência).
                    // A lógica esperada para S é um XOR de 3 entradas.
                    expS  = A ^ B ^ Te;

                    // A lógica esperada para Ts depende do modo M.
                    // As expressões são idênticas às do design para consistência.
                    expTs = (M==1'b0) ? ((A & B) | (A & Te) | (B & Te)) // Carry-out esperado
                                      : ((~A & B) | (~A & Te) | (B & Te));   // Borrow-out esperado

                    // 3. Compara a saída do DUT com o resultado esperado e reporta erro se divergirem.
                    if (S !== expS || Ts !== expTs) begin
                        $error("ERRO! M=%b A=%b B=%b Te=%b | S=%b(exp:%b) Ts=%b(exp:%b)",
                               M,A,B,Te, S,expS, Ts,expTs);
                    end

                    // Imprime os valores atuais do DUT no console para acompanhamento.
                    $display(" %1d %1d %1d %1d  | %1d %1d", M,A,B,Te,S,Ts);

                    // Aguarda mais 9ns para completar um ciclo de 10ns por vetor de teste.
                    // Isso cria uma forma de onda limpa e organizada no GTKWave.
                    #9;
                end
            end
        end
    end

    $display("\nSimulacao concluida com sucesso. Nenhuma divergencia encontrada.");
    $finish;
  end
endmodule
```

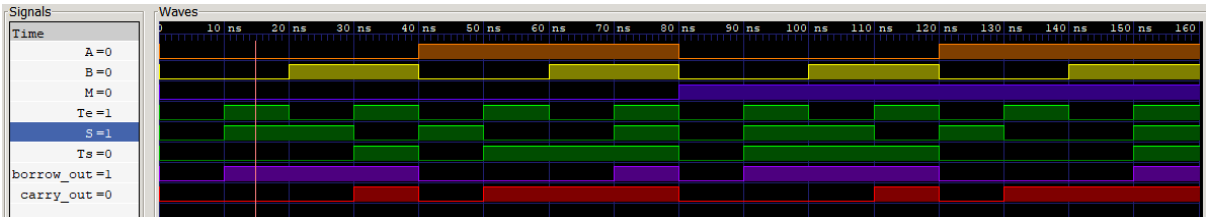
O conteúdo a seguir é extra. Testei compilando localmente no notebook. Usando o compilado IcarusVerilog e o gtkwave.

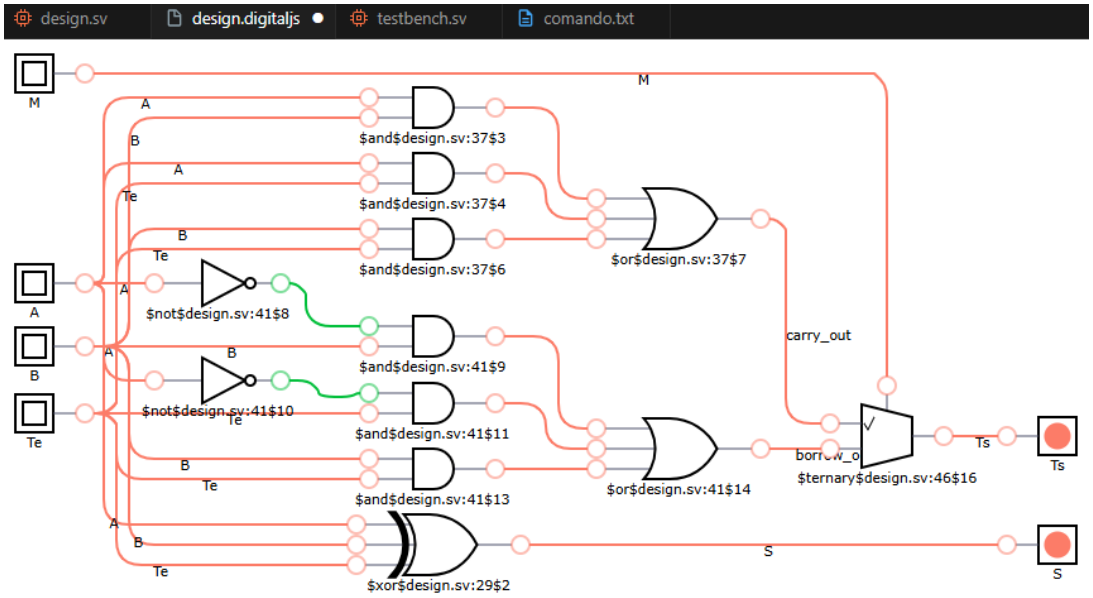
```
PS C:\Users\manoe\OneDrive\TI\EmbarcaTech\Segunda Fase\02_Segunda_Fase_FPGA\Unid_07\Atividade\Unid_07_Cap_01> iverilog -v
Icarus Verilog version 12.0 (devel) (s20150603-1539-g2693dd32b)

Copyright (c) 2000-2021 Stephen Williams (steve@icarus.com)
```

```
PS C:\Users\manoe\OneDrive\TI\EmbarcaTech\Segunda Fase\02_Segunda_Fase_FPGA\Unid_07\Atividade\Unid_07_Cap_01> gtkwave -v

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI
```

```
PS C:\Users\manoe\OneDrive\TI\EmbarcaTech\Segunda Fase\02_Segunda_Fase_FPGA\Unid_07\Atividade\Unid_07_Cap_01> iverilog -g2012 -o sim design.sv testbench.sv
PS C:\Users\manoe\OneDrive\TI\EmbarcaTech\Segunda Fase\02_Segunda_Fase_FPGA\Unid_07\Atividade\Unid_07_Cap_01> vvp sim
VCD info: dumpfile dump.vcd opened for output.
 M A B Te | S Ts  (Resultado do DUT)
----------|-----------------------
 0 0 0 0  | 0 0
 0 0 0 1  | 1 0
 0 0 1 0  | 1 0
 0 0 1 1  | 0 1
 0 1 0 0  | 1 0
 0 1 0 1  | 0 1
 0 1 1 0  | 0 1
 0 1 1 1  | 1 1
 1 0 0 0  | 0 0
 1 0 0 1  | 1 1
 1 0 1 0  | 1 1
 1 0 1 1  | 0 1
 1 1 0 0  | 1 0
 1 1 0 1  | 0 0
 1 1 1 0  | 0 0
 1 1 1 1  | 1 1

Simulacao concluida com sucesso. Nenhuma divergencia encontrada.
testbench.sv:81: $finish called at 160000 (1ps)
```

```
# 1. Compilar os arquivos do projeto (habilitando SystemVerilog 2012)

# -o sim: define o nome do arquivo de saída compilado como "sim"

iverilog -g2012 -o sim design.sv testbench.sv


# 2. Executar a simulação

# Este comando roda o testbench e gera o arquivo "dump.vcd" com as formas de onda.

vvp sim


# 3. Visualizar as formas de onda

# Abre o arquivo gerado no passo anterior com o programa GTKWave.

gtkwave dump.vcd
```





Esse é circuito sintetizado direto do código design.sv

Em vez de tratar o problema como um único e complexo circuito de 4 entradas (M, A, B, Te), a solução mais inteligente é usar a estratégia de "dividir para conquistar", onde M atua como o sinal de controle. Logo solução foi usar M como um multiplexador, ajudando na solução separando a soma da subtração.

Saida S: S = a^B^Te

A operação OU Exclusivo (XOR) descreve perfeitamente e da forma mais simples possível o comportamento da soma de bits.

A variável M não tem efeito nenhum sobre a lógica da saída S

A saída Ts é uma única saída, mas o valor que ela assume depende do modo de operação (M): O circuito calcula duas lógicas em paralelo, o tempo todo:

- carry_out = (A & B) | (A & Te) | (B & Te); -> O "vai-um" da soma.
- borrow_out = (~A & B) | (~A & Te) | (B & Te); -> O "empresta-um" da subtração.

A linha final assign Ts = (M == 0) ? carry_out : borrow_out; atua como um Multiplexador (MUX). A variável M é a chave seletora:

- Se M for 0, o MUX conecta a saída Ts ao resultado de carry_out.
- Se M for 1, o MUX conecta a saída Ts ao resultado de borrow_out.

1) M como Sinal de Controle (Modo): Você percebeu corretamente que M não é um dado como A e B. Ele é um sinal de controle que define o "modo de operação" do circuito: Soma ou Subtração.
2) Separando o Problema: Ao usar M para separar as funções, você transforma um problema grande em dois problemas menores e muito mais simples:
   - Quando M = 0: O circuito só precisa se preocupar em ser um Somador Completo. As entradas são A, B, Te e as saídas são Soma e Carry-out.
   - Quando M = 1: O circuito só precisa se preocupar em ser um Subtrator Completo. As entradas são A, B, Te e as saídas são Diferença e Borrow-out.
3) Usando o Multiplexador para Unir as Soluções: Depois de resolver os dois problemas menores separadamente, o multiplexador (controlado por M) age como uma "chave" que seleciona qual resultado será enviado para a saída final.
   - Para a saída S, descobrimos que a lógica (A ^ B ^ Te) é a mesma nos dois modos, então nem precisamos de um multiplexador para ela.
   - Para a saída Ts, a lógica é diferente. Por isso, calculamos o carry_out e o borrow_out e usamos M para decidir qual deles se torna o Ts.

Para a Saída S:

A	B	Te	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Expressão Booleana Simplificada

A'B'C + A'BC' + AB'C' + ABC

Para a saída S: A' B' C + A' B C' + A B' C' + A B C

Pelo Mapa de Karnaugh não foi possível reduzir o circuito contudo, podemos simplificar usando álgebra de boole.

Passo 1: Reorganizar os termos para agrupar fatores comuns

- s = (A' B' Cin + A' B Cin') + (A B' Cin' + A B Cin)

Passo 2: Fatorar os termos comuns (A' e A)

Colocar A' em evidência no primeiro grupo e A em evidência no segundo grupo.

- s = A' (B' Cin + B Cin') + A (B 'Cin' + B Cin)

Passo 3: Identificar as expressões de XOR e XNOR

- Porta XOR (OU Exclusivo): A definição de A $\oplus$ B é A' B + A B'.

- Porta XNOR (NÃO-OU Exclusivo): A definição de (A $\oplus$ B)' é A' B' + A B.
- O termo (B' Cin + B Cin') é exatamente a definição de B $\oplus$ Cin.
- O termo (B' Cin' + B Cin) é exatamente a definição da porta XNOR, ou seja, (B $\oplus$ Cin)'.

Passo 4: Substituir as expressões de XOR e XNOR na equação

- s = A' (B $\oplus$ Cin) + A (B $\oplus$ Cin)'

Passo 5: Reconhecer a forma final da porta XOR

Seja Y = (B $\oplus$ Cin), então Y' = (B $\oplus$ Cin)'.

- s = A'Y+AY'

Passo 6: Substituir de volta e chegar à conclusão

- s = A $\oplus$ Y $\Longrightarrow$ s = A $\oplus$ (B $\oplus$ Cin)
- s = A $\oplus$ B $\oplus$ Cin

Em SystemVerilog, o operador para XOR é o ^, então a expressão final é s = a^B^Te;

Para entrada Te

A entrada Te é o que torna o nosso circuito um somador/subtrator completo. Ela permite que vários desses blocos de 1 bit sejam conectados em cascata para criar somadores/subtratores de múltiplos bits (como 4, 8, ou 32 bits).

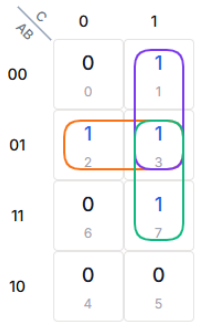
Imagine somar 8 bits:

1) O primeiro somador (para o bit menos significativo) começa com Te = 0.
2) A saída Ts (carry-out) do primeiro somador é conectada na entrada Te (carry-in) do segundo somador.
3) A saída Ts do segundo é conectada na entrada Te do terceiro, e assim por diante.

Saída Ts

O Ts também tem um circuito simplificado, mas a maneira como chegamos nessa simplificação é diferente. Não usamos um único Mapa de Karnaugh de 4 entradas. Em vez disso, usamos a estratégia de "dividir para conquistar" que conversamos antes.

A	B	Te	S	Ts
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



Expressão simplificada para o "empresta-um" (borrow_out) da subtração:

$$Ts = (\neg A \cdot B) + (\neg A \cdot Te) + (B \cdot Te)$$