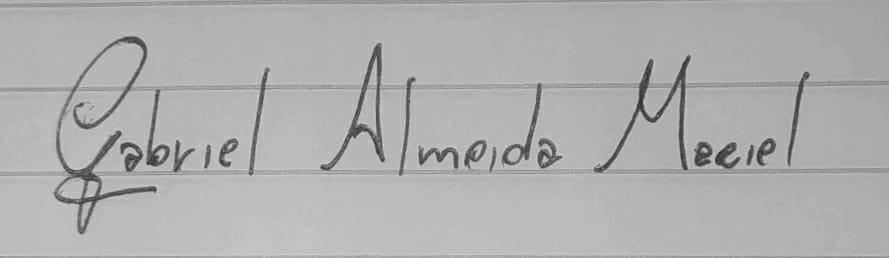


**UNIVERSIDADE FEDERAL DE PERNAMBUCO - UFPE**

**CENTRO DE INFORMÁTICA**

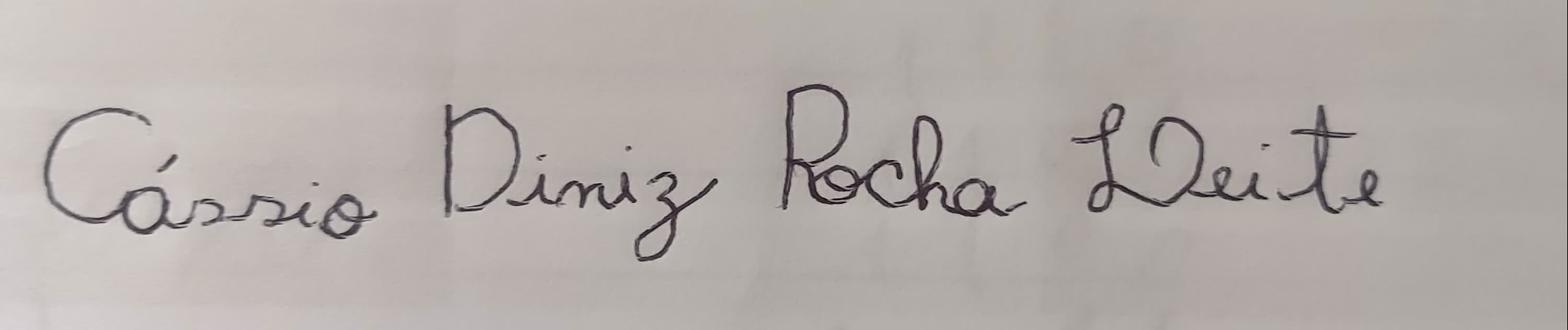
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**  
   
Os integrantes do Grupo 04 da APS 3 confirmam a participação dos membros abaixo na resolução da atividade.



GABRIEL ALMEIDA MACIEL



VITÓRIA DAS DORES DA SILVA NASCIMENTO



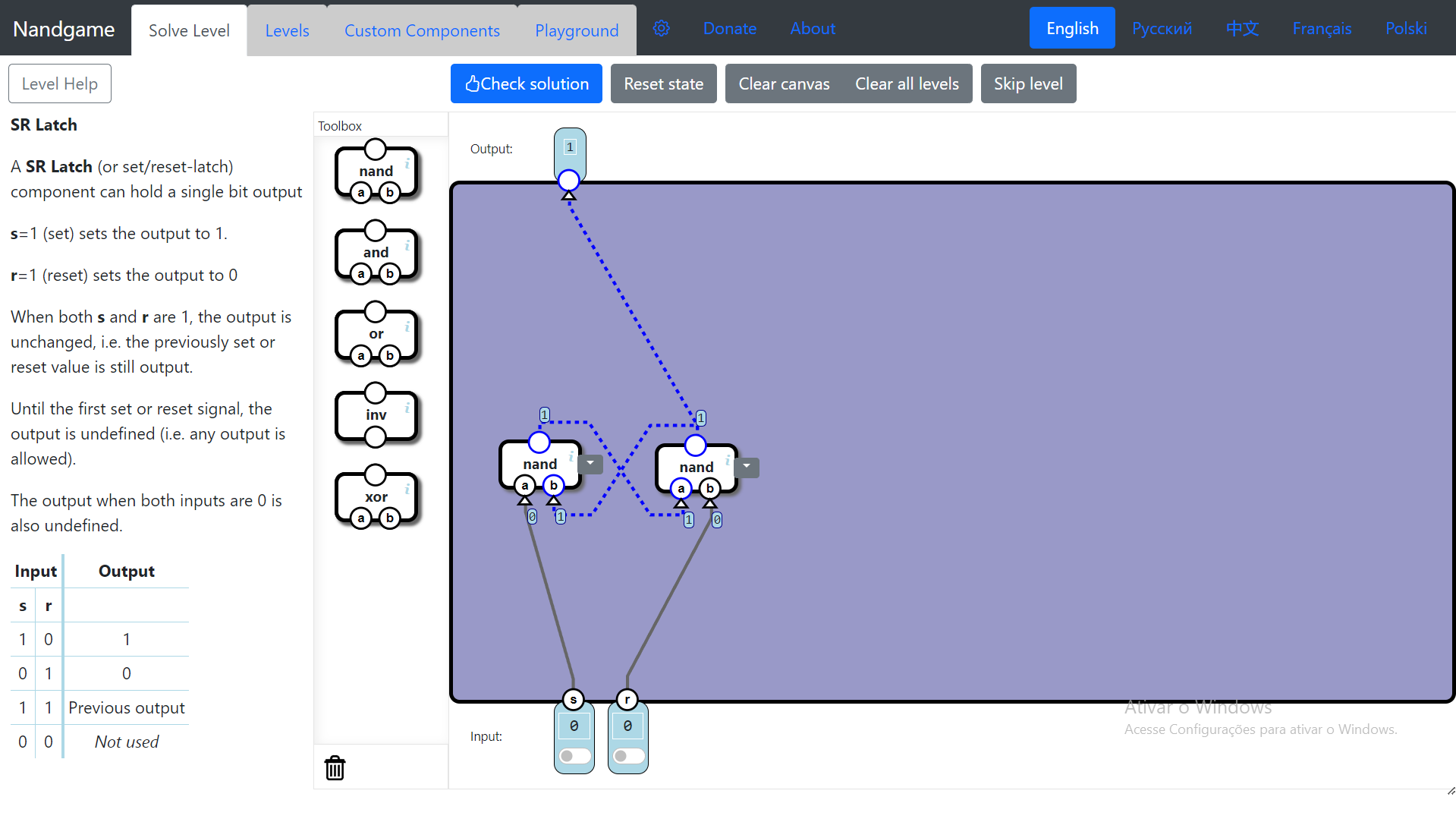
CASSIO DINIZ ROCHA LEITE

**APS-3: Nível do Sistema de Computador**

**JOGAR**

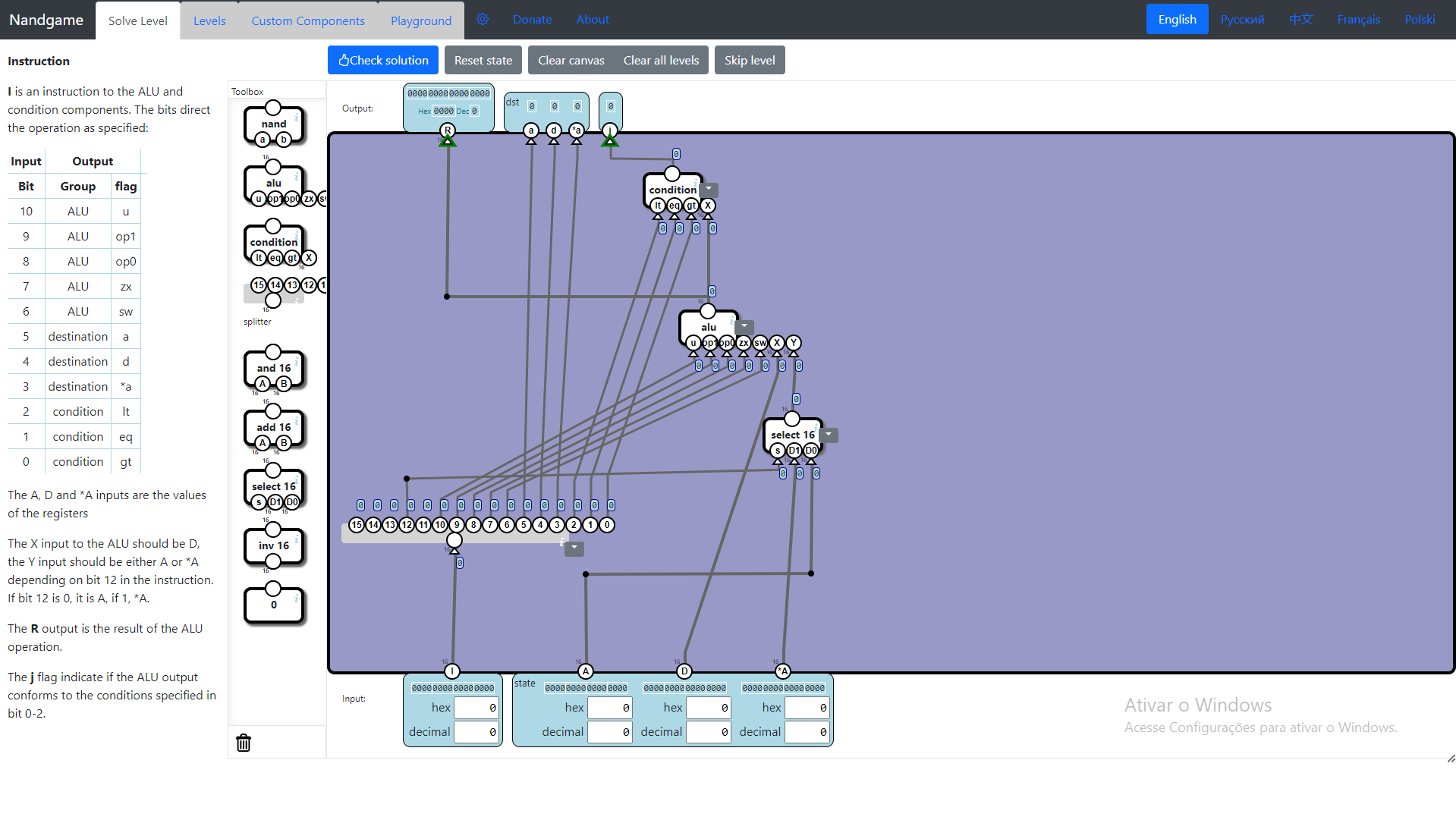
NANDGAME

**COMBINED MEMORY**



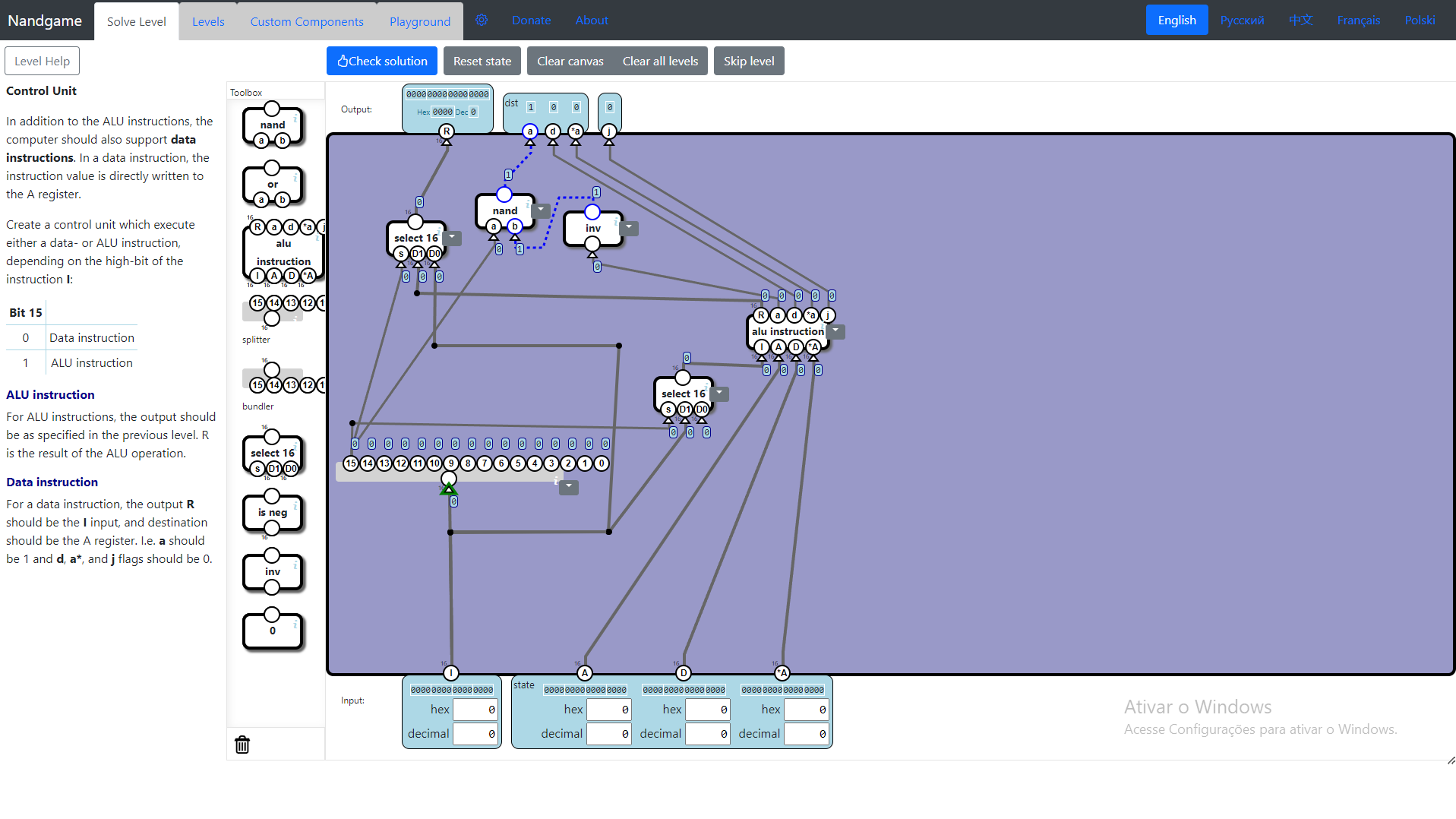
O SR latch utiliza duas portas NAND interligadas. Possui duas entradas, Set (S) e Reset (R). Quando S é 1 e R é 0, Q é definido como 1; quando S é 0 e R é 1, Q é resetado para 0. Quando ambas as entradas são 0, o latch mantém seu estado atual, enquanto a condição S = 1 e R = 1 é inválida, pois leva a um estado indeterminado. Assim, o SR latch armazena um bit de informação, alternando entre os estados de set e reset conforme as entradas.

**INSTRUTION**

****

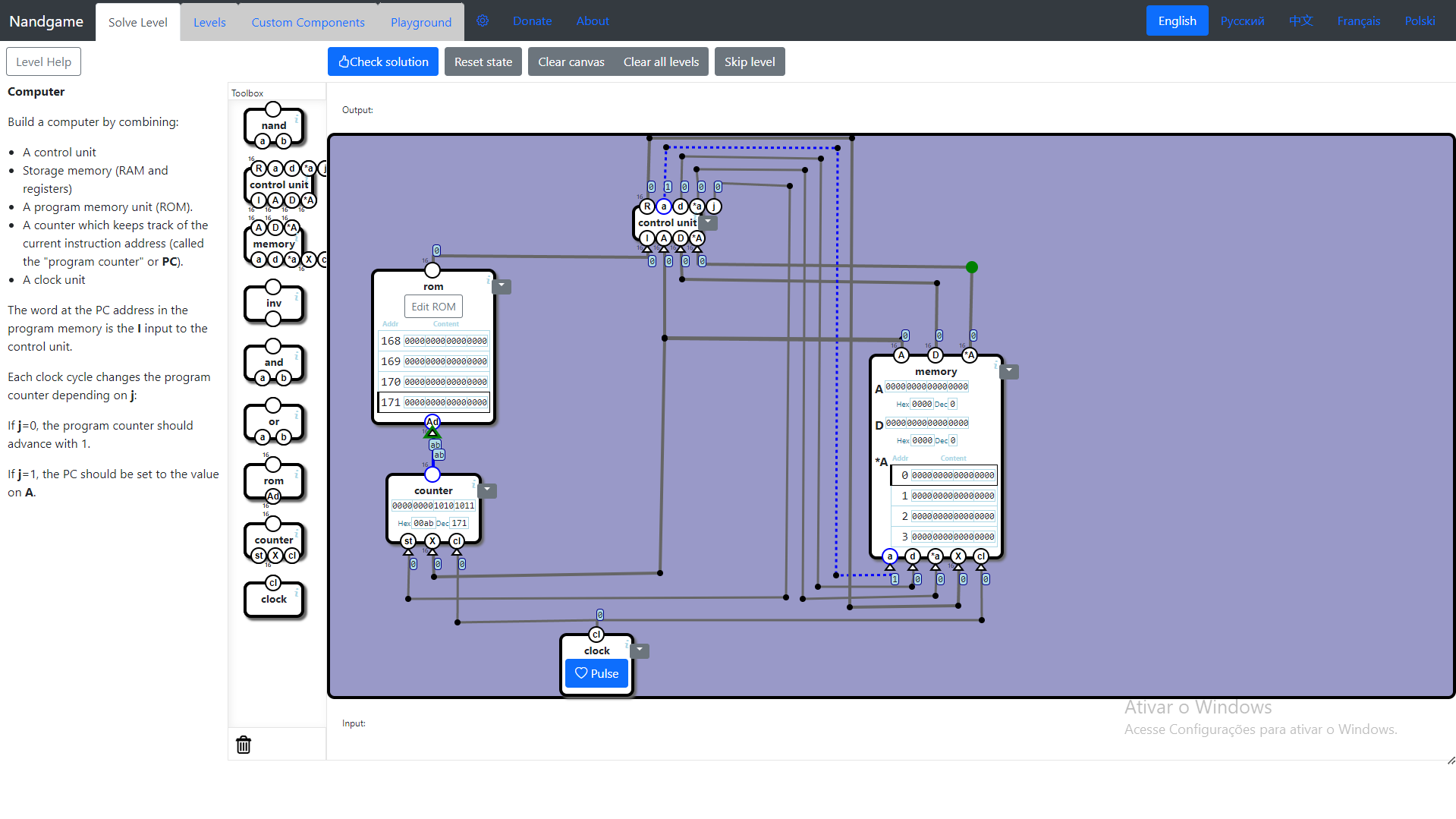
O D latch foi implementado usando um seletor (ou mux) retroalimentado, uma forma não tão convencional de fazer, mas que é equivalente logicamente. D0 recebe a saída do seletor, que também é o bit armazenado, e D1 recebe ST (store). Quando ST é 0, o valor da saída é o mesmo armazenado, e quando é 1, ele recebe a entrada D e a armazena.

**CONTROL UNIT**



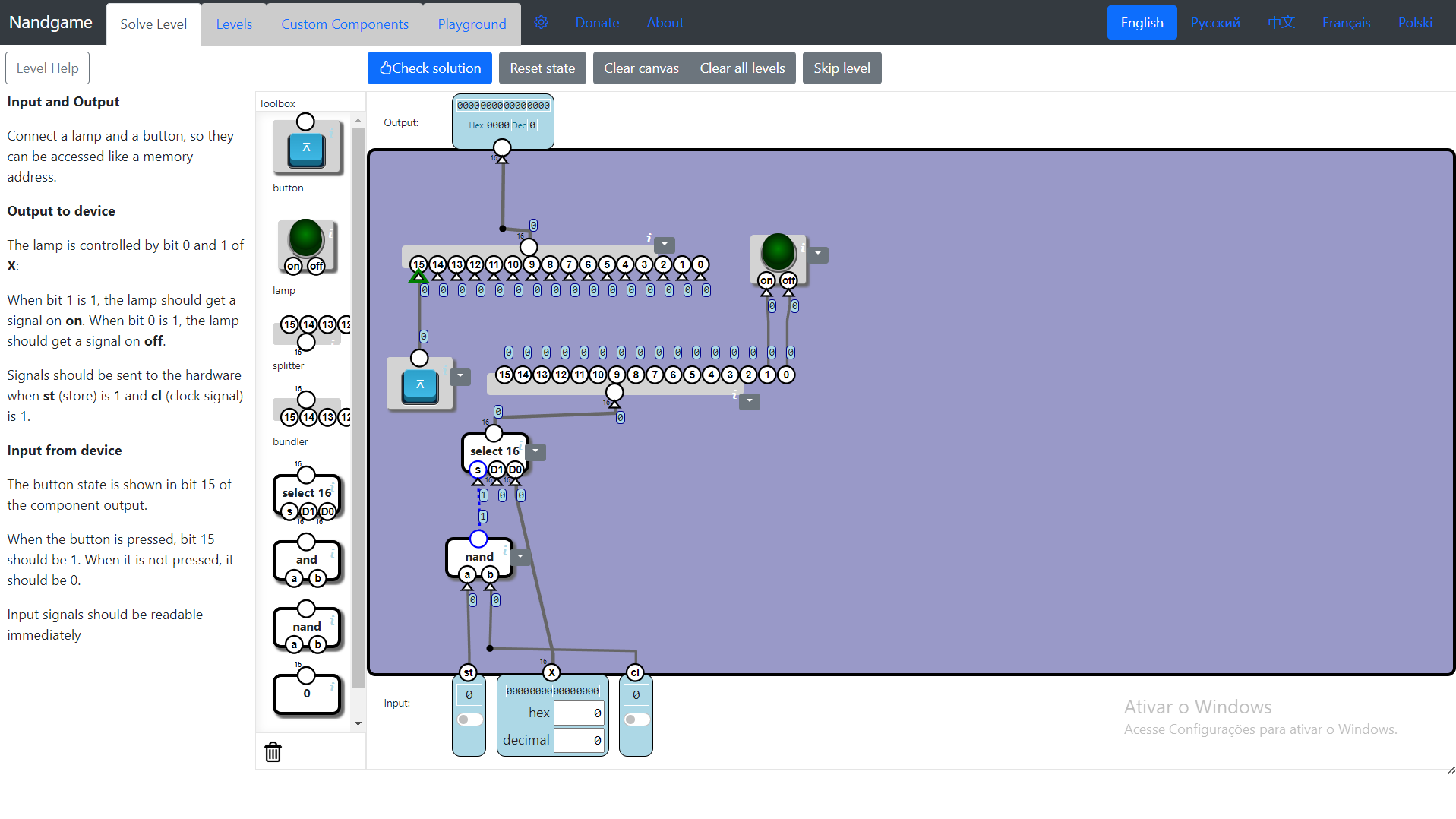
O Flip-Flop armazena e tem como saída um bit; o bit armazenado não muda imediatamente, sendo necessário esperar o clock. Quando o clock passa de 0 para 1, o bit armazenado muda e, consequentemente, a saída também. É implementado usando dois latches D. O primeiro latch recebe o D na entrada D, e o store do primeiro latch recebe a saída de uma porta AND que recebe o CL (clock) e a entrada ST. O segundo latch recebe a saída do primeiro em D e também recebe a saída de uma porta NAND; mas, desta vez, a porta NAND tem CL negado e ST como entradas.

**COMPUTER**

****

O registrador é implementado usando Flip-Flops. Para um registrador de 2 bits, são necessários dois Flip-Flops. Ambos os Flip-Flops recebem ST na entrada ST e CL na entrada CL. O primeiro Flip-Flop recebe D0 e tem como saída o primeiro bit do registrador; já o segundo recebe D1 e sua saída é o segundo bit do registrador.

**INPUT AND OUTPUT**

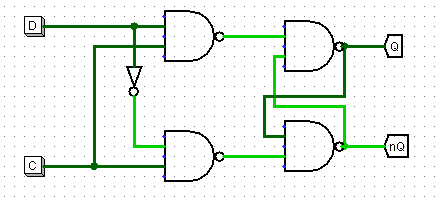
****

Para o contador, usaremos um registrador. A entrada ST será sempre ativa para que ele sempre registre o próximo valor. A entrada X do contador receberá a saída de um seletor que deve selecionar entre o valor X do input ou o valor atual do contador incrementado em 1. Quem fará a seleção é o bit ST. Quando ST é 0, teremos o valor incrementado, e quando é 1, teremos o valor do input X.

**PARTE II: DESENVOLVER**

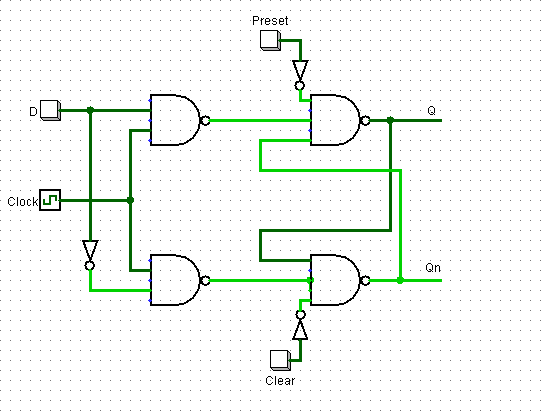
DIAGRAMA DE BLOCOS

**D LATCH**

****

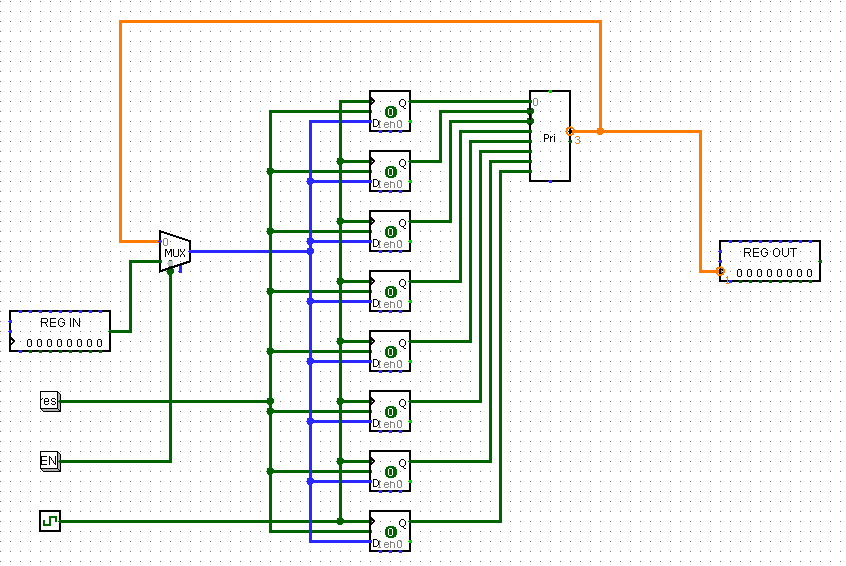
A implementação de um D Latch deve ser feita dessa forma (o que não ocorre exatamente no verilog por ter problemas com retro alimentação), mantendo o bit armazenado e resetando e setando conforme a entrada D.

**D FLIP FLOP**

****

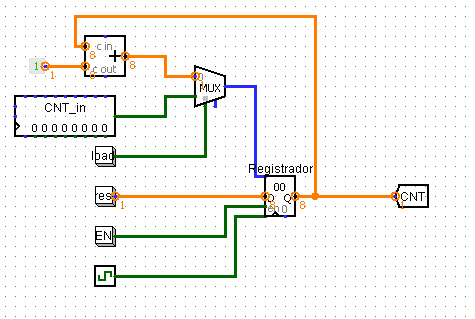
Implementação de forma parecida com o latch, também temos mais duas entradas: preset e reset, são assincronas e podem ser selecionadas a qualquer momento, diferente da entrada D que só causa alteração em momentos específicos do clock.

**REGISTER**

****

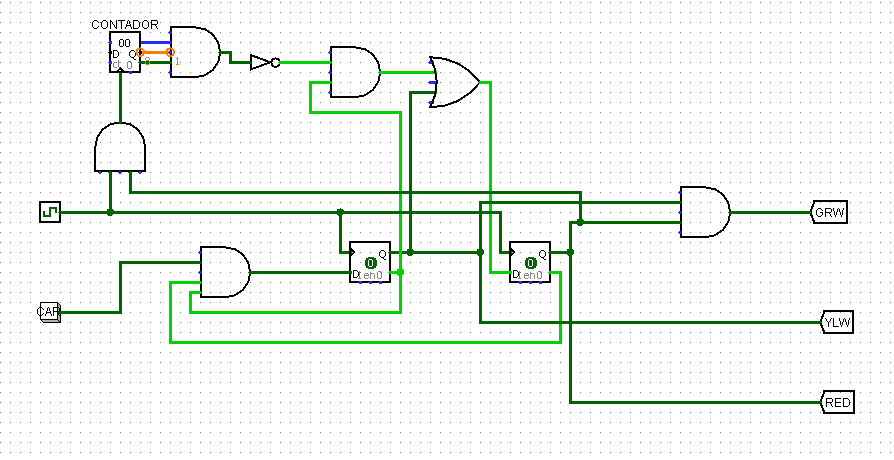
O Registrador implementado dessa forma funciona com uma sequência de flip-flops e usa um mux para definir se a entrada D dos flip-flops será o REG in ou o valor atual do registrador incrementado 1.

**COUNTER**

****

O contador utiliza principalmente um registrador para fazer sua contagem, o próximo número registrado será o valor do contador incrementado 1 ou o do CNT in, dependendo se o load está ativo ou não, é usado um mux para selecionar qual dos dois será usado.

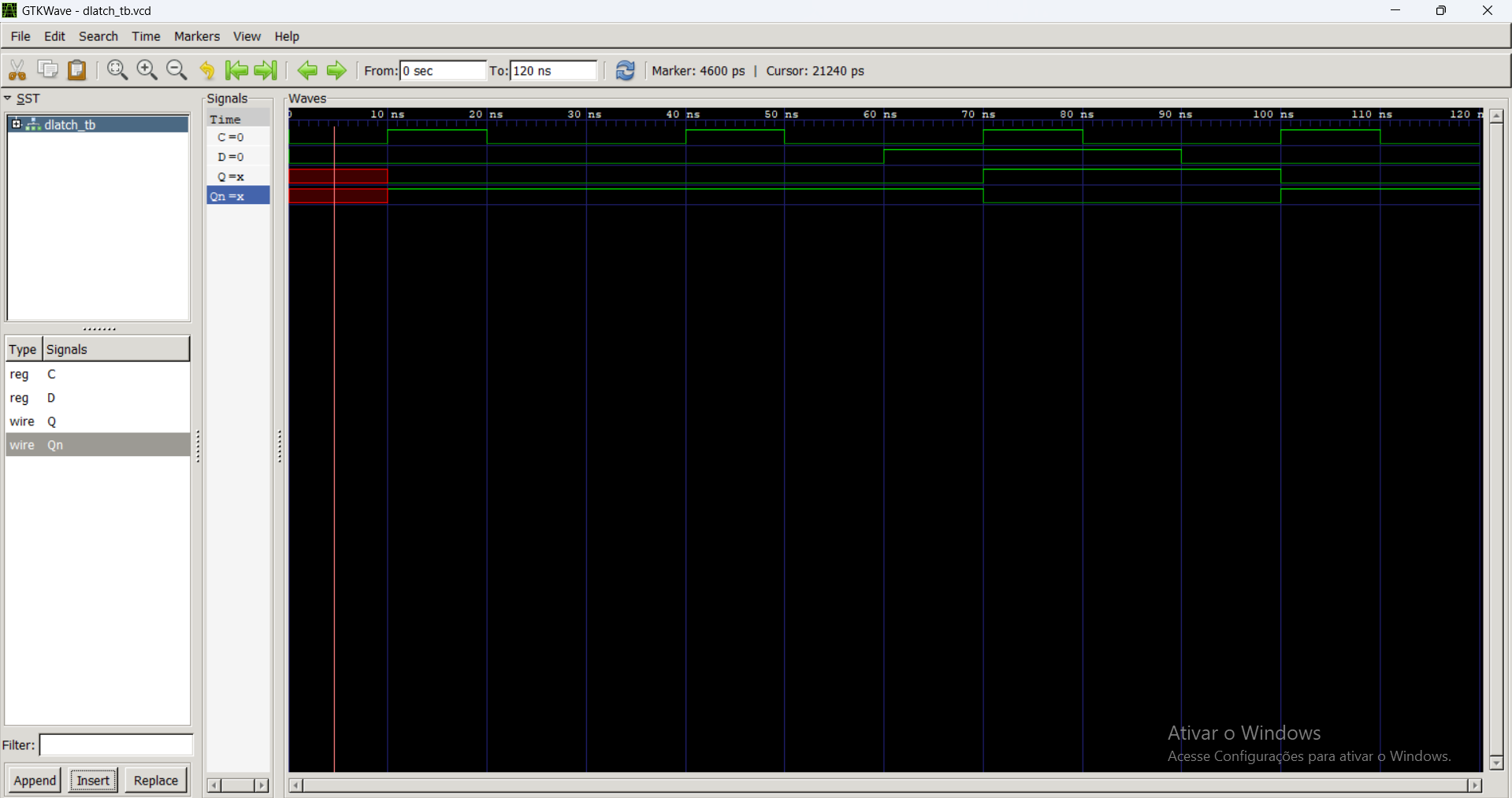
**TRAFFIC LIGHT CONTROL**

****

Para o controlador de semáforo teremos 3 bits de saída, imaginando como uma máquina de estados finitos, o estado GRW = 1 é quando o semáforo está verde, YLW = 1, quando está amarelo e RED = 1, quando está vermelho, não existem caso em que mais de dois estarão ativos pois o circuito não permite nenhuma combinação para chegar nesse estado. quando CAR é ativado, caso o primeiro e o segundo Flip-Flop estejam desativados ele passa para o próximo estado (de 00 -> 10) que é o estado em que o semáforo está amarelo, logo após o proximo pulso do semáforo ele passa para vermelho (11) onde se inicia uma contagem do counter, quando o counter chegar á 15 (00001110) ele volta para o estado inicial (00) em que GRW = 1.

TESTBENCH

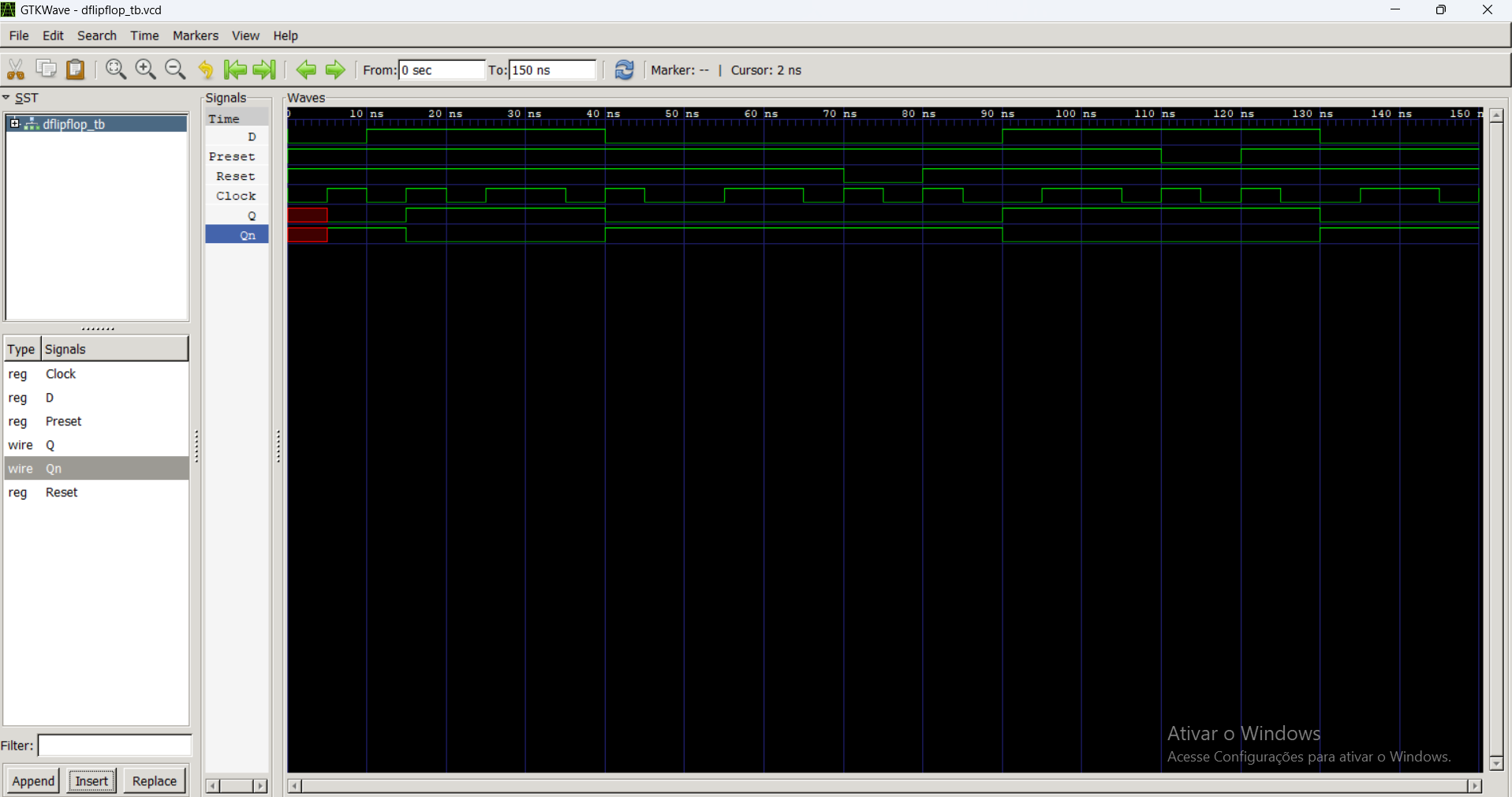
**D LATCH**

****

O D latch apresenta o funcionamento esperado, no início nenhum bit é setado, então aparece sem nenhum estado definido e todo o restante das formas de

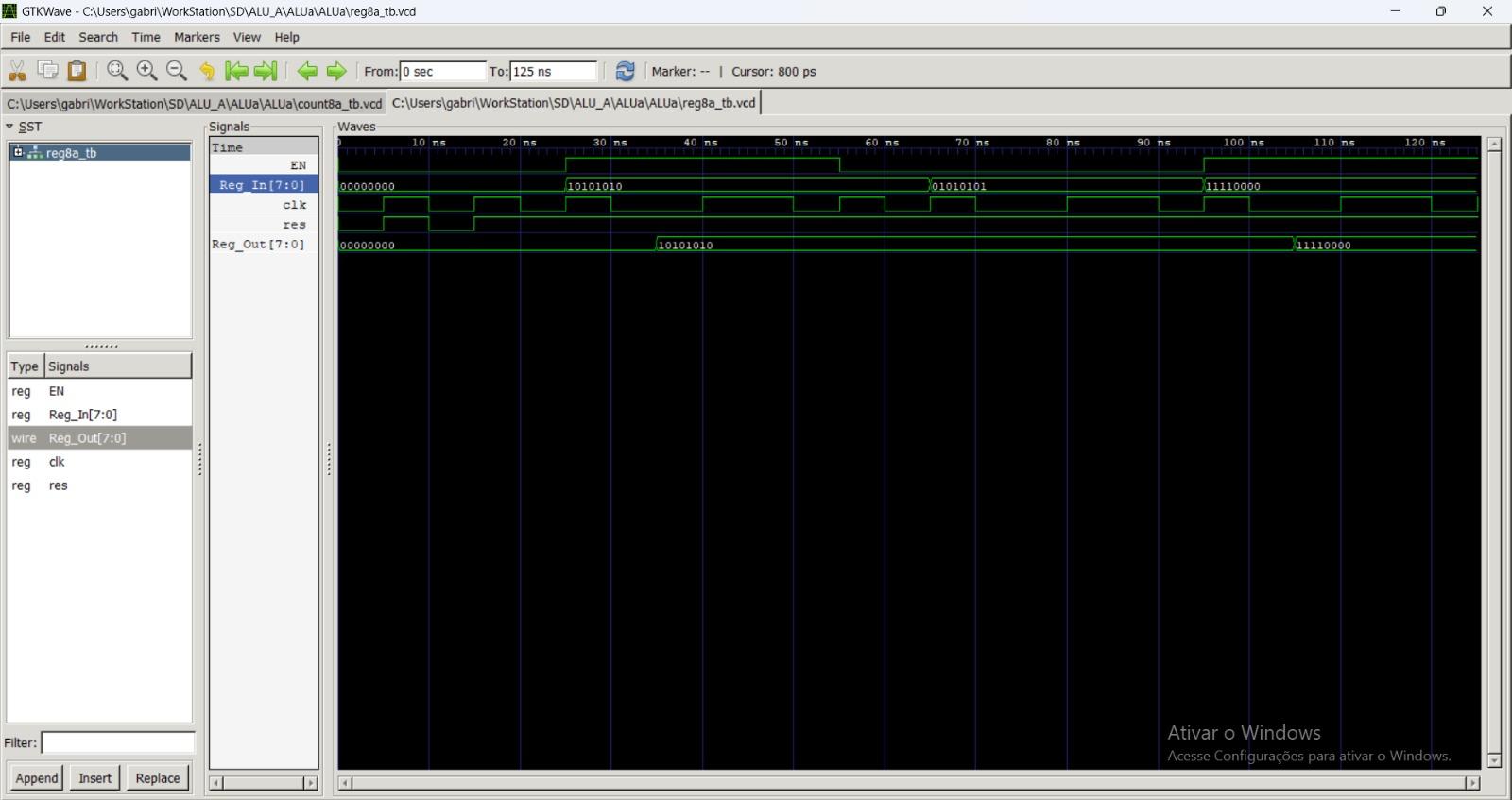
onda apresentam o funcionamento esperado.

**D FLIP FLOP**

****

Similar ao D latch, no início do testbench nenhum bit está armazenado e logo em seguida o seu funcionamento apresenta o resultado esperado.

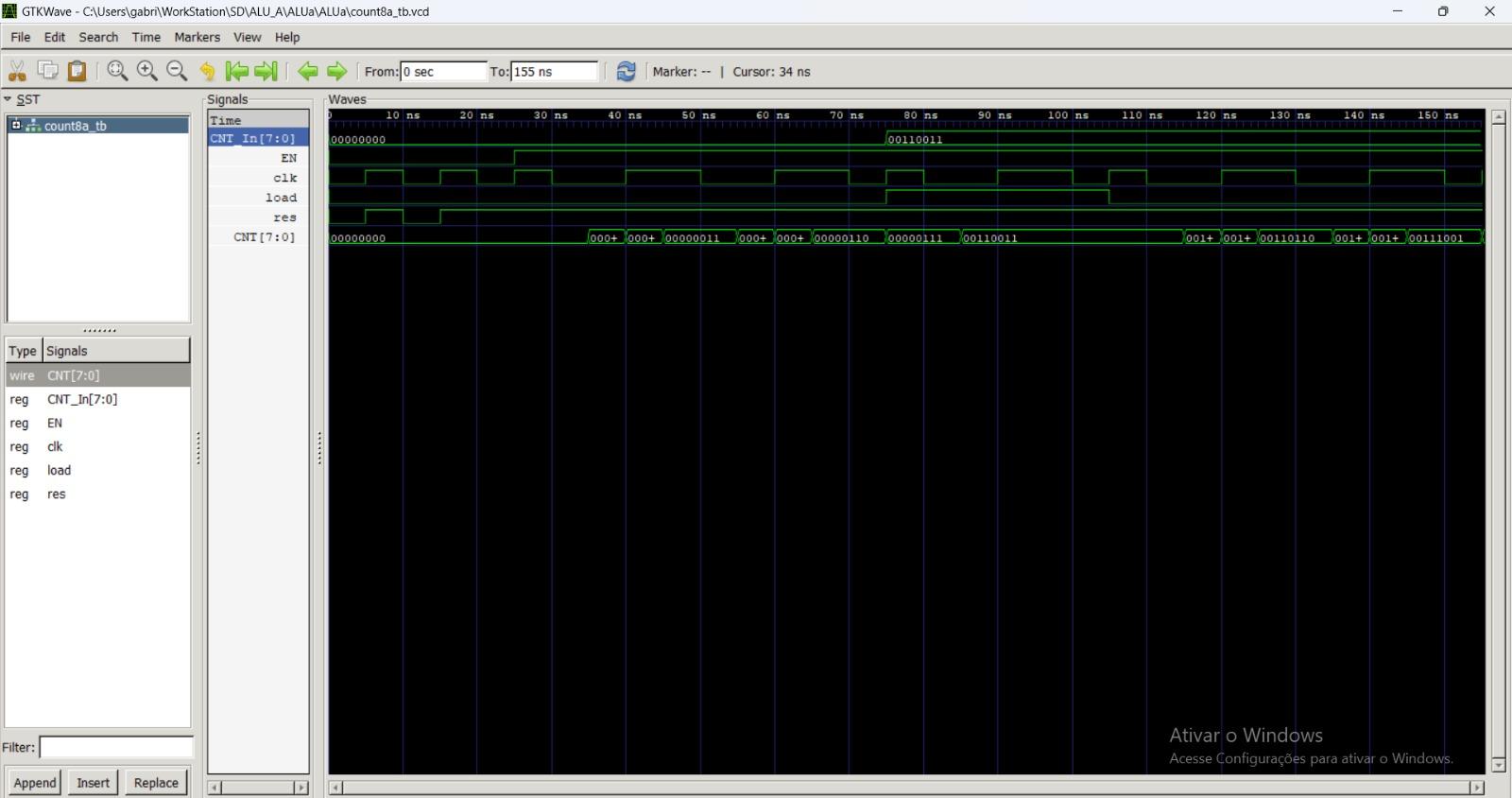
**REGISTER**

****

O registrador está cumprindo o seu papel de registrar o que e/ou incrementar de acordo com a configuração dos bits de entrada, mostrando que passou no

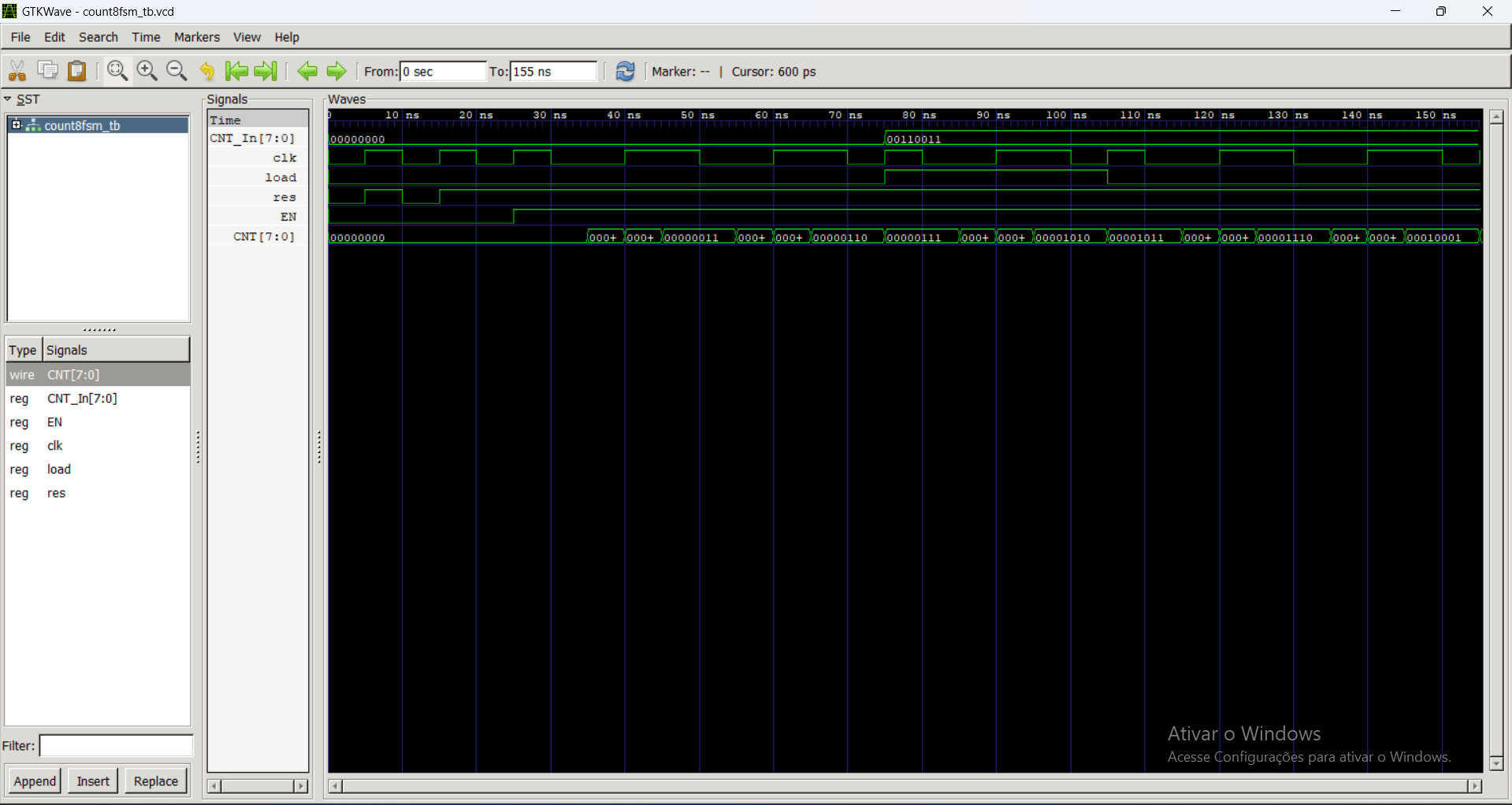
testbench.

**COUNTER**

****

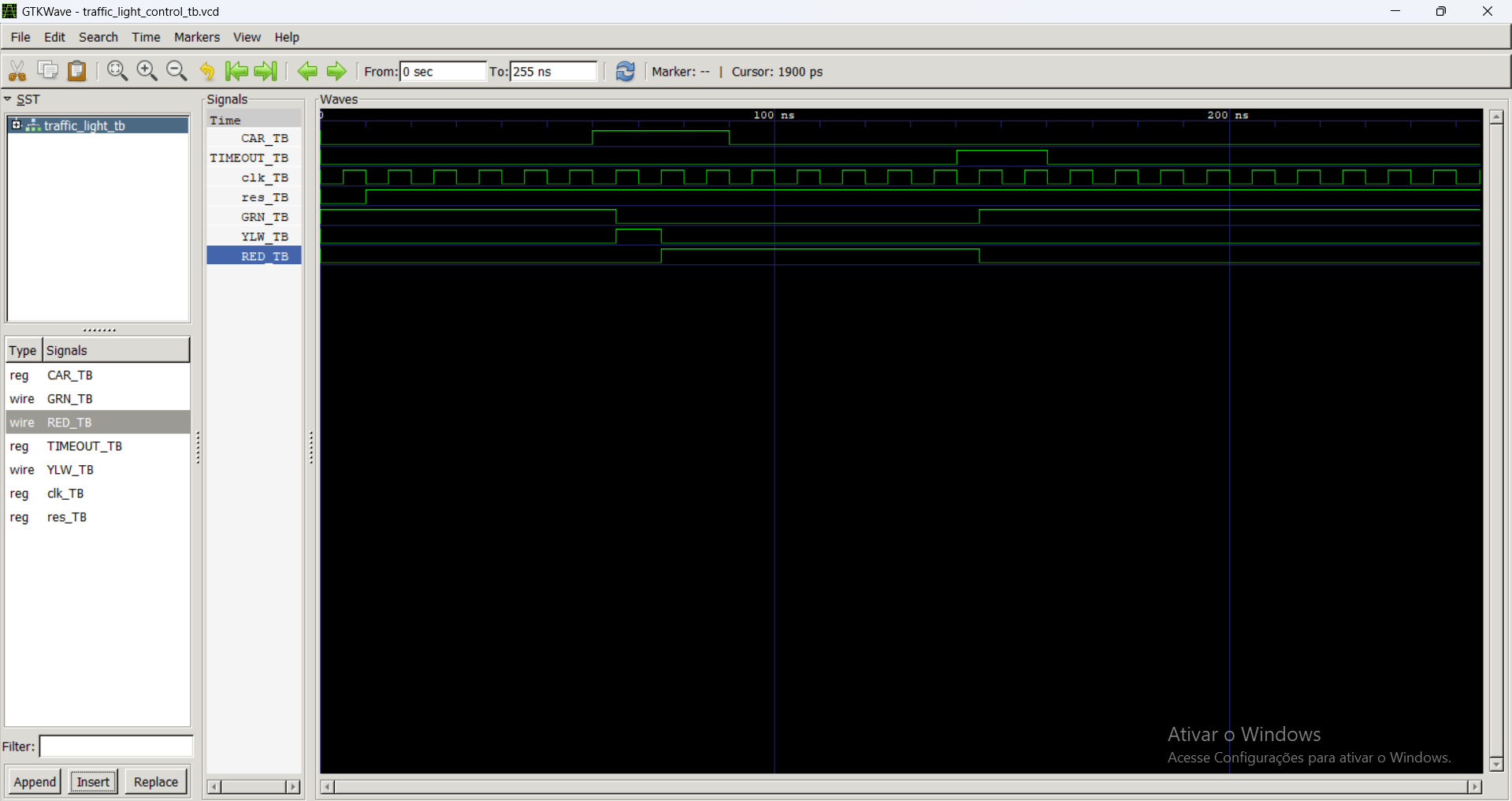
As formas de onda do contador apresentam o funcionamento correto, ele conta caso o LOAD esteja inativo e recebe o CNT in caso esteja ativo.

**COUNTER FSM**



A imagem acima é a forma de onda do testbench do contador implementado usando abordagem de 3 blocos: bloco de memória, bloco de próximo estado e bloco de saída.

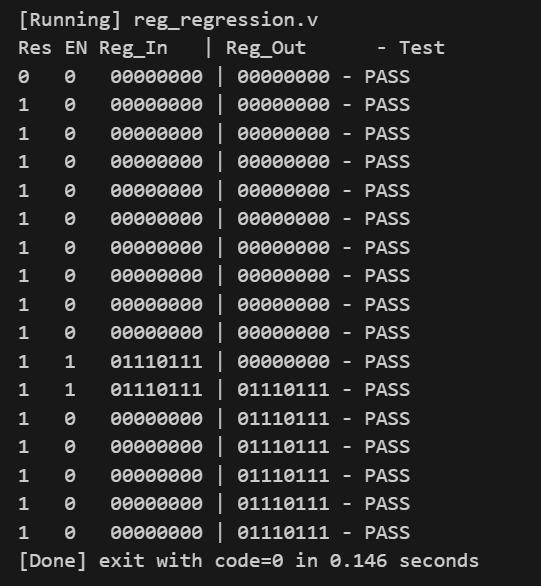
**TRAFFIC LIGHT CONTROL**

****

As formas de onda do do controlador demonstram que ele funciona corretamente. A descrição do controlador pede um controlador que espere durante 15 segundos, no nosso caso, o controlador espera 15 períodos do relógio, imaginando um relógio de frequência de 1hz o controlador espera a quantidade necessária.

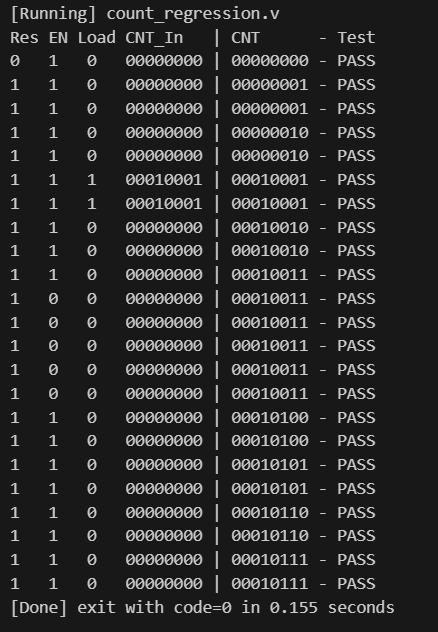
REGRESSION

**REGISTRADOR**

****

A captura de tela acima demonstra que o registrador passou nos testes propostos no teste de regressão.

**CONTADOR**

****

A captura de tela acima demonstra que o contador passou nos testes propostos no teste de regressão.

CÓDIGOS

**REGISTRADOR**

**`timescale 1ns/1ps**

**`include "dflipflop.v"**

**module reg8 (**

**output [7:0] Reg\_Out,**

**input clk, res, EN,**

**input [7:0] Reg\_In**

**);**

**wire[7:0] Q, Qn, saida\_mux;**

**mux2\_1\_8b mux(.a(Q), .b(Reg\_In), .sel(EN), .y(saida\_mux));**

**dflipflop dff0 (.Q(Q[0]), .Qn(Qn[0]), .Clock(clk), .Reset(res), .Preset(1'b1), .D(saida\_mux[0]));**

**dflipflop dff1 (.Q(Q[1]), .Qn(Qn[1]), .Clock(clk), .Reset(res), .Preset(1'b1), .D(saida\_mux[1]));**

**dflipflop dff2 (.Q(Q[2]), .Qn(Qn[2]), .Clock(clk), .Reset(res), .Preset(1'b1), .D(saida\_mux[2]));**

**dflipflop dff3 (.Q(Q[3]), .Qn(Qn[3]), .Clock(clk), .Reset(res), .Preset(1'b1), .D(saida\_mux[3]));**

**dflipflop dff4 (.Q(Q[4]), .Qn(Qn[4]), .Clock(clk), .Reset(res), .Preset(1'b1), .D(saida\_mux[4]));**

**dflipflop dff5 (.Q(Q[5]), .Qn(Qn[5]), .Clock(clk), .Reset(res), .Preset(1'b1), .D(saida\_mux[5]));**

**dflipflop dff6 (.Q(Q[6]), .Qn(Qn[6]), .Clock(clk), .Reset(res), .Preset(1'b1), .D(saida\_mux[6]));**

**dflipflop dff7 (.Q(Q[7]), .Qn(Qn[7]), .Clock(clk), .Reset(res), .Preset(1'b1), .D(saida\_mux[7]));**

**assign Reg\_Out = Q;**

**endmodule**

**CONTADOR**

`timescale 1ns/1ps

`include "reg8a.v"

module count8 (

output [7:0] CNT,

input clk, res, EN, load,

input [7:0] CNT\_In

);

wire [7:0] reg\_out, next\_val\_incremented, saida\_mux;

wire cout;

reg [7:0] next\_val;

reg8 rg8 (.Reg\_Out(reg\_out), .clk(clk), .res(res), .EN(EN), .Reg\_In(next\_val));

full\_adder\_8b fa (.a(reg\_out), .b(8'b0000\_0001), .cin(1'b0), .sum(next\_val\_incremented), .cout(cout));

mux2\_1\_8b mx (.y(saida\_mux), .a(next\_val\_incremented), .b(CNT\_In), .sel(load));

always @(\*) begin

next\_val = saida\_mux;

end

assign CNT = reg\_out;

endmodule

**CONTADOR FSM**

`timescale 1ns/1ps

module count8fsm (

output reg [7:0] CNT,

input wire clk, res, EN, load,

input wire [7:0] CNT\_In

);

*// Estados*

parameter RESET = 2'b00, IDLE = 2'b01, LOAD = 2'b10, COUNT = 2'b11;

reg [1:0] current\_state, next\_state;

initial begin

current\_state = RESET;

next\_state = RESET;

CNT = 8'b0000\_0000;

end

*// Bloco de memória de estado*

always @(posedge clk or negedge res) begin

if (!res)

current\_state <= RESET;

else

current\_state <= next\_state;

end

*// Bloco de lógica de próximo estado*

always @(current\_state or EN or load) begin

if (current\_state == RESET)

next\_state = IDLE;

else if (current\_state == IDLE)

next\_state = EN ? (load ? LOAD : COUNT) : IDLE;

else if (current\_state == LOAD)

next\_state = COUNT;

else if (current\_state == COUNT)

next\_state = EN ? COUNT : IDLE;

else

next\_state = RESET;

end

*// Bloco de lógica de saída*

always @(posedge clk or negedge res) begin

if (!res)

CNT <= 8'b0000\_0000;

else if (current\_state == RESET)

CNT <= 8'b0000\_0000;

else if (current\_state == IDLE)

CNT <= CNT;

else if (current\_state == LOAD)

CNT <= CNT\_In;

else if (current\_state == COUNT)

CNT <= CNT + 1;

else

CNT <= 8'b0000\_0000;

end

endmodule

**CONTROLADOR DE SEMÁFORO**

`timescale 1ns/1ps

module traffic\_light\_control (

output reg GRN, YLW, RED,

input wire CAR, TIMEOUT, clk, res

);

*// Estados*

parameter VERDE = 2'b00, AMARELO = 2'b01, VERMELHO = 2'b10, WAIT = 2'b11;

reg [1:0] current\_state, next\_state;

initial begin

current\_state = VERDE;

next\_state = VERDE;

GRN = 1'b1;

YLW = 1'b0;

RED = 1'b0;

end

*// Bloco de memória de estado*

always @(posedge clk or negedge res) begin

if (!res)

current\_state <= VERDE;

else

current\_state <= next\_state;

end

*// Bloco de lógica de próximo estado*

always @(current\_state or CAR or TIMEOUT) begin

if (current\_state == VERDE && CAR)

next\_state = AMARELO;

else if (current\_state == AMARELO)

next\_state = VERMELHO;

else if (current\_state == VERMELHO)

next\_state = WAIT;

else if (current\_state == WAIT && TIMEOUT)

next\_state = VERDE;

else

next\_state = current\_state;

end

*// Bloco de lógica de saída*

always @(current\_state) begin

case (current\_state)

VERDE: begin

GRN = 1'b1;

YLW = 1'b0;

RED = 1'b0;

end

AMARELO: begin

GRN = 1'b0;

YLW = 1'b1;

RED = 1'b0;

end

VERMELHO: begin

GRN = 1'b0;

YLW = 1'b0;

RED = 1'b1;

end

WAIT: begin

GRN = 1'b0;

YLW = 1'b0;

RED = 1'b1;

end

default: begin

GRN = 1'b1;

YLW = 1'b0;

RED = 1'b0;

end

endcase

end

endmodule

\*Todos os arquivos podem ser encontrados em APS3.zip