**MINI PROJECT**

**Controller Area Network Transmission Module Using Verilog HDL**

**Abstract:**

Controller Area Network or CAN protocol is a method of communication between various electronic devices. It defines a standard for efficient and reliable communication between sensor, actuator, controller, and other nodes in real-time applications. This provides a mechanism which is incorporated in the hardware and the software by which different electronic modules can communicate with each other using a common cable. CAN protocol is a message-based protocol, not an address-based protocol. All nodes in the system receive every message transmitted on the bus (and will acknowledge if the message was properly received). It is up to each node in the system to decide whether the message received should be immediately discarded or kept to be processed.

**Outcomes:**

The main aim is to design, development, and implementation of a CAN module using Verilog HDL is that to show that the designed CAN module can replace a standalone CAN controller which occupies extra space in the system. It has significant use in the industry of automobiles. Also implementing the CAN with the help of custom RAM is aimed at removing the need for FPGA specific RAM. The proposed design of CAN module is an integration of lower-level modules. The desired Verilog implementation of CAN module can help in simulating different systems at the design level instead of using physical ICs. The design includes a simple scheme that aims in reduction of circuit complexity and chances of hardware failure without requiring any extra logic circuitry.

// Code your design here

module can\_tx(

output reg tx,

output reg can\_bitstuff,

output reg txing,

input rx,

input[10:0] address,

input clk,

input baud\_clk,

input rst,

input [7:0] data,

input send\_data,

input bitstuffed\_output,

input clear\_to\_tx

);

assign rx\_buf = rx;

parameter all\_ones = 15'b111111111111111;

parameter idle = 8'h0, start\_of\_frame = 8'h1, addressing =8'h2 ,rtr = 8'h3 ,ide = 8'h4, reserve\_bit = 8'h5, num\_of\_bytes = 8'h6,

data\_out = 8'h7, crc\_out = 8'h8, crc\_delimiter = 8'h9 , ack = 8'hA, ack\_delimiter = 8'hB, end\_of\_frame = 8'hC, waiting = 8'hD;

parameter bytes = 5'd8;

reg[10:0] address\_count = 0, crc\_count = 0, eof\_count = 0 , data\_bit\_count = 0, data\_byte\_count = 0;

reg[7:0] c\_state=0, n\_state=0;

initial txing = 0;

reg[14:0] crc\_output, crc\_holder;

wire one\_shotted\_send;

wire[14:0] crc\_buff;

CRC cyclic\_red\_check(data, one\_shotted\_send, crc\_buff,rst,clk);//caliing crc module

OneShot os(send\_data, clk, rst, one\_shotted\_send);//calling one shot pulse module

always @(crc\_buff or crc\_holder) begin

if(crc\_buff != all\_ones)

crc\_output <= crc\_buff;

else

crc\_output <= crc\_holder;

end

always @ (posedge clk or posedge rst) begin

if(rst == 1) begin

crc\_holder <= 15'd0;

end

else begin

crc\_holder <= crc\_output;

end

end

//Update Logic

always @ (posedge baud\_clk or posedge rst) begin

if(rst == 1) begin

c\_state <= 32'd0;

end

else begin

c\_state <= n\_state;

end

end

//Counting Logic

always @ (posedge baud\_clk) begin

case(c\_state)

idle: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

waiting: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

start\_of\_frame:begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

addressing: begin

address\_count <= address\_count + 1'b1;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

rtr: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

ide: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

reserve\_bit: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

num\_of\_bytes: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= data\_byte\_count +1'b1;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

data\_out: begin

address\_count <= 11'd0;

data\_bit\_count<= data\_bit\_count +1'b1;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

crc\_out: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= crc\_count + 1'b1;

eof\_count <= 11'd0;

end

crc\_delimiter: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

ack: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

ack\_delimiter:begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

end\_of\_frame: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= eof\_count +1'b1;

end

default: begin

address\_count <= 11'd0;

data\_bit\_count<= 11'd0;

data\_byte\_count<= 11'd0;

crc\_count <= 11'd0;

eof\_count <= 11'd0;

end

endcase

end

//Next State Logic

always @ (c\_state or rx\_buf or data or send\_data or address\_count or bitstuffed\_output or data\_byte\_count

or data\_bit\_count or crc\_count or eof\_count or clear\_to\_tx or crc\_output) begin

case(c\_state)

idle: begin

if(send\_data && clear\_to\_tx) begin

n\_state <= start\_of\_frame;

end

else begin

n\_state <= idle;

end

end

start\_of\_frame: begin

if(!rx\_buf) begin

n\_state <= addressing;

end

else begin

n\_state <= waiting;

end

end

waiting: begin

if(send\_data && clear\_to\_tx) begin

n\_state <= start\_of\_frame;

end

else begin

n\_state <= waiting;

end

end

addressing: begin

if(rx\_buf != bitstuffed\_output) begin

n\_state <= waiting; //Lost Arbitration

end

else if(address\_count == 11'd10) begin

n\_state <= rtr;

end

else begin

n\_state <= addressing;

end

end

rtr: begin

n\_state <= ide;

end

ide: begin

n\_state <= reserve\_bit;

end

reserve\_bit: begin

n\_state <= num\_of\_bytes;

end

num\_of\_bytes: begin

if(data\_byte\_count == 11'd3) begin

n\_state <= data\_out;

end

else begin

n\_state <= num\_of\_bytes;

end

end

data\_out: begin

if(data\_bit\_count == 11'd63) begin

n\_state <= crc\_out;

end

else begin

n\_state <= data\_out;

end

end

crc\_out: begin

if(crc\_count == 11'd14) begin

n\_state <= crc\_delimiter;

end

else begin

n\_state <= crc\_out;

end

end

crc\_delimiter: begin

n\_state <= ack;

end

ack: begin

n\_state <= ack\_delimiter;

end

ack\_delimiter: begin

n\_state <= end\_of\_frame;

end

end\_of\_frame: begin

if(eof\_count == 11'd6) begin

n\_state <= idle;

end

else begin

n\_state <= end\_of\_frame;

end

end

default:

begin

n\_state <= idle;

end

endcase

end

//Output Logic

always @(c\_state or address or data or crc\_output or crc\_count or data\_byte\_count or data\_bit\_count or address\_count) begin

case(c\_state)

idle: begin

tx <= 1;

can\_bitstuff <= 0;

txing <= 1'b0;

end

addressing: begin

tx <= address[11'd10-address\_count];

can\_bitstuff <= 1;

txing <= 1'b1;

end

start\_of\_frame: begin

tx<= 0;

can\_bitstuff <= 1'b0;

txing <= 1'b1;

end

rtr: begin

tx <= 0;

can\_bitstuff <= 1;

txing <= 1'b1;

end

ide: begin

tx <= 0;

can\_bitstuff <= 1;

txing <= 1'b1;

end

reserve\_bit: begin

tx <= 0;

can\_bitstuff <= 1;

txing <= 1'b1;

end

num\_of\_bytes: begin

tx <= bytes[11'd3-data\_byte\_count];

can\_bitstuff <= 1;

txing <= 1'b1;

end

data\_out: begin

tx <= data[11'd63-data\_bit\_count];

can\_bitstuff <= 1;

txing <= 1'b1;

end

crc\_out: begin

tx <= crc\_output[11'd14-crc\_count];

can\_bitstuff <= 1;

txing <= 1'b1;

end

crc\_delimiter: begin

tx <= 1;

can\_bitstuff <= 0;

txing <= 1'b1;

end

ack: begin

tx <= 1;

can\_bitstuff <= 0;

txing <= 1'b1;

end

ack\_delimiter:begin

tx <= 1;

can\_bitstuff <= 0;

txing <= 1'b1;

end

end\_of\_frame: begin

tx <= 1;

can\_bitstuff <= 0;

txing <= 1'b1;

end

waiting: begin

tx <= 1;

can\_bitstuff <= 0;

txing <= 1'b0;

end

default: begin

tx <= 1;

can\_bitstuff <= 0;

txing <= 1'b1;

end

endcase

end

endmodule

module CRC (

input [7:0] data\_in,

input crc\_en,

output reg [4:0] crc\_out,

input clk,

input reset

);

reg [4:0] lfsr\_q, lfsr\_c;

always @\* begin

lfsr\_c[0] = data\_in[7] ^ data\_in[5] ^ data\_in[4] ^ data\_in[3] ^ data\_in[0] ^ lfsr\_q[4] ^ lfsr\_q[3] ^ lfsr\_q[2] ^ lfsr\_q[0];

lfsr\_c[1] = data\_in[7] ^ data\_in[6] ^ data\_in[5] ^ data\_in[2] ^ data\_in[0] ^ lfsr\_q[4] ^ lfsr\_q[3] ^ lfsr\_q[1];

lfsr\_c[2] = data\_in[6] ^ data\_in[5] ^ data\_in[4] ^ data\_in[1] ^ data\_in[0] ^ lfsr\_q[4] ^ lfsr\_q[2];

lfsr\_c[3] = data\_in[7] ^ data\_in[5] ^ data\_in[4] ^ data\_in[3] ^ data\_in[1] ^ lfsr\_q[3] ^ lfsr\_q[1] ^ lfsr\_q[0];

lfsr\_c[4] = data\_in[6] ^ data\_in[5] ^ data\_in[2] ^ data\_in[1] ^ lfsr\_q[4] ^ lfsr\_q[2] ^ lfsr\_q[1] ^ lfsr\_q[0];

end

always @(posedge clk, posedge reset) begin

if (reset)

lfsr\_q <= 0;

else

lfsr\_q <= lfsr\_c;

end

always @(posedge clk) begin

crc\_out <= lfsr\_q;

end

endmodule

module OneShot(

input pulse,

input clk,

input rst,

output reg out

);

initial out = 0;

parameter waiting\_l = 2'b00, on = 2'b01, waiting\_h = 2'b10;

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

always @ (posedge clk or posedge rst) begin

if(rst) begin

current\_state <= waiting\_l;

end

else begin

current\_state <= next\_state;

end

end

always @ (current\_state or pulse) begin

if(current\_state == on) begin

next\_state <= waiting\_h;

end

else if(current\_state == waiting\_h) begin

if(pulse) begin

next\_state <= waiting\_h;

end

else begin

next\_state <= waiting\_l;

end

end

else if(pulse) begin

next\_state<= on;

end

else begin

next\_state<= waiting\_l;

end

end

always @(current\_state or rst) begin

if(rst)

out <= 1'b0;

else if(current\_state == on)

out <= 1'b1;

else

out <= 1'b0;

end

endmodule

**TESTBENCH CODE:**

module can\_tx\_tb;

parameter CLK\_PERIOD = 10;

parameter SIM\_TIME = 500;

// Signals

reg clk = 0;

reg baud\_clk = 0;

reg rst = 1;

reg send\_data = 0;

reg bitstuffed\_output = 0;

reg clear\_to\_tx = 1;

reg rx = 0;

reg [10:0] address = 11'd0;

reg [7:0] data = 8'd0;

reg pulse;

wire tx;

wire can\_bitstuff;

wire txing;

wire out;

// Instantiate the modules

can\_tx dut (

.tx(tx),

.can\_bitstuff(can\_bitstuff),

.txing(txing),

.rx(rx),

.address(address),

.clk(clk),

.baud\_clk(baud\_clk),

.rst(rst),

.data(data),

.send\_data(send\_data),

.bitstuffed\_output(bitstuffed\_output),

.clear\_to\_tx(clear\_to\_tx)

);

OneShot os (

.pulse(pulse),

.clk(clk),

.rst(rst),

.out(out)

);

// Clock generation

always #((CLK\_PERIOD/2)) clk = ~clk;

always #5 baud\_clk = ~baud\_clk;

initial begin

rst = 1;

#20 rst = 0;

#50 send\_data = 1;

#10 send\_data = 0;

#100 rx = 1;

#10 rx = 0;

#50 address = 11'd123;

#70 data = 8'd255;

#150 pulse = 1;

#10 pulse = 0;

#200 $finish;

end

// Display output

always @(posedge clk) begin

$display("Time = %0t: tx = %b, can\_bitstuff = %b, txing = %b, out = %b", $time, tx, can\_bitstuff, txing, out);

end

initial begin

$dumpfile("dump.vcd");

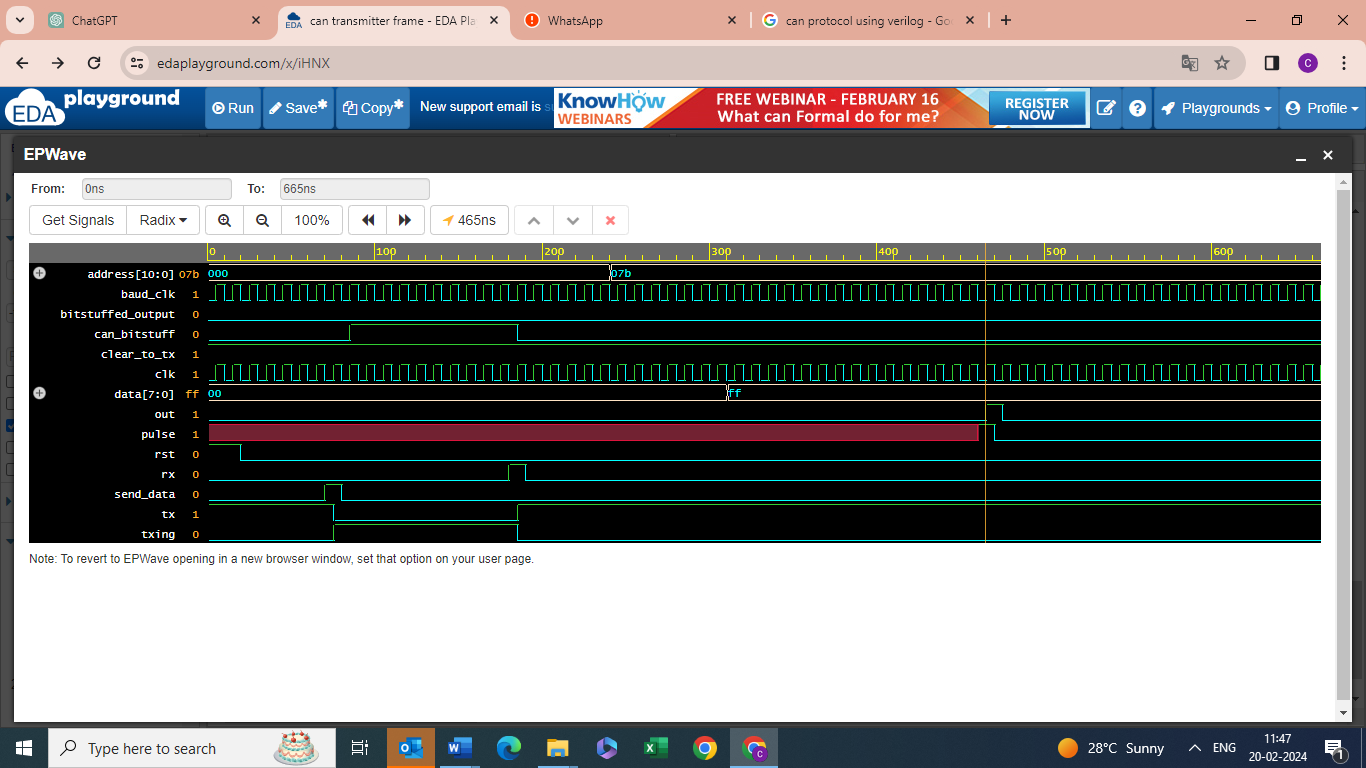
$dumpvars(0);

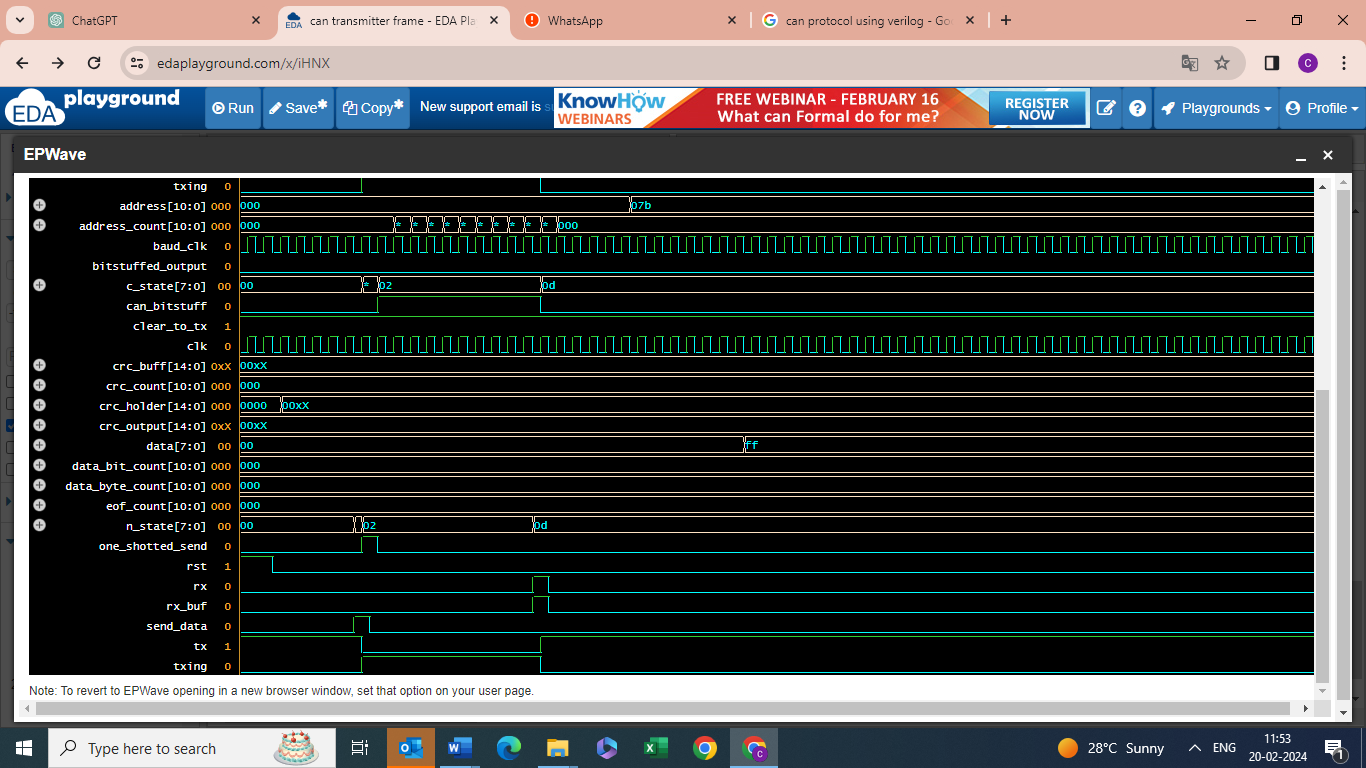
end

endmodule

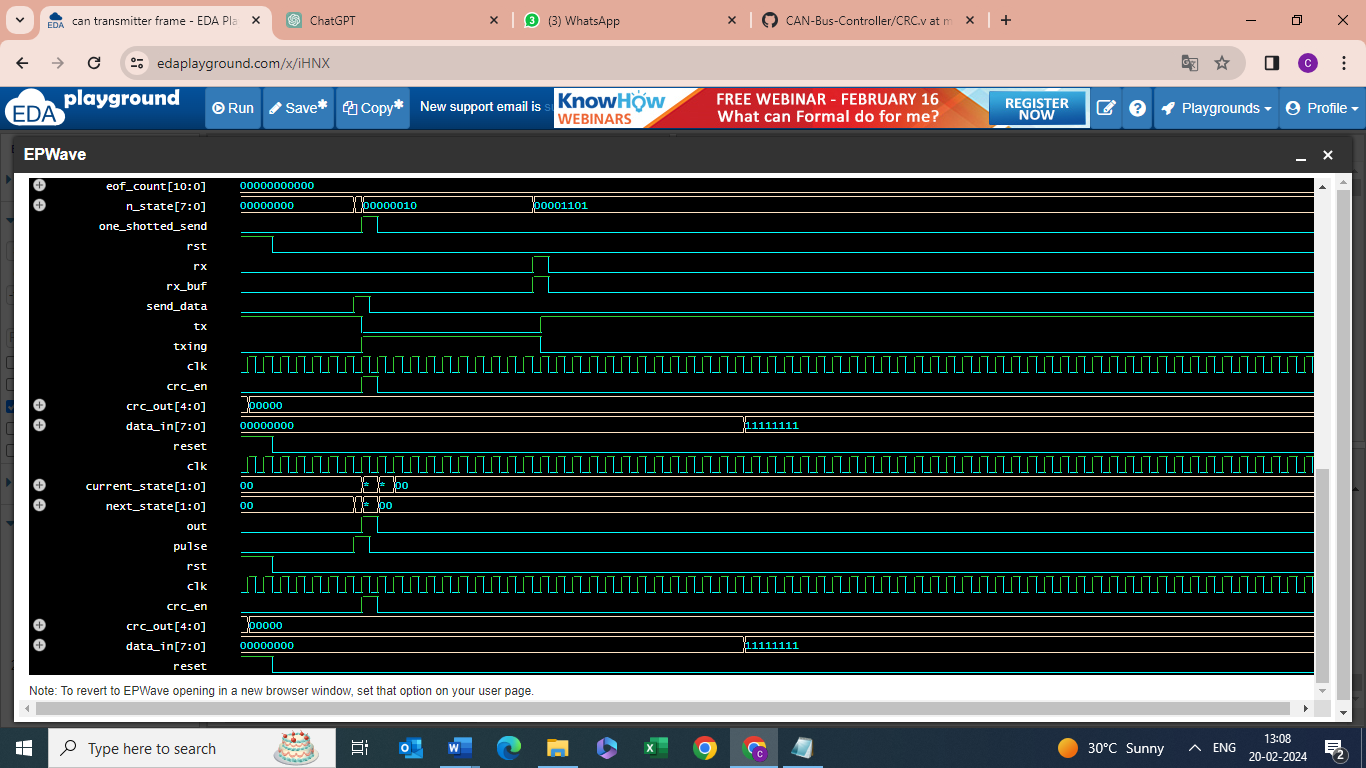
**OUTPUT:**

**Can\_tx\_tb**

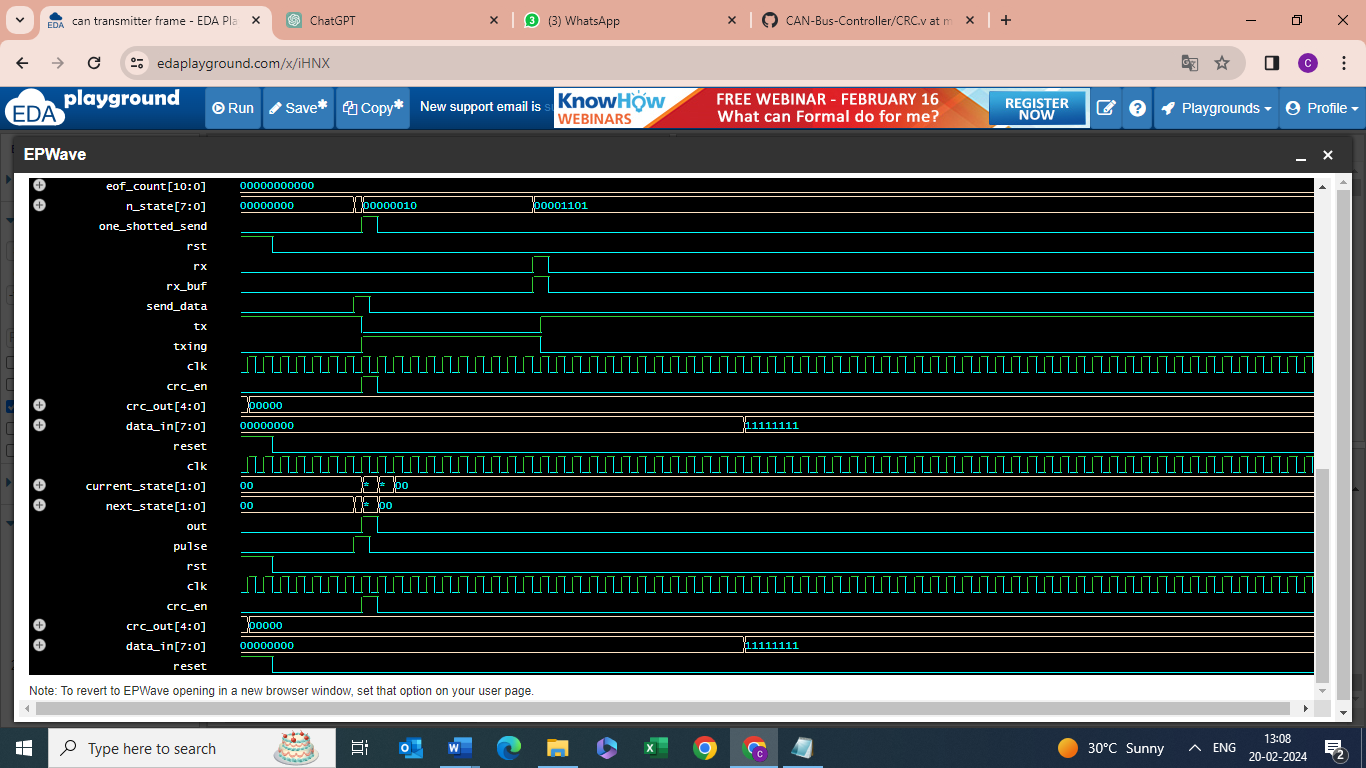


**Dut\_tb:**

**os\_tb:**



**crc\_tb:**



Time = 5: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 15: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 25: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 35: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 45: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 55: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 65: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 75: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 85: tx = 0, can\_bitstuff = 0, txing = 1, out = 0

Time = 95: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 105: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 115: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 125: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 135: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 145: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 155: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 165: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 175: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 185: tx = 0, can\_bitstuff = 1, txing = 1, out = 0

Time = 195: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 205: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 215: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 225: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 235: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 245: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 255: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 265: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 275: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 285: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 295: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 305: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 315: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 325: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 335: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 345: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 355: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 365: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 375: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 385: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 395: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 405: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 415: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 425: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 435: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 445: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 455: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 465: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 475: tx = 1, can\_bitstuff = 0, txing = 0, out = 1

Time = 485: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 495: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 505: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 515: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 525: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 535: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 545: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 555: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 565: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 575: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 585: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 595: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 605: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 615: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 625: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 635: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 645: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 655: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

Time = 665: tx = 1, can\_bitstuff = 0, txing = 0, out = 0

$finish called from file "testbench.sv", line 68.