1. feladat: a modul létrehozása

New project: HDL, next, next, finish.

New source -> verilog module next, next, finish.

```
`timescale 1ns / 1ps
module uart_module(
    input clk,
    input rst,
    input [3:0] bcd0,
    input [3:0] bcd1,
    output tx_out
 );

reg [4:0] tx_cntr; // counter: which bit we put on the output
reg [29:0] tx_shr; // shiftregister: we put in this all three chars at the same time
reg tx_en; //enabling flag
reg tx_reg; // register for output

reg [8:0] bd; // for 576000 bit/sec with 16MHz we need to count for 16M/576k=277 what we can represent with 9 bits

// assig the output reg to the actual output
assign tx_out = tx_reg;

always@(posedge clk)
begin
    if(rst)
    begin
        // reset set all signals to default values
        tx_en <= 0;
        tx_shr <= 0;
        bd <= 0;
        tx_cntr <= 0;
        tx_reg <= 1;
    end

    else
    begin
    // instead of 'rategen' module
        if(bd==277) // 16000000/57600 = 277
        begin
            bd <= 0;
            tx_en <= 1;
        end
        else
        begin
            bd <= bd + 1;
            tx_en <= 0;
        end
    end

    if(tx_en)
    begin
        if(tx_cntr==0)
```

```
        begin
            //bcd1
            tx_shr[0] <= 1'b0; //start
            tx_shr[7:1] <= {3'b011,bcd1[3:0]}; //hexa 30 (ASCII '0') + bcd
            tx_shr[8] <= bcd1[0]+bcd1[1]+bcd1[2]+bcd1[3]+1; //parity
            tx_shr[9] <= 1; //stop

            //bcd0
            tx_shr[10] <= 0; //start
            tx_shr[17:11] <= {3'b011,bcd0[3:0]}; //hexa 30 (ASCII '0') + bcd
            tx_shr[18] <= bcd0[0]+bcd0[1]+bcd0[2]+bcd0[3]+1; //parity
            tx_shr[19] <= 1; //stop

            // add carriage return char
            tx_shr[20] <= 0; //start
            tx_shr[27:21] <= 7'b0001101; // CR in ascii is 0x0d
            tx_shr[28] <= 0; //parity
            tx_shr[29] <= 1; //stop

            tx_cntr <= 29; // set counter for counting down
            tx_reg <= 1; //set output 1
        end
        else
        begin
            // otherwise put out the LSB to the output then shift right and count down
            tx_reg <= tx_shr[0];
            tx_shr <= tx_shr >> 1;
            tx_cntr <= tx_cntr - 1;
        end
    end
end
endmodule
```

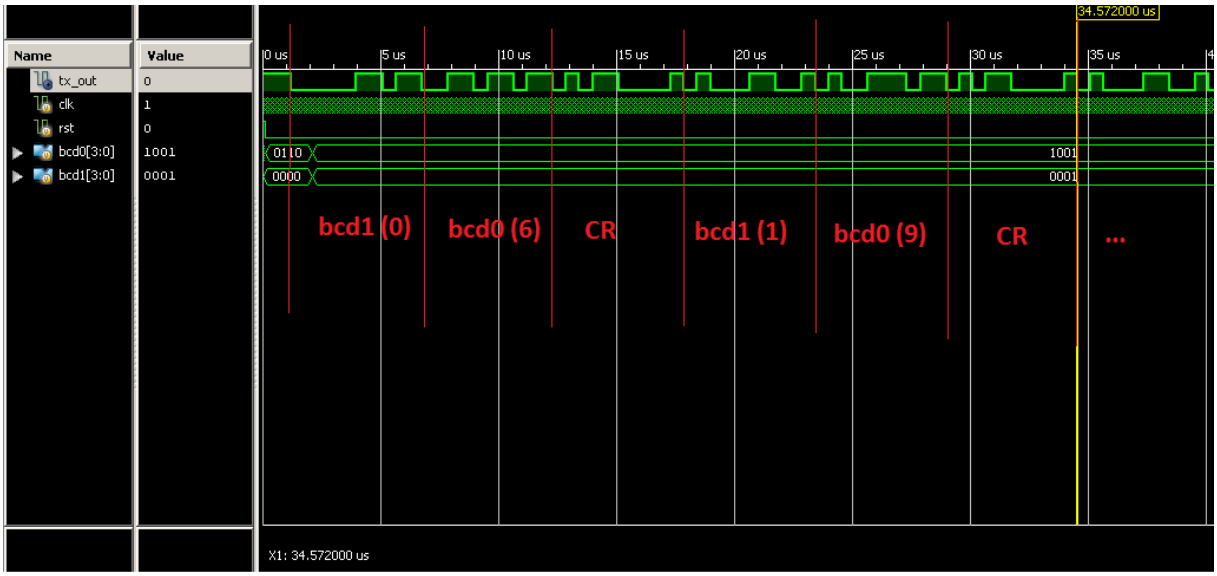
2. feladat: tesztmodul létrehozása

Silulation view, Add new source -> new verilog test fixture, next, next, finish.

```
`timescale 1ns / 1ps

module uart_test;

    // Inputs
    reg clk;
    reg rst;
    reg [3:0] bcd0;
    reg [3:0] bcd1;

    // Outputs
    wire tx_out;

    // Instantiate the Unit Under Test (UUT)
    uart_module uut (
        .clk(clk),
        .rst(rst),
        .bcd0(bcd0),
        .bcd1(bcd1),
        .tx_out(tx_out)
```

```
    );

    initial begin
        clk = 1;
        rst = 1;
        bcd0 = 0;
        bcd1 = 0;

        #100;
        rst=0;
        // Add stimulus here

        #1
        // junius 19. = 0619
        bcd1 = 0;
        bcd0 = 6;

        #2000  // long enough
        bcd1 = 1;
        bcd0 = 9;
    end
    always #1 clk=~clk;

endmodule
```

3. feladat: szimuláció (ISim Simulation)



A következő jelsorozatot látjuk:

0 0000 110 1 1	bcd1 (0) + ascii ’30’
0 0110 110 1 1	bcd0 (6) + ascii ’30’
0 1011000 0 1	ascii ’CR’
0 1000 110 0 1	bcd1 (1) + ascii ’30’
0 1001 110 1 1	bcd0 (9) + ascii ’30’
0 1011000 0 1	ascii ’CR’

Zöld: start bit, kék: paritás bit, piros: stop bit.

Az adó három karaktert küld folyamatosan: bcd1, bcd0 és egy kocsi-visszát, én június 19-én láttam meg a napvilágot, így a 06 19 jelsorozatot állítottam elő.

Nézzük meg a bcd karakterek hogyan állnak elő, például az második bcd1 (1) a következőképpen áll elő (ez az egyetlen „izgalmasabb” eset):

- 0: start bit
- 1000: 1 = 4’b0001 -> ezeket LSB-MSB sorrendben kell kiküldeni.
- 110: Mivel a ’0’ karakter ascii kódja 30, ezért hozzá kell még adni 30-at (a 0 és 9 közötti számok 4 biten elférnek, így ez a fenti kódban egy konkatenálás volt) 3 = 3’b011
- 0: Mivel a paritásunk páratlan, azaz páratlanra kell kiegészíteni.
- 1: stop bit

4. feladat: ellenőrző kérdések

1. ellenőrző kérdés

Egyetlen ASCII karakter UART átvitele hány bit kiküldését jelenti a házi feladatban adott paraméterek esetén? Pontosan hogy épül fel ez a keret, milyen sorrendben kell kiküldeni a biteket?

1 start bit + 7 adat bit + 1 paritás bit + 1 stop bit = 10 bit

start -> LSB…MSB -> paritás -> stop

2. ellenőrző kérdés

Hogy kapjuk meg egy 4 bites BCD számjegy 8 bites ASCII kódját? Mit mutat a paritásbit, milyen hardver elemmel számítható a legegyszerűbben, ha az összes adatbit rendelkezésre áll? Páros paritás esetén milyen paritás érték tartozik az ABh adathoz?

A ’0’ karakter ASCII kódja 30h, ehhez kell hozzáadni a számjegy értékét, így kapjuk meg a küldendő számot, pl. 5: 30h+ 5 = 0011 0101

A paritás bit ellenőrzésre szolgál, és legegyszerűbben XOR kapuval lehet megvílósítani.

ABh = 1010 1011, mivel ez páratlan, és páros paritásnál párosra kell kiegészíteni, ezért a paritás bit értéke ez esetben 1.

3. ellenőrző kérdés

Ha a rendszer órajele 16 MHz, mekkora a házi feladatként létre hozott UART modul sebességének hibája százalékosan kifejezve, a specifikált értékhez képest? Okozhat e ez problémát? (Válaszát indokolja!)

Egy számlálóval állítjuk elő az engedélyező jelet, aminek feltételét egy osztással kapjuk, nálam ez

16M/57.6k = 277.77777 => azaz 277-re állítottam a feltételt, viszont 16M/277 = 57761.73, azaz keletkezik egy 1/57.6k – 1/57761.73 = 48.61 ns-os bitidő különbség.

Ahhoz, hogy ténylegesen hibátlan legyen a mintavételezés, a stop bit 16 mintavételéből a 8-es és 9-es értéknek ténylegesen a stop bitből kell származniuk.

Maximális hiba/keret: 1/57761.73 * 9/16 = 9.738 us

Maximálisan megengedett: 9.738 us / 10 bit = 0.9738 us

A maximális hiba / bitidő = a számláló kerekítéséből származó hiba / maximálisan megengedett:

48.61 ns / 0.9738 us = 4.99 %.