

**DIGITAL SYSTEM DESIGN APPLICATION**

**EHB436E CRN: 11280**

**PROJECT2**

**Salih Ömer Ongün**

**040220780**

**UART PROTOCOL**

Communication protocols are very important in the use of electronic integrated circuits and devices. Certain rules must be followed for two or more electronic components to be understood and communicated correctly. For this reason, there are many protocols in communication systems. The UART protocol is one of them.

UART uses two wires to transmit and receive data. UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end.[1]

diyagram, plan, çizgi, piksel içeren bir resim

Açıklama otomatik olarak oluşturuldu[1]

The clk signal is not used for synchronization of data in UART. Data is sent asynchronously. There is no data transmission in UART based on the clk signal, there is asynchronous transmission. Tx and rx process the data they obtain according to the specified baud rate. There are different baud rates. It is necessary to decide in advance which rate to use.

 [1]

Normally, the logic 1 bit is always on. When the start bit becomes logic zero, the system starts working. It usually has 8 bits of data. In addition, the stop bit is sent at the end to stop the system. The stop bit is logic 1.

metin, diyagram, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

[2]

The transmitter receives data in parallel. It samples within itself with the baud rate frequency and gives parallel data to the output serially. The receiver receives the data serially. It samples the data with the baud rate frequency determined within itself and gives the serial data to the output in parallel.

metin, diyagram, plan, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

diyagram, metin, çizgi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldudiyagram, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

[1]

**ALGORITHM**

I create four modules. These are uart\_rx, uart\_tx, baud\_gen, uart\_top. The uart\_rx module acts as the receiver of the uart. The uart\_tx module acts as the transmitter of the uart. It has a baud gen frequency divider structure. Therefore, it produces the baud rate we need and 8x equivalent of the baud rate. Uart\_top is the top module where I combine all the modules.

**Baud Rate Generator Module**

This module produces the desired baud rate and 8x the equivalent of the baud rate, thanks to its parameterized structure from the internal 100 MHz clk signal. For this project, the baud rate should be 9600 and 115200.

**Design Source**

**module** baud\_gen **#(parameter** **[**16**:**0**]** freq**=** 115200**)**

**(**

**input** clk**,**

**input** rst**,**

**input** baud\_en**,**

**output** count\_8x\_ready**,**

**output** count\_baud\_ready

**);**

**localparam** **integer** clk\_freq **=** 100000000**;** // Internal clock frequency (100 MHz)

**reg** count\_8x\_ready\_reg**,**count\_baud\_ready\_reg**;**

**reg** **[**13**:**0**]** counter\_8x**;**

**reg** **[**2**:**0**]** counter\_baud**;**

**always** **@(posedge** clk**)** **begin**

**if(**rst**==**1'b1**)** **begin**

counter\_baud **<=** 3'b0**;** // baud counter

count\_baud\_ready\_reg **<=** 0**;** // baud counter result

counter\_8x **<=** 13'b0**;** // 8x counter

count\_8x\_ready\_reg **<=** 0**;** // 8x counter result

**end**

**else** **if** **(**baud\_en **==** 1**)** **begin**

**if(**counter\_8x **==((**clk\_freq**/(**freq**\***8**))-**1**))** **begin**

counter\_8x **<=** 13'b0**;**

count\_8x\_ready\_reg **<=** 1**;**

**if(**counter\_baud **==** 7**)** **begin**

counter\_baud **<=** 3'b0**;**

count\_baud\_ready\_reg **<=** 1**;**

**end**

**else** **begin**

counter\_baud **<=** counter\_baud **+** 1**;**

count\_baud\_ready\_reg **<=** 0**;**

**end**

**end**

**else** **begin**

counter\_8x **<=** counter\_8x **+** 1**;**

count\_8x\_ready\_reg **<=** 0**;**

count\_baud\_ready\_reg **<=** 0**;**

**end**

**end**

**else** **begin**

count\_8x\_ready\_reg **<=** 0**;**

counter\_8x **<=** 13'b0**;**

counter\_baud **<=** 13'b0**;**

count\_baud\_ready\_reg **<=** 0**;**

**end**

**end**

**assign** count\_8x\_ready **=** count\_8x\_ready\_reg**;**

**assign** count\_baud\_ready **=** count\_baud\_ready\_reg**;**

**endmodule**

First, I defined our internal 100 MHz frequency for frequency division in the module. Then I created counters for both 8x and baud rate. Since 8x will have higher frequency and lower period, I calculated it first and gave it to the output. Then, when 8x becomes logical 1 every 8 times, I obtained the desired baud rate. For the operations to take place, that is, for the frequency division operation to take place, baud\_en must be 1. If it becomes zero and then becomes one again, the counting process starts over. I used this in the receiver.

**Simulation Source**

**module** baud\_gen\_tb**();**

// Parameters

**parameter** **[**16**:**0**]** freq **=** 115200**;**

// Testbench Signals

**reg** clk **=** 1'b0**;** // Clock

**reg** rst **=** 1'b0**;** // Reset

**reg** baud\_en **=** 1'b0**;** // Enable signal

**wire** count\_8x\_ready**;** // 8x clock ready

**wire** count\_baud\_ready**;** // Baud clock ready

// Instantiate the DUT (Device Under Test)

baud\_gen **#(.**freq**(**freq**))** uut **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**baud\_en**(**baud\_en**),**

**.**count\_8x\_ready**(**count\_8x\_ready**),**

**.**count\_baud\_ready**(**count\_baud\_ready**)**

**);**

// Generate a 100 MHz clock (period = 10 ns)

**always** **begin**

**#**5 clk **=** **~**clk**;**

**end**

**initial** **begin**

rst **=** 1**;**

**#**10**;**

rst **=** 0**;**

baud\_en **=** 1**;**

**#**11000**;**

baud\_en **=** 0**;**

**#**200**;**

$finish**;**

**end**

**endmodule**

**Simulation Waveform**

ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturulduFirst, I will simulate the baud rate for 115200 Hz frequency.

As you can see in the picture, the code is working correctly.

ekran görüntüsü, metin, çizgi içeren bir resim

Açıklama otomatik olarak oluşturulduSecond,I will simulate the baud rate for 9600 Hz frequency.

As you can see in the picture, the code is working correctly.

**UART TRANSMITTER**

**DESIGN SOURCE**

**module** uart\_tx **#(parameter** **[**16**:**0**]** freq**=** 115200**)**

**(**

**input** clk**,**

**input** rst**,**

**input** **[**7**:**0**]** d\_in**,**

**input** tx\_en**,**

**output** seri\_out**,**

**output** **reg** start**,**

**output** **reg** busy**,**

**output** done

**);**

**wire** count\_baud\_ready**,**out\_of\_use**;**

**reg** done\_reg**,** seri\_out\_reg**,**baud\_en**;**

**reg** **[**2**:**0**]** bit\_in**;**

**reg** **[**7**:**0**]** buffer**;**

**localparam** IDLE **=** 2'b00**;**

**localparam** START **=** 2'b01**;**

**localparam** DATA **=** 2'b10**;**

**localparam** STOP **=** 2'b11**;**

**reg** **[**1**:**0**]** state**;**

baud\_gen **#(.**freq**(**freq**))** divider

**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**baud\_en**(**baud\_en**),**

**.**count\_8x\_ready**(**out\_of\_use**),**

**.**count\_baud\_ready**(**count\_baud\_ready**)**

**);**

**always** **@(posedge** clk**)** **begin**

**if** **(**rst **==** 1'b1**)** **begin**

state **<=** IDLE**;**

done\_reg **<=** 0**;**

start **<=** 0**;**

busy **<=** 0**;**

seri\_out\_reg **<=** 1**;**

baud\_en **<=** 0**;**

**end**

**else** **begin**

**case(**state**)**

IDLE**:** **begin**

done\_reg **<=** 0**;**

seri\_out\_reg **<=** 1**;**

baud\_en **<=** 0**;**

buffer **<=** d\_in**;**

**if** **(**tx\_en **==** 1**)** **begin**

state **<=** START**;**

bit\_in **<=** 0**;**

**end**

**end**

START**:** **begin**

start **<=** 1**;**

seri\_out\_reg **<=** 0**;**

baud\_en **<=** 1**;**

**if(**count\_baud\_ready **==** 1**)** **begin**

state **<=** DATA**;**

**end**

**end**

DATA**:** **begin**

start **<=** 0**;**

busy **<=**1**;**

seri\_out\_reg **<=** buffer**[**0**];**

**if(**count\_baud\_ready **==** 1**)** **begin**

**if(**bit\_in **==** 7**)** **begin**

state **<=** STOP**;**

bit\_in **<=** 0**;**

**end**

**else** **begin**

buffer**[**6**:**0**]** **<=** buffer**[**7**:**1**];**

bit\_in **<=** bit\_in **+** 1**;**.

**end**

**end**

**end**

STOP**:** **begin**

seri\_out\_reg **<=** 1**;**

**if(**count\_baud\_ready **==**1**)** **begin**

state **<=** IDLE**;**

baud\_en **<=** 0**;**

busy **<=**0**;**

done\_reg **<=** 1**;**

**end**

**end**

**default:** **begin**

state **<=** IDLE**;**

done\_reg **<=** 0**;**

seri\_out\_reg **<=** 1**;**

baud\_en **<=** 0**;**

**end**

**endcase**

**end**

**end**

**assign** done **=** done\_reg**;**

**assign** seri\_out **=** seri\_out\_reg**;**

**endmodule**

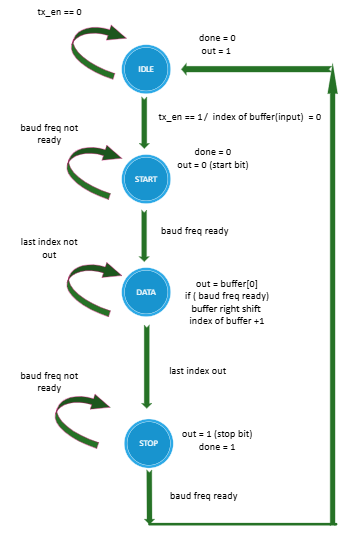
Since I made the transmitter before writing the top module, I had to instantiate the baud\_gen module in the transmitter. When I made the top module, I connected them all in the top module.

Our transmitter normally remains in IDLE state. If we make each enable signal logical 1, the system starts working and goes to the START state.

In the START state, our output becomes logical 0 to create the start bit during 1 baud frequency. Then it goes to DATA state. In DATA, first the first value of the input is given to the output and waits for 1 baud frequency. Then, with the expression “buffer[6:0] <= buffer[7:1];”, I shift the input that I assigned to the buffer one bit to the right and give it to the output. It also waits for 1 baud frequency. This process goes until the last bit of the input. Then it goes to STOP state.

In the stop state, the stop bit, logical 1, is given to the output during 1 baud frequency. Finally, the done output becomes 1 and the process ends.

**Finite State Machine Transmitter**

****

**Simulation Source**

**module** uart\_tx\_tb**;**

**parameter** **[**16**:**0**]** freq **=** 115200**;**

**reg** clk **=** 1'b0**;**

**reg** rst **=** 1'b0**;**

**reg** **[**7**:**0**]** d\_in**;**

**reg** tx\_en **=** 1'b0**;**

**wire** seri\_out**;**

**wire** done**;**

**integer** file**,** r**;**

**reg** **[**7**:**0**]** stimulus\_data**;**

**integer** i**;**

uart\_tx **#(.**freq**(**freq**))** uut **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**d\_in**(**d\_in**),**

**.**tx\_en**(**tx\_en**),**

**.**seri\_out**(**seri\_out**),**

**.**done**(**done**)**

**);**

// Saat sinyali üretimi

**always** **#**5 clk **=** **~**clk**;** // 10ns periodlu (100 MHz)

**initial** **begin**

file **=** $fopen**(**"stimulus2.txt"**,** "r"**);**

**if** **(**file **==** 0**)** **begin**

$display**(**"Error: Cannot open stimulus2.txt"**);**

$finish**;**

**end**

rst **=** 1**;**

d\_in **=** 8'b0**;**

tx\_en **=** 0**;**

**#**8000**;**

rst **=** 0**;**

**for** **(**i **=** 0**;** i **<** 4**;** i **=** i **+** 1**)** **begin**

r **=** $fscanf**(**file**,** "%b\n"**,** stimulus\_data**);**

**if** **(**r **==** 1**)** **begin**

d\_in **=** stimulus\_data**;**

tx\_en **=** 1**;**

**#**10**;**

tx\_en **=** 0**;**

**wait** **(**done **==** 1**);**

**#**15000**;**

**end**

**end**

$fclose**(**file**);**

$finish**;**

**end**

**endmodule**

I wrote python code to generate random numbers. In the simulation, I took 4 data from the file I created in python and checked it.

**import** random

**def** gen\_binary**():**

**with** **open(**"stimulus2.txt"**,**"w"**)** **as** f**:**

**for** k **in** **range(**0**,**4**):**

number **=** random**.***getrandbits***(**8**)**

bin\_num **=** **format(**number**,** "0b"**)**

**print(len(**bin\_num**))**

**if** **len(**bin\_num**)<**8**:**

**for** i **in** **range(**0**,**8**-len(**bin\_num**)):**

bin\_num **=** "0" **+** **str(**bin\_num**)**

f**.***write***(**bin\_num **+**"\n"**)**

**print(**"----------------"**)**

**Phyton Code**

10110100

01101110

11011000

10110011

**File Content**

**Simulation Waveform**

ekran görüntüsü, çizgi, renklilik içeren bir resim

Açıklama otomatik olarak oluşturuldu

As seen in the picture, the code works correctly. I took 4 different 8-bit random numbers from the file. I tested it and serial\_out gave the correct output.

**UART RECEIVER**

**DESIGN SOURCE**

**module** uart\_rx **#(parameter** **[**16**:**0**]** freq**=** 115200**)**

**(**

**input** clk**,**

**input** rst**,**

**input** d\_in**,**

**input** rx\_en**,**

**output** **[**7**:**0**]** d\_out**,**

**output** reg start**,**

**output** reg busy**,**

**output** done

**);**

**reg** **[**3**:**0**]** count\_ones**;**

**reg** **[**3**:**0**]** count\_zeros**;**

**reg** **[**2**:**0**]** counter\_halfperiod**;**

**wire** count\_baud\_ready**,**count\_8x\_ready**;**

**reg** done\_reg**,**baud\_en**;**

**reg** **[**7**:**0**]** d\_out\_reg**;**

**reg** **[**3**:**0**]** bit\_in**;**

**localparam** IDLE **=** 2'b00**;**

**localparam** START **=** 2'b01**;**

**localparam** DATA **=** 2'b10**;**

**localparam** STOP **=** 2'b11**;**

**reg** **[**1**:**0**]** state**;**

baud\_gen **#(.**freq**(**freq**))** divider

**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**baud\_en**(**baud\_en**),**

**.**count\_8x\_ready**(**count\_8x\_ready**),**

**.**count\_baud\_ready**(**count\_baud\_ready**)**

**);**

**always** **@(posedge** clk**)** **begin**

**if** **(**rst **==** 1'b1**)** **begin**

state **<=** IDLE**;**

done\_reg **<=** 0**;**

start **<=** 0**;**

busy **<=** 0**;**

baud\_en **<=** 0**;**

counter\_halfperiod **<=** 2'b0**;**

count\_ones **<=** 0**;**

count\_zeros **<=** 0**;**

**end**

**else** **begin**

**case(**state**)**

IDLE**:** **begin**

done\_reg **<=** 0**;**

baud\_en **<=** 0**;**

**if** **(**rx\_en **==** 1**)** **begin**

state **<=** START**;**

bit\_in **<=** 0**;**

start **<=**1**;**

**end**

**end**

START**:** **begin**

**if(**d\_in **==** 0**)** **begin**

baud\_en **<=** 1**;**

**if(**counter\_halfperiod **<** 4**)** **begin**

**if(**count\_8x\_ready **==** 1**)** **begin**

counter\_halfperiod **=** counter\_halfperiod **+** 1**;**

**end**

**end**

**else** **begin**

state **<=** DATA**;**

busy **<=**1**;**

start **<=** 0**;**

counter\_halfperiod **=** 2'b0**;**

baud\_en **<=** 0**;**

**end**

**end**

**end**

DATA**:** **begin**

baud\_en **<=** 1**;**

**if(**count\_baud\_ready **==** 1**)** **begin**

**if(**count\_ones **>** count\_zeros**)** **begin**

d\_out\_reg **=** **{**1'b1**,**d\_out\_reg**[**7**:**1**]};**

**end**

**if(**count\_ones **<** count\_zeros**)** **begin**

d\_out\_reg **=** **{**1'b0**,**d\_out\_reg**[**7**:**1**]};**

**end**

**if(**bit\_in **==** 7**)** **begin**

busy **<=** 0**;**

state **<=** STOP**;**

bit\_in **<=** 0**;**

**end**

**else** **begin**

bit\_in **<=** bit\_in **+** 1**;**

**end**

count\_ones **=** 3'b0**;**

count\_zeros **=** 3'b0**;**

**end**

**else** **begin**

**if(**count\_8x\_ready **==** 1**)** **begin**

**if(**d\_in **==** 1**)** **begin**

count\_ones **=** count\_ones **+** 1**;**

**end**

**else** **begin**

count\_zeros **=** count\_zeros **+** 1**;**

**end**

**end**

**end**

**end**

STOP**:** **begin**

**if(**count\_baud\_ready **==** 1**)** **begin**

state **<=** IDLE**;**

baud\_en **<=** 0**;**

done\_reg **<=** 1**;**

**end**

**end**

**default:** **begin**

state **<=** IDLE**;**

done\_reg **<=** 0**;**

d\_out\_reg **<=** 1**;**

baud\_en **<=** 0**;**

**end**

**endcase**

**end**

**end**

**assign** done **=** done\_reg**;**

**assign** d\_out **=** d\_out\_reg**;**

**endmodule**

Since I made the receiver before writing the top module, I had to instantiate the baud\_gen module in the receiver. When I made the top module, I connected them all in the top module.

Receiver is normally in IDLE state. When rx\_en gets logic 1, it switches to START state and the system starts to work.

In START state, it waits for half of the baud period when it detects the zero (start) bit. I measured half of the baud period with 8x baud frequency 4 times. After waiting half of the baud period, it switches to DATA state.

In the DATA state, it samples from the d\_in input every 8x frequency. When it receives 8 times (when the baud frequency is ready), it compares the number of ones with the number of zeros. Whichever one is more, it outputs it. When all values in the input are read (I checked with bit\_in), it switches to STOP state.

In STOP state, it waits for the baud period. Then it gives the stop bit (logic 1) and returns to IDLE state.

**SIMULATION WAVEFORM**

**ekran görüntüsü, renklilik, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Because d\_out has a value of X when the script starts running. It gives X value to the bits that are not filled until it is filled with the first 8 bits. Then it recovers. The top photo shows that it gives the result of the first data correctly. The value we give is the test\_data value, we give this value in series with d\_in and see it at the output in parallel with d\_out.

**ekran görüntüsü, renklilik, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**

I gave 4 different values in the top photo. The code correctly extracted them all in parallel.

**FINITE STATE MACHINE**

metin, ekran görüntüsü, diyagram, daire içeren bir resim

Açıklama otomatik olarak oluşturuldu

**SIMULATION SOURCE**

**module** uart\_rx\_tb**;**

**parameter** freq **=** 115200**;** // Baud rate

**reg** clk **=** 0**;**

**reg** rst **=** 0**;**

**reg** d\_in **=** 1**;** // IDLE STATE

**reg** rx\_en **=** 0**;**

**wire** **[**7**:**0**]** d\_out**;**

**wire** done**;**

**reg** **[**7**:**0**]** test\_data**;**

**reg** **[**7**:**0**]** expected\_data**;**

**integer** file**,** i**;**

// Clock Ã¼retimi (100 MHz clock, 10 ns period)

**always** **#**5 clk **=** **~**clk**;**

// Test edilen UART RX modÃ¼lÃ¼

uart\_rx **#(.**freq**(**freq**))** uut **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**d\_in**(**d\_in**),**

**.**rx\_en**(**rx\_en**),**

**.**d\_out**(**d\_out**),**

**.**done**(**done**)**

**);**

// UART veri gÃ¶nderim simÃ¼lasyonu

**task** send\_uart\_byte**(input** **[**7**:**0**]** para\_in**);**

**integer** i**;**

**begin**

// Start bit (0)

d\_in **=** 0**;**

rx\_en **=** 1**;**

**#(**8640**);**

// Data bitleri (LSB'den MSB'ye)

**for** **(**i **=** 0**;** i **<** 8**;** i **=** i **+** 1**)** **begin**

d\_in **=** para\_in**[**i**];**

**#(**8640**);**

**end**

// Stop bit (1)

d\_in **=** 1**;**

**#(**8640**);**

**end**

**endtask**

// Test senaryosu

**initial** **begin**

file **=** $fopen**(**"stimulus2.txt"**,** "r"**);**

**if** **(**file **==** 0**)** **begin**

$display**(**"Error: Cannot open stimulus2.txt"**);**

$finish**;**

**end**

$display**(**"Starting UART RX testbench..."**);**

rst **=** 1**;**

**#**10000**;**

rst **=** 0**;**

rx\_en **=** 0**;**

**#**45**;**

i **=** 0**;**

**while** **(!**$feof**(**file**))** **begin**

$fscanf**(**file**,** "%b\n"**,** test\_data**);**

expected\_data **=** test\_data**;**

rx\_en **=** 1**;** // RX etkinleştir

send\_uart\_byte**(**test\_data**);**

**#**100**;**

rx\_en **=** 0**;** // Yeni veri için devreyi sıfırla

**#**60**;**

**if** **(**d\_out **==** expected\_data**)** **begin**

$display**(**"Test %0d Passed! Sent: %b, Received: %b"**,** i **+** 1**,** expected\_data**,** d\_out**);**

**end** **else** **begin**

$display**(**"Test %0d Failed! Sent: %b, Received: %b"**,** i **+** 1**,** expected\_data**,** d\_out**);**

**end**

i **=** i **+** 1**;** // Sayaç doğru artırılıyor

**end**

$display**(**"Test completed!"**);**

$fclose**(**file**);**

$finish**;**

**end**

**endmodule**

I created a task to test the receiver in the simulation code. I tested it by taking 4 data from the file I created with the Phyton code. I compared the result I obtained with the result that should be and printed it to the console.

**TOP MODULE**

**DESIGN SOURCE**

**module** uart\_top **#(parameter** **[**16**:**0**]** freq **=** 115200**)** **(**

**input** clk**,**

**input** rst**,**

**input** **[**7**:**0**]** data\_in\_tx**,**

**input** tx\_en**,**

**input** rx\_en**,**

**output** **[**7**:**0**]** data\_out\_rx**,**

**output** tx\_done**,**

**output** rx\_done**,**

**output** tx\_start**,**

**output** rx\_start**,**

**output** tx\_busy**,**

**output** rx\_busy**,**

**output** tx\_out

**);**

**wire** tx\_baud\_en**,**tx\_count\_baud\_ready**;**

**wire** rx\_baud\_en**,**rx\_count\_baud\_ready**,**rx\_count\_8x\_ready**;**

// RX için baud gen modülü

baud\_gen **#(.**freq**(**freq**))** rx\_baud\_gen **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**baud\_en**(**rx\_baud\_en**),**

**.**count\_8x\_ready**(**rx\_count\_8x\_ready**),**

**.**count\_baud\_ready**(**rx\_count\_baud\_ready**)**

**);**

// TX için baud gen modülü

baud\_gen **#(.**freq**(**freq**))** tx\_baud\_gen **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**baud\_en**(**tx\_baud\_en**),**

**.**count\_8x\_ready**(),**

**.**count\_baud\_ready**(**tx\_count\_baud\_ready**)**

**);**

// UART alıcı modülü (RX)

uart\_rx **#(.**freq**(**freq**))** rx\_inst **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**d\_in**(**tx\_out**),**

**.**rx\_en**(**rx\_en**),**

**.**baud\_en**(**rx\_baud\_en**),**

**.**count\_8x\_ready**(**rx\_count\_8x\_ready**),**

**.**start**(**rx\_start**),**

**.**busy**(**rx\_busy**),**

**.**count\_baud\_ready**(**rx\_count\_baud\_ready**),**

**.**d\_out**(**data\_out\_rx**),**

**.**done**(**rx\_done**)**

**);**

// UART verici modülü (TX)

uart\_tx **#(.**freq**(**freq**))** tx\_inst **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**d\_in**(**data\_in\_tx**),**

**.**tx\_en**(**tx\_en**),**

**.**count\_baud\_ready**(**tx\_count\_baud\_ready**),**

**.**start**(**tx\_start**),**

**.**busy**(**tx\_busy**),**

**.**baud\_en**(**tx\_baud\_en**),**

**.**seri\_out**(**tx\_out**),** // Seri çıkış

**.**done**(**tx\_done**)** // Verici işlem tamamlandı sinyali

**);**

**endmodule**

I instantiate a transmitter, a receiver and two baud\_gen. A baud\_gen for receiver other for transmitter. I made all connections in top module. In the top module, I receive 8-bit parallel data and give it to the input of the transmitter. I give the output of the transmitter to the input of the receiver. I give the output of the receiver to the output of the top module.

**SIMULATION SOURCE**

I give the file I created in Phyton to the transmitter input. Finally, I compare the data\_out I get from the receiver with the data from the file.

**module** uart\_top\_tb**();**

**reg** clk **=** 1'b0**;**

**reg** rst **=** 1'b0**;**

**reg** **[**7**:**0**]** data\_in\_tx**;**

**reg** tx\_en **=** 0**;**

**reg** rx\_en **=** 0**;**

**wire** **[**7**:**0**]** data\_out\_rx**;**

**wire** tx\_done**;**

**wire** rx\_done**;**

**wire** tx\_busy**;**

**wire** rx\_busy**;**

**wire** tx\_start**;**

**wire** rx\_start**;**

**wire** tx\_out**;**

**integer** i**;**

// UART top modül örneği

uart\_top uut **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**data\_in\_tx**(**data\_in\_tx**),**

**.**tx\_en**(**tx\_en**),**

**.**rx\_en**(**rx\_en**),**

**.**data\_out\_rx**(**data\_out\_rx**),**

**.**tx\_done**(**tx\_done**),**

**.**rx\_done**(**rx\_done**),**

**.**tx\_busy**(**tx\_busy**),**

**.**rx\_busy**(**rx\_busy**),**

**.**tx\_start**(**tx\_start**),**

**.**rx\_start**(**rx\_start**),**

**.**tx\_out**(**tx\_out**)**

**);**

**integer** file**;**

**reg** **[**7**:**0**]** stimulus\_data**[**0**:**3**];**

// Saat sinyali üretimi

**always** **#**5 clk **=** **~**clk**;** // 10ns periyotlu (100 MHz) saat

**initial** **begin**

file **=** $fopen**(**"stimulus2.txt"**,** "r"**);**

**if** **(**file **==** 0**)** **begin**

$display**(**"Error: Cannot open stimulus2.txt"**);**

$finish**;**

**end**

**for** **(**i **=** 0**;** i **<** 4**;** i **=** i **+** 1**)** **begin**

$fscanf**(**file**,** "%b\n"**,** stimulus\_data**[**i**]);**

**end**

$fclose**(**file**);**

rst **=** 1**;**

**#**20000**;**

rst **=** 0**;**

tx\_en **=** 0**;**

rx\_en **=** 0**;**

**for** **(**i **=** 0**;** i **<** 4**;** i **=** i **+** 1**)** **begin**

data\_in\_tx **=** stimulus\_data**[**i**];**

tx\_en **=** 1**;**

**#**10**;**

tx\_en **=** 0**;**

rx\_en **=** 1**;**

**wait(**tx\_done**);**

**#**1000**;**

**if** **(**data\_out\_rx **==** stimulus\_data**[**i**])** **begin**

$display**(**"Data received correctly: %h"**,** data\_out\_rx**);**

**end** **else** **begin**

$display**(**"Error: Received data %h, expected %h"**,** data\_out\_rx**,** stimulus\_data**[**i**]);**

**end**

**#**20000**;**

**end**

$finish**;** // Test tamamlandı

**end**

**endmodule**

**SIMULATION WAVEFORM**

ekran görüntüsü, renklilik içeren bir resim

Açıklama otomatik olarak oluşturuldu

I receive the data from the file with data\_in\_tx and receive it from the transmitter. I give tx\_out to the output in serial. At the same time, I give the tx\_out output of the transmitter to the input of the receiver. Finally, the data coming in with data\_in\_tx comes out of the receiver correctly as data\_out\_tx.

**BLOCK DIAGRAM**

metin, diyagram, çizgi, plan içeren bir resim

Açıklama otomatik olarak oluşturuldu

**UTILIZATION REPORTS**

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Our WHS value is 5.839 ns. Therefore, 10 – 5.839 = 4.161 ns. Maximum clk frequency is

**1/ 4.161 ns = 240 MHz**

ekran görüntüsü, metin, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**POST – TIMING IMPLEMENTATION SIMULATION**

Top Module Simulation

yazılım, metin, multimedya yazılımı, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Baud Generator Simulation

metin, yazılım, multimedya yazılımı, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Receiver Simulation

yazılım, multimedya yazılımı, metin, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Transmitter Simulation

**REFERENCES**

[1] Peňa, E., & Legaspi, M. G. (n.d.). *UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter*. Analog Devices. <https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-> communication-protocol.html

[2] Kelvin. (2022, December 15). *UART Communication Protocol and how it works - latest open tech from SEEED*. Latest Open Tech From Seeed. https://www.seeedstudio.com/blog/2022/09/08/uart-communication-protocol-and-how-it-works/

[3] *01signal: Vivado: Finding the “maximal frequency” after synthesis*. (n.d.). https://www.01signal.com/vendor-specific/xilinx/vivado-minimal-period/

[4] [Mehmet Burak Aykenar]. (2021, January 11). *VHDL ile FPGA PROGRAMLAMA - Ders18: UART Receiver Tasarımı ve Simülasyonu - Sıfırdan Kod Yazma* [Video]. Youtube. <https://www.youtube.com/watch?v=d7XPkZDWWQY&t=1052s>

[5] [[Mehmet Burak Aykenar]. (2021, January 8). *VHDL ile FPGA PROGRAMLAMA - Ders16: VHDL UART Transmitter Sıfırdan Kod Yazma* [Video]. YouTube. <https://www.youtube.com/watch?v=5MzjKMZfLTk&t=1904s>

[6] [Muhammed Kocaoğlu]. (2023, April 2). *SystemVerilog - UART Transmitter* [Video]. Youtube. <https://www.youtube.com/watch?v=bh_RxlzEnvU&t=757s>

[7] [Muhammed Kocaoğlu]. (2023, April 14). *SystemVerilog - UART Receiver* [Video]. Youtube. <https://www.youtube.com/watch?v=va-q8QKqeMg&t=596s>