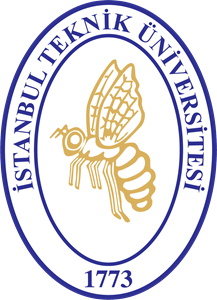
**DIGITAL SYSTEM DESIGN APPLICATIONS**

**(CRN: 11275)**

**THE REPORT OF PROJECT – 2**



Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

Yusuf Tekin – 040200043

1. **Baud Rate Gen**

Figure 1 - Baud Rate Gen Verilog Code

`timescale 1ns **/** 1ps

**module** baud\_gen **#(parameter** Rx\_resolution **=** 16**)(**

**input** clk**,**

**input** rst**,**

**input** BR\_mode**,** // 0 -> 9600 and 1 -> 115200

**output** **reg** clk\_Tx**,**

**output** **reg** clk\_Rx

**);**

**parameter** clk\_freq **=** 100000000**;** //100MHz

**localparam** DIV\_9600 **=** **(**clk\_freq **/** 9600**)** **-**1**;**

**localparam** DIV\_9600\_Rx **=** **(**clk\_freq **/** **(**9600 **\*** Rx\_resolution**))** **-**1**;**

**localparam** DIV\_115200 **=** **(**clk\_freq **/** 115200**)** **-**1**;**

**localparam** DIV\_115200\_Rx **=** **(**clk\_freq **/** **(**115200 **\*** Rx\_resolution**))** **-**1**;**

**reg** **[**15**:**0**]** baud\_count **=** 0**;**

**reg** **[**15**:**0**]** baud\_count\_Rx **=** 0**;**

**reg** **[**15**:**0**]** baud\_limit**;**

**reg** **[**15**:**0**]** baud\_limit\_Rx**;**

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

**if(**BR\_mode**)** **begin**

baud\_limit **<=** DIV\_115200**;**

baud\_limit\_Rx **<=** DIV\_115200\_Rx**;**

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

baud\_limit **<=** DIV\_9600**;**

baud\_limit\_Rx **<=** DIV\_9600\_Rx**;**

**end**

**end**

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

**if(**rst**)** **begin**

baud\_count **<=** 0**;**

clk\_Tx **<=** 1'b0**;**

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

**if(**baud\_count **<** baud\_limit**)** **begin**

baud\_count **<=** baud\_count **+**1**;**

clk\_Tx **<=** 1'b0**;**

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

baud\_count **<=** 0**;**

clk\_Tx **<=** 1'b1**;**

**end**

**end**

**end**

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

**if(**rst**)** **begin**

baud\_count\_Rx **<=** 0**;**

clk\_Rx **<=** 1'b0**;**

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

**if(**baud\_count\_Rx **<** baud\_limit\_Rx**)** **begin**

baud\_count\_Rx **<=** baud\_count\_Rx **+**1**;**

clk\_Rx **<=** 1'b0**;**

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

baud\_count\_Rx **<=** 0**;**

clk\_Rx **<=** 1'b1**;**

**end**

**end**

**end**

**endmodule**

Figure 2 - Baud Rate Gen Testbench Code

`timescale 1ns **/** 1ps

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

**wire** clk\_Rx**,** clk\_Tx**;**

**reg** rst**,** clk**,** BR\_mode**;**

baud\_gen **#(.**Rx\_resolution**(**16**))** uut**(**

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

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

**.**BR\_mode**(**BR\_mode**),**

**.**clk\_Tx**(**clk\_Tx**),**

**.**clk\_Rx**(**clk\_Rx**)**

**);**

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

**initial** **begin**

clk **=** 1'b0**;** rst **=** 1'b1**;** BR\_mode **=** 1'b0**;**

**#**10**;** rst **=** 1'b0**;**

**#**1000000**;**

rst **=** 1'b1**;** BR\_mode **=** 1'b1**;**

**#**10**;** rst **=** 1'b0**;**

**#**1000000**;**

$finish**();**

**end**

**endmodule**

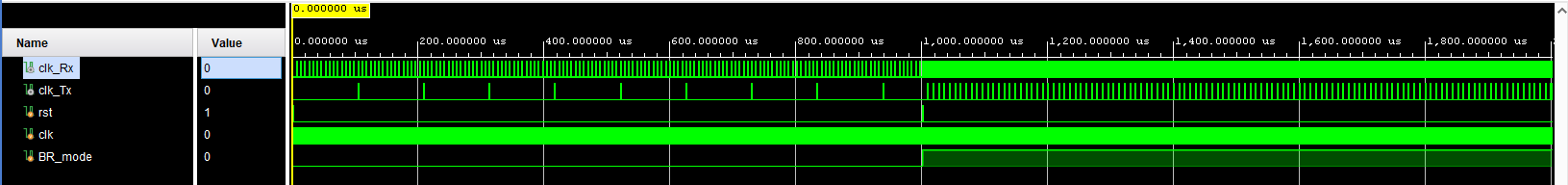


Figure 3 - Baud Rate Gen Behavioral Simulation

In this baud rate generator design the resolution of the receiver of the UART is parametric and chosen to be 16 for the continuation of the project. To be able to switch between the 9600 Hz and 115200 Hz baud rate clocks, there is a BR\_mode input implemented in the design. For this 1-bit input “0” chooses 9600 Hz and 1 chooses 115200 Hz. In the behavioral simulation it can be seen that the clocks are working as intended with the 1/16 ratio.

1. **Transmitter Unit**

Figure 4 - Transmitter Unit Verilog Code

`timescale 1ns **/** 1ps

**module** uart\_tx**(**

**input** clk\_Tx**,**

**input** clk**,**

**input** rst**,**

**input** Tx\_enable**,**

**input** **[**7**:**0**]** Tx\_data\_in**,**

**output** **reg** Tx\_data\_out**,**

**output** **reg** start\_flag**,**

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

**output** **reg** done

**);**

**reg** **[**3**:**0**]** bit\_index **=** 4'b0**;**

**reg** **[**7**:**0**]** data\_buffer **=** 8'b0**;**

**reg** **[**1**:**0**]** state **=** 2'b00**;**

**parameter** IDLE **=** 2'b00**,** START **=** 2'b01**,**

DATA **=** 2'b10**,** STOP **=** 2'b11**;**

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

**if(**rst**)** **begin**

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

Tx\_data\_out **<=** 1'b1**;**

start\_flag **<=** 1'b0**;**

busy **<=** 1'b0**;**

done **<=** 1'b0**;**

bit\_index **<=** 4'b0**;**

**end** **else** **if(**clk\_Tx**)** **begin**

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

IDLE**:** **begin**

Tx\_data\_out **<=** 1'b1**;**

busy **<=** 1'b0**;**

done **<=** 1'b0**;**

bit\_index **<=** 4'b0**;**

**if(**Tx\_enable**)** **begin**

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

data\_buffer **<=** Tx\_data\_in**;**

Tx\_data\_out **<=** 1'b0**;**

start\_flag **<=** 1'b1**;**

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

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

**end**

**end**

START**:** **begin**

Tx\_data\_out **<=** data\_buffer**[**bit\_index**];**

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

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

start\_flag **<=** 1'b0**;**

busy **<=** 1'b1**;**

**end**

DATA**:** **begin**

start\_flag **<=** 1'b0**;**

**if(**bit\_index **<** 8**)** **begin**

Tx\_data\_out **<=** data\_buffer**[**bit\_index**];**

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

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

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

bit\_index **<=** 0**;**

busy **<=** 1'b0**;**

done **<=** 1'b1**;**

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

**end**

**end**

STOP**:** **begin**

Tx\_data\_out **<=** 1'b1**;**

data\_buffer **<=** 8'b0**;**

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

**end**

**default:** **begin**

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

**end**

**endcase**

**end**

**end**

**endmodule**

Figure 5 - Transmitter Unit Testbench Code

`timescale 1ns **/** 1ps

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

**wire** done**,** busy**,** start\_flag**,** Tx\_data\_out**;**

**reg** clk**,** clk\_Tx**,** rst**,** enable**;**

**reg** **[**7**:**0**]** Tx\_data\_in**;**

**reg** **[**15**:**0**]** baud\_count **=** 0**;**

**reg** **[**15**:**0**]** baud\_limit**;**

**localparam** DIV\_9600 **=** **(**100000000 **/** 9600**)** **-**1**;**

uart\_tx uut**(**

**.**clk\_Tx**(**clk\_Tx**),**

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

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

**.**Tx\_enable**(**enable**),**

**.**Tx\_data\_in**(**Tx\_data\_in**),**

**.**Tx\_data\_out**(**Tx\_data\_out**),**

**.**start\_flag**(**start\_flag**),**

**.**busy**(**busy**),**

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

**);**

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

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

**if(**rst**)** **begin**

baud\_count **<=** 0**;**

clk\_Tx **<=** 1'b0**;**

baud\_limit **<=** DIV\_9600**;**

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

**if(**baud\_count **<** baud\_limit**)** **begin**

baud\_count **<=** baud\_count **+**1**;**

clk\_Tx **<=** 1'b0**;**

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

baud\_count **<=** 0**;**

clk\_Tx **<=** 1'b1**;**

**end**

**end**

**end**

**initial** **begin**

clk **=** 1'b0**;** rst **=** 1'b1**;** **#**10**;**

rst **=** 1'b0**;** enable **=** 1'b1**;** **#**1000**;**

Tx\_data\_in **=** 8'b1100\_0011**;**

**#**1000000**;**

Tx\_data\_in **=** 8'b1111\_1111**;**

**#**1000000**;**

Tx\_data\_in **=** 8'b0000\_0000**;**

**#**1000000**;**

Tx\_data\_in **=** 8'b1010\_1010**;**

**#**1000000**;**

enable **=** 1'b0**;**

$finish**();**

**end**

**endmodule**

A black screen with green lines

Description automatically generated

Figure 6 - Transmitter Behavioral Simulation

The transmitter unit uses an input which is determined in the baud rate generator as a clock for the state transitions. At standby, the transmitter stays at the IDLE state until both Tx\_enable goes to logic-1 and the start bit goes to logic-0 while giving logic-1s to the output. When these two conditions are met at the next baud rated clock cycle the state machine operates in the START state which makes the start flag go high for a baud rated time. After the next baud rated clock hits high the 8-bit data already saved in the buffer starts to be given as the output in order (from least to most significant bit) and the busy signal goes high in the DATA state. At last, only the done signal goes high, and the output starts to give 1s until the state goes to IDLE and waits for the input again.

1. **Receiver Unit**

Figure 7 - Receiver Unit Verilog Code

`timescale 1ns **/** 1ps

**module** uart\_rx **(**

**input** clk**,**

**input** rst**,**

**input** clk\_Rx**,**

**input** Rx\_data\_in**,**

**input** Rx\_enable**,**

**output** **reg** **[**7**:**0**]** Rx\_data\_out**,**

**output** **reg** Rx\_start**,**

**output** **reg** Rx\_busy**,**

**output** **reg** Rx\_done

**);**

**parameter** IDLE **=** 2'b00**,** START **=** 2'b01**,**

DATA **=** 2'b10**,** STOP **=** 2'b11**;**

**reg** **[**1**:**0**]** state **=** 2'b00**;**

**reg** **[**3**:**0**]** sample\_count**;**

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

**reg** **[**7**:**0**]** data\_buffer**;**

**reg** **[**3**:**0**]** bit\_weight **=** 0**;**

**reg** error **=** 0**;**

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

**if(**rst**)** **begin**

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

Rx\_start **<=** 1'b0**;**

Rx\_busy **<=** 1'b0**;**

Rx\_done **<=** 1'b0**;**

Rx\_data\_out **<=** 8'b0**;**

sample\_count **<=** 4'b0**;**

bit\_index **<=** 4'b0**;**

data\_buffer **<=** 8'b0**;**

bit\_weight **<=** 4'b0**;**

error **<=** 1'b0**;**

**end** **else** **if(**clk\_Rx**)** **begin**

/\*if(error) begin

state <= IDLE;

end\*/

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

IDLE**:** **begin**

Rx\_busy **<=** 1'b0**;**

sample\_count **<=** 4'b0**;**

bit\_index **<=** 4'b0**;**

error **<=** 1'b0**;**

bit\_weight **<=** 4'b0**;**

**if(**Rx\_enable **&&** **~**Rx\_data\_in**)** **begin**

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

Rx\_start **<=** 1'b1**;**

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

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

**end**

**end**

START**:** **begin**

**if(**sample\_count **<** 7**)** **begin**

sample\_count **<=** sample\_count **+** 1**;**

bit\_weight **<=** bit\_weight **+** Rx\_data\_in**;**

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

sample\_count **<=** 0**;**

**if(**bit\_weight **<** 4**)** **begin** //If mostly zeros

sample\_count **<=** 0**;**

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

Rx\_start **<=** 1'b0**;**

Rx\_busy **<=** 1'b1**;**

bit\_weight **<=** 4'b0**;**

**end** **else** **begin** // If not mostly zeros

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

error **<=** 1'b1**;**

Rx\_start **<=** 1'b0**;**

**end**

**end**

**end**

DATA**:** **begin**

**if(**sample\_count **<** 7**)** **begin**

sample\_count **<=** sample\_count **+** 1**;**

bit\_weight **<=** bit\_weight **+** Rx\_data\_in**;**

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

sample\_count **<=** 0**;**

bit\_weight **<=** 4'b0**;**

**if(**bit\_index **<** 8**)** **begin**

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

**if(**bit\_weight **<** 4**)** **begin** //Majority Low: bit = 0

data\_buffer**[**bit\_index**]** **<=** 1'b0**;**

**end** **else** **begin** //Majority High: bit = 1

data\_buffer**[**bit\_index**]** **<=** 1'b1**;**

**end**

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

bit\_index **<=** 4'b0**;**

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

Rx\_busy **<=** 1'b0**;**

**end**

**end**

**end**

STOP**:** **begin**

**if** **(**sample\_count **<** 7**)** **begin**

sample\_count **<=** sample\_count **+** 1**;**

bit\_weight **<=** bit\_weight **+** Rx\_data\_in**;**

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

sample\_count **<=** 0**;**

bit\_weight **<=** 4'b0**;**

**if** **(**bit\_weight **>** 3**)** **begin** // Valid stop bit (majority high)

Rx\_data\_out **<=** data\_buffer**;** // Output received data

Rx\_done **<=** 1'b1**;**

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

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

error **<=** 1'b1**;** // Framing error

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

**end**

**end**

**end**

**endcase**

**end**

**end**

**endmodule**

Figure 8 - Receiver Unit Testbench Code

`timescale 1ns **/** 1ns

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

// Parameters for 9600 baud, 8x oversampling

**localparam** DIV\_9600\_Rx **=** **(**100000000 **/** **(**9600 **\*** 8**))** **-** 1**;**

**reg** **[**15**:**0**]** baud\_count\_Rx **=** 0**;**

**reg** **[**15**:**0**]** baud\_limit\_Rx**;**

**reg** clk**,** rst**,** clk\_Rx**,** Rx\_data\_in**,** Rx\_enable**;**

**wire** **[**7**:**0**]** Rx\_data\_out**;**

**wire** Rx\_start**,** Rx\_busy**,** Rx\_done**;**

uart\_rx uut **(**

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

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

**.**clk\_Rx**(**clk\_Rx**),**

**.**Rx\_data\_in**(**Rx\_data\_in**),**

**.**Rx\_enable**(**Rx\_enable**),**

**.**Rx\_data\_out**(**Rx\_data\_out**),**

**.**Rx\_start**(**Rx\_start**),**

**.**Rx\_busy**(**Rx\_busy**),**

**.**Rx\_done**(**Rx\_done**)**

**);**

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

// Baud clock generation: 8x baud clock for 9600 baud rate

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

**if** **(**rst**)** **begin**

baud\_count\_Rx **<=** 0**;**

clk\_Rx **<=** 1'b0**;**

baud\_limit\_Rx **<=** DIV\_9600\_Rx**;**

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

**if** **(**baud\_count\_Rx **<** baud\_limit\_Rx**)** **begin**

baud\_count\_Rx **<=** baud\_count\_Rx **+** 1**;**

clk\_Rx **<=** 1'b0**;**

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

baud\_count\_Rx **<=** 0**;**

clk\_Rx **<=** 1'b1**;**

**end**

**end**

**end**

**always** **@(\*)** **begin**

**if(**Rx\_done **==** 1'b1**)** **begin**

**if(**Rx\_data\_out **==** 8'b1110\_0001**)** **begin**

$display**(**"Frame 1: Received data is correct = %b"**,** Rx\_data\_out**);**

**end** **else** **if(**Rx\_data\_out **==** 8'b1010\_1010**)** **begin**

$display**(**"Frame 2: Received data is correct = %b"**,** Rx\_data\_out**);**

**end** **else** **if(**Rx\_data\_out **==** 8'b0000\_1111**)** **begin**

$display**(**"Frame 3: Received data is correct = %b"**,** Rx\_data\_out**);**

**end** **else** **if(**Rx\_data\_out **==** 8'b1111\_1111**)** **begin**

$display**(**"Frame 4: Received data is correct = %b"**,** Rx\_data\_out**);**

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

$display**(**"Received data is not correct = %b"**,** Rx\_data\_out**);**

**end**

**end**

**end**

**initial** **begin**

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

rst **=** 1'b1**;**

Rx\_data\_in **=** 1'b1**;**

**#**10**;**

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

Rx\_enable **=** 1'b1**;**

**#**1000**;**

// Frame 1: 8'b11100001 (Start bit = 0, Data = 11100001, Stop bit = 1)

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Start bit

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 0

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 1

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 2

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 3

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 4

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 5

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 6

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 7

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Stop bit

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

// Frame 2: 8'b10101010 (Start bit = 0, Data = 10101010, Stop bit = 1)

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Start bit

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 0

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 1

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 2

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 3

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 4

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 5

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 6

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 7

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Stop bit

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

// Frame 3: 8'b00001111 (Start bit = 0, Data = 00001111, Stop bit = 1)

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Start bit

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 0

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 1

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 2

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 3

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 4

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 5

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 6

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Bit 7

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Stop bit

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

// Frame 4: 8'b11111111 (Start bit = 0, Data = 00001111, Stop bit = 1)

Rx\_data\_in **=** 1'b0**;** **#**104167**;** // Start bit

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 0

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 1

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 2

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 3

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 4

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 5

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 6

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Bit 7

Rx\_data\_in **=** 1'b1**;** **#**104167**;** // Stop bit

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

Rx\_data\_in **=** 1'b1**;** **#**104167**;**

$finish**();**

**end**

**endmodule**

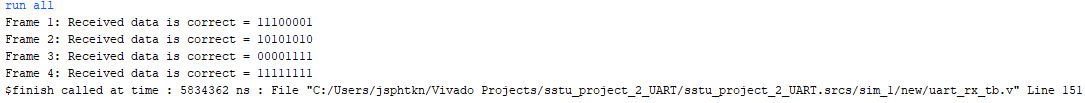


Figure 9 - Receiver Unit TCL Console Output

A screenshot of a computer screen

Description automatically generated

Figure 10 - Receiver Unit Behavioral Simulation

The receiver unit has a more complicated structure than the transmitter. A buffer is defined likewise however, the purpose of the buffer is to hold the 8-bit inputs one by one while shifting until the done signal goes high. To calculate the resolution of the input signals to determine whether the start or stop bits are correct, bit weight calculations are done. With the help of the error register, accuracy of the output signals can be calculated. In the design, the x8 baud calculations are done by using a sampling counter and the bit weight calculations. Since the resolution for each bit is 8, whether the bit weight is higher or lower than 4 is what determines the detected logic. The output and done signals are chosen to be held until the next start bit to make the observations easier. Furthermore, the testbench is designed to display TCL Console feedback for whether the output values are matching with the given inputs (Figure 9).

1. **Top Module**

Figure 11 - Top Module Verilog Code

`timescale 1ns **/** 1ps

**module** uart\_top**(**

**input** clk**,**

**input** rst**,**

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

**input** Tx\_en**,**

**input** Rx\_en**,**

**input** BR\_mode**,**

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

**);**

**wire** signal**,** clk\_Rx**,** clk\_Tx**;**

uart\_rx receiver\_1**(**

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

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

**.**clk\_Rx**(**clk\_Rx**),**

**.**Rx\_data\_in**(**signal**),**

**.**Rx\_enable**(**Rx\_en**),**

**.**Rx\_data\_out**(**data\_out**),**

**.**Rx\_start**(),**

**.**Rx\_busy**(),**

**.**Rx\_done**());**

uart\_tx transmitter\_1**(**

**.**clk\_Tx**(**clk\_Tx**),**

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

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

**.**Tx\_enable**(**Tx\_en**),**

**.**Tx\_data\_in**(**data\_in**),**

**.**Tx\_data\_out**(**signal**),**

**.**start\_flag**(),**

**.**busy**(),**

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

**);**

baud\_gen BR\_generator**(**

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

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

**.**BR\_mode**(**BR\_mode**),**

**.**clk\_Tx**(**clk\_Tx**),**

**.**clk\_Rx**(**clk\_Rx**)**

**);**

**endmodule**

Figure 12 - Top Module Texstbench Code

`timescale 1ns **/** 1ns

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

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

**reg** rst**,** clk**,** Tx\_en**,** Rx\_en**;**

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

**reg** **[**1**:**0**]** BR\_mode**;** // 0 -> 9600 and 1 -> 115200

uart\_top uut**(**

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

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

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

**.**Tx\_en**(**Tx\_en**),**

**.**Rx\_en**(**Rx\_en**),**

**.**BR\_mode**(**BR\_mode**),**

**.**data\_out**(**data\_out**)**

**);**

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

**initial** **begin**

clk **=** 1'b0**;** BR\_mode **=** 1'b0**;** //9600

Tx\_en **=** 1'b0**;** Rx\_en **=** 1'b0**;**

data\_in **=** 8'b0000\_0000**;**

rst **=** 1'b1**;** **#**10**;** rst **=** 1'b0**;**

Tx\_en **=** 1'b1**;** Rx\_en **=** 1'b1**;**

data\_in **=** 8'b1100\_0011**;** Tx\_en **=** 1'b1**;**

**#**1000000**;**

data\_in **=** 8'b1111\_1111**;** Tx\_en **=** 1'b1**;**

**#**1100000**;**

data\_in **=** 8'b0000\_0000**;** Tx\_en **=** 1'b1**;**

**#**1200000**;**

data\_in **=** 8'b1010\_1010**;** Tx\_en **=** 1'b1**;**

**#**750000**;** Tx\_en **=** 1'b0**;**

**#**1500000**;**

BR\_mode **=** 1'b1**;** //115200

Tx\_en **=** 1'b0**;** Rx\_en **=** 1'b0**;**

data\_in **=** 8'b0000\_0000**;**

rst **=** 1'b1**;** **#**10**;** rst **=** 1'b0**;**

Tx\_en **=** 1'b1**;** Rx\_en **=** 1'b1**;**

data\_in **=** 8'b1100\_0011**;**

**#**100000**;**

data\_in **=** 8'b1111\_1111**;**

**#**150000**;**

data\_in **=** 8'b0000\_0000**;**

**#**160000**;**

data\_in **=** 8'b1010\_1010**;**

**#**200000**;**

Tx\_en **=** 1'b0**;** Rx\_en **=** 1'b0**;**

**#**10000**;**

$finish**();**

**end**

**endmodule**

A green and blue lines on a black background

Description automatically generated

Figure 13 - Top Module Behavioral Simulation

The top module combines both transmitter and receiver with the addition of the baud generator. In the simulation (Figure 13) the inputs (195, 225, 0, 170) are given to the transmitter and expected to be obtained from receiver. The simulation switches the baud rates at the proper moment to test it out. In this testbench the output results are correct with both baud rates. However, if the delays between the inputs are given shorter, the receiver’s error register goes high, and the results are corrupted. Since the delay values are chosen long enough, the different phases chosen for the input delays do not affect the system. Which is what a asynchronous system should operate.

A screenshot of a graph

Description automatically generated

Figure 14 - Utilization Summary

A screenshot of a computer

Description automatically generated

Figure 15 - Timing Summary

A screenshot of a computer

Description automatically generated

Figure 16 - Post Implementation Timing Simulation

1. **Explaining UART**

UART (Universal Asynchronous Receiver-Transmitter) is a asynchronous serial communication device which the data format and transmission speeds are configurable. It has two main parts which are transmitter and receiver. The transmitter has multiple bits of input and a 1-bit output. It sends data bits one by one from the transmitter and receives one by one from the receiver. After receiving the data, the receiver stores that data in a buffer until the stop bit arrives. Just after receiving the stop bit, the output of the receiver reflects the buffer values. In this design 1-bit start bit, 8-bit data bits and 1-bit stop bit are used.

The transmission starts operating when both the enable is high, and the start bit comes. After sending 8 bits of data, it stops after the stop bit. The same system goes with the receiver, it starts with the start bit and stores data with 8 data bits until the stop bit occurs. This system doesn’t require a common clock to operate, however, both the transmitter and the receiver must operate at the same baud rate. When the receiver stores the data, the receiver samples the incoming signal at 8x the baud rate during each bit period. It determines the value of each bit by taking multiple samples and choosing the majority logic level.

A diagram of a block diagram

Description automatically generated

Figure 17 - Transmitter and Receiver State Diagram

A diagram of a computer wiring

Description automatically generated with medium confidence

Figure 18 - Block Diagram of the Top Design

**References**

* “KeyStone Architecture Universal Asynchronous Receiver/Transmitter (UART) User Guide.” Available: https://www.ti.com/lit/ug/sprugp1/sprugp1.pdf
* “Universal asynchronous receiver-transmitter,” *Wikipedia*. https://en.wikipedia.org/wiki/Universal\_asynchronous\_receiver-transmitter
* E. Pena and M. G. Legaspi, “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