Electrical & Computer Engineering Department
University of Thessaly

CE430 – Digital Circuits Laboratory
2nd Assignment 2018-2019
Victoria Biliouri 2138

UART – Universal Asynchronous Receiver Transmitter

UART is a serial, asynchronous, communication protocol, which is used for data tranfer between two, or more devices. These devices can have asynchronous clocks with each other. UART is widely known because it is simple and easy to use.

PURPOSE

The purpose of this project is the implementation of a serial communication system, using the UART protocol (Universal Asynchrocous Receiver Transmitter). This system consists of a UART Transmitter, who transfers data, and a UART Receiver, who receives data. The data that I have chosen to transmit, is a sequence of the four 8-bit symbols:

10101010 (AA) , 01010101 (55) , 11001100 (CC) , 10001001 (89)

In the last part of this project, the hexadecimal number of each symbol is displayed on the fpga screen, using the 7-segments indication driver I have already created.

In order to facilitate ourselves , we devided the project in four parts.

PART A – BAUD RATE CONTROLLER

In the first part I have created a Baud Rate Controller, which sets the rhythm that the data are transferred from the UART Transmitter and are sampled by the UART Receiver. This controller cannot be avoided, as its absence can cause the loss of our data or the creation of new, unexpected ones. In order for the system to know in which rhythm it should be working on, we use a signal (baud_select) that is defined at the beginning of the system function. The signal – baud rate matching is described below.
The creation of the sampling signal that we are using in the following parts, is achieved through a 14-bit counter, which increases until it reaches its maximum value, depending on the baud_select signal. The maximum value for this counter is:

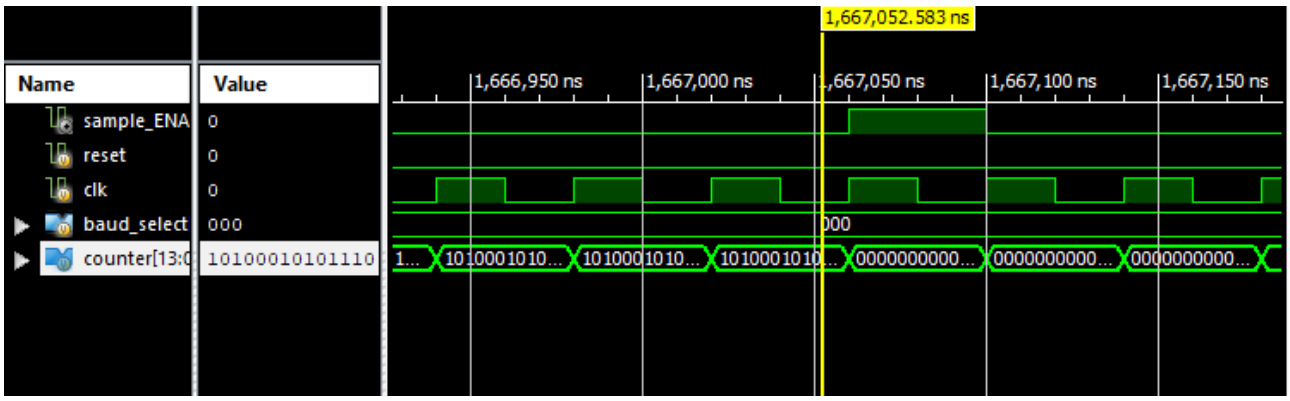
1/(16 x Baud Rate)

baud_select	Baud Rate	Maximum Counter Value	Binary value
3'b000	300bits/sec	10415	10100010101111

3'b001	1200bits/sec	2604,165~=2604	101000101100
3'b010	4800bits/sec	651,04~=651	1010001011
3'b011	9600bits/sec	325,521~=326	101000110
3'b100	19200bits/sec	162,7605~=163	10100011
3'b101	38400bits/sec	81,38~=81	1010001
3'b110	57600bits/sec	54,2535~=54	110110
3'b111	115200bits/sec	27,127~=27	11011

Initially, when reset takes the 1 value, the counter is initialized with 0 value. Then it increases once in every clock period (20nsec), until it reaches its maximum value when it turns the sample_ENABLE signal into 1 for one clock period, and it returns to its initial value.
Finally I created a verilog testbench in order to check the waveforms of my controller.

Simulation results:



PART B – UART TRANSMITTER

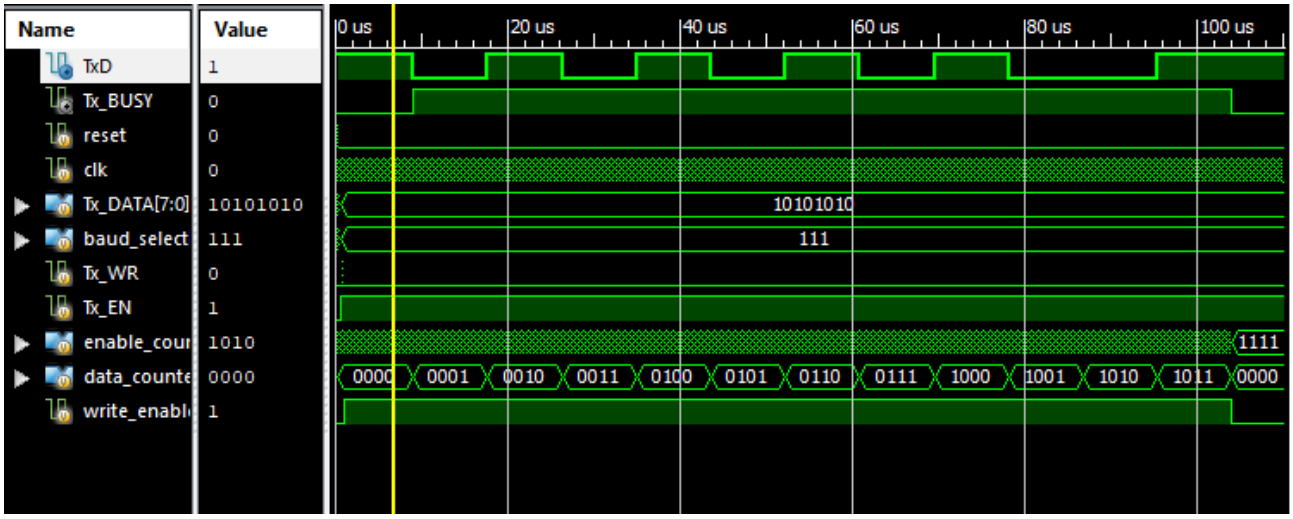
In part B, we created the UART transmitter. The transmitter should function when the signal Tx_EN gets the value '1' and when the Tx_WR gets the value '1' for at least 1 circle (“write_enable” bit stays '1' while we have data to transmit). The data that we want to be transfered are stored in the register Tx_DATA. Every bit of the symbol that is going to be transfered, is to be sent for 16 circles, that the sample enable signal gets the value '1'. For this reason we have created a counter (“enable_counter”), which increases every time the sample enable signal is '1'. When it reaches its maximum value (16), it returns to '0' and this means that the next bit of our data can be now transmitted. The counter “data_counter” declares which bit of the symbol is transmitted,

The TxD register carries the bits that are transferred to the UART receiver. The first bit is called start bit and it is always a '0'. That is because when no symbol is sent, TxD gets the value '1', so by transmitting a '0' as a start bit, the receiver can understand that some data are about to be sent. Then, we can send the 8-bit symbol, the parity bit and th stop bit.

The parity bit is '0' when there is an even number of '1' bits, or '1' when there is an odd number of '1' bits. In particular, it is calculated by adding all of the bits of the 8-bit symbol.
The stop bit is the 11^{th} bit that is transmitted, and it has the value '1' , to make the receiver recognize that the message is over.

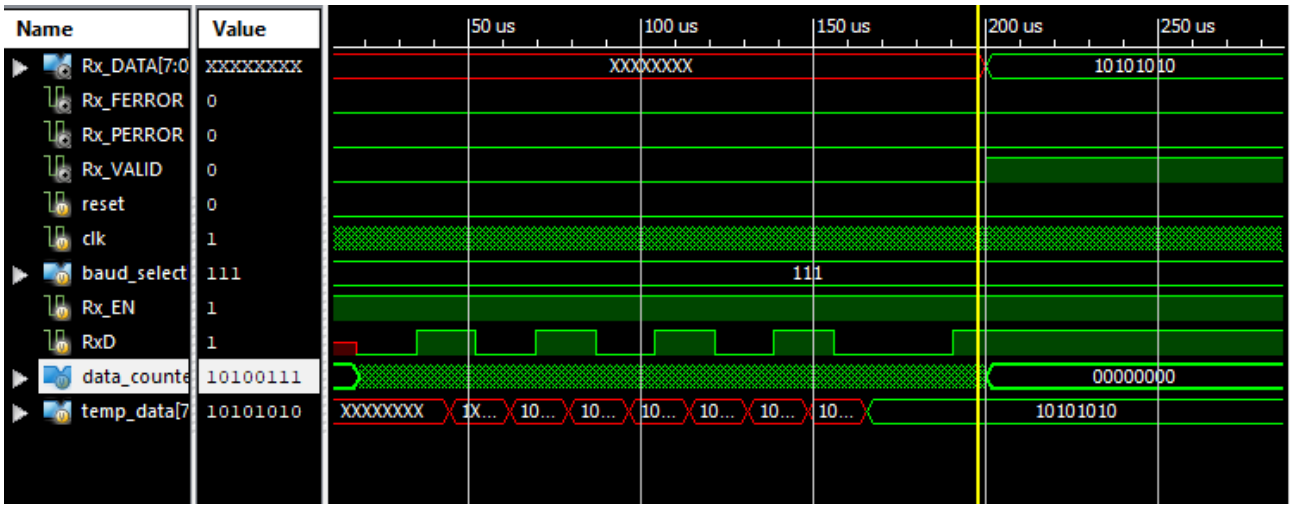
Finally while our data are transmitted, the Tx_BUSY signal gets the value '1', in order to show us that we cannot send a new message.

Simulation Results:

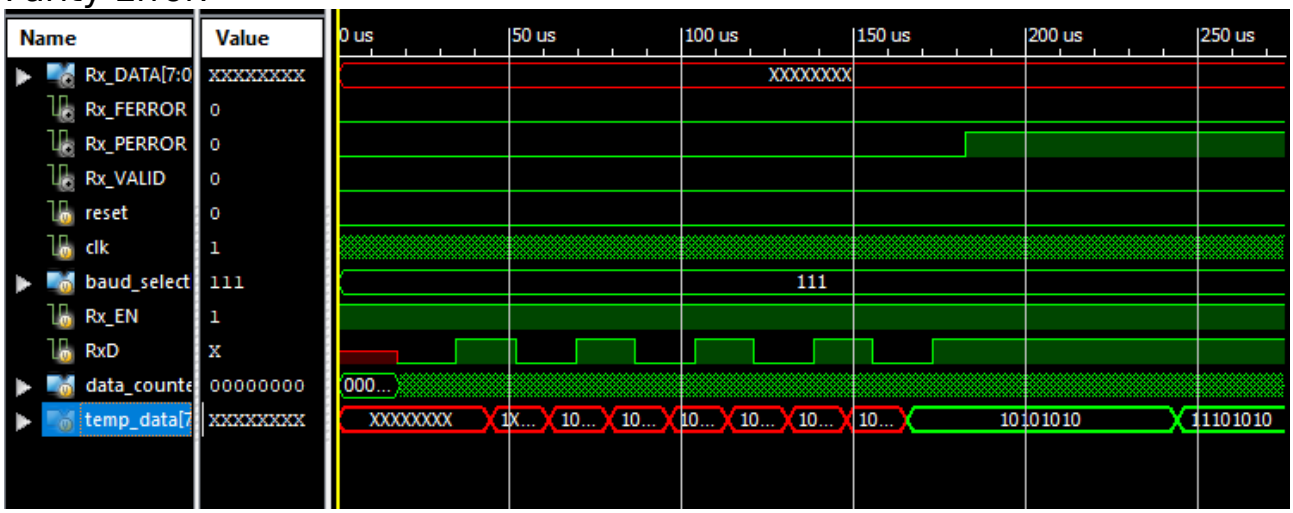


PART C – UART RECEIVER
In a corresponding way we have created the UART Receiver. Firstly, the receiver can sample every 16 circles, in accordance with the sample enable signal, which occurs by the baud controller. In order to get the right data, we sample in the 8^{th} circle of each trasmission, and this is controlled by the “data_counter” counter.
The receiver is able to receive data, only when the Rx_EN signal is '1'. After the transmission of the data, we should chech whether the parity bit that we have received is correct (so we set Rx_PERROR = 0), or not (Rx_PERROR = 1). In the same way we check if the stop bit is '1' (Rx_FERROR = 0), or '0' (Rx_FERROR = 0).
Finally, if we do not have any errors, we can set the Rx_VALID signal as '1'.

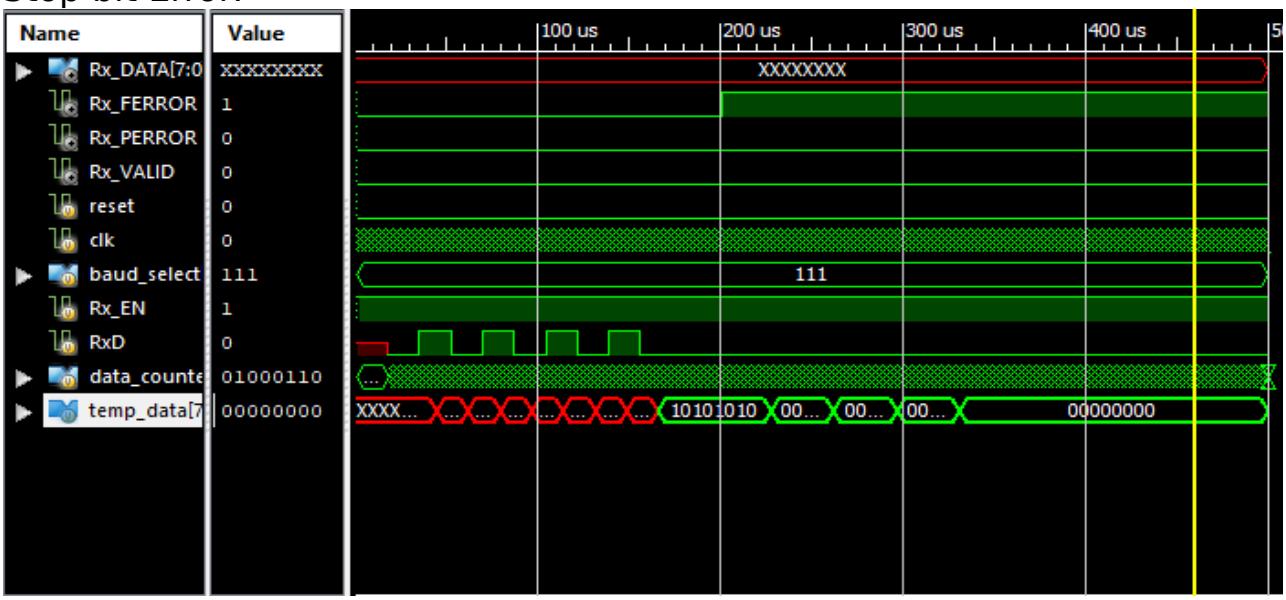
Simulation Results:



Parity Error:



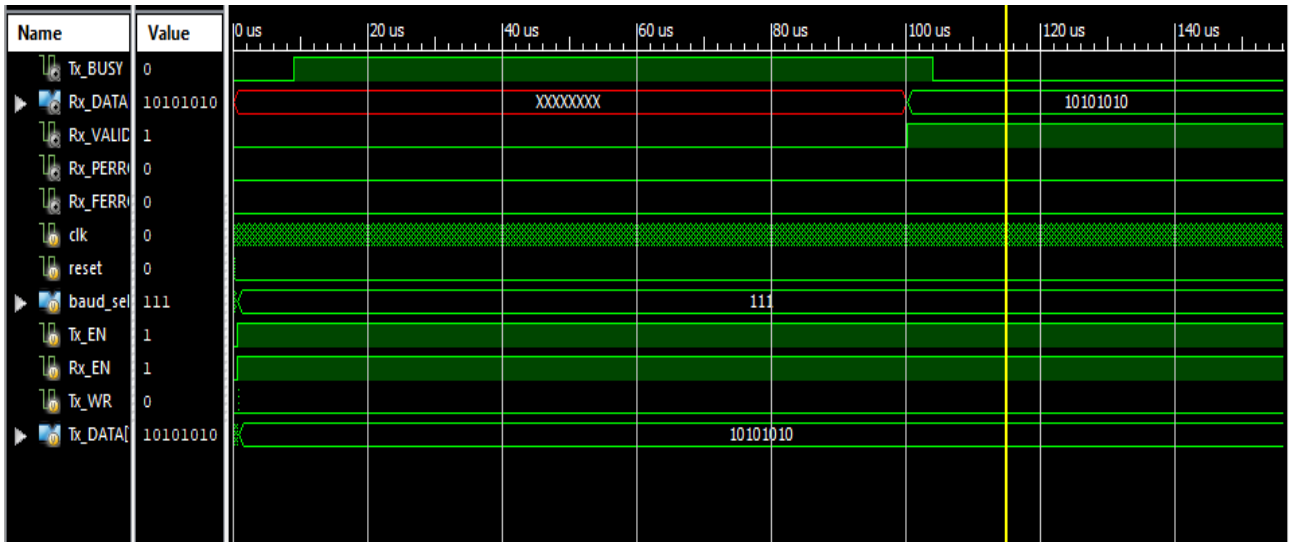
Stop bit Error:



PART D – UART TRANSMITTER - RECEIVER

In the last part, we have to join the two previous parts into one.

Simulation Results:



PART D - Additional

The additional part is to display the hexadecimal value of the symbol that we transmit, in the 7-segments display of Spartan 3 FPGA.
In order to accomplish this, we changed the 7-segments indicate driver of Project 1, so that it can use only the an0 and an1 lights of the screen, we added the symbols that we want to show, and we reduced some of the counters to accommodate this project's needs.

Simulation Results: