Παναγιώτης Κρεμμύδας 1435
17/11/2016

Lab exercize 2:

Universal Asynchronous Receiver Transmitter

CE430: Digital Circuits Lab

Electrical& Computer Engineering

University of Thessaly

Contents

Introduction:

This is the second out of four assignments in the context of the Digital Circuits Lab course.

The purpose of this assignment is to implement the serial data transfer between two distinct systems running different clocks using the Universal Asynchronous Receiver &Transmitter protocol. The system consists of a UART transmitter and a UART reciever with a single serial connection between them. The bitstream transmitted is utilized by the reciever to drive the 7segment display implemented in the previous assignment.

The assignment includes simulation on a behavioral& gate level using the Xilinx ISE and Modelsim tools and generation of the corresponding bitfile for testing on the provided Spartan3 FPGA prototyping board.

Part 1 - Baud Rate controller

Implementation

The purpose of this module is to allign the clock of the machine using it to the specified rate declared by the UART protocol. In order for this to be achieved specified clock counters for each bitrate used by the UART protocol have to be assigned:

Clocks_Baud_Rate_8= systemclockfrequency/115200

Clocks_Baud_Rate_7= systemclockfrequency/57600;

Clocks_Baud_Rate_6= systemclockfrequency/38400;

Clocks_Baud_Rate_5= systemclockfrequency/19200;

Clocks_Baud_Rate_4= systemclockfrequency/9600;

Clocks_Baud_Rate_3= systemclockfrequency/4800;

Clocks_Baud_Rate_2= systemclockfrequency/1200;

The value represented by the above counters is the number of ticks necessary until the next bit should be transmitted. The transmitter uses that exact number but the reciever needs the subdivision of that in order to sample the incoming vectors.

Example: In the case of a 50.000.000hz clock like the one impemented in the SPARTAN 3 board and in the case of a Receiver, the maximum possible error in the bitrate would be in the 115200 bits/second mode and is calculated for that specific frequency to be:

SystemClockFrequency/(Bitrate*SamplesPerBit)= 50.000.000/(115200*16)= 27.1267361111

The floating part of the division is lost, leading to an error in the calculation of the actual rate

Rate achieved in the SPARTAN3 implementation: 115740.740741bits/second

In the case of the SPARTAN3 board the error is (0.1267361111)/(27.1267361111)= 0.00467199999

Error=0.4671999%

The greater the bitrate and the samples per bit, the bigger the possible error depending on the floating point.

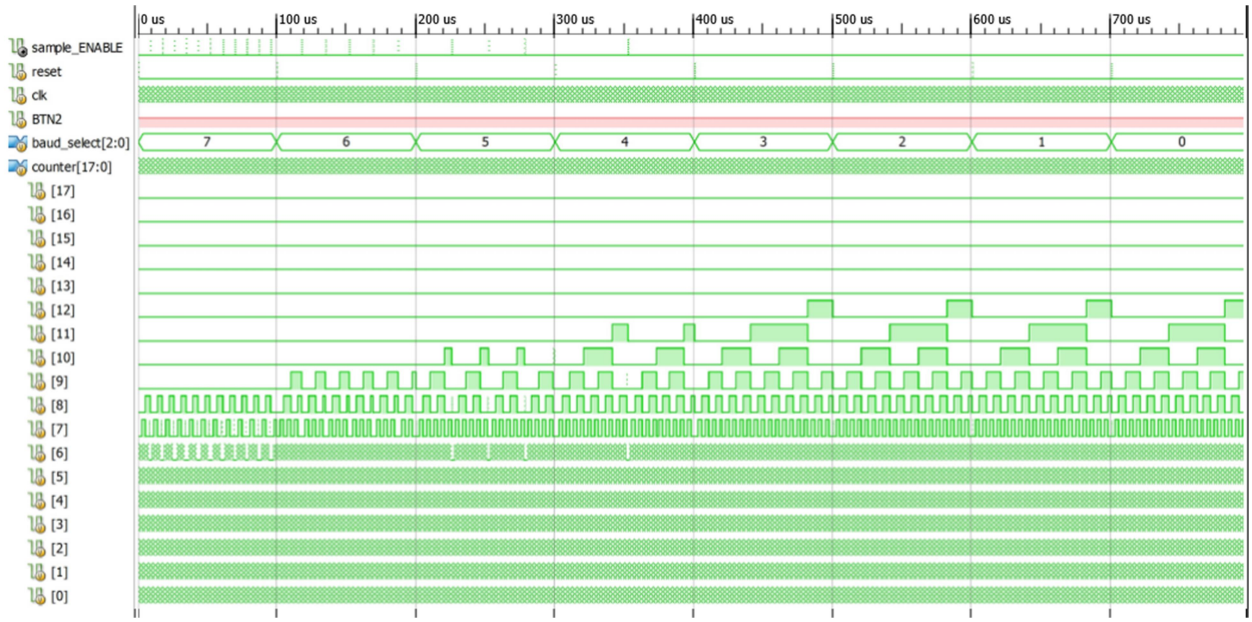
Will always be less than $(0.5/27)$= 0.01851851851 = 1.85% for clock frequencies around that magnitude and a x16 sampling frequency.

In the current implementation the reciever reallignes the sampling frequency in the beginning of every transfer to be in the middle of the start bit.

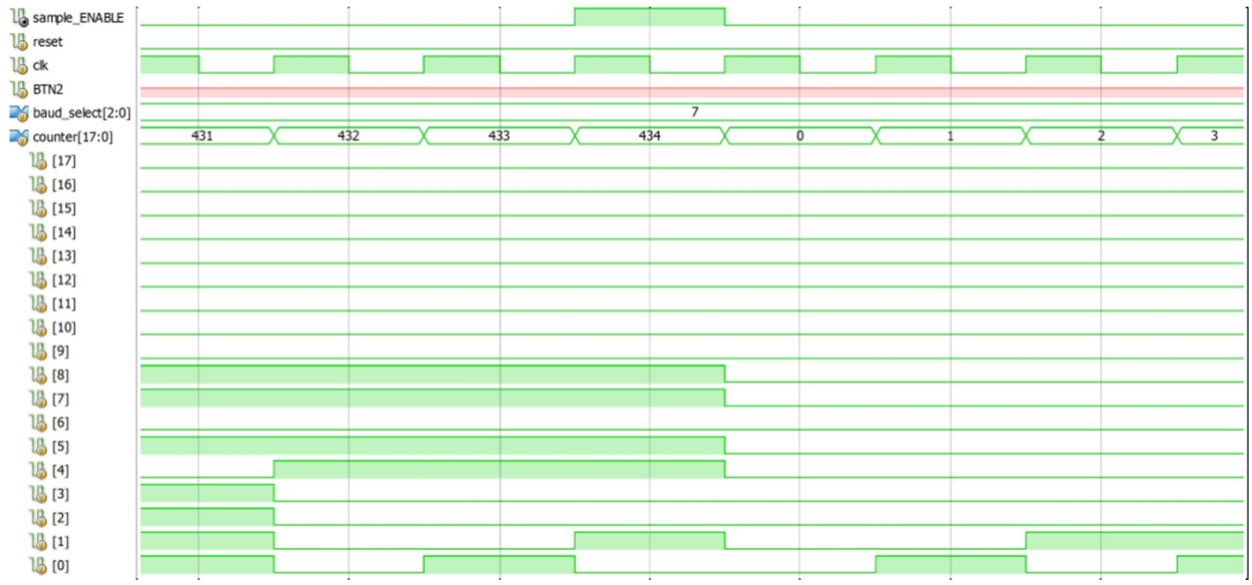
In the worst case scenario that the rounded clock numbers of the reciever and the transmitter are leading to a 4.545% faster and 4.545% slower bitrate each then the bits transmitted will missallign by the end of the transmittion.

(In practice any frequency more than 10Mhz for the transmition of a 11bit vector could give the expected results and perform within error margin)

Verification



Those are the different baud rates that are implemented using a different clock counter limit that gives the desired sample_ENABLE signal to the UART transmitter and receiver modules.



434 clocks for a 50mh system give a bitrate of 115200bits/second. Sample Enable signal is given right on the posedge of the last configured number of clocks counted.

Experiment/Resulting implementation

FPGA board testing was not necessary for this part of the assignment

Part 2 - UART Trasmitter

Implementation

Transmittion of the specified data using the preconfigured baud rate. Offering availability interfacing to the system about undergoing data transfers or being ready to receive next symbols.

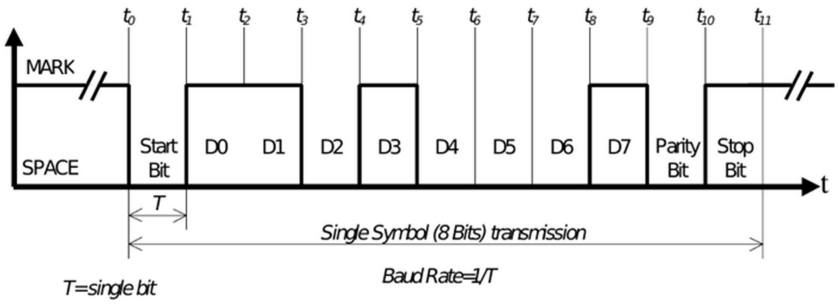
baud_rate_sampler_transmitter . A specified instance of the baud rate is used for the frequency of 50Mhz.

Tx_BUSY . Is high if the module is actively transmitting a bitstream. It controls the bit_slot counting. If the bitstream is finished its low and no longer Busy that's when the bit_slot stays in the 11^{th} state, the bit stop state. If there is another bitsream to be transmitted its possible to be loaded right during the stop bit transmittion and be transmitted right away after it.

myNextBitstreamISready A register is needed to capture the Tx_WR 1cycle signal and hold the information that the next vector is ready for transmittion to the Tx_BUSY variable right when its necessary. That will make it possible for the next bitstream to get ready for transmittion without delay.

bit_slot counter. Control the state of the bitstream that the Transmitter is currently in.

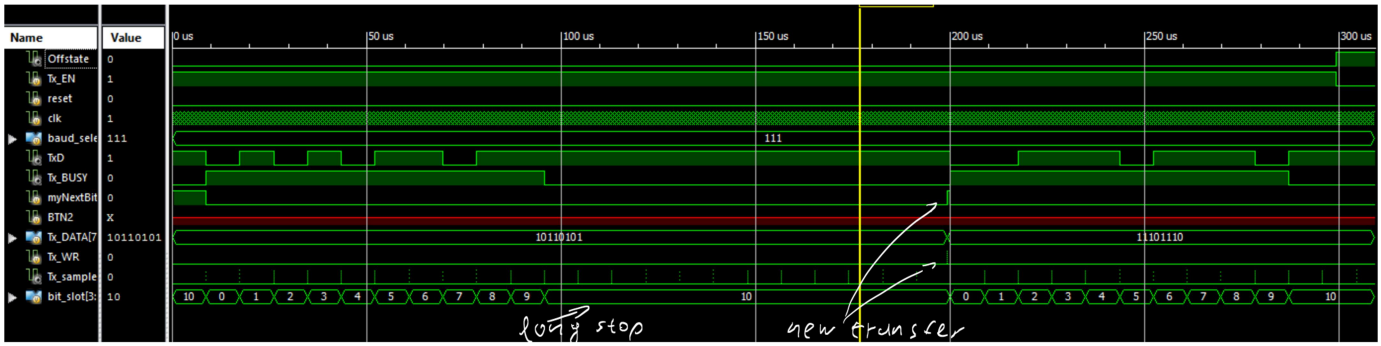
TxD output. Is assigned to the prescpecified values depending on the state.



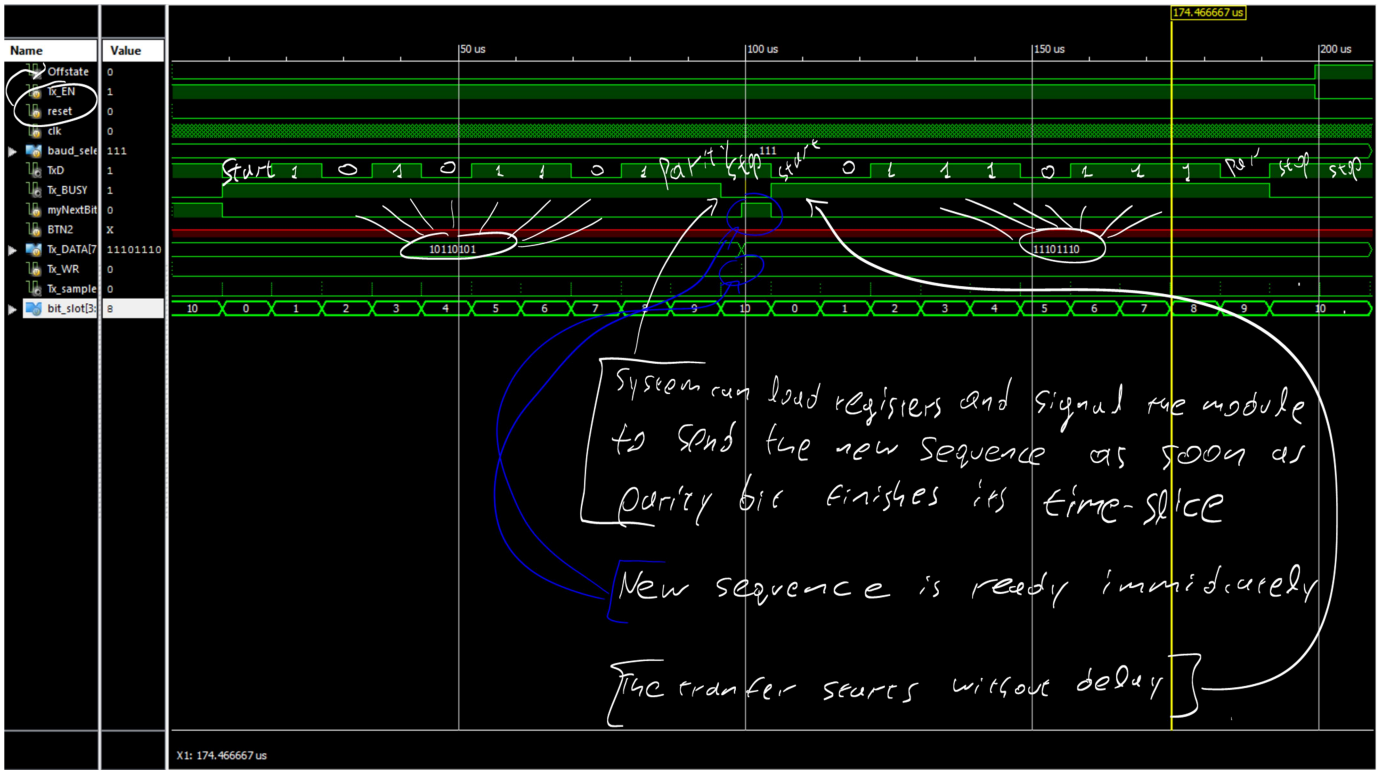
https://github.com/kmd178/Digital_Systems_lab2_UART/tree/f0c13de39b9474ea367813f4a6748acce614e394

Verification

Long stop between vector transmitions.



Vector transmitions without stop:



Experiment/Resulting implementation

FPGA board testing was not necessary for this part of the assignment

Part 3 - UART Reciever:

Reception of the specified data using samples alligned to the middle of the incoming transmition slots. Inform the system in case of transmition errors using frame and parity checking.

baud_rate_sampler_transmitter . A specified instance of the baud rate is used for the frequency of 20Mhz.
transmittionFirstSlot=1'b0; //Flags the state which the samples_counter is currently counting samples in the first Slot

middleSlot_sample : Depending if the transmittion is currently in the first slot or not it will either signal in 8 baud rate samples or 16. Controls bit_slot counter,

bit_slot counter. Is the FSM that declares the states of initialization of the output bistream data or availability of the transmittion to the system and its error messages.

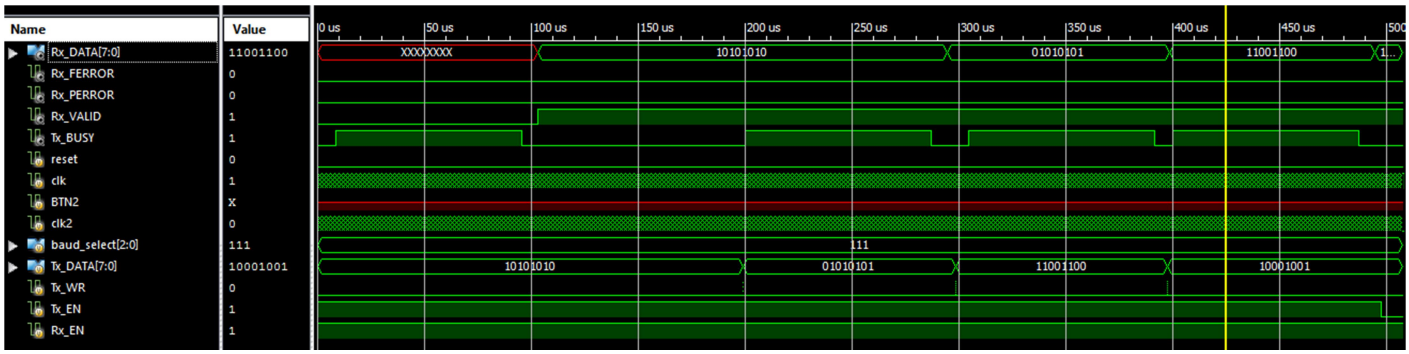
Rx_DATA output. Is assigned the values of the incoming bits corresponding to DATA.

Implementation

https://github.com/kmd178/Digital_Systems_lab2_UART/tree/221aeb1798d346b4635ec6729aa30054b12d10dd

Verification

Rx_DATA is receiving the values assigned by Tx_DATA as expected.



Experiment/Resulting implementation

FPGA board testing was not necessary for this part of the assignment

Part 4 - Full system UART integration:

Implementation

Transmittion of preconfigured messages using the Tx_DATA, Tx_Busy and Rx_Valid interface of the above modules

change_symbol module Signals the change of the Tx_DATA inputed symbol on the negedge of the Tx_BUSY signal.

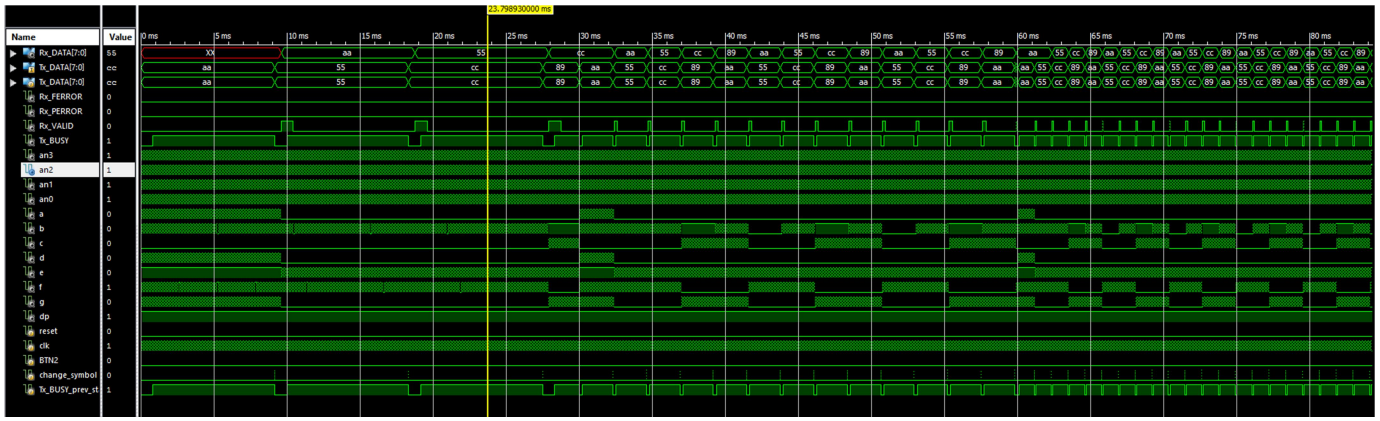
CurrentSymbol: Controls the set of characters assigned to be transmitted

ledDataFeeder: Uses Rx_Data as input and cycles it through the LED display of the FPGA unit

LEDdecoder: Decodes the characters received to the corresponding parts of the LED unit that need to be activated.

anti_bounce_reset: Stabilizes the signals of the reset and input buttons that are used to Reset or Reset and change the baud rate of the transmittion accordingly

Verification



Testbench is signalling the input button for rotation of the message to appear in the waveforms. Reset function is tested as well. The input button and reset are flactuating in short time intervals before set on or off for mechanical button simulation .

For easy waveform crosschecking a short 6bit counter for short intervals is used to control faster if the button pressed and if the stabilized signal should pass as an output.

Output waveforms on segment and anode registers are generally performing as expected. Rx_Valid is later implemented to reset to 0 after a clock period. Its later utilized to enable the change of the LED letters to the expected valid transmittion data.

Experiment/Resulting implementation

FPGA board testing will take place on 23/11/2016

Conclusions:

Is Implementation using one register less in baud_rate improving the design?

Uncertain-Unclear: The delay itself that expresses registers is

implemented with buffers. Implementing sample_Enable as a wire could improve the design in certain situations.

Why cant memory be assigned using always @(*);

(*) Refers to all possible left hand side changes. Memory assignments dont have any because they are constants, and they are never accessed. Initial block should be used instead.

Should baud_select_table be a register?

Yes. Bitfile compiler will automatically put it into blockrams.

sample_ENABLE=&(counter^~18'b000000000110110010)causes excesive skew and should not be used to implement an equallity check signal