UART TRANSMISSION ON THE DE2-115 BOARD

UNIVERSITY OF NEVADA, LAS VEGAS

GERARDO GOMEZ-MARTINEZ

Universal Asynchronous Receiver / Transmitter(UART) is one of the simplest forms of digital communication. UART allows us to send data serially; therefore we only need two wires to send data to our computer, one for the data and a second one for ground. Information sent through UART is sent in packets which usually consist of one start bit, one stop bit, and a byte (8-bits) of data. Other variations, such as 7-bits of data and additional parity bit, are also widely used. The receiver and the transmitter have to know at which rate the data is going to be sent; this rate is known as the BAUD rate. The BAUD rate signifies the number of bits sent per second.

For this project we will be sending data from an FPGA to a computer. We begin the process by writing Verilog code to transmit the data. The idea of the transmitter code is rather simple; the data that is being sent is shifted and assigned to the TxD output to send the data. A Baud rate of 9600 is being used here, which means 9600 bits are sent each second. To achieve this, the clock inside the FPGA is divided as the FPGA has a 50MHz clock, by dividing it we can create the equivalent of a 9600/second clock. We need our shifter to shift each byte 9600 times per second, meaning we need the equivalent of a 9.6kHz signal. The internal clock has a speed of 50MHz, therefore we need to find how many clock ticks we need to count up to before we send the signal, to send each bit, to the transmitter. To compute this we use the formula:

$$\frac{internal\ clock}{x} = signal\ frequency$$

Substituting with the values we get:

$$\frac{50X10^6}{x} = 9600$$

Solving for x we get:

$$x \approx 5208$$

This tells us the number of clock ticks that need to pass before we send each bit. We use a count variable inside our code to count the number of clock ticks, and once this value reaches 5208 we shift our data being sent and assign the rightmost value to the TxD output. The complete code for the transmitter module can be found below.

```
/* Module transmits UART data from an FPGA
    Inputs: clk, the internal clock of the FPGA (50MHz)
                reset, resets the transmitter to it's initial state
                transmit, transmitter only operates if this bit is on
                data, the data to be transmitted
    Output: TxD, the serial output
*/
module transmitter(input clk, reset, transmit,
                         input [7:0] data,
                         output reg TxD);

reg[4:0] bitCounter; // counts the number of bits that have been sent
// counts the number of clock ticks, used to divide the internal clock
reg[31:0] counter;

// the current and next state of the transmitter
reg state, nextState;

// register used to hold the value that is currently being sent
reg[9:0] rightShiftReg;
// determines the operations that should be done in the current state
reg shift, load, clear;


always @ (posedge clk)
begin
    if (reset) begin
        state <= 0;
        counter <= 0;
        bitCounter <= 0;
    end
    else begin
        counter <= counter+1;
        if (counter >= 5208) begin // divides the clock for a Baud rate of 9600
            // Once the value has been reached, sets the next state, resets the
            // counter and performs the operations of the current state
            state <= nextState;
            counter <= 0;
            if (load)
                // Sets the data to be sent including a start bit (0), and a stop bit (1)
                rightShiftReg <= {1'b1, data[7:0], 1'b0};
            if (clear)
                bitCounter <= 0;
            if (shift) begin
                rightShiftReg <= rightShiftReg>>1;
                bitCounter <= bitCounter+1;
            end
        end
    end
```

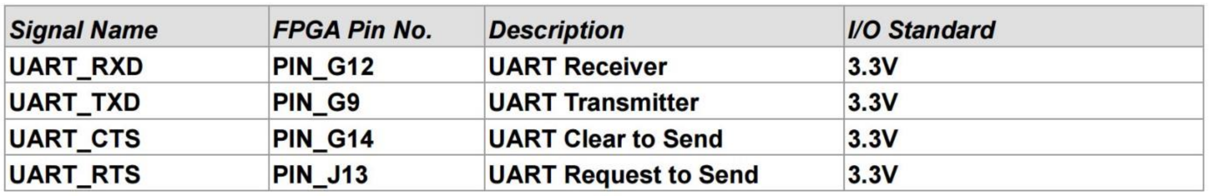
```
end

// state machine for the transmitter
always @(state or bitCounter or transmit)
begin
    load <= 0;
    shift <= 0;
    clear <= 0;
    TxD <= 1;

    case (state)
    // initial state, if transmit is set initializes for data transmission
    0: begin
        if (transmit == 1) begin
            nextState <= 1;
            load <= 1;
            shift <= 0;
            clear <= 0;
        end
        else begin
            nextState <= 0;
            TxD <= 1;
        end
    end
    // sets the operations for this state and stays here until all 10 bits have been sent
    1: begin
        if (bitCounter >= 9) begin
            nextState <= 0;
            clear <= 1;
        end
        else begin
            nextState <= 1;
            shift <= 1;
            TxD <= rightShiftReg[0];
        end
    end
    endcase
end

endmodule
```

After compiling the code, the output of the ports in the module have to be assigned to the correct pins on the FPGA. For the DE2-115, the pin numbers used for the RS232 port are listed on Table 1-1. For this project we are only concerned about the TxD output; therefore, the TxD output from our module has to be assigned to PIN_G9. This is the only assignment needed for this project, as we will not be using the receiver nor the hand-shaking lines. If using a different FPGA, the process will be similar but the pin numbers may differ, so it will be necessary to find the correct ones.



Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
UART_RXD	PIN_G12	UART Receiver	3.3V
UART_TXD	PIN_G9	UART Transmitter	3.3V
UART_CTS	PIN_G14	UART Clear to Send	3.3V
UART_RTS	PIN_J13	UART Request to Send	3.3V

Table 1-1 RS-232 Pin Assignments

If using a RS232 cable, there is no need to worry about how to make the connections; the cable should just be connected from the FPGA to the computer. If using an FTDI chip to convert to USB, the connections would have to be made as follows:

PIN 5 in the FPGA's RS232 header should be wired to the RxD receiver on the FTDI chip.

PIN 2 in the FPGA's RS232 header should be wired to the Ground input of the FTDI chip.

The complete output and port numbers can be seen in Figure 1-1. The port numbers are also marked on the RS232 header.

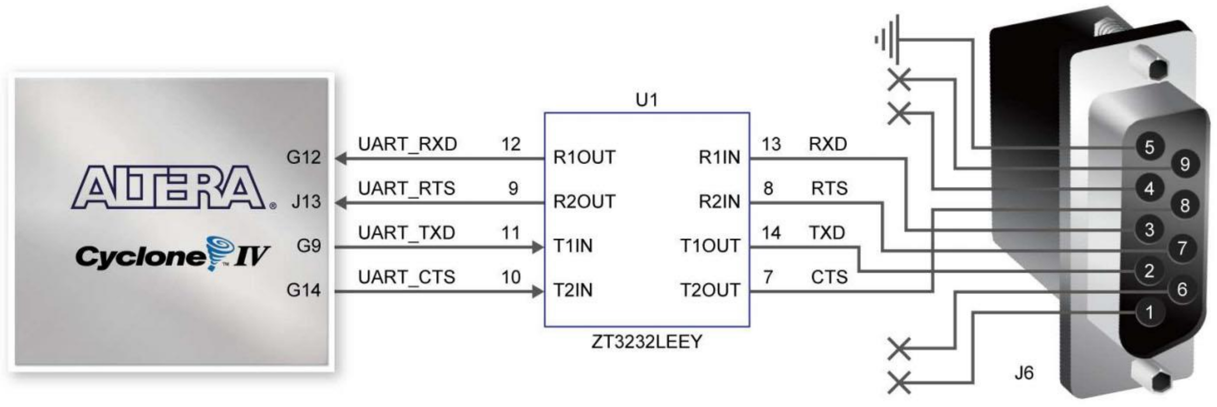


Figure 1-1 Connections between FPGA and RS-232 (ZT3232) chip

Once the connection is made, we need to determine a way to read and interpret the data that is coming from the FPGA. An easy way to read the data coming in is by using a terminal program such as *Tera Term*. These types of programs usually work great for most applications; the only problem is that they usually expect to receive characters, so the data being sent would be interpreted as an ASCII character. Another option for greater flexibility is to write a computer program that can read data coming in from a COM port, in this case the UART data. A simple program that can read data from a COM port, interpret it as an integer, and print it to the screen can be found bellow. The code is written in Visual C++, which provides functions to easily read from the COM port. A precompiled version of the Terminal can also be found here http://cmosedu.com/jbaker/students/gerardo/projects/ByteTerminal.exe.

```
#include "stdafx.h"
#include <iostream>
#include <atlstr.h>
#include <Windows.h>

// Function reads one byte of data from the specified COM port
int readByte(HANDLE);
int main(int argc, char* argv[])
{
    int comPortNum; // The COM port number
    CString comPort; // String containing the COM port info
    int readData; // Data read from the COM port
    DCB dcb; // Settings for the UART communication

    // Asks the user for the COM port data
    std::cout << "Enter COM port number: ";
    std::cin >> comPortNum;
    comPort.Format(_T("COM%d"), comPortNum);

    // Attempts to open the COM port
    HANDLE hPort = CreateFile(
        comPort,
        GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
        );

    // If the COM port cannot be opened it asks the user to enter a valid COM port and
    // continues asking until a valid port has been entered
    while (!GetCommState(hPort, &dcb))
    {
        CloseHandle(hPort);
        std::cout << "Unable to open COM port" << std::endl
            << "Enter a valid COM port number: ";
        std::cin >> comPortNum;

        comPort.Format(_T("COM%d"), comPortNum);

        hPort = CreateFile(
            comPort,
            GENERIC_READ,
            0,
            NULL,
            OPEN_EXISTING,
            0,
            NULL
            );

    }

    // Sets settings for UART
    dcb.BaudRate = CBR_9600; //9600 Baud
    dcb.ByteSize = 8; //8 data bits
    dcb.Parity = NOPARITY; //no parity
    dcb.StopBits = ONESTOPBIT; //1 stop

    // Attempts to apply the settings and if unsuccessful it exits with a code of 0x100
    if (!SetCommState(hPort, &dcb))
    {
        std::cout << "There was an error" << std::endl;
        return 0x100;
    }

    // Loops forever reading data from the COM port and printing it out
    while (1)
    {
        readData = readByte(hPort);
        std::cout << readData;
    }

    return 0;
}

int readByte(HANDLE hPort)
```

```
{
    BYTE Byte;
    DWORD dwBytesTransferred;
    DWORD dwCommModemStatus;
    int data;

    SetCommMask(hPort, EV_RXCHAR | EV_ERR); //receive character event
    WaitCommEvent(hPort, &dwCommModemStatus, 0); //wait for character

    if (dwCommModemStatus &  EV_RXCHAR)
        ReadFile(hPort, &Byte, 1, &dwBytesTransferred, 0); //read 1
    else if (dwCommModemStatus & EV_ERR)
        std::cout << "There was an error\n";

    data = Byte;

    return data;
}
```