Jacobs University Bremen

CA-ECE-803: Digital design

Title: UART Transmitter

Lab report: 4

Experiment conducted by:
Betelhem Nebebe
Maria Namachanja
Neeha Hammad

Instructor: Prof. Dr. Fangning Hu
Execution date: May 6, 2021

1. Introduction

In this lab we learnt how to:

- Design starting from a specification and block diagram
- Properly clock something at a fraction of the clock speed (tx_bit)
- Code a state machine, counter, and multiplexer in VHDL
- Instance one entity (tx) inside another (tx_test)
- Debug using simulation

2. Setup

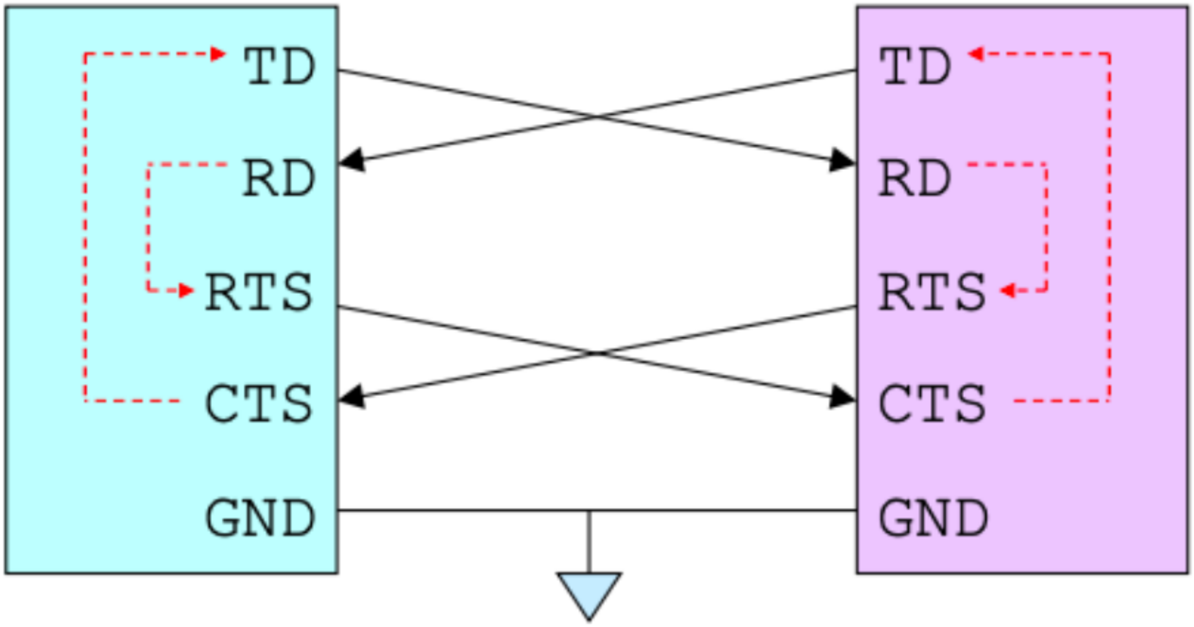
Key Words:

○ UART: Universal Asynchronous Receiver/Transmitter
○ TX: Transmitter
○ RX: Receiver
○ TD: Transmit Data (Data output)
○ RD: Receive Data (Data input)
○ RTS: Request to Send (Flow control output)
○ CTS: Clear to Send (Flow control input)
○ Half Duplex: Transmits one direction at a time (now obsolete)
○ Full Duplex – transmits both directions at the same time
○ RTS/CTS Flow control for full duplex serial ports :
  ■ RX assert RTS when it is ready to receive
  ■ TX waits for CTS before transmitting

Serial Port Devices:

○ Mouse: One of the most commonly used devices for serial ports
○ Modem: Used commonly with older computers
○ Network: Medium of communication between 2 computers.
○ Printer: Mostly used with older printers only.
○ ASCII Terminal (TTY): Like the Hyperterm interface.

Combining Two Ports (Full Duplex):



Serial Communication Format:

○ Data is transmitted sequentially, one bit at a time.
○ To inform the receiver that a new byte is arriving, a "start bit" (a zero) is sent first. A start bit can start at any time.
○ Then the data is transmitted, LSB (least significant bit) first, and MSB (most significant bit) last.
○ At the end, one or two "stop bits" (ones) are transmitted.
○ A frame consist of :

- 1 start bit (a zero)
- 7 or 8 data bits LSB (least significant bit) first
- 1 optional parity bit
- 1 or 2 stop bits (ones)

○ Between transmissions, the transmitter transmits a high.
○ The bit time is determined by the baud rate which is given in units of BPS (bits per second).
○ Transmitter and receiver do not share a clock (hence the asynchronous nature).

Serial Frame:

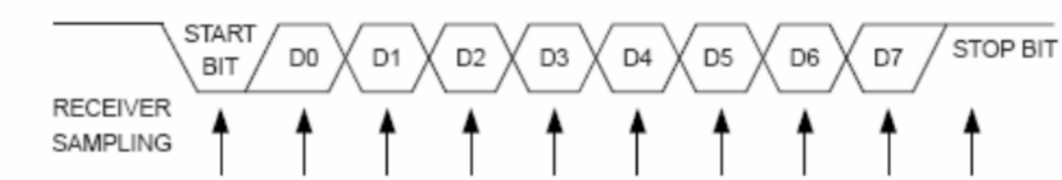


Figure 1. UART Communication Frame Format

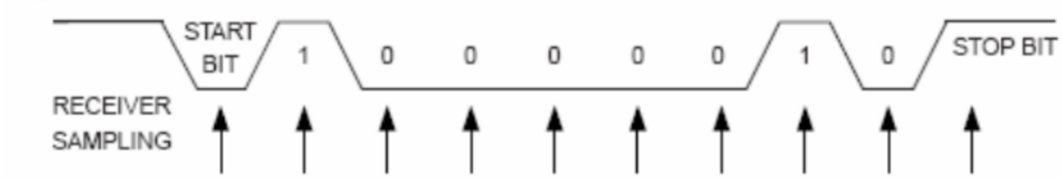
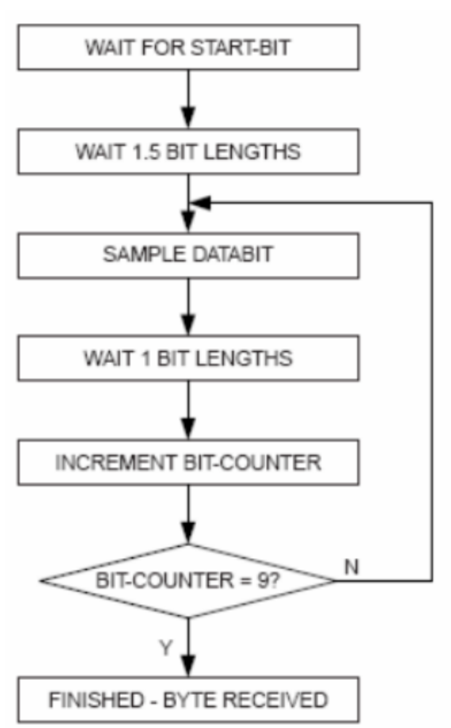
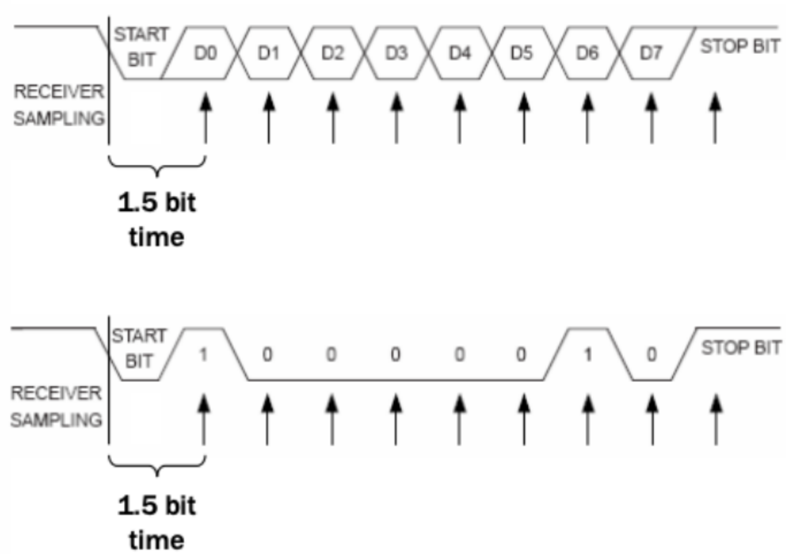


Figure 2. Serial frame of ASCII "A" ($41)

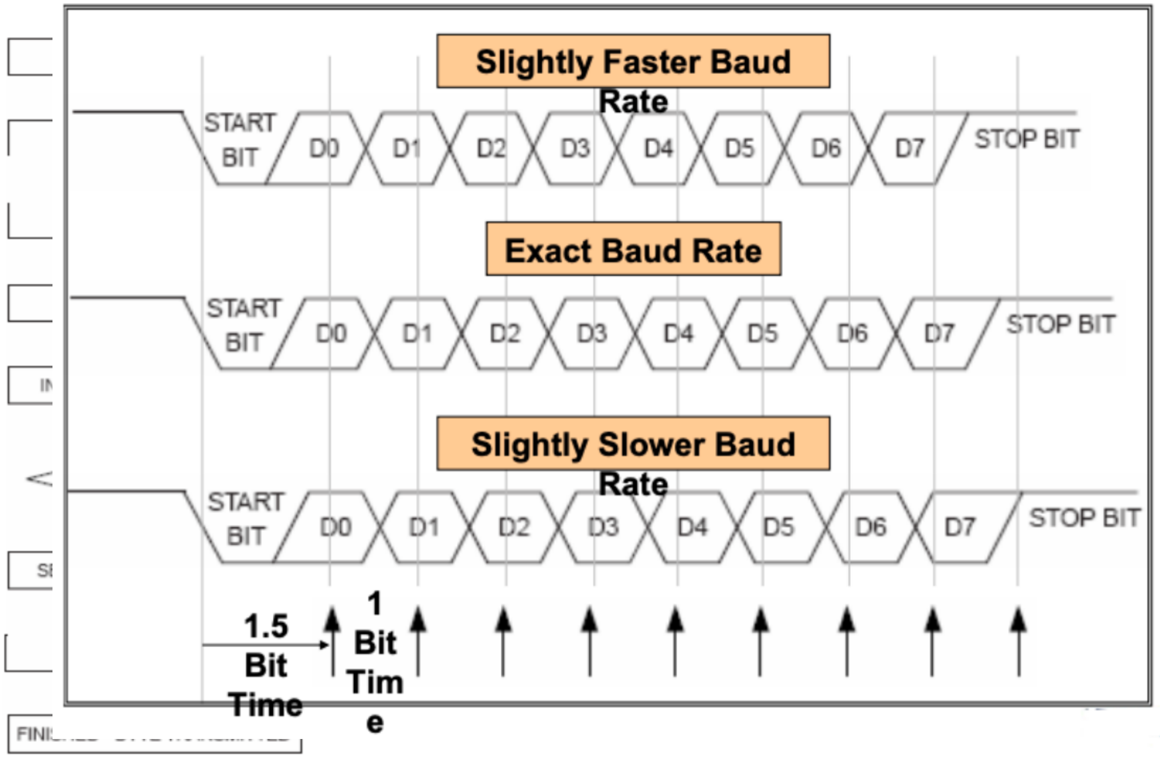
Transmitting:

Receiving



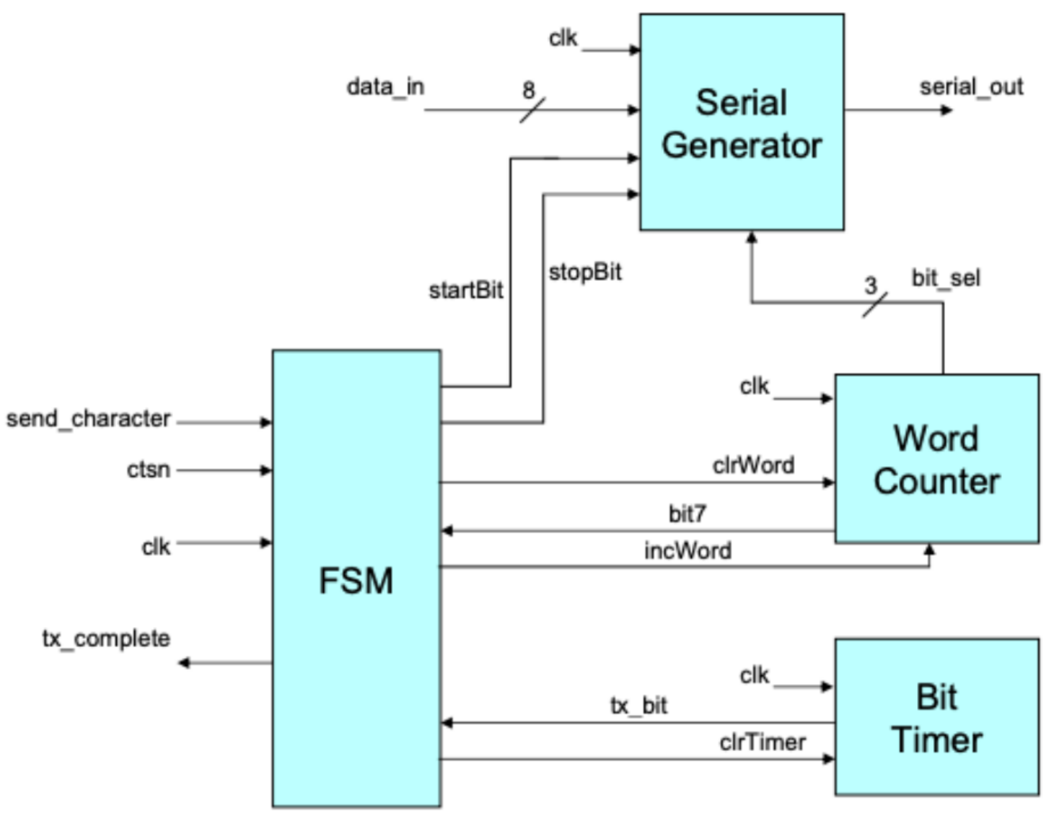


Receiver Sampling Time

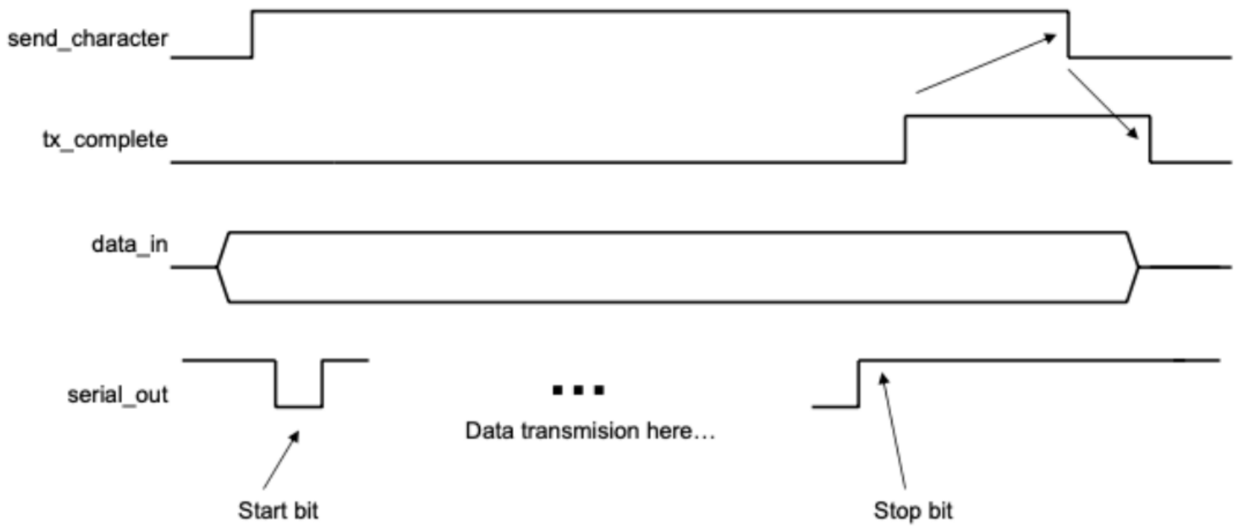


UART Receiver:

Transmitter Block Diagram:



FSM Transmitter:



3. Procedure

Create a VHDL source file called tx.vhd. The inputs and outputs for the tx entity are given in the table below:

INPUTS	Bits	OUTPUTS	Bits
clk	1	serial_out	1
data_in	8	tx_complete	1
send_character	1		
rst	1		

- tx Inputs:
  - clk is the system 50 MHz clock
  - data_in is the byte that you want to send
  - send_character is a control signal that is high when the byte is ready to be sent
  - rst is the reset generated by the clock/reset generator that pulses high on power on.
- tx Outputs:

■ serial_out is the serial data to be transmitted. (Connected to txd of the Serial Port interface).
■ tx_complete goes high to signal that the byte has been sent. This should cause send_character to be de-asserted externally.

● Coding the transmitter:
  ○ Using the lab notes for design references, we coded the transmitter.
  ○ Since the transmitter is placed inside a bigger design, debouncing and interference are resolved.
  ○ We used the Multi-Level Coding Style for the Finite State Machine (FSM).
  ○ Hand-Shaking as per the diagram above.

● Compilation and simulation of the design to show that it will transmit at least two different Bytes:
  ○ Initially we had Bit Timer output a tx_bit every 4 cycles for speed but before output we used 19,200 baud for the number of cycles.
  ○ Hand-Shaking was verified using the diagram.
  ○ Our simulation correctly matched the Professor's demonstration.
  ○ We saved our waveform file.

tx_testbench.vhd file:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tx_test is
   port(
     clk,rst,button: in std_logic;
     switches:in std_logic_vector(7 downto 0);
     txd,done:out std_logic
   );
end entity;

architecture tx_test of tx_test is
   component tx is
     port(
           clk,send_character,rst:in std_logic;
```

```
            data_in:in std_logic_vector(7 downto 0);
            serial_out,tx_complete:out std_logic
     );
   end component;
   signal send_character,debounce1,debounce2,timer1,timer2:std_logic;
   signal timer:unsigned(19 downto 0);
begin
   tx_component:tx port map(
     clk=>clk,send_character=>send_character,rst=>rst,
     data_in=>switches,
     serial_out=>txd,tx_complete=>done
   );

   debounce:process(clk,button)
   begin
     if clk'event and clk='1' then
            if rst='1' then
                    debounce1<='0';
                    debounce2<='0';
                    timer1<='0';
                    timer2<='0';
                    timer<=(others=>'0');
            else
                    debounce1<=button;
                    if timer1='1' and timer2='0' then
                           debounce2<=debounce1;
                    end if;
                     if (debounce1 ='1' and debounce2='0') or (debounce1='0' and
    debounce2='1') then
                            timer<=timer+1;
                    else
                            timer<=(others=>'0');
                    end if;
                    timer1<=timer(13);--1ms
                    timer2<=timer1;
            end if;
     end if;
```

```
      send_character<= debounce2;
   end process;
end tx_test;
```

This file is the "top-level" file that contains all of the I/O and interface logic for connecting the tx.vhd file to the pins on the FPGA. Step 6 and 7 describe the specific functionality of this top-level file. The inputs and outputs for the tx_test entity are given in the table below:

INPUTS	Bits	OUTPUTS	Bits
clk_in	1	txd	1
switch	8	done	1
button	1		
reset	1		

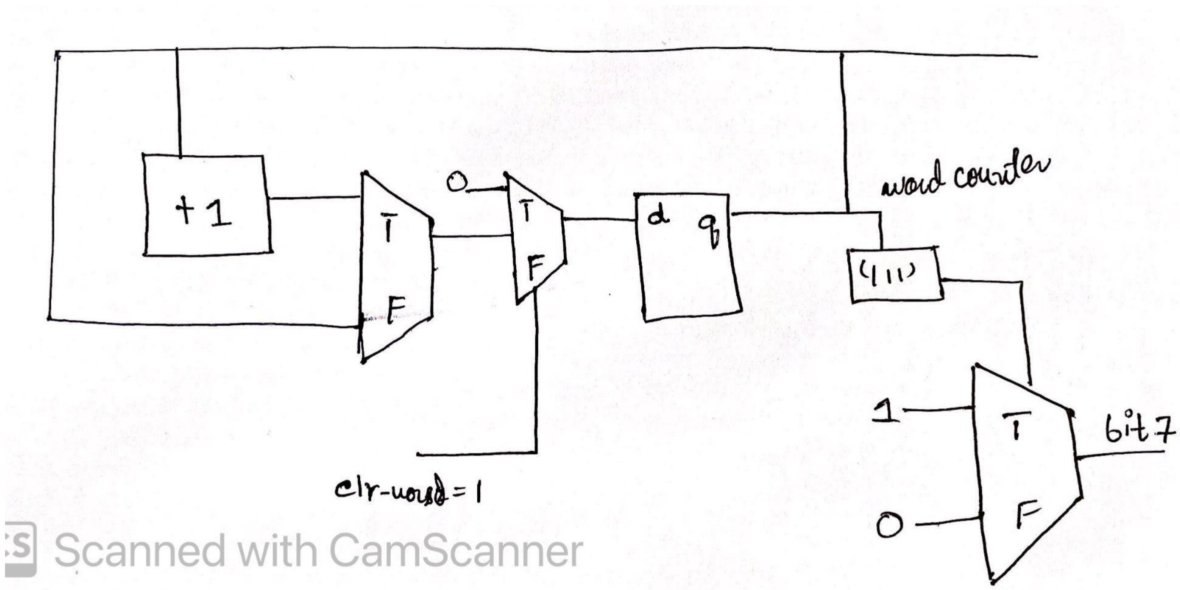
- tx_test Inputs:
  - clk_in is the system 50 MHz clock (tied to system clock input)
  - switch is the byte that you want to send (tied to slide switches)
  - button is the button that is pressed to send a byte (tied to the rightmost push button).
  - reset is tied to a button that is used for resetting your UART. Connect this to the reset input of your transmitter.
- tx_test Outputs:
  - txd is the serial transmit line of the serial port interface
  - done is the tx_complete signal from tx

Feel free to add other debug aids to your design. Suggestions: wrap your 7-segment design to use as a debug display circuit, use LEDs.

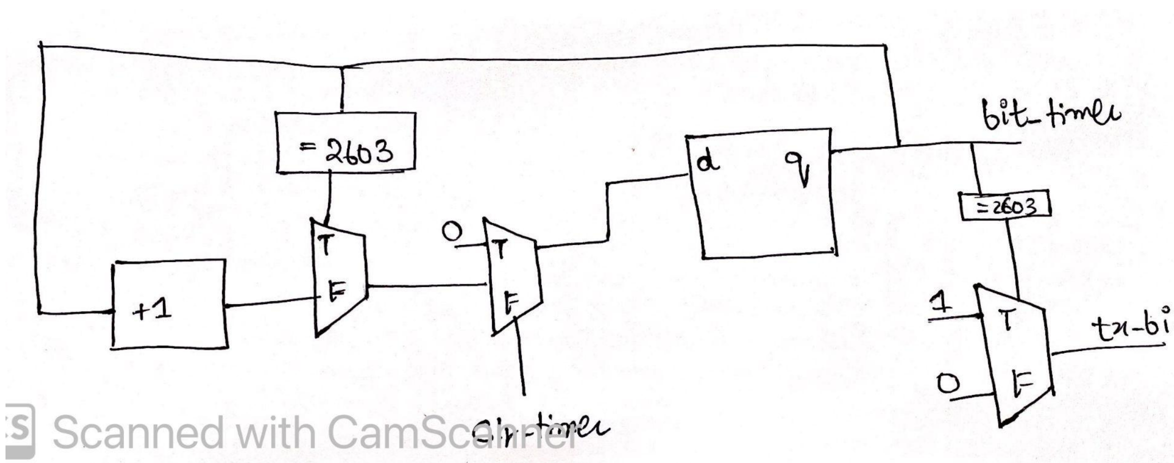
- Create a UCF file for your design. **Done in Lab.**
- Synthesize and download your design to the board. **Done in Lab**
- Connect a serial cable from the lab board to the computer. Run ttpro313 on the Computer. **Checked by the TAs.**

4. Execution

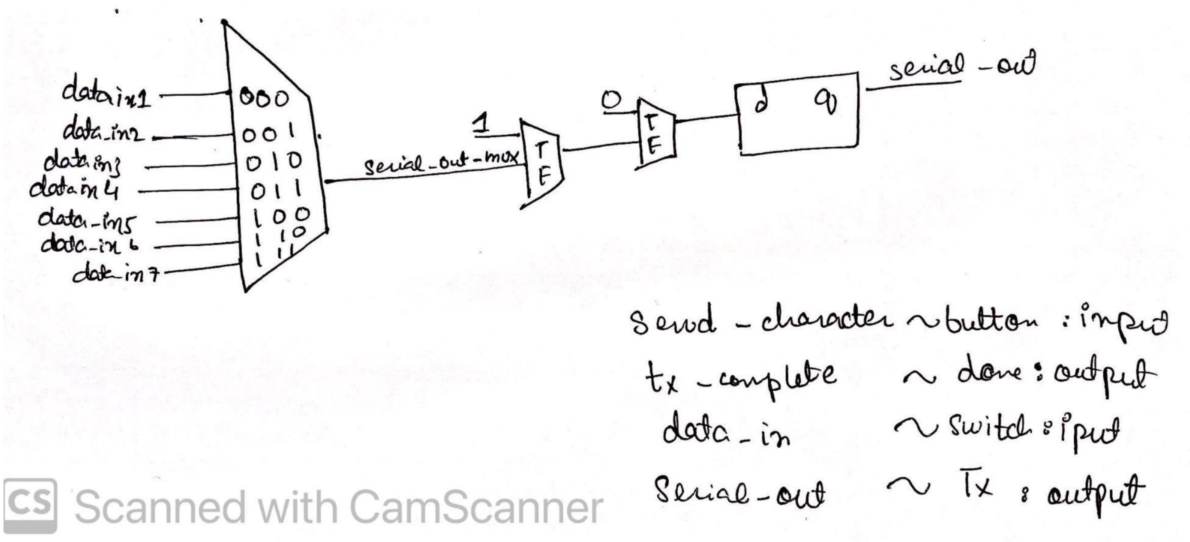
Word Counter:



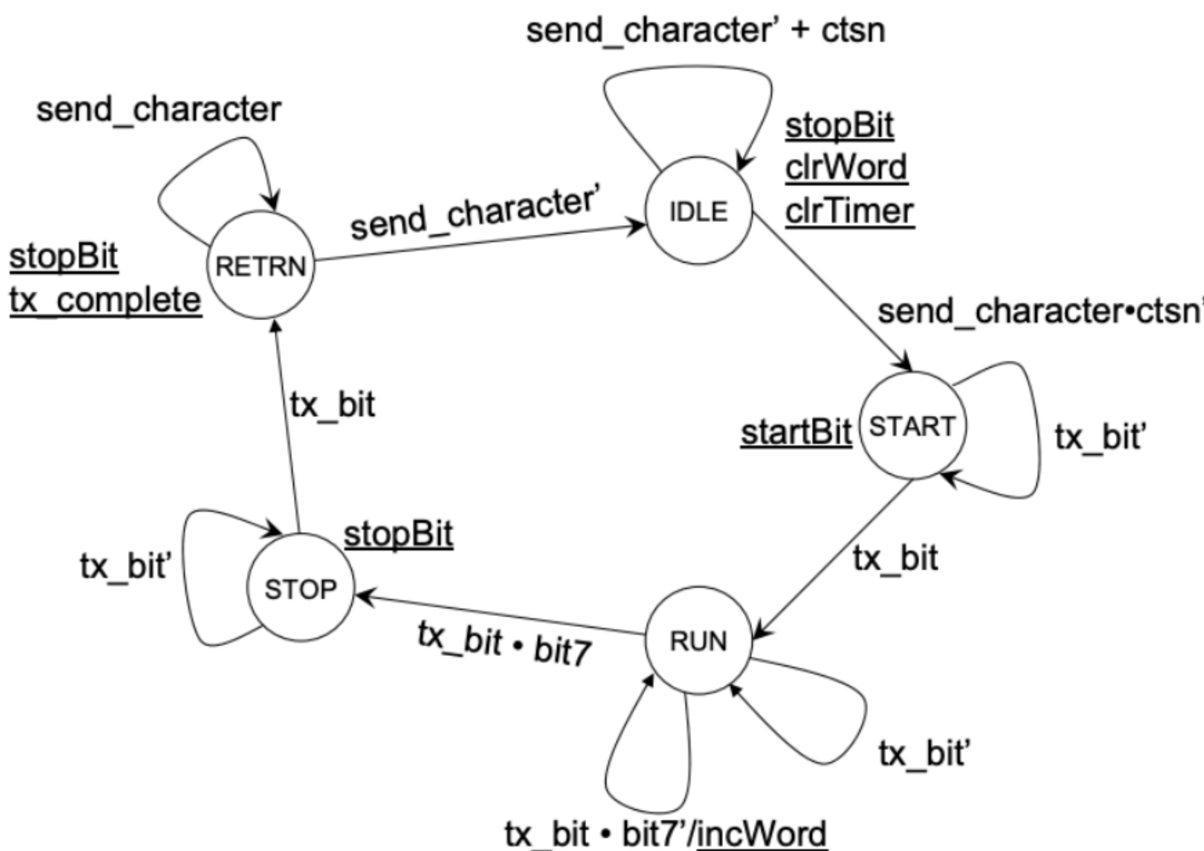
Bit Timer:



Serial Generator:



State Diagram of Finite State Machine



VHDL Code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY tx IS
	PORT (
		rst, clk : IN std_logic;
		data_in : IN std_logic_vector(7 DOWNTO 0);
		send_character : IN std_logic;
		serial_out, tx_complete : OUT std_logic
	);
END tx;
ARCHITECTURE uart OF tx IS
	--Signals for bit-timer, word-timer, FSM, and serial-decoder
	TYPE states IS (idle, start, run, stop, rtn);
	SIGNAL state_reg, state_next : states;
	SIGNAL bit_timer_reg, bit_timer_next : unsigned(15 DOWNTO 0);
	SIGNAL bit7 : std_logic;
	SIGNAL start_bit, stop_bit : std_logic;
	SIGNAL serial_out_next, serial_out_mux : std_logic;
	SIGNAL clr_timer, tx_bit : std_logic;
	SIGNAL clr_word, inc_word : std_logic;
	SIGNAL word_count, word_next, bit_sel : unsigned (2 DOWNTO 0);
BEGIN
	-- Bit timer counter
	bit_timer : PROCESS (clk)
	BEGIN
		IF clk'EVENT AND clk = '1' THEN
			bit_timer_reg <= bit_timer_next;
		END IF;
	END PROCESS;
	bit_timer_next <= (OTHERS => '0') WHEN (clr_timer = '1') ELSE
			(OTHERS => '0') WHEN (bit_timer_reg = 2604) ELSE
			bit_timer_reg + 1;
	tx_bit <= '1' WHEN bit_timer_reg = 2604 ELSE '0';
	-- Word counter
	word_counter : PROCESS (clk)
```

```
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        word_count <= word_next;
    END IF;
END PROCESS;
word_next <=
    (OTHERS => '0') WHEN (clr_word = '1') ELSE
    (word_count + 1) WHEN (inc_word = '1') ELSE
    word_count;
    bit7 <= '1' WHEN word_count = "111" ELSE '0';
    bit_sel <= word_count;
    -- serial generator
    PROCESS (clk)
    BEGIN
        IF clk'EVENT AND clk = '1' THEN
            serial_out <= serial_out_next;
        END IF;
    END PROCESS;
    serial_out_next <=
        '1' WHEN (stop_bit = '1') ELSE
        '0' WHEN (start_bit = '1') ELSE
        serial_out_mux;
        WITH bit_sel SELECT
        serial_out_mux <=
            data_in(0) WHEN "000",
            data_in(1) WHEN "001",
            data_in(2) WHEN "010",
            data_in(3) WHEN "011",
            data_in(4) WHEN "100",
            data_in(5) WHEN "101",
            data_in(6) WHEN "110",
            data_in(7) WHEN "111",
            '0' WHEN OTHERS;
            -- FSM
            fsm_state_reg :
            PROCESS (clk, rst)
        BEGIN
            IF rst = '1' THEN
```

```
            state_reg <= idle;
        ELSIF clk'EVENT AND clk = '1' THEN
            state_reg <= state_next;
        END IF;
    END PROCESS;
    -- Next state and output logic depending on past and present state and
inputs (Mealy and
    Moore)
    fsm_logic : PROCESS (state_reg, send_character, tx_bit, bit7)
    BEGIN
        state_next <= state_reg;
        stop_bit <= '0';
        start_bit <= '0';
        clr_word <= '0';
        clr_timer <= '0';
        inc_word <= '0';
        tx_complete <= '0';
        CASE state_reg IS
            WHEN idle =>
                IF send_character = '1' THEN
                    state_next <= start;
                END IF;
                stop_bit <= '1';
                clr_word <= '1';
                clr_timer <= '1';
            WHEN start =>
                IF tx_bit = '1' THEN
                    state_next <= run;
                END IF;
                start_bit <= '1';
            WHEN run =>
                IF tx_bit = '1' THEN
                    IF bit7 = '1' THEN
                        state_next <= stop;
                    ELSE
                        inc_word <= '1';
                    END IF;
                END IF;
```

```
WHEN stop =>
          IF tx_bit = '1' thenstate_next <= rtn;
           END IF;
           stop_bit <= '1';
 WHEN rtn =>
           IF send_character = '0' THEN
                               state_next <= idle;
                     END IF;
                     stop_bit <= '1';
                     tx_complete <= '1';
END CASE;
END PROCESS;
END uart;
```

UCF File:

The following tables were used to construct the UCF file.

Table 4-1: **Slider Switch Connections**

Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12

Table 4-3: **LED Connections to the Spartan-3 FPGA**

LED	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0
FPGA Pin	P11	P12	N12	P13	N14	L12	P14	K12

Table 4-2: **Push Button Switch Connections**

Push Button	BTN3 (User Reset)	BTN2	BTN1	BTN0
FPGA Pin	L14	L13	M14	M13

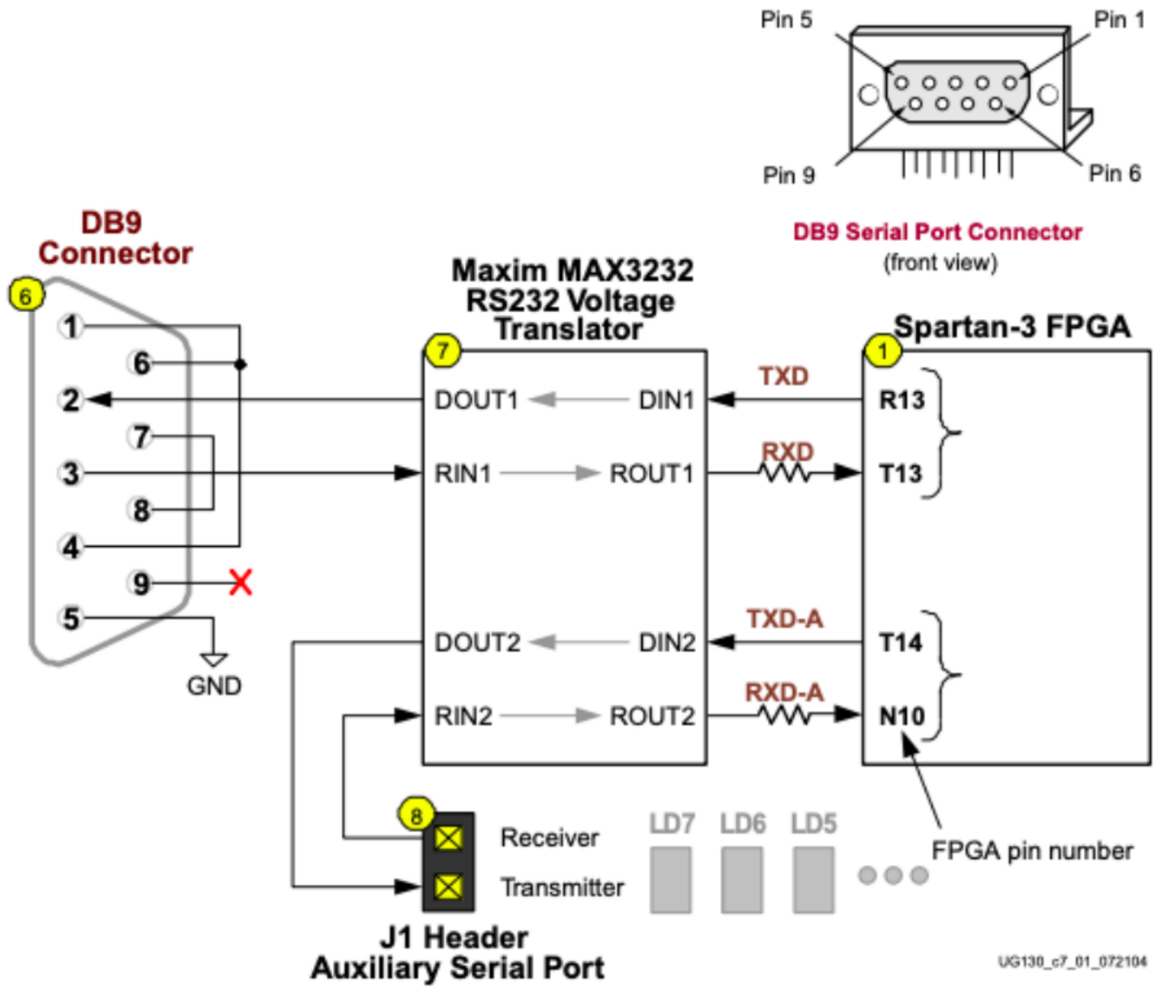


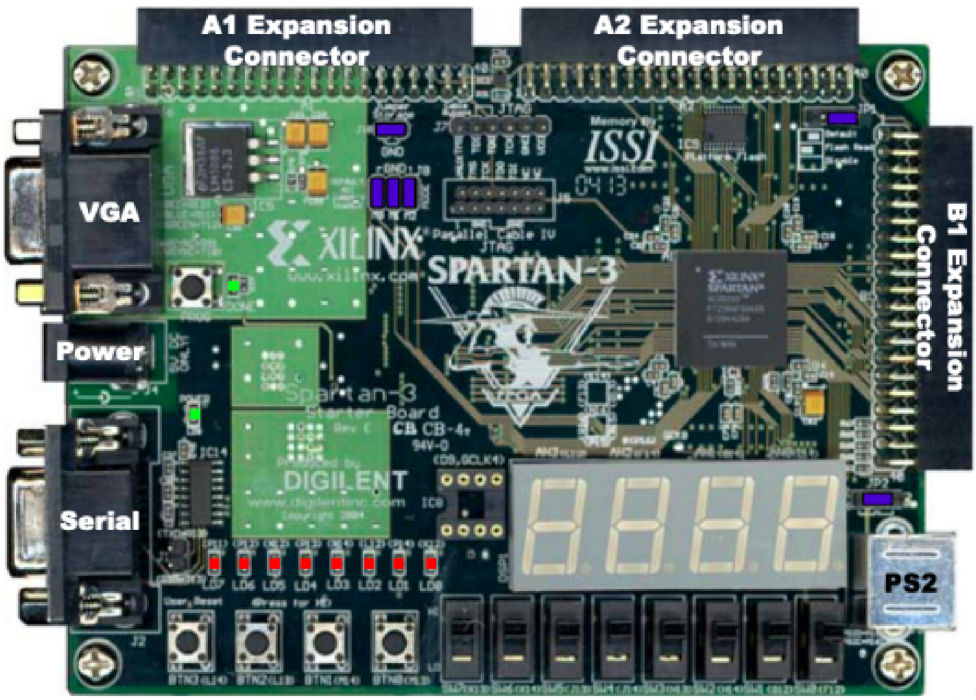
Figure 7-1: **RS-232 Serial Port**

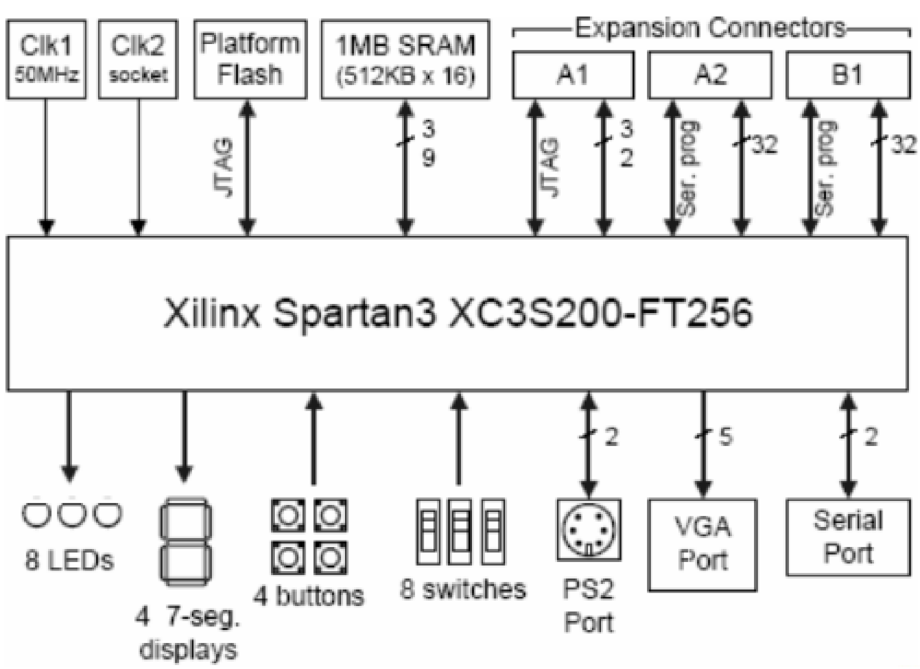
The Spartan®-3 FPGA Starter Kit board has an RS-232 serial port. The RS-232 transmit and receive signals appear on the female DB9 connector, labeled J2.

```
# Spartan - 0 switches 4 to 7
NET "switches<0>" LOC="F12" | IOSTANDARD=LVTTL;
NET "switches<1>" LOC="G12" | IOSTANDARD=LVTTL;
NET "switches<2>" LOC="H14" | IOSTANDARD=LVTTL;
NET "switches<3>" LOC="H13" | IOSTANDARD=LVTTL;
NET "switches<4>" LOC="J14" | IOSTANDARD=LVTTL;
NET "switches<5>" LOC="J13" | IOSTANDARD=LVTTL;
NET "switches<6>" LOC="K14" | IOSTANDARD=LVTTL;
NET "switches<7>" LOC="K13" | IOSTANDARD=LVTTL;
# Spartan - 3 Button 0
```

NET "button" LOC="M13" | IOSTANDARD=LVTTL;
Spartan - 3 LEDs 0
NET "led" LOC="K12" | IOSTANDARD=LVCMOS33 ;
Spartan - 3 Transmit Data (Pre Lab)
NET "txd" LOC="R13" | IOSTANDARD=LVCMOS33;

Circuit Design:





5. Conclusion

In this lab, we spent a significant amount of time in understanding, implementing and simulating the Universal Asynchronous Receiver/Transmitter. The most challenging part was comprehending the given data and then reproducing it. Our preliminary conceptual diagrams were confirmed with the TAs's. Overall, the lab helped us improve our grip on the understanding of the functionality of FPGA as wellas VHDL level coding.

References

http://fpga-fhu.user.jacobs-university.de/?page_id=402

https://www.xilinx.com/support/documentation/boards_and_kits/ug130.pdf