UNIVERSITY of NOTTINGHAM MALAYSIA

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

HARDWARE DESCRIPTION LANGUAGES FOR PROGRAMMABLE LOGIC
MODULE CODES: EEEE4075 & EEEE4076

VHDL Laboratory – RS232 Communication

1 EQUIPMENT LIST

- PC (running Windows XP)
- Xilinx ISE 10.1/14.1 software
- Xilinx Spartan-3E Starter Kit including:
  - Xilinx Spartan-3E Starter Board
  - 5V power supply unit
  - Low cost JTAG cable
  - Spartan-3E Starter Kit Board User Guide (Available on moodle)

2 COURSE REQUIREMENTS

This Laboratory forms part of the assessment towards the Hardware Descriptive Languages for Programmable Logic course (EEEE4075 & EEEE4076). A formal report must be **submitted online (Moodle)** on or before **Monday, 16th December 2019**. The usual penalties for late submission again apply.

The software used for this laboratory is the Xilinx ISE (Version 10.1/14.1) set of tools, which can be accessed via the computers in Micro processor Laboratory (Room DA19). If you encounter any software problems then contact Mr. MOHD SAIFUL BAHRIN (MD.Saiful@nottingham.edu.my)

NOTE: Please ensure that you have read through all source codes listed in Appendix and fully understood them before this lab session!

3 AIMS

The aim of this Laboratory is to

- Learn how to describe state machines based on VHDL coding.
- Provide basic skills in implementing communication protocols using state-machine design.
- Provide practical experience in digital hardware design.

Your task is to:

- o Understand RS232 communication protocol.
- o Ensure the receiver (Rx)design is bug free
- o Design the transmitter (Tx) design
- o Verify the RxTx controller design in simulation environment using the test bench provided
- o Implement the design in the Spartan-3E FPGA board.
- o Test the RS232 communication design to confirm the functionality.

NOTE: Make sure the RS232 cable (DB9) is properly connected between the PC and the Spartan-3E board!

4 A BRIEF INTRODUCTION TO RS232 COMMUNICATION PROTOCOL

RS232 communication link is used to connect between DTE equipment and DCE equipment. Fig.1 gives an example of RS232 communication link, which implements the interface between a computer (DTE) and a Modem device (DCE) by the RS232 cable. Although more powerful communication links have been becoming available, such as USB, IEEE1394 and etc, RS232 is still one of the most widely used serial communication links, because of its low cost and reliability.

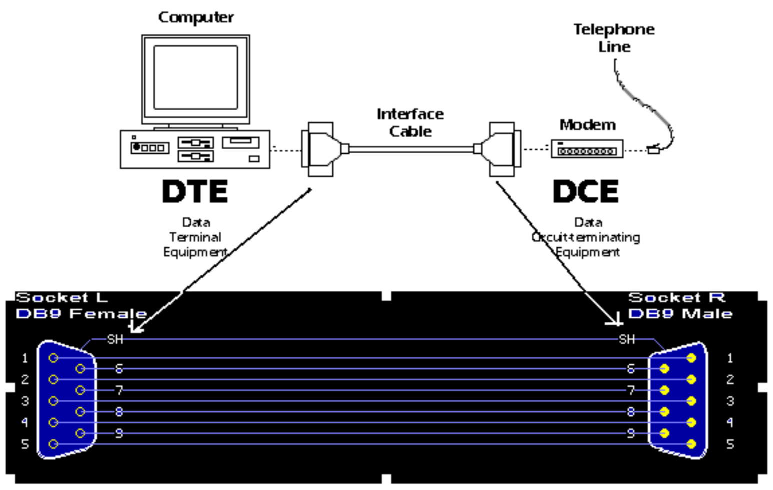


Fig.1 RS232 Communication link

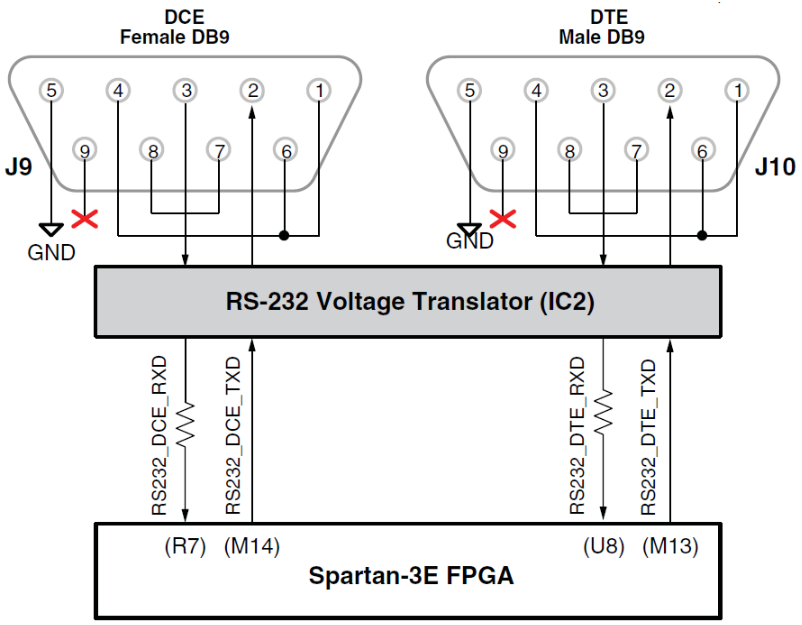


Fig 2 RS232 Connector in the Spartan 3E FPGA board

A brief description of RS232 signal pin definitions (DB9, DTE) are given below.

Pin 1 - Received Line Signal Detector (CD): This signal is relevant when the DCE device is a modem.
Pin 2 - Transmitted Data (TxD): This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic '1', negative voltage).

NOTE: Pin 2 on the DCE device is commonly labeled "Received Data", although by the EIA232 standard it should still be called Transmitted Data because the data is thought to be destined for a remote DTE device.

Pin 3 - Received Data (RxD): This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic '1', negative voltage).

NOTE: Pin 3 on the DCE device is commonly labeled "Transmitted Data", although by the EIA232 standard it should still be called Received Data because the data is thought to arrive from a remote DTE device.

Pin 4 - DTE Ready (DTR): This signal is asserted (logic '0', positive voltage) by the DTE device when it wishes to open a communications channel. If the DCE device is a modem, the assertion of DTE Ready prepares the modem to be connected to the telephone circuit, and, once connected, maintains the connection. When DTE Ready is deasserted (logic '1', negative voltage), the modem is switched to "on-hook" to terminate the connection.
Pin 5 – Ground: All signals are referenced to a common ground, as defined by the voltage on pin 5.
Pin 6 – DEC Ready: When originating from a modem, this signal is asserted (logic '0', positive voltage) when the following three conditions are all satisfied:

1 - The modem is connected to an active telephone line that is "off-hook";

2 - The modem is in data mode, not voice or dialing mode; and

3 - The modem has completed dialing or call setup functions and is generating an answer tone.

If the line goes "off-hook", a fault condition is detected, or a voice connection is established, the DCE Ready signal is deasserted (logic '1', negative voltage).

Pin 7 - Request to Send (RTS): This signal is asserted (logic '0', positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device.

Pin 8 - Clear to Send (CTS): This signal is asserted (logic '0', positive voltage) by the DCE device to inform the DTE device that transmission may begin.

Pin 9 - Ring Indicator (RI): This signal is relevant when the DCE device is a modem.

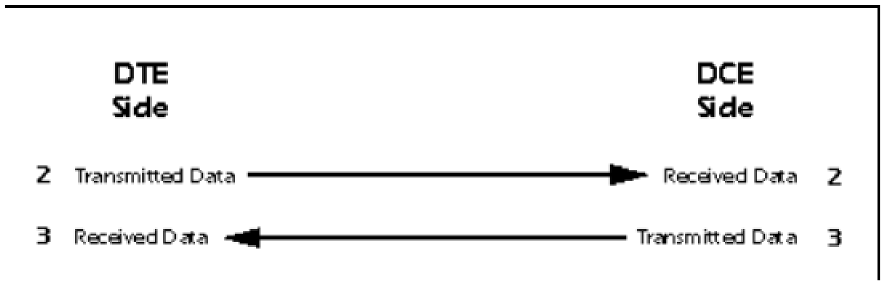


Fig 3 Key signal connection between DTE and DCE

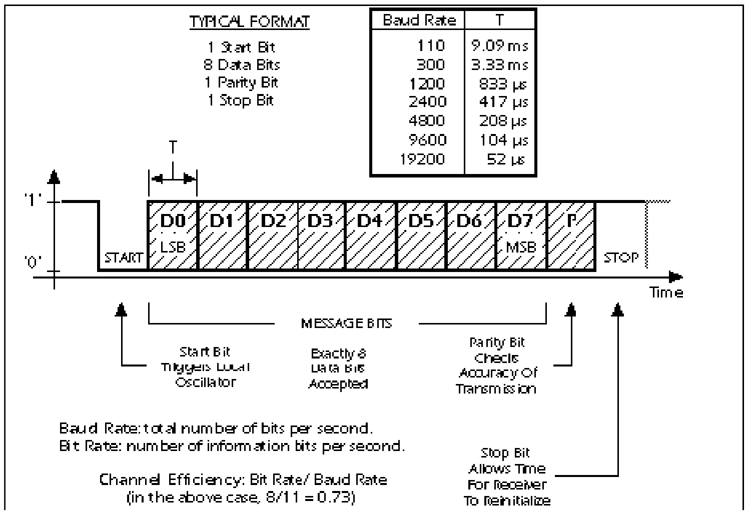

TYPICAL FORMAT
1 Start Bit
8 Data Bits
1 Parity Bit
1 Stop Bit
Baud Rate | T
110 | 9.09 ms
300 | 3.33 ms
1200 | 833 µs
2400 | 417 µs
4800 | 208 µs
9600 | 104 µs
19200 | 52 µs
T
'1'
'0'
START
D0
LSB
D1
D2
D3
D4
D5
D6
D7
MSB
P
STOP
Time
MESSAGE BITS
Start Bit Triggers Local Oscillator
Exactly 8 Data Bits Accepted
Parity Bit Checks Accuracy Of Transmission
Baud Rate: total number of bits per second.
Bit Rate: number of information bits per second.
Channel Efficiency: Bit Rate/ Baud Rate
(in the above case, 8/11 = 0.73)
Stop Bit Allows Time For Receiver To Reinitialize

Fig.4 RS232 Data Format

However, Transmitted Data (TxD) and Received Data (RxD) are the most important signals in RS232 serial communication. With these two signals connected as in Fig.3, the basic data communication can be implemented. The data format for TxD and RxD signals are given in Fig.4.

It can be seen from Fig.4 that a typical RS232 data packet consists of one start bit, 8 message bits, one parity bit and one stop bit, in which the start bit is always active low, while the stop bit is always active high. In the message area, the least significant bit (LSB) comes first and the most significant bit (MSB) comes last. In a practical implementation, the number of message bits and stop bits can be configured, respectively. Furthermore, the parity bit can be disabled for a reliable communication link to improve the channel efficiency, which is equal to the number of message bits divided by the total bits. Baud Rate, which is directly proportional to the channel's bandwidth and inversely proportional to the channel's noise levels, specifies the number of RS232 data bits to be transmitted/received per second. Therefore, the effective data rate can be worked out from the Baud Rate and the Channel efficiency as follows:

Effective data rate (bytes/sec) = (Baud rate * Channel efficiency)/8

In our RS232 communication design, we are using the Baud rate of 9600, one start bit, 8 message bits, no parity bit and one stop bit, so the Effective data rate is (9600 * (8/10))/8 bytes/sec, which is 960 bytes/sec. Obviously, this achieve the highest channel efficiency for the case of 8 message bits, because the start/stop bits are required and can not be disabled in the RS232 communication.
The above brief RS232 introduction should help you to understand the VHDL source codes. In addition, you can refer to the materials listed in Reference section if necessary.

5 PROJECT FILES

Source File	Type
RS232Rxd.vhd	VHDL Design File
RS232Txd.vhd	VHDL Design File
RS232.vhd	VHDL Design File
D4to7.vhd	VHDL Design File
Scan4Digit.vhd	VHDL Design File
TopLevelRS232VHDL.vhd	VHDL Design File
tb_TopLevelRS232VHDL.vhd	VHDL Test Bench File

6 SOURCE FILES

RS232Rxd.vhd: RS232 receiver.

RS232Txd.vhd: RS232 transmitter.

RS232.vhd simply packs the above RS232 receiver and transmitter into a single VHDL design.

TopLevelRS232VHDL.vhd is the top-level design and includes three components, D4to7, Scan4Digit and RS232, so that it allows in-coming RS232 serial data to be received and displayed on 7-segment LEDs, while the 8-bit parallel data set by eight slide switches can be sent out over the RS232 cable.

Tb_ TopLeveRS232VHDL.vhd is a test bench file for the entire RS232 communication design.

D4to7.vhd & Scan4Digit.vhd: Please refer to the lab 1 or lab 2 for these two files.

7 IMPLEMENTATION, PROGRAMMING AND TESTING THE DESIGN

7.1 Programming your RS232 communication design

BTN0 is used as a system reset signal and BTN1 is used to send out the data, which is set by eight slider switches, through RS232 cable. On the other hand, the data received from RS232 cable will be displayed on 4-character 7-segment LEDs. Specifically, the current data (8 bits) are displayed on the two right characters, while the previous data are shifted to the two left characters.

7.2 Setting up HyperTerminal (if the operating system is Windows 7 or above then use Putty for hyperterminal connection http://www.putty.org/)

Because you are going to use HyperTerminal program to test your the RS232 communication design, you need to set up the HyperTerminal as follows:
1) Launch HyperTerminal by clicking HyperTerminal, which can be located as in Fig.5.

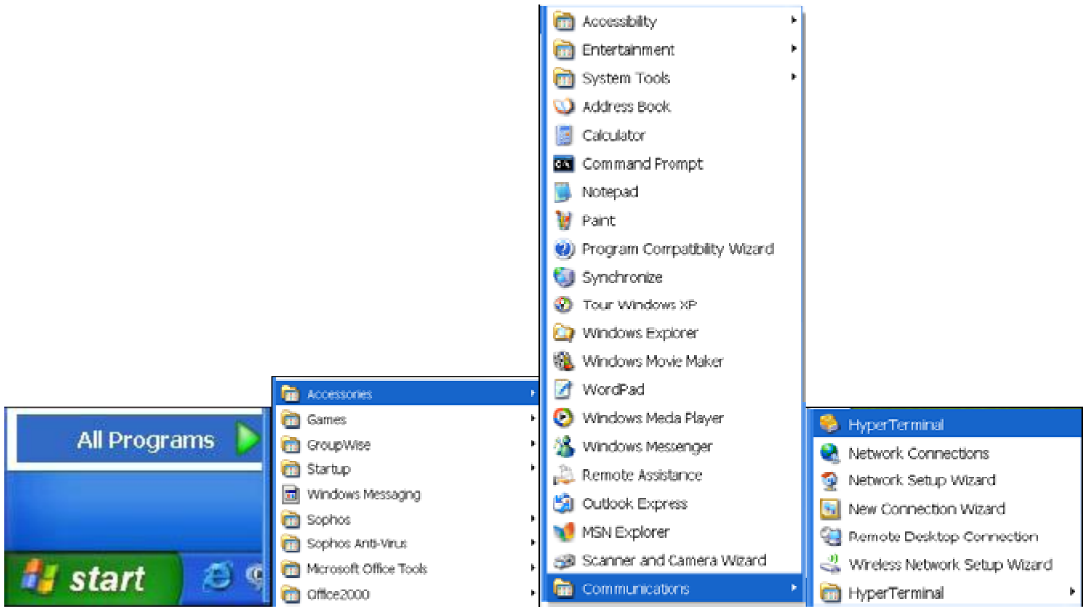
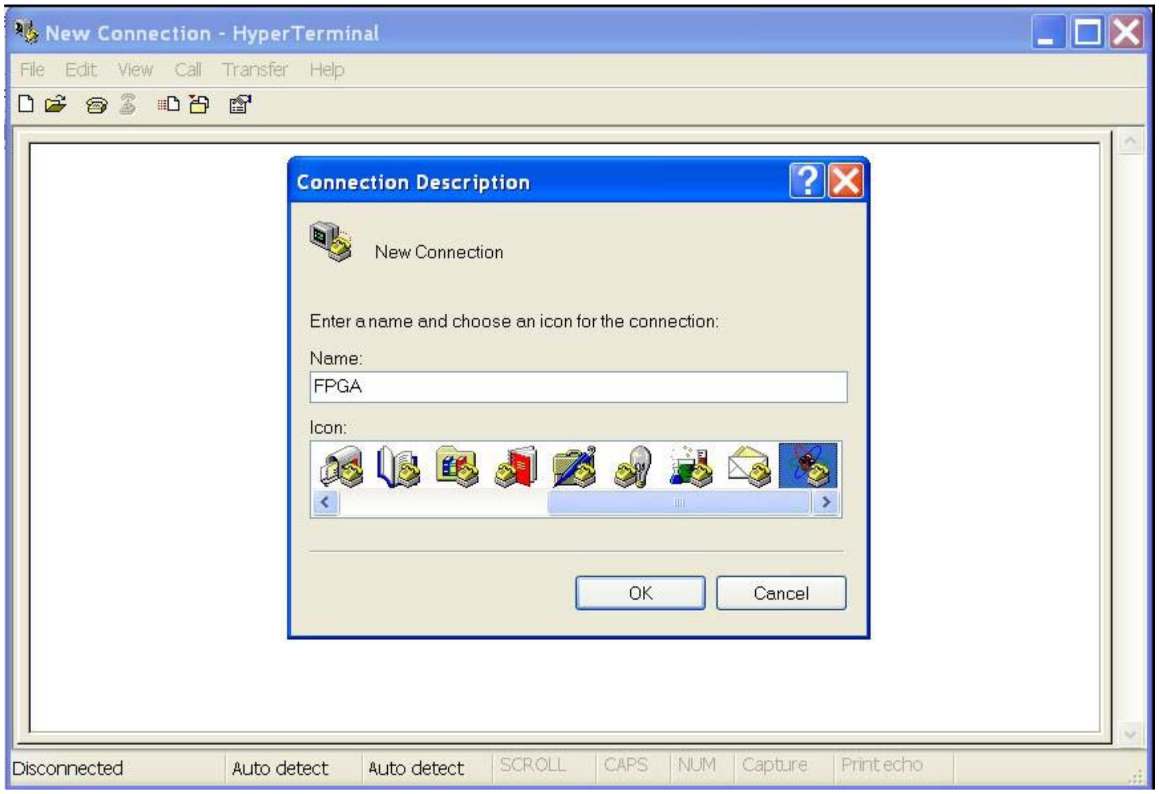


Fig.5 Launch HyperTerminal

2) Now you are led to Fig.6, in which you need to enter a name you like, FPGA for example and choose an icon for the connection you are going to set up, and then click OK.



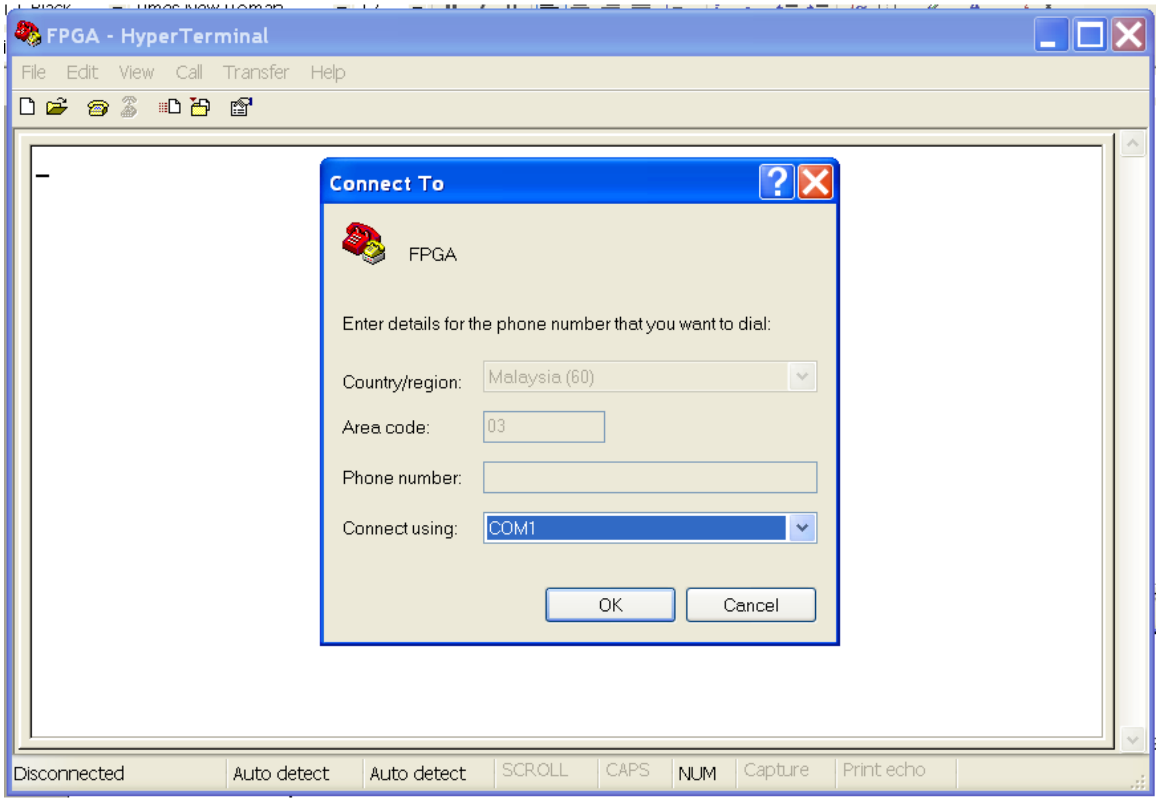
Enter a name and choose an icon for the connection:

Name: FPGA

Icon:

Fig.6 New HyperTerminal Connection

3) You should have a window that look like Fig.7, and simply click OK to get RS232 port setting windows as shown in Fig.8



Enter details for the phone number that you want to dial:

Country/region: Malaysia (60)

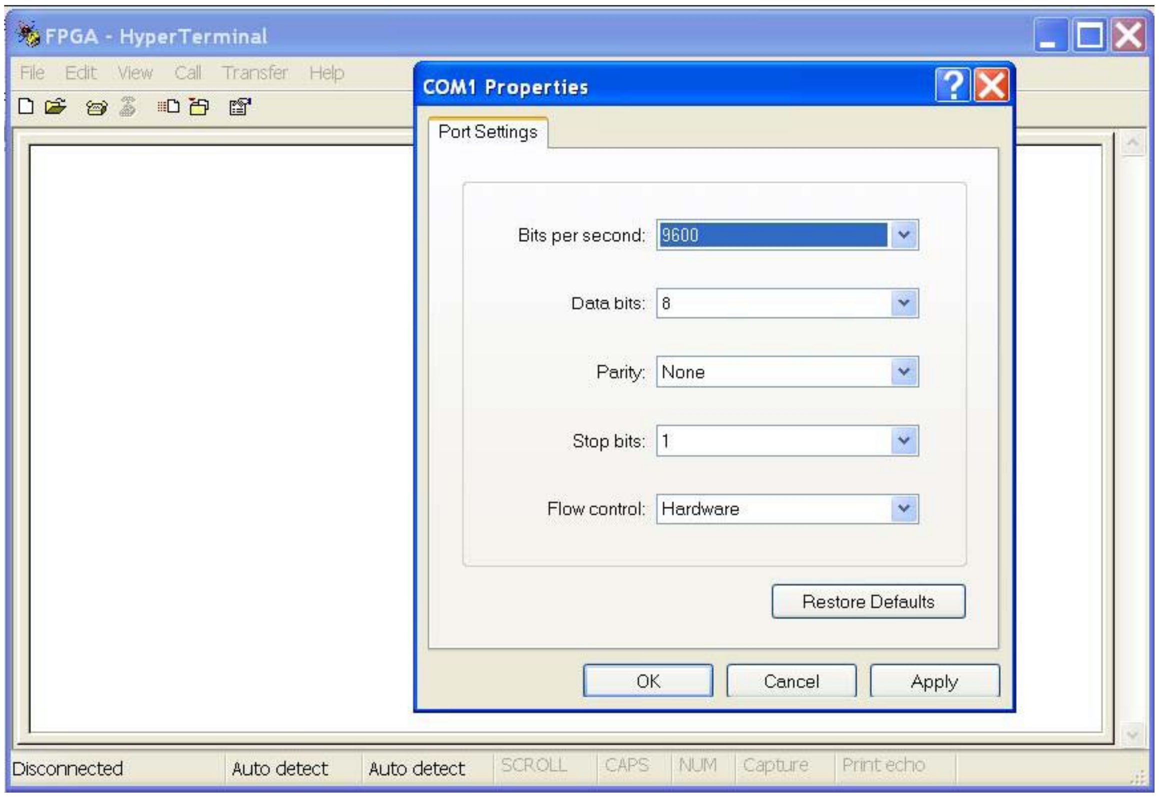
Area code: 03

Phone number:

Connect using: COM1

Fig.7 HyperTerminal

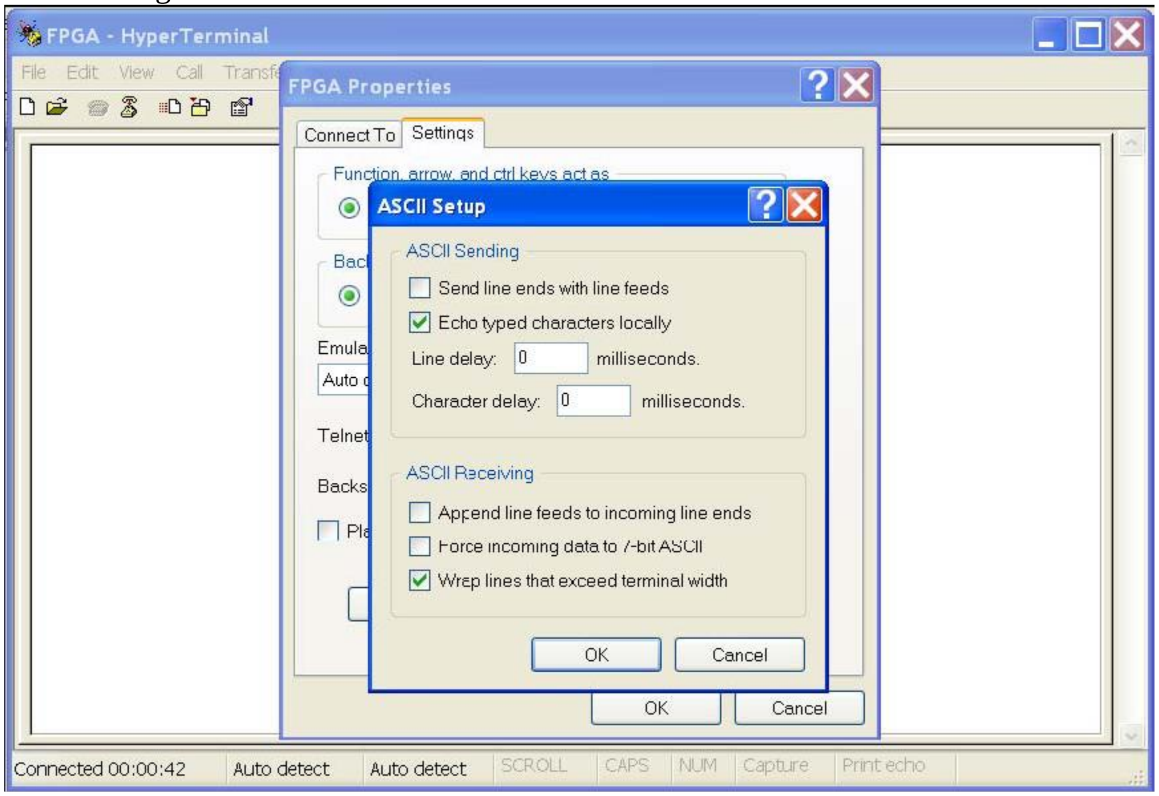
4) Select “9600”, “8”, “none”, “1” and “hardware” for Bits per second, Data bits, Parity, Stop bits and Flow control, respectively, and then click OK.



Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: Hardware

Fig.8 RS232 Port Setting

5) Select "Properties" under menu File to get a window called as FPGA Properties, in which ASCII setup window as shown in Fig.9 can be opened by pressing the button of "ASCII setup…" under the "Settings" tab. In the ASCII setup window, select "Echo typed characters locally" and then click OK. By clicking OK again in the window of "FPGA Properties", now you have properly set up and connected your HyperTerminal shown in Fig.10.



ASCII Sending

- [ ] Send line ends with line feeds
- [✓] Echo typed characters locally

Line delay: 0 milliseconds.

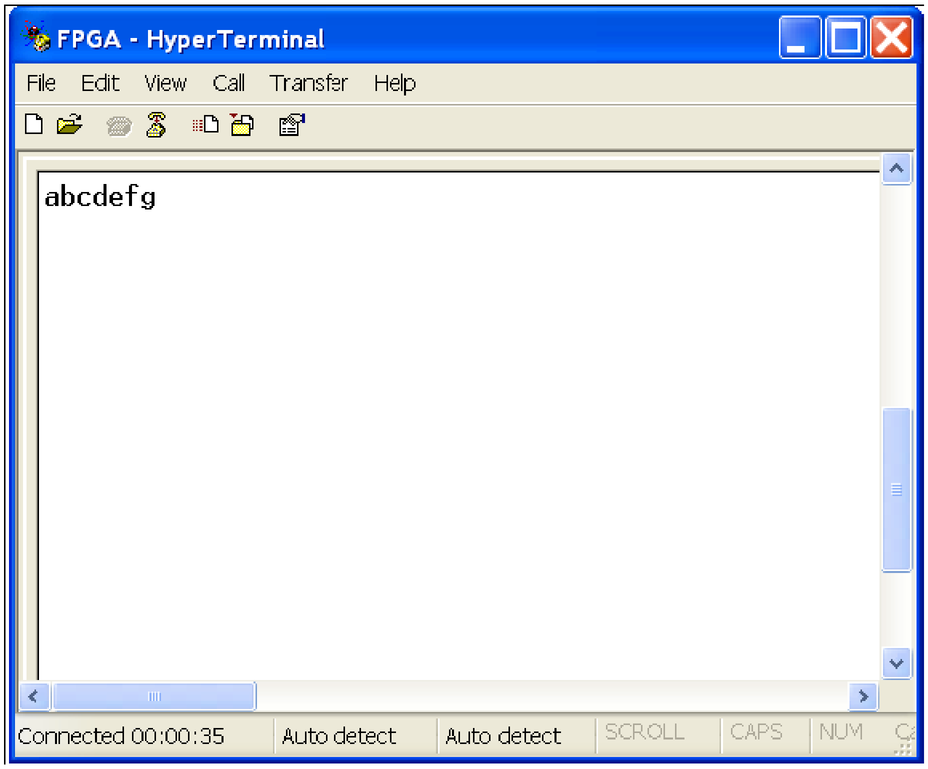
Character delay: 0 milliseconds.

ASCII Receiving

- [ ] Append line feeds to incoming line ends
- [ ] Force incoming data to 7-bit ASCII
- [✓] Wrap lines that exceed terminal width

Connected 00:00:42 | Auto detect | Auto detect | SCROLL | CAPS | NUM | Capture | Print echo

Fig.9 ASCII Setup



```
abcdefg
```

Fig.10 The Connected HyperTerminal

7.3 Testing your RS232 communication design

Now you are ready to test your RS232 communication design. The testing can be carried out in both directions as follows:

NOTE: Please be aware that ASCII (American Standard Code for Information Interchange) character standard is used in the HyperTerminal!

From the PC to the Spartan-3E board:

Enter any character in the HyperTerminal as shown in Fig.10, in two right-most characters 7-segment LEDs, you should see the ASCII code corresponding to the character you typed in. This ASCII code should be shifted into two left-most characters 7-segment LEDs after you enter another character, the ASCII code of which will be displayed in two right-most characters 7-segment LEDs. In other words, the ASCII codes keep moving from right to left whenever you type in a new character in the HyperTerminal. For example, "41" will be displayed in the 7-segment LEDs when you enter "A" in the HyperTerminal. In order to facilitate your test, the full ASCII codes are given in Table 1.

From the Sapartan-3E board to the PC:

Select any ASCII code through eight slide switches and send the ASCII code to the PC by pressing button BTN1, in the HyperTerminal, you should see the character corresponding to the ASCII code. Please check the results against Table 1.

Table 1 ASCII character standard:

ASCII (HEX)	SYMBOL	ASCII (HEX)	SYMBOL	ASCII (HEX)	SYMBOL	ASCII (HEX)	SYMBOL
00	NUL	20	(SPACE)	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	“	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	‘	47	G	67	g
08	BS	28	(	48	H	68	h
09	TAB	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	

8 DESIGN TASKS

The VHDL design files for the RS232 controller are provided. The receiver design is fully functional but transmitter design is incomplete.
So you are expected to

- Understand the function of the receiver design.
- Provide a new design for the transmitter part of the controller

9 FUNCTIONAL SIMULATIONS

With the provided test bench file, you are required to carry out functional & timing simulation by using ISE/Modelsim Behavioral simulation.

Having read through the VHDL source codes, you should know that the baud rate for the RS232 communication design is 9600 bits/second. To achieve the high reliability for the RS232 reception, the sampling clock with the frequency, which is equal to 16 times the baud rate, is adopted, in other words, the required sampling frequency is 153,600 Hz. On the other hand, as you know, 50MHz is the only available clock signal on the Spartan-3 starter board.

Therefore, the sampling frequency has to be generated from the on-board 50MHz clock by a clock divider with the divider factor of 325. That means that 325 cycles of the 50MHz clock is equivalent to one clock cycle of the sampling clock. In other words, from the viewpoint of simulation, 325 cycles of the 50MHz system clock are required to simulate one-cycle of the sampling clock, which is not efficient for the simulation of our RS232 communication design. There is an easy way to work around this problem, that is, *disable the clock divider by disconnecting iClock16x from iCount9(8) and then routing SystemClock signal directly to iClock16x signal in TopLevelRS232.vhd. Do* ***NOT*** *forget to enable the clock divider again after simulation.*

The following tasks require to be completed through your simulation:

- Print the functional/timing simulation waveforms with the main internal signals of RS232 communications design
- Using the simulation waveforms to explain the test results in section 7

10 REPORT:

A formal report should be written based on the above sections as a guide and include print outs of all simulation runs. Add a brief description at each stage of your observations. The expected length of the report is about twenty pages.
Your report should also consist of the following:
1) The state machines of the RS232 transmitter and receiver to detail the state transfers.
2) Explain the HDL design files and test bench files that has been newly developed. Include device utilization from the synthesis report
3) Simulation results with appropriate explanation.
3) VHDL codes of complete design (Please do not submit the complete project file)

REFERENCES

[1] Christopher E. Strangio, the RS232 Standard: a tutorial with signal names and definitions, http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html
[2] Christopher E. Strangio, Data Communications Basics: a brief introduction to digital data transfer, http://www.camiresearch.com/Data_Com_Basics/data_com_tutorial.html
[3] Xilinx, ISE In-depth Tutorial
[4] Xilinx, Spartan-3E Starter Kit Board User Guide

NOTE: A DEDUCTION OF 5% ABSOLUTE PER WORKING DAY WILL BE DEDUCTED FROM WORK SUBMITTED LATE.

Appendix

Testbench_toplevel

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

entity tb_TopLevelRS232 is
end tb_TopLevelRS232;

architecture behav of tb_TopLevelRS232 is

      component TopLevelRS232
      port( Reset, Send, Rxd: in std_logic;
            SystemClock: in std_logic;
            DataIn: in std_logic_vector (7 downto 0);
            An: out std_logic_vector (3 downto 0);
            Ca, Cb, Cc, Cd, Ce, Cf, Cg: out std_logic;
            Txd: out std_logic
            );
      end component;

      signal tb_reset, tb_send, tb_rxd, tb_systemclock : std_logic;
      signal tb_DataIn : std_logic_vector(7 downto 0) ;
      signal tb_an : std_logic_vector (3 downto 0);
      signal tb_ca, tb_cb, tb_cc, tb_cd, tb_ce, tb_cf, tb_cg, tb_dp :
std_logic;
      signal tb_txd : std_logic;
      signal tb_RxdData : std_logic_vector (7 downto 0);

      constant period : time := 6.5 us;

begin

      -- unit uder test
      UUT : TopLevelRS232 port map(
            Reset=>tb_reset,
            Send=>tb_send,
            Rxd=>tb_rxd,
            SystemClock=>tb_systemclock,
            DataIn=>tb_dataIn,
            An=>tb_an,
            Ca=>tb_ca,
            Cb=>tb_cb,
            Cc=>tb_cc,
            Cd=>tb_cd,
            Ce=>tb_ce,
            Cf=>tb_cf,
            Cg=>tb_cg,
            Txd=>tb_txd
            );

      -- produce clock signal
```

```
process
begin

tb_systemclock <= '1';
wait for period/2;
tb_systemclock <= '0';
wait for period/2;

end process;

-- produce reset signal
process
begin

tb_reset <= '1';
wait for 1.5*period;
tb_reset <= '0';
wait;

end process;

-- produce enable signal
process
begin

tb_rxd <= '1';
wait for 5.5*period;
tb_rxd <= '0'; -- Start bit
wait for 16*period;
tb_rxd <= '0'; -- Bit 0
wait for 16*period;
tb_rxd <= '1'; -- Bit 1
wait for 16*period;
tb_rxd <= '0'; -- Bit 2
wait for 16*period;
tb_rxd <= '0'; -- Bit 3
wait for 16*period;
tb_rxd <= '0'; -- Bit 4
wait for 16*period;
tb_rxd <= '1'; -- Bit 5
wait for 16*period;
tb_rxd <= '1'; -- Bit 6
wait for 16*period;
tb_rxd <= '0'; -- Bit 7
wait for 16*period;
tb_rxd <= '1'; -- Stop bit
wait for 16*period;
wait;

end process;

-- produce Send signal
process
begin

tb_send <= '0';
wait for 200*period;
```

```
        tb_send <= '1';
        wait for period;
        wait;

        end process;

        -- produce reset signal
        process
        begin

        tb_dataIn <= x"62";
        wait;

        end process;

end behav;
```

TopLevelRS232.vhd

```
entity Top_Level_RS232 is
port ( Rxd: in std_logic;
                    send,Reset: in std_logic;
                    SystemClock: in std_logic;
                    DataIn: in std_logic_vector (7 downto 0);
                    An: out std_logic_vector (3 downto 0);
                    Ca, Cb, Cc, Cd, Ce, Cf, Cg: out std_logic;
                    Txd: out std_logic;
                    Rxd:in std_logic);
end Top_Level_RS232;




architecture Behavioral of Top_Level_RS232 is


component RS232
            port( Reset, Clock16x, Rxd: in std_logic;
                        Send: in std_logic;
                        DataIn: in std_logic_vector(7 downto 0);
                        DataOut1: out std_logic_vector (7 downto 0);
                        Txd: out std_logic);
      end component;

      component D4to7
            port ( Q: in std_logic_vector (3 downto 0);
                        Seg: out std_logic_vector (6 downto 0));
      end component;

      component scan4digit
      port ( Digit3, Digit2, Digit1, Digit0: in std_logic_vector(6 downto 0);
                  Clock: in std_logic; An : out std_logic_vector(3 downto 0);
                  Ca, Cb, Cc, Cd, Ce, Cf, Cg: out std_logic);
      end component;
```

```
	signal iClock16x: std_logic;

	signal iClock : std_logic;
	signal iReset : std_logic;

	signal iDigitOut3, iDigitOut2, iDigitOut1, iDigitOut0: std_logic_vector (6 downto 0);
	signal iDataOut1: std_logic_vector (7 downto 0);
	signal iDataOut2: std_logic_vector (7 downto 0);
	signal iCount9: std_logic_vector (8 downto 0);


begin

	iDataOut2 <= DataIn;

process (SystemClock)
	begin
		if rising_edge(SystemClock) then
			if Reset = '1' then
				iCount9 <= (others=>'0');
			elsif
				iCount9 = "101000101" then -- the divider is 325, or "101000101"
				iCount9 <= (others=>'0');
			else iCount9 <= iCount9 + '1';
			end if;
		end if;
end process;

iClock16x <= iCount9(8);


	U1: RS232 port map (
				Reset => Reset,
				Clock16x => iClock16x,
				Rxd => Rxd,
				Send => Send,
				DataIn => DataIn,
				DataOut1 => iDataOut1,
				Txd => Txd);

	U2: D4to7 port map (
				Q => iDataOut1(3 downto 0),
				Seg => iDigitOut0);

	U3: D4to7 port map (
				Q => iDataOut1(7 downto 4),
				Seg => iDigitOut1);

	U4: D4to7 port map (

				Q => iDataOut2(3 downto 0),
				Seg => iDigitOut2);

	U5: D4to7 port map (

				Q => iDataOut2(7 downto 4),
                                        Seg => iDigitOut3);

        U6: scan4digit port map (
                                Digit3 => iDigitOut3,
                                Digit2 => iDigitOut2,
                                Digit1 => iDigitOut1,
                                Digit0 => iDigitOut0,
                                Clock => SystemClock,
                                An => An,
                                Ca => Ca,
                                Cb => Cb,
                                Cc => Cc,
                                Cd => Cd,
                                Ce => Ce,
                                Cf => Cf,
                                Cg => Cg);


end Behavioral;
```

RS232.vhd

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

entity RS232 is
        port( Reset, Clock16x, Rxd, Send: in std_logic;
              DataIn: in std_logic_vector(7 downto 0);
      DataOut1: out std_logic_vector (7 downto 0);
      Txd: out std_logic);
end RS232;

architecture RS232_Arch of RS232 is

        component Rs232Rxd
        port( Reset, Clock16x, Rxd: in std_logic;
      DataOut1: out std_logic_vector (7 downto 0));
        end component;

        component Rs232Txd
        port( Reset, Send, Clock16x: in std_logic;
              DataIn: in std_logic_vector(7 downto 0);
              Txd: out std_logic);
        end component;

begin

        u1: Rs232Rxd port map(
                Reset => Reset,
                Clock16x => Clock16x,
                Rxd => Rxd,
                DataOut1 => DataOut1);

        u2: Rs232Txd port map(
                Reset => Reset,
                Send => Send,
                Clock16x => Clock16x,
                DataIn => DataIn,
                Txd => Txd);

end RS232_Arch;
```

Rs232Rxd.vhd

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

entity Rs232Rxd is
        port( Reset, Clock16x, Rxd: in std_logic;
            DataOut1: out std_logic_vector (7 downto 0));
end Rs232Rxd;

architecture Rs232Rxd_Arch of Rs232Rxd is

        attribute enum_encoding: string;
-- state definitions
        type stateType is (stIdle, stData, stStop, stRxdCompleted);
        attribute enum_encoding of statetype: type is "00 01 11 10";

        signal presState: stateType;
        signal nextState: stateType;
        signal iReset, iRxd1, iRxd2, iClock1xEnable, iClock1x, iEnableDataOut: std_logic ;
        signal iClockDiv: std_logic_vector (3 downto 0) ;
        signal iDataOut1, iShiftRegister: std_logic_vector (7 downto 0) ;
        signal iNoBitsReceived: std_logic_vector (3 downto 0) ;

begin

        process (Clock16x)
        begin

        if Clock16x'event and Clock16x = '1' then
                if Reset = '1' or iReset = '1' then
                        iRxd1 <= '1';
                        iRxd2 <= '1';
                        iClock1xEnable <= '0';
                        iClockDiv <= (others=>'0');
                else
                        iRxd1 <= Rxd;
                        iRxd2 <= iRxd1;
                end if;
                if iClock1xEnable = '1' then
                        iClockDiv <= iClockDiv + '1';
                elsif iRxd1 = '0' and iRxd2 = '1' then
                        iClock1xEnable <= '1';
                end if;
        end if;

        end process;

        iClock1x <= iClockDiv(3);
```

```
	process (iClock1xEnable, iClock1x)
	begin

	if iClock1xEnable = '0' then
		iNoBitsReceived <= (others=>'0');
		presState <= stIdle;
	elsif iClock1x'event and iClock1x = '1' then
		iNoBitsReceived <= iNoBitsReceived + '1';
		presState <= nextState;
	end if;

	if iClock1x'event and iClock1x = '1' then
		if iEnableDataOut = '1' then
			iDataOut1 <= iShiftRegister;
		else
			iShiftRegister <= Rxd & iShiftRegister(7 downto 1);
		end if;
	end if;

	end process;

	DataOut1 <= iDataOut1;

	process (presState, iClock1xEnable, iNoBitsReceived)
	begin

	-- signal defaults
	iReset <= '0';
	iEnableDataOut <= '0';
	case presState is
		when stIdle =>
			if iClock1xEnable = '1' then
				nextState <= stData;
			else
				nextState <= stIdle;
			end if;
		when stData =>
			if iNoBitsReceived = "1001" then
				iEnableDataOut <= '1';
				nextState <= stStop;
			else
				iEnableDataOut <= '0';
				nextState <= stData;
			end if;
		when stStop =>
			nextState <= stRxdCompleted;
		when stRxdCompleted =>
			iReset <= '1';
			nextState <= stIdle;
		end case;

	end process;

end Rs232Rxd_Arch;
```

Rs232Txd.vhd

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

entity Rs232Txd is
        port( Reset, Send, Clock16x: in std_logic;
            DataIn: in std_logic_vector(7 downto 0);
            Txd: out std_logic);
end Rs232Txd;

architecture Rs232Txd_Arch of Rs232Txd is

        attribute enum_encoding: string;
        -- state definitions
        type stateType is (stIdle, stData, stStop, stTxdCompleted);
        attribute enum_encoding of stateType: type is "00 01 11 10";

        signal presState: stateType;
        signal nextState: stateType;
        signal iSend, iReset, iClock1xEnable, iEnableTxdBuffer, iEnableShift: std_logic;
        signal iTxdBuffer: std_logic_vector (8 downto 0);
        signal iClockDiv: std_logic_vector (3 downto 0);
        signal iClock1x: std_logic;
        signal iNoBitsSent: std_logic_vector (3 downto 0);

begin
```

DESIGN THE TRANSMITTER

```
end Rs232Txd_Arch;
```

--D4 to 7 Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity D4to7 is
   Port ( q : in  STD_LOGIC_VECTOR (3 downto 0);
          seg : out  STD_LOGIC_VECTOR (6 downto 0));
end D4to7;

architecture Behavioral of D4to7 is

begin
```

TNK

```
Seg<= "1111110" when q = "00000" else
          "0110000" when q = "000001" else
          "1101101" when q = "000010" else
          "1111001" when q = "00011" else
          "0110011" when q = "00100" else
          "1011011" when q = "00101" else
          "1011111" when q = "00110" else
          "1110000" when q = "00111" else
          "1111111" when q = "01000" else
          "1111011" when q = "01001" else
          "1110111" when q = "01010" else
          "0011111" when q = "01011" else
          "1001110" when q = "01100" else
          "0111101" when q = "01101" else
          "1001111" when q = "01110" else
          "1000111" when q = "01111" else
          "0000000";

end Behavioral;
```

--Scan4Digit Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity scan4Digit is
   Port ( digit0 : in  STD_LOGIC_VECTOR (6 downto 0);
          digit1 : in  STD_LOGIC_VECTOR (6 downto 0);
          digit2 : in  STD_LOGIC_VECTOR (6 downto 0);
          digit3 : in  STD_LOGIC_VECTOR (6 downto 0);
          clock : in  STD_LOGIC;
          an : out  STD_LOGIC_VECTOR (3 downto 0);
          ca : out  STD_LOGIC;
          cb : out  STD_LOGIC;
          cc : out  STD_LOGIC;
          cd : out  STD_LOGIC;
          ce : out  STD_LOGIC;
          cf : out  STD_LOGIC;
          cg : out  STD_LOGIC);
end scan4Digit;

architecture Behavioral of scan4Digit is
signal iCount16: std_logic_vector (15 downto 0) := (others=>'0');
signal iDigitOut: std_logic_vector (6 downto 0);
begin

-- Generate the scan clock 50MHz/2**16 (763Hz)
process(Clock)
begin
```

```
        if Clock'event and Clock='1' then
                iCount16 <= iCount16 + '1';
        end if;
end process;

--Send four digits to four 7-segment display using scan mode
with iCount16 (15 downto 14) select
        iDigitOut <= Digit0 when "00", -- Connect Digit0 to the 7-segment display
        Digit1 when "01", -- Connect Digit1 to the 7-segment display
        Digit2 when "10", -- Connect Digit2 to the 7-segment display
        Digit3 when "11", -- Connect Digit3 to the 7-segment display
        Digit0 when others;

with iCount16 (15 downto 14) select
        An <= "1110" when "00", -- with AN0 low only
        "1101" when "01", -- with AN1 low only
        "1011" when "10", -- with AN2 low only
        "0111" when "11", -- with AN3 low only
        "1110" when others;
        Ca <= iDigitOut(6);
        Cb <= iDigitOut(5);
        Cc <= iDigitOut(4);
        Cd <= iDigitOut(3);
        Ce <= iDigitOut(2);
        Cf <= iDigitOut(1);
        Cg <= iDigitOut(0);
end Behavioral;
```

******************************BEST WISHES**************************