IMPLEMENTATION OF UNIVERSAL ASYNCHRONOUS TRANSMITTER RECEIVER (UART) USING FPGA TECHNOLOGY

A dissertation submitted
in partial fulfilment of the requirements for the award of the degree of

Master of Engineering
In
Electronics and Communication Engineering

By

Vijay Pal

Roll No. 8738

Department of Electronics and Communication Engineering,
Delhi College of Engineering, University of Delhi
Session 2004-2006

ACKNOWLEDGEMENT

I wish to acknowledge our sincere thanks to my guide **Mr. A.K. Singh** in Electronics and Communication Department for there suggestions excellent guidance and timely advice which has made my thesis work success.

I express my deep sense of gratitude to Dr. **Asok Bhattacharyya**, H.O.D. (Department of Electronics and Communication Engineering) for there inspirations and timely help in conducting this work. I am indebted to the entire faculty and non teaching staff of Electronics and communication department, who had very helpful and cooperative to me at all times.

Vijay Pal (8738)

Department of Electronics and Communication Engineering
Delhi College of Engineering, University of Delhi
Session 2004-2006

DEPARTMENT OF ELECTRONICSANDCOMMUNICATION
DELHI COLLEGE OF ENGINEERING
UNIVERSITY OF DELHI
DELHI

CERTIFICATE

Certified that the thesis work entitled

IMPLEMENTATION OF UNIVERSAL ASYNCHRONOUS TRANSMITTER RECEIVER (UART) USING FPGA TECHNOLOGY

is bonafied work carried by

Vijay Pal (8738)

In partial fulfilment for the award of degree of Master of Engineering in Electronics and Communication Engineering of the University of Delhi during the year 2004-2006.It is certified that all corrections/suggestions indicated for internal assessment have been in corporate in the report deposited in the Departmental library. The project report has been approved as it satisfied the academic requirements in respect of thesis work prescribed for the Master of Engineering Degree.

Signature of Guide

Mr. A.K. Singh

Abstract

The Universal Asynchronous Receiver and Transmitter (UART) is a single chip device that provides a half duplex Asynchronous Receiver and Transmitter and a Baud rate generator for serial communication. As a peripheral device of a microcomputer system, the UART receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion. The Asynchronous Communication Element can support data rates from DC to 1.5 M Baud. The UART Asynchronous Communication Element (ACE) is a high performance programmable Universal Asynchronous Receiver/Transmitter (UART) and Baud Rate Generator (BRG) on a single chip (FPGA). The ACE's receiver circuitry converts start, data, stop, and parity bits into a parallel data word. The transmitter circuitry converts a parallel data word into serial form and appends the start, parity, and stop bits. The word length is 8 data bits. 2 Stop Bits are provided. The Baud Rate Generator divides the clock by a divisor programmable from 1 to 2^{16}-1 to provide standard RS-232C baud rates when using any one of three industry standard baud rate crystals (1.8432MHz, 2.4576MHz, or 3.072MHz) or any other depending on the clock frequency will be used to the user as after implementing it onto FPGA the clock frequency of 92 MHz can be used. A programmable buffered clock output (BAUDOUT) provides either a buffered oscillator or 16X (16 times the data rate) baud rate clock for general purpose system use.

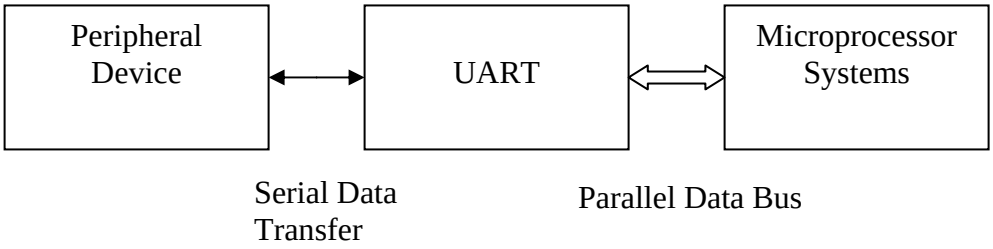
Table of Contents

Introduction

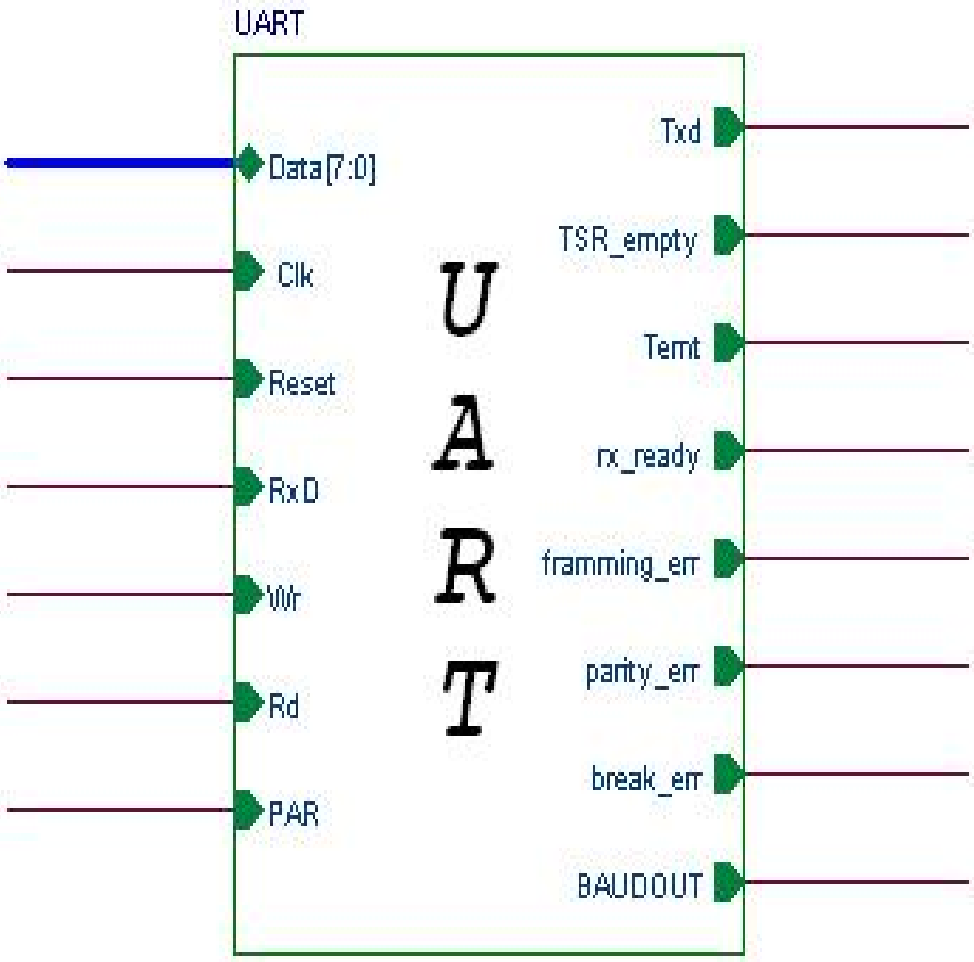
We will demonstrate, on a “Real-life” example, how a sound HDL technology can be used in conjunction with modern synthesis and simulation tool.

Computer and microprocessor base system often send and receive data in a parallel format, frequently these systems must communicate with external device that send and/or receive serial data. An interfacing device used to accomplish this conversion is the UART.

A UART is a specially designed integrated circuit that contain all the register and synchronizing circuitry necessary to receive data in a serial form and to convert and transmit it in parallel form and vice versa.



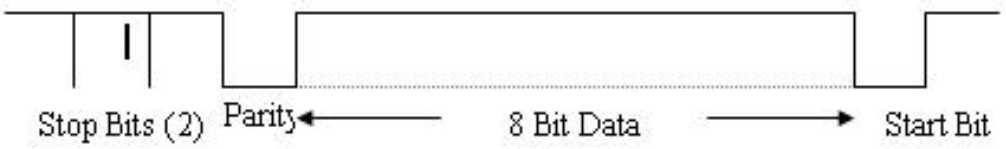
The Universal Asynchronous Receiver and Transmitter (UART) is a single chip device that provides a half duplex Asynchronous Receiver and Transmitter and a Baud rate generator for serial communication. As a peripheral device of a microcomputer system, the UART receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion. The UART Asynchronous Communication Element (ACE) is a high performance programmable Universal Asynchronous Receiver/Transmitter (UART) and Baud Rate Generator (BRG) on a single chip (FPGA). The ACE's receiver circuitry converts start, data, stop, and parity bits into a parallel data word. The transmitter circuitry converts a parallel data word into serial form and appends the start, parity, and stop bits. The word length is 8 data bits. 2 Stop Bits are provided. The Baud Rate Generator divides the clock by a divisor programmable from 1 to 2^{16}-1 to provide standard RS-232C baud rates when using any one of three industry standard baud rate crystals (1.8432MHz, 2.4576MHz, or 3.072MHz) or any other depending on the clock frequency will be used to the user as after implementing it onto FPGA the clock frequency of 92 MHz can be used. A programmable buffered clock output (BAUDOUT) provides either a buffered oscillator or 16X (16 times the data rate) baud rate clock for general purpose system use.



Overview of UART

Functional Description

Word Format



Description

1. The Transmitter accepts the parallel data from the CPU and converts it into a serial bit stream at the output.
2. It automatically sends a start bit (1) followed by the data stream (8 Bits), an optional parity bit (1), and the programmed number of stop bits (2).
3. The Least Significant Bit of the Data is transmitted first.
4. The Receiver accepts the start bit followed by the data, parity and stop bits and converts the serial data into Parallel format.

Data	:	8 Bits.
Start Bit	:	Logic ‘0’ (high to low Transmission).
Parity Bit	:	one Bit (Par = ‘1’: Odd Parity, Par = ‘0’: Even Parity)
Stop Bit	:	Two Bits (Logic high ‘1’).

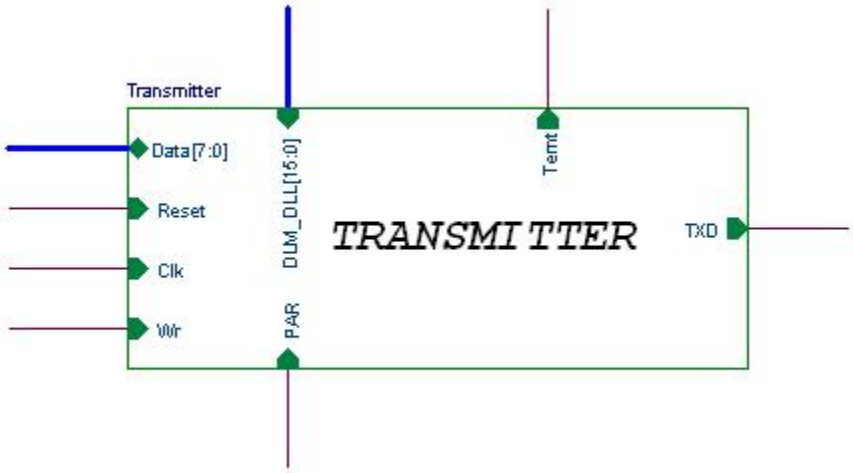
5. The Receiver Calculates the Parity on the Incoming Data and gives out the Parity Error if any.
6. It also calculates the Framing Error, i.e. it detects the invalid Stop Bit.
7. It generates a Break Error if it encounters a continuous Sequence of 0’s.

Transmitter

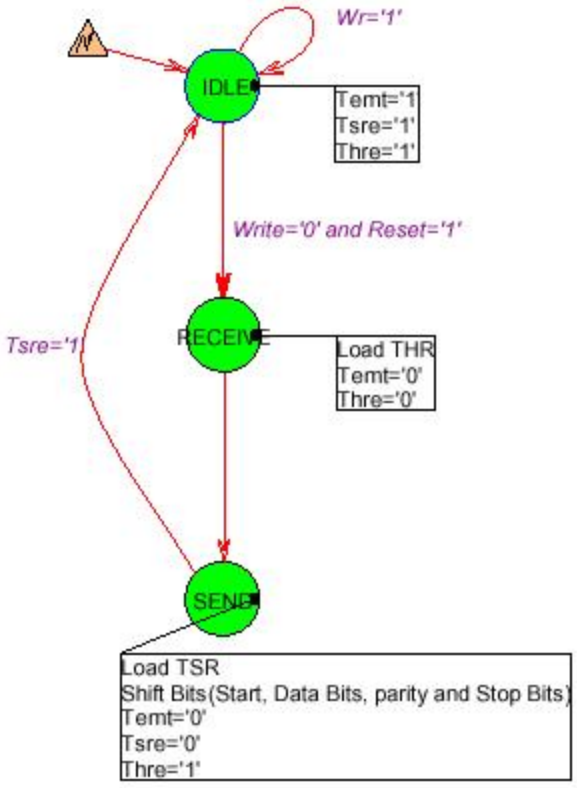
Description

1. The serial Transmitter block consists of Transmitter Hold Register (THR) and Transmitter Shift Register (TSR).
2. Transmitter hold Register Empty (THRE) and Transmitter Shift Register Empty (TSRE) are two signals which indicate the status of the THR and TSR.
3. To transmit the Data (8 Bit) Microprocessor should place the Data on to the 8 bit Data Bus.
4. Microprocessor should perform a write operation only if the THR is empty (i.e., Thre = '1')
5. Thre is set high automatically when the word is transferred from THR to TSR.
6. When the Transmitter is in the Idle state both the Thre and Tsre are set high indicating that both the registers are empty.
7. The first word written causes Thre to be low indicating that the THR is loaded.
8. THR loads the TSR with the available contents and returns a low and Tsre is made low indicating that the TSR is loaded.
9. Thre should remain low for at least the duration of the transfer of data from TSR.
10. After the completion of serial transfer of data from TSR THR will be loaded with a new data and Thre is made low again.

Block Diagram



FSM Implementation



Transmitter State Machine

IDLE

1. When Reset = ‘0’, Transmitter enters into the Idle State and remains in that state as long as Reset is Low (‘0’).
2. thre = ‘1,Tsre = ‘1’ and temt = ‘1’ indicating that the THR and TSR is empty.
3. THR is initialized with zeros and TSR is initialized to one’s.
4. Txd input is always high (‘1’).
5. Parity is assigned with PAR input.
6. A 4- Bit counter (bit_count) is initialized to zero’s which will sample each bit 16 times.
7. If Wr = ‘1’, Transmitter remains in the idle state.
8. If Wr = ‘0’, Transmitter goes into the RECEIVE State.

RECEIVE

1. THR gets loaded with the Data present on the Data Bus.
2. Thre = ‘0’ indicating that the THR is loaded.
3. Transmitter goes to the SEND State.

SEND

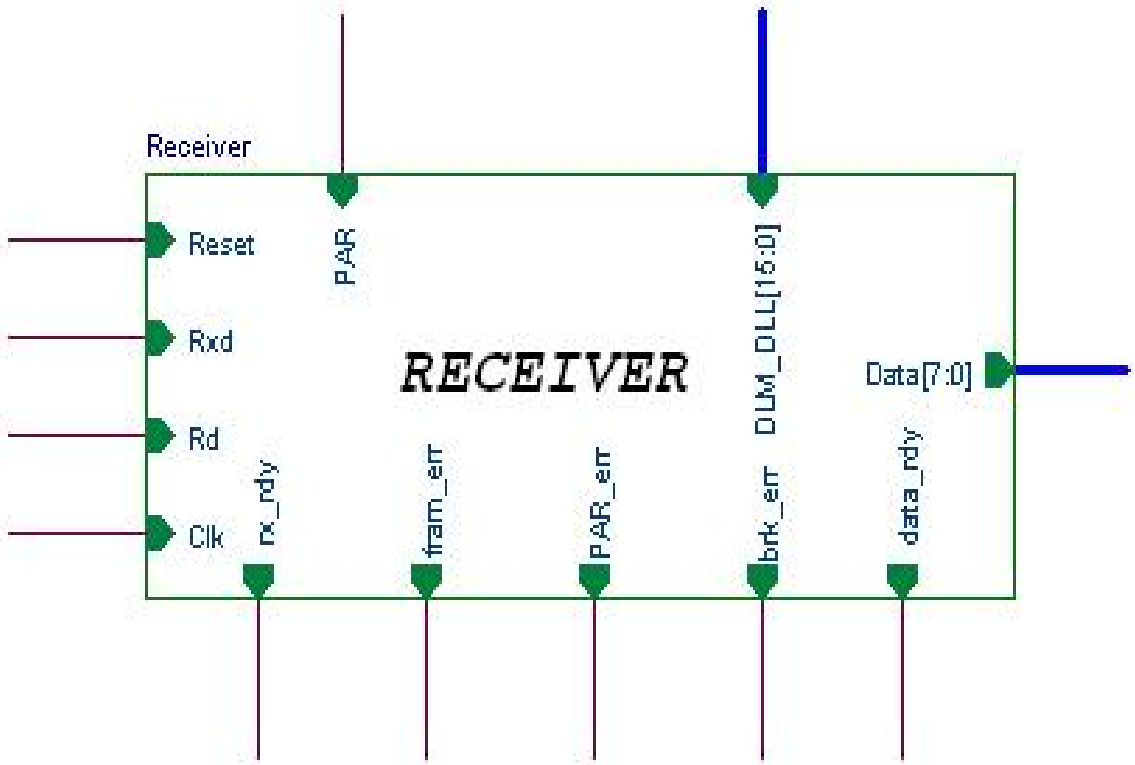
1. The Transmitter loads the TSR with the Contents of THR with a ‘0’ in the LSB.
2. The Transmitter sends the contents of TSR with respect to Bit_time So that each Bit is sampled 16 times and the value of the Bit at 8th sample is sensed and sent out.
3. The Transmitter sends the Start Bit(LSB of TSR) first.
4. The Transmitter calculates the Parity On the Data and sends it as the Parity Bit.
5. The Transmitter sends the Data one by one starting with the Start Bit followed by the Data word, Parity Bit and two Stop Bits.

RECEIVER

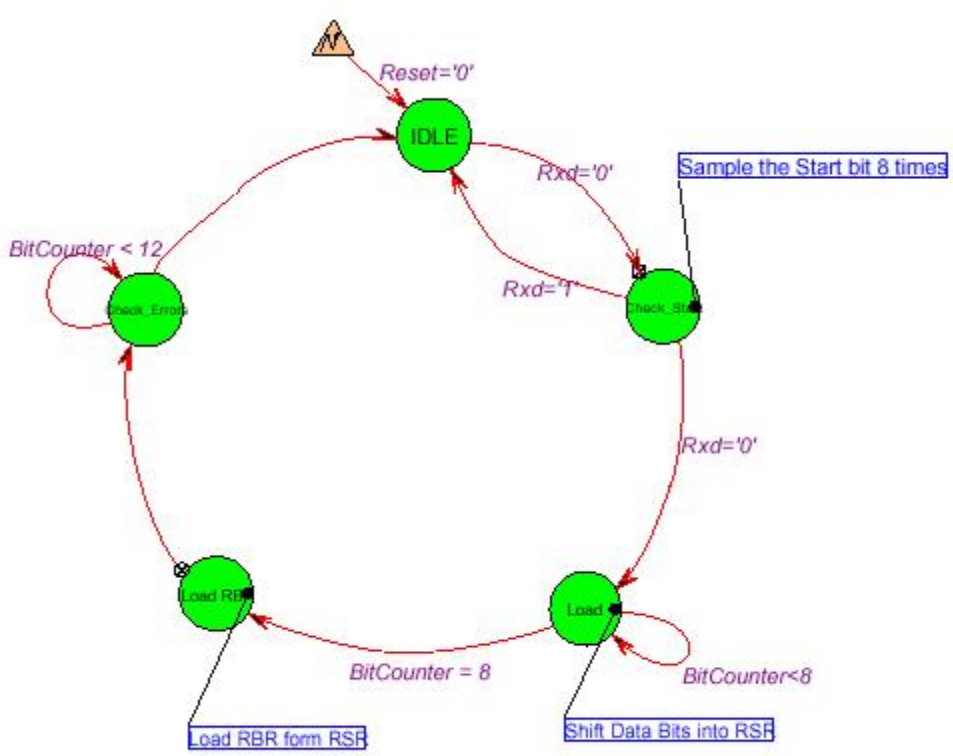
Description

1. Rxd will be the asynchronous input to the Receiver Block.
2. The Receiver Block will contain Receiver Shift Register (RSR) and Receiver Buffer Register (RBR).
3. A start Bit is detected from a high to low transition on the Rxd input.
4. After the detection of Start Bit, the contents of Rxd are loaded onto RSR one by one.
5. RSR is loaded with the 8 bit Data followed by parity and stop bits.
6. After RSR getting loaded RSR loads the RBR with the available data followed by parity bit and stop bit.
7. After the transfer of data from RSR to RBR the Receiver gives out a data_ready signal indicating that the Receiver is ready to transfer the data.
8. CPU will be able to read from the Receiver by giving a read signal.
9. Receiver will calculate the Parity on the Data word depending on even or odd parity.
10. It checks with the input parity and gives out a parity error (parity_err) signal if any.
11. It also checks for the incorrect stop bits and gives out a framing error if any.
12. If a new Data is being written before the previous data has been read then the Receiver will generate a overrun error.

Block Diagram



FSM Implementation



Receiver State Machine

IDLE STATE

Initially on Reset, the Receiver goes into an IDLE state. In this state the Receiver waits for the low going start bit on the Serial input (Rxd) line. The receiver goes to the CHECK_START state when a falling edge occurs on the Rxd input.

CHECK_START

The Receiver samples the Rxd line when Rx_clk_rise is ‘1’. If the Rxd is found to be ‘1’, the state machine moves back to the IDLE state. If the Start bit is ‘0’ still, the state machine moves into the LOAD state

LOAD

In this state, the Serial input data on the Rxd line is shifted into the Receiver’s Shift Register when ever the Rx_clk_rise signal is ‘1’. The state machine receives 8 data bits and then proceeds to the LOAD_RBR state.

LOAD_RBR

In this state, the Receiver’s Buffer Register (RBR)is loaded from the Shift Register(RSR).

CHK_ERR

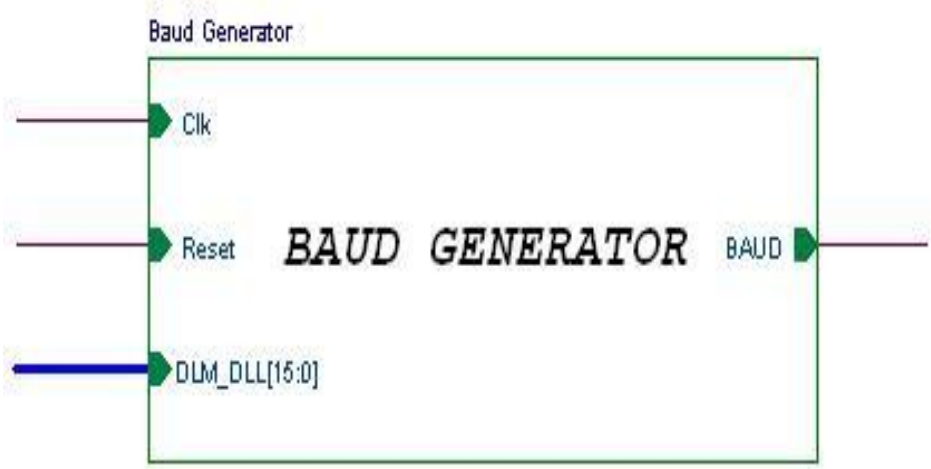
The Receiver in this state checks for the possible Parity, Framing and the Break Errors on the Rxd line.

BAUD Generator

Description

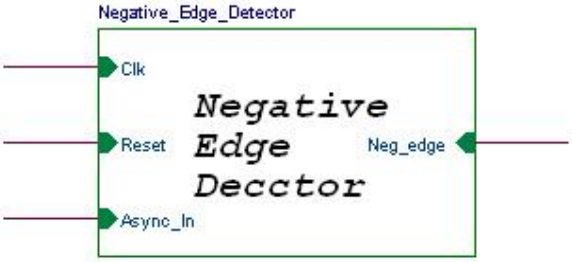
1. When Rst = ‘1’ Counter gets initialized and the output is 0.
2. It consists of a 16 bit Counter which gets loaded with the value in DLM_DLL when the count is 0.
3. When the Count is not equal to 0 it gets decremented by 1 with respect to the CLK.
4. See Wave forms.

Block Diagram

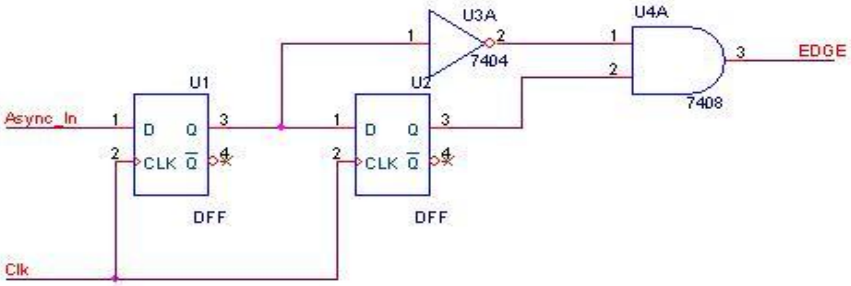


Negative Edge Detector

Block Diagram



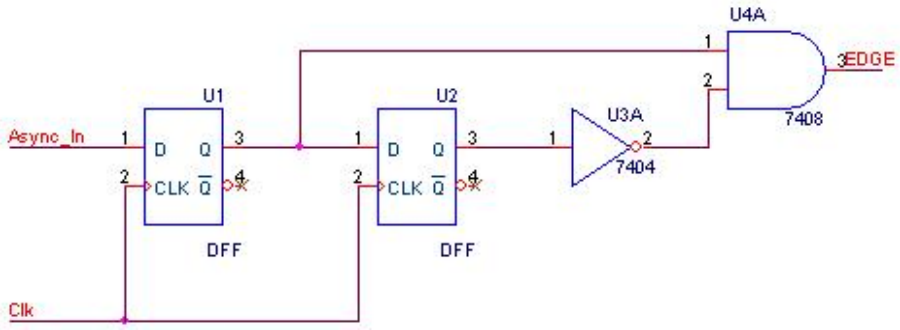
Circuit Diagram



Positive Edge Detector

Block Diagram

Circuit Diagram



Main Features

1. Asynchronous RS232 character-based transmit / receive function
2. Very compact (around 137 slices in the Virtex family are used)
3. Total Equivalent Gate Count for device is 2338.
4. This includes the baud rates generator.
5. Good timing performance. Maximum clock frequency can be used is 120.380MHz.
6. Maximum combinational path delay: 13.836ns.
7. Internal Baud rate generator with any baud rates selectable code.
8. Synthesizable to any technology (FPGA or ASIC).
9. Code tested on Xilinx Foundation Series F2.1i Synthesis tool.

Pin Description

S.No	Name	I/O	Width	Function
1	Data_Bus	I/O	8 bits	Holds 8 bit data from which CPU can read from Rx or write to Tx.
2	Reset	I	1 Bit	When Active (‘1’) UART will be in IDLE State.
3	U_Clk	I	1 Bit	Clk input to the Rx and Tx.
4	Rxd	I	1 Bit	Serial input data to the UART
5	Txd	O	1 Bit	Serial output data from UART
6	Rd	I	1 Bit	When Active (‘0’), CPU reads from UART
7	Wr	I	1 Bit	When Active (‘0’), CPU writes to UART
8	CS	I	1 Bit	When Active (‘0’), UART is selected
9	PAR	I	1 Bit	When 1: Odd Parity calculation, 0: Even parity Calculation.
10	DLM_DLL	I	16 Bits	Decides the Baud Rate.
11	Tsr_Empty	O	1 Bit	Indicates that the Tx is ready to Accept New Data.
12	Rx_Ready	O	1 Bit	Indicates that the Rx has accepted the Data.
13	Parity_err	O	1 Bit	Indicates that a parity error has occurred.
14	Framing_err	O	1 Bit	Indicates an Invalid Stop Bit.
15	Break_err	O	1 Bit	Indicates that a Stream of 0’s has been transmitted.
16	BAUDOUT	O	1 Bit	Baud rate clock for general purpose system use

Understanding the Code

Before going for the design flow and synthesis, let's understand the code of the modules used in it. As I have already described the various modules and their functions, now we will see how they are actually implemented through VHDL. The various blocks used are:

1. Baud Rate Generator.
2. Negative Edge Detector.
3. Positive Edge Detector.
4. Transmitter Clock Generator.
5. Receiver Clock Generator.
6. Transmitter Module.
7. Receiver Module.
8. Top Module (used to synchronize all other modules).

Code files used are:

uart_pkg.vhd

Declares the package UART. It is used here to store the common declarations like here it is used to store dlm_dll(Divisor).

baud_generaotr.vhd

Generates a signal that is 16 times the UART clock, in baud rate.

positive_edge_detector.vhd

Implements the rising edge detector

negative_edge_detector.vhd

Implements the falling edge detector

transmitter_clock.vhd

Generates a single pulse every 16 Cycles of Baudx16

receiver_clock.vhd

Generates the Baud Clock for Receiver

transmitter.vhd

Implements the UART TRANSMITTER Module

receiver.vhd

Implements the UART RECEIVER Module

uart.vhd

Top Module of UART

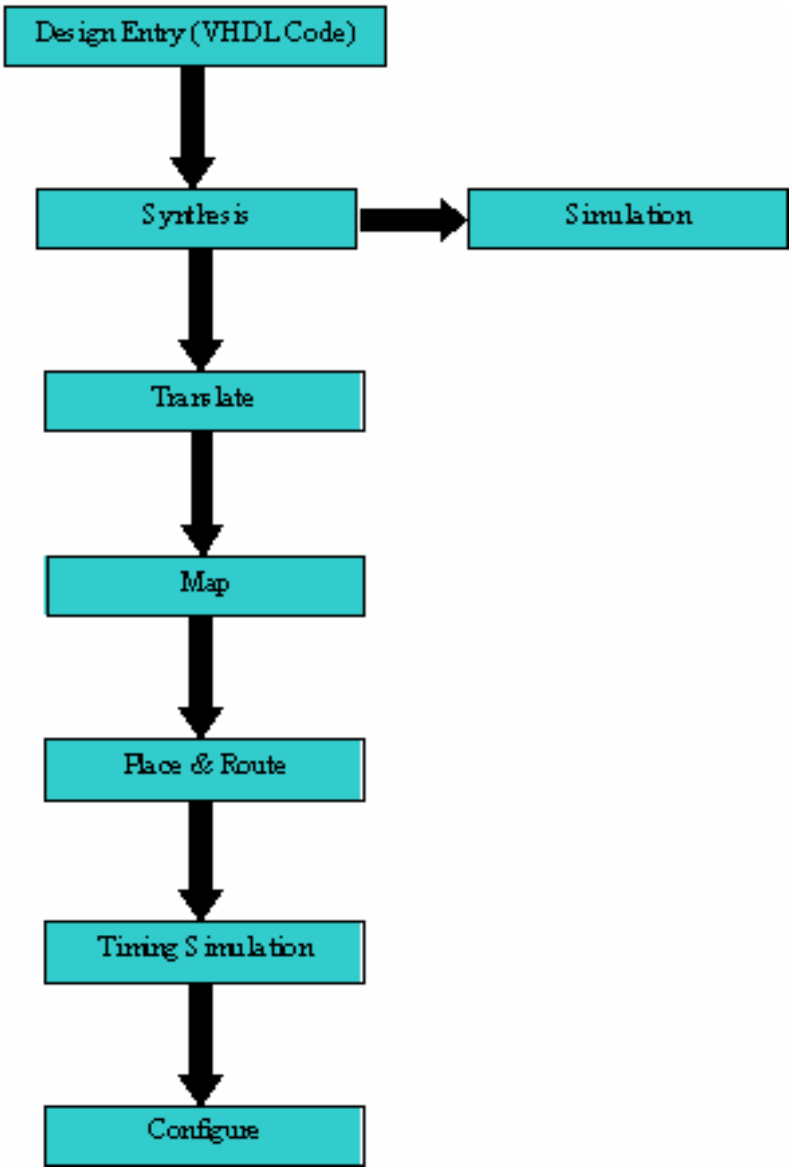
Uarttb.vhd

Test bench to check the transmitter receiver function.

Code files are given in the Appendix.

Design Flow

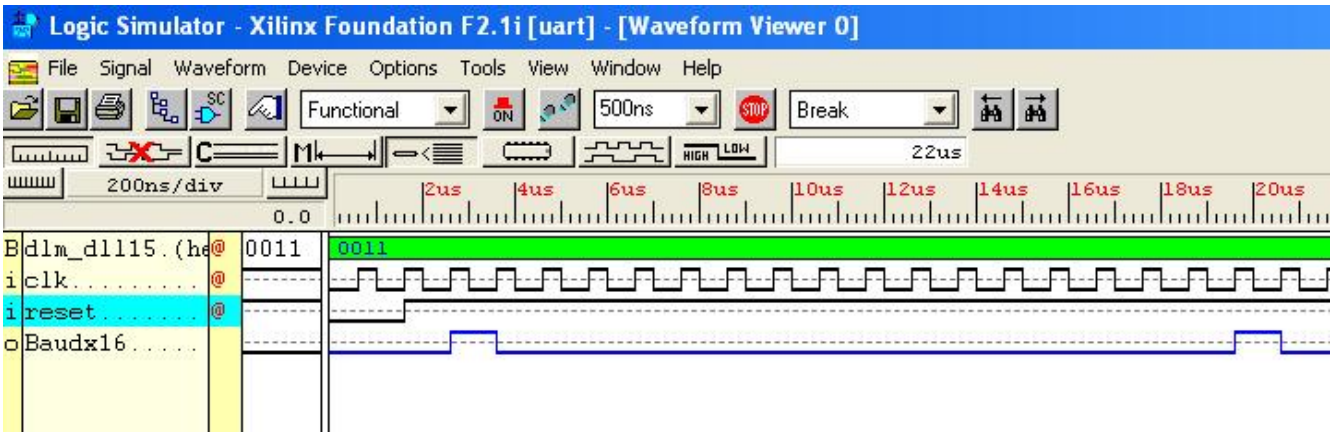
The original files were given in VHDL. The first Step is to synthesize your design. Syntax is checked here. Errors and violations of synthesis constraints occurring in the source code are reported. As we will see later, initially some mistakes had to be corrected and some improvements were needed. The next steps are summarized in the design flow chart below, using the Xilinx tools.



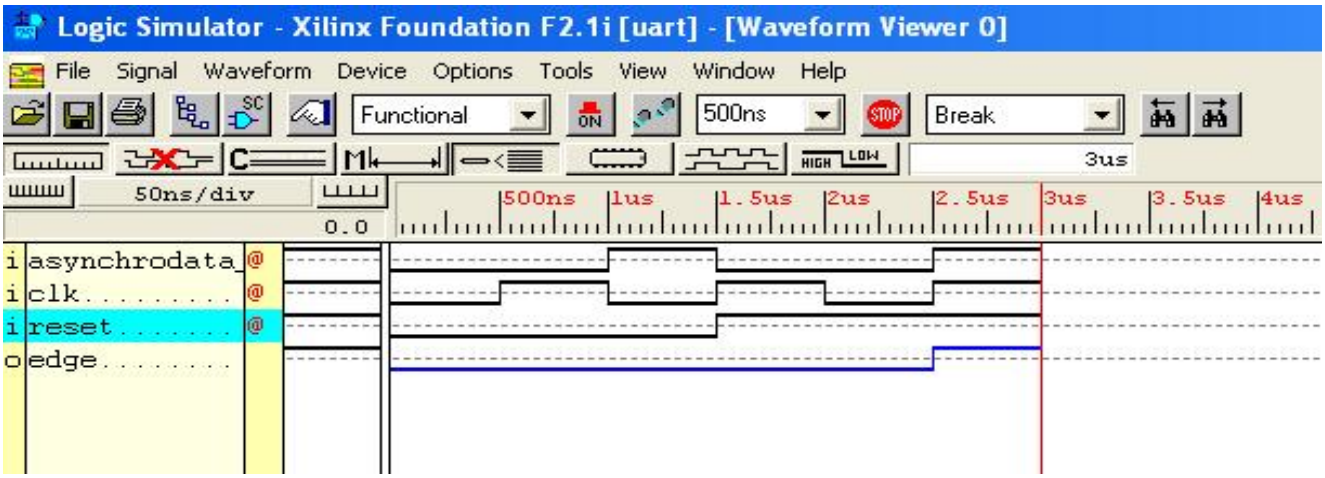
Design Flow

Output Waveforms

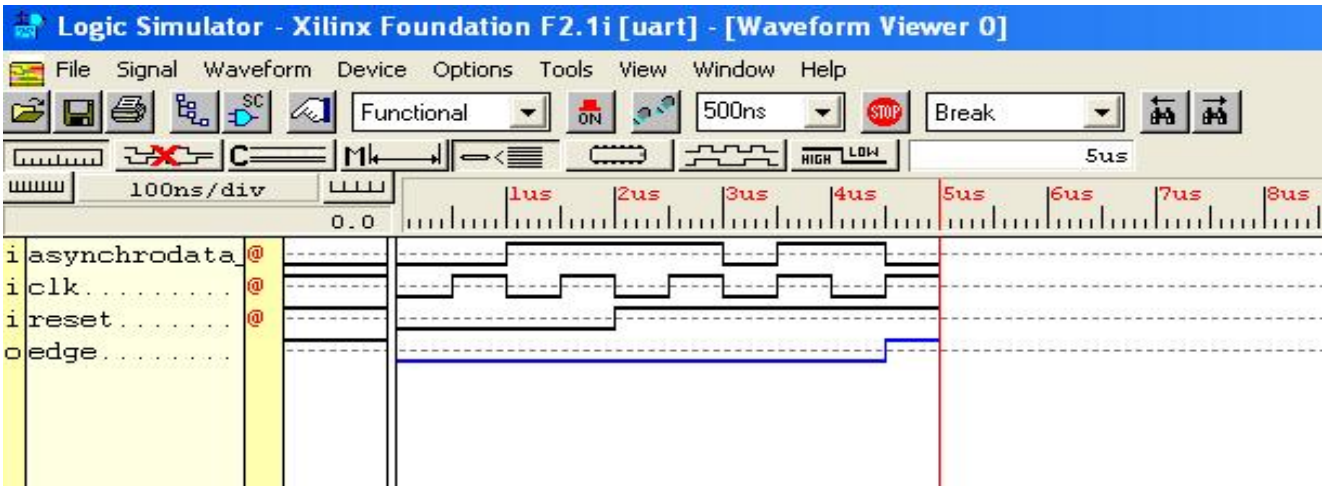
Baud Generator Output



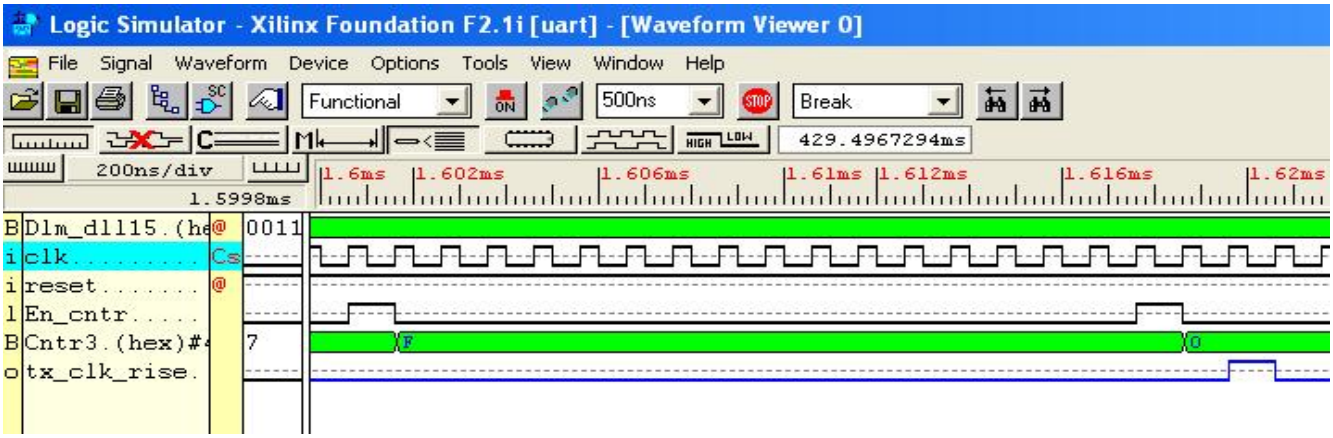
Positive Edge Detector Output



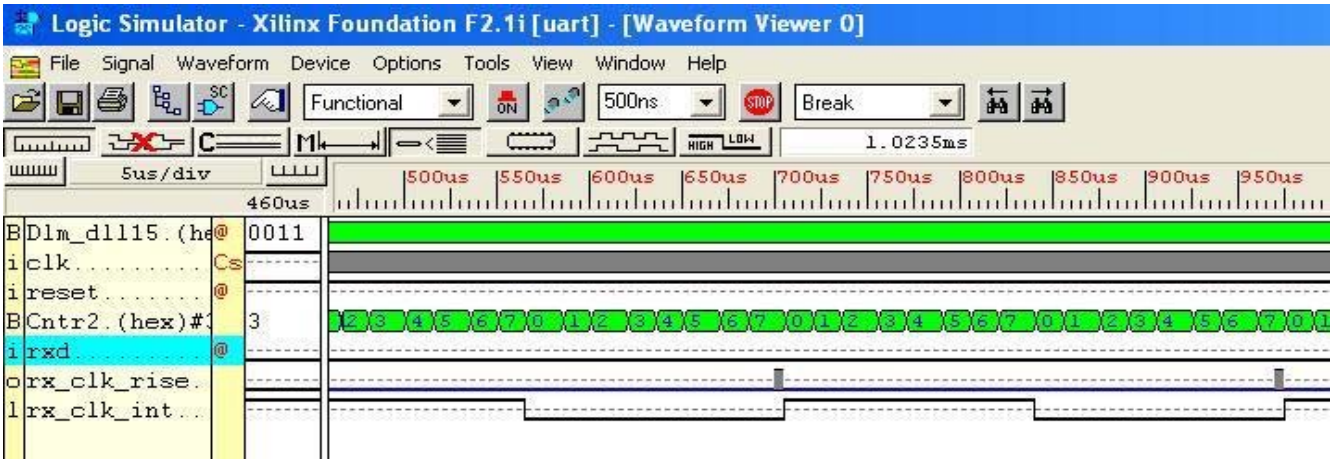
Negative Edge Detector Output



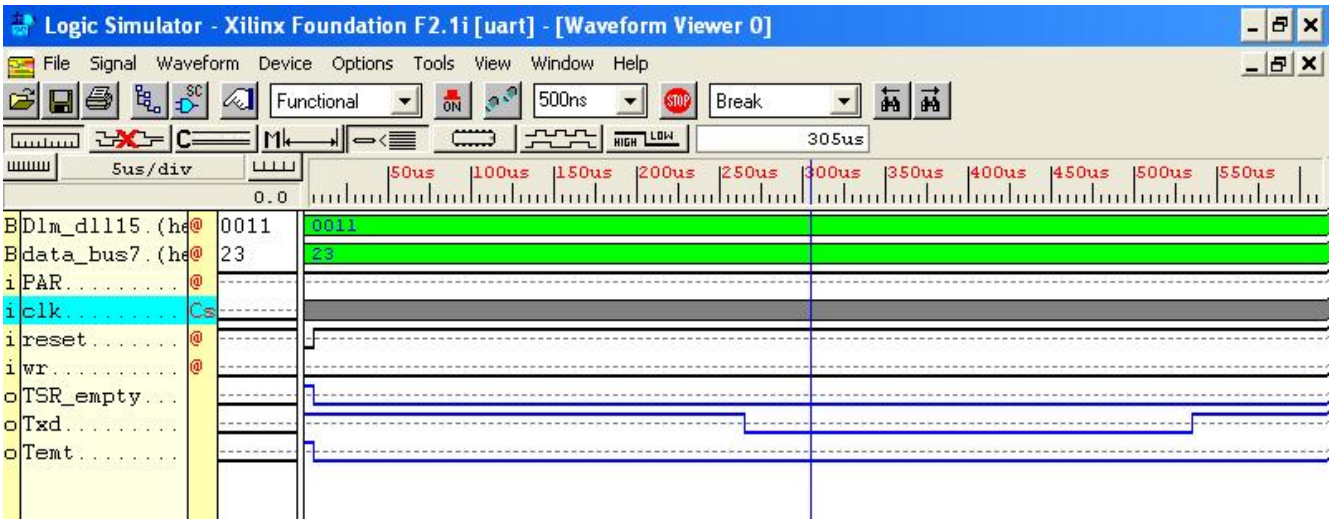
Transmitter Clock Output



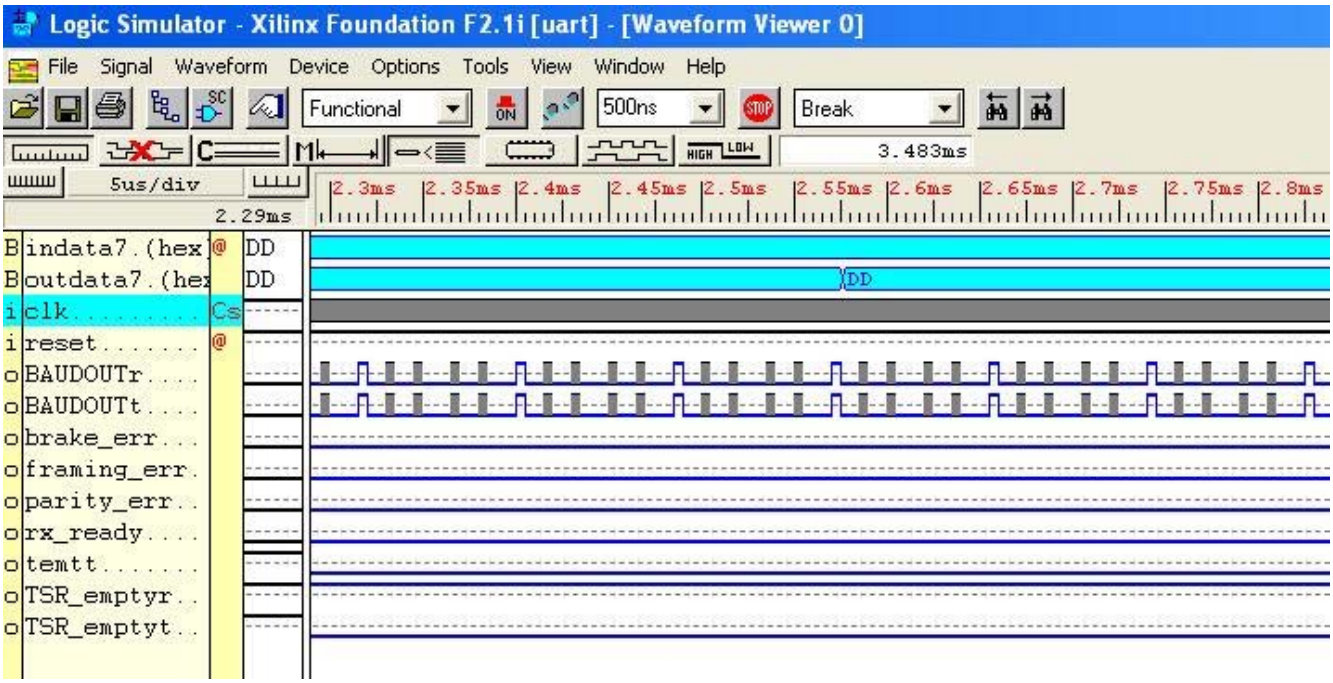
Receiver Clock Output



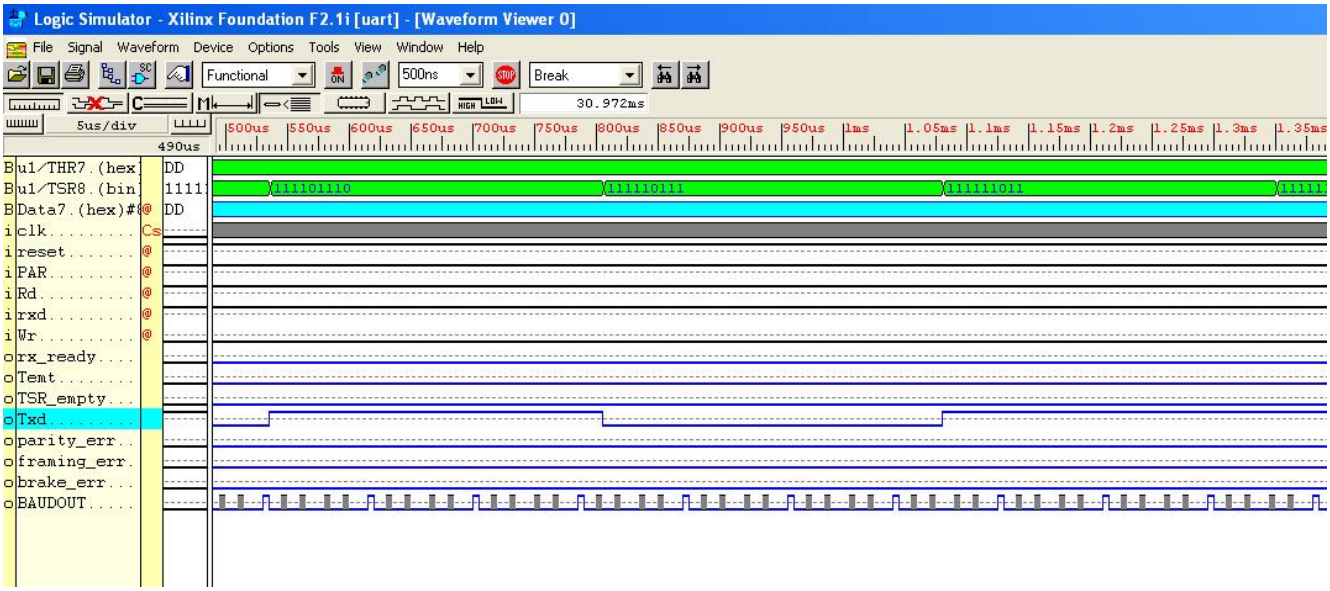
Transmitter Output



Receiver Output



UART Output


Logic Simulator - Xilinx Foundation F2.1i [uart] - [Waveform Viewer 0]
File Signal Waveform Device Options Tools View Window Help
Functional
500ns
Break
30.972ms
5us/div
490us
u1/THR7.(hex)
u1/TSR8.(bin)
Data7.(hex)#
clk
reset
PAR
Rd
rxd
Wr
rx_ready
Temt
TSR_empty
Txd
parity_err
framing_err
brake_err
BAUDOUT
11101110
11110111
11111011

Appendices

VHDL Codes

uart_pkg.vhd

```
1: ----------------------------------------------------------------------------
2: --Function : Package declaration of UART
3: --File : uart_pkg.vhd
4: --Date : 20/6/2006
5:--
6: --Version : 1.1
7: ----------------------------------------------------------------------------
8: Library ieee;
9: use ieee.std_logic_1164.all;
10:
11: package uart_pkg is
12:
13: constant dlm_dll : std_logic_vector(15 downto 0) := "0000000000010001";
14:
15: end uart_pkg;
```

baud_generaotr.vhd

```
1: ----------------------------------------------------------------------
2: --Function : Generate the 16 times the baud rate signal
3: -- from Clk (UART Clock)
4: --File : baud_generator.vhd
5: --Date : 20/6/2006
6: --
 7: --Version : 1.1
8: ----------------------------------------------------------------------
9:
10: Library ieee;
11: use ieee.std_logic_1164.all;
12: use ieee.std_logic_unsigned.all;
13:
14: Entity baud_generator is
15: port(
16: clk : in std_logic;
17: reset : in std_logic;
18: dlm_dll : in std_logic_vector(15 downto 0);
19: Baudx16 : out std_logic
20: );
21: End baud_generator;
22:
23: Architecture Arch of baud_generator is
24:
25: Signal counter : std_logic_vector(15 downto 0);
26:
27: Begin
28:
29: process(clk,reset)
30: Begin
31: If(reset = '0')then
32:
33: counter <= (others=> '0');
34: Baudx16 <= '0';
35:
36: Elsif(clk'event and clk = '1')then
```

```
37:
38: If(counter = "0000000000000000")then
39: counter <= dlm_dll - 1;
40: baudx16 <= '1';
41: Else
42: counter <= counter - 1;
43: baudx16 <= '0';
44: End if;
45:
46: End if;
47: End process;
48:
49: End Arch;
```

positive_edge_detector.vhd

```
1: ----------------------------------------------------------------------
2: --Function : Implements the rising edge detector
3: --File : positive_edge_detector.vhd
4: --Date : 20/6/2006
5: --
 6: --Version : 1.1
7: ----------------------------------------------------------------------
8:
9: Library ieee;
10: use ieee.std_logic_1164.all;
11:
12: Entity positive_edge_detector is
13: port(
14: asynchrodata_in : in std_logic;
15: reset : in std_logic;
16: clk : in std_logic;
17: edge : out std_logic
18: );
19: End positive_edge_detector;
20:
21: Architecture Arch of positive_edge_detector is
22: Signal q1,q2 : std_logic;
23:
24: begin
25:
26: edge <= q1 and (not q2) ; --positive edge detect
27:
28: process(clk,reset)
29: Begin
30: If(reset = '0')then
31: q1 <= '0';
32: q2 <= '0';
33: Elsif(clk'event and clk = '1')then
34: q1 <= asynchrodata_in;
35: q2 <= q1;
36: End if;
```

37:

38: **End proces**s;

39:

40: **End** Arch;

negative_edge_detector.vhd

```
1: ----------------------------------------------------------------------
2: --Function : Implements a Falling Edge Detector
3: --File : negative_edge_detector.vhd
4: --Date : 20/6/2006
5: --
6: --Version : 1.1
7: ----------------------------------------------------------------------
8: Library ieee;
9: use ieee.std_logic_1164.all;
10:
11: Entity negative_edge_detector is
12: port(
13: asynchrodata_in : in std_logic;
14: reset : in std_logic;
15: clk : in std_logic;
16: edge : out std_logic
17: );
18: End negative_edge_detector;
19:
20:
21: Architecture Arch of negative_edge_detector is
22: Signal q1,q2 : std_logic;
23: begin
24:
25: edge <= (not q1) and q2; -- Detect the Falling Edge
26:
27: process(clk,reset)
28: Begin
29: If(reset = '0')then
30: q1 <= '0';
31: q2 <= '0';
32: Elsif(clk'event and clk = '1')then
33: q1 <= asynchrodata_in;
34: q2 <= q1;
35: End if;
36:
```

```
37: End process;
38:
39: End Arch;
```

transmitter_clock.vhd

```
1: ----------------------------------------------------------------------
2: --Function : Generate One High Pulse every 16 Cycles of Baudx16
3: --File : transmitter_clock.vhd
4: --Date : 20/6/2006
5: --
6: --Version : 1.1
7: ----------------------------------------------------------------------
8:
9: Library ieee;
10: use ieee.std_logic_1164.all;
11: use ieee.std_logic_unsigned.all;
12:
13:
14: Entity transmitter_clock is
15: port(
16: clk : in std_logic;
17: reset : in std_logic;
18: Dlm_dll : in std_logic_vector(15 downto 0);
19: tx_clk_rise : out std_logic
20: );
21: End transmitter_clock;
22:
23:
24: Architecture Arch of transmitter_clock is
25:
26: component baud_generator
27: port(
28: clk : in std_logic;
29: reset : in std_logic;
30: Dlm_dll : in std_logic_vector(15 downto 0);
31: Baudx16 : out std_logic
32: );
33: End component;
34:
35: component negative_edge_detector
36: port(
```

```
37: asynchrodata_in : in std_logic;
38: reset : in std_logic;
39: clk : in std_logic;
40: edge : out std_logic
41: );
42: End component;
43:
44: Signal Cntr : std_logic_vector(3 downto 0);
45: Signal En_cntr : std_logic;
46:
47: Begin
48:
49: u1 : baud_generator
50: port map(clk => clk, reset => reset, Dlm_dll => Dlm_dll, Baudx16 => En_cntr);
51:
52: u2 : negative_edge_detector
53: port map( asynchrodata_in => cntr(3), reset => reset, clk => clk, edge => tx_clk_rise);
54:
55: process(reset,clk)
56: Begin
57: If(reset = '0')then
58: Cntr <= (others=>'0');
59: Elsif(clk'event and clk = '1')then
60: If(En_cntr = '1')then
61: Cntr <= Cntr + 1;
62: End if;
63: End if;
64: End process;
65:
66: End Arch;
```

receiver_clock.vhd

```
1: ----------------------------------------------------------------------
2: --Function : Generates the Baud Clock for Receiver
3: --File : receiver_clock.vhd
4: --Date : 20/6/2006
5:--
6: --Version : 1.1
7: ----------------------------------------------------------------------
8:
9: Library ieee;
10: use ieee.std_logic_1164.all;
11: use ieee.std_logic_unsigned.all;
12:
13: Entity receiver_clock is
14: port(
15: clk : in std_logic;
16: reset : in std_logic;
17: rxd : in std_logic;
18: Dlm_dll : in std_logic_vector(15 downto 0);
19: rx_clk_rise : out std_logic
20: );
21: End receiver_clock;
22:
23: Architecture Arch of receiver_clock is
24:
25: Signal Cntr : std_logic_vector(3 downto 0);
26: Signal reset_cntr : std_logic;
27: Signal En_cntr : std_logic;
28: Signal rx_clk_int : std_logic;
29:
30: component baud_generator
31: port(
32: clk : in std_logic;
33: reset : in std_logic;
34: dlm_dll : in std_logic_vector(15 downto 0);
35: Baudx16 : out std_logic
36: );
```

```
37: End component;
38:
39: component negative_edge_detector
40: port(
41: asynchrodata_in : in std_logic;
42: reset : in std_logic;
43: clk : in std_logic;
44: edge : out std_logic
45: );
46: End component;
47:
48: component positive_edge_detector
49: port(
50: asynchrodata_in : in std_logic;
51: reset : in std_logic;
52: clk : in std_logic;
53: edge : out std_logic
54: );
55: End component;
56:
57: Begin
58:
59: rx_clk_int <= cntr(3);
60:
61: u1 : baud_generator
62: port map( clk => clk, reset => reset, Dlm_dll => Dlm_dll, Baudx16 => En_cntr);
63:
64: u2 : negative_edge_detector
65: port map( asynchrodata_in => rxd, reset => reset, clk => clk, edge => reset_cntr);
66:
67: u3 : positive_edge_detector
68: port map( asynchrodata_in => rx_clk_int, reset => reset, clk => clk, edge => rx_clk_rise);
69:
70: process(reset,clk)
71: Begin
72: If(reset = '0')then
73: Cntr <= (others=>'0');
74: Elsif(clk'event and clk = '1')then
75: If(reset_cntr = '1')then
```

```
76: Cntr <= (others=>'0');
77: Elsif(En_cntr = '1')then
78: Cntr <= Cntr + 1;
79: End if;
80: End if;
81:
82: End process;
83:
84: End Arch;
```

transmitter.vhd

```
1: ----------------------------------------------------------------------
2: --Function : Implements the UART TRANSMITTER Module
3: --File : transmitter.vhd
4: --Date : 20/6/2006
5: --
6: --Version : 1.1
7: ----------------------------------------------------------------------
8:
9: Library ieee;
10: use ieee.std_logic_1164.all;
11: use ieee.std_logic_unsigned.all;
12:
13: Entity transmitter is
14: port(
15: data_bus : in std_logic_vector(7 downto 0); -- Data bus from CPU
16: PAR : in std_logic; -- '1' => Odd Parity
17: clk : in std_logic; -- UART Crystal i/p Clock
18: reset : in std_logic; -- Master Reset
19: wr : in std_logic; -- From CPU interface
20: Dlm_dll : in std_logic_vector(15 downto 0); -- Divisor Latch
21: Txd : out std_logic; -- o/p of Transmitter
22: TSR_empty : out std_logic;
23: Temt : out std_logic -- Transmitter / READY signal
24: );
25: End transmitter;
26:
27: Architecture Arch of transmitter is
28:
29: Component transmitter_clock
30: port(
31: clk : in std_logic;
32: reset : in std_logic;
33: Dlm_dll : in std_logic_vector(15 downto 0);
34: tx_clk_rise : out std_logic
35: );
36: End component;
```

```
37:
38: type state_type is (idle,receive,send);
39:
40: signal curr_state,nxt_state : state_type;
41: signal curr_thre,nxt_thre : std_logic; -- Current & Next Value of THRE
42: signal nxt_tsre : std_logic; -- Current & Next Value of TSRE
43: signal load_TSR,load_THR : std_logic; -- '1' => load TSR / THR
44: signal THR : std_logic_vector(7 downto 0); -- Tx Hold Register
45: signal TSR : std_logic_vector(8 downto 0); -- Tx Shift Register
46: signal bit_count : std_logic_vector(3 downto 0); -- count no of transmitted bits
47: signal clr : std_logic; -- '1' , => Clear Bit Counter
48: signal parity : std_logic;
49: signal bit_time : std_logic; -- Goes High After bit Duration
50:
51: Begin
52:
53: TSR_empty <= nxt_tsre;
54:
55: u1 : transmitter_clock
56: port map(clk => clk, reset => reset, Dlm_dll => Dlm_dll, tx_clk_rise => bit_time);
57:
58: process(curr_state,wr,curr_thre,bit_count,bit_time)
59: begin
60: nxt_thre <= '1';
61: nxt_tsre <= '1';
62: temt <= '0';
63: load_TSR <= '0';
64: load_THR <= '0';
65: clr <= '0';
66:
67: case curr_state is
68:
69: when IDLE =>
70: clr <= '1';
71: temt <= '1';
72: nxt_state <= IDLE;
73:
74: if(curr_thre ='1' and wr='0')then -- Load THR when Empty
75: nxt_state <= RECEIVE;
```

```
76: nxt_thre <= '0';
77: load_THR <= '1';
78: end if;
79:
80:
81: when RECEIVE =>
82: clr <= '1';
83: nxt_state <= SEND;
84: load_TSR <= '1';
85: nxt_tsre <= '0';
86:
87: when SEND =>
88: nxt_tsre <= '0';
89: nxt_state <= SEND;
90: if(bit_count="1100")then -- Set TSR Empty Flag
91: nxt_tsre <= '1';
92: nxt_state <= IDLE; -- To IDLE state
93: else
94: if(bit_time = '1')then
95: nxt_state <= SEND;
96: end if;
97: end if;
98:
99: when others=>
100: nxt_state <= idle;
101:
102: end case;
103:
104: end process;
105:
106: process(reset,clk)
107: begin
108: if(reset='0')then
109: curr_state <= IDLE;
110: curr_thre <= '1';
111: THR <= (others=>'0');
112: TSR <= "111111111";
113: parity <= PAR;
114: bit_count <= "0000";
```

```
115: Txd <= '1';
116: elsif(clk'event and clk = '1')then
117: curr_state <= nxt_state;
118: curr_thre <= nxt_thre;
119: If(bit_count = "0000")then
121: parity <= PAR;
122: End if;
123:
124: If(clr /= '1')then
125: If(bit_time = '1')then
126: bit_count <= bit_count + 1;
127: parity <= parity xor TSR(0);
128: TSR <= '1' & TSR(8 downto 1);
129: if(bit_count="1001")then
130: Txd <= parity;
131: else
132: Txd <= TSR(0);
133: end if;
134: end if;
135: Else
136: if(bit_time = '1')then
137: bit_count <= "0000";
138: End if;
8139: End if;
140:
141:
142: if(load_THR='1')then
143: THR <= data_bus;
144: elsif(load_TSR='1')then
145: TSR <= THR & '0';
146: End if;
147:
148: End if;
149:
150: end process;
151:
152: End Arch;
153:
```

receiver.vhd

```
1: ----------------------------------------------------------------------
2: --Function : Implements the UART RECEIVER Module
3: --File : receiver.vhd
4: --Date : 20/6/2006
5: --
6: --Version : 1.1
7: ----------------------------------------------------------------------
8:
9: library ieee;
10: use ieee.std_logic_1164.all;
11: use ieee.std_logic_unsigned.all;
12:
13:
14: entity receiver is
15: port (
16: clk : in std_logic;
17: reset : in std_logic;
18: rxd : in std_logic;
19: read : in std_logic;
20: par : in std_logic;
21: dlm_dll : in std_logic_vector(15 downto 0);
22: data_bus : out std_logic_vector(7 downto 0);
23: rx_ready : out std_logic;
24: framing_err : out std_logic;
25: parity_err : out std_logic;
26: brake_err : out std_logic
27: );
28:
29: end receiver;
30:
31:
32: architecture Arch of receiver is
33:
34: type state_type is (idle, check_start,load1,loadRBR,chk_err);
35: signal current_state : state_type;
36: signal next_state : state_type;
```

```
37: signal RSR : std_logic_vector(7 downto 0);
38: signal RBR : std_logic_vector(7 downto 0);
39: signal bit_counter : std_logic_vector(3 downto 0);
40: signal shift_bit : std_logic;
41: signal shift_byte : std_logic;
42: signal parity : std_logic;
43: signal rx_clk_rse : std_logic;
44: signal rx_clk_rise : std_logic;
45: signal edge : std_logic;
46: signal chk_brake : std_logic;
47: signal err_clr : std_logic;
48: signal data_ready : std_logic;
49: signal FE_int : std_logic;
50: signal PE_int : std_logic;
51: signal incr_bit : std_logic;
52:
53: component Receiver_Clock
54: port(
55: clk : in std_logic;
56: reset : in std_logic;
57: rxd : in std_logic;
58: dlm_dll : in std_logic_vector(15 downto 0);
59: rx_clk_rise : out std_logic
60: );
61: end component;
62:
63: component negative_Edge_detector
64: port (
65: asynchrodata_in : in std_logic;
66: reset : in std_logic;
67: clk : in std_logic;
68: edge : out std_logic
69: );
70: end component;
71:
72: begin
73:
74: u1 : receiver_clock
75: port map(clk => clk, reset => reset, rxd => rxd, dlm_dll => dlm_dll,
```

```
76: rx_clk_rise =>rx_clk_rse);
77:
78: u2 : negative_edge_detector
79: port map( asynchrodata_in => rxd, reset => reset, clk => clk, edge => edge);
80:
81: data_bus <= RBR when read = '0' else "ZZZZZZZZ";
82:
83: process(clk,reset)
84: begin
85: if(reset = '0')then
86: current_state <= idle;
87: RSR <= (others => '0');
88: RBR <= (others => '0');
89: parity_err <= '0';
90: framing_err <= '0';
91: bit_counter <= (others => '0');
92: parity <= par;
93: chk_brake <= '0';
94: brake_err <= '0';
95: rx_ready <= '0';
96: rx_clk_rise <= '0';
97:
98: elsif(clk'event and clk = '1')then
99: current_state <= next_state;
100: rx_clk_rise <= rx_clk_rse;
101: if(rx_clk_rse = '1')then
102: chk_brake <= chk_brake or rxd;
103: if(incr_bit = '1')then
104: bit_counter <= bit_counter + 1;
105: end if;
106:
107: if(shift_bit = '1')then
108: RSR <=rxd & RSR(7 downto 1);
109: end if;
110:
111: if(bit_counter <= "1000")then
112: parity <= parity xor rxd;
113: elsif(bit_counter = "1100")then
114: bit_counter <= "0000";
```

```
115: RSR <= (others => '0');
116: end if;
117:
118: if(bit_counter = "1000")then
119: parity_err <= '0';
120: framing_err<= '0';
121: brake_err <= '0';
122: elsif(bit_counter = "1001")then
123: if(rxd /= parity)then
124: Parity_err <= '1';
125: end if;
126: elsif(bit_counter = "1010")then
127: if(rxd = '0')then
128: Framing_err <= '1';
129: end if;
130: elsif(bit_counter = "1011")then
131: if(chk_brake = '0')then
132: brake_err <= '1';
133: end if;
134: end if;
135: end if;
136:
137: if(shift_byte = '1')then
138: RBR <= RSR(7 downto 0);
139: end if;
140:
141: if(err_clr = '1')then
142: rx_ready <= '0';
143: parity <= par;
144: chk_brake <= '0';
145: end if;
146:
147: end if;
148: end process;
149:
150: process(current_state,rxd,edge,rx_clk_rise,bit_counter,parity)
151: begin
152: err_clr <= '0';
153: FE_int <= '0';
```

```
154: PE_int <= '0';
155: incr_bit<= '1';
156:
157: case current_state is
158:
159: when idle =>
160: shift_bit <= '0';
161: shift_byte <= '0';
162: data_ready <= '0';
163: incr_bit <= '0';
164:
165: if(edge = '1')then
166: next_state <= check_start;
167: else
168: next_state <= idle;
169: end if;
170:
171: when check_start =>
172: shift_bit <= '0';
173: shift_byte <= '0';
174: data_ready <= '0';
175: err_clr <= '1';
176:
177: if(rxd = '0')then
178: if(rx_clk_rise = '1')then
179: next_state <= load1;
180: else
181: next_state <= check_start;
182: end if;
183: else
184: next_state <= idle;
185: end if;
186:
187: when load1 =>
188: shift_bit <= '1';
189: shift_byte <= '0';
190: data_ready <= '0';
191: next_state <= load1;
192:
```

```
193: if(rx_clk_rise = '1')then
194: if(bit_counter <= "1000")then
195: next_state <= load1;
196: else
197: next_state <= loadRBR;
198: shift_bit <= '0';
199: end if;
200: end if;
201:
202: when loadRBR =>
203: shift_bit <= '0';
204: shift_byte <= '1';
205: data_ready <= '1';
206: next_state <= chk_err;
207:
208: when chk_err =>
209: shift_bit <= '0';
210: shift_byte <= '0';
211: data_ready <= '0';
212: next_state <= chk_err;
213:
214: if(rx_clk_rise = '1')then
215: if(bit_counter = "1100")then
216: incr_bit <= '0';
217: next_state <= idle;
218: else
219: next_state <= chk_err;
220: end if;
221: end if;
222: when others =>
223: next_state <= idle;
224: shift_bit <= '0';
225: shift_byte <= '0';
226: end case;
227: end process;
228: end Arch;
```

uart.vhd

```
1: ----------------------------------------------------------------------
2: --Function : Top Module of UART
3: --File : uart.vhd
4: --Date : 20/6/2006
5: --
6: --Version : 1.1
7: ----------------------------------------------------------------------
8:
9: Library ieee;
10: use ieee.std_logic_1164.all;
11: use work.uart_pkg.all;
12:
13: Entity uart is
14: Port(
15: clk : in std_logic;
16: reset : in std_logic;
17: RXD : in std_logic;
18: Rd : in std_logic;
19: Wr : in std_logic;
20: PAR : in std_logic;
21: Data : inout std_logic_vector(7 downto 0);
22: Txd : out std_logic;
23: TSR_empty : out std_logic;
24: Temt : out std_logic;
25: rx_ready : out std_logic;
26: framing_err : out std_logic;
27: parity_err : out std_logic;
28: brake_err : out std_logic;
29: BAUDOUT : out std_logic
30: );
31: End uart;
32:
33: Architecture Arch of uart is
34:
35: component receiver
36: port (
```

```
37: clk : in std_logic;
38: reset : in std_logic;
39: rxd : in std_logic;
40: read : in std_logic;
41: par : in std_logic;
42: dlm_dll : in std_logic_vector(15 downto 0);
43: data_bus : out std_logic_vector(7 downto 0);
44: rx_ready : out std_logic;
45: framing_err : out std_logic;
46: parity_err : out std_logic;
47: brake_err : out std_logic
48: );
49: end component;
50:
51: Component transmitter
52: port(
53: data_bus : in std_logic_vector(7 downto 0);
54: PAR : in std_logic;
55: clk : in std_logic;
56: reset : in std_logic;
57: wr : in std_logic;
58: Dlm_dll : in std_logic_vector(15 downto 0);
59: Txd : out std_logic;
60: TSR_empty : out std_logic;
61: Temt : out std_logic
62: );
63: End component;
64:
65: component baud_generator
66: port(
67: clk : in std_logic;
68: reset : in std_logic;
69: Dlm_dll : in std_logic_vector(15 downto 0);
70: Baudx16 : out std_logic
71: );
72: end component;
73:
74: Signal Divisor_val : std_logic_vector(15 downto 0);
75:
```

```
76: Begin
77:
78: Divisor_val <= DLM_DLL;
79:
80: u1 : transmitter
81: port map( data_bus => data, PAR => par, clk => clk, reset => reset, wr =>wr,
82: Dlm_dll => Divisor_val, Txd => Txd, TSR_empty =>TSR_empty, Temt => Temt);
83:
84: u2 : receiver
85: port map( clk => clk, reset => reset, rxd => Rxd, read => rd, par => par,
86: dlm_dll => Divisor_val, data_bus => data, rx_ready => rx_ready, framing_err => framing_err,
87: parity_err => parity_err, brake_err => brake_err);
88:
89: u3 : baud_generator
90: port map( clk => clk, reset => reset, Dlm_dll => Divisor_val, Baudx16 => BAUDOUT);
91:
92: End Arch;
```

uarttb.vhd

```
1: ----------------------------------------------------------------------
2: --Function : Test bench to check the working of receiver
3: --File : uarttb.vhd
4: --Date : 20/6/2006
5: --
6: --Version : 1.1
7: ----------------------------------------------------------------------
8: Library ieee;
9: use ieee.std_logic_1164.all;
10: use work.uart_pkg.all;
11:
12: Entity uarttb is port
13: ( clk : in std_logic;
14: reset : in std_logic;
15: indata : inout std_logic_vector(7 downto 0);
16: outdata : inout std_logic_vector(7 downto 0);
17: temtt : out std_logic;
18: BAUDOUTt : out std_logic;
19: BAUDOUTr : out std_logic;
20: TSR_emptyt : out std_logic;
21: TSR_emptyr : out std_logic;
22: framing_err : out std_logic;
23: parity_err : out std_logic;
24: brake_err : out std_logic;
25: rx_ready : out std_logic
26: );
27: End uarttb;
28:
29: Architecture Arch of uarttb is
30:
31: component uart
32: Port(
33: clk : in std_logic; reset : in std_logic; rxd : in std_logic; Rd : in std_logic;
34: Wr : in std_logic; PAR : in std_logic; Data : inout std_logic_vector(7 downto 0);
35: Txd : out std_logic; TSR_empty : out std_logic; Temt : out std_logic;
36: rx_ready : out std_logic; framing_err : out std_logic; parity_err : out std_logic;
```

```
37: brake_err : out std_logic; BAUDOUT : out std_logic);
38: End Componen t;
39:
40: signal one,zero,txdd: std_logic;
41: Begin
42: one<='1';
43: zero<='0';
44:
45: x1 : uart port map(clk=>clk, reset=>reset, rxd=>one, Rd=>one, Wr=>zero, PAR=>one,
46: Data=>indata, TSR_empty=>TSR_emptyt,
47: Txd=>txdd,Temt=>temtt,BAUDOUT=>BAUDOUTt);
48: x2 : uart port map(clk=>clk, reset=>reset, rxd=>txdd, Rd=>zero, Wr=>one, PAR=>one,
49: Data=>outdata,TSR_empty=>TSR_emptyr,
50: rx_ready=>rx_ready,framing_err=>framing_err,
51: parity_err=>parity_err, brake_err=>brake_err,BAUDOUT=>BAUDOUTr);
52: End Arch;
```

Synthesis Reports

Translation Report

Reading NGO file "d:/vijaypal/projects/projects/uart/uart/xproj/ver1/uart.ngo"

...

Reading component libraries for design expansion...

Annotating constraints to design from file "uart.ucf" ...

Checking timing specifications ...

Checking expanded design ...

NGDBUILD Design Results Summary:

Number of errors: 0

Number of warnings: 0

Writing NGD file "uart.ngd" ...

Writing NGDBUILD log file "uart.bld"...

Map Report

Xilinx Mapping Report File for Design 'uart'

Design Information

Command Line : map -p xcv50-6-cs144 -o map.ncd uart.ngd uart.pcf

Target Device : xv50

Target Package : cs144

Target Speed : -6

Mapper Version : virtex -- C.16

Mapped Date : Tue Jun 20 03:20:29 2006

Design Summary

Number of errors: 0

Number of warnings: 1

Number of Slices: 137 out of 768 17%

Slice Flip Flops: 125

4 input LUTs: 219

Number of Slices containing

unrelated logic: 0 out of 137 0%

Number of bonded IOBs: 21 out of 94 22%

Number of GCLKs: 1 out of 4 25%

Number of GCLKIOBs: 1 out of 4 25%

Total equivalent gate count for design: 2,338

Additional JTAG gate count for IOBs: 1,056

Table of Contents

Section 1 - Errors

Section 2 - Warnings

WARNING:basmapmgr:4 - All of the external outputs in this design are using slew rate limited output drivers. The delay on speed critical outputs can be dramatically reduced by designating them as fast outputs in the schematic.

Section 3 - Design Attributes

Section 4 - Removed Logic Summary

---

2 block(s) optimized away

Section 5 - Removed Logic

Optimized Block(s):

TYPE	BLOCK
GND	C1146
FDCE	u2/rx_ready_reg

To enable printing of redundant blocks removed and signals merged, set the detailed map report option and rerun map.

Section 6 - Added Logic

Section 7 - Expanded Logic

To enable this section, set the detailed map report option and rerun map.

Section 8 - Signal Cross-Reference

To enable this section, set the detailed map report option and rerun map.

Section 9 - Symbol Cross-Reference

To enable this section, set the detailed map report option and rerun map.

Section 10 - IOB Properties

---

clk (GCLKIOB)

BAUDOUT (IOB) : SLEW=SLOW DRIVE=12

Data<0> (IOB) : SLEW=SLOW DRIVE=12

Data<1> (IOB) : SLEW=SLOW DRIVE=12

Data<2> (IOB) : SLEW=SLOW DRIVE=12

Data<3> (IOB) : SLEW=SLOW DRIVE=12

Data<4> (IOB) : SLEW=SLOW DRIVE=12
Data<5> (IOB) : SLEW=SLOW DRIVE=12
Data<6> (IOB) : SLEW=SLOW DRIVE=12
Data<7> (IOB) : SLEW=SLOW DRIVE=12
TSR_empty (IOB) : SLEW=SLOW DRIVE=12
Temt (IOB) : SLEW=SLOW DRIVE=12
Txd (IOB) : SLEW=SLOW DRIVE=12
brake_err (IOB) : SLEW=SLOW DRIVE=12
framing_err (IOB) : SLEW=SLOW DRIVE=12
parity_err (IOB) : SLEW=SLOW DRIVE=12
rx_ready (IOB) : SLEW=SLOW DRIVE=12

Place and Route Report

Device utilization summary:

Number of External GCLKIOBs	1 out of 4	25%
Number of External IOBs	21 out of 94	22%
Number of SLICEs	137 out of 768	17%
Number of GCLKs	1 out of 4	25%

Overall effort level (-ol): 2 (set by user)
Placer effort level (-pl): 2 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl): 2 (set by user)

Starting initial Placement phase. REAL time: 3 secs
Finished initial Placement phase. REAL time: 3 secs
Starting the placer. REAL time: 3 secs
Placement pass 1 .....

Placer score = 17715

Optimizing ...

Placer score = 13920

Starting IO Improvement. REAL time: 4 secs

Placer score = 11985

Finished IO Improvement. REAL time: 4 secs

Placer completed in real time: 4 secs

Writing design to file "uart.ncd".

Total REAL time to Placer completion: 4 secs

Total CPU time to Placer completion: 3 secs

Pad Report

Pinout by Pin Name:

Pin Name	Direction	Pin Number
BAUDOUT	OUTPUT	A4
Data<0>	BIDIR	H4
Data<1>	BIDIR	N5
Data<2>	BIDIR	N8
Data<3>	BIDIR	L8
Data<4>	BIDIR	L6
Data<5>	BIDIR	M6
Data<6>	BIDIR	M8
Data<7>	BIDIR	J3
PAR	INPUT	J4
Rd	INPUT	D9
TSR_empty	OUTPUT	H3

Temt	OUTPUT	J2
Txd	OUTPUT	H1
Wr	INPUT	K5
brake_err	OUTPUT	K6
clk	INPUT	M7
framing_err	OUTPUT	L4
parity_err	OUTPUT	N4
reset	INPUT	B8
rx_ready (DOUT_BUSY)	OUTPUT	C11
rxd	INPUT	M4

Dedicated or Special Pin Name	Pin Number
CCLK	B13
CS	D10
D1	E10
D2	E12
D3	F11
D4	H12
D5	J13
D6	J11
D7	K10
DIN_D0	C12
DONE	M12
DOUT_BUSY (rx_ready)	C11
GND	B9
GND	A1
GND	N12
GND	E11
GND	G10
GND	B11
GND	L9
GND	J12

GND	L5
GND	L3
GND	J1
GND	F1
GND	C7
GND	D5
GND	E4
GND	L7
INIT	L13
IRDY	F2
IRDY	F13
M0	M1
M1	L2
M2	N2
PROGRAM	L12
TCK	C3
TDI	A11
TDO	A12
TMS	B1
TRDY	G13
TRDY	G1
VCCINT	G3
VCCINT	M9
VCCINT	N6
VCCINT	M5
VCCINT	G12
VCCINT	A9
VCCINT	B6
VCCINT	C5
VCCO	B2
VCCO	M13
VCCO	N13
VCCO	D7

VCCO	G11
VCCO	G2
VCCO	A13
VCCO	N1
VCCO	B12
VCCO	N7
VCCO	A2
VCCO	M2
WRITE	C10

Pinout by Pin Number:

Pin Number	Pin Name	Direction	Constraint
A1	(GND)		
A2	(VCCO)		
A3	---	UNUSED	
A4	BAUDOUT	OUTPUT	
A5	---	UNUSED	
A6	---	UNUSED	
A7	---	UNUSED	
A8	---	UNUSED	
A9	(VCCINT)		
A10	---	UNUSED	
A11	(TDI)		
A12	(TDO)		
A13	(VCCO)		
B1	(TMS)		
B2	(VCCO)		
B3	---	UNUSED	
B4	---	UNUSED	
B5	---	UNUSED	

B6	(VCCINT)		
B7	---	UNUSED	
B8	reset	INPUT	
B9	(GND)		
B10	---	UNUSED	
B11	(GND)		
B12	(VCCO)		
B13	(CCLK)		
C1	---	UNUSED	
C2	---	UNUSED	
C3	(TCK)		
C4	---	UNUSED	
C5	(VCCINT)		
C6	---	UNUSED	
C7	(GND)		
C8	---	UNUSED	
C9	---	UNUSED	
C10	(WRITE)		
C11	rx_ready (DOUT_BUSY)	OUTPUT	
C12	(DIN_D0)		
C13	---	UNUSED	
D1	---	UNUSED	
D2	---	UNUSED	
D3	---	UNUSED	
D4	---	UNUSED	
D5	(GND)		
D6	---	UNUSED	
D7	(VCCO)		
D8	---	UNUSED	
D9	Rd	INPUT	
D10	(CS)		
D11	---	UNUSED	
D12	---	UNUSED	

D13	---	UNUSED	
E1	---	UNUSED	
E2	---	UNUSED	
E3	---	UNUSED	
E4	(GND)		
E10	(D1)		
E11	(GND)		
E12	(D2)		
E13	---	UNUSED	
F1	(GND)		
F2	(IRDY)		
F3	---	UNUSED	
F4	---	UNUSED	
F10	---	UNUSED	
F11	(D3)		
F12	---	UNUSED	
F13	(IRDY)		
G1	(TRDY)		
G2	(VCCO)		
G3	(VCCINT)		
G4	---	UNUSED	
G10	(GND)		
G11	(VCCO)		
G12	(VCCINT)		
G13	(TRDY)		
H1	Txd	OUTPUT	
H2	---	UNUSED	
H3	TSR_empty	OUTPUT	
H4	Data<0>	BIDIR	
H10	---	UNUSED	
H11	---	UNUSED	
H12	(D4)		
H13	---	UNUSED	

J1	(GND)			
J2	Temt	OUTPUT		
J3	Data<7>	BIDIR		
J4	PAR	INPUT		
J10	---	UNUSED		
J11	(D6)			
J12	(GND)			
J13	(D5)			
K1	---	UNUSED		
K2	---	UNUSED		
K3	---	UNUSED		
K4	---	UNUSED		
K5	Wr	INPUT		
K6	brake_err	OUTPUT		
K7	---	UNUSED		
K8	---	UNUSED		
K9	---	UNUSED		
K10	(D7)			
K11	---	UNUSED		
K12	---	UNUSED		
K13	---	UNUSED		
L1	---	UNUSED		
L2	(M1)			
L3	(GND)			
L4	framing_err	OUTPUT		
L5	(GND)			
L6	Data<4>	BIDIR		
L7	(GND)			
L8	Data<3>	BIDIR		
L9	(GND)			
L10	---	UNUSED		
L11	---	UNUSED		
L12	(PROGRAM)			

L13	(INIT)			
M1	(M0)			
M2	(VCCO)			
M3	---	UNUSED		
M4	rxd	INPUT		
M5	(VCCINT)			
M6	Data<5>	BIDIR		
M7	clk	INPUT		
M8	Data<6>	BIDIR		
M9	(VCCINT)			
M10	---	UNUSED		
M11	---	UNUSED		
M12	(DONE)			
M13	(VCCO)			
N1	(VCCO)			
N2	(M2)			
N3	---	UNUSED		
N4	parity_err	OUTPUT		
N5	Data<1>	BIDIR		
N6	(VCCINT)			
N7	(VCCO)			
N8	Data<2>	BIDIR		
N9	---	UNUSED		
N10	---	UNUSED		
N11	---	UNUSED		
N12	(GND)			
N13	(VCCO)			

Post Layout Timing Report

```
--------------------------------------------------------------------------------
Xilinx TRACE, Version C.16
Copyright (c) 1995-1999 Xilinx, Inc.  All rights reserved.

Design file:              uart.ncd
Physical constraint file: uart.pcf
Device,speed:             xcv50,-6 (C 1.1.2.8 Advanced)
Report level:             error report
--------------------------------------------------------------------------------

WARNING:Timing - No timing constraints found, doing default enumeration.

================================================================================
Timing constraint: Default period analysis
 2248 items analyzed, 0 timing errors detected.
 Minimum period is   8.307ns.
 Maximum delay is  13.836ns.
--------------------------------------------------------------------------------

================================================================================
Timing constraint: Default net enumeration
 269 items analyzed, 0 timing errors detected.
 Maximum net delay is   6.689ns.
--------------------------------------------------------------------------------


All constraints were met.
```

Data Sheet report:

All values displayed in nanoseconds (ns)

Setup/Hold to clock clk

Source Pad	Setup to clk (edge)	Hold to clk (edge)
Data<7>	1.860(R)	0.000(R)
Data<4>	1.747(R)	0.000(R)
Data<1>	1.955(R)	0.000(R)
Data<5>	1.791(R)	0.000(R)
Data<2>	1.975(R)	0.000(R)
Data<6>	1.778(R)	0.000(R)
Data<3>	2.217(R)	0.000(R)
Data<0>	1.636(R)	0.000(R)
rxd	4.006(R)	0.000(R)
Wr	5.657(R)	0.000(R)
PAR	2.266(R)	0.000(R)

Clock clk to Pad

Destination Pad	to PAD clk (edge)
Temt	10.626(R)
Data<7>	11.175(R)
brake_err	10.310(R)
Data<4>	10.646(R)
parity_err	10.187(R)
framing_err	10.594(R)

Data<1> | 10.827(R)|
TSR_empty | 12.715(R)|
BAUDOUT | 10.211(R)|
Data<5> | 10.383(R)|
Data<2> | 10.402(R)|
Data<6> | 10.363(R)|
Txd | 10.568(R)|
Data<3> | 10.264(R)|
Data<0> | 10.615(R)|

Clock to Setup on destination clock clk

	Src/Dest	Src/Dest	Src/Dest	Src/Dest
Source Clock	Rise/Rise	Fall/Rise	Rise/Fall	Fall/Fall
clk	8.299			

Pad to Pad

Source Pad	Destination Pad	Delay
Rd	Data<7>	10.589
Rd	Data<4>	13.320
Rd	Data<1>	12.389
Rd	Data<5>	13.383
Rd	Data<2>	13.344
Rd	Data<6>	13.774
Rd	Data<3>	13.836
Rd	Data<0>	10.589

Timing summary:

Timing errors: 0 Score: 0

Constraints cover 2248 paths, 269 nets, and 1034 connections (100.0% coverage)

Design statistics:

Minimum period: 8.307ns (Maximum frequency: 120.380MHz)

Maximum combinational path delay: 13.836ns

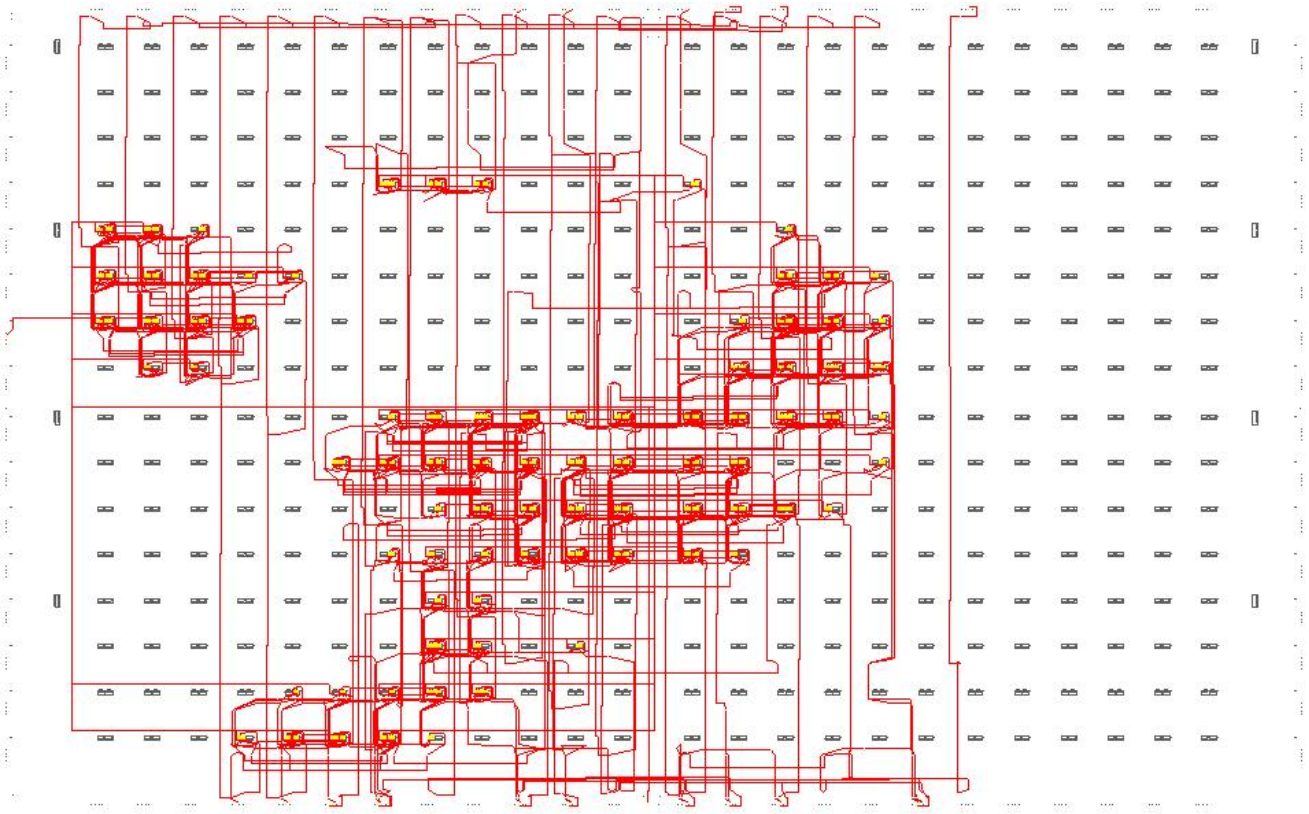
Maximum net delay: 6.689ns

Analysis completed Tue Jun 20 03:20:46 2006

Placement and Routing in FPGA

The FPGA Editor is a graphical application for displaying and configuring Field Programmable Gate Arrays (FPGAs). You can use this application to place and route critical components before running the automatic place and route tools on your design. You can also use the FPGA Editor to manually finish placement and routing if the routing program does not completely route your design.

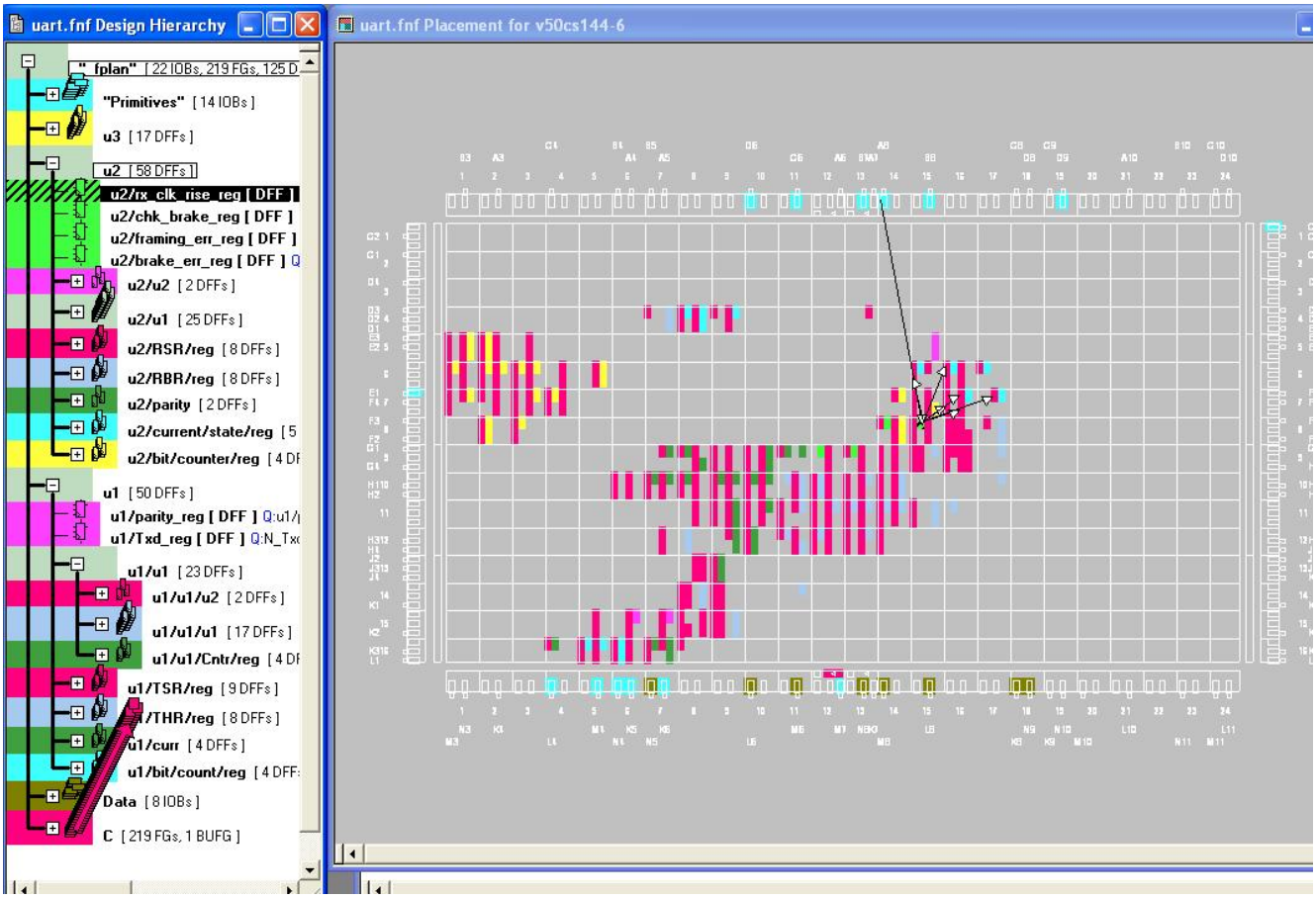
The FPGA Editor requires a Native Circuit Description (NCD) file. This file contains the logic of your design mapped to components (such as CLBs and IOBs). In addition, the FPGA Editor reads from and writes to a Physical Constraints File (PCF).



Placement and Routing in FPGA(V50CS144)

Floor Planner

The Floorplanner is a graphical placement tool that allows you to manually or automatically place logic from a mapped design (NCD file) into a floorplan of the selected FPGA.



FloorPlan of FPGA(V50CS144)

Conclusion

This project was very enriching, from the beginning to the end, right through the tests with the softwares and finally the work at the laboratory with sophisticated tools.

Despite the tests couldn’t take place with the logic analyzer, this brought us some new knowledge with new professional and interesting tools such as Xilinx.

We achieved an efficient design which is working in simulation mode. Any improvements in this project, which can be thought about, could be applied in future.

References

Bhaskar, J. *VHDL Primer*, 3rd ed., Pearson Education Asia, 2001.

Perry, Douglas L., *VHDL*, 3rd ed., Tata McGraw Hill, 2001.

Navabi, Zainalabedin, *VHDL Analysis and Modelling of Digital Systems*, 2nd ed., McGraw Hill, 2001.

Gaonkar, Ramesh S., *Microprocessor Architecture, Programming, and Application*, 2nd ed. New Age International Publishers, 1996.

OKI Semiconductors, Universal Synchronous Asynchronous Receiver Transmitter, January 1998.

HARRIS Semiconductors, 82C50A CMOS Asynchronous Communication Element, March 1997.