

Department of Electronic & Telecommunication Engineering
University of Moratuwa



EN2111 - Electronic Circuit Design

UART Implementation in FPGA

Group 09

Name: Alahakoon U.M.Y.B (210027C)

Ahamath M.J (210024N)

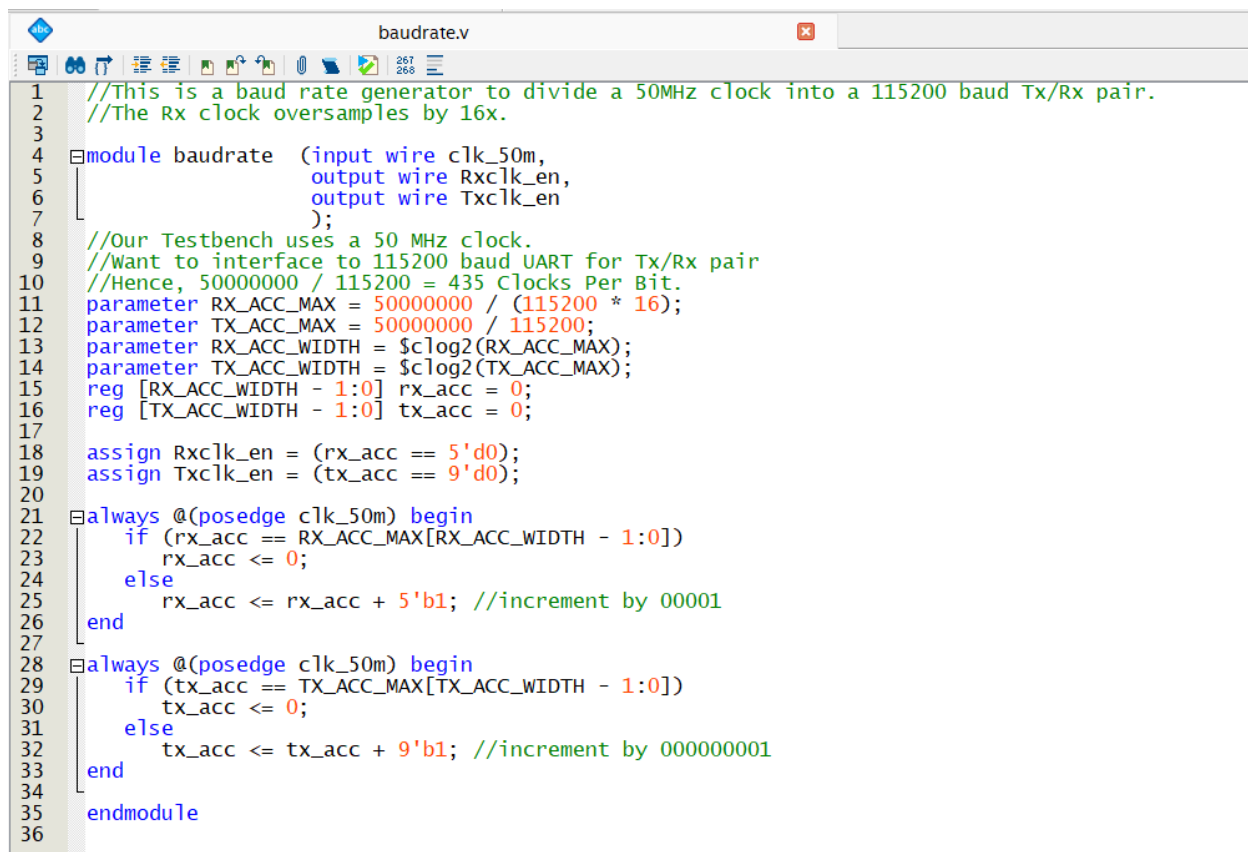
Date: 08/05/2024

Introduction

This report focusses on implementation of a UART transceiver and a receiver on the EP4CE22F17C6 FPGA board, consisting of three main parts: the simulation timing diagram, the test benches that goes with it, and the RTL code. UART, being an asynchronous communication protocol, relies on start and stop bits to delineate data packets, without the need for a clock signal. The RTL code defines the UART module's operation in detail. It allows devices to communicate serially by defining data packets with start and stop bits. Simultaneously, a carefully designed testbench enables comprehensive verification of the module's performance in many environments. Furthermore, by providing essential insights into the temporal dynamics of the exchange of information, the timing diagram captured during simulation directs optimization efforts toward smooth bidirectional communication. After being programmed into the FPGA, the Tx and Rx pins enable bidirectional serial communication with external devices, providing an environment for smooth data transfer.

01. RTL code for UART

- Baud Rate Code



```
1 //This is a baud rate generator to divide a 50MHz clock into a 115200 baud Tx/Rx pair.
2 //The Rx clock oversamples by 16x.
3
4 module baudrate (input wire clk_50m,
5                 output wire Rxcclk_en,
6                 output wire Txcclk_en
7                 );
8 //Our Testbench uses a 50 MHz clock.
9 //Want to interface to 115200 baud UART for Tx/Rx pair
10 //Hence, 50000000 / 115200 = 435 Clocks Per Bit.
11 parameter RX_ACC_MAX = 50000000 / (115200 * 16);
12 parameter TX_ACC_MAX = 50000000 / 115200;
13 parameter RX_ACC_WIDTH = $clog2(RX_ACC_MAX);
14 parameter TX_ACC_WIDTH = $clog2(TX_ACC_MAX);
15 reg [RX_ACC_WIDTH - 1:0] rx_acc = 0;
16 reg [TX_ACC_WIDTH - 1:0] tx_acc = 0;
17
18 assign Rxcclk_en = (rx_acc == 5'd0);
19 assign Txcclk_en = (tx_acc == 9'd0);
20
21 always @(posedge clk_50m) begin
22     if (rx_acc == RX_ACC_MAX[RX_ACC_WIDTH - 1:0])
23         rx_acc <= 0;
24     else
25         rx_acc <= rx_acc + 5'b1; //increment by 00001
26 end
27
28 always @(posedge clk_50m) begin
29     if (tx_acc == TX_ACC_MAX[TX_ACC_WIDTH - 1:0])
30         tx_acc <= 0;
31     else
32         tx_acc <= tx_acc + 9'b1; //increment by 000000001
33 end
34
35 endmodule
36
```

• Transmitter Code

Date: May 07, 2024

transmitter.v

Project: UART

```

1  module transmitter(  input wire [7:0] data_in, //input data as an 8-bit register/vector
2                      input wire wr_en, //enable wire to start
3                      input wire clk_50m,
4                      input wire clken, //clock signal for the transmitter
5                      output reg TX, //a single 1-bit register variable to hold transmitting
6
7                      bit
8                      output wire Tx_busy //transmitter is busy signal
9                      );
10
11  initial begin
12      Tx = 1'b1; //initialize Tx = 1 to begin the transmission
13  end
14  //Define the 4 states using 00,01,10,11 signals
15  parameter TX_STATE_IDLE = 2'b00;
16  parameter TX_STATE_START = 2'b01;
17  parameter TX_STATE_DATA = 2'b10;
18  parameter TX_STATE_STOP = 2'b11;
19
20  reg [7:0] data = 8'h00; //set an 8-bit register/vector as data,initially equal to 00000000
21  reg [2:0] bit_pos = 3'h0; //bit position is a 3-bit register/vector, initially equal to 000
22  reg [1:0] state = TX_STATE_IDLE; //state is a 2 bit register/vector,initially equal to 00
23
24  assign Tx_busy = (state != TX_STATE_IDLE); //We assign the BUSY signal when the transmitter
25  is not idle
26
27  always @(posedge clk_50m) begin
28      case (state) //Let us consider the 4 states of the transmitter
29      TX_STATE_IDLE: begin //We define the conditions for idle or NOT-BUSY state
30          if (wr_en) begin
31              state <= TX_STATE_START; //assign the start signal to state
32              data <= data_in; //we assign input data vector to the current data
33              bit_pos <= 3'h0; //we assign the bit position to zero
34          end
35      end
36      TX_STATE_START: begin //we define the conditions for the transmission start state
37          if (clken) begin
38              TX <= 1'b0; //set Tx = 0 indicating transmission has started
39              state <= TX_STATE_DATA;
40          end
41      end
42      TX_STATE_DATA: begin
43          if (clken) begin
44              if (bit_pos == 3'h7) //we keep assigning Tx with the data until all bits have been
45              transmitted from 0 to 7
46                  state <= TX_STATE_STOP; // when bit position has finally reached 7, assign
47                  state to stop transmission
48              else
49                  bit_pos <= bit_pos + 3'h1; //increment the bit position by 001
50                  TX <= data[bit_pos]; //Set TX to the data value of the bit position ranging from 0-7
51              end
52          end
53      end
54      TX_STATE_STOP: begin
55          if (clken) begin
56              TX <= 1'b1; //set Tx = 1 after transmission has ended
57              state <= TX_STATE_IDLE; //Move to IDLE state once a transmission has been completed
58          end
59      end
60      default: begin
61          TX <= 1'b1; // always begin with Tx = 1 and state assigned to IDLE
62          state <= TX_STATE_IDLE;
63      end
64  endcase
65  end
66
67  endmodule
68
69

```

• Receiver Code

```

Date: May 07, 2024
receiver.v
Project: UART

1 module receiver (input wire Rx,
2   output reg ready, // default 1 bit reg
3   input wire ready_clr,
4   input wire clk_50m,
5   input wire clken,
6   input wire RX_en,
7   input wire reset,
8   output reg [7:0] data // 8 bit register
9 );
10 initial begin
11   ready = 1'b0; // initialize ready = 0
12   data = 8'b0; // initialize data as 00000000
13 end
14 // Define the 4 states using 00,01,10 signals
15 parameter RX_STATE_START = 2'b00;
16 parameter RX_STATE_DATA = 2'b01;
17 parameter RX_STATE_STOP = 2'b10;
18
19 reg [1:0] state = RX_STATE_START; // state is a 2-bit register/vector, initially equal to 00
20 reg [3:0] sample = 0; // This is a 4-bit register
21 reg [3:0] bit_pos = 0; // bit position is a 4-bit register/vector, initially equal to 000
22 reg [7:0] scratch = 8'b0; // An 8-bit register assigned to 00000000
23
24 always @(posedge clk_50m) begin
25   if (reset) begin
26     data = 0;
27     scratch = 0;
28     end
29   if (RX_en) begin
30     if (ready_clr)
31       ready <= 1'b0; // This resets ready to 0
32     if (clken) begin
33       case (state) // Let us consider the 3 states of the receiver
34         RX_STATE_START: begin // We define conditions for starting the receiver
35           if (Rx || sample != 0) // start counting from the first low sample
36             sample <= sample + 4'b1; // increment by 0001
37           if (sample == 15) begin // once a full bit has been sampled
38             state <= RX_STATE_DATA; // start collecting data bits
39             bit_pos <= 0;
40             scratch <= 0;
41             end
42           RX_STATE_DATA: begin // We define conditions for starting the data collecting
43             sample <= sample + 4'b1; // increment by 0001
44             if (sample == 4'b0) begin // we keep assigning Rx data until all bits have 01
45               scratch[bit_pos[2:0]] <= Rx;
46               bit_pos <= bit_pos + 4'b1; // increment by 0001
47             end
48             if (bit_pos == 8 && sample == 15) // when a full bit has been sampled and
49               state <= RX_STATE_STOP; // bit position has finally reached 7, assign state
50             to stop
51             RX_STATE_STOP: begin
52               // Our baud clock may not be running at exactly the
53               // same rate as the transmitter. If we thing that
54               // we're at least half way into the stop bit, allow
55               // transition into handling the next start bit.
56               if (sample == 15 || (sample >= 8 && !Rx)) begin
57                 state <= RX_STATE_START;
58                 data <= scratch;
59                 ready <= 1'b1;
60                 sample <= 0;
61               end
62             else begin
63               sample <= sample + 4'b1;
64             end
65           end
66         end
67       end
68     end
69   end
70 end
71

```

```

Date: May 07, 2024
receiver.v
Project: UART

72 end
73 default: begin
74   state <= RX_STATE_START; // always begin with state assigned to START
75 end
76 endcase
77 end
78 end
79 end
80 endmodule
81
82

```

• UART Code

Project Navigator

- transmitter_tb.v
- uart.v
- transmitter.v
- receiver_tb.v
- receiver.v
- baudrate_tb.v
- baudrate.v

Tasks

Compilation

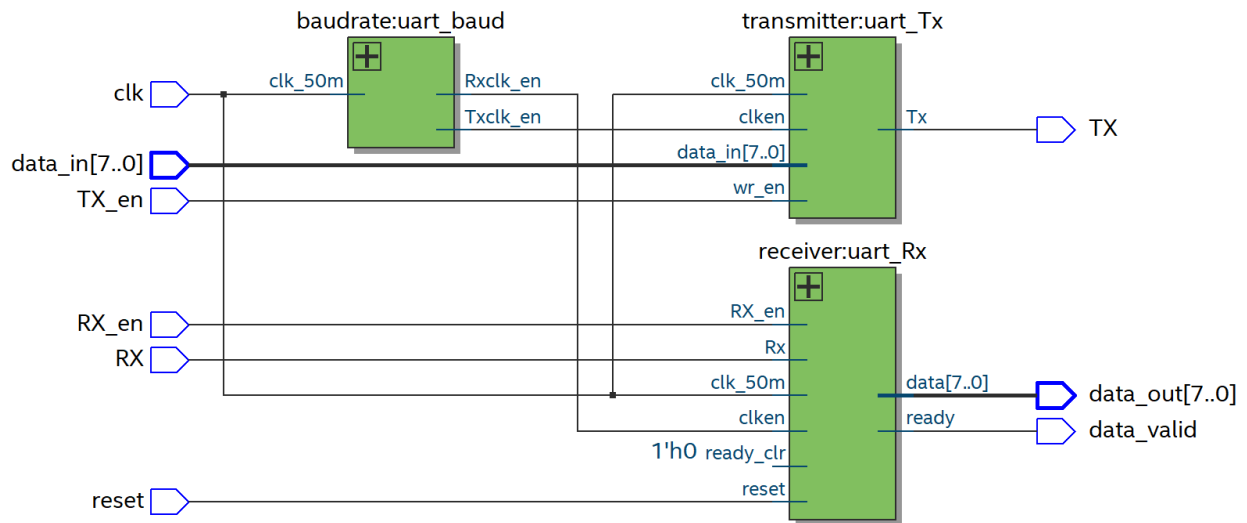
- View Report
- Analysis & Elaboration
- Partition Merge
- Netlist Viewers
- Design Assistant (Post-Mapp
- I/O Assignment Analysis
- Fitter (Place & Route)
- Assembler (Generate program
- TimeQuest Timing Analysis
- EDA Netlist Writer

```

uart.v

1 module uart(output wire [7:0] data_out,
2   output wire data_valid,
3   output wire TX,
4   input wire [7:0] data_in,
5   input wire clk,
6   input wire TX_en,
7   input wire RX_en,
8   input wire RX,
9   input wire reset
10 );
11
12
13 reg ready_clr = 0;
14 wire Txc1k_en, Rxc1k_en;
15
16 baudrate uart_baud( .clk_50m(clk), //50MHz Clock
17   .Rxc1k_en(Rxc1k_en), // Reciever sampling clock 16 samples per bit
18   .Txc1k_en(Txc1k_en) // Transmitter clock 115,200 bps
19 );
20
21 transmitter uart_Tx( .data_in(data_in),
22   .wr_en(TX_en),
23   .clk_50m(clk),
24   .clken(Txc1k_en),
25   .Tx(TX)
26 );
27
28 receiver uart_Rx( .Rx(RX),
29   .ready(data_valid),
30   .ready_clr(ready_clr),
31   .clk_50m(clk),
32   .clken(Rxc1k_en),
33   .data(data_out),
34   .RX_en(RX_en),
35   .reset(reset)
36 );
37
38 endmodule

```

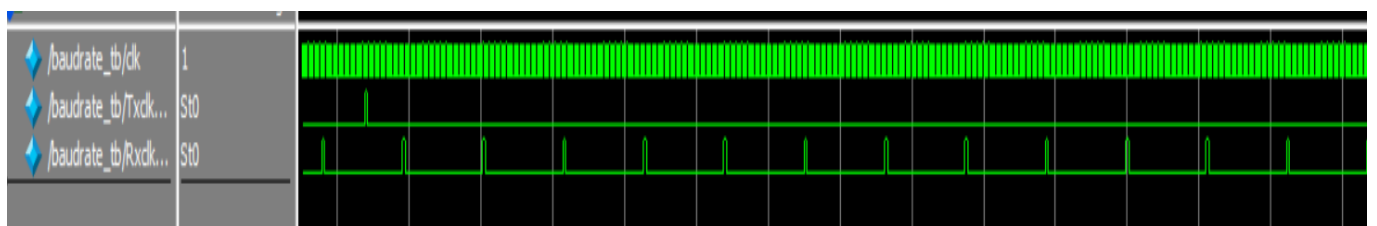


02. Testbenches

- Baud Rate Tb

```

1 //timescale 10ns/1ps
2 module baudrate_tb;
3
4 reg clk = 0;
5 wire Txclk_en,Rxclk_en;
6
7 baudrate br1(.clk_50m(clk),
8               .Rxclk_en(Rxclk_en)
9               .Txclk_en(Txclk_en)
10              );
11
12 always
13 begin
14   #1 clk = ~clk;
15 end
16
17 endmodule
  
```

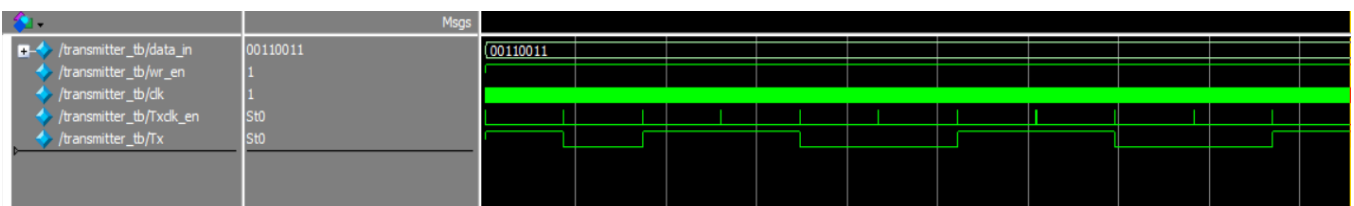


• Transmitter Tb

```

1 module transmitter_tb;
2   reg [7:0] data_in = 8'b00110011;
3   reg wr_en = 1;
4   reg clk = 0;
5   wire Txc1k_en, Rxc1k_en;
6   wire Tx;
7   wire Tx_busy;
8   baudrate br1(.clk_50m(clk),
9               .Rxc1k_en(Rxc1k_en),
10              .Txclk_en(Txc1k_en));
11   transmitter tr1(.data_in(data_in),
12                 .wr_en(wr_en),
13                 .clk_50m(clk),
14                 .clken(Txc1k_en),
15                 .Tx(Tx),
16                 .Tx_busy(Tx_busy));
17   initial #5 wr_en = 1;
18   always
19   begin
20     #1 clk = ~clk;
21   end
22 endmodule

```



• Receiver Tb

Date: May 07, 2024

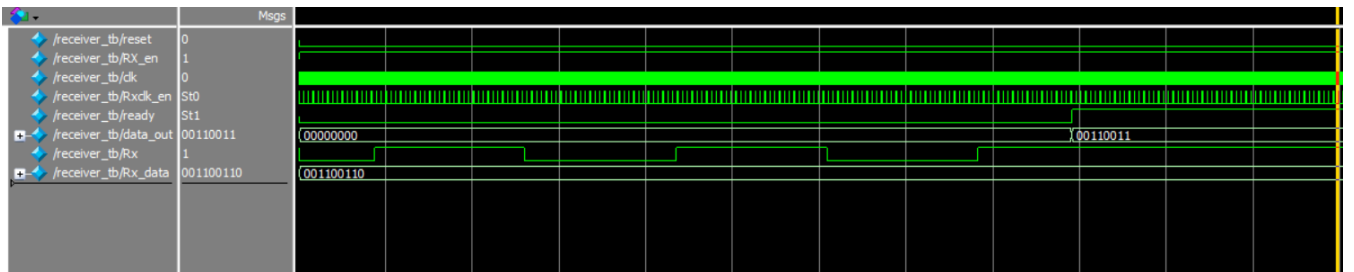
receiver_tb.v

Project: UART

```

1 module receiver_tb;
2   reg reset = 0;
3   reg RX_en = 1;
4   reg clk = 0;
5   wire Txc1k_en, Rxc1k_en;
6   reg ready_clr = 0;
7   wire ready;
8   wire [7:0] data_out;
9   reg Rx = 1'b1;
10  reg [8:0] Rx_data = 9'b001100110;
11  integer i = 0;
12  baudrate br1(.clk_50m(clk),
13              .Rxc1k_en(Rxc1k_en),
14              .Txclk_en(Txc1k_en));
15  receiver rx1(.Rx(Rx),
16             .ready(ready),
17             .ready_clr(ready_clr),
18             .clk_50m(clk),
19             .clken(Rxc1k_en), //We assign Tx clock to enable clock
20             .data(data_out),
21             .RX_en(RX_en),
22             .reset(reset));
23  always
24  begin
25    #1 clk = ~clk;
26  end
27  always @(posedge Txc1k_en)
28  begin
29    if (i!=9) begin
30      Rx = Rx_data[i];
31      i = i+1;
32    end
33    else Rx = 1;
34  end
35 endmodule

```

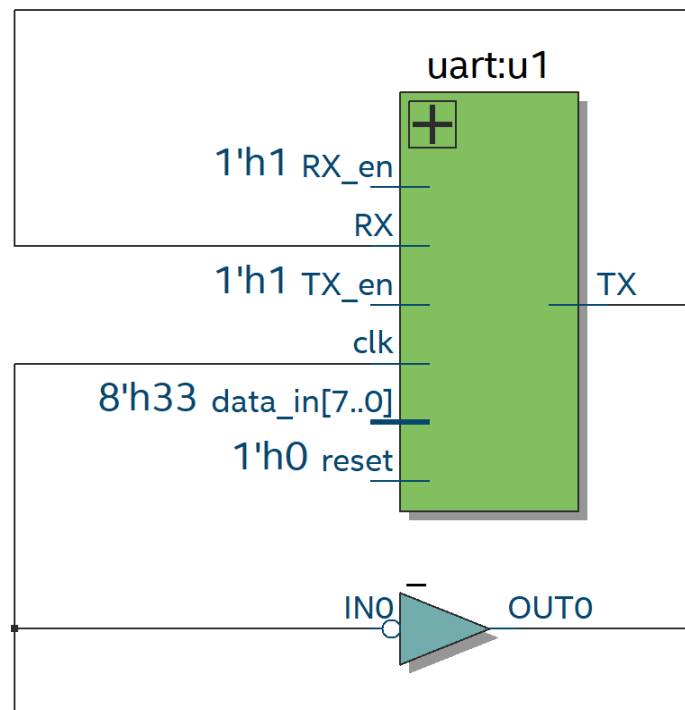
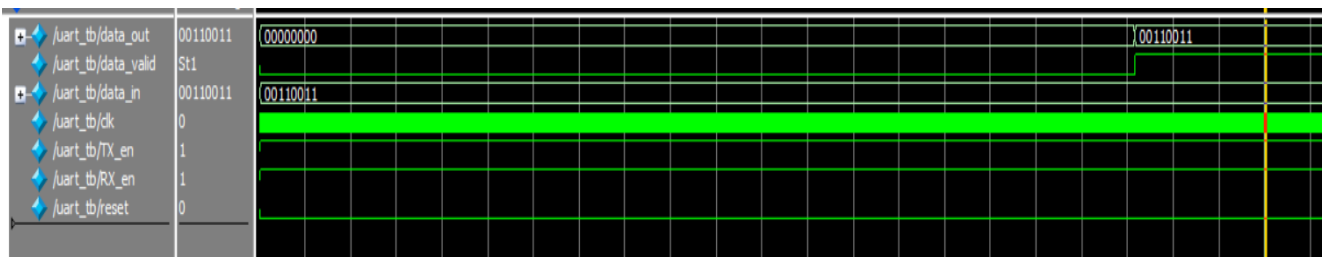


- UART Tb

```

1 module uart_tb;
2     wire [7:0] data_out;
3     wire data_valid;
4     reg [7:0] data_in = 8'b00110011;
5     reg clk = 0;
6     reg TX_en = 1;
7     reg RX_en = 1;
8     reg reset = 0;
9     wire loopback;
10
11
12     uart u1(.data_out(data_out),
13             .data_valid(data_valid),
14             .TX(loopback),
15             .data_in(data_in),
16             .clk(clk),
17             .TX_en(TX_en),
18             .RX_en(RX_en),
19             .RX(loopback),
20             .reset(reset)
21             );
22
23     initial #5 TX_en = 1;
24
25     always
26     begin
27         #1 clk = ~clk;
28     end
29
30 endmodule
31

```



03. Timing Analysis

The screenshot displays the Quartus Prime IDE interface for a project named 'uart.v'. The 'Project Navigator' on the left shows the project files, including 'UART.out.sdc', 'uart_tb.v', 'transmitter_tb.v', 'uart.v', 'transmitter.v', 'receiver_tb.v', 'receiver.v', 'baudrate_tb.v', and 'baudrate.v'. The 'Tasks' pane below it shows the compilation tasks, with 'TimeQuest Timing Analyzer' selected. The 'Table of Contents' pane in the center lists the compilation steps, including 'Flow Summary', 'Flow Settings', 'Flow Non-Default Global Settings', 'Flow Elapsed Time', 'Flow OS Summary', 'Flow Log', 'Analysis & Synthesis', 'Fitter', 'Assembler', 'TimeQuest Timing Analyzer', 'Summary', 'Parallel Compilation', 'SDC File List', 'Clocks', 'Slow 1200mV 85C Model', 'Slow 1200mV 0C Model', 'Fast 1200mV 0C Model', 'Multicorner Timing Analysis Summary', 'Advanced I/O Timing', 'Clock Transfers', 'Report TCCS', 'Report RSKM', 'Unconstrained Paths', 'Messages', 'EDA Netlist Writer', and 'Flow Messages'. The 'Flow Summary' window on the right shows the compilation status as 'Successful' and provides detailed statistics for the UART project.

Flow Summary	
Flow Status	Successful - Tue May 07 16:00:56 2024
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	UART
Top-level Entity Name	uart
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	127 / 22,320 (< 1 %)
Total registers	58
Total pins	23 / 154 (15 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)
