



Advanced Electronics Design Automation
Student ID: 0308686

Assignment No. 2

Asynchronous Serial Data Transmitter

Rev: 01	Dt: 03/10/2005	
Student: Deepak Katkoria	ID:0308686	



1. Summary: A transmitter's mission is to serve as 'parallel-in serial-out'. The Transmitter receives parallel data bytes which are written to the system asynchronously and send out serially in format. In its serial out operation, the transmitter is working synchronously with input high clock frequency as compared to serial out rate.

2. Introduction: Design of Asynchronous Serial Data Transmitter

It has been designed with a register-based architecture. Simple precautions are taken to limit the possibility of incorrect operation due to false triggering.

Proper operation of the transmitter is initialized by a master reset cycle, setting the 'Txout' signal to logic-1. A write cycle begins with the rising edge of the write input and data is written into the shift register. Here write signal is always low for more than three clock cycles. Once the data is registered the busy signal goes high and remain up to the last bit of data is there.

i) Transmitter Block Diagram

All internal operations of the Transmitter are synchronized to the master 'Clock' input. Shift

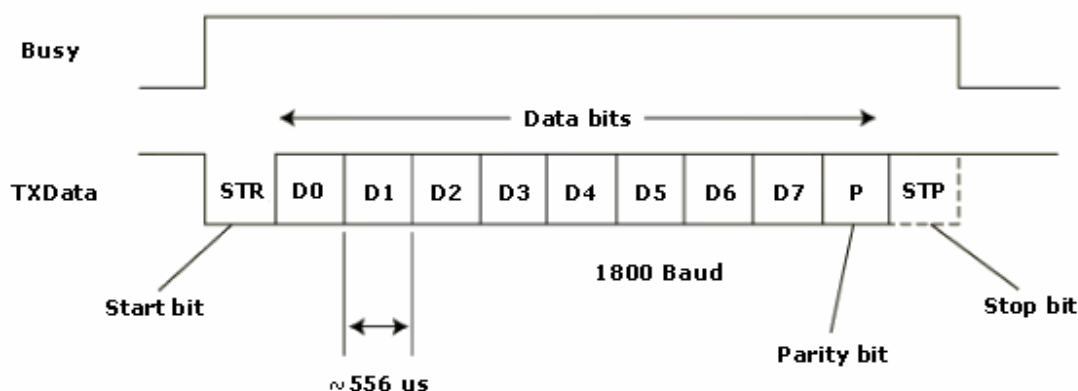
Any asynchronous external control signals should be synchronized to the master clock using appropriate circuitry. During transmission, the 'Busy' output is made active until the entire character has been output which prevents attempts to overwrite the contents of the transmit shift register prior to completing the transmission of a character.

Many systems need to have error detection and correction implemented into design. By adding a single bit to each byte of data, the occurrence of single-bit errors may be monitored. This extra bit is referred to as the parity bit. And same type of detection technique is used in transmitter. Depend upon the odd ones or even ones selection, a bit is generated and concatenate with data bits.

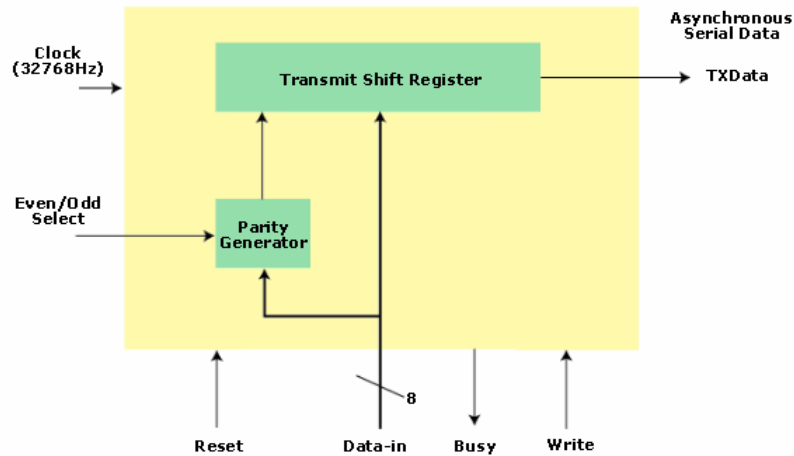
When parity select is changed in input pin then output will update the format after one clock cycle. Suppose if parity select is changed asynchronously to clock then that value will get update on next transaction.

Following a reset operation, the Transmitter outputs a serial data character whenever a new value is written to it. The serial data rate is 1800 bits-per-second (556uS per bit).

Each transmitted character contains a single parity check bit appended to the 8-bit data byte. The parity bit is generated to make the total number of logic-1's in the 9-bit word formed by the data byte and parity bit either even or odd, depending on the state of the 'Even/Odd select' input signal.



Above figure illustrates the format of the serial data output by the Transmitter. Each character is sandwiched between a logic-0 Start bit and a single logic-1 Stop bit. The data bits are transmitted least significant bit first.



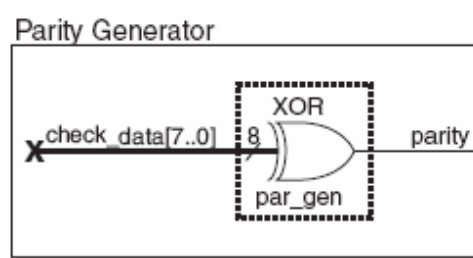
Whole Architecture is divided into three major parts.

- a) Shift register
- b) Parity Generator
- c) Glue Logic

Shift register: Circuits which transfer data serially (one bit at a time). ASDT is composed of an Eleven-bit shift register. This parallel in and serial output shift register is controlled by Clear, Load and shift signals.

The 'Clear' input causes a synchronous reset operation. It is used to force all sequential logic circuitry into a known initial state i.e. logic-1. Load signal is used as write to registers and shift signal is used as enable the shift operation in right direction. The main clock 32768 Hz ($T=30.5\mu\text{s}$) is used to sample all signal. Shifting beyond the last bit (stop bit) cause the 'TxOut' port to remain at logic-1.

Parity Generator: It is made of XOR gate with multiplexer (2:1) at the output of the parity.



The schematic of symbol of XOR parity generator (8 bit) is shown above.

Applying a logic-1 to odd_nt_even (multiplexer select line) results in NOT parity otherwise parity at the output of the Parity generator.

It means if Datain having odd number of ones and odd_nt_even signal is selecting odd then output will be inverted otherwise will be the same.

Glue Logic: This module combine the above two block and generate the signals depend upon the conditions. Parity output is concatenated with Datain to transmit to next block of shift register.



ii). Identify Combinational and Sequential Functions

ASDT is mixed logic. As it is mentioned in the above section, the transmit shift register and parity bit generator are the main blocks of the design. The transmit shift register is a synchronous sequential logic. It consists of an 11-bit shift register and works with a 32768 Hz main clock.

Through shift register glue logic, the busy signal is derived.

The parity bit generator is purely a combinational circuit. It is made of cascaded XOR gates on parallel data lines.

Depending on the Odd_not_Even signal, parity (Parout) is calculated. The output of parity is always synchronous with the 8-bit data and is updated on the final output. Parout is only considered valid at the time of sampling.



3. VHDL code of ASDT

- a) VHDL- HDL code
 - i) Asynchronous Glue Logic
 - ii) Parity Bit Generator
 - iii) Transmit Shift Register
 - iv) Clock Division

i) ASYNCHRONOUS GLUE LOGIC

```
-----
                          ASYNCHRONOUS GLUE LOGIC
-----
--TITTLE: ASYNCHRONOUS GLUE LOGIC
--PROJECT: ASYNCHRONOUS SERIAL DATA TRANSMITTER
-----
--FILE NAME:ASYN_GLUE.VHD
--FILE TYPE:VHDL-HDL
-----

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ASYN_GLUE IS
  PORT
  (
    CLK :           IN STD_LOGIC;
    RESET :        IN STD_LOGIC;
    WRITE :        IN STD_LOGIC;
    PAROUT:        IN STD_LOGIC;
    DATAIN:       IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    CLEAR :        OUT STD_LOGIC;
    LOAD :         OUT STD_LOGIC;
    SHIFT :        OUT STD_LOGIC;
    BUSY :         OUT STD_LOGIC;
    DATAOUT:      OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
  );
END ASYN_GLUE;

ARCHITECTURE ASYN_GLUE_ARCHITECTURE OF ASYN_GLUE IS

SIGNAL COUNT1:INTEGER RANGE 0 TO 255;  --to generate the control signal w.r.t. to 556 us bit rate
SIGNAL VALID,WIRE_BUSY:STD_LOGIC:=0';
BEGIN

PROCESS (RESET,CLK,WRITE)
BEGIN
  IF clk'EVENT AND clk = '1' THEN
    IF RESET = '0' THEN
      COUNT1<= 0;
      VALID <= '0';
      WIRE_BUSY <= '0';
    ELSIF WRITE = '0' THEN
      COUNT1<= 1;
      WIRE_BUSY <= '0';
      VALID <= '1';
      --at this point load will trigger the shift register
    
```



```
ELSIF COUNT1 < 234 AND COUNT1 >=1 THEN -- when write = '1'
    COUNT1<= COUNT1 + 1; -- count will assit to generate a busy/shift signal
    WIRE_BUSY <= '1';
    IF COUNT1 < 18 AND COUNT1 >=1 THEN --generate a signal which will be
--recognize by clkout(540us)
        VALID <='1';
    ELSE
        VALID <='0';
    END IF;
ELSE
    COUNT1<=0;
    VALID <= '0' ;
    WIRE_BUSY <= '0';
    END IF;
END IF;
LOAD <= VALID;
BUSY <= WIRE_BUSY;
SHIFT <= WIRE_BUSY;
DATAOUT <= PAROUT & DATAIN;

END PROCESS;
CLEAR <= RESET; --buffer reset to shift register
END ASYN_GLUE_ARCHITECTURE;
```

ii) Parity Bit Generator

Parity Bit Generator

--Tittle: Parity Bit Generator
--Project: Asynchronous Serial Data Transmitter

--File Name:par_mod.vhd
--File Type:VHDL-HDL

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE IEEE.STD_LOGIC_UNSIGNED.ALL;
-- ENTITY DECLARATION
ENTITY PAR_MOD IS
    PORT (
        ODD_OR_EVEN: IN STD_LOGIC;
        DATAIN : IN STD_LOGIC_VECTOR(7 DOWNT0 0);
        PAROUT : OUT STD_LOGIC
    );
END PAR_MOD;

-- ARCHITECTURE BODY
ARCHITECTURE PAR_MOD_ARCHITECTURE OF PAR_MOD IS
--DEFAULT EVEN
SIGNAL NT_PAR,PAR: STD_LOGIC;
BEGIN
PAR <= (((DATAIN(7) XOR DATAIN(6))XOR (DATAIN(5) XOR DATAIN(4)))
        XOR ((DATAIN(3)XOR DATAIN(2)) XOR (DATAIN(1) XOR DATAIN(0))));
-- GIVES ODD PARITY
```



```
NT_PAR <= NOT PAR;
PAROUT <= PAR WHEN ODD_OR_EVEN = '0' ELSE NT_PAR;

END PAR_MOD_ARCHITECTURE;
```

iii) TRANSMIT SHIFT REGISTER

TRANSMIT SHIFT REGISTER

--Title: Transmit Shift Register
--Project: Asynchronous Serial Data Transmitter

--File Name: tx_shft.vhd
--File Type: VHDL-HDL 87

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY TX_SHFT IS
    PORT(
        CLKOUT:          IN STD_LOGIC;
        CLEAR:           IN STD_LOGIC;
        LOAD:            IN STD_LOGIC;
        SHIFT:           IN STD_LOGIC;
        DATAOUT:        IN STD_LOGIC_VECTOR(8 DOWNTO 0);
        TxDATA:          OUT STD_LOGIC
    );
END TX_SHFT;
```

-- ARCHITECTURE BODY

ARCHITECTURE TX_SHFT_ARCHITECTURE OF TX_SHFT IS

```
SIGNAL DAT_PAR: STD_LOGIC_VECTOR(11 DOWNTO 0) := (OTHERS => '1');
BEGIN
```

```
PROCESS(CLKOUT, CLEAR, LOAD, SHIFT, DATAOUT)
```

```
BEGIN
```

```
    IF CLKOUT'EVENT AND CLKOUT='1' THEN
```

```
        IF CLEAR='0' THEN
```

```
            DAT_PAR <= (OTHERS => '1'); --set output to logic '1'
```

```
        ELSIF LOAD='1' THEN
```

```
            DAT_PAR <= ('1' & DATAOUT & '0' & '1'); --stop bit, data, start bit
```

```
        ELSIF SHIFT='1' THEN
```

```
            DAT_PAR(10 DOWNTO 0) <= DAT_PAR(11 DOWNTO 1); -- shifting right
```

```
            DAT_PAR(11) <= '1';
```

```
        END IF;
```

```
    END IF;
```

```
TxDATA <= DAT_PAR(0); -- least bit is transmitted out
```



```
END PROCESS;  
END TX_SHFT_ARCHITECTURE;
```

iv) Clock Division

Clock Division

```
LIBRARY IEEE;  
USE IEEE.STD_LOGIC_1164.ALL;  
USE IEEE.STD_LOGIC_ARITH.ALL;  
--USE IEEE.STD_LOGIC_UNSIGNED.ALL;  
  
ENTITY CLK_DIV IS  
  PORT (  
    CLK :          IN STD_LOGIC;  
    RESET :       IN STD_LOGIC;  
    CLKOUT :      OUT STD_LOGIC  
  );  
END CLK_DIV;  
  
ARCHITECTURE CLK_DIV_ARCHITECTURE OF CLK_DIV IS  
  
  SIGNAL COUNTER: STD_LOGIC_VECTOR (3 DOWNTO 0);--for 4-bit counter  
  SIGNAL COUNTERTC :STD_LOGIC :='1';           --terminal count for last flip-flop  
  SIGNAL CLK_OUT :STD_LOGIC :='0';             --internal clkout signal  
  
  BEGIN  
  PROCESS (RESET,CLK)  
  
  BEGIN  
    IF RESET = '0' THEN  
      COUNTER <= (OTHERS =>'0');  
      COUNTERTC <= '0';  
    ELSIF CLK'EVENT AND CLK = '1' THEN  
      IF COUNTER = "1000" THEN -- when count = 8, generate a signal to toggle the t-F/F  
        COUNTERTC <='0';  
        COUNTER<="0000";  
      ELSE --- logic at the clk_out will remain same  
        COUNTERTC <='1';  
        COUNTER <= COUNTER + 1;  
      END IF;  
  
      IF COUNTERTC = '0' THEN -- to give 50 % clock cycle synchronised to terminal count  
        CLK_OUT <= NOT CLK_OUT;  
      END IF;  
    END IF;  
    CLKOUT<= CLK_OUT;  
  
  END PROCESS;  
END CLK_DIV_ARCHITECTURE;
```




4. Simulation of each Design using Leapfrog

Each major module of the ASDT is simulated individually with considering all possible case. This simulation testing is more functional tested otherthen for timing. Lots of precaution is taken into account to get ease of final top level de-bugging.

For transmit shift register simulation, two different data stream are checked. Here Parity bit is consider as a datain[8] bit without counting the odd_or_even signal.

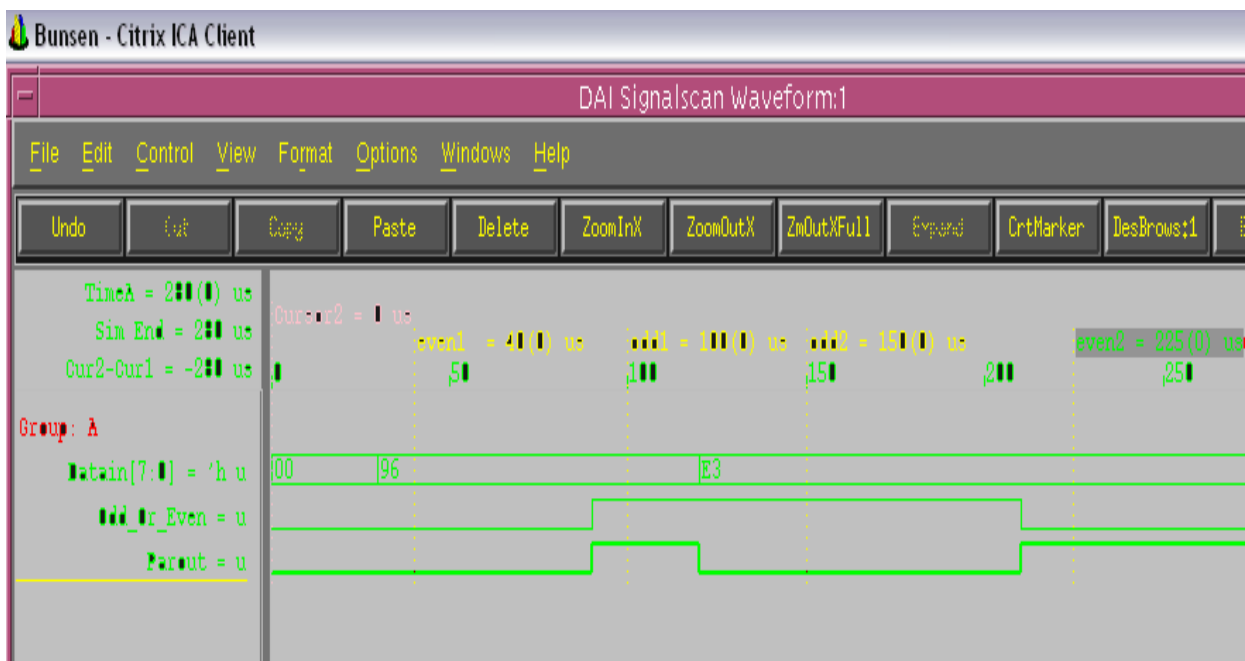
For each module, following section describes the test vectors and their result.

i) Parity Bit Generator

This module is combinational. Here all four possiblity is shown.

- a) Datain[7:0] = 96h = "1001_0110" (number of 1's = even)
Odd_or_even = '0' (number of 1's output = even)
Parout = '0' (result = number of 1's are even)

- b) Datain[7:0] = 96h = "1001_0110" (number of 1's = even)
Odd_or_even = '1' (number of 1's output = odd)
Parout = '1' (result = number of 1's are odd)



- c) Datain[7:0] = E3h = "1110_0011" (number of 1's = odd)
Odd_or_even = '1' (number of 1's output = odd)
Parout = '0' (result = number of 1's are odd)

- d) Datain[7:0] = E3h = "1110_0011" (number of 1's = odd)
Odd_or_even = '0' (number of 1's output = even)
Parout = '1' (result = number of 1's are even)



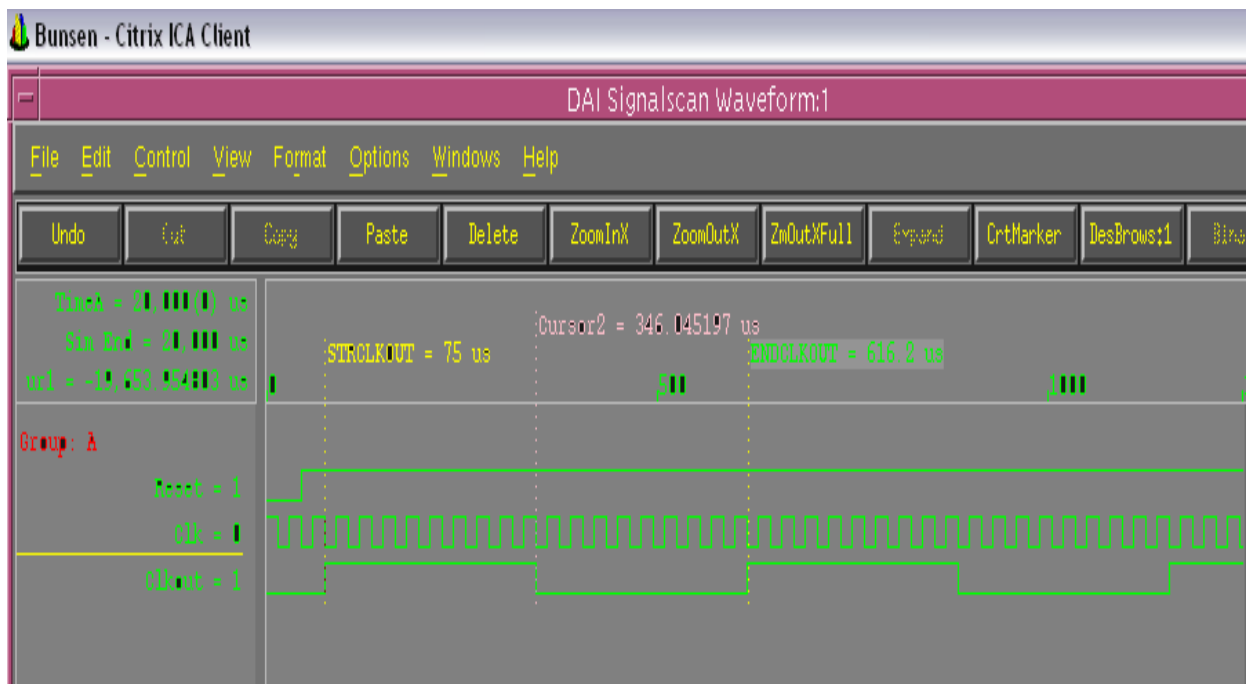
ii) Clock Division

ASDT transmit serial data at the rate of 556us/bit. To transfer with such low speed, ASDT is using internal clock divider. Transmit shift register use this clock to send serial data with start and stop bit.

Reset = '1'

Clk = 30 us (master clock)

Clkout = 616 - 75 ~ 541 us (master clock divide by 18)





iii) Transmit Shift Register

All transaction of register is synchronised to rising CLKOUT clock. When LOAD goes high, it loads all parallel data into the parallel in serial out register.

Input shift triggers all flip-flop to shift the content by one cycle. The last flip-flop shifts the least bit of DATAOUT first. Consequently each bit gets shifted towards the Txdata Port.

On the last bit (stop bit) shift signal goes low, to indicate that now there is no data to transfer.

* When Load = '1', Dataout gets latch in register

* When shift = '1' (on rising edge of clock), Txdata indicates the start of operation.

Then Txdata sends the output in following format.

[start bit = '0' | Do | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8/parity bit | stop bit = '1']

a) Dataout[8:0] = "096h" = 0 1001 0110 (0 = parity bit, 96 = datain)

[start bit = '0' | Do = '0' | D1 = '1' | D2 = '1' | D3 = '0' | D4 = '1' | D5 = '0' | D6 = '0' | D7 = '1' | parity bit = '0' | stop bit = '1']

b) Dataout[8:0] = "1E2h" = 1 1110 0010 (0 = parity bit, 96 = datain)

[start bit = '0' | Do = '0' | D1 = '1' | D2 = '0' | D3 = '0' | D4 = '0' | D5 = '1' | D6 = '1' | D7 = '1' | parity bit = '1' | stop bit = '1']





5. Integration at top Level

Asynchronous Serial Data Transmitter top-model

ASDT Top Model

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE IEEE.STD_LOGIC_UNSIGNED.ALL;

LIBRARY work;

ENTITY ASDT IS
    port
    (
        CLK :                IN STD_LOGIC;
        ODD_OR_EVEN :        IN STD_LOGIC;
        RESET :              IN STD_LOGIC;
        WRITE :              IN STD_LOGIC;
        DATAIN :            IN STD_LOGIC_VECTOR(7 downto 0);
        TxDATA :             OUT STD_LOGIC;
        BUSY :               OUT STD_LOGIC
    );
END ASDT;

ARCHITECTURE ASDT_TOP OF ASDT IS

    COMPONENT ASYN_GLUE
        PORT (
            CLK :                IN STD_LOGIC;
            RESET :              IN STD_LOGIC;
            WRITE :              IN STD_LOGIC;
            PAROUT :             IN STD_LOGIC;
            DATAIN :            IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            CLEAR :              OUT STD_LOGIC;
            LOAD :               OUT STD_LOGIC;
            SHIFT :              OUT STD_LOGIC;
            BUSY :               OUT STD_LOGIC;
            DATAOUT :           OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT TX_SHFT
        PORT (
            CLKOUT :            IN STD_LOGIC;
            CLEAR :             IN STD_LOGIC;
            LOAD :              IN STD_LOGIC;
            SHIFT :             IN STD_LOGIC;
            DATAOUT :          IN STD_LOGIC_VECTOR(8 DOWNTO 0);
            TxDATA :            OUT STD_LOGIC
        );
    END COMPONENT;

    COMPONENT CLK_DIV
```



```
PORT (
  CLK:          IN STD_LOGIC;
  RESET:       IN STD_LOGIC;
  CLKOUT:      OUT STD_LOGIC
);
END COMPONENT;

COMPONENT PAR_MOD
  PORT (
    ODD_OR_EVEN:  IN STD_LOGIC;
    DATAIN:     IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    PAROUT:      OUT STD_LOGIC
  );
END COMPONENT;

signal CLEAR,CLKOUT,PAROUT,SHIFT,LOAD: STD_LOGIC;
signal DATAOUT : STD_LOGIC_VECTOR(8 DOWNTO 0);

BEGIN
BLK1 : ASYN_GLUE
PORT MAP(
  CLK=>      CLK,
  RESET=>    RESET,
  WRITE=>    WRITE,
  PAROUT=>   PAROUT,
  DATAIN=>  DATAIN,
  CLEAR=>    CLEAR,
  LOAD=>     LOAD,
  SHIFT=>    SHIFT,
  BUSY=>     BUSY,
  DATAOUT=> DATAOUT
);

BLK2 : TX_SHFT
PORT MAP(
  CLKOUT=>   CLKOUT,
  CLEAR=>    CLEAR,
  LOAD=>     LOAD,
  SHIFT=>    SHIFT,
  DATAOUT=> DATAOUT,
  TxDATA=>   TxDATA
);

BLK3 : CLK_DIV
PORT MAP(
  CLK=>      CLK,
  RESET=>    RESET,
  CLKOUT=>   CLKOUT
);

BLK4 : PAR_MOD
PORT MAP(
  ODD_OR_EVEN=>ODD_OR_EVEN,
  DATAIN=>    DATAIN,
  PAROUT=>     PAROUT
);

END ASDT_TOP;
```



6. Test bench for ASDT

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
```

```
entity ASDT_tstbnch is
port ( txdata,busy: out std_logic);
end ASDT_tstbnch;
```

```
ARCHITECTURE BNCH OF ASDT_tstbnch IS
```

```
COMPONENT ASDT
```

```
port (
    CLK :           IN STD_LOGIC;
    ODD_OR_EVEN :   IN STD_LOGIC;
    RESET :         IN STD_LOGIC;
    WRITE :         IN STD_LOGIC;
    DATAIN :       IN STD_LOGIC_VECTOR(7 downto 0);
    TxDATA :        OUT STD_LOGIC;
    BUSY :          OUT STD_LOGIC);
```

```
END COMPONENT;
```

```
constant period: time:= 30 us; --to generate signal w.r.t clk=30us
constant period2:time := 260 * period; --to generate signal w.r.t clkout
constant period3: time:= 320 * period;
```

```
SIGNAL CLK,ODD_OR_EVEN,RESET,WRITE: STD_LOGIC;
SIGNAL DATAIN: STD_LOGIC_VECTOR(7 downto 0);
```

```
BEGIN
```

```
ASDT_UUT: ASDT
PORT MAP (CLK,ODD_OR_EVEN,RESET,WRITE,DATAIN,TxDATA,BUSY);
```

```
clock_gen : process
begin
    clk <= '1', '0' after 15 us; -----T=30.5us
    wait for 30 us;
end process;
```

```
reset <= '0', '1' after 45 us, '0' after period2, '1' after period2 + 30 us ;
```

```
datain<= "00000000", "10010110" after 275 us, "11100010" after period2;
```

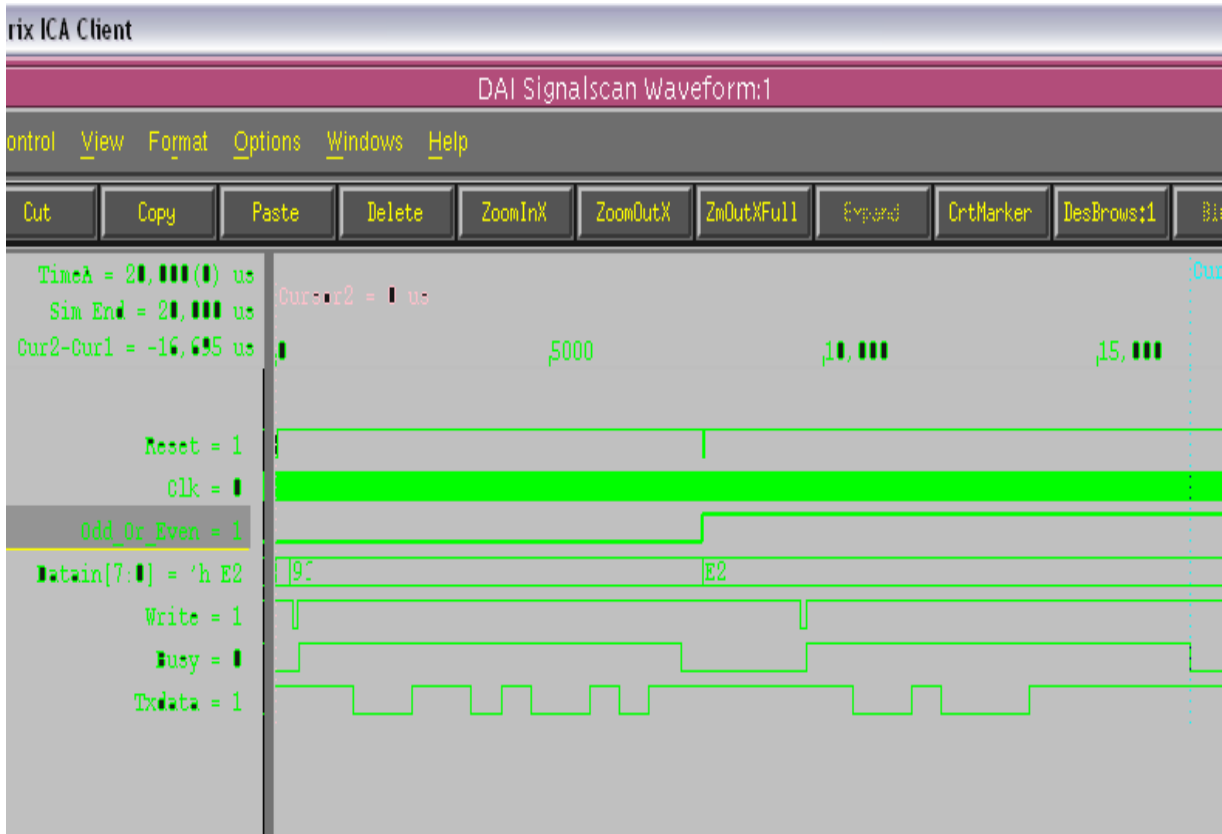
```
write<= '1','0' after 315 us, '1' after 405 us, '0' after period3, '1' after period3 + 90 us;
```

```
odd_or_even <= '0', '1' after period2;
```

```
end;
```



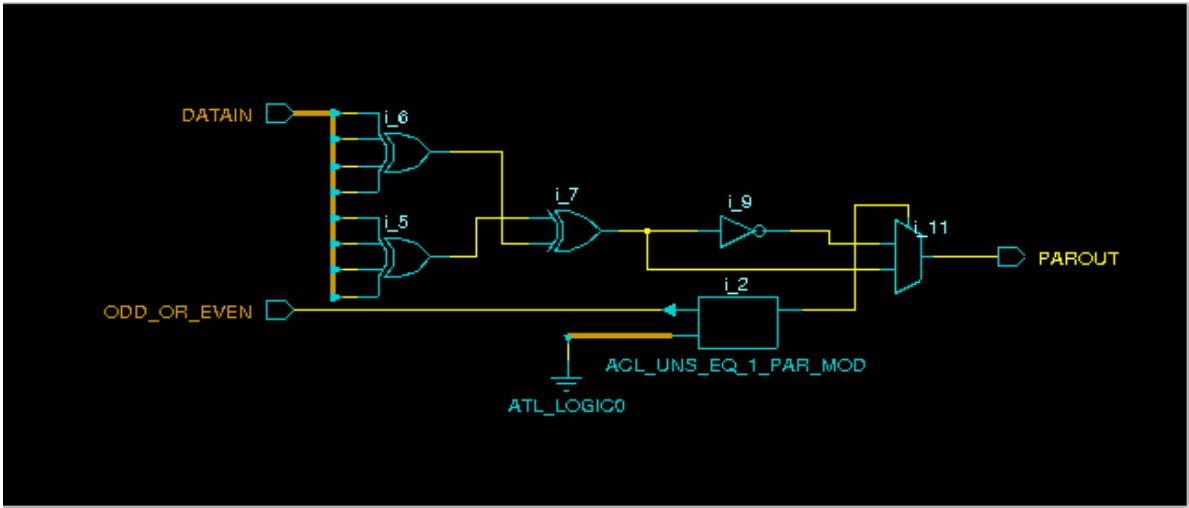

b) The below waveform shows the top level verification of the ASDT.





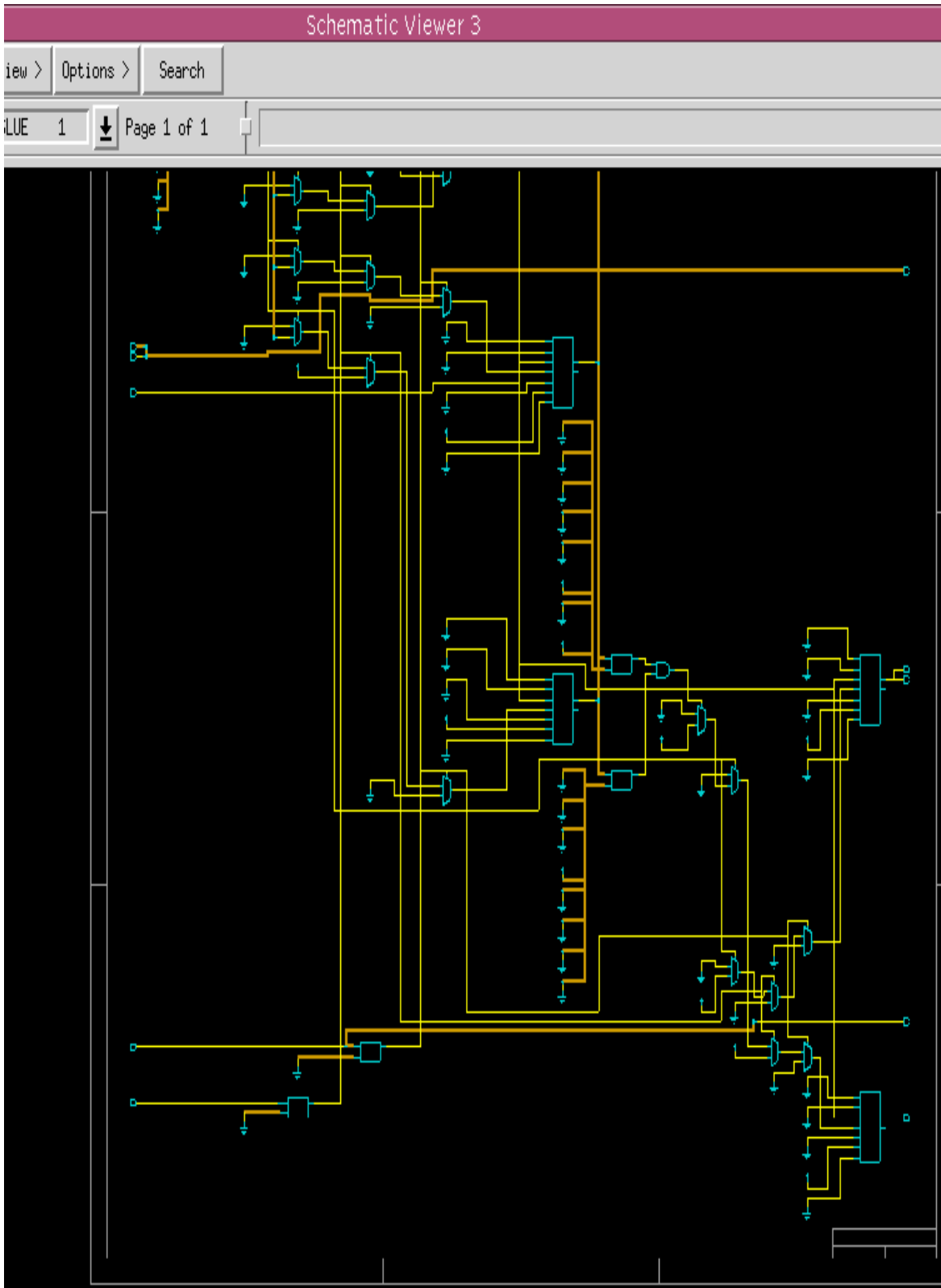
7. Hierarchical Schematic Diagram for the ASDT using Logic Synthesis

a) Parity Generator:



Synthesis Logic of PAR_MOD.Vhd

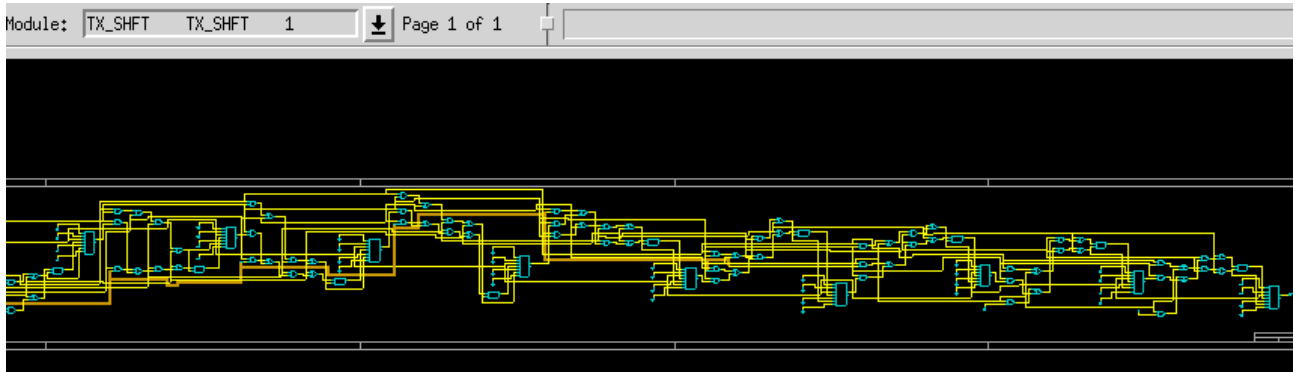
b) Asynchronous Glue



Synthesis Logic of ASYC_GLUE.Vhd

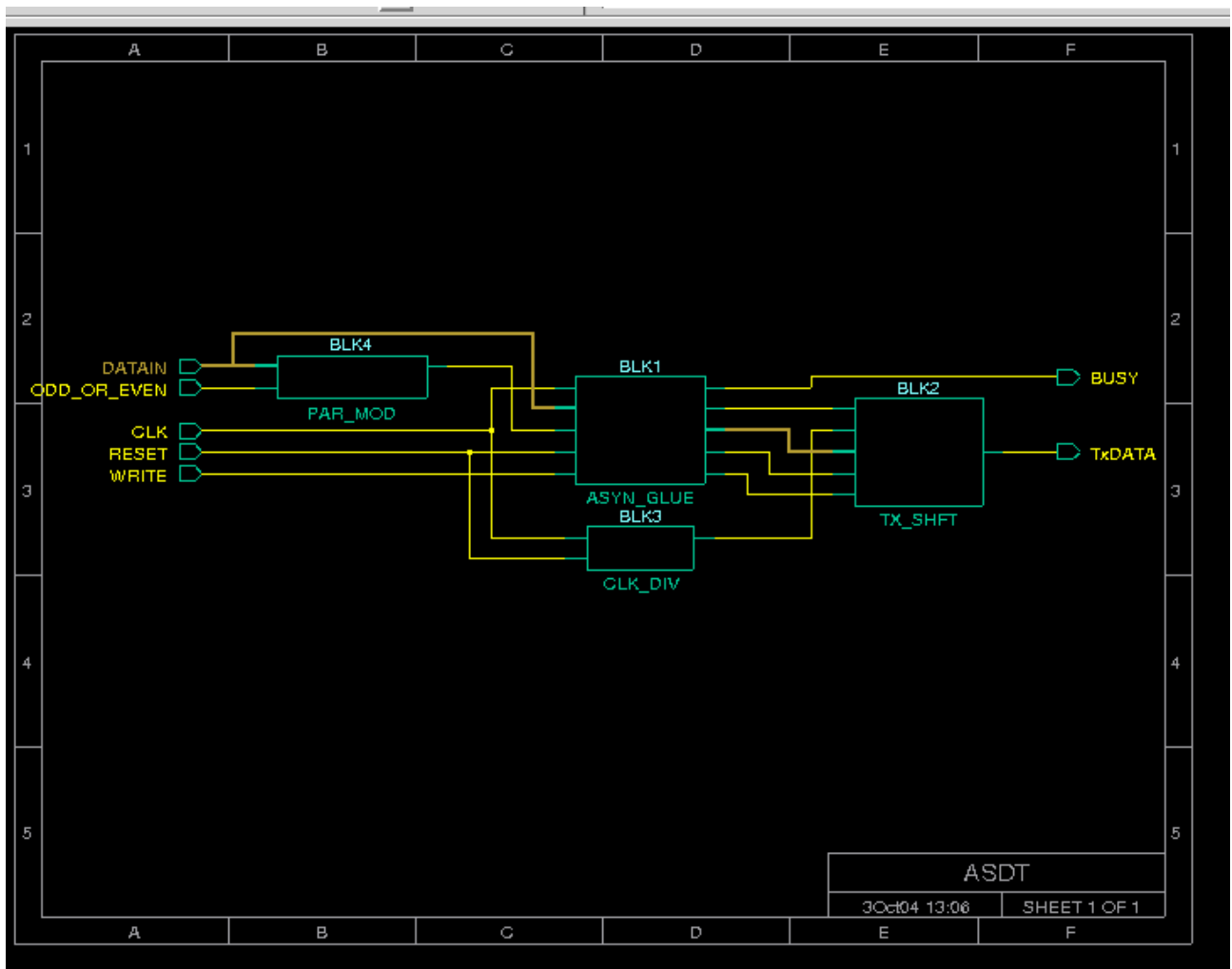


C) Transmit Shift Register:



Synthesis Logic of TX_SFHT.Vhd

D) Top Level Schematic



Schematic Diagram of ASDT



8. Verilog – HDL Module Descriptions

a) Parity Bit Generator

i) Parity Bit Generator:-Verilog Code

Parity Bit Generator

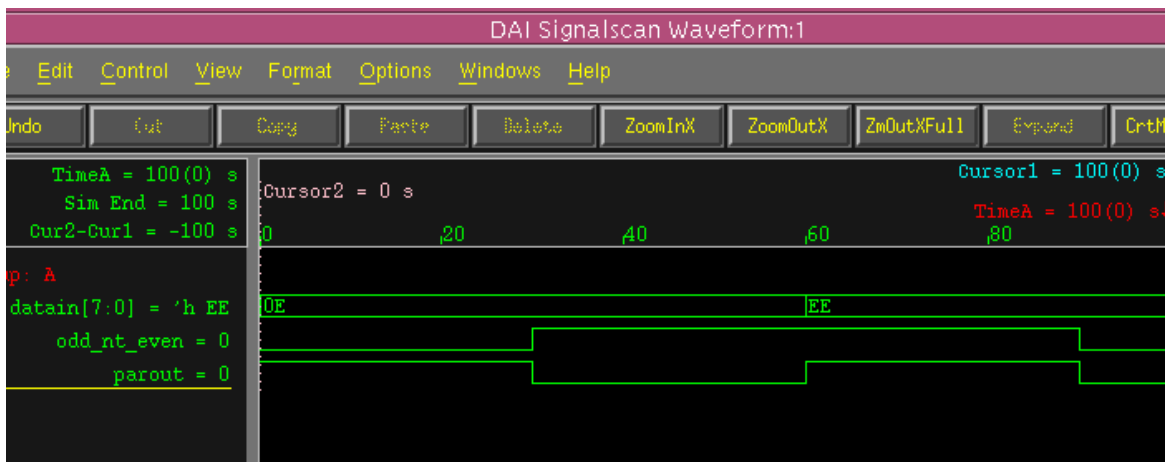
```
//-----  
//Title: Parity Bit Generator  
//Project: Asynchronous Serial Data Transmitter  
//-----  
//File Name:par_gen.v  
//File Type:Verilog-HDL  
//-----  
// Description:  
// This module is combinational. Applying a logic-1 to'odd_nt_even' results in overall  
// odd parity of'datain' and 'parout'; otherwise the overall party is even.  
  
module par_gen  
(datain,      // parallel input data for register  
 odd_nt_even, // input to select parity  
 parout       // parity output  
);  
//-----Input Ports-----  
input [7:0] datain;  
input odd_nt_even;  
//-----Output Ports-----  
output parout;  
//-----Internal data & net type-----  
reg parout;  
wire par_out;  
//-----Code Starts Here-----  
  
//explicit usage of continuous assignent  
assign par_out = ^datain; //par_out will be high if number of 1's are odd  
  
always @ (par_out or odd_nt_even)  
begin  
    if (!odd_nt_even) // if odd_nt_even = 0  
        parout = par_out; // if number of 1's are even then output will be low otherwise high  
    else  
        parout = ~ par_out; //if number of 1's are odd then output will be low  
    end  
endmodule
```



ii) Parity Bit Generator:-Test bench

```
module par_gentst_bnch;  
wire parout;  
  
reg [7:0]datain;  
reg odd_nt_even;  
  
par_gen DUT(datain,odd_nt_even,parout); //module instantiation  
  
initial  
begin  
  
datain = 8'b 00001110;  
odd_nt_even = 0;  
# 30 odd_nt_even = 1;  
# 30 datain = 8'b 11101110;  
# 30 odd_nt_even = 0;  
# 10 $stop;  
end  
endmodule
```

iii) Parity Bit Generator:-Simulated Waveform



- | | |
|--|--|
| a) Datain[7:0] = 0Eh = "00001110"
Odd_or_even = '0'
Parout = '1' | (number of 1's = odd)
(number of 1's output = even)
(result = number of 1's are even) |
| b) Datain[7:0] = 0Eh = "00001110"
Odd_or_even = '1'
Parout = '0' | (number of 1's = odd)
(number of 1's output = odd)
(result = number of 1's are odd) |
| c) Datain[7:0] = E3h = "11101110"
Odd_or_even = '1'
Parout = '1' | (number of 1's = even)
(number of 1's output = odd)
(result = number of 1's are odd) |
| d) Datain[7:0] = E3h = "11101110"
Odd_or_even = '0'
Parout = '0' | (number of 1's = even)
(number of 1's output = even)
(result = number of 1's are even) |



b) Transmit Shift Register

i) Transmit Shift Register:-Verilog Code

Transmit Shift Register

```
module tx_shft (
clear,          // reset to shift register
load,          // enable for shift register
clkout,        // clock input
shift,         // shift input for register
datain,        // parallel input data for register
txout          // serial out data
);
//-----Input Ports-----
input [8:0] datain;
input clear,load,clkout,shift;
//-----Output Ports-----
output txout;
reg txout;
//-----Internal net & data types-----
// Note that lenght of shift register can be changed.If you want to replace your project
// specific blocks here:you can replace this width.

parameter      lenght= 12; //lenght of shift register - would be lenght-1
reg [lenght-1:0] dat_par;
//-----Code Starts Here-----

always @ (negedge clear or posedge clkout)      // clock event or control
begin
    if (! clear)                                // conditional statement to reset the signal to high state
        begin
            dat_par <= 12'b111111111111;
            txout <= dat_par[0];
        end
    else if (load)                               // loading the valid data
        begin
            dat_par <= {1'b1,datain,1'b0,1'b1};
            txout <= dat_par[0];
        end
    else if (shift)                              // shifting right
        begin
            dat_par <= {1'b1,dat_par[10:1]};
            txout <= dat_par[0];                // loading the least bit to serial output
        end
    end
endmodule
```



ii) Transmit Shift Register:-Test bench

```
module tx_shftst_bnch;

wire txout;
reg clear,load,clkout,shift;
reg [8:0] datain;

tx_shft DUT (clear,load,clkout,shift,datain,txout); // module instantiation

initial
begin
    clkout = 1'b0;          // clock generation
    forever
        #275 clkout = ~clkout;
end
initial
fork
    datain = 9'b011101110;

    clear = 1'b0;
    #600 clear = 1'b1;

    load = 1'b0;
    #835 load = 1'b1;
    #1410 load = 1'b0;

    shift = 1'b0;
    #1440 shift = 1'b1;
    #8000 shift = 1'b0;

    #9000 $stop; //stop the simulation after 9000 unit

join
endmodule
```



ii) Transmit Shift Register:- Simulated Waveform



* When Load = '1', Dataout gets latch in register

* When shift = '1' (on rising edge of clock), Txdata indicates the start of operation.

Then Txdata sends the output in following format.

[start bit = '0' | Do | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | parity bit | stop bit = '1']

a) Dataout[8:0] = "0EEh" = 0 1110 1110 (0 = parity bit, EE = datain)

[start bit = '0' | Do = '0' | D1 = '1' | D2 = '1' | D3 = '1' | D4 = '0' | D5 = '1' | D6 = '1' | D7 = '1' | parity bit = '0' | stop bit = '1']



9. Conclusion:

In short, the ASDT is very easy to use. Simple considerations to timing and flexibility makes design a clean solution for serial transmitter application.

References

- [1] **Ian Elliott** of Northumbria University, www.ami.ac.uk
- [2] **VHDL for designers**, Stefan Sjöholm