

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

## Serial/Parallel UART with 1-bit Error Correction using Hamming Code

By

Zhou Zheng (zz529)

Amay Yadav (ady231)

Thomas Raphel (tr1147)

### Introduction

Our project is a basic simulation of UART (Universal asynchronous receiver/transmitter). UART is the microchip with programming that controls a computer's interface to its attached serial devices. Its functions are simple and limited to the following:

- Converts the bytes it receives from the computer along parallel circuits into a single serial bit stream for outbound transmission
- On inbound transmission, converts the serial bit stream into the bytes that the computer handles.
- Adds a parity bit (if it's been selected) on outbound transmissions and checks the parity of incoming bytes (if selected) and discards the parity bit.
- Adds start and stop delineators on outbound and strips them from inbound transmissions.
- Handles interrupts from the keyboard and mouse (which are serial devices with special ports).

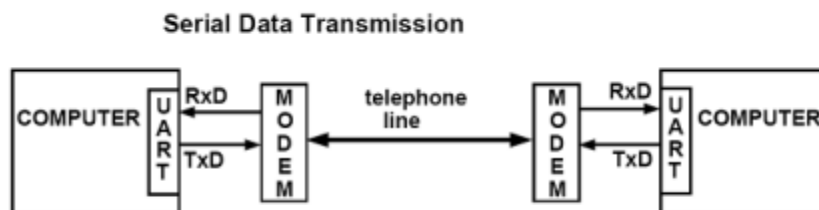


Figure 2.1: Serial Data Transmission

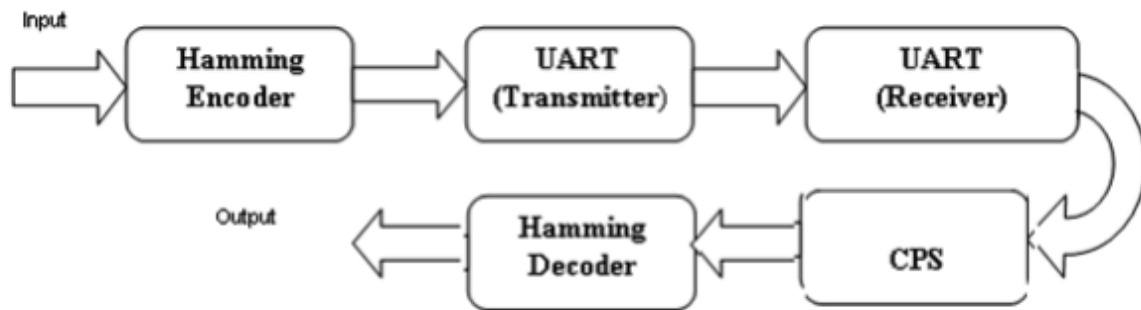
## Serial/Parallel UART with 1-bit Error Correction using Hamming Code

Our project emulates this behavior on a FPGA board. We have taken an **8-bit input** on the FPGA using the 8 switches. The **input data is parallel** as we set all the switches right at the start. To demo the transmission on the FPGA board we have used a **baud generator** to slow down the clock from **100 MHz to 1 Hz**, so that we can catch the transmission on the 8-leds.

Now after taking the 8-bit input data it is converted to a **12-bit codeword** using **hamming encoder** to add **parity bits**. The program also adds a **start bit** and an **end bit (delimiter)** making them **14 bits**.

These 14 bits are transmitted to the receiver and transmitted to the receiver bit by bit i.e. serially. This where the code makes the parallel input to serial output (**PISO**).

Once the receiver receives the 14-bits transmitted to it takes in the serial input to parallel output (**SIPO**). The receiver then removes the first and last bit and then **checks for errors** using the checksum or parity using hamming decoder. If no error is seen then the parity bits are removed from the codeword and the output is seen on the **7-segment display**. If there is an error in any input or non-parity bit then that is corrected and then the output is seen. More than one bit error cannot be corrected.



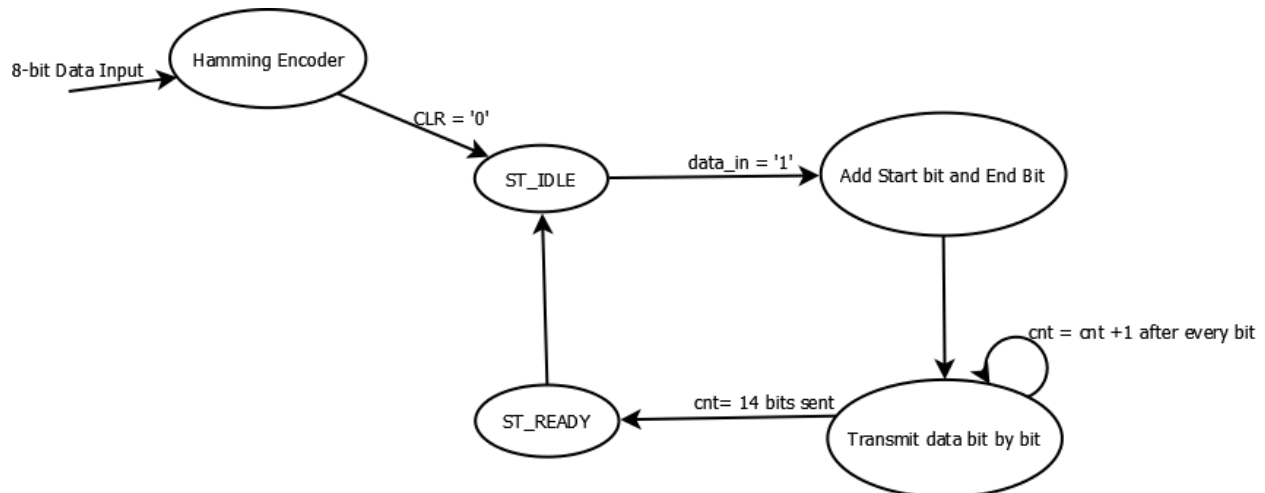
Youtube video =><https://www.youtube.com/watch?v=hNIQu1dE688&feature=youtu.be>

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

## High-Level Overview

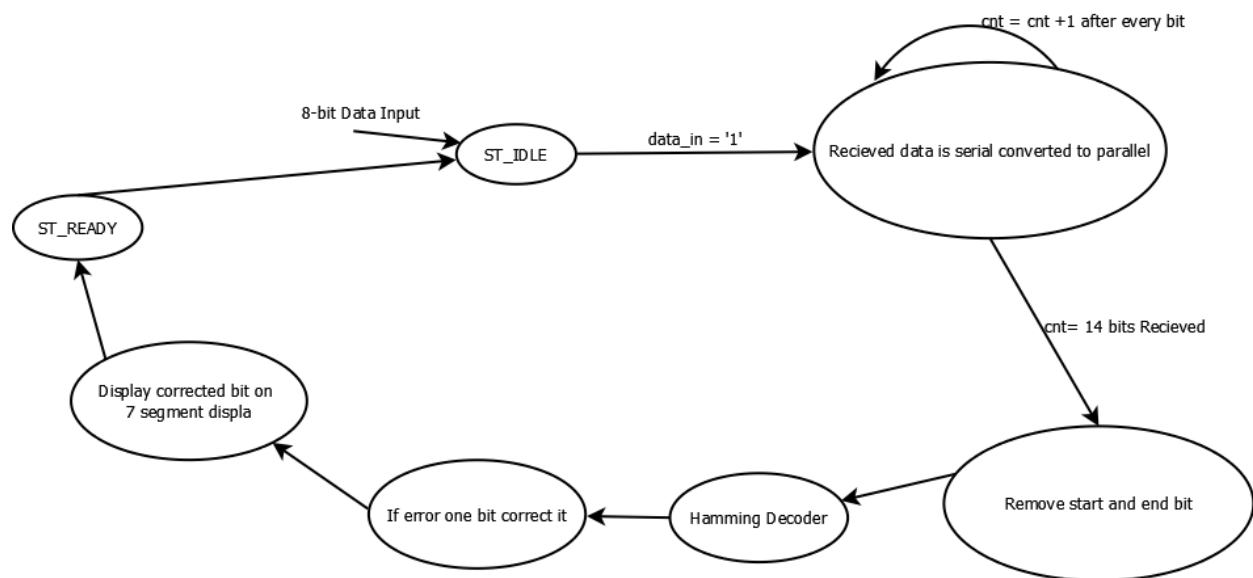
### Transmitter:

The transmitter takes the 12 bit input from the ParityGenerator block, converts it into serial ,bit by bit at each clock. A delimiter bit is also added to it. A total of 13 bits are transmitted.The bits moving serially outward can be seen by the movement of leds 7 to 0.



### Receiver:

The receiver block has a 1 bit input and a 12 bit output.Each bit enters the receiver in a rising edge and is put into a variable temp\_data and is pushed inward ,1 at a time in each clock cycle.After all the bits have entered,it is flushed out or made all 1's.It is then sent to the error correcting block along with the hamming decoder.



# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

## HammingEncoder:

Hamming encoder converts 8-bit data bits to 12-bit codeword by inserting parity bits. Parity bits are placed at powers of 2. Remaining positions are filled with data bits. Here we are dealing with even parity check. Parity bits are added based on the algorithm. Using X-OR gate we can generate parity bits as shown here. Finally the result obtained was shown below.

P1 = XOR of bits (3, 5, 7, 9, 11)

P2 = XOR of bits (3, 6, 7, 10, 11)

P4 = XOR of bits (5, 6, 7, 12)

P8 = XOR of bits (9, 10, 11, 12)

Here is an example:

A byte of data: 10011010 => 9A (x16)

Create the data word, leaving spaces for the parity bits: `_ _ 1 _ 0 0 1 _ 1 0 1 0`

Calculate the parity for each parity bit (a ? represents the bit position being set):

- Position 1 checks bits 3, 5, 7, 9, 11:  
`? _ 1 _ 0 0 1 _ 1 0 1 0`. Even parity so set position 1 to a 0: `0 _ 1 _ 0 0 1 _ 1 0 1 0`
- Position 2 checks bits 3, 6, 7, 10, 11:  
`0 ? 1 _ 0 0 1 _ 1 0 1 0`. Odd parity so set position 2 to a 1: `0 1 1 _ 0 0 1 _ 1 0 1 0`
- Position 4 checks bits 5, 6, 7, 12:  
`0 1 1 ? 0 0 1 _ 1 0 1 0`. Odd parity so set position 4 to a 1: `0 1 1 1 0 0 1 _ 1 0 1 0`
- Position 8 checks bits 9, 10, 11, 12:  
`0 1 1 1 0 0 1 ? 1 0 1 0`. Even parity so set position 8 to a 0: `0 1 1 1 0 0 1 0 1 0 1 0`
- Code word: 011100101010 => 72A (x16).

## Hamming Decoder includes Error-Correction:

In the decoder side, the same process of hamming is done. i.e., the parity bits are regenerated using the same encoding procedure.

i.e., P1 = XOR of bits (3, 5, 7, 9, 11)

P2 = XOR of bits (3, 6, 7, 10, 11)

P4 = XOR of bits (5, 6, 7, 12)

P8 = XOR of bits (9, 10, 11, 12)

Now, these bits are XORed with corresponding bits in the message word we have now, to check if they are the same or not.

XOR gives 1 if the two inputs are not same.

We give the 'OR' of all parity bits generated in the receiver side to a variable 'temperror'.

## Serial/Parallel UART with 1-bit Error Correction using Hamming Code

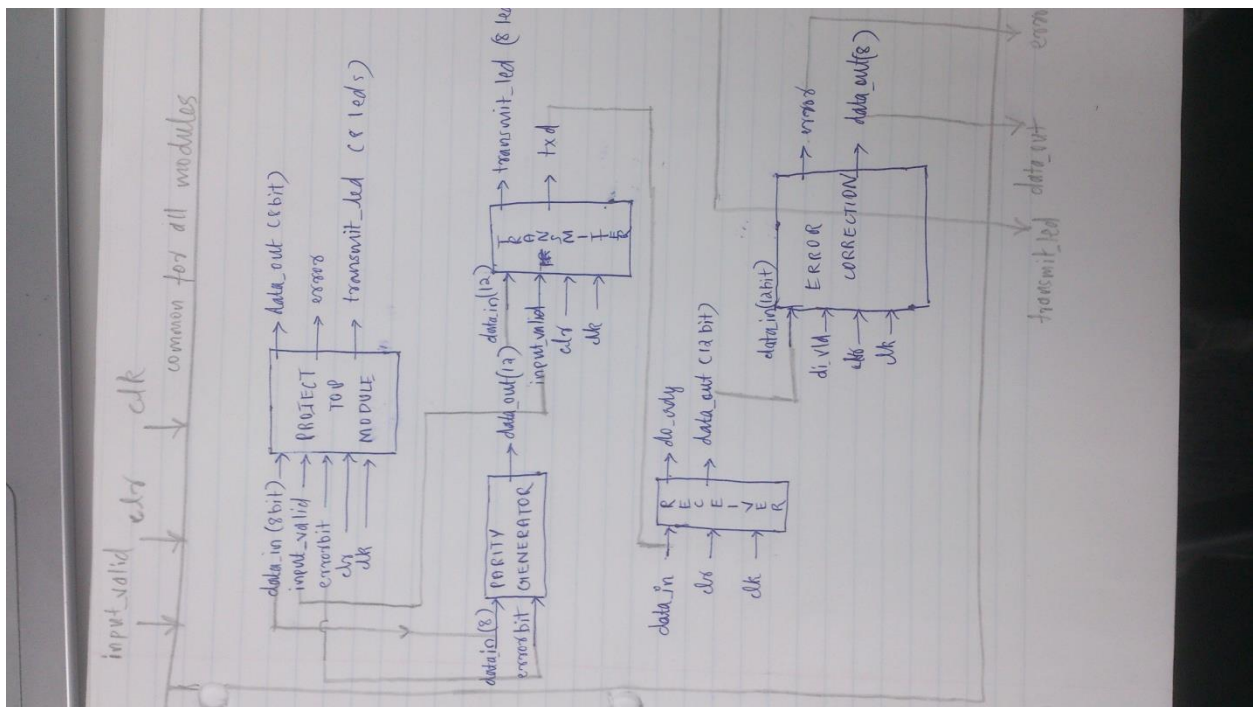
If temperror is '1', an error has occurred and has to be corrected.

The location of error can be found out easily and corrected by just toggling the data at the location of error.

## Baud Rate Generator Unit

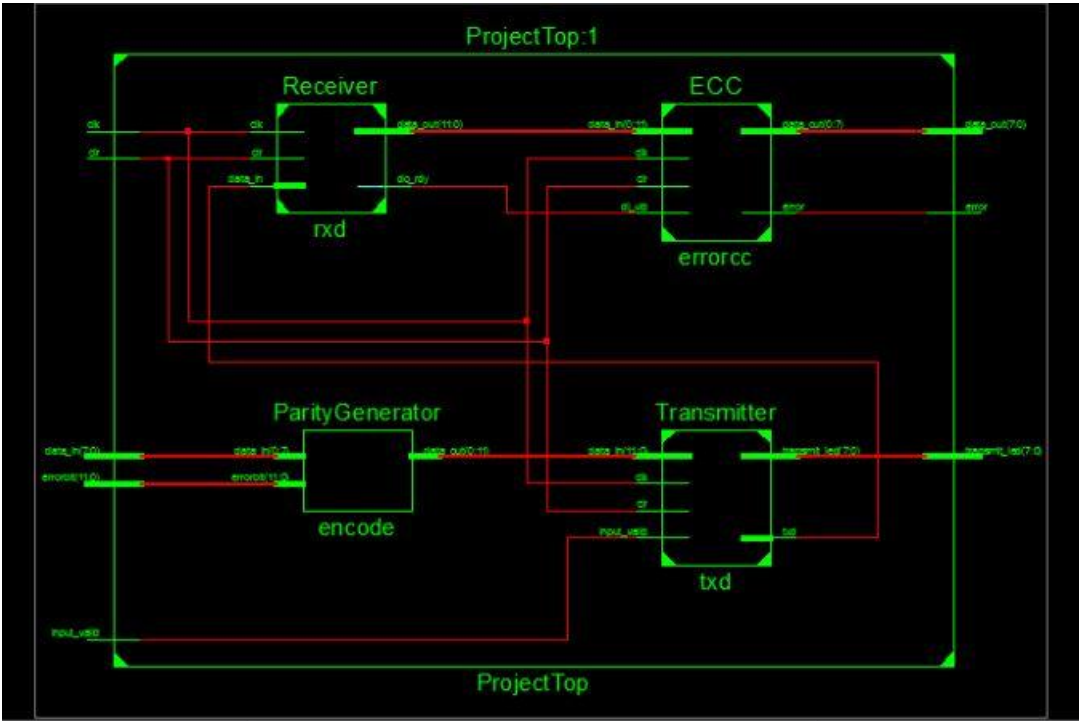
A programmable baud rate generator unit is considered in UART to produce desired frequencies for communication. By " $\text{baudclock} = \text{systemclock} / \text{baud} * 16 * \text{divisor}$ ", the output frequency of the baud rate generator, is determined. By changing the divisor value, frequency of baudclock can be changed to required value. Our baudclock gives out a frequency of **2 Hz**.

### HAND DRAWN RTL:-



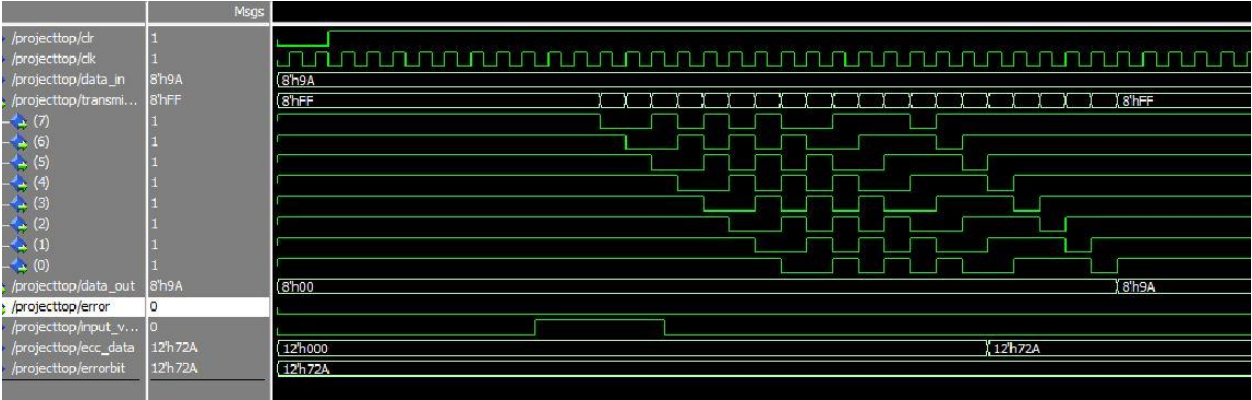
# Serial/Parallel UART with 1-bitError Correction using Hamming Code

## RTL FROM XILINX:-



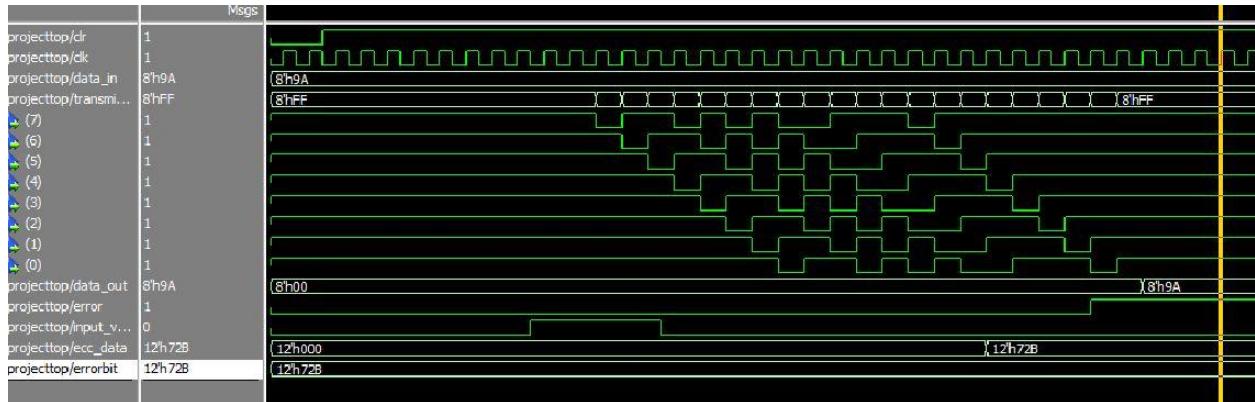
## SCREENSHOTS:-

### 1)functional simulation (without error introduced)

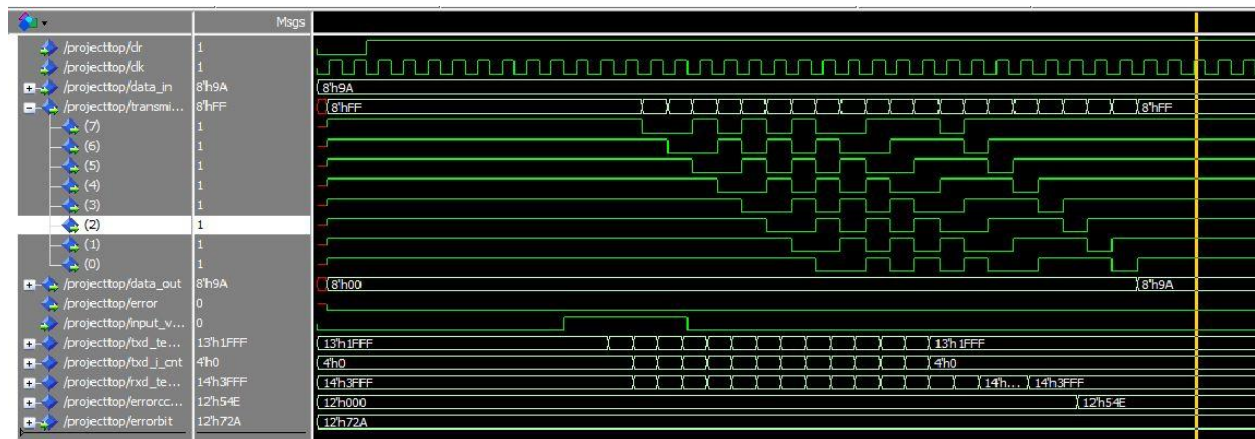


# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

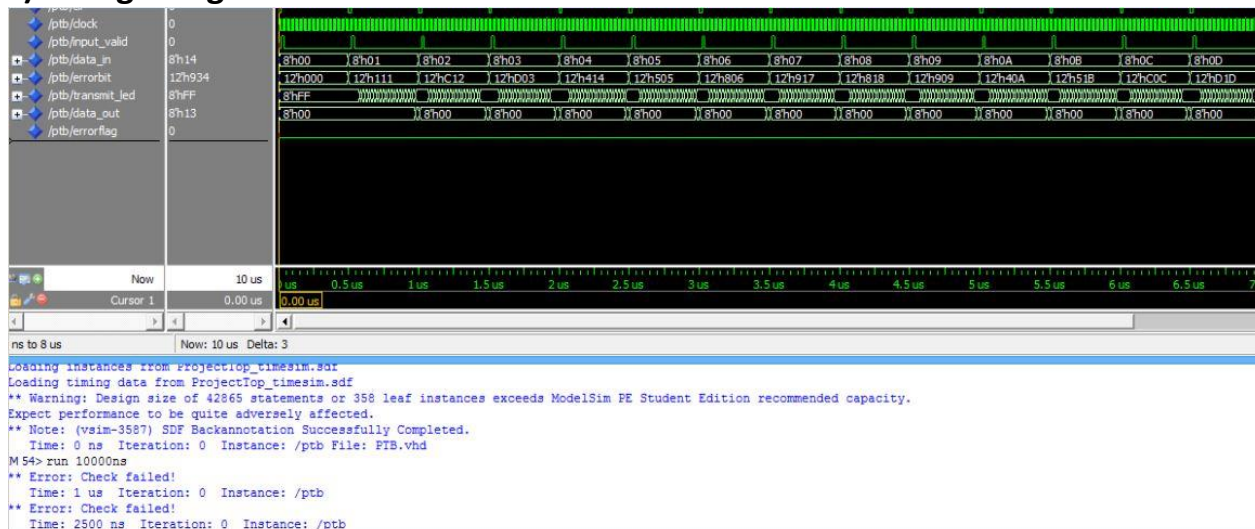
## 2)functional simulation (with an error introduced)



## 3)timing simulation without any error introduced

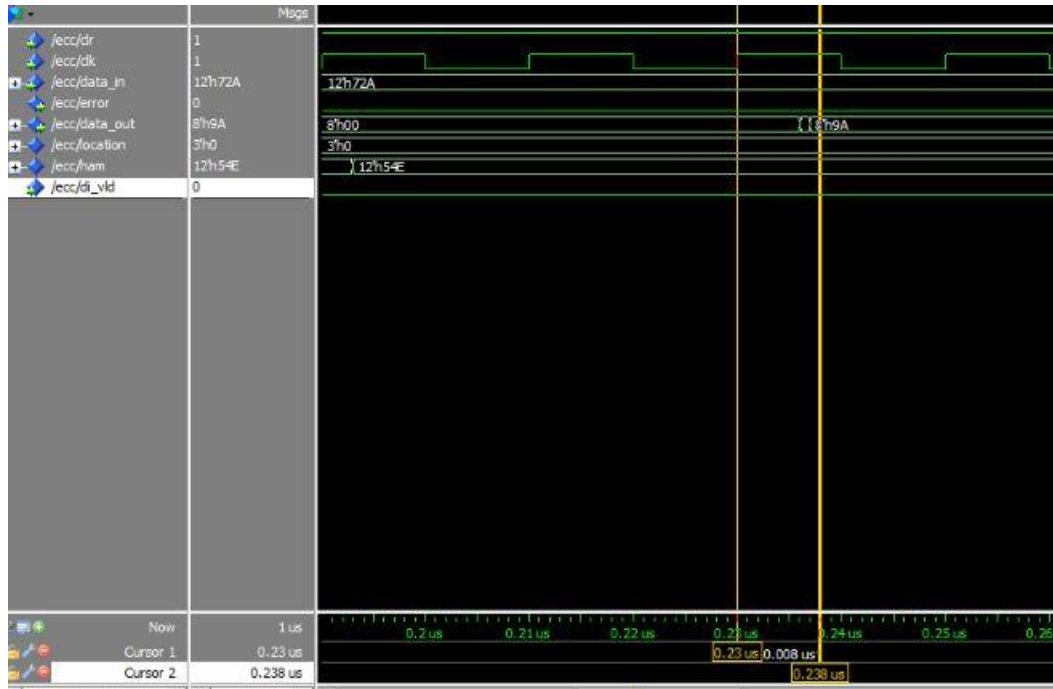


## 4)timing using testbench



# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

## 5) maximum frequency of operation



Max delay = 8ns

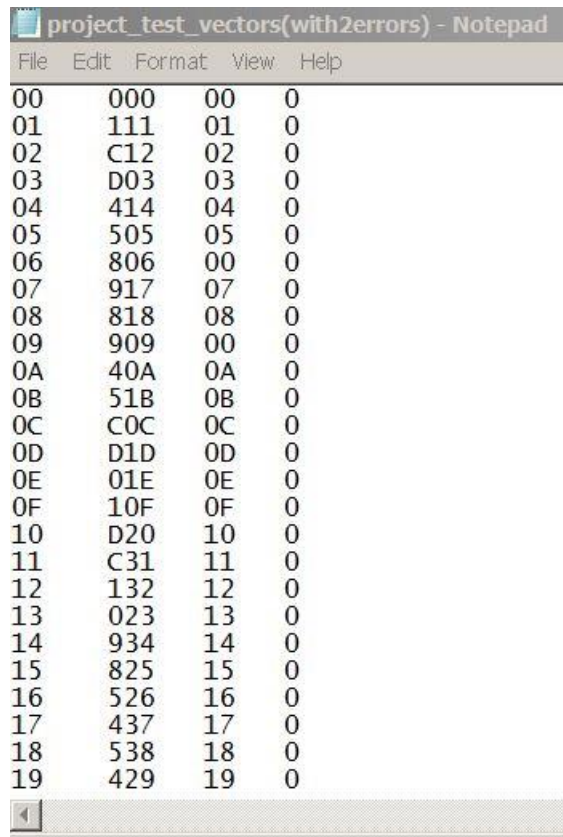
Max frequency = 125MHz

And board is only 100Mhz



# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

6) test vector file from which data inputs were read:



00	000	00	0
01	111	01	1
02	C12	02	0
03	D03	03	0
04	414	04	0
05	505	05	0
06	806	00	0
07	917	07	0
08	818	08	0
09	909	00	0
0A	40A	0A	0
0B	51B	0B	0
0C	C0C	0C	0
0D	D1D	0D	0
0E	01E	0E	0
0F	10F	0F	0
10	D20	10	0
11	C31	11	0
12	132	12	0
13	023	13	0
14	934	14	0
15	825	15	0
16	526	16	0
17	437	17	0
18	538	18	0
19	429	19	0

## Serial/Parallel UART with 1-bit Error Correction using Hamming Code

**7)output text file to which it was written:**

A screenshot of a Windows Notepad application window titled "project\_tb\_output(with2errors) - Notepad". The menu bar includes File, Edit, Format, View, and Help. The text area contains a series of lines: 0, 0, 0, 0|, 0, 0, 3500\_00\_06, 0, 0, 5000\_00\_09, followed by a vertical column of 17 zeros. A scroll bar is visible at the bottom left.

### RESOURCE UTILIZATION:-

Number of errors: 0

Number of warnings: 0

### Slice Logic Utilization:

Number of Slice Registers: 133 out of 18,224 1%

Number used as Flip Flops: 133

Number used as Latches: 0

Number used as Latch-thrus: 0

Number used as AND/OR logics: 0

Number of Slice LUTs: 138 out of 9,112 1%

Number used as logic: 127 out of 9,112 1%

Number using O6 output only: 57

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

Number using O5 output only:	39
Number using O5 and O6:	31
Number used as ROM:	0
Number used as Memory:	0 out of 2,176 0%
Number used exclusively as route-thrus:	11
Number with same-slice register load:	8
Number with same-slice carry load:	3
Number with other load:	0

## Slice Logic Distribution:

Number of occupied Slices:	46 out of 2,278 2%
Number of MUXCYs used:	52 out of 4,556 1%
Number of LUT Flip Flop pairs used:	145
Number with an unused Flip Flop:	43 out of 145 29%
Number with an unused LUT:	7 out of 145 4%
Number of fully used LUT-FF pairs:	95 out of 145 65%
Number of unique control sets:	11
Number of slice register sites lost to control set restrictions:	43 out of 18,224 1%

A LUT Flip Flop pair for this architecture represents one LUT paired with one Flip Flop within a slice. A control set is a unique combination of clock, reset, set, and enable signals for a registered element. The Slice Logic Distribution report is not meaningful if the design is over-mapped for a non-slice resource or if Placement fails.

## IO Utilization:

Number of bonded IOBs:	32 out of 232 13%
Number of LOCed IOBs:	32 out of 32 100%

# Serial/Parallel UART with 1-bitError Correction using Hamming Code

## Specific Feature Utilization:

Number of RAMB16BWERs:	0 out of	32	0%
Number of RAMB8BWERs:	0 out of	64	0%
Number of BUFIO2/BUFIO2_2CLKs:	0 out of	32	0%
Number of BUFIO2FB/BUFIO2FB_2CLKs:	0 out of	32	0%
Number of BUFG/BUFGMUXs:	2 out of	16	12%
Number used as BUFGs:	2		
Number used as BUFGMUX:	0		
Number of DCM/DCM_CLKGENs:	0 out of	4	0%
Number of ILOGIC2/ISERDES2s:	0 out of	248	0%
Number of IODELAY2/IODRP2/IODRP2_MCBs:	0 out of	248	0%
Number of OLOGIC2/OSERDES2s:	0 out of	248	0%
Number of BSCANs:	0 out of	4	0%
Number of BUFHs:	0 out of	128	0%
Number of BUFPLLs:	0 out of	8	0%
Number of BUFPLL_MCBs:	0 out of	4	0%
Number of DSP48A1s:	0 out of	32	0%
Number of ICAPs:	0 out of	1	0%
Number of MCBs:	0 out of	2	0%
Number of PCILOGICSEs:	0 out of	2	0%
Number of PLL_ADVs:	0 out of	2	0%
Number of PMVs:	0 out of	1	0%
Number of STARTUPs:	0 out of	1	0%
Number of SUSPEND_SYNCs:	0 out of	1	0%

Average Fanout of Non-Clock Nets: 3.14

Peak Memory Usage: 268 MB

# **Serial/Parallel UART with 1-bit Error Correction using Hamming Code**

Total REAL time to MAP completion: 18 secs

Total CPU time to MAP completion: 15 secs

## **Comment on timing report:**

According to the timing report, the total delay present in the design is 8 nanoseconds, through 1 level of logic, with 99% of the time spent on logic, and 1% spent on route.

## **CHALLENGES FACED DURING PROJECT :**

A) Problems faced by us during the project were different we wanted to output a faster transmission by transmitting on every edge of the clock but when we did that it couldn't be implemented on the FPGA because the board can only work one complete cycle of the clock and so we had to change our code completely.

B) We then thought of using the switch as our clock so that the transmission could be shown clearly visible but soon realized that we were not capable of doing that because then our 7 segment display wouldn't work. Thus, then we changed our clock by using a baud generator which uses the on-board oscillator which is reduced to from 100MHz to 2Hz.

C) We also had issues with the 7-segment display as it needed to be clocked with the output and the error-bit and it was taken care of appropriately by using our baud generator.

D) We had some problems with the state machines but even they were taken care of in the end.

## **YOUTUBE VIDEO LINK:**

Youtube video => <https://www.youtube.com/watch?v=hNlQu1dE688&feature=youtu.be>

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

VHDL CODE:

PARITY GENERATOR BLOCK:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; --use CONV_INTEGER

entity ParityGenerator is
port(data_in:   in std_logic_vector(0 to 7);
      errorbit:   in std_logic;
      data_out:   out std_logic_vector(0 to 11));
end ParityGenerator;

architecture ParityGeneratorBehavioral of ParityGenerator is
signal ham : std_logic_vector(0 to 11);
begin
ham(0) <= ham(2) XOR ham(4) XOR ham(6) XOR ham(8) XOR ham(10);
ham(1) <= ham(2) XOR ham(5) XOR ham(6) XOR ham(9) XOR ham(10);
ham(2) <= data_in(0);
ham(3) <= ham(4) XOR ham(5) XOR ham(6) XOR ham(11);
ham(4) <= data_in(1);
ham(5) <= data_in(2);
ham(6) <= data_in(3);
ham(7) <= ham(8) XOR ham(9) XOR ham(10) XOR ham(11);
ham(8) <= data_in(4);
ham(9) <= data_in(5);
ham(10) <= data_in(6);
ham(11) <= data_in(7);
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
data_out <= ham OR ("00000000000" & errorbit);  
end ParityGeneratorBehavioral;
```

TOP MODULE:

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL; --use CONV_INTEGER  
  
entity ProjectTop is  
    port(data_in:   in std_logic_vector(7 downto 0);  
          input_valid: in std_logic;  
          errorbit:   in std_logic;  
          clr:        in std_logic;  
          clk:        in std_logic;  
          data_out: out std_logic_vector (7 downto 0);  
          error:      out std_logic;  
          transmit_led: out std_logic_vector (7 downto 0)  
    );  
end ProjectTop;
```

architecture ProjectTopBehavioral of ProjectTop is

```
    -- Encoder components  
    component ParityGenerator is  
        port(data_in:   in std_logic_vector(0 to 7);  
              errorbit:   in std_logic;  
              data_out:   out std_logic_vector(0 to 11));  
    end component ParityGenerator;
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

-- Transmitter components

component Transmitter is

```
port(data_in:  in std_logic_vector(11 downto 0);
      input_valid: in std_logic;
      clr:      in std_logic;
      clk:      in std_logic;
      transmit_led: out std_logic_vector (7 downto 0);
      txd:      out std_logic);
end component Transmitter;
```

-- Receiver components

component Receiver is

```
port(data_in:  in std_logic;
      clr:      in std_logic;
      clk:      in std_logic;
      do_rdy:    out std_logic;
      data_out: out std_logic_vector (11 downto 0));
end component Receiver;
```

-- ECC components

component ECC is

```
port(data_in:  in std_logic_vector(0 to 11);
      di_vld:  in std_logic;
      clr:      in std_logic;
      clk:      in std_logic;
      error:    out std_logic;
      data_out: out std_logic_vector(0 to 7)
);
```



# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
end component ECC;

signal tx_data: std_logic;
signal en_data: std_logic_vector (0 to 11);
signal ecc_data: std_logic_vector (0 to 11);
signal di_flag: std_logic;

begin

encode: ParityGenerator port map (data_in => data_in, data_out => en_data, errorbit => errorbit);

txd: Transmitter port map (clk => clk, clr => clr, data_in => en_data, input_valid => input_valid,
transmit_led => transmit_led, txd => tx_data);

rxd: Receiver port map (clk => clk, clr => clr, data_in => tx_data, data_out => ecc_data, do_rdy =>
di_flag);

errorcc: ECC port map (clk => clk, clr => clr, data_in => ecc_data, di_vld => di_flag, error => error,
data_out => data_out);

end ProjectTopBehavioral;
```

TRANSMITTER:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; --use CONV_INTEGER
```

entity Transmitter is

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
port(data_in: in std_logic_vector(11 downto 0);
      input_valid: in std_logic;
      clr: in std_logic;
      clk: in std_logic;
      transmit_led: out std_logic_vector (7 downto 0);
      txd: out std_logic);
end Transmitter;
```

architecture TransmitterBehavioral of Transmitter is

```
TYPE StateType IS (ST_IDLE, ST_DATA_IN, ST_TRANSMIT);
```

```
SIGNAL state : StateType;
```

```
SIGNAL i_cnt: STD_LOGIC_VECTOR(3 DOWNT0 0);
```

```
SIGNAL temp_data: STD_LOGIC_VECTOR(12 DOWNT0 0);
```

```
SIGNAL temp_txd: STD_LOGIC;
```

```
SIGNAL led0: std_logic;
```

```
SIGNAL led1: std_logic;
```

```
SIGNAL led2: std_logic;
```

```
SIGNAL led3: std_logic;
```

```
SIGNAL led4: std_logic;
```

```
SIGNAL led5: std_logic;
```

```
SIGNAL led6: std_logic;
```

```
SIGNAL led7: std_logic;
```

```
begin
```

```
--state
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
process(clr, clk)
begin
    if (clr = '0') then
        state <= st_idle;
    elsif (clk'event and clk = '1') then
        case state is
            when st_idle =>
                if(input_valid = '1') then
                    state <= st_data_in;
                end if;
            when st_data_in =>
                state <= st_transmit;
            when st_transmit =>
                if (i_cnt = "1100") then
                    state <= st_idle;
                end if;
            end case;
        end if;
    end process;
```

--i counter

```
PROCESS(clr, clk)
BEGIN
    IF(clr='0') THEN i_cnt<="0000";
    ELSIF(clk'EVENT AND clk='1') THEN
        IF(state=st_transmit) THEN
            IF(i_cnt="1100") THEN i_cnt<="0000";
            ELSE i_cnt<=i_cnt+1;
        END IF;
    END IF;
```

## Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
        END IF;

    END IF;

END PROCESS;


process(clr, clk)
begin
    if(clr = '0') then
        temp_data <= "11111111111111";

    elsif (clk'event and clk = '1') then
        if (state = st_data_in) then
            temp_data <= data_in & '0';

        else
            temp_data(0) <= temp_data(1);
            temp_data(1) <= temp_data(2);
            temp_data(2) <= temp_data(3);
            temp_data(3) <= temp_data(4);
            temp_data(4) <= temp_data(5);
            temp_data(5) <= temp_data(6);
            temp_data(6) <= temp_data(7);
            temp_data(7) <= temp_data(8);
            temp_data(8) <= temp_data(9);
            temp_data(9) <= temp_data(10);
            temp_data(10) <= temp_data(11);
            temp_data(11) <= temp_data(12);
            temp_data(12) <= '1';
        end if;
    end if;
end process;
```

## Serial/Parallel UART with 1-bitError Correction using Hamming Code

```
                end if;

            end if;

        end process;

        txd <= temp_data(0);
        temp_txd <= temp_data(0);

        --transmitting data led
        process(clr, clk)
        begin
            if(clr = '0') then

                led0 <= '1';
                led1 <= '1';
                led2 <= '1';
                led3 <= '1';
                led4 <= '1';
                led5 <= '1';
                led6 <= '1';
                led7 <= '1';

            elsif (clk'event and clk = '1') then

                led0 <= temp_txd;
                led1 <= led0;
                led2 <= led1;
                led3 <= led2;
                led4 <= led3;
                led5 <= led4;
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
        led6 <= led5;

        led7 <= led6;

    end if;

end process;

transmit_led <= led0 & led1 & led2 & led3 & led4 & led5 & led6 & led7;

end TransmitterBehavioral;
```

RECEIVER:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL; --use CONV_INTEGER
```

entity Receiver is

```
    port(data_in:   in std_logic;
          clr:      in std_logic;
          clk:      in std_logic;
          do_rdy:   out std_logic;
          data_out: out std_logic_vector (11 downto 0));
```

end Receiver;

architecture ReceiverBehavioral of Receiver is

```
TYPE   StateType IS (ST_IDLE, ST_DATA_IN, ST_FLUSH);
```

```
SIGNAL state : StateType;
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
SIGNAL temp_data: STD_LOGIC_VECTOR(13 DOWNT0 0);
```

```
SIGNAL i_cnt: STD_LOGIC_VECTOR(3 DOWNT0 0);
```

```
begin
```

```
--state movements
```

```
process(clr, clk)
```

```
begin
```

```
    if (clr = '0') then
```

```
        state <= st_idle;
```

```
    elsif (clk'event and clk = '1') then
```

```
        case state is
```

```
            when st_idle =>
```

```
                if(data_in = '0') then
```

```
                    state <= st_data_in;
```

```
                end if;
```

```
            when st_data_in =>
```

```
                if (i_cnt = "1100") then
```

```
                    state <= st_flush;
```

```
                end if;
```

```
            when st_flush =>
```

```
                state <= st_idle;
```

```
        end case;
```

```
    end if;
```

```
end process;
```

```
--i counter
```

```
PROCESS(clr, clk)
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
BEGIN

IF(clr='0') THEN i_cnt<="0000";

ELSIF(clk'EVENT AND clk='1') THEN

    IF(state=st_data_in) THEN

        IF(i_cnt="1100") THEN i_cnt<="0000";

        ELSE i_cnt<=i_cnt+1;

        END IF;

    END IF;

END IF;

END PROCESS;

--Listening

PROCESS(clr, clk)

BEGIN

    IF(clr='0') THEN

        temp_data <= "1111111111111111";

    ELSIF(clk'EVENT AND clk='1') THEN

        IF (state = st_data_in) THEN

            temp_data(0) <= data_in;

            temp_data(1) <= temp_data(0);

            temp_data(2) <= temp_data(1);

            temp_data(3) <= temp_data(2);

            temp_data(4) <= temp_data(3);

            temp_data(5) <= temp_data(4);

            temp_data(6) <= temp_data(5);

            temp_data(7) <= temp_data(6);

            temp_data(8) <= temp_data(7);

            temp_data(9) <= temp_data(8);

            temp_data(10) <= temp_data(9);
```



## Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
temp_data(11) <= temp_data(10);
temp_data(12) <= temp_data(11);
temp_data(13) <= temp_data(12);
ELSIF (state = st_flush) THEN
temp_data <= "11111111111111";
END IF;
END IF;
END PROCESS;
```

--Valid Packet

```
PROCESS(clr, clk)
```

```
BEGIN
```

```
IF(clr='0') THEN
```

```
data_out <= "00000000000000";
```

```
ELSIF(clk'EVENT AND clk='1') THEN
```

```
IF(state = st_flush) THEN
```

```
data_out(0) <= temp_data(12);
```

```
data_out(1) <= temp_data(11);
```

```
data_out(2) <= temp_data(10);
```

```
data_out(3) <= temp_data(9);
```

```
data_out(4) <= temp_data(8);
```

```
data_out(5) <= temp_data(7);
```

```
data_out(6) <= temp_data(6);
```

```
data_out(7) <= temp_data(5);
```

```
data_out(8) <= temp_data(4);
```

```
data_out(9) <= temp_data(3);
```

```
data_out(10) <= temp_data(2);
```

```
data_out(11) <= temp_data(1);
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
        END IF;

    END IF;

END PROCESS;

--Data out ready
PROCESS(clr, clk)
BEGIN
    IF(clr='0') THEN
        do_rdy <= '0';
    ELSIF(clk'EVENT AND clk='1') THEN
        IF(state = st_flush) THEN
            do_rdy <= '1';
        ELSE
            do_rdy <= '0';
        END IF;
    END IF;
END IF;

END PROCESS;

end ReceiverBehavioral;

ERROR CORRECTION BLOCK:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; --use CONV_INTEGER

entity ECC is
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
port(data_in: in std_logic_vector(0 to 11);
      di_vld: in std_logic;
      clr: in std_logic;
      clk: in std_logic;
      error: out std_logic;
      data_out: out std_logic_vector(0 to 7)
);

end ECC;

architecture ECCBehavioral of ECC is

TYPE StateType IS (ST_IDLE, ST_DATAIN, ST_CHECK, ST_ECC, ST_DATAOUT);
SIGNAL state : StateType;

signal ham : std_logic_vector(0 to 11);
signal parity: std_logic_vector(0 to 3);
signal location: std_logic_vector(0 to 3);
signal temperror: std_logic;

begin
--state
process(clr, clk)
begin
    if (clr = '0') then
        state <= st_idle;
    elsif (clk'event and clk = '1') then
        case state is
            when st_idle =>
                if(di_vld = '1') then
```

## Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```

        state <= st_datain;
    end if;
    when st_datain =>
        state <= st_check;
    when st_check =>
        if(temperror = '1') then
            state <= st_ecc;
        else
            state <= st_dataout;
        end if;
    when st_ecc =>
        state <= st_dataout;
    when st_dataout =>
        state <= st_idle;
    end case;
end if;
end process;

--load data
PROCESS(clr, clk)
BEGIN
    IF(clr='0') THEN
        ham <= "000000000000";
    ELSIF(clk'EVENT AND clk='1') THEN
        IF(state=st_datain) THEN
            ham <= data_in;
        ELSIF(state=st_ecc) THEN
            ham(conv_integer(location(3) & location(2) & location(1) & location(0))-1) <=
NOT (ham(conv_integer(location(3) & location(2) & location(1) & location(0))-1));
        end if;
    end if;
end process;
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
END IF;

END IF;

END PROCESS;


--Compute Parity
parity(0) <= ham(2) XOR ham(4) XOR ham(6) XOR ham(8) XOR ham(10);
parity(1) <= ham(2) XOR ham(5) XOR ham(6) XOR ham(9) XOR ham(10);
parity(2) <= ham(4) XOR ham(5) XOR ham(6) XOR ham(11);
parity(3) <= ham(8) XOR ham(9) XOR ham(10) XOR ham(11);


--Compute Error bit location
location(0) <= parity(0) XOR ham(0);
location(1) <= parity(1) XOR ham(1);
location(2) <= parity(2) XOR ham(3);
location(3) <= parity(3) XOR ham(7);


--error flag
temperror <= location(0) OR location(1) OR location(2) OR location(3);


--check error flag
PROCESS(clr, clk)
BEGIN
    IF(clr='0') THEN
        error <= '0';
    ELSIF(clk'EVENT AND clk='1') THEN
        IF(state=st_check) THEN
            error <= temperror;
        END IF;
    END IF;
END IF;
```

# Serial/Parallel UART with 1-bit Error Correction using Hamming Code

```
END PROCESS;

--dataout
PROCESS(clr, clk)
BEGIN
    IF(clr='0') THEN
        data_out <= "00000000";
    ELSIF(clk'EVENT AND clk='1') THEN
        IF(state=st_dataout) THEN
            data_out(0) <= ham(2);
            data_out(1) <= ham(4);
            data_out(2) <= ham(5);
            data_out(3) <= ham(6);
            data_out(4) <= ham(8);
            data_out(5) <= ham(9);
            data_out(6) <= ham(10);
            data_out(7) <= ham(11);
        END IF;
    END IF;
END PROCESS;

end ECCBehavioral;
```