Proceedings of the Fifth International Conference on Communication and Electronics Systems (ICCES 2020)
IEEE Conference Record # 48766; IEEE Xplore ISBN: 978-1-7281-5371-1

# *Design of UART Module using ASMD Technique*

Sowmya K B
Department of Electronics and Communication Engineering
R.V college of Engineering
Bangalore, India.
kb.sowmya@gmail.com

Shreyans Gomes
Department of Electronics and Communication Engineering
R.V college of Engineering
Bangalore, India.
shreyansgomes.ec17@rvce.edu.in

Vishnusai Reddy Tadiparthi
Department of Electronics and Communication Engineering
R.V college of Engineering
Bangalore, India.
vishnusaireddyt.ec17@rvce.edu.in

***Abstract* - Universal Asynchronous Receiver Transmitter (UART) is an integrated circuit, which is commonly included in microcontrollers and it is usually operated at a baud rate of 20Mbps, which is usually achieved by using a clock of 20MHz. Due to its advantages such as high reliability, long-distance range, and low cost, it is widely used in the data communication process, especially in serial communications over a computer or peripheral device serial port. As indicated or hinted by the designation of the term 'universal', the format and transmission speeds of data are configurable to achieve the required operating condition(s). FIFO (First-In-First-Out) technique is used to store the data temporarily during high-speed transmissions and it is also used for synchronization. Usually, the data is serially transmitted one by one into the channel. The output is taken at a sampling period of 10 clock cycles and parallel stored as the data arrives. The results obtained from this experiment indicate that the working of the proposed model can be correlated to or is similar to the results obtained from the theoretical approach in the form of frame arrangement and stacking of data. This paper proposes a change in the frame format to involving the microcontroller or microprocessor, which will control the operation of UART based on the extra bit(s) in the frame, to achieve option(s) for power saving. To design such a circuit at a gate level is tedious and consumes a large amount of time due to the increasing complexity of integrated circuit technology, hence the use of hardware description language such as Verilog HDL is becoming popular because it makes it easy to design a circuit of any complexity. Verilog is pretty different from VHDL. Verilog is not as verbose as VHDL, so that is why it is more compact and hardware modeling is better in it than VHDL, even though it has a lower level of programming constructs than VHDL.**

***Keywords – UART, Transmitter, Receiver, Control unit, Datapath unit.***

## I. INTRODUCTION

Data transmission is achieved through the use of UART via a serial port on the computer to transmit and receive data. Parallel-to-serial conversion is performed at the transmitter side and serial-to-parallel conversion is performed at the receiver side. Buffer(s) are present at both the transmitter and receiver, which are used to contain data, that is the data is accumulated here before being used for further processing. Generally, an 8-bit long format is used to send the data from the transmitter. The first bit is called Start bit, which is '0', and the bits which follow the start bits are data bits, whose length ranges between 5 and 7 bits. The ensuing bit is the parity bit. It is added as a checksum to check for any data corruption while it travels through a channel, In the end, a stop bit of '1' is added to indicate the end of the frame. The data travels in a frame in the above format through a channel. The data, which is captured at the receiver end, is sampled at a rate of 11 clock cycles per second. The captured data is then transferred to a shift register for accumulation. These both receiver and transmitter architectures are helpful in the successful implementation of the UART. It may also include different techniques to control the data flow such as the baud rate generator. Data conversion in UART happens at very high speed without loss of data. The data exchanged is of American Standard Code for Information Interchange (ASCII) format. Besides RS-232, RS-422 and RS-485 standards have been commonly applied to UART chips nowadays. This paper proposed a modified frame format, where it made use of hamming code for error detection and flags for indicating the required action to microprocessor like resend data or stop sending data etc. Di stands for data bit i, while Pi stands for parity bit i. P3 is the extra parity bit. The frame format is as shown below:

| Start bit | P 0 | P 1 | D 0 | P 2 | D 1 | D 2 | D 3 | P 3 | er | en | ack | Stop bit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Fig.1. The frame format for UART

The bits *en, ack,* and *er* are given as inputs by the microprocessor. The bit *en* is used by the receiver side to indicate the transmitter to halt transmission. This doesn't halt the transmission of data indefinitely. The microcontroller or microprocessor will wait for time t, the time required to transmit once from the transmitter to the receiver. This has to be coded into the microprocessor's program. This may be due to various reasons, such as an error is detected or the rate of processing of data is slower than the rate of transmission and

reception of data. The bit *er* is used to indicate that there was an error in the transmitted data. Although a hamming code generator is used to encode the data, there is still a chance that the data may be corrupted due to various factors, such as glitches or EMI, etc. At the receiver side, the program which is running on the microprocessor will carry out the detection of errors, based on the output of the hamming code decoder. The received data is sent back as a check to see if the received data is correct and an acknowledgment is sent to the receiver side. The *ack* is used by the microprocessor to indicate the transmitter that data is received properly without any error and that er bit is valid. The acknowledgment sent by the receiver during the first transmission is in the form of this *en* and *er* bits.

## II. LITERATURE SURVEY

Implementation and working of UART and frame format are important in retrieving the data bits after they are transmitted into the channel [2]. Understanding of UART at the circuit level, the inclusion of various flags to determine the status of data bits was also a motivation behind this paper [6]. The operation of transmitter and receiver and utilization of the clock dividing technique was also implemented [2]. Power analysis of a circuit helped us to understand the need of using minimal power when it is implemented on FPGA [1]. The different number of bits and its usage in UART frame format helped us to utilize its flexibility as much as possible [3].

## III. TRANSMITTER ARCHITECTURE

The processor provides the input signal and the output in a serial data stream, a status signal, and two error signals. The transmitter architecture is composed of a hamming code generator, a control unit, a data register, a data shift register, and a status register which is used to keep track of the number of bits that are transmitted and it is included in the datapath unit. The controller has two types of inputs namely external input and internal input. *Load_T_dreg*, *By_ready, T_by, Clock, rst_b* are classified as external inputs, while *BC_lt_BCmax* is an internal input. The outputs of this control unit namely *Load_T_DR, Load_T_shftreg, start, shift,* and *clear* are purely internal, and they serve as internal inputs to datapath unit while *D_Bus* serves as an external input to datapath unit with *Serial_out* being the external output. This output is the one that transmits the parallel data into the channel in a serial fashion (as shown in figure 2).

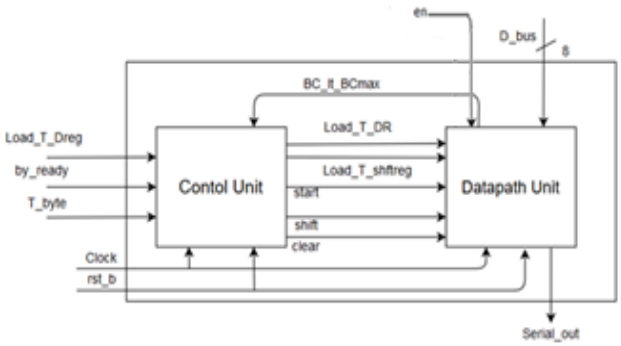


Fig.2. Transmitter Architecture Block Diagram

It works based on the ASMD chart. Here, the transmitter goes through three idle states, *waiting* and *sending*. These states are saved as 01, 11, and 10 respectively and the next state is achieved by performing left shift operation on the current state once and the added bit, the last is the output of XOR operation to the bits of the previous stage. When the active low reset signal, *rst_b,* is asserted, the state machine enters an *idle* state, *bi_count* is flushed and *T_shftreg* is loaded with all 1's. In this state, if an active edge of Clock occurs while the *rst_b* is high, the system checks for *Load_T_dreg*. When this state is asserted by the external host, the output signal *Load_T_DR* will load the content of *D_Bus* into *T_d_reg*. After this, at the positive edge of clock pulse the machine will check for *By_ready* signal. When the system asserts this signal at the positive edge of clock cycle machine enters into *Load_T_shftreg* where the data register *T_shftreg* is filled with the data from *T_dreg* with an additional *er, en* and stop bit. After that, the state is driven to *waiting* unless *T_by* is assessed at the positive edge of the clock the state remains the same. Once *T_by* is asserted at the positive edge of the clock the state is driven to *sending* before which a *start* action will take place and data is continuously transmitted until the counter reaches *BC_lt_BCmax* (maximum allowable value). Once this maximum value is reached, the stop bit is appended and the registers are flushed while the state is returned to *idle* (as shown in figure 3).

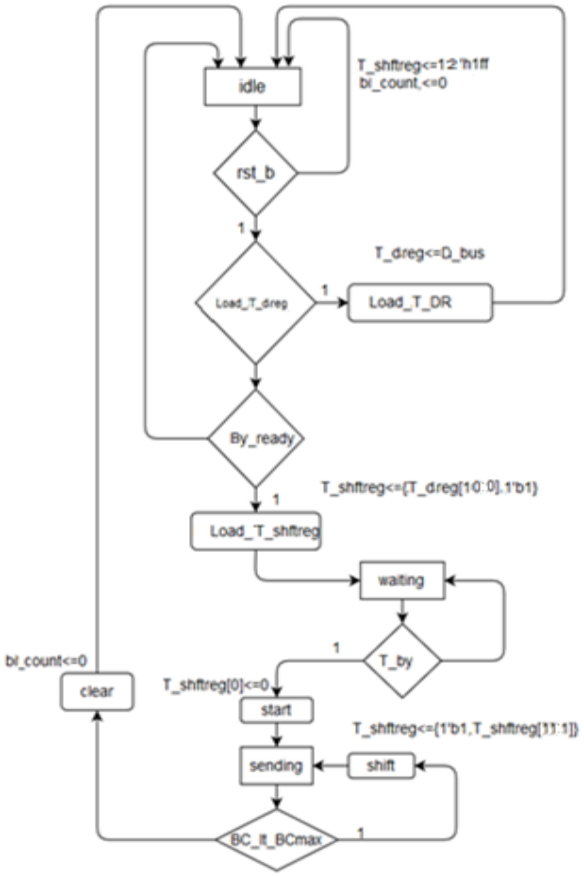


Fig.3. ASMD flow diagram for Transmitter architecture

## IV. RECEIVER ARCHITECTURE

The UART receiver receives the serial bit stream of data, removes the Start bit, reads the flag bits, decodes the hamming code and transfers the received data in a parallel format to a storage register connected to the host data bus and the rest of the outputs, such a syndrome vector and *en* and *er* to the microcontroller. There must be separate Clock and reset signal at the receiver side, which must be synchronized with the transmitter's clock. This can be done in various ways. One such method is the use of a Phase-locked loop. Here, the data is taken as samples by sampling the received data stream at a regular interval of time. The sampling rate is 11 times the clock period. It also has a control unit and a datapath unit. The control unit has external inputs which are *Samp_clock, rst_b, read_not_ready_in*. The outputs of this control unit, which are *clr_Samp_counter, inc_Samp_counter, clr_b_counter, inc_b_counter, shift,* and *load,* serve as inputs to the datapath unit, which also has an external input called *Serial_in*. The output from the datapath unit is *Ser_in, SC_eq, SC_lt,* and *BC_eq*. Along with this it also has an external output named *RCV_dreg,* which gives the data output in parallel form (as shown in figure 4). The *RCV_dreg* is sent to the hamming code decoder and the output syndrome vector and the received data and outputs are sent to the microprocessor.

It also has three states like three transmitter side and they are operated similarly. They are named as *idle, starting* and *receiving*. In the idle state when *rst_b* is not equal to 0 and *SC_in* is 1, the state machine enters the starting state, where it checks for continuous 0's. When 4 consecutive 0 appears, the machine recognizes it as a start bit and enters the *receiving* state. This can be handled by *inc_Samp_counter*. Once it enters receiving state, *clr_Samp_counter* is enabled to clear the sample counter and bits are taken after every 11 clock cycles, this can be handled by *inc_Samp_counter* and *clr_Samp_counter*. To check the number of bits received *inc_bi_counter* is used to increment a counter and shift the bits in *RCV_shftreg*. Along with this, *BC_eq* is used to count the number of bits received

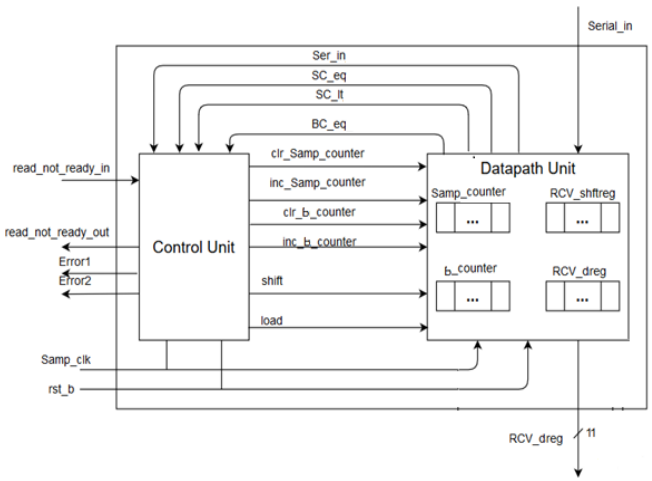


Fig.4. Receiver Architecture Block Diagram

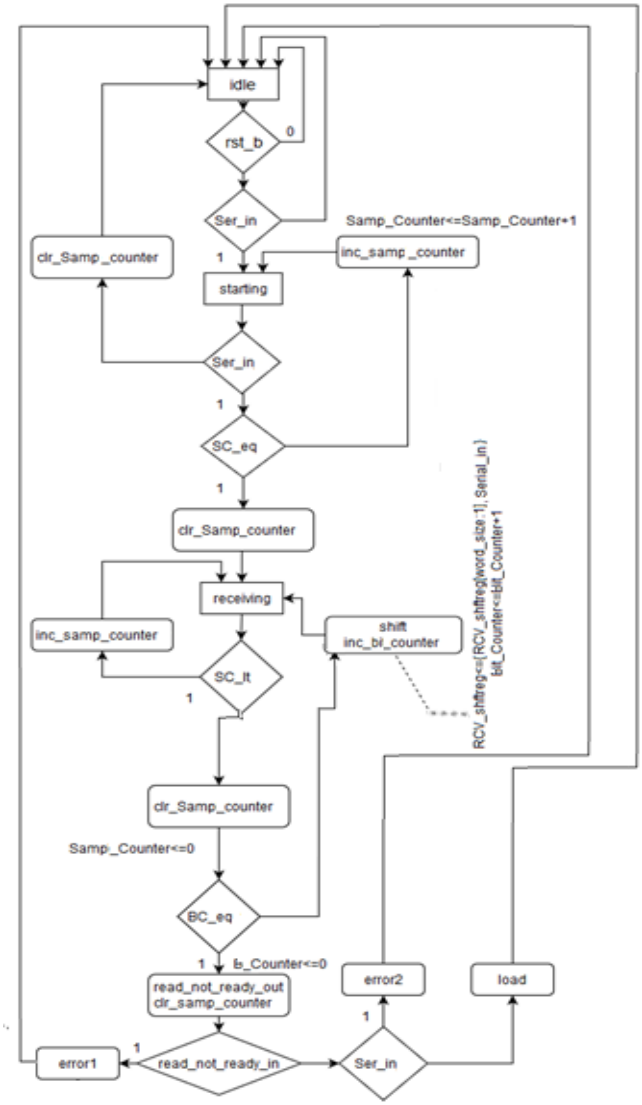


Fig.5. ASMD flow diagram for Receiver architecture

Once 11 bits are received it checks for signal from the host. If *read_not_ready_in* is enabled an error signal, *Error1,* is asserted and the state machine moves to the idle state. If it is not 1 and if *Ser_in* is 1 an error signal, *Error2,* is asserted, the state machine also moves to the idle state. If *Ser_in* is also zero then load signal will be asserted and the parallel data will be loaded into *RCV_dreg* which is the output passed to the processor and the state machine moves into the idle state (as shown in figure 5).

## V. RESULT ANALYSIS

### A. Transmitter Output

Here, it is observed that, when *rst_b* is 0 the system enters into reset mode where everything is set to initial condition. When *Load_T_Dreg* is 1 and *rst_b* is 1 during a positive

edge of the clock the system checks for *By_ready*. Similarly, when *By_ready* is 1 and *rst_b* is 1 at the positive edge of the clock, system checks for *T_by* and when *T_by* is 1 and *rst_b* is 1 at the positive edge of the clock the system starts transmitting data. The required data is encoded using a hamming code generator and is passed onto *d_bus*. The data present on the *d_bus* is transmitted serially on *serial_out*. The input is represented by *d_bus*. At 40ns a start bit of value 0 is sent followed by the data in the *d_bus* serially. After sending the data it is observed that a stop bit of 1 is sent to signal the end of the frame (as shown in figure 6). A slightly modified version of the ASMD diagram was used for testing of this ASMD diagram since the testing was done using software tools and no hardware was used.

Fig.6. Transmitter Output

### *B. Receiver Output*

In the receiver end, when a data is received at *Serial_in* it checks for 4 continuous 0's which will signal that a start bit received. The *Sample_clk* takes the input data at every 10$^{th}$ clock cycle. The data received is obtained serially at the input. *RCV_datareg* shows the loading of serially obtained data into parallel form by shifting it towards LSB bit by bit. This process takes place until the *BC_eq_8* becomes 1 and passes it to the next state. It is also seen that, when all the bits are transmitted it sends a signal *read_not_ready_out* to the host processor to stop sending data for a while. At the same time, an *Error2* signal is asserted for a while since the *Serial_in* data are 0. The zero-bit received at the end is stop bit which signals the end of the frame (as shown in figure 7). This data is sent to a hamming code decoder which will decode the data and send it to a microprocessor or microcontroller for further use.

## VI. RTL DIAGRAM AND SYNTHESIS

The RTL view for Transmitter and Receiver is generated and various information on design such as nets, cells, and I/O ports are tabulated. These values give us information regarding the resources utilized for the implementation of transmitter and receiver architectures. Also, they provide us the insight on minimum required cells, ports, nets, LUTs (Lookup tables), etc.

Fig.7. Receiver Output

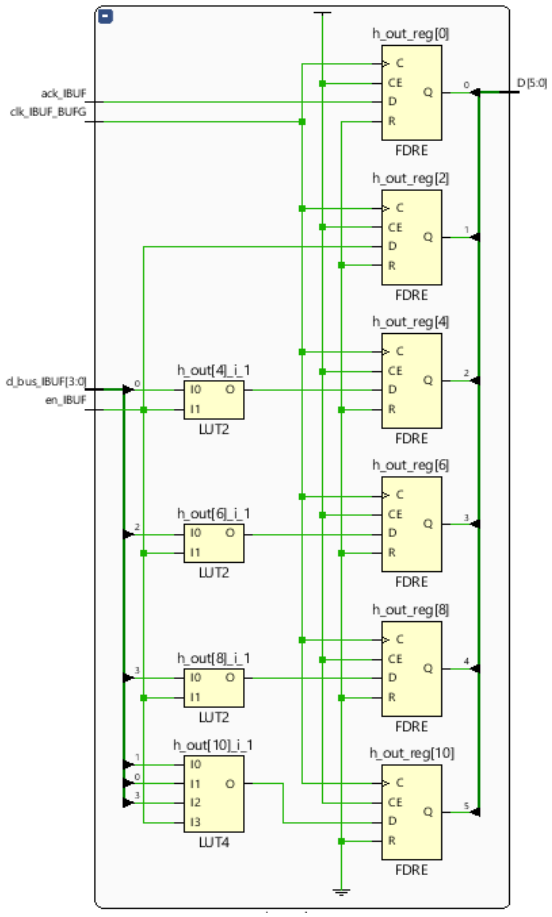


Fig.8. RTL Schematic of the hamming code generator

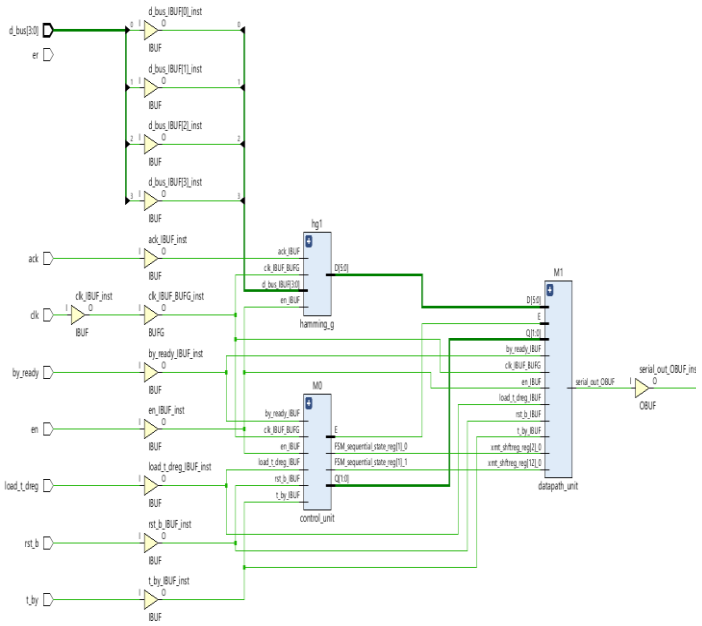

Fig.9. RTL Schematic of Transmitter

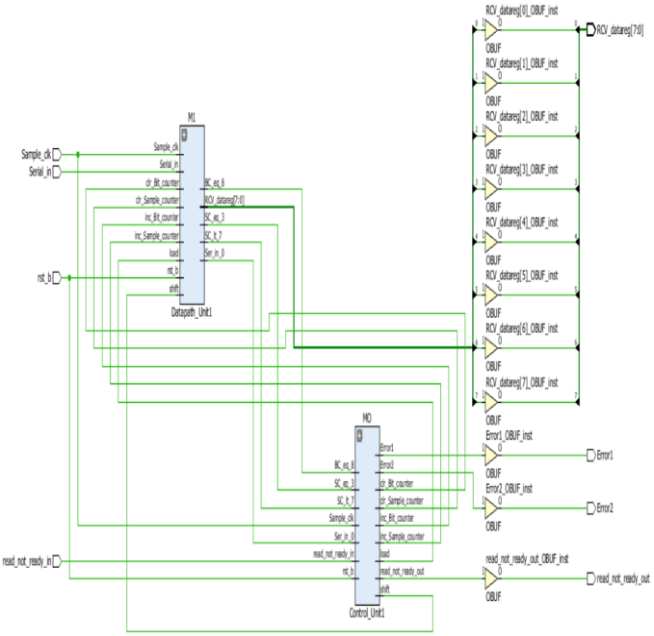

Fig.10. RTL Schematic of Receiver

9 Cells 15 Nets

| Name | Cell | Cell Pin Count |
|---|---|---|
| M0 | control_unit | 11 |
| M0/FSM_sequential_state[0]_i_1 | LUT5 | 6 |
| M0/FSM_sequential_state[1]_i_1 | LUT3 | 4 |
| M0/FSM_sequential_state[1]_i_2 | LUT1 | 2 |
| M0/FSM_sequential_state_reg[0] | FDCE | 5 |
| M0/FSM_sequential_state_reg[1] | FDCE | 5 |
| M0/xmt_datareg[10]_i_1 | LUT5 | 6 |
| M0/xmt_shftreg[10]_i_1 | LUT6 | 7 |
| M0/xmt_shftreg[12]_i_2 | LUT2 | 3 |

Fig. 11. Pin details of Transmitter's Control unit

23 Cells 41 Nets

| Name | Cell | Cell Pin Count |
|---|---|---|
| M1 | datapath_unit | 18 |
| M1/xmt_datareg_reg[0] | FDRE | 5 |
| M1/xmt_datareg_reg[10] | FDRE | 5 |
| M1/xmt_datareg_reg[2] | FDRE | 5 |
| M1/xmt_datareg_reg[4] | FDRE | 5 |
| M1/xmt_datareg_reg[6] | FDRE | 5 |
| M1/xmt_datareg_reg[8] | FDRE | 5 |
| M1/xmt_shftreg | LUT4 | 5 |
| M1/xmt_shftreg[0]_i_1 | LUT3 | 4 |
| M1/xmt_shftreg[10]_i_2 | LUT3 | 4 |
| M1/xmt_shftreg[12]_i_1 | LUT6 | 7 |
| M1/xmt_shftreg[2]_i_1 | LUT3 | 4 |
| M1/xmt_shftreg[4]_i_1 | LUT3 | 4 |
| M1/xmt_shftreg[6]_i_1 | LUT3 | 4 |
| M1/xmt_shftreg[8]_i_1 | LUT3 | 4 |
| M1/xmt_shftreg_inferred__0/i_ | LUT5 | 6 |
| M1/xmt_shftreg_reg[0] | FDRE | 5 |
| M1/xmt_shftreg_reg[10] | FDSE | 5 |
| M1/xmt_shftreg_reg[12] | FDRE | 5 |
| M1/xmt_shftreg_reg[2] | FDSE | 5 |
| M1/xmt_shftreg_reg[4] | FDSE | 5 |
| M1/xmt_shftreg_reg[6] | FDSE | 5 |
| M1/xmt_shftreg_reg[8] | FDSE | 5 |

Fig.12. Pin details of Transmitter's Datapath Unit

16 Cells 13 I/O Ports 36 Nets

| Name | Cell | Cell Pin Count |
|---|---|---|
| M0 | control_unit | 11 |
| M1 | datapath_unit | 18 |
| ack_IBUF_inst | IBUF | 2 |
| by_ready_IBUF_inst | IBUF | 2 |
| clk_IBUF_BUFG_inst | BUFG | 2 |
| clk_IBUF_inst | IBUF | 2 |
| d_bus_IBUF[0]_inst | IBUF | 2 |
| d_bus_IBUF[1]_inst | IBUF | 2 |
| d_bus_IBUF[2]_inst | IBUF | 2 |
| d_bus_IBUF[3]_inst | IBUF | 2 |
| en_IBUF_inst | IBUF | 2 |
| hg1 | hamming_g | 13 |
| load_t_dreg_IBUF_inst | IBUF | 2 |
| rst_b_IBUF_inst | IBUF | 2 |
| serial_out_OBUF_inst | OBUF | 2 |
| t_by_IBUF_inst | IBUF | 2 |

Fig. 13. Pin details of transmitter

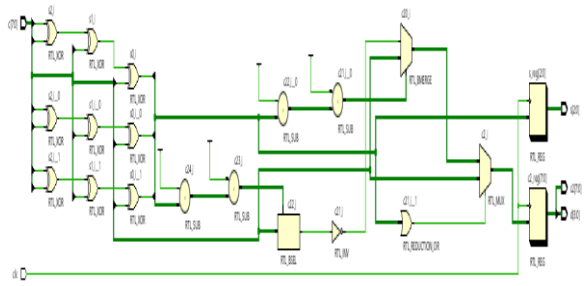

Fig.14. RTL Schematic of hamming code decoder

The following table gives information on cells, nets, I/O ports, LUTs and FFs used in Transmitter and Receiver architecture. Pin details and RTL schematic of the receiver's datapath and control unit are complicated and thus it's inclusion in this paper is not desirable, but the details regarding the cells, nets, I/O ports, LUTs and FFs are shown in TABLE 1.

TABLE 1. DESIGN UTILISATION INFORMATION

| Architecture | Cells | Nets | I/O ports | LUTs | FFs |
|---|---|---|---|---|---|
| Transmitter | 16 | 36 | 13 | 0 | 0 |
| Receiver | 42 | 114 | 37 | 14 | 27 |

Power consumption is a major factor in designing circuits. Excess power can lead to heating up of devices which is not desirable. The power consumption and its details are shown in figure 15. Here the power consumed by the clock signal and IO ports is 0.001W. The static power consumed by the device is 1.363W which will lead to the total consumption of 1.364W of power. The junction temperature is 51.9°C. The power supply specification is also shown in figure 15.

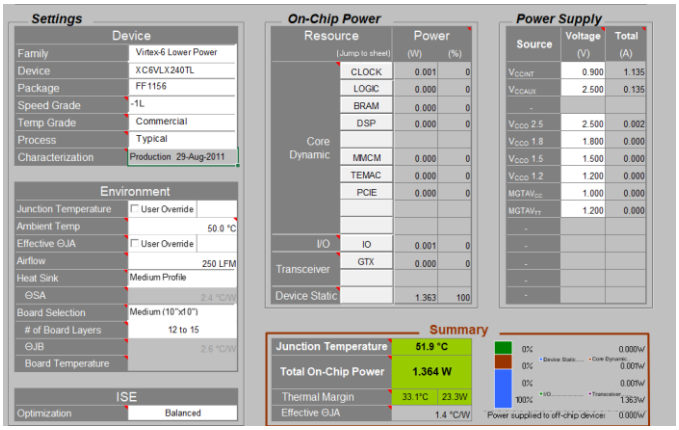


Fig.15. Power consumption details

## VII. CONCLUSION

The obtained result is quite similar to that of the theoretical approach. In the theoretical approach, the frame consists of one start bit, followed by 5-7 data bits, one parity bit, and one-stop bit. Here, the data are transmitted serially in almost the same fashion but with slightly more complex architecture(s) and frame format. On the receiver end similar to the theoretical approach the start bit and stop bit are dropped and the data is loaded parallelly into the receiver data register. The microprocessor is more involved in this design than in the general design. This design is not optimized for low power mode or power saving. The testing of the code should be done using hardware by making use of devices like FPGA. Further, there is more involvement of the microprocessor, so there is more load on the microprocessor. This implies that there is a need for microprocessors that have higher load handling capabilities or external peripherals that can handle this task. These modifications ensure that the data corruption is less and that the transmitter side and receiver side can send data and halt the transmission of data if required.

## REFERENCES

[1] M.Poorani and Mrs.R.Kurunjimalar, "Design implementation of UART and SPI in single FGPA", Intelligent Systems and Control (ISCO), 2016 10th International Conference, 03 November 2016, ISBN: 16429505, DOI: 10.1109/ISCO.2016.7726983.

[2] Ritesh Kumar Agrawal and Vivek Ranjan Mishra, "The Design of High Speed UART", Proceedings of 2013 IEEE Conference on Information and Communication Technologies, pp. 388–390 (2013).

[3] Nennie Farina Mahat, "Design of a 9-bit UART Module Based on Verilog HDL", IEEE-ICSE 2012 Proc., 2012, Kuala Lumpur, Malaysia.

[4] Swati Dubey and Amrita Singh, "A Review Paper on Implementation of UART", International Journal of Science, Engineering and Technology Research (IJSETR) Volume 5, Issue 5, May 2016, ISSN: 2278 – 7798.

[5] Dr. Garima Bandhawarkar Wakhle, Iti Aggarwal and Shweta Gaba, "Synthesis and Implementation of UART using VHDL Codes", International Symposium on Computer, Consumer and Control, 2012, DOI: 10.1109/IS3C.2012.10

[6] Naresh Patel, Vatsalkumar Patel, Vikaskumar Patel, "VHDL Implementation of UART with Status Register ", International Conference on Communication Systems and Network Technologies, 2012, DOI: 10.1109/IS3C.2012.10, CSNT 2012

[7] Mohd Yamani Idna Idris, Mashkuri Yaacob, Zaidi Razak, "A STUDY OF HARDWARE DESIGN AND OVERHEAD IN BUILT-IN-SELF-TEST ABLE UART", Winter International Synposium on Information and Communication Technologies, 2004, SOICT 2004.

[8] Ruchin, Chandan Mahto, Pawan Whig, "Design & Simulation of Dynamic UART Using Scan Path Technique (USPT)", International Journal of Electrical Electronics & Computer Science Engineering Special Issue - TeLMISR 2015, ISSN: 2348-2273, IJEECSE 2015.

[9] U. Nanda and S. K. Pattnaik, "Universal Asynchronous Receiver and Transmitter (UART)," 2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, 2016, pp. 1-5. DOI: 10.1109/ICACCS.2016.7586376.

[10] H. Chun-zhi, X. Yin-shuicand W. Lun-yao, "A universal asynchronous receiver transmitter design," 2011 International Conference on Electronics, Communications and Control (ICECC), Ningbo, 2011, pp 691-694. DOI: 10.1109/ICECC.2011.6066542.

[11] Y. Wang and K. Song, "A new approach to realize UART," Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology, Harbin, 2011, pp. 2749-2752. DOI: 10.1109/EMEIT.2011.6023602.

[12] Y. Fang and X. Chen, "Design and simulation of UART Serial Communication Module based on VHDL," 2011 3rd International Workshop on Intelligent Systems and Applications, Wuhan, 2011, pp. 1-4.DOI:10.1109/ISA.2011.5873448.