Journal Homepage: - www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)

Article DOI: 10.21474/IJAR01/21400
DOI URL: http://dx.doi.org/10.21474/IJAR01/21400

RESEARCH ARTICLE

DESIGN AND IMPLEMENTATION OF UART USING VERILOG: PARALLEL-IN-SERIAL-OUT AND SERIAL-IN-PARALLEL-OUT MODULES

Punam Chati[1], Ketan Shrirao[2], Hemant Pise[2] , Sagar Mallik[2] and Ayush Bobde[2]

1. Assistant Professor
2.Students Research Scholar, Department of Electronics and Telecommunications, Priyadarshini College of Engineering, Nagpur, India

Manuscript Info

Manuscript History
Received: 14 May 2025
Final Accepted: 17 June 2025
Published: July 2025

Key words:-
UART, Verilog, PISO, SIPO, Parallel-to-Serial Conversion, Serial-to-Parallel Conver.

Abstract

The Universal Asynchronous Receiver-Transmitter (UART) is a key component in digital communication, enabling efficient serial data transmission with minimal hardware overhead. This paper presents the design and implementation of a UART module using Verilog Hardware Description Language (HDL). The focus is on two essential component s: Parallel In Serial Out (PISO) and Serial In Parallel Out (SIPO) modul es, which facilitate data conversion for transmission and reception.The PISO module converts parallel data into a serialized format for transmis sion, while the SIPO module reconstructs serial data into parallel form upon reception. Both modules are designed with modularity and scalabi lity in mind, ensuring efficient resource utilization and adherence to communication protocols.To ensure robust operation, the design incorp orates start and stop bits, as well as clock synchronization to handle varying baud rates.The proposed design was verified through simulatio ns under diverse conditions, demonstrating accurate functionality and reliability. Synthesis results confirmed compliance with timing constrai nts and efficient hardware resource usage, making the design suitable for FPGA implementation. The design was implemented and synthesis in QUARUS II .This paper provides a comprehensive approach to UART design, offering insights into its practical applications in embed ded systems and digital communication. The modular design ensures ease of integration into complex systems while maintaining high performance and reliability

Introduction:-

The Universal Asynchronous Receiver-Transmitter (UART) facilitates serial communication between devices by converting data between parallel and serial forms. Its operation involves two main processes: data transmission and data reception. These processes rely on synchronization, framing, and optional error-checking mechanisms to ensure reliable communication.

Corresponding Author:- Punam Chati
Address:- **Assistant Professor**

1.**Data Transmission**:
The transmitter in UART converts parallel data into a serial bitstream using the Parallel-In-Serial-Out (PISO) method. Before transmission, the data is encapsulated into a frame that includes:

- Start Bit: A single bit, always set to 0 (low), that signals the beginning of the data frame to the receiver.
- Data Bits: The actual data to be transmitted, typically 5 to 8 bits per frame, depending on the configuration.
- Stop Bits: One or more bits, set to 1 (high), indicating the end of the data frame.

The serial data is transmitted over the TX (transmit) line. The transmitter ensures that the data rate matches the agreed baud rate to maintain synchronization with the receiver.

2. Data Reception:
The receiver in UART processes incoming serial data on the RX (receive) line using the Serial-In-Parallel-Out (SIPO) method. It performs the following tasks:

- Start Bit Detection: The receiver continuously monitors the RX line and begins decoding upon detecting a start bit (low signal).
- Data Bit Extraction: It reads the data bits at intervals defined by the baud rate, reconstructing the original parallel data.
- Stop Bit Validation: The receiver confirms the presence of stop bits to ensure the frame's completeness.

Synchronization and Baud Rate:
UART communication is asynchronous, meaning it does not use a shared clock signal. Instead, both the transmitter and receiver must operate at the same baud rate Any mismatch in baud rate can lead to data corruption.

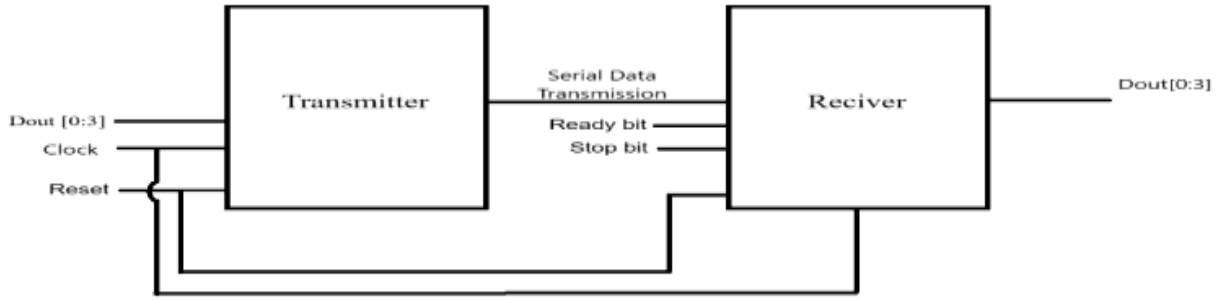


Fig 1. UART Module Block Diagram

Asynchronous Communication:
In asynchronous communication, data is sent one bit at a time at a constant rate, without a shared clock. The receiver uses the agreed-upon baud rate to sample incoming bits and synchronize using start and stop bits. This eliminates the need for a dedicated clock line, simplifying hardware connections.
Transmission:

1. Data Preparation: Parallel data is loaded into the UART.
2. Framing: The UART frames the data with a start bit, data bits, an optional parity bit, and one or more stop bits.
3. Serial Transmission: Framed data is sent serially over the communication line, bit by bit.

Reception:

1. Bit Synchronization: The receiving UART identifies the start bit to align with the data stream.
2. Frame Processing: Data bits are extracted, verified using parity (if enabled), and start/stop bits are discarded.
3. Parallel Output: Serial data is converted back into parallel form for the receiving device.

UART's asynchronous nature and efficient framing make it ideal for low-speed communication in embedded systems and microcontrolle

PISO (Parallel-In Serial-Out) Shift Register

Functionality:

A **PISO shift register** takes multiple bits of data as parallel input and converts them into a serial output, allowing data to be transmitted over a single line. This is especially useful for reducing the number of connections required in communication systems or digital circuits.

1. **Data Loading**: Data is loaded simultaneously into all flip-flops of the register through the parallel input lines when a control signal (e.g., a "load" signal) is activated.
2. **Data Shifting**: After loading, the register shifts the data serially, one bit at a time, to the output with each clock pulse.
3. **Control Signals**: Typically requires control signals like "Load" (to enable parallel loading) and "Shift" (to enable serial shifting).

Key Applications:

A Serial-In Parallel-Out (SIPO) shift register is commonly used to deserialize serial data for parallel processing, making it useful in communication systems where data is transmitted serially but needs to be processed in parallel. It's also widely employed to expand microcontroller I/O capabilities, allowing control of multiple output devices—such as LEDs, relays, or displays—using just a few microcontroller pins. Additionally, SIPO registers serve as temporary data storage buffers in digital systems, helping manage data flow and synchronization between components operating at different speeds..

SIPO (Serial-In Parallel-Out) Shift Register

Functionality:

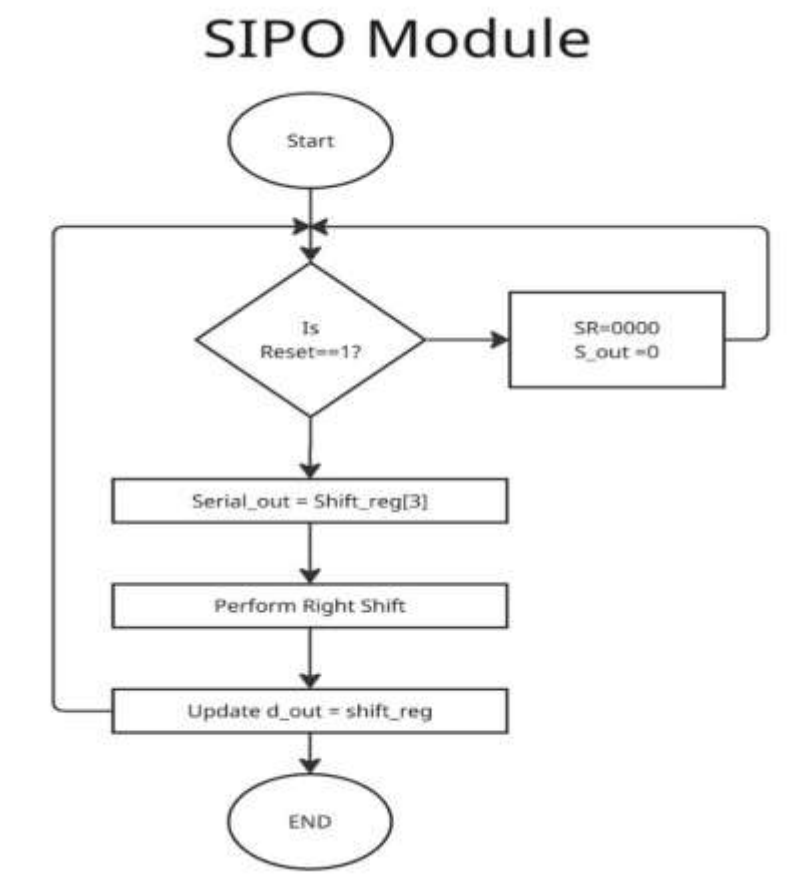
A **SIPO shift register** takes data in serial form, shifts it through a chain of flip-flops with each clock pulse, and then outputs it in parallel. This is used to convert data received over a single line into a parallel format for processing.

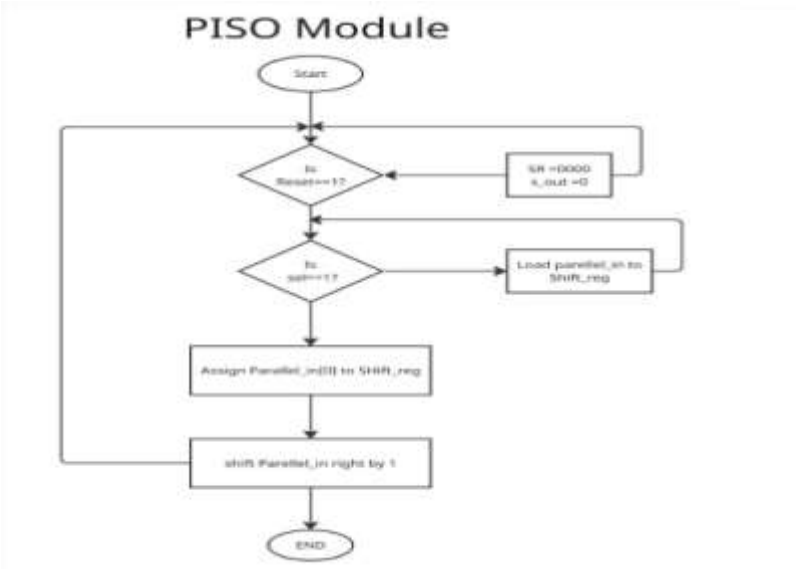
1. **Data Input**: Data is provided one bit at a time through the serial input line, synchronized with a clock.
2. **Data Shifting**: Each clock pulse shifts the data into successive stages of the register.
3. **Parallel Output**: Once the entire sequence is shifted in, the parallel output lines hold the complete data set.

Key Applications:

SIPO shift registers are widely used for serializing parallel data for efficient transmission over communication lines. In protocols like UART (Universal Asynchronous Receiver/Transmitter), data is often converted from parallel to serial form (and vice versa) to match the requirements of serial communication. Additionally, SIPO shift registers help in reducing the number of pins needed when interfacing microcontrollers with external devices such as LED arrays or display drivers. Instead of assigning multiple GPIO pins for each data line, a single serial line can be used to shift data into the register and then output it in parallel, thereby optimizing resource usage in embedded systems.

The given code implements a Parallel Input Serial Output (PISO) shift register. Initially, a 4-bit register q is declared to hold input data, and a 1-bit output v is used to transmit the serial data. The inputs include a 4-bit parallel data d, a selection signal sel, a clock signal clk, and an active-high reset signal. On every rising edge of the clock, the system first checks the reset condition. If the reset signal is high, the register q is cleared by setting all bits to zero. If not in reset, the behavior depends on the value of sel. When sel is low (0), it indicates a parallel load operation, so the 4-bit input d is loaded into the register q. When sel is high (1), the module enters serial output mode: it assigns the least significant bit of q to the output v and shifts all bits of q one position to the right. This process continues on each clock cycle, effectively shifting out the 4-bit data serially through v.

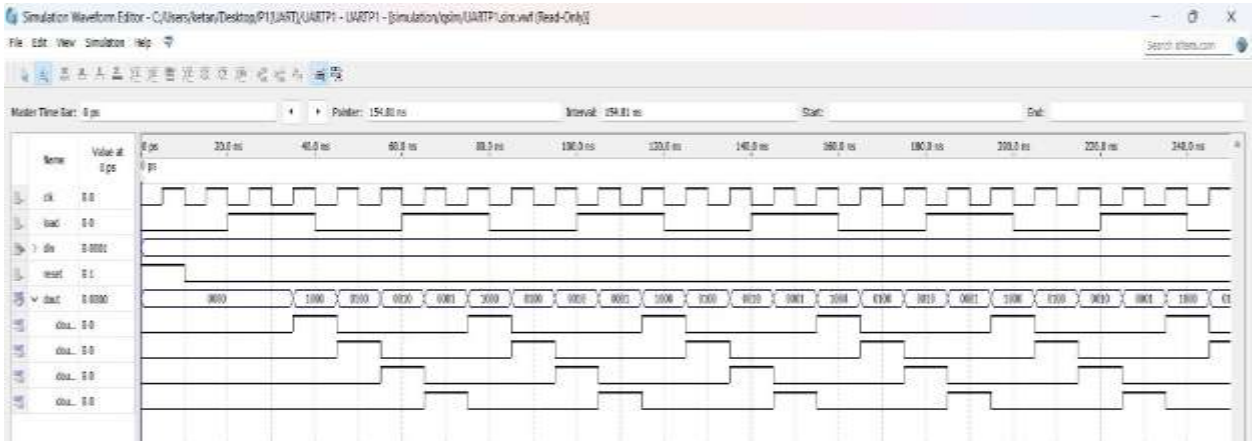




The Serial-In Parallel-Out (SIPO) shift register algorithm operates based on three main inputs: a serial data input (`din`), a clock signal (`clk`), and an active-high reset signal (`reset`). It produces a 4-bit parallel output (`dout`) using an internal 4-bit register `s` that stores the shifted bits. Initially, the contents of the register `s` are undefined. However, whenever the `reset` signal is high, the register is cleared, and all bits of `s` are set to zero. The behavior of the module is triggered on the rising edge of the `clk` signal or whenever the `reset` signal becomes active. If `reset` is high, the register is cleared immediately, and no shifting takes place during that clock cycle. If `reset` is low on a rising clock edge, the shift operation is performed: each bit in `s` is shifted one position to the left (`s[3]`

takes the value of `s[2]`, `s[2]` takes `s[1]`, `s[1]` takes `s[0]`), and the new serial input bit (`din`) is loaded into the least significant bit `s[0]`. After the shift operation, the contents of the register `s` are assigned to the parallel output `dout`. This process repeats with every clock pulse or reset event, enabling the serial input to be captured and output as parallel data over time.

Fig 3. UART SIPO Module



Output:

Fig 3.Simulation result of receive

This waveforms display changes in signal states, helping to analyze how data is transferred bit by bit. Engineers and students use these results to check if the UART is working correctly—whether it's sending and receiving data as expected. This is useful for debugging, verifying baud rates, and ensuring proper timing between transmitted and received sign

Conclusion:-

In this project, we successfully designed and simulated a UART system using Verilog, focusing on Parallel-In-Serial-Out (PISO) and Serial-In-Parallel-Out (SIPO) modules. Our implementation showed how UART plays a key role in serial communication by efficiently converting data between parallel and serial formats. While we faced challenges like timing synchronization and ensuring accurate data sampling, troubleshooting these issues deepened our understanding of digital communication. In the future, this design can be extended by adding error detection, adjustable baud rates, or integrating with other protocols. Overall, this project enhanced our practical skills in hardware design and Verilog coding.

References:-

1.Tianjun Zhan "Verilog HDL-based implementation of UART design", Proc. SPIE 13552, International Conference on Physics,Photonics, and Optical Engineering (ICPPOE 2024), 135523A (7 March 2025).
2.S. Saha, M. A. Rahman, A. Thakur, “Design and Implementation of a BIST Embedded High Speed RS- 422 Utilized UART over FPGA,” Fourth International Conference on Computing, Communications Networking Technologies (ICCCNT), pp. 1-5, 4-6 July 2013.
4.Dhanadravye AD, Thorat SS. A review on implementation of UART using different techniques. International Journal of Computer Science and Information Technologies. 2014 Jan.
5.Du, Zihang, et al. "Verilog implementation of configurable UART module." Second International Conference on Statistics, Applied Mathematics, and Computing Science (CSAMCS 2022). Vol. 12597. SPIE, 2023.