Serial Device Driver and Protocol

Introduction:

This project focuses on developing an embedded system for communication and control, employing a Microchip PIC32 microcontroller. The system utilizes UART (Universal Asynchronous Receiver-Transmitter) communication to interact with an external device, facilitated by the implementation of a robust communication protocol.

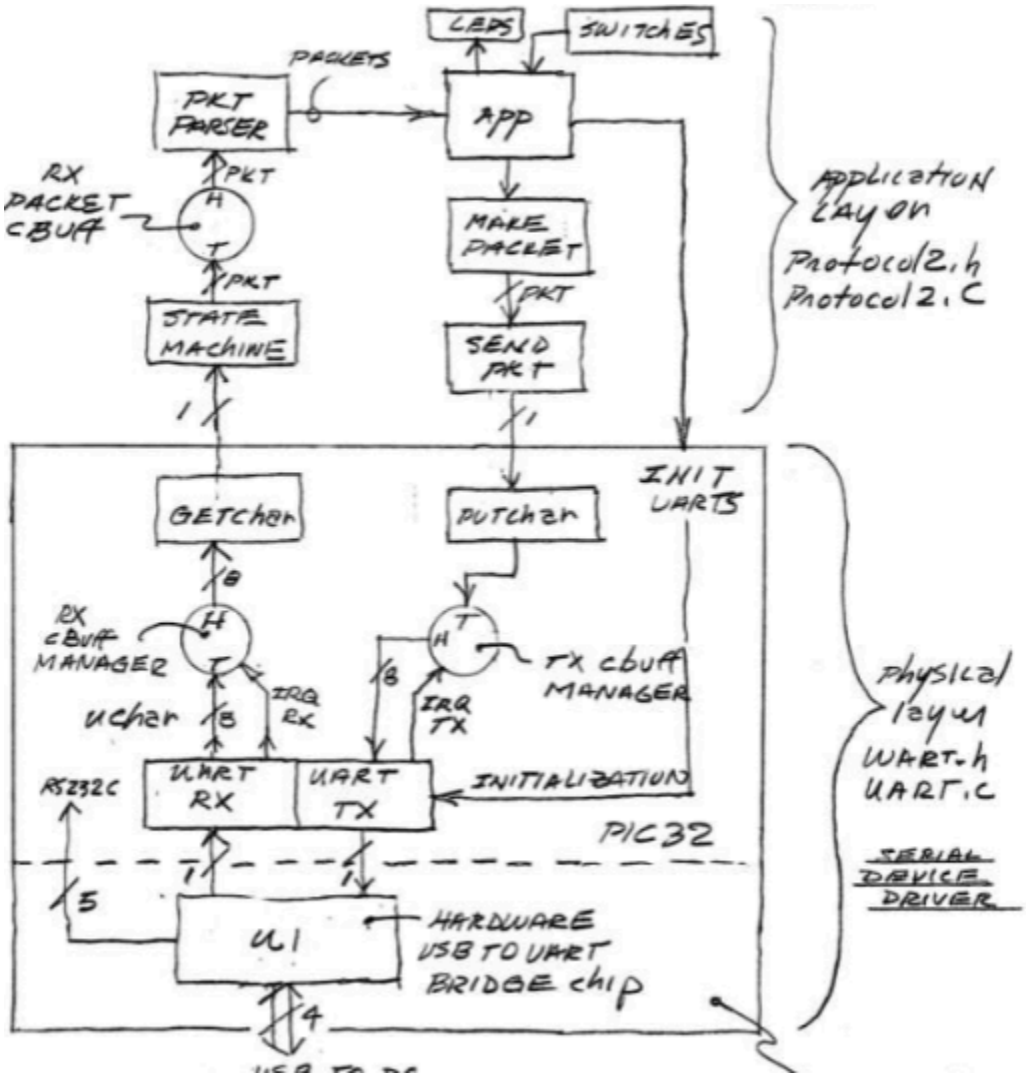
Project Background:

This project centers around the development of an embedded system using the Microchip PIC32 microcontroller, focusing on efficient communication and control. In three key tasks, it establishes a Universal Asynchronous Receiver-Transmitter (UART) communication setup, incorporates buffer functions and interrupts for enhanced data management, and rigorously tests the system through LED control, status retrieval, and complex packet exchanges. Rooted in the practicality of embedded systems, the project addresses the real-world need for reliable communication protocols and interrupt-driven mechanisms, offering a systematic and concise exploration of these essential components in the realm of embedded systems.

Project Requirements:

- **UART Communication Setup:**
    - Initialize and activate UART components for serial communication.
    - Set up baud rate for efficient data transmission.
    - Enable UART modules and configure status registers.
- **Buffer Functions and Interrupts**:
    - Implement a buffer with head, tail, and data components.
    - Develop functions to check buffer status and facilitate data flow.
    - Enable interrupts for transmission and reception.
    - Handle collisions during data enqueueing and dequeuing.
- **System Testing:**
    - Test LED control through Python console commands.
    - Retrieve and transmit LED status as comprehensive packets.
    - Implement a PING-PONG test for complex packet exchanges.
    - Verify functionality through systematic LED, status, and packet tests.

High Level Design:



RX Register Concept:

- Purpose: The Receive (RX) register is a vital element in UART communication, responsible for storing incoming data.
- Functionality: When external devices transmit data to the microcontroller, the RX register captures this data, allowing the microcontroller to process and utilize it.
- Efficiency: Understanding and utilizing the RX register ensures the microcontroller efficiently receives and processes incoming data, facilitating effective communication with external devices.

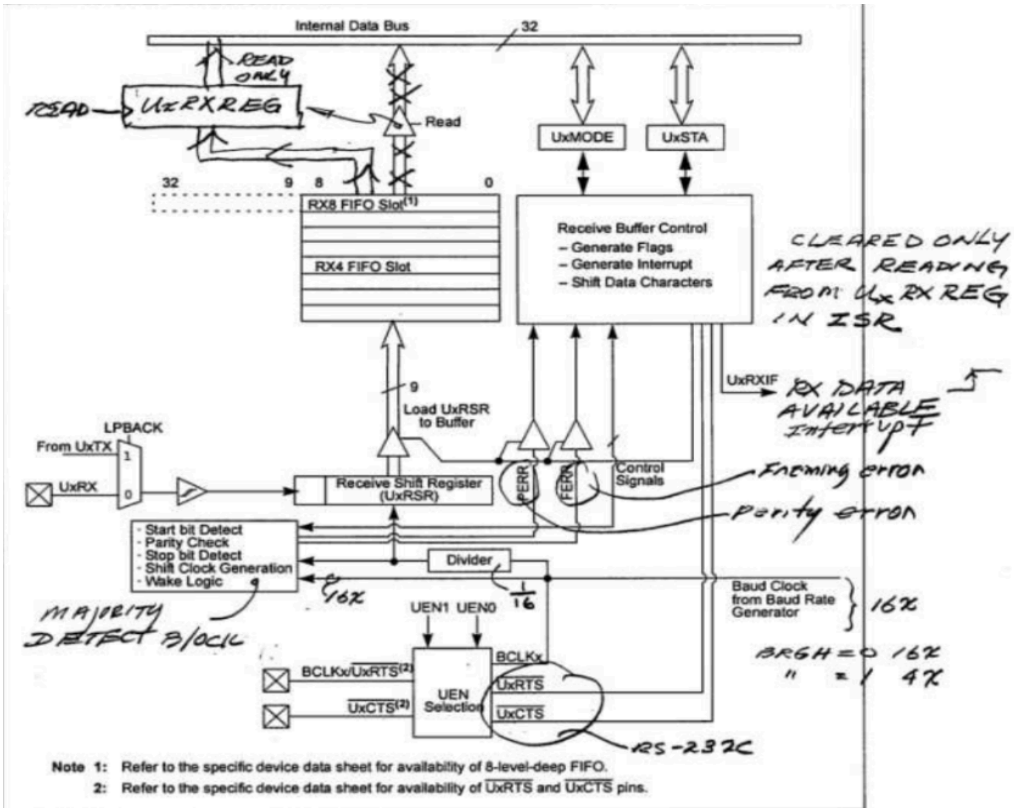


Figure . 1|UART RX Block Diagram

TX Register Concept:

- Purpose: The Transmit (TX) register is a crucial component in UART communication, serving as the storage for outgoing data.
- Usage: When data needs to be sent, it is loaded into the TX register. The UART module then transmits this data serially.
- Efficiency: By understanding and managing the TX register, efficient serial communication is achieved, ensuring the seamless flow of data between the microcontroller and external devices.

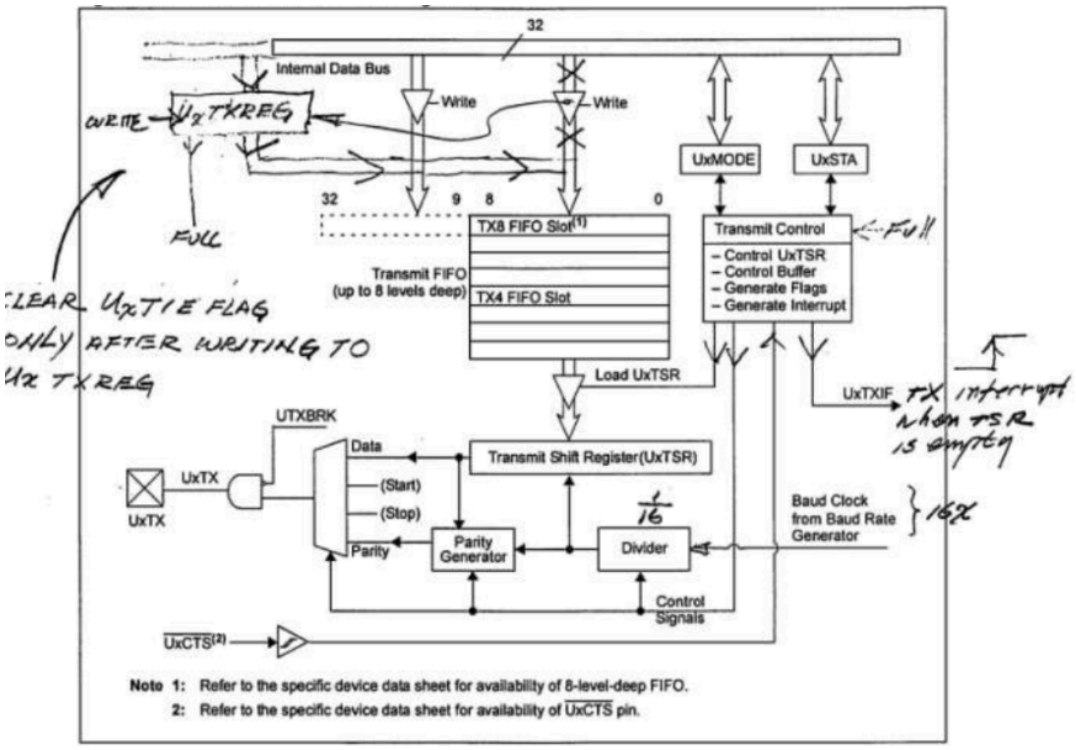


Figure . 1|UART TX Block Diagram

Buffer Concept:

- Role: A buffer acts as a temporary storage area for data, essential for managing information flow between different parts of the program.
- Components: It typically includes a head (indicating next data ready), a tail (highlighting the next available space), and the actual data storage.
- Operations: When data is read, the head increments, and when data is written, the tail increments. Checking head and tail values helps determine if the buffer is empty or full.
- Significance: Buffers play a crucial role in handling data asynchronously, preventing data loss, and facilitating smooth communication between different components of the program. In this project, buffers are used to manage incoming and outgoing UART data efficiently.

Interrupt Signal Concept:

- Purpose: Interrupts serve as signals that temporarily halt the main program to address specific events or conditions.
- Activation: Enabled through registers, interrupts can be triggered by events like data reception or transmission, allowing the program to respond promptly.
- Priority: Configurable priority levels determine the order in which interrupts are addressed.
- Significance: In this project, interrupts enhance the program's responsiveness by allowing immediate actions in response to events like incoming data (RX) or the need for data transmission (TX). They streamline communication without constant polling, optimizing program efficiency.

Detail Design:

System Overview:

- **UART Initialization**:
    - Activate UART1 module: Set the ON bit in U1MODE register.
    - Baud Rate Setup: Use the formula UxBRG = FPB / (16 * Baud Rate) - 1, where FPB is the Peripheral Bus Frequency.
    - Enable Transmit and Receive: Set U1STAbits.UTXEN and U1STAbits.URXEN bits in U1STA register.

- **Buffer Implementation**:
    - Circular Buffer: Maintain a circular buffer with head, tail, and a data array.
    - Buffer Status: Implement functions to check if the buffer is empty or full based on head and tail positions.
    - Enqueue and Dequeue: Functions to add data to the buffer and retrieve data from it.
- **Interrupts**:
    - Enable interrupts for TX and RX: Set IEC0bits.U1TXIE and IEC0bits.U1RXIE bits.
    - Priority Configuration: Set interrupt priority using IPC6 register.
    - Interrupt Service Routine (ISR): Implement separate ISRs for TX and RX interrupts.
- **Packet Handling**:
    - State Machine: Create a state machine for packet reception with states like IDLE, GET_LENGTH, GET_ID, GET_PAYLOAD, GET_TAIL, and WRAP_UP.
    - Checksum Calculation: Implement checksum calculation using the Berkeley Standard Distribution (BSD) algorithm.
- **LED Control**:
    - Initialize LEDs: Use a function LEDs_INIT() to set up LED pins.
    - Get LED Status: Function LEDS_GET() to retrieve the status of all LEDs.
    - Set LED Status: Function LEDS_SET(leds) to set the status of LEDs based on the input.
- **Testing**:

- LED_SET Test: Verify the ability to toggle the status of an LED through a console command.
- GET_LED_STATUS Test: Press a "Get_status" button in the console to receive the current LED status in hexadecimal format.
- PINGPONG Test: Dispatch a packet containing a number; expect the board to send back the packet with the payload's value divided by two.

Future Considerations:

- Allow for expandability, accommodating additional functionalities.
- Enhance error detection and recovery mechanisms.
- Optimize code for performance and memory usage.

Closure note:

In conclusion, this project involves the implementation of a UART-based communication system on a microcontroller, focusing on key aspects such as UART initialization, circular buffer management, interrupt handling, LED control, and packet processing. By carefully designing and integrating these components, the system enables bidirectional communication between the microcontroller and a computer terminal. The utilization of a state machine for packet reception, along with checksum calculations and payload manipulation, ensures robust and reliable data transmission. Testing procedures validate the functionality of LED toggling, status retrieval, and the more intricate PINGPONG scenario. This project not only serves as a practical exercise in embedded systems but also lays the groundwork for future enhancements and optimizations in similar communication applications.