Uart Implementation Reference

v1.0.0

Written by
Kevin LASTRA

Contents

1 Version

Version	Date	Author	Description
v1.0.0	0	Kevin Lastra	Initial release of the UART Implemenation Reference

2 Introduction

2.1 Purpose of the manual

The purpose of this manual is twofold. First, it serves as a comprehensive resource to document the Implementation of a Universal Asynchronous Receiver-Transmitter (UART). This process has been undertaken not only as a journey of self-learning but also as a way to share knowledge and contribute to the broader community of developers, engineers, and enthusiasts.

2.2 Audience

This manual is primarly intended for design engineers who are involved in the development and implementation of communication systems, particularly those working with embedded systems, microcontrollers, and hardware interfaces.

3 UART protocol overview

The Universal Asynchronous Receiver-Transmitter (UART) is a fundamental hardware communication protocol used for asynchronous serial communication.
This protocol is widely used in embedded systems, microcontrollers, and communication devices due to its simplicity and effectiveness in handling relatively low-speed data transfer.

3.1 UART fundamentals

3.1.1 Reception and transmission

UART allows two devices to exchange data using only two wires, RX for receive data and TX for transmit data.

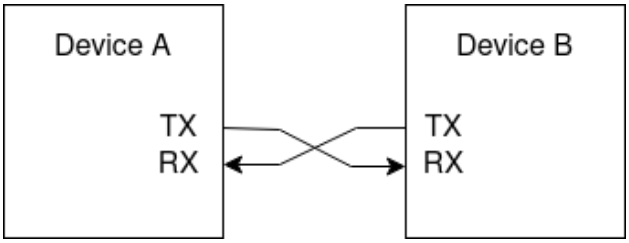

Figure 1: Two UART's interfaces connected

3.1.2 Asynchronous communication

An asynchronous communication refers to a type of data transmission where the sender and receiver do not rely on a shared clock signal for synchronization. Instead, each device operates using its own internal clock and relies on specific timing conventions to ensure data is transmitted and received correctly.

In asynchronous UART communication, the data is sent in discrete chunks (called data frames) over the communication channel, with the timing governed by agreed-upon parameters like baud rate and frame size.

3.1.3 Baud rate

The baud rate defines the speed at which data is transmitted over the communication channel. It is usually expressed in bits per second (bps).
In the context of UART the baud rate specifies how many bits of data can be transmitted each second. Both transmitter and receiver must be set to the same baud rate.

The general formula for calculating the baud rate is:

$$BaudRate = \frac{ClockFrequency}{Divisor}$$

3.1.4 Data frame structure

A UART data frame typically consists of:

Range name	**Description**	**Implementation bit length**
Start bit	Signals the beginning of data transmission	1
Data bits	The actual data being sent	5 - 9
Parity	Used for error checking	0 - 1
Stop bit	Signals the end of the data transmission	1 - 2

3.2 Advantages and limitations

Advantages:

1. Minimal pins required, essay to implement on a system.
2. Low cost, because of its minimal hardware requirements the UART is a cost-effective solution.
3. Widely supported.
4. Simplicity in software implementation.

limitations:

1. Distance limitations, the maximum range depends on the baud rate, the quality of the wires and the environment (e.g. electromagnetic interferences).
2. Relatively slow data transfer.
3. Limited to two devices.
4. Susceptibility to baud rate mismatch.
5. Limited error detection, UART error detection is limited to parity checks and stop bits.

4 Implementation

4.1 Design

The UART implementation is composed of four primary modules: a Receiver, a Transmitter, a Clock Generator, and a control and status register with an AXI4 system interface.

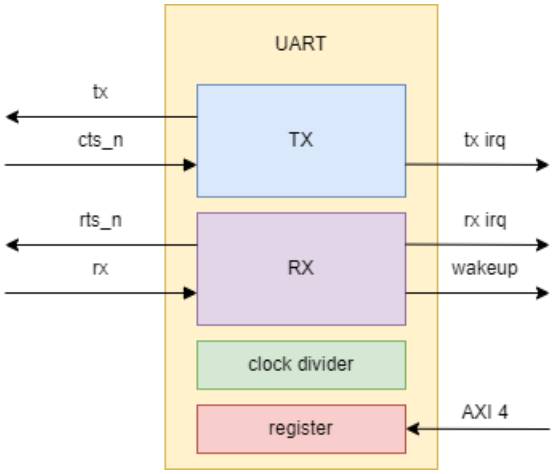

Figure 2: Implementation diagram.

4.1.1 Uart configuration

TODO : configuration structure

4.1.2 Data frame

The received and transmitted data frame is built has follows :

Start	Data	Parity	Stop
1	8	*1	1

(*) The parity bit is an optional feature.

4.1.3 Receiver (RX)

This module is responsible for sampling and decoding incoming serial data. It detects the start bit, samples each data bit based on the configured baud rate, check the parity bit and verifies the stop bit. Once a complete and valid frame is received, the data byte is stored in an asynchronous FIFO for further processing by the system.

TODO : rx status

4.1.4 Transmitter (TX)

The transmitter handles the serialization of data from an asynchronous FIFO. It prepends a start bit, appends a parity bit and a stop bit, and transmits the bits sequentially according to the configured baud rate.

TODO : tx status

4.1.5 Interrupts

TODO : Explains when is latch and posedge event

4.1.6 Wakeup line

TODO : Explains when is latch and posedge event

4.2 Control and status registers

Register	Access type	Offset	Reset	Description
divider	RW	0x00	0x0	Used to generate the UART clock by dividing the system clock.
rx_data	RO	0x04	0x0	Last received data. One-time read access.
rx_status	RO	0x08	0x0	RX module status:

<table>
<tr><td></td><td></td><td></td><td></td><td><table><tr><th>Bit</th><th>Name</th></tr><tr><td>0</td><td>Fifo empty</td></tr><tr><td>1</td><td>Fifo full</td></tr><tr><td>2</td><td>Overrun error</td></tr><tr><td>3</td><td>Parity error</td></tr><tr><td>4</td><td>Framing error</td></tr></table></td></tr>
<tr><td>rx_irq_mask</td><td>RW</td><td>0x0C</td><td>0x0</td><td>Enables interrupts for specific bits in rx_status</td></tr>
<tr><td>tx_data</td><td>WO</td><td>0x10</td><td>0x0</td><td>Write data to TX asynchronous FIFO. If full overrun_error status will be set to 1.</td></tr>
<tr><td>tx_status</td><td>RO</td><td>0x14</td><td>0x0</td><td>TX module status:<table><tr><th>Bit</th><th>Name</th></tr><tr><td>0</td><td>Fifo full</td></tr><tr><td>1</td><td>Fifo empty</td></tr><tr><td>2</td><td>Overflow error</td></tr></table></td></tr>
<tr><td>tx_irq_mask</td><td>RO</td><td>0x18</td><td>0x0</td><td>Enables interrupts for specific bits in tx_status</td></tr>
<tr><td>control</td><td>RW</td><td>0x1C</td><td>0x0</td><td>Configure UART:<table><tr><th>Bit</th><th>Name</th></tr><tr><td>0</td><td>Flush tx FIFO</td></tr><tr><td>1</td><td>Flush rx FIFO</td></tr><tr><td>2</td><td>Enable/Disable flow control</td></tr><tr><td>3</td><td>Enable/Disable parity bit</td></tr><tr><td>4</td><td>Master device. Used only on mode SIMPLEX or HALFDUPLEX.
1 - Use only TX line
0 - Use only RX line</td></tr><tr><td>[5:6]</td><td>Uart mode:
2’b00 : SIMPLEX
2’b01 : HALFDUPLEX
2’b20 : FULLDUPLEX</td></tr></table></td></tr>
<tr><td>version</td><td>RO</td><td>0x20</td><td></td><td>Hold the current version implemented</td></tr>
</table>

4.2.1 Instantiation

TODO : # Input and Outputs # Axi4 interface # Parameters

4.2.2 Instantiation

TODO : # FMAX?