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.

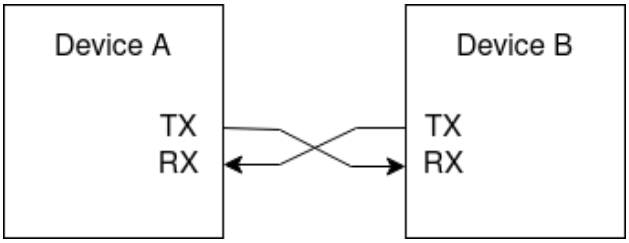

Figure 1: Two UART's interfaces connected

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.

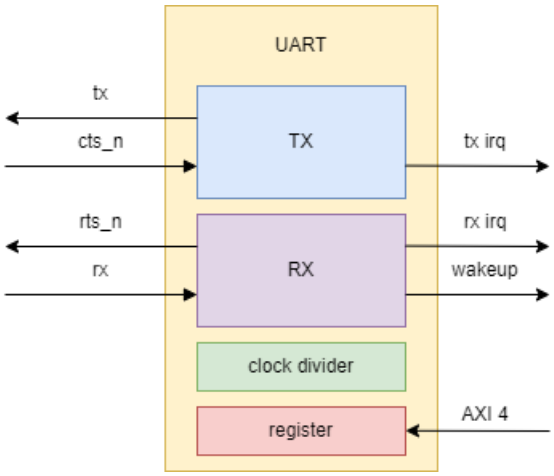

Figure 2: Implementation diagram.

4.1.1 Uart configuration

The UART configuration register allows control over various operational aspects of the UART. It includes the following fields:

1. **flush_rx / flush_tx**: Used to clear the RX and TX asynchronous FIFOs, respectively.
2. **flow_control**: Enables or disables hardware flow control using the CTS_N (Clear To Send) and RTS_N (Request To Send) signals.
3. **parity**: Enables or disables parity checking and generation for data frames.
4. **master**: Defines which part of the UART (transmitter or receiver) is active. This is only used in SIMPLEX or HALFDUPLEX communication, where only one direction is active at a time.

5. **mode**: Specifies the UART communication mode:

   (a) **FULLDUPLEX**: Both transmitter and receiver are active and operate simultaneously. This is the standard mode for most UART communication.

   (b) **HALFDUPLEX**: Transmitter and receiver share the same communication line, but only one can be active at a time. The master field determines which is currently active.

   (c) **SIMPLEX**: Communication occurs in only one direction. The master field is used to enable either TX or RX, but not both.

This register is typically configured through the AXI4 interface during initialization or reconfiguration of the UART.

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 (if enabled) 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.

The RX module provides a status register that reflects the current state of the receiver and any error conditions. The following bits are included:

1. **fifo_empty**: Indicates that the RX FIFO is empty and no data is available to read.

2. **fifo_full**: Indicates that the RX FIFO is full and cannot accept more data.

3. **overrun_error**: Set when new data arrives while the FIFO is full, meaning data has been lost because the system did not read the previous data in time.

4. **parity_error**: Set when parity checking is enabled and the received data has an invalid or unexpected parity bit. This may also occur if parity is disabled but a parity bit is present in the incoming frame.

5. **framing_error**: Set when the received frame does not conform to the expected UART format.

These status bits can be used by the system to monitor the integrity and availability of received data and handle UART errors appropriately.

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.
The transmitter module provides a status register that reports the current state of the TX FIFO and potential write errors. It includes the following bits:

1. **fifo_empty**: Indicates that the TX FIFO is empty and no data is waiting to be transmitted.

2. **fifo_full**: Indicates that the TX FIFO is full and cannot accept additional data.

3. **overrun_error**: Set when the system writes new data to the TX FIFO while it is already full. In this case, the new data is lost.

These status bits help the system manage data transmission and ensure that write operations are performed safely and efficiently.

4.1.5 Interrupts

The UART module generates an interrupt when one or more bits in the status register are set and the corresponding bits in the interrupt mask are enabled. The interrupt signal remains active as long as at least one masked status condition is true.

4.1.6 Wakeup line

The wakeup signal is asserted by the RX module when it detects valid incoming serial data. The signal remains active for the duration of the reception, typically 8 to 9 clock cycles, corresponding to the length of a valid UART frame (start bit, data bits, optional parity, and stop bit).

This can be used to wake up the system from low-power modes when data activity is detected on the RX line.

4.2 Control and status registers

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

<table>
<tr><th>Register</th><th>Access type</th><th>Offset</th><th>Reset</th><th>Description</th></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>

4.3 Integration

This section describes how to instantiate and integrate the UART module into other projects, including its parameters, ports, and interfaces.

4.3.1 Parameters

The module includes configurable parameters to adapt the UART to different system requirements:

Name	Length	Description
REG_ADDR_MAP	1	Defines the base addresses of the UART internal registers for AXI4 access

4.3.2 Inputs

Name	Length	Description
clk	1	System clock
rst_n	1	Active-low reset signal
rx	1	Serial input line
cts_n	1	Clear to send signal

4.3.3 Outputs

Name	Length	Description
tx	1	Serial output line
rts_n	1	Request to send signal
wakeup	1	Wakeup signal
rx_irq	1	rx interrupt request
tx_irq	1	tx interrupt request

4.3.4 Interface

AXI4 Slave Interface: Used to access the configuration and status registers.

4.3.5 Instantiation example

```
uart #(
  .REG_ADDR_MAP(0x5000_0000)
) uart_inst (
  .clk(clk),
  .rst_n(rst_n),
  .rx(rx_line),
  .tx(tx_line),
  .rts_n(rts_n_line),
  .cts_n(cts_n_line),
  .wakeup(uart_wakeup),
  .rx_irq(uart_irq),
  .tx_irq(uart_irq),
  .bus(axi4)
);
```

Listing 1: UART Module Instantiation

5 Synthesis Corners

Metric	Value	Conditions
FMAX	X	X
LUTs	X	X
FFs	X	X
BRAMs	X	X