Logotipo

Descrição gerada automaticamente

UNIVERSITÁ DI PISA

ENGEGNERIA DELL’INFORMAZIONE

**Project Report**

**UART Receiver**

**Bruno Augusto Casu Pereira de Sousa**

**Electronics Systems a.a. 2022/2023**

Pisa, 2023

**Contents**

[1. Introduction 3](#_Toc125413468)

[2. Architecture description 5](#_Toc125413469)

[**2.2** **Synchronization module (rx\_synch)** 5](#_Toc125413470)

[3. VHDL code 6](#_Toc125413471)

[4. Testing and verification 6](#_Toc125413472)

[5. Synthesis and implementation 7](#_Toc125413473)

[6. Conclusions 7](#_Toc125413474)

# **1. Introduction**

The project presented in this document is the implementation of the Receiver part of a Universal Asynchronous Receiver/Transmitter (UART) peripheral. The UART is a simple protocol for bi-directional serial data transmission, generally used for microcontroller applications. The peripheral device then provides a hardware interface for fast serial-to-parallel data conversion from external devices sending to the CPU, and for parallel-to-serial data conversion when receiving from the CPU.

The UART hardware interface consists in two data lines, *tx* and *rx*:

Texto

Descrição gerada automaticamente

Figure 1 – UART hardware interface

For the transmission of data, the UART protocol itself defines that the data lines must be kept in high (‘1’) to represent the idle state. To start a new transmission, the line must be set to GND (‘0’) for one period, indicating the start bit. After the start of the frame, the word bits are transmitted, followed by a parity bit (optional). To complete the transmission, a defined number of stop bits must be transmitted, keeping the UART lines at high and thus leaving it in the idle state at the end of the frame. An example with different configurations is shown in Figure 2:

Interface gráfica do usuário, Texto, Aplicativo, Tabela, Excel

Descrição gerada automaticamente

Figure 2 – UART protocol formats (Source: Texas Instruments)

A common use of the UART is to provide an interface for character transmission from CPU to peripheral devices, such as sensors in an IoT application, to control and obtain the measurements from the devices. Another possible usage is to connect the CPU to another external serial device, working as a converter, for example, to an USB port connected. By using this implementation, the UART can reduce the process demand from the CPU, as it uses a much-simplified protocol and an easy to build dedicated hardware.

To build a peripheral UART in a microcontroller architecture, the design must implement efficient ways to read and write the data from the serial lines to and to perform the serial-to-parallel and parallel-to-serial operations. A possible way to do so is by using a shift register connected at the input and output ports of the UART, combined with a set of converters that will read/write the transmitted words in a buffer.

The peripheral must also be integrated the available resources of the platform, by making available error and control signals for the system interrupt control. An example of hardware implemented UART peripheral by Texas Instruments was studied to guide the final design of the module developed, and to understand the overall behavior of the peripheral. The block diagram presented will highlight the receiver part of the UART peripheral:

Diagrama

Descrição gerada automaticamente

Figure 3 – KeyStone UART partial block diagram (Source: Texas Instruments)

By focusing on the key elements of the receiver, this section can be divided in a set of components:

1. *Receiver Timing and Control*: will track the start and end of the transmission according to the settings and protocol specification (will produce a Frame Error in case of transmitter frequency mismatch).
2. *Receiver Shift Register*: connected to the rx data line, will store the word data and parity bit, until data is ready to be moved to the buffer.
3. *Receiver Buffer Register*: buffer for the UART output data
4. *Receiver FIFO*: a fifo type buffer (memory array) used to store multiple frames received, removes the necessity of triggering the CPU at each operation.

Considering the project requirements, some adaptations will be necessary for this overall scheme of the UART receiver developed. A validation signal must be produced at the output of the module, therefore a control block must be present, to check for any errors in the protocol during the reception. This signal is then asserted for one UART clock cycle, informing that the data at the buffer is ok to be read.

Also, the FIFO mode will not be implemented in the receiver peripheral. A simplified diagram of the initial idea for the UART receiver is shown in Figure 4. On the Architecture description section, the final implementation will be discussed.

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente**

Figure 4 – UART receiver schematic (initial idea)

# **2. Architecture description**

With the UART peripheral example studied, aligned with the project requirements, a model was developed to perform the necessary operation of the protocol. The required settings for this receiver were:

1. Word size (W) = 7 bits
2. Baud rate (B) = 112500
3. Parity (P): even
4. Number of Stop bits (S) = 2
5. Oversampling rate (OS\_RATE) = 8

Within this UART module, three main components were designed to handle the protocol operation, as well as an addition independent block, used for a particular situation of the protocol. The modules will be detailed in the next sections.

## **2.1 Synchronization component (rx\_synch)**

This component will take care of receiving the data from the serial line and will provide the frame synchronization. An internal counter based on the UART internal clock will be used to sample the rx line, reading the values of the transmitted bits (from the design specification, the UART clock time is eight times the Baud rate of the UART transmitter). A second counter based on this sampling count will then copy the rx value to a parallel output, connected to the buffer component.

This method of sampling the input data resembles more the operation of a demultiplexing device, rather then a proper shift register. The block diagram of a Demux and its behavior are shown in Figures 5 and 6 (on the signal representation, consider an over sampling rate of 4 times, instead of 8; and a word size of only 4 bits):

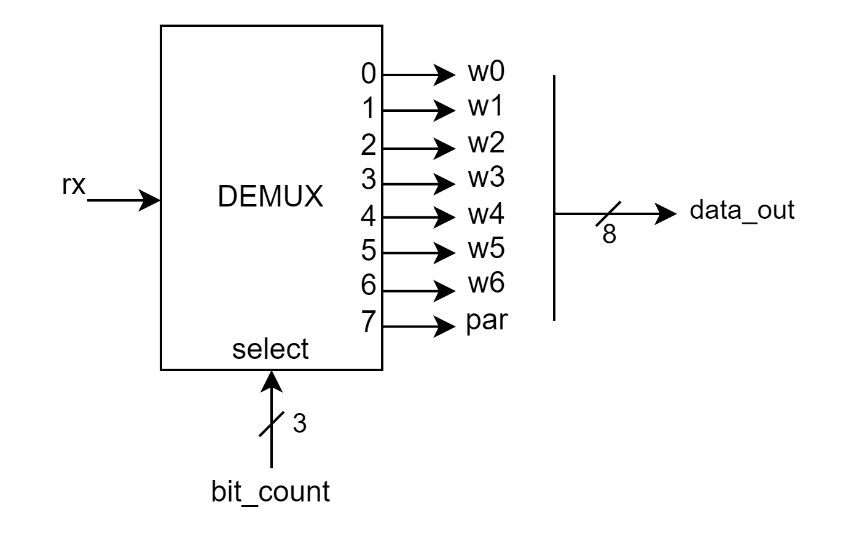


Figure 5 – Demux block diagram

Forma

Descrição gerada automaticamente com confiança média

Figure 6 – UART rx sampling using a Demux

This design choice is then an alternative way of getting the transmitted bits from the serial line to a parallel output, instead of the shift register approach used on the Texas Instruments device shown. The bit\_count in the Demux then selects the output number and is incremented after OS\_RATE clock cycles, to sample the input at the middle, or as close as possible, of the expected bit transmitted.

No benefits were observed when implementing this component as the serial-to-parallel converter. The main difference that can be discussed is that the individual outputs of this block are kept constant during the reception of the word (after the first bit is set to the Demux output, that value is no longer altered during the frame transmission, and so on for the other bits).

Along with the input sampling, the synchronizer component will generate four internal control signals, informing the other UART components the current state of the protocol transaction. Those signals are:

1. *frame\_start*: indicates the start of a frame, it is set to ‘1’ after the rx line goes down, and is kept after 4 UART clock cycles (oversampling rate/2).
2. *frame\_stop*: indicates when the last stop bit is detected.
3. *frame\_error*: indicates when the start or stop conditions of the UART protocol are violated (usually frame errors occurs when there is a mismatch between transmitter and receiver baud rate values).
4. *uart\_data\_ready*: indicates that all the word bits and parity are being outputted by the Demux, is set to ‘1’ when the last bit of the word (parity) is sampled.

Along with the input sampling, the synchronizer component will generate four internal control signals, informing

RESET\_S,

IDLE\_S,

START\_DETECT\_S,

RECEIVE\_DATA\_S,

DATA\_READY\_S,

STOP\_DETECT\_S,

FRAME\_END\_S,

FRAME\_ERROR\_S

Diagrama

Descrição gerada automaticamente

Forma

Descrição gerada automaticamente com confiança média

## **2.2 Buffer component (rx\_buff)**

## **2.3 Control component (rx\_control)**

## **2.4 Break counter (break\_counter)**

# **3. VHDL code**

# **4. Testing and verification**

Test strategy (Test-plan) and related Testbench for verification; a detailed, though not exhaustive, verification is required, including error situations and borderline cases of functioning.

Imagem de vídeo game

Descrição gerada automaticamente com confiança média

Figure X – UART receiver schematic (initial idea)

# **5. Synthesis and implementation**

maximum clock frequency (critical path), elements used (slice, LUT, etc.) and estimated power consumption. Comment on any warning messages.

# **6. Conclusions**

Synchronization block behavior at the beginning of a new UART transmission (Start bit detection).

Synchronization block behavior when pushing data in the internal shift register.

Buffer behavior when receiving Data from the Synchronizer block.

Tela de jogo de vídeo game

Descrição gerada automaticamente

UART behavior when data is ready at the buffer and the Validation check is executed (The transmitted frame has a parity inconsistency).

Tela de computador com jogo

Descrição gerada automaticamente