Une image contenant logo

Description générée automatiquement

|  |
| --- |
|  |
| RS232\_SYSCON  SPECIFICATION |
|  |
|  |

**MASTERY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Company** | **Name** | **Function** | **Date** |
| Written by: |  | John Clayton |  | 2001/08/13 |
| Approved by: |  |  |  |  |
| Authorized by: |  |  |  |  |

**REFERENCE DOCUMENT**

|  |  |
| --- | --- |
| **Title** | **Reference** |
| **Rs232\_syscon** | https://github.com/fabriziotappero/ip-cores/blob/system\_controller\_rs232\_system\_controller/web\_uploads/rs232\_syscon.pdf |

Table of contents

[1. Introduction 5](#_Toc181092534)

[1.1. Presentation of the document 5](#_Toc181092535)

[1.2. Principle and application domain 5](#_Toc181092536)

[1.3. Hardware 5](#_Toc181092537)

[2. Global description of module X 6](#_Toc181092538)

[2.1. Architecture of module X 6](#_Toc181092539)

[2.2. Functional diagrap of module X 6](#_Toc181092540)

[2.3. Specifications of module X : 6](#_Toc181092541)

[2.4. Module X interface description (IO Ports) 7](#_Toc181092542)

[2.5. List of registers: 7](#_Toc181092543)

[2.6. Registers Description: 7](#_Toc181092544)

[3. Implementation of module X: 8](#_Toc181092545)

[3.1. Submodule 1: 8](#_Toc181092546)

[Description of submodule 1 : 8](#_Toc181092547)

[Functional description of submodule 1 : 8](#_Toc181092548)

[Submodule 1 i nterface description: 8](#_Toc181092549)

[3.2. Submodule 2: 9](#_Toc181092550)

[Description of submodule 2 : 9](#_Toc181092551)

[Functional description of submodule 2 : 9](#_Toc181092552)

[Submodule 2 i nterface description: 9](#_Toc181092553)

[4. Simulations : 10](#_Toc181092554)

[APPENDIX 11](#_Toc181092555)

List of figures

***Aucune entrée de table d'illustration n'a été trouvée.***

List of tables

[Table 1 : Module X I/O Interface 7](#_Toc181094301)

[Table 2 : Submodule 1 I/O Interface 8](#_Toc181094302)

[Table 3 : Submodule 2 I/O Interface 9](#_Toc181094303)

# Introduction

## Presentation of the document

This document describes some of the design features of “rs232\_syscon.v” (a softcore written in Verilog.) It is intended to facilitate new users in understanding what is available in the core, and how to use it. Also, the Verilog code itself is replete with comments, so that additional insights into the operation of this core can be gained by reviewing the code.

## Principle and application domain

The name “rs232\_syscon” comes from rs232 (serial communication standard) and an abbreviated contraction of “system controller.” The core actually uses LVTTL levels for the serial connection, and the user is responsible for providing level shifting translators to achieve rs232 standard voltage levels.

The rs232\_syscon project was conceived on May 30, 2001. It’s purpose was to develop a “serial-port-to-bus-interface” core suitable for debugging some other ps2\_mouse and ps2\_keyboard interface cores which were under development at the time… Those other cores were quickly completed, but the actual development of rs232\_syscon was more complicated than originally thought, and it ended up taking longer to complete. Luckily, as its development progressed, rs232\_syscon became simpler instead of more complicated.

After many hours of debugging and coding, the rs232\_syscon core is now functional, and it has been successfully used to test out memory blocks and register blocks as part of a “system on a chip” (SOC) design effort.

This document describes the following:

* The connection diagram of rs232\_syscon.
* The command syntax of rs232\_syscon.
* The serial interface BAUD rate generators.

## Targeted hardware

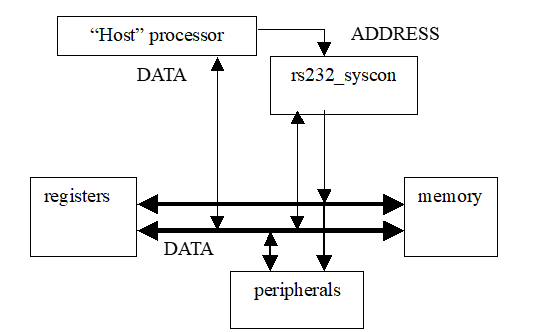
In reality, rs232\_syscon is a simple core to use – once it is connected to the data and address buses and the BAUD rate is adjusted, it can be used immediately. It does not contain any architecture specific blocks, so it easily ports to different FPGA and even ASIC platforms.

# Global description of rs232\_syscon

## Architecture

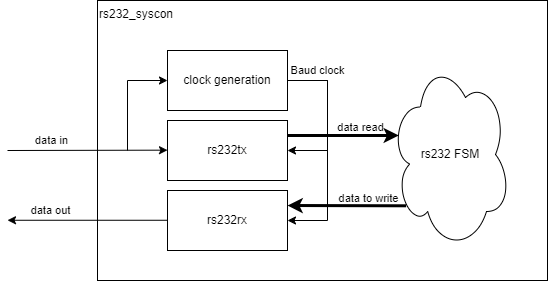
A top-level block diagram of rs232\_syscon being used in a system is shown in figure2.1 below.

Since the address bus is not bidirectional, it is an output from the host processor, and also an output from rs232\_syscon. In order to select which device gets to drive the address bus, a multiplexer is implemented inside of rs232\_syscon. This introduces some additional delay in the address bus, which is considered an acceptable tradeoff in exchange for the enhanced debugging capability of rs232\_syscon. Besides, when debugging is completed, the mux can be removed (or “hard wired” by a parameter at compile time, which will also result in the eventual removal of the mux because of optimization in the synthesis and routing tools…)



## Functional diagram

The diagram below shows the internal architecture of the rs232\_syscon



## Specifications

The data bus is implemented as a tri-state bus, so that it can be bi-directional without requiring the use of multiplexers. The designers of rs232\_syscon were aware that the Wishbone standard seems to encourage a split data bus (dat\_i for input data and dat\_o for output data) but found that the Wishbone standard also allows for tri-state connections (See Wishbone spec. page 66). The tri-state data bus was chosen in order to reduce the number of internal interconnects needed to implement the bus. If a tri-state bus is unacceptable for your application, the rs232\_syscon block can be easily modified to add “dat\_i” and “dat\_o” ports in place of the existing “dat\_io” port, and the tri-state buffering can be removed. This is not difficult for a Verilog programmer to accomplish, and it does not require any major functional modifications to the rs232\_syscon block. The same handshaking structure that is used for address bus multiplexing (“master\_br\_o” and “master\_bg\_i”) could also be used to control the data bus multiplexers.

The handshaking scheme in rs232\_syscon allows the rs232\_syscon to request access to the bus from the normal bus master. This is accomplished through the “master\_br\_o” and “master\_bg\_i” pins. Once the bus request (br) is detected at the normal bus master, it should finish the current operation, and then assert and keep asserting “master\_bg” to rs232\_syscon. As long as the bus grant (bg) line is asserted into rs232\_syscon, then rs232\_syscon will know that it has control of the bus. Also, when rs232\_syscon finishes generating its bus cycles, it does not check or wait for the bus grant line to be deasserted. Therefore, those who wish to test peripherals, memory or registers without another master on the bus, can simply tie “master\_bg\_i” high, or just connect it to the rs232\_syscon’s “master\_br\_o” and forget about that handshaking interface.

The bus cycles generated by rs232\_syscon are one clock long. The clock which is used with rs232\_syscon can vary up to the maximum speed allowed by the architecture in which it is being used. In a Xilinx SpartanII device (XC2S200) it synthesized with a maximum clock speed of around 45 MHz, although most of the testing was done at around 25 MHz. The length of each bus cycle is extended until the “ack\_i” signal is received by rs232\_syscon. If the watchdog timer expires before “ack\_i” is received, then a bus error message is generated for the user. Similarly, if the “err\_i” signal is received, then a bus error message is generated for the user.

## Rs232\_syscon interface description (IO Ports)

Table 1 : Module X I/O Interface

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Name** | **Size** | **Function** |
| **Inputs** | clk\_i | 1 | Clock input |
| reset\_i | 1 | Resets rs232\_syscon unit |
| master\_bg\_i | 1 | Grants bus to rs232\_syscon |
| ack\_i | 1 | Wishbone bus cycle acknowledge |
| err\_i | 1 | Wishbone bus cycle error |
| rs232\_rxd\_i | 1 | rs232 serial port data input |
| **Input/Output** | dat\_io | parameter | data bus (tri-state) |
| **Outputs** | rst\_o | 1 | Wishbone reset output |
| master\_br\_o | 1 | Requests bus for rs232\_syscon |
| stb\_o | 1 | Wishbone strobe output |
| cyc\_o | 1 | Wishbone cycle output (wired to stb\_o in this version) |
| adr\_o | parameter | address bus |
| we\_o | 1 | Wishbone write enable output |
| rs232\_txd\_o | 1 | rs232 serial port data output |