ELSYS DESIGN

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
    1.1. Presentation of the document .... **Erreur ! Signet non défini.**
    1.2. Principle and application domain 5
    1.3. Hardware 6
2. Global description of module X 6
    2.1. Architecture of module X 6
    2.2. Functional diagrap of module X 7
    2.3. Specifications of module X : 7
    2.4. Module X interface description (IO Ports) 8
    2.5. List of registers: .... **Erreur ! Signet non défini.**
    2.6. Registers Description: .... **Erreur ! Signet non défini.**
3. Implementation of module X: .... **Erreur ! Signet non défini.**
    3.1. Submodule 1: .... **Erreur ! Signet non défini.**
        Description of submodule 1 : .... **Erreur ! Signet non défini.**
        Functional description of submodule 1 : .... **Erreur ! Signet non défini.**
        Submodule 1 i   nterface description: .... **Erreur ! Signet non défini.**
    3.2. Submodule 2: .... **Erreur ! Signet non défini.**
        Description of submodule 2 : .... **Erreur ! Signet non défini.**
        Functional description of submodule 2 : .... **Erreur ! Signet non défini.**
        Submodule 2 i   nterface description: .... **Erreur ! Signet non défini.**
4. Simulations : .... **Erreur ! Signet non défini.**

APPENDIX .... **Erreur ! Signet non défini.**

List of figures

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

List of tables

Table 1 : Module X I/O Interface *8*
Table 2 : Submodule 1 I/O Interface .... ***Erreur ! Signet non défini.***
Table 3 : Submodule 2 I/O Interface .... ***Erreur ! Signet non défini.***

1. Introduction

1.1. 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.

1.2. 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.

1.3. Targeted hardware

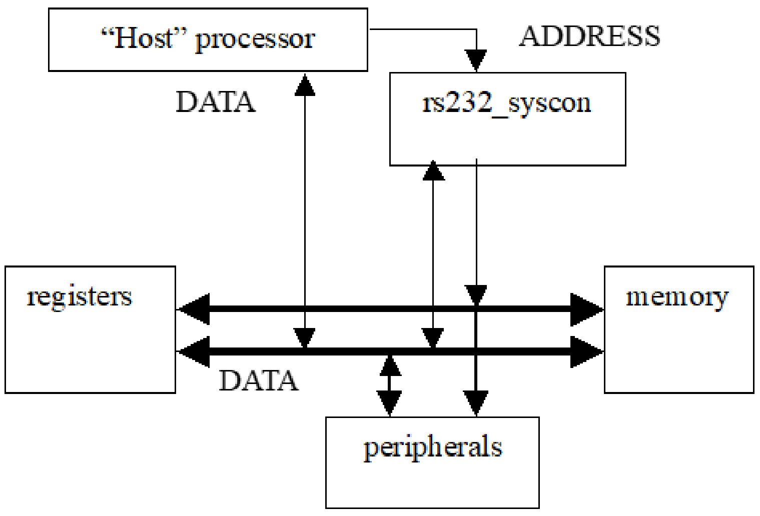
In reality, rs232_syscon is a simple core to use – once it is connected to the data and address buses and the BAUD rate is adjusted, it can be used immediately. It does not contain any architecture specific blocks, so it easily ports to different FPGA and even ASIC platforms.

2. Global description of rs232_syscon

2.1. Architecture

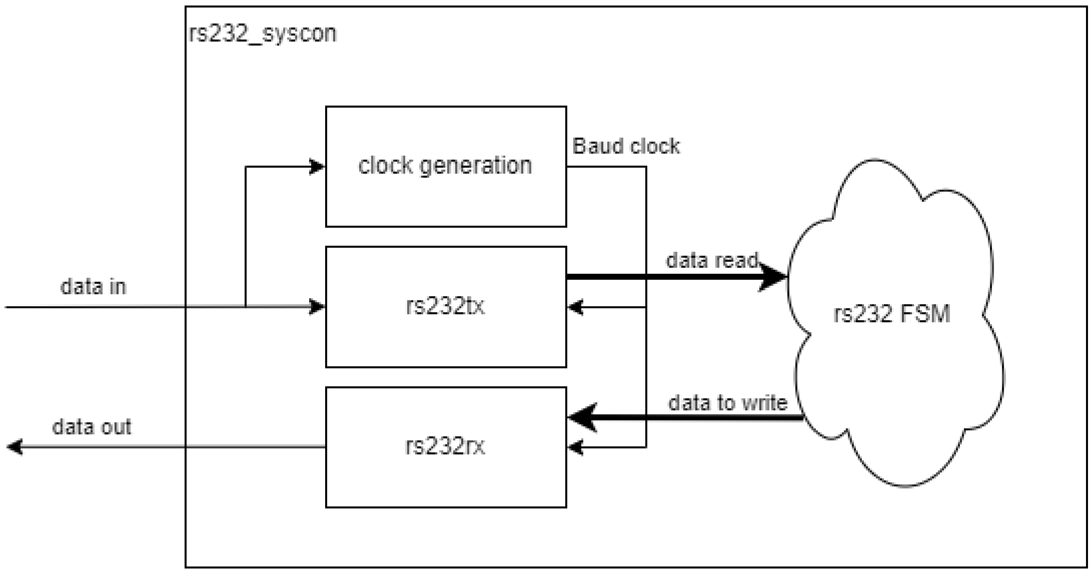
A top-level block diagram of rs232_syscon being used in a system is shown in figure2.1 below.

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…)



2.2. Functional diagram

The diagram below shows the internal architecture of the rs232_syscon



2.3. 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.

2.4. 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