UART to Bus Core Specifications

Written for publication on:

File name:	*UART to Bus Core Specifications*
Version:	*1.0*
Creation Date:	*February 12, 2010*
Update Date:	*February 25, 2012*
Author:	*Moti Litochevski*

Table of Contents

Index of Tables

Index of Figures

1. Preface

1.1. Scope

This document describes the UART to Bus IP core operation, architecture and interfaces.

1.2. Revision History

Rev	*Date*	*Author*	*Description*
0.1	02/13/10	Moti Litochevski	First Draft
0.2	04/02/10	Moti Litochevski	Adding test bench description section
0.3	04/15/11	Moti Litochevski	Added Lattice MachXO device utilization provided by Paul V. Shatov.
1.0	02/25/12	Moti Litochevski	Updated interface description and synthesis results after adding bus request/grant mechanism.

1.3. Abbreviations

UART	Universal Asynchronous Receiver / Transmitter

2. Introduction

The UART to Bus IP Core is a simple command parser that can be used to access an internal bus via a UART interface. This core can be used during initial board debugging or as a permanent solution when high speed interfaces are not required. The internal bus is designed with address bus of 16 bits and data bus of 8 bits.

The core implements a very basic UART transmit & receive blocks which share a common baud rate generator and a command parser. The parser supports two modes of operation: text mode commands and binary mode commands. Text mode commands are designed to be used with a hyper terminal software and enable easy access to the internal bus. Binary mode commands are more efficient and also support buffered read & write operations with or without automatic address increment.

The core was verified using Icarus Verilog simulator with two test benches: the first tests the text mode protocol and the second tests the binary protocol. The test bench uses a register file model to simulate write and read operations.

The following table summarizes the synthesis results of the core for different FPGA families.

Manufacturer	*Family*	*Device*	*Device Utilization*	*Elements Utilization*	*Fmax*
Xilinx	Spartan 3	xc3s50-5pq208	24.00%	186 Slices	>150MHz
Xilinx	Virtex 5	xc5vlx30-3ff324	2.00%	111 Slices	>200MHz
Altera	Cyclone III	ep3c5f256c6	5.00%	255 LEs	>200MHz
Altera	Startix III	ep3sl50f484c2	<1%	167 Registers 218 ALUTs	>200MHz
Lattice[1]	MachXO	LCMXO2280C-4T144C	10.00%	116 Slices	> 100MHz

Table 1: Synthesis Results for Different FPGA Devices

The above results where obtained using the following software versions:

- Xilinx ISE Webpack 11.4
- Altera Quartus Web Edition 9.1sp2

NOTE:
*The UART to Bus core is **not** Wishbone compatible although modifying it for Wishbone is probably possible.*

1 Lattice device synthesis results provided by Paul V. Shatov. Results provided before adding bus request/grant mechanism.

3. Architecture

The UART to Bus architecture is fairly simple. The core is includes a UART interface module, which includes both receive and transmit modules, and the command parser. The following figure depicts a block diagram of the core.

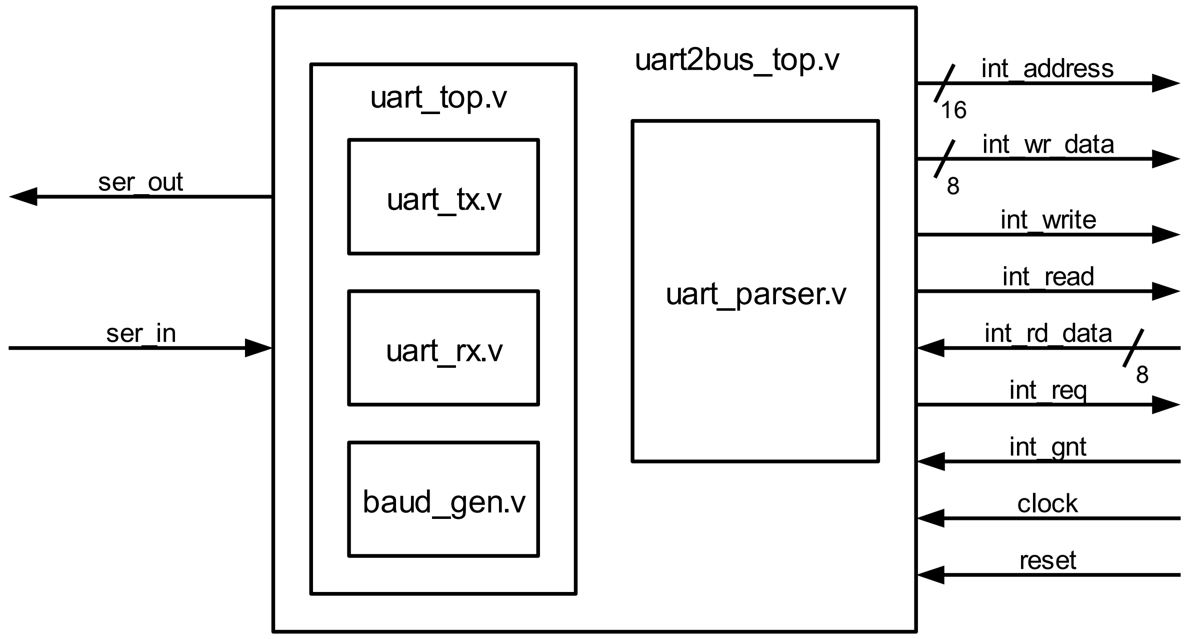

Figure 1: UART to Bus Core Block Diagram

The UART interface is based on an implementation found in the c16 project in OpenCores (http://www.opencores.org/project,c16). The interface includes a UART receive and transmit modules that share a single baud rate generator module. The baud rate is set using two constants defined at the core top module which are calculated as follows:

$$\text{D\_BAUD\_FREQ} = \frac{16 \cdot BaudRate}{gcd(GlobalClockFreq, 16 * BaudRate)}$$

$$\text{D\_BAUD\_LIMIT} = \frac{GlobalClockFreq}{gcd(GlobalClockFreq, 16 \cdot BaudRate)} - \text{D\_BAUD\_FREQ}$$

A short Scilab script which calculates the above parameters is added under the “scilab” directory with the core files.

The interface between the “uart_parser.v” module and the “uart_top.v” is very simple and uses only five signals. For cases where the UART interface is not possible or another interface is preferred, the “uart_parser.v” module can be used as is with a different interface implementation.

uart_top是uart功能模块，串行输入输出啥的
uart_parser是接口模块，改成总线形式

4. Operation

This section describes the protocols used to access the internal bus from the UART interface. As mentioned above the parser supports two modes of operation: text & binary commands. To distinguish between the two protocols all binary commands start with a value of zero which will not be sent when using the text protocol. The following drawing depicts a simplified state machine of the parser. The figure does not include some transitions used to abort illegal command sequences.

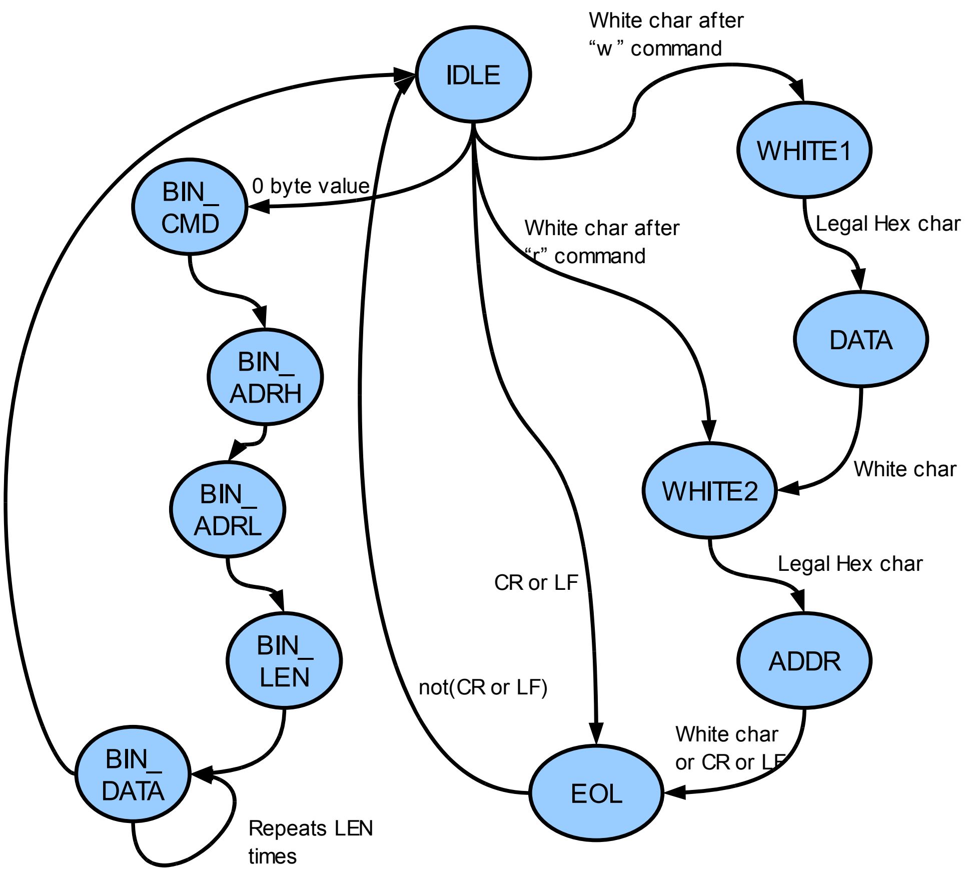

Figure 2: Parser State Diagram

In the state diagram above the states on the right are used for the text mode protocol and the states on the left are used for the binary mode protocol.

The following sub-sections describe each of the protocols.

4.1. Text Mode Protocol

识别字符

The text mode protocol includes only two commands: address read and address write. All values are in HEX format. The parser checks for both upper and lower characters, detects both space (0x20) and tab (0x09) as white spaces and both LF (0x0A) and CR (0x0D) as end of line. Commands which do not follow the required sequence or contain illegal characters are aborted.

Address read command format:

1st Field	*2nd Field*	*3rd Field*	*4th Field*
'R' or 'r'	White space – single or multiple	**Address** to read in Hex format, for example: 'D5A0'.	End of line, CR or LF characters.

Table 2: Text Protocol Read Command Format

On the reception of the EOL character the core will read the given address and transmit the read value in two Hex characters followed by CR & LF characters.

先发数据后发地址

Address write command format:

1st Field	*2nd Field*	*3rd Field*	*4th Field*	*5th Field*	*6th Field*
'W' or 'w'	White space – single or multiple	**Data** to write in Hex format, for example: '4F'.	White space – single or multiple	**Address** to write in Hex format, for example: 'D5A0'.	End of line, CR or LF characters.

Table 3: Text Protocol Write Command Format

On the reception of the EOL character the core will write the given address. No transmission is sent back to the sender.

4.2. Binary Mode Protocol

The binary mode protocol is much more efficient since it parsers the sent values and does not need to convert them from ASCII. The protocol uses a single command which can be either read or write operation of a configurable number of bytes. Commands are optionally acknowledged on completion. The binary command format is detailed in Table 4.

Byte #	*Name*	*Description*
1	Binary Command Indicator	Constant zero byte prefix to indicate the start of a binary command.
2	Command	This byte is the command selection and options byte and has the following bit assignment: *Bit #* | *Description* [7:6] | Not used [5:4] | Command selection: **2'b00** = NOP command, sends ACK if requested. **2'b01** = Read command. **2'b10** = Write command. [3:2] | Not used

		1 — Address Auto Increment Enable: Set to **0** to **enable** address auto increment. Set to **1** to **disable** address auto increment. 0 — Send ACK Flag: Set to **1** to send ACK byte at command completion. Notes: • The NOP command can be used to verify that the core is responding on the UART. • The address auto increment option increments the internal bus address after every bus read or write operation. This is required when reading a buffer from a RAM. When reading data from a FIFO it is more convenient to turn auto address increment off.
3	Address High Byte	High 8-bits of the 16-bit operation start address.
4	Address Low Byte	Low 8-bits of the 16-bit operation start address.
5	Length	This byte indicates the length of buffer to read or write. Note that a value of 0 signs a buffer length of 256 bytes which is the maximum buffer length.
6 to 5+LEN	Data	This field only exists in write commands and it contains the data to be written. The data length should equal the length indicated by the Length field.

Table 4: Binary Protocol Command Format

In response to read command and when an acknowledge byte is requested the following binary message is transmitted by the core.

Byte #	*Name*	*Description*
1 to LEN	Data	This field only exists in response to read commands and it contains the data read. The data length equals the length indicated by the Length field in the command.
LEN+1	ACK	The value of the ACK byte is 0x5a and is only sent if bit 0 in the command byte is set.

Table 5: Binary Protocol Return Message Format

5. Core Interfaces

The following table summarizes the core interface ports.

Name	*Direction*	*Width*	*Description*
clock	input	1	Global core clock signal.
reset	input	1	Global core asynchronous reset, active high.
ser_in	input	1	UART serial input to the core.
ser_out	output	1	UART serial output from the core.
int_address	output	16	Internal address bus.
int_wr_data	output	8	Data value to write to address on the clock cycle with int_write active.
int_write	output	1	An active high write control signal. This signal shall only be valid for a single clock cycle per address to be written to.
int_read	output	1	An active high read control signal. This signal shall only be valid for a single clock cycle per address to be read from.
int_rd_data	input	8	Data value read from address. This signal is sampled by the core on the next clock cycle following int_read signal active cycle.
int_req	output	1	Internal bus access request signal. The core will assert this signal to request access to the internal bus before every bus access. This signal will be asserted until bus access is granted.
int_gnt	input	1	Internal bus grant signal. The core will release the request signal and execute the bus access when this signal is asserted. To disable bus request/grant mechanism set this to logic '1'.

异步高电平有效

一堆总线接口

写使能

Table 6: Core Interfaces Description

Note:
The port direction in the table above is as defined in the core top module.

6. Test Bench Description

仿真器是啥玩意！环境不能用……

The 'verilog\bench' directory contains two test benches files and required tasks and modules. Compilation batch files are included in the 'verilog\sim\icarus' directory used to simulate the core using Icarus Verilog. The directory inclues two compilation batch files: one for binary mode protocol simulation and the second for text mode protocol simulation.

The directory also includes batch file to run the simulation, 'run.bat', and another to call gtkwave to view the simulation VCD output file.

Note that for binary mode protocol simulation the test bench reads the commands from 'test.bin' file also included in the directory. The file structure is straight forward and is explained in the respective test bench file.

这个uart的功能:
每个时钟的上升沿，看它
int_address,int_rd_data,int_wr_data，以及int_read，和
int_write决定其的操作情况