UART16550 Core Technical Manual

Jacob Gorban

Jan 13, 2021

CONTENTS

Revision 0.6.1

Jacob Gorban - gorban@opencores.org

w/ some updates by Luke E. McKay

CHAPTER

ONE

INTRODUCTION

The UART (Universal Asynchronous Receiver/Transmitter) core provides serial communication capabilities, which allow communication with modem or other external devices, like another computer using a serial cable and RS232 protocol. This core is designed to be maximally compatible with the industry standard National Semiconductors' 16550A device.

Features:

1. WISHBONE interface in 32-bit or 8-bit data bus modes (selectable)
2. FIFO only operation
3. Register level and functionality compatibility with NS16550A (but not 16450).
4. Debug Interface in 32-bit data bus mode.

CHAPTER

TWO

IO PORTS

2.1 WISHBONE interface signals

Port	Width	Direction	Description
CLK	1	Input	Block's clock input
WB_RST_I	1	Input	Asynchronous Reset
WB_ADDR_I	5 or 3	Input	Used for register selection
WB_SEL_I	4	Input	Select signal
WB_DAT_I	32 or 8	Input	Data input
WB_DAT_O	32 or 8	Output	Data output
WB_WE_I	1	Input	Write or read cycle selection
WB_STB_I	1	Input	Specifies transfer cycle
WB_CYC_I	1	Input	A bus cycle is in progress
WB_ACK_O	1	Output	Acknowledge of a transfer

2.2 Other internal signals

Port	Width	Direction	Description
INT_O	1	Output	Interrupt output
BAUD_O	1	Output	Optional baud rate output signal. The signal here is the 16 x actual baud rate. It is enabled if UART_HAS_BAUDRATE_OUTPUT is defined

2.3 External (off-chip) connections

Port	Width	Direction	Description
STX_PAD_O	1	Output	The serial output signal
SRX_PAD_I	1	Input	The serial input signal
RTS_PAD_O	1	Output	Request To Send
DTR_PAD_O	1	Output	Data Terminal Ready
CTS_PAD_I	1	Input	Clear To Send
DSR_PAD_I	1	Input	Data Set Ready
RI_PAD_I	1	Input	Ring Indicator
DCD_PAD_I	1	Input	Data Carrier Detect

CHAPTER

THREE

CLOCKS

Clocks table:

Name	Source	Rates (MHz)	Description		
		Max	Min	Resolu-tion	
clk	WISHBONE bus	1258Mhz for 1200 bps	3.6864 for 115200 bps		WISHBONE clock

Due to the 16-bit clock divider 1258Mhz is the maximum frequency if the minimum bps required is 1200. Similarly due to the core design a minimum frequency of 3.6864Mhz is required to obtain a maximum of 115200 bps. These translate to the maximum bps is clock rate / 32 and the minimum bps is the clock rate divided by 1024. The 1258Mhz shown is the (data rate + 2.376% error) * 1024 which pushes the limt on an acceptable baud mismatch.

CHAPTER

FOUR

REGISTERS

Registers list:

Name	Address	Width	Access	Description
Receiver Buffer	0	8	R	Receiver FIFO output
Transmitter Holding Register (THR)	0	8	W	Transmit FIFO input
Interrupt Enable	1	8	RW	Enable/Mask interrupts generated by the UART
Interrupt Identification	2	8	R	Get interrupt information
FIFO Control	2	8	W	Control FIFO options
Line Control Register	3	8	RW	Control connection
Modem Control	4	8	W	Controls modem
Line Status	5	8	R	Status information
Modem Status	6	8	R	Modem Status

In addition, there are 2 Clock Divisor registers that together form one 16-bit.

The registers can be accessed when the 7th (DLAB) bit of the Line Control Register is set to '1'. At this time the above registers at addresses 0-1 can't be accessed.

Name	Address	Width	Access	Description
Divisor Latch Byte 1 (LSB)	0	8	RW	The LSB of the divisor latch
Divisor Latch Byte 2	1	8	RW	The MSB of the divisor latch

When using 32-bit data bus interface, additional read-only registers are available for debug purposes:

Name	Address	Width	Access	Description
Debug 1	8	32	R	First debug register
Debug 2	12	32	R	Second debug register

4.1 Interrupt Enable Register (IER)

This register allows enabling and disabling interrupt generation by the UART.

Bit #	Access	Description
0	RW	Received Data available interrupt '0' - disabled '1' - enabled
1	RW	Transmitter Holding Register empty interrupt '0' - disabled '1' - enabled
2	RW	Receiver Line Status Interrupt '0' - disabled '1' - enabled
3	RW	Modem Status Interrupt '0' - disabled '1' - enabled
7-4	RW	Reserved. Should be logic '0'.

Reset Value: 00h

4.2 Interrupt Identification Register (IIR)

The IIR enables the programmer to retrieve what is the current highest priority pending interrupt.

Bit 0 indicates that an interrupt is pending when it's logic '0'. When it's '1' - no interrupt is pending.

The following table displays the list of possible interrupts along with the bits they enable, priority, and their source and reset control.

Bit 3	**Bit 2**	**Bit 1**	**Priority**	**Interrupt Type**	**Interrupt Source**	**Interrupt Reset Control**
0	1	1	1st	Receiver Line Status	Parity, Overrun or Framing errors or Break Interrupt	Reading the Line Status Register
0	1	0	2nd	Receiver Data available	FIFO trigger level reached	FIFO drops below trigger level
1	1	0	2nd	Timeout Indication	There's at least 1 character in the FIFO but no character has been input to the FIFO or read from it for the last 4 Char times.	Reading from the FIFO (Receiver Buffer Register)
0	0	1	3rd	Transmitter Holding Register empty	Transmitter Holding Register Empty	Writing to the Transmitter Holding Register or reading IIR.
0	0	0	4th	Modem Status	CTS, DSR, RI or DCD.	Reading the Modem status register.

Bits 4 and 5: Logic '0'.

Bits 6 and 7: Logic '1' for compatibility reason.

Reset Value: C1h

4.3 FIFO Control Register (FCR)

The FCR allows selection of the FIFO trigger level (the number of bytes in FIFO required to enable the Received Data Available interrupt). In addition, the FIFOs can be cleared using this register.

Bit #	Access	Description
0	W	Ignored (Used to enable FIFOs in NS16550D). Since this UART only supports FIFO mode, this bit is ignored.
1	W	Writing a '1' to bit 1 clears the Receiver FIFO and resets its logic. But it doesn't clear the shift register, i.e. receiving of the current character continues.
2	W	Writing a '1' to bit 2 clears the Transmitter FIFO and resets its logic. The shift register is not cleared, i.e. transmitting of the current character continues.
5-3	W	Ignored
7-6	W	Define the Receiver FIFO Interrupt trigger level '00' - 1 byte '01' - 4 bytes '10' - 8 bytes '11' - 14 bytes

Reset Value : 11000000b

4.4 Line Control Register (LCR)

The line control register allows the specification of the format of the asynchronous data communication used. A bit in the register also allows access to the Divisor Latches, which define the baud rate. Reading from the register is allowed to check the current settings of the communication.

Bit #	Access	Description
1-0	RW	Select number of bits in each character ‘00’ - 5 bits ‘01’ - 6 bits ‘10’ - 7 bits ‘11’ - 8 bits
2	RW	Specify the number of generated stop bits ‘0’ - 1 stop bit ‘1’ - 1.5 stop bits when 5-bit character length selected and 2 bits otherwise Note that the receiver always checks the first stop bit only.
3	RW	Parity Enable ‘0’ - No parity ‘1’ - Parity bit is generated on each outgoing character and is checked on each incoming one.
4	RW	Even Parity select ‘0’ - Odd number of ‘1’ is transmitted and checked in each word (data and parity combined). In other words, if the data has an even number of ‘1’ in it, then the parity bit is ‘1’. ‘1’ - Even number of ‘1’ is transmitted in each word.
5	RW	Stick Parity bit. ‘0’ - Stick Parity disabled ‘1’ - If bits 3 and 4 are logic ‘1’, the parity bit is transmitted and checked as logic ‘0’. If bit 3 is ‘1’ and bit 4 is ‘0’ then the parity bit is transmitted and checked as ‘1’.
6	RW	Break Control bit ‘1’ - the serial out is forced into logic ‘0’ (break state). ‘0’ - break is disabled
7	RW	Divisor Latch Access bit. ‘1’ - The divisor latches can be accessed ‘0’ - The normal registers are accessed

Reset Value: 00000011b

4.5 Modem Control Register (MCR)

The modem control register allows transferring control signals to a modem connected to the UART.

Bit #	Access	Description
0	W	Data Terminal Ready (DTR) signal control ‘0’ - DTR is ‘1’ ‘1’ - DTR is ‘0’
1	W	Request To Send (RTS) signal control ‘0’ - RTS is ‘1’ ‘1’ - RTS is ‘0’
2	W	Out1. In loopback mode, connected Ring Indicator (RI) signal input
3	W	Out2. In loopback mode, connected to Data Carrier Detect (DCD) input.
4	W	Loopback mode ‘0’ - normal operation ‘1’ - loopback mode. When in loopback mode, the Serial Output Signal (STX_PAD_O) is set to logic ‘1’. The signal of the transmitter shift register is internally connected to the input of the receiver shift register. The following connections are made: DTR -> DSR RTS -> CTS Out1 -> RI Out2 -> DCD
7-5	W	Ignored

Reset Value: 0

4.6 Line Status Register (LSR)

Bit #	Access	Description
0	R	Data Ready (DR) indicator. '0' - No characters in the FIFO '1' - At least one character has been received and is in the FIFO.
1	R	Overrun Error (OE) indicator '1' - If the FIFO is full and another character has been received in the receiver shift register. If another character is starting to arrive, it will overwrite the data in the shift register but the FIFO will remain intact. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' - No overrun state
2	R	Parity Error (PE) indicator '1' - The character that is currently at the top of the FIFO has been received with parity error. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' - No parity error in the current character
3	R	Framing Error (FE) indicator '1' - The received character at the top of the FIFO did not have a valid stop bit. Of course, generally, it might be that all the following data is corrupt. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' - No framing error in the current character
4	R	Break Interrupt (BI) indicator '1' -A break condition has been reached in the current character. The break occurs when the line is held in logic 0 for a time of one character (start bit + data + parity + stop bit). In that case, one zero character enters the FIFO and the UART waits for a valid start bit to receive next character. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' - No break condition in the current character
5	R	Transmit FIFO is empty. '1' - The transmitter FIFO is empty. Generates Transmitter Holding Register Empty interrupt. The bit is cleared when data is being been written to the transmitter FIFO. '0' - Otherwise
6	R	Transmitter Empty indicator. '1' - Both the transmitter FIFO and transmitter shift register are empty. The bit is cleared when data is being been written to the transmitter FIFO. '0' - Otherwise
7	R	'1' - At least one parity error, framing error or break indications have been received and are inside the FIFO. The bit is cleared upon reading from the register. '0' - Otherwise.

4.7 Modem Status Register (MSR)

The register displays the current state of the modem control lines. Also, four bits also provide an indication in the state of one of the modem status lines. These bits are set to '1' when a change in corresponding line has been detected and they are reset when the register is being read.

Bit #	Access	Description
0	R	Delta Clear To Send (DCTS) indicator '1' - The CTS line has changed its state.
1	R	Delta Data Set Ready (DDSR) indicator '1' - The DSR line has changed its state.
2	R	Trailing Edge of Ring Indicator (TERI) detector. The RI line has changed its state from low to high state.
3	R	Delta Data Carrier Detect (DDCD) indicator '1' - The DCD line has changed its state.
4	R	Complement of the CTS input or equals to RTS in loopback mode.
5	R	Complement of the DSR input or equals to DTR in loopback mode.
6	R	Complement of the RI input or equals to Out1 in loopback mode.
7	R	Complement of the DCD input or equals to Out2 in loopback mode.

4.8 Divisor Latches

The divisor latches can be accessed by setting the 7^{th} bit of LCR to '1'. You should restore this bit to '0' after setting the divisor latches in order to restore access to the other registers that occupy the same addresses. The 2 bytes form one 16-bit register, which is internally accessed as a single number. You should therefore set all 2 bytes of the register to ensure normal operation. The register is set to the default value of **0 on reset**, which disables all serial I/O operations in order to ensure explicit setup of the register in the software. The value set should be equal to (system clock speed) / (16 x desired baud rate).

The internal counter starts to work when the LSB of DL is written, so when setting the divisor, write the MSB first and the LSB last.

4.9 Debug 1

This register is only available when the core has 32-bit data bus and 5-bit address bus.

It is read only and is provided for debugging purposes of chip testing as it is not part of the original UART16550 device specifications. Reading from the does not influence core's bahaviour.

Bit #	Access	Description
7-0	R	Line Status Register value.
11-8	R	Interrupt Enable Register value (bits 3-0).
15-12	R	Interrupt Identifier Register value (bits 3-0).
23-16	R	Line Control Register value.
31-24	R	Modem Status Register value.

4.10 Debug 2

This register is only available when the core has 32-bit data bus and 5-bit address bus.

It is read only and is provided for debugging purposes of chip testing as it is not part of the original UART16550 device specifications. Reading from the does not influence core's bahaviour.

Bit #	Access	Description
2-0	R	Transmitter FSM state
7-3	R	Number of characters in Transmitter FIFO (tf_count)
11-8	R	Receiver FSM state
16-12	R	Number of characters in Receiver FIFO (rf_count)
18-17	R	Modem Control Register value (bits 4-0)
23-19	R	FIFO Control Register value (bits 7-6)
31-24	R	Reserved. Returned value is 0.

CHAPTER

FIVE

OPERATION

This UART core is very similar in operation to the standard 16550 UART chip with the main exception being that only the FIFO mode is supported. The scratch register is removed, as it serves no purpose.

This core can operate in 8-bit data bus mode or in 32-bit bus mode, which is now the default mode.

The 32-bit mode is fully WISHBONE compatible and it uses the WISHBONE [SEL_I] signal to properly receive and return 8-bit data on 32-bit data bus. The 8-bit version might have problems in various WISHBONE implementations because a 32-bit master reading from 8-bit bus can expect data on different bytes of the 4-byte word, depending on the register address.

Also, in 32-bit data bus mode, the [ADR_I] is 5 and not 3 bits wide.

In addition, in the 32-bit data bus mode a debug interface is present in the system. This interface has 2 32-bit registers that can be read to provide non-intrusive look into the core's registers and other internal values of importance.

The selection between 32- and 8-bits data bus modes is performed by defining DATA_BUS_WIDTH_8 in uart_defines.v, uart_top.v or on the compiler/synthesizer tool command line.

5.1 Initialization

Upon reset the core performs the following tasks:

1. The receiver and transmitter FIFOs are cleared.
2. The receiver and transmitter shift registers are cleared
3. The Divisor Latch register is set to 0.
4. The Line Control Register is set to communication of 8 bits of data, no parity, 1 stop bit.
5. All interrupts are disabled in the Interrupt Enable Register.

For proper operation, perform the following:

1. Set the Line Control Register to the desired line control parameters. Set bit 7 to '1' to allow access to the Divisor Latches.
2. Set the Divisor Latches, MSB first, LSB next.
3. Set bit 7 of LCR to '0' to disable access to Divisor Latches. At this time the transmission engine starts working and data can be sent and received.
4. Set the FIFO trigger level. Generally, higher trigger level values produce less interrupt to the system, so setting it to 14 bytes is recommended if the system responds fast enough.
5. Enable desired interrupts by setting appropriate bits in the Interrupt Enable register.

Remember that (Input Clock Speed)/(Divisor Latch value) = 16 x the communication baud rate. Since the protocol is asynchronous and the sampling of the bits is performed in the perceived middle of the bit time, it is highly immune to small differences in the clocks of the sending and receiving sides, yet no such assumption should be made when calculating the Divisor Latch values.

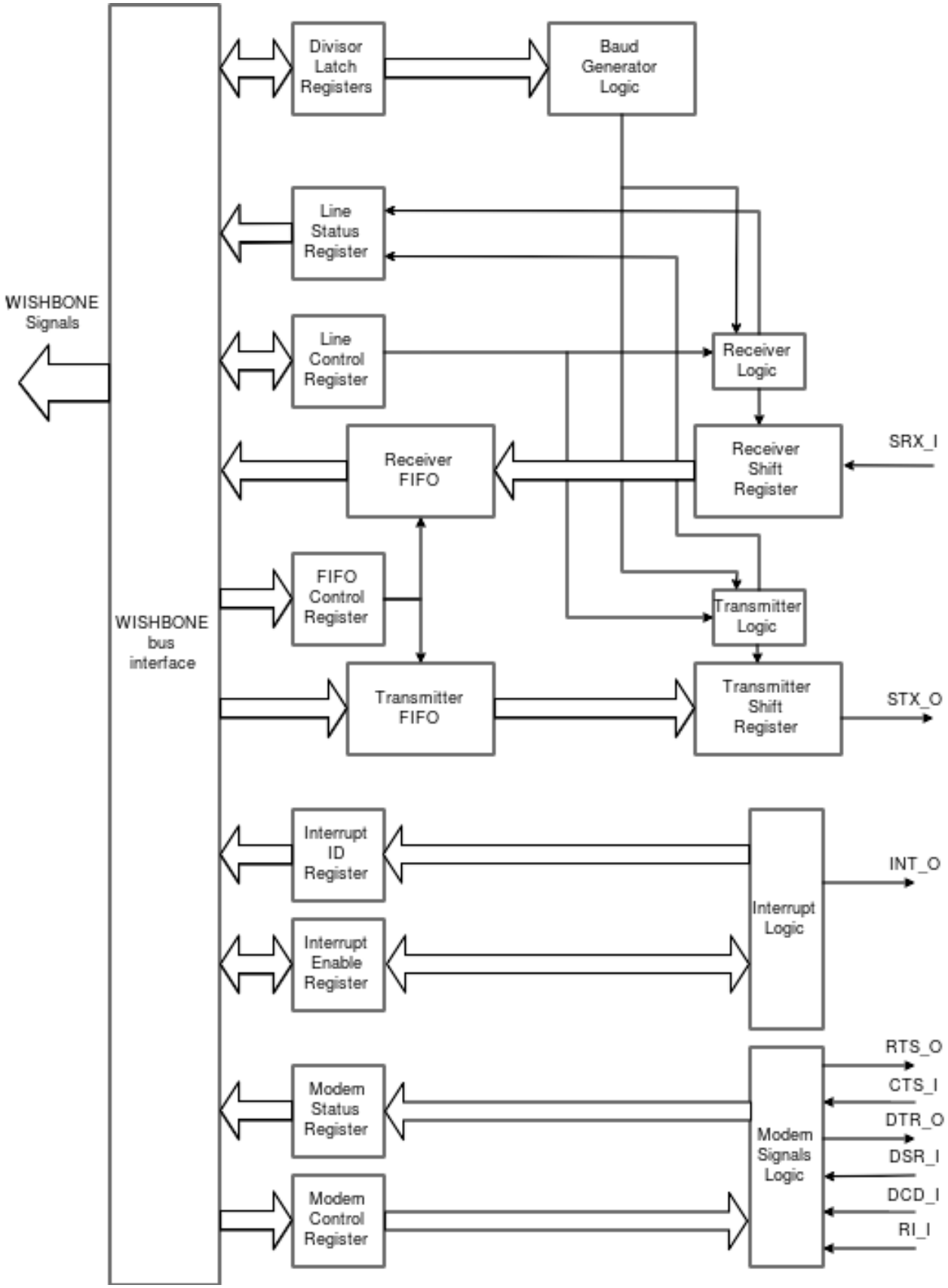
CHAPTER

SIX

ARCHITECTURE

The core implements the WISNBONE SoC bus interface for communication with the system. It has an 8-bit data bus for compatibility reason. The core requires one interrupt. It requires 2 pads in the chip (serial in and serial out) and, optionally, another six modem control signals, which can otherwise be implemented using general purpose I/Os on the chip.

The block diagram of the core is on the following page.


WISHBONE
Signals
WISHBONE
bus
interface
Divisor
Latch
Registers
Baud
Generator
Logic
Line
Status
Register
Line
Control
Register
Receiver
Logic
Receiver
FIFO
Receiver
Shift
Register
SRX_I
FIFO
Control
Register
Transmitter
Logic
Transmitter
FIFO
Transmitter
Shift
Register
STX_O
Interrupt
ID
Register
Interrupt
Enable
Register
Interrupt
Logic
INT_O
Modem
Status
Register
Modem
Control
Register
Modem
Signals
Logic
RTS_O
CTS_I
DTR_O
DSR_I
DCD_I
RI_I

CHAPTER

SEVEN

DESIGN VERIFICAITON

Following files are making an UART16550 PHY and are used for testing:

uart_device_if_defines.v	defines related to PHY
uart_device_if_memory.v	Module for initializing PHY (reading commands from vapi.log file)
uart_device_if.v	Uart PHY with additional feature for testing
vapi.log	File with commands (expected data, data to be send, etc.)

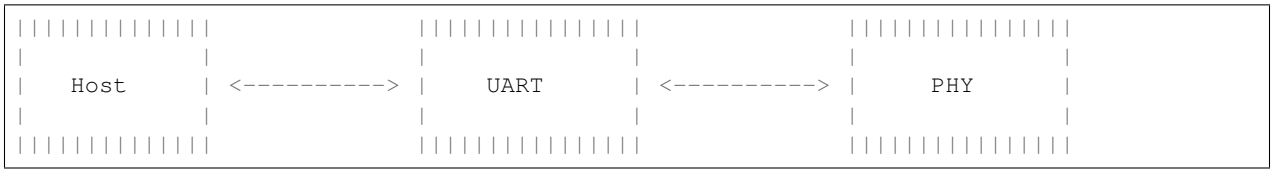
OPERATION:

uart_device_if.v is a uart PHY and connects to the uart_top.v. PHY takes commands from vapi.log file. Depending on command it can:

- set a mode (5, 6, 7, 8-bit, parity, stop bits, etc.)
- set a frequency divider (dll)
- send a character
- receive a character and compare it to the expected one
- send a glitch (after a certain period of time)
- send a break
- detect a break
- Check if fifo is empty/not empty (and generate an error if expected value differs from actual)
- delay (does nothing for certain number of characters)

On the other side of uart some kind of host must be connected that controls the phy.

This is the structure:



PHY must know how host sets th UART and work in the same mode. Besides that it must know what host is sending or expecting to receive. Operation of the PHY must be written in the vapi.log file.

When I was using this testing environment, I used OpenRISC1200 as a host. Everything is fully operational. UART was also tested in hardware (on two different boards), running uCLinux in both, interrupt and polling mode.

CHAPTER

EIGHT

HISTORY

8.1 Authors & Contributors

- Jacob Gorban
- Igor Mohor
- Tadej Markovic
- Olof Kindgren

8.2 Changes

Note: This section comes from a Changes file that was noted that it was being maintained since 25.5.2001. There are several later updates that can be found in the repository history, so this section is out of date.

8.2.1 29.07.2002

Reverted to have uart_defines.v file to be included in the verilog files. It seems that it's been a bad idea in the first place.

8.2.2 22.07.2002

Notice that this file hasn't been updated for a while so not all changed are present.

Bug Fixes:

- Possible loss of sync and bad reception of stop bit on slow baud rates fixed. Problem reported by Kenny.Tung.
- Bad (or lack of) loopback handling fixed. Reported by Cherry Withers.

Improvements:

- Made FIFO's as general inferrable memory where possible. So on FPGA they should be inferred as RAM (Distributed RAM on Xilinx). This saves about 1/3 of the Slice count and reduces P&R and synthesis times.
- Added optional baudrate output (baud_o). This is identical to BAUDOUT* signal on 16550 chip. It outputs 16xbit_clock_rate - the divided clock. It's disabled by default. Define UART_HAS_BAUDRATE_OUTPUT to use.

Note: The uart_defines.v file is no longer included in the source files. So keep this in mind when doing simulation. Add it manually. I've done this, so that you could you your own define files for different configurations. I need this for the IrDA core I develop. You can just uncomment the `includes if you want the old behaviour. The uart_fifo.v file is no longer used. Intead uart_rfifo.v and uart_tfifo.v file are now present. Also raminfr.v in the new inferred ram module.

Check the new core and I hope you'll like it.

8.2.3 10.08.2001

- Modified naming of top signals and defines to be unique and easy to integrate
- Changed the directory structure of the core to new structure as described in OpenCores coding guidelines. !!!
- Fixed (I hope) the detection of break condition
- Added top level parameters for data width and address line width

8.2.4 23.06.2001

- With the help of Bob Kirstein another two bugs were fixed: 1. Trasmitter was sending stop bit two 16xclock cycle slonger than needed. 2. Receiver was losing 1 16xclock cycle on each character and went out of sync.
- Major change: I have modified the divisor latch register to be 16-bit long instead of 32 as I thought was necessary for higher speed systems. Thanks to Rick Wright for pointing this out. So now, DL3 and DL4 register bytes are not used. Documentation is updated to follow this change.
- Note that more than 1 stop bit in a byte i snot implemented.

8.2.5 02.05.2001

- Fixed transmitter and receiver - the start and the stop bits were sent and received complemented. Big thanks go to Bob Kirstein for pointing this out to me.

8.2.6 31.05.2001

- Minor changes in register reading code
- Changed FCR to be 2 bits wide (reset bits are not needed) and instead enabled the rx_reset and tx_reset signals which I forgot to implement.
- Changed defines for FCR.
- Cleaned ports that were not connected in top-level.
- Changed the code to have only one FIFO module instead of two to overcome versioning problem on the cost of some additional gate count. UART_RX_FIFO was modified a little and renamed to UART_FIFO.
- UART_RX_FIFO.v and UART_TX_FIFO.v files removed from the project.
- Changes to receiver and transmitter modules concerning FIFO handling.
- Commented out `include "UART_defines" in all files but UART_top.v and test bench.
- Modified test bench a little for a little better check.

8.2.7 29.05.2001

- Fixed: Line Control Register block didn't have wb_rst_i in its sensitivity list
- Fixed: Modem Status Register block didn't have wb_rst_i in its sensitivity list and didn't set reset value
- Fixed rf_pop, lsr_mask, msi_reset and threi_clear not being synthesizable in release 1.7. (Thanks to Pavel Korenski for pointing this to me)

8.2.8 27.05.2001

Thanks to Rick Wright for pointing me many of my bugs.

- Fixed the rf_pop and lsr_mask flags not being deasserted.
- Fixed Time-Out interrupt not being masked by bit 0 in IER
- Fixed interrupt logic not being masked by IER
- Fixed bit 0 (interrupt pending) of IIR being set incorrectly
- Fixed Modem Status Register bits 3:0 handling (didn't work as should have)
- Fixed modem status interrupt to be related to bits [3:0] (deltas) instead of the bits 7:4 of MSR. This way the interrupt is cleared upon reading from the MSR.
- Fixed THRE interrupt not being reset by reading IIR
- Changed Receiver and Transmitter FIFO, so that they do not use the FIFO_inc.v file because of problems with #include command.
- Removed FIFO_inc.v from CVS tree.
- Updated specifications .pdf file

8.3 Legacy Bugs From OpenCores

1. [X] wishbone SEL_I problem
2. [ ] Three bugs
3. [X] Typo in documentation
4. [X] VHDL Implementation Request
5. [X] student
6. [X] Need to fix commenting-out style in rtl/verilog/uart_defines.v
7. [X] Need to fix commenting-out style in rtl/verilog/uart_defines.v
8. [X] about rs
9. [ ] Does uart_int.v testcase run successfully?
10. [ ] TERI in Modem Status Register Not To Specification
11. [ ] Problem with 16550 UART core
12. [X] Fatal bug in uart_receiver.v: srx_pad_i is not synchronized at all
13. [ ] suggested receiver core fixes
14. [X] Xilinx iSim generates "out of valid range" error

8.3.1 1. wishbone SEL_I problem - DONE - *No Issue*

- opened almost 16 years by ocghost
  - c.noble commented almost 10 years ago

    The VHDL 16550 UART generates an error using Xilinx iSim (I'm using 12.3 M.70d). Ther error is: ERROR: Value -2147483648 is out of valid range : 0 TO 8

    This is caused by line 29 of gh_uart_Rx_8bit.vhd, which reads: num_bits : in integer; I think that this should read num_bits : in integer:=8;

    Charlie

  - c.noble commented almost 10 years ago

    damn damn damn

    Sorry - posted this to the wrong core. Please can you remove it from here?

    My bad.

Since this bug was created in error this is complete.

8.3.2 2. Three bugs - Unknown - *Under Investigation*

- opened almost 16 years by ocghost
  - ocghost commented almost 16 years ago

    Codes at uart_regs.v

    1. Changes tf_push to combinational logic
    2. Changes `msr[`UART_MS_CDCD:`UART_MS_CCTS]` value from `{dcd_c, ri_c, dsr_c, cts_c}` to `{~dcd_c, ~ri_c, ~dsr_c, ~cts_c}`
    3. Change modem outputs value from:

```
assign rts_n = mcr[`UART_MC_RTS];
assign dtr_n = mcr[`UART_MC_DTR];
```

    To:

```
assign rts_n = ~mcr[`UART_MC_RTS];
assign dtr_n = ~mcr[`UART_MC_DTR];
```

  - ocghost commented almost 15 years ago

    The tf_push thing is quite a problem. This is because the fifo latches in the data the clock cycle after the write enable. If your write data remains valid for one more clock cycle then there is no problem, if the write data changes after the write enable (As in my case) you will get garbage written to the transmit fifo. As chenxj suggested changing the tf_push to combinational so it is valid one clock cycle earlier will fix the problem.

  - vchan commented over 13 years ago

    I agree with items 2 & 3. Rev. 0.6 of the specification call out the modem inputs and outputs to be inverted. I also tested my 16550 DUART in my ASCI and they are also inverted.

  - ocghost commented about 13 years ago

    I too have confirmed in hardware that 2 & 3 are definite bugs.

It's not completely clear yet but it appears the architecture was changed to store/use the modem control bits differently. @TODO Do these still need changed or is this handled?

tf_push is still a registered signal so it would still be delayed by a cycle. It's not completely clear yet whether this is fixed or still an issue. Others seemed to agree with the modem bits being incorrect but not with this signal clocking issue. @TODO Make sure the data is clocked in at the correct time.

8.3.3 3. Typo in documentation - DONE - *Added clarification*

- opened over 15 years by ocghost
  - ocghost commented over 15 years ago

    file: UART_spec.pdf On Page 7 clocks table: Rates max: 1258 MHz for 1200bps ? seems not to be correct
  - rfajardo commented about 12 years ago

    I believe it wants to say that if you have a frequency higher than 1258MHz you won't be able to get a baud rate of 1200 bps.

Added a blurb to this document below the mentioned table to explain the situation.

8.3.4 4. VHDL Implementation - DONE - *No Issue*

- opened over 15 years by ocghost
  - ocghost commented over 15 years ago

  Dear all,

  Does anyone have the VHDL Implementation of UART16550?

  Thanks and Regards, Jagan J

It's not realistic to reimplement this core as VHDL, so this is more of a question to point to a different core. Since this bug isn't relevent considering this complete. @TODO I'm no VHDL master, but maybe it might be worth investigating a 'wrapper' so this core could be instantiated from VHDL?

8.3.5 5. student - DONE - *Fixed github issue #4*

- opened about 15 years by paul_cooke_98
  - paul_cooke_98 commented about 15 years ago

    MCR register is not readable from address `UART_REG_MC.

    The following fix:

```
  `UART_REG_MS  : wb_dat_o = msr;
  `UART_REG_SR  : wb_dat_o = scratch;
  default:  wb_dat_o = 8'b0; // ??
endcase // case(wb_addr_i)
```

    changed to:

```
  `UART_REG_MS  : wb_dat_o = msr;
  `UART_REG_SR  : wb_dat_o = scratch;
  `UART_REG_MC    : wb_dat_o = {4'b000, mcr };
```

(continues on next page)

(continued from previous page)

```
  default:  wb_dat_o = 8'b0; // ??
endcase // case(wb_addr_i)
```

- ocghost commented about 15 years ago

  and update the sensitivity list:

```
always @(dl or dlab or ier or iir or scratch or lcr or lsr or msr or rf_data_
↪out or wb_addr_i or wb_re_i )
```

  changed to:

```
always @(dl or dlab or ier or iir or scratch or lcr or lsr or msr or rf_data_
↪out or wb_addr_i or wb_re_i or mcr)
```

- prashantpd commented over 2 years ago

  Type your text here

By inspection the read of the MCR appears to still be broken. The fix provided here, in essence, should fix this issue. @TODO this needs done.

8.3.6 6. Need to fix commenting-out style in rtl/verilog/uart_defines.v - DONE - *Fixed github issue #3*

- opened over 14 years by ocghost
  - ocghost commented over 14 years ago

    Hi. There are numerous instances in rtl/verilog/uart_defines.v of using single line comments ("//") in a `define statement. For me (at least), the Modelsim simulator barfs with this.

    For example, if you define `` `TRUE 1 // Hi mom` `` in a header file and then later on include that header file and try to assign something the macro TRUE: `` `foo <= `TRUE;` `` with a simplistic substitution, you would get `` `foo <= 1 // Hi mom;` `` which is a syntax error (missing semicolon).

    This is why you should define `` `TRUE 1 /* Hi mom using no single line comments! */` ``

    Some C compilers have the same issue. This is why you should not use single line comments on a `define or #define macro assignment!

    Here is my proposed patch to rtl/verilog/uart_defines.v. This is the output of diff -c3 (your version) (my patched version).

    Regards, Jeff:

```
*** dist/uart_defines.v Fri Sep 12 00:26:58 2003
--- uart_defines.v Mon May 15 10:53:55 2006
***************
*** 152,234 ****
  // `define UART_HAS_BAUDRATE_OUTPUT

! // Register addresses
! `define UART_REG_RBUART_ADDR_WIDTH'd0 // receiver buffer
! `define UART_REG_TRUART_ADDR_WIDTH'd0 // transmitter
! `define UART_REG_IEUART_ADDR_WIDTH'd1 // Interrupt enable
! `define UART_REG_IIUART_ADDR_WIDTH'd2 // Interrupt identification
```

(continues on next page)

(continued from previous page)

```
! `define UART_REG_FCUART_ADDR_WIDTH'd2 // FIFO control
! `define UART_REG_LCUART_ADDR_WIDTH'd3 // Line Control
! `define UART_REG_MCUART_ADDR_WIDTH'd4 // Modem control
! `define UART_REG_LSUART_ADDR_WIDTH'd5 // Line status
! `define UART_REG_MSUART_ADDR_WIDTH'd6 // Modem status
! `define UART_REG_SRUART_ADDR_WIDTH'd7 // Scratch register
! `define UART_REG_DL1UART_ADDR_WIDTH'd0 // Divisor latch bytes (1-2)
  `define UART_REG_DL2UART_ADDR_WIDTH'd1

! // Interrupt Enable register bits
! `define UART_IE_RDA 0 // Received Data available interrupt
! `define UART_IE_THRE 1 // Transmitter Holding Register empty interrupt
! `define UART_IE_RLS 2 // Receiver Line Status Interrupt
! `define UART_IE_MS 3 // Modem Status Interrupt
!
! // Interrupt Identification register bits
! `define UART_II_IP 0 // Interrupt pending when 0
! `define UART_II_II 3:1 // Interrupt identification
!
! // Interrupt identification values for bits 3:1
! `define UART_II_RLS 3'b011 // Receiver Line Status
! `define UART_II_RDA 3'b010 // Receiver Data available
! `define UART_II_TI 3'b110 // Timeout Indication
! `define UART_II_THRE 3'b001 // Transmitter Holding Register empty
! `define UART_II_MS 3'b000 // Modem Status
!
! // FIFO Control Register bits
! `define UART_FC_TL 1:0 // Trigger level
!
! // FIFO trigger level values
! `define UART_FC_1 2'b00
! `define UART_FC_4 2'b01
  `define UART_FC_8 2'b10
  `define UART_FC_14 2'b11

! // Line Control register bits
! `define UART_LC_BITS 1:0 // bits in character
! `define UART_LC_SB 2 // stop bits
! `define UART_LC_PE 3 // parity enable
! `define UART_LC_EP 4 // even parity
! `define UART_LC_SP 5 // stick parity
! `define UART_LC_BC 6 // Break control
! `define UART_LC_DL 7 // Divisor Latch access bit

! // Modem Control register bits
  `define UART_MC_DTR 0
  `define UART_MC_RTS 1
  `define UART_MC_OUT1 2
  `define UART_MC_OUT2 3
! `define UART_MC_LB 4 // Loopback mode

! // Line Status Register bits
! `define UART_LS_DR 0 // Data ready
! `define UART_LS_OE 1 // Overrun Error
! `define UART_LS_PE 2 // Parity Error
! `define UART_LS_FE 3 // Framing Error
! `define UART_LS_BI 4 // Break interrupt
```

(continues on next page)

(continued from previous page)

```
! `define UART_LS_TFE 5 // Transmit FIFO is empty
! `define UART_LS_TE 6 // Transmitter Empty indicator
! `define UART_LS_EI 7 // Error indicator

! // Modem Status Register bits
! `define UART_MS_DCTS 0 // Delta signals
  `define UART_MS_DDSR 1
  `define UART_MS_TERI 2
  `define UART_MS_DDCD 3
! `define UART_MS_CCTS 4 // Complement signals
  `define UART_MS_CDSR 5
  `define UART_MS_CRI 6
  `define UART_MS_CDCD 7

! // FIFO parameter defines

  `define UART_FIFO_WIDTH 8
  `define UART_FIFO_DEPTH 16
--- 152,234 ----
// `define UART_HAS_BAUDRATE_OUTPUT

! /* Register addresses */
! `define UART_REG_RBUART_ADDR_WIDTH'd0 /* receiver buffer */
! `define UART_REG_TRUART_ADDR_WIDTH'd0 /* transmitter */
! `define UART_REG_IEUART_ADDR_WIDTH'd1 /* Interrupt enable */
! `define UART_REG_IIUART_ADDR_WIDTH'd2 /* Interrupt identification */
! `define UART_REG_FCUART_ADDR_WIDTH'd2 /* FIFO control */
! `define UART_REG_LCUART_ADDR_WIDTH'd3 /* Line Control */
! `define UART_REG_MCUART_ADDR_WIDTH'd4 /* Modem control */
! `define UART_REG_LSUART_ADDR_WIDTH'd5 /* Line status */
! `define UART_REG_MSUART_ADDR_WIDTH'd6 /* Modem status */
! `define UART_REG_SRUART_ADDR_WIDTH'd7 /* Scratch register */
! `define UART_REG_DL1UART_ADDR_WIDTH'd0 /* Divisor latch bytes (1-2) */
  `define UART_REG_DL2UART_ADDR_WIDTH'd1

! /* Interrupt Enable register bits */
! `define UART_IE_RDA 0 /* Received Data available interrupt */
! `define UART_IE_THRE 1 /* Transmitter Holding Register empty interrupt */
! `define UART_IE_RLS 2 /* Receiver Line Status Interrupt */
! `define UART_IE_MS 3 /* Modem Status Interrupt */
!
! /* Interrupt Identification register bits */
! `define UART_II_IP 0 /* Interrupt pending when 0 */
! `define UART_II_II 3:1 /* Interrupt identification */
!
! /* Interrupt identification values for bits 3:1 */
! `define UART_II_RLS 3'b011 /* Receiver Line Status */
! `define UART_II_RDA 3'b010 /* Receiver Data available */
! `define UART_II_TI 3'b110 /* Timeout Indication */
! `define UART_II_THRE 3'b001 /* Transmitter Holding Register empty */
! `define UART_II_MS 3'b000 /* Modem Status */
!
! /* FIFO Control Register bits */
! `define UART_FC_TL 1:0 /* Trigger level */
!
! /* FIFO trigger level values */
! `define UART_FC_1 2'b00 /*  */
```

(continues on next page)

(continued from previous page)

```
! `define UART_FC_4 2'b01 /*  */
  `define UART_FC_8 2'b10
  `define UART_FC_14 2'b11

! /* Line Control register bits */
! `define UART_LC_BITS 1:0 /* bits in character */
! `define UART_LC_SB 2 /* stop bits */
! `define UART_LC_PE 3 /* parity enable */
! `define UART_LC_EP 4 /* even parity */
! `define UART_LC_SP 5 /* stick parity */
! `define UART_LC_BC 6 /* Break control */
! `define UART_LC_DL 7 /* Divisor Latch access bit */

! /* Modem Control register bits */
  `define UART_MC_DTR 0
  `define UART_MC_RTS 1
  `define UART_MC_OUT1 2
  `define UART_MC_OUT2 3
! `define UART_MC_LB 4 /* Loopback mode */

! /* Line Status Register bits */
! `define UART_LS_DR 0 /* Data ready */
! `define UART_LS_OE 1 /* Overrun Error */
! `define UART_LS_PE 2 /* Parity Error */
! `define UART_LS_FE 3 /* Framing Error */
! `define UART_LS_BI 4 /* Break interrupt */
! `define UART_LS_TFE 5 /* Transmit FIFO is empty */
! `define UART_LS_TE 6 /* Transmitter Empty indicator */
! `define UART_LS_EI 7 /* Error indicator */

! /* Modem Status Register bits */
! `define UART_MS_DCTS 0 /* Delta signals */
  `define UART_MS_DDSR 1
  `define UART_MS_TERI 2
  `define UART_MS_DDCD 3
! `define UART_MS_CCTS 4 /* Complement signals */
  `define UART_MS_CDSR 5
  `define UART_MS_CRI 6
  `define UART_MS_CDCD 7

! /* FIFO parameter defines */

  define UART_FIFO_WIDTH 8
  define UART_FIFO_DEPTH 16
*** 238,246 ****
  `define UART_FIFO_REC_WIDTH 11

! `define VERBOSE_WB 0 // All activity on the WISHBONE is recorded
! `define VERBOSE_LINE_STATUS 0 // Details about the lsr (line status␣
↪register)
! `define FAST_TEST 1 // 64/1024 packets are sent
--- 238,246 ----
`define UART_FIFO_REC_WIDTH 11

! `define VERBOSE_WB 0 /* All activity on the WISHBONE is recorded */
! `define VERBOSE_LINE_STATUS 0 /* Details about the lsr (line status␣
↪register) */
```

(continues on next page)

(continued from previous page)

```
! `define FAST_TEST 1 /* 64/1024 packets are sent */
```

This seems a simple enough request, but was never implemented. @TODO Look into a coding standard and implement that standard, which would make sense to include this.

8.3.7 7. Need to fix commenting-out style in rtl/verilog/uart_defines.v - DONE - *Duplicate*

- opened over 14 years by ocghost
  - ocghost commented over 14 years ago

    Hi. There are numerous instances in rtl/verilog/uart_defines.v of using single line comments ("//") in a `define statement. For me (at least), the Modelsim simulator barfs with this.

    For example, if you define `` `TRUE 1 // Hi mom` `` in a header file and then later on include that header file and try to assign something the macro TRUE: `` `foo <= `TRUE; `` with a simplistic substitution, you would get `` `foo <= 1 // Hi mom; `` which is a syntax error (missing semicolon).

    This is why you should define `` `TRUE 1 /* Hi mom using no single line comments! */` ``

    Some C compilers have the same issue. This is why you should not use single line comments on a `define or #define macro assignment!

    Here is my proposed patch to rtl/verilog/uart_defines.v. This is the output of diff -c3 (your version) (my patched version).

    Regards, Jeff:

```
*** dist/uart_defines.v Fri Sep 12 00:26:58 2003
--- uart_defines.v Mon May 15 10:53:55 2006
***************
*** 152,234 ****
  // `define UART_HAS_BAUDRATE_OUTPUT

! // Register addresses
! `define UART_REG_RBUART_ADDR_WIDTH'd0 // receiver buffer
! `define UART_REG_TRUART_ADDR_WIDTH'd0 // transmitter
! `define UART_REG_IEUART_ADDR_WIDTH'd1 // Interrupt enable
! `define UART_REG_IIUART_ADDR_WIDTH'd2 // Interrupt identification
! `define UART_REG_FCUART_ADDR_WIDTH'd2 // FIFO control
! `define UART_REG_LCUART_ADDR_WIDTH'd3 // Line Control
! `define UART_REG_MCUART_ADDR_WIDTH'd4 // Modem control
! `define UART_REG_LSUART_ADDR_WIDTH'd5 // Line status
! `define UART_REG_MSUART_ADDR_WIDTH'd6 // Modem status
! `define UART_REG_SRUART_ADDR_WIDTH'd7 // Scratch register
! `define UART_REG_DL1UART_ADDR_WIDTH'd0 // Divisor latch bytes (1-2)
  `define UART_REG_DL2UART_ADDR_WIDTH'd1

! // Interrupt Enable register bits
! `define UART_IE_RDA 0 // Received Data available interrupt
! `define UART_IE_THRE 1 // Transmitter Holding Register empty interrupt
! `define UART_IE_RLS 2 // Receiver Line Status Interrupt
! `define UART_IE_MS 3 // Modem Status Interrupt
!
! // Interrupt Identification register bits
```

(continues on next page)

(continued from previous page)

```
! `define UART_II_IP 0 // Interrupt pending when 0
! `define UART_II_II 3:1 // Interrupt identification
!
! // Interrupt identification values for bits 3:1
! `define UART_II_RLS 3'b011 // Receiver Line Status
! `define UART_II_RDA 3'b010 // Receiver Data available
! `define UART_II_TI 3'b110 // Timeout Indication
! `define UART_II_THRE 3'b001 // Transmitter Holding Register empty
! `define UART_II_MS 3'b000 // Modem Status
!
! // FIFO Control Register bits
! `define UART_FC_TL 1:0 // Trigger level
!
! // FIFO trigger level values
! `define UART_FC_1 2'b00
! `define UART_FC_4 2'b01
  `define UART_FC_8 2'b10
  `define UART_FC_14 2'b11

! // Line Control register bits
! `define UART_LC_BITS 1:0 // bits in character
! `define UART_LC_SB 2 // stop bits
! `define UART_LC_PE 3 // parity enable
! `define UART_LC_EP 4 // even parity
! `define UART_LC_SP 5 // stick parity
! `define UART_LC_BC 6 // Break control
! `define UART_LC_DL 7 // Divisor Latch access bit

! // Modem Control register bits
  `define UART_MC_DTR 0
  `define UART_MC_RTS 1
  `define UART_MC_OUT1 2
  `define UART_MC_OUT2 3
! `define UART_MC_LB 4 // Loopback mode

! // Line Status Register bits
! `define UART_LS_DR 0 // Data ready
! `define UART_LS_OE 1 // Overrun Error
! `define UART_LS_PE 2 // Parity Error
! `define UART_LS_FE 3 // Framing Error
! `define UART_LS_BI 4 // Break interrupt
! `define UART_LS_TFE 5 // Transmit FIFO is empty
! `define UART_LS_TE 6 // Transmitter Empty indicator
! `define UART_LS_EI 7 // Error indicator

! // Modem Status Register bits
! `define UART_MS_DCTS 0 // Delta signals
  `define UART_MS_DDSR 1
  `define UART_MS_TERI 2
  `define UART_MS_DDCD 3
! `define UART_MS_CCTS 4 // Complement signals
  `define UART_MS_CDSR 5
  `define UART_MS_CRI 6
  `define UART_MS_CDCD 7

! // FIFO parameter defines
```

(continues on next page)

(continued from previous page)

```
  `define UART_FIFO_WIDTH 8
  `define UART_FIFO_DEPTH 16
--- 152,234 ----
// `define UART_HAS_BAUDRATE_OUTPUT

! /* Register addresses */
! `define UART_REG_RBUART_ADDR_WIDTH'd0 /* receiver buffer */
! `define UART_REG_TRUART_ADDR_WIDTH'd0 /* transmitter */
! `define UART_REG_IEUART_ADDR_WIDTH'd1 /* Interrupt enable */
! `define UART_REG_IIUART_ADDR_WIDTH'd2 /* Interrupt identification */
! `define UART_REG_FCUART_ADDR_WIDTH'd2 /* FIFO control */
! `define UART_REG_LCUART_ADDR_WIDTH'd3 /* Line Control */
! `define UART_REG_MCUART_ADDR_WIDTH'd4 /* Modem control */
! `define UART_REG_LSUART_ADDR_WIDTH'd5 /* Line status */
! `define UART_REG_MSUART_ADDR_WIDTH'd6 /* Modem status */
! `define UART_REG_SRUART_ADDR_WIDTH'd7 /* Scratch register */
! `define UART_REG_DL1UART_ADDR_WIDTH'd0 /* Divisor latch bytes (1-2) */
  `define UART_REG_DL2UART_ADDR_WIDTH'd1

! /* Interrupt Enable register bits */
! `define UART_IE_RDA 0 /* Received Data available interrupt */
! `define UART_IE_THRE 1 /* Transmitter Holding Register empty interrupt */
! `define UART_IE_RLS 2 /* Receiver Line Status Interrupt */
! `define UART_IE_MS 3 /* Modem Status Interrupt */
!
! /* Interrupt Identification register bits */
! `define UART_II_IP 0 /* Interrupt pending when 0 */
! `define UART_II_II 3:1 /* Interrupt identification */
!
! /* Interrupt identification values for bits 3:1 */
! `define UART_II_RLS 3'b011 /* Receiver Line Status */
! `define UART_II_RDA 3'b010 /* Receiver Data available */
! `define UART_II_TI 3'b110 /* Timeout Indication */
! `define UART_II_THRE 3'b001 /* Transmitter Holding Register empty */
! `define UART_II_MS 3'b000 /* Modem Status */
!
! /* FIFO Control Register bits */
! `define UART_FC_TL 1:0 /* Trigger level */
!
! /* FIFO trigger level values */
! `define UART_FC_1 2'b00 /*  */
! `define UART_FC_4 2'b01 /*  */
  `define UART_FC_8 2'b10
  `define UART_FC_14 2'b11

! /* Line Control register bits */
! `define UART_LC_BITS 1:0 /* bits in character */
! `define UART_LC_SB 2 /* stop bits */
! `define UART_LC_PE 3 /* parity enable */
! `define UART_LC_EP 4 /* even parity */
! `define UART_LC_SP 5 /* stick parity */
! `define UART_LC_BC 6 /* Break control */
! `define UART_LC_DL 7 /* Divisor Latch access bit */

! /* Modem Control register bits */
  `define UART_MC_DTR 0
  `define UART_MC_RTS 1
```

(continues on next page)

(continued from previous page)

```
  `define UART_MC_OUT1 2
  `define UART_MC_OUT2 3
! `define UART_MC_LB 4 /* Loopback mode */

! /* Line Status Register bits */
! `define UART_LS_DR 0 /* Data ready */
! `define UART_LS_OE 1 /* Overrun Error */
! `define UART_LS_PE 2 /* Parity Error */
! `define UART_LS_FE 3 /* Framing Error */
! `define UART_LS_BI 4 /* Break interrupt */
! `define UART_LS_TFE 5 /* Transmit FIFO is empty */
! `define UART_LS_TE 6 /* Transmitter Empty indicator */
! `define UART_LS_EI 7 /* Error indicator */

! /* Modem Status Register bits */
! `define UART_MS_DCTS 0 /* Delta signals */
  `define UART_MS_DDSR 1
  `define UART_MS_TERI 2
  `define UART_MS_DDCD 3
! `define UART_MS_CCTS 4 /* Complement signals */
  `define UART_MS_CDSR 5
  `define UART_MS_CRI 6
  `define UART_MS_CDCD 7

! /* FIFO parameter defines */

  define UART_FIFO_WIDTH 8
  define UART_FIFO_DEPTH 16
*** 238,246 ****
  `define UART_FIFO_REC_WIDTH 11

! `define VERBOSE_WB 0 // All activity on the WISHBONE is recorded
! `define VERBOSE_LINE_STATUS 0 // Details about the lsr (line status␣
↪register)
! `define FAST_TEST 1 // 64/1024 packets are sent
--- 238,246 ----
`define UART_FIFO_REC_WIDTH 11

! `define VERBOSE_WB 0 /* All activity on the WISHBONE is recorded */
! `define VERBOSE_LINE_STATUS 0 /* Details about the lsr (line status␣
↪register) */
! `define FAST_TEST 1 /* 64/1024 packets are sent */
```

– ocghost commented over 14 years ago

  Sorry. Hit refresh on browser and it re-submitted my request. Pls delete this duplicate request.

– ocghost commented over 13 years ago

  hai , This is very use for me. I need full code which ar e designed in Verilog.

  regard Thiru

– ocghost commented over 13 years ago

  hai , This is very use for me. I need full code which ar e designed in Verilog.

  regard Thiru

This is a duplicate of #6 so this can be ignored.

8.3.8 8. about rs - DONE - *No Issue*

- opened over 14 years by ocghost
  - ocghost commented over 14 years ago

    I want to know about rs code,would like you to give me some vhdl code abuout rs? Thank you !!

Doesn't even seem to be a coherent thought about this project, so this can be ignored.

8.3.9 9. Does uart_int.v testcase run successfully? - Unknown - *Under Investigation*

- opened almost 14 years by ocghost
  - ocghost commented almost 14 years ago

    I tried to run the supplied testcase, "uart_int.v", but it fails in ModelSim. The message in uart_interrupts_verbose.log is shown below:

    Time: 5734521200 (testbench.tx_fifo_status_changing) *E, Bit 5 of LSR register not '1'!

    I just wanted to verify that someone has successfully run the testcase before I spend any more time troubleshooting it.

    Regards, Dalton
  - robg commented about 13 years ago

    I encountered an equal problem, which was related to the wb_sel_i decoding. The testcase (i.e. the wb_master) is always setting wb_sel_i to 4'hF which leads to masking off the lower two bits of the wb_addr (see uart_wb.v lines 293 and 301), because the case statements hit the default case.

    My solution was to change the lines 293 and 301 to:

    default: wb_adr_int_lsb = wb_adr_is1:0;

    If you dislike this 'hack', simply make the wb_master use the correct wb_sel_i values.

    best regards Robert
  - robg commented about 13 years ago

    Well, the again. That simply got the simulation up and running, but it didn't fix the bug you reported. Sorry.

I'm getting this same sort of error after getting the simulation mostly functional. @TODO this should be investigated a little further as the design verification is enhanced.

8.3.10 10. TERI in Modem Status Register Not To Specification - Unknown - *Under Investigation*

- opened over 13 years by vchan
  - vchan commented over 13 years ago

    According to the Rev. 0.6 Specification, that bit is supposed to be set only when going from low to high state. However, the code is treating it same as the other three Delta indicator bits. The specification is correct. Need to change the code such that TERI bit is set only when delayed_modem_signal2 and ~ri.

This sounds valid and I'm not seeing a changeset associated with a potential fix so this could be a real issue that needs fixed? @TODO this should be investigated further.

8.3.11 11. Problem with 16550 UART core - Unknown - *Under Investigation*

- opened about 12 years by valentina.lomi
  - valentina.lomi commented about 12 years ago

    Hallo, I'm not sure I found a bug but I used the core and I found a problem. I'm using 9600 baudrate with fifo trigger level set to 1 and no dma. Transmission goes perfectly but reception has the following strange behaviour: if I send to the UART the letter A, it receives another character (0). If, after letter A I send the letter B, it still receives 0. Only after 15 transmissions of characters, the UART actually receives letter A, and following another transmission of a character, it receives the letter B and so on. It seems that there is a delay of 15 characters that I can't remove. I've tried to reset the FIFO during initialization of the UART and tried to flush it but without any result. Can anyone help me?

    Thank you Valentina

  - gbaron commented about 12 years ago

    I've come across similar issues. I think it's to do with how the FIFO's are implemented in the design. It doesn't seem to port very well to FPGA architectures.

    jwiseman commented almost 12 years ago I have the exact same problem. Implemented on a Altera Cyclone II FPGA. I also have a problem with the transmit FIFO; It transmits each character it enter 14 times. Any ideas on how to get this code to port correctly?

    Jason

  - nigeldean commented almost 12 years ago

    Jason - I am using this core on an Altera Cyclone EP1C6Q240 and had the same problems, but making the modifications described by Oskar Lopez (search in forum) solved them. Change the receive FIFO for one from the GH_VHDL library (gh_fifo_async_usrf.vhd) and modify it as described. The only difference I found was where he describes adding the signal:

    signal iRD : STD_LOGIC:='0';

    I had a problem with compiling this with initialisation, so I have:

    signal iRD : STD_LOGIC;

    Which worked ok. Hope this helps.

    Nigel

  - jwiseman commented almost 12 years ago

    Thanks Nigel! This fixed the recieve FIFO problem. However the transmit FIFO still has problems. I tried to use the library function gh_fifo_async_uswf.vhd with the same changes as the usrf version, and now the data loaded is transmitted (good), but 13x in a row starting with the last loaded value!

    Jason

  - nigeldean commented almost 12 years ago

    Sorry Jason - I forgot that I also had other problems using the core, with Tx, but these were due to interfacing issues with my processor bus and control lines (CS), and not particularly the core itself.

    I'm using a Freescale Coldfire processor with 16 bit bus & found I needed an 8 bit latch on the RD port of the 16550 and also some other logic to control CS - to ensure that only a single CLK edge occurs while CS is valid - this is important otherwise you will get multiple writes. I'm using 4.096MHz for the bitrate clock which is derived from a 32.768MHz system clock (32.768/8) and 15.5MHz for CLK input to 16550, which is derived from the 60MHz processor clock (60MHz/4). When I first used the core & had problems (with Tx) I did some simulations and found that for some reason (as yet unexplained) the core did not work properly with a common bus going to both Tx & Rx ports - this is why I used an 8 bit latch on the

Rx port which seems to work fine. It was only after I fixed the Tx problems that I realised the Rx also was not working properly - then I tried the fix described by Oscar Lopez & this sorted the Rx.

I will send you (off line) the 16550 part of my circuit which may help.

– charrier commented almost 12 years ago

Hi Nigel,

I'm very interessed in the modification you've done on this circuit too. I'm not formed with verilog and I couldn't do the modifications as you describe.

I've got some problems too with this IP implemented on EP2C8 device. I just use this IP as RS232 protocol (use only Tx,Rx,CTS,RTS) First, I tried to communicate only with Tx/Rx and some bytes are lost during transmission. I think my problem comes from FIFO managing. I think the following post upped on the reported bugs by el_tacano76@h... on 21-Dec-2005 is the problem.

This is because the fifo latches in the data the clock cycle after the write enable. If your write data remains valid for one more clock cycle then there is no problem, if the write data changes after the write enable (As in my case) you will get garbage written to the transmit fifo. As chenxj suggested changing the tf_push to combinational so it is valid one clock cycle earlier will fix the problem.

The autor of the IP say also in the description of the uart_regs.v that there is a known problem and that it could be fixed with inserting a wait state on all wishbone transfer... I'm trying to do this but without success for the moment.

Have you got an idea ? Thanks.

– renko commented about 3 years ago

Solutions for Valentina's problem?

These notes seem confusing since they reference VHDL, so maybe this issue doesn't actaully belong to this core? @TODO this should be investigated further.

8.3.12 12. Fatal bug in uart_receiver.v: srx_pad_i is not synchronized at all - DONE - *Fixed May 21, 2004*

- opened over 11 years by ehliar
  – ehliar commented over 11 years ago

  srx_pad_i is not synchronized to the clock domain of the UART in uart_receiver.v.

  Without synchronization I'm not able to use the UART for anything but text entry. With synchronization change I can easily download a 6 megabyte text file over the UART without any problems.

It seems this was acknowledged as a bug and was fixed but the issue was never closed.|

8.3.13 13. suggested receiver core fixes - @TODO - *Partial, correct/needed?*

- opened over 10 years by eteam
  – eteam commented over 10 years ago

  All of these apply to module uart_receiver.v

  I agree completely with ehliar, input signal srx_pad_i absolutely must be registered with the UART clock (clk) to synchronise to the clock domain. Without the sync register (or two), the stability of the state machine in uart_receiver.v is gravely compromised. Suggest two stages of register, and only the final delay stage should be used anywhere else in the design.

reg rbit_in is unused, should be deleted.

in state machine code, there is an ambiguity in the coding. Note the sr_rec_prepare state section, starting at line 313, where the counter rcounter16 is initialised to 4'b1110 (line 323). Outside the IF-ELSE statement is another assignment for the same counter: rcounter16 <= #1 rcounter16_minus_1; (line 328). This second assignment should be placed inside the ELSE statment.

Original code snippet sr_rec_prepare:

```
begin
  case (lcr/`UART_LC_BITS/1:0) // number of bits in a word
    2'b00 : rbit_counter <= #1 3'b100;
    2'b01 : rbit_counter <= #1 3'b101;
    2'b10 : rbit_counter <= #1 3'b110;
    2'b11 : rbit_counter <= #1 3'b111;
  endcase
  if (rcounter16_eq_0) begin
    rstate <= #1 sr_rec_bit;
    rcounter16 <= #1 4'b1110;
    rshift <= #1 0;
  end
  else
    rstate <= #1 sr_rec_prepare;
    rcounter16 <= #1 rcounter16_minus_1;
end
```

Revised code snippet sr_rec_prepare:

```
begin
  case (lcr/`UART_LC_BITS/1:0) // number of bits in a word
    2'b00 : rbit_counter <= #1 3'b100;
    2'b01 : rbit_counter <= #1 3'b101;
    2'b10 : rbit_counter <= #1 3'b110;
    2'b11 : rbit_counter <= #1 3'b111;
  endcase
  if (rcounter16_eq_0) begin
    rstate <= #1 sr_rec_bit;
    rcounter16 <= #1 4'b1110;
    rshift <= #1 0;
  end
  else begin // added
    rstate <= #1 sr_rec_prepare;
    rcounter16 <= #1 rcounter16_minus_1; // moved inside the IF-ELSE lines
  end // added end
```

The part of this that is referencing the need for receiver syncronizers is fixed, but the second part of this with regard to the counter has not changed. @TODO I'm not sure if this matters or is actually required so further investigation is necessary.

8.3.14 14. Xilinx iSim generates “out of valid range” error - DONE - *No Issue*

- opened almost 10 years by c.noble
  - c.noble commented almost 10 years ago

    The VHDL 16550 UART generates an error using Xilinx iSim (I’m using 12.3 M.70d). Ther error is: ERROR: Value -2147483648 is out of valid range : 0 TO 8

    This is caused by line 29 of gh_uart_Rx_8bit.vhd, which reads: num_bits : in integer; I think that this should read num_bits : in integer:=8;

    Charlie

  - c.noble commented almost 10 years ago

    damn damn damn

    Sorry - posted this to the wrong core. Please can you remove it from here?

    My bad.

This referenced the wrong core, so this issue can be ignored.