BUS_UART

November 21, 2024

Jay Convertino

Contents

1 Usage

1.1 Introduction

BUS UART is a core for interfacing over RS232 UART to a bus of choice. The core will process data to and from the UART. The data can then be accessed over a BUS, currently AXI lite or Wishbone Classic, and processed as needed. All input and output over the bus goes into FIFOs that is then tied to the AXIS UART core. The following is information on how to use the device in an FPGA, software, and in simulation.

1.2 Dependencies

The following are the dependencies of the cores.

- fusesoc 2.X
- iverilog (simulation)
- cocotb (simulation)

1.2.1 axi_lite_uart Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up_uart:1.0.0
  - AD:common:up_axi:1.0.0
- dep_tb
  - AFRL:simulation:axis_stimulator
  - AFRL:utility:sim_helper

1.2.2 wishbone_classic_uart Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up_uart:1.0.0
  - AFRL:bus:up_wishbone_classic:1.0.0

1.2.3 up_uart Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device_converter:axis_uart:1.0.0
  - AFRL:buffer:fifo

1.3 In a Project

First, pick a core that matches the target bus in question. Then connect the BUS UART core to that bus. Once this is complete the UART pins will need to be routed so they match the UART device or other.

2 Architecture

This core is made up of other cores that are documented in detail in there source. The cores this is made up of are the,

- **axis_uart** Interface with UART and present the data over AXIS interface (see core for documentation).
- **fifo** Used for RX and TX FIFO instances. Set to 16 words buffer max (see core for documentation).
- **up_axi** An AXI Lite to uP converter core (see core for documentation).
- **up_wishbone_classic** A wishbone classic to uP converter core (see core for documentation).
- **up_uart** Takes uP bus and coverts it to interface with the RX/TX FIFOs and the AXIS UART (see module documentation for information 5).

For register documentation please see up_uart in 5

3 Building

The BUS UART is written in Verilog 2001. It should synthesize in any modern FPGA software. The core comes as a fusesoc packaged core and can be included in any other core. Be sure to make sure you have meet the dependencies listed in the previous section.

3.1 fusesoc

Fusesoc is a system for building FPGA software without relying on the internal project management of the tool. Avoiding vendor lock in to Vivado or Quartus. These cores, when included in a project, can be easily integrated and targets created based upon the end developer needs. The core by itself is not a part of a system and should be integrated into a fusesoc based system. Simulations are setup to use fusesoc and are a part of its targets.

3.2 Source Files

3.2.1 axi_lite_uart File List

- src
  - src/axi_lite_uart.v
- tb
  - tb/tb_uart.v

3.2.2 wishbone_classic_uart File List

- src
  - src/wishbone_classic_uart.v
- tb
  - tb/tb_wishbone_slave.v

3.2.3 up_uart File List

- src
  - src/up_uart.v
- tb
  - tb/tb_up_uart.v

3.3 Targets

3.3.1 axi_lite_uart Targets

- default

  Info: Default for IP intergration.

- sim

  Info: Base simulation using icarus as default.

- sim_rand_data

  Info: Use random data as sim input.

- sim_rand_ready_rand_data

  Info: Use random data with a random ready as sim input.

- sim_8bit_count_data

  Info: Use counter data as sim input.

- sim_rand_ready_8bit_count_data

  Info: Use counter data with a random ready as sim input.

3.3.2 wishbone_classic_uart Targets

- default

  Info: Default for IP intergration.

- sim

  Info: Base simulation using icarus as default.

3.3.3 up_uart Targets

- default

  Info: Default for IP intergration.

- sim

  Info: Base simulation using icarus as default.

3.4 Directory Guide

Below highlights important folders from the root of the directory.

1. **docs** Contains all documentation related to this project.
   - **manual** Contains user manual and github page that are generated from the latex sources.
2. **src** Contains source files for the core
3. **tb** Contains test bench files for iverilog and cocotb
   - **cocotb** testbench files

4 Simulation

There are a few different simulations that can be run for this core.

4.1 iverilog

iverilog is used for simple test benches for quick verification, visually, of the core.

4.2 cocotb

Future simulations will use cocotb. This feature is not yet implemented.

5 Module Documentation

up_uart is the module that integrates the AXIS UART core. This includes FIFO's that have there inputs/outputs for data tied to registers mapped in the uP bus. The uP bus is the microprocessor bus based on Analog Devices design. It resembles a APB bus in design, and is the bridge to other buses BUS UART can use. This makes changing for AXI Lite, to Wishbone to whatever quick and painless.

axi_lite_uart module adds a AXI Lite to uP (microprocessor) bus converter. The converter is from Analog Devices.

wishbone_classic_uart module adds a Wishbone Classic to uP (microprocessor) bus converter. This converter was designed for Wishbone Classic only, NOT pipelined.

The next sections document these modules in great detail. up_uart contains the register map explained, and what the various bits do.

axi_lite_uart.v

AUTHORS

JAY CONVERTINO

DATES

2024/02/29

INFORMATION

Brief

AXI Lite UART is a core for interfacing with UART devices.

License MIT

axi_lite_uart

```
module axi_lite_uart #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
CLOCK_SPEED
 =
100000000,
parameter
BAUD_RATE
```

```
 =
115200,
parameter
PARITY_ENA
 =
0,
parameter
PARITY_TYPE
 =
0,
parameter
STOP_BITS
 =
1,
parameter
DATA_BITS
 =
8,
parameter
RX_DELAY
 =
0,
parameter
RX_BAUD_DELAY
 =
0,
parameter
TX_DELAY
 =
0,
parameter
TX_BAUD_DELAY
 =
0
) ( input aclk, input arstn, input s_axi_aclk, input s_axi_aresetn, input s_
```

AXI Lite based uart device.

Parameters

ADDRESS_WIDTH
parameter
Width of the axi address bus

CLOCK_SPEED
parameter
This is the aclk frequency in Hz

BAUD_RATE
parameter
Serial Baud, this can be any value including non-standard.

PARITY_ENA
parameter
Enable Parity for the data in and out.

PARITY_TYPE
parameter
Set the parity type, 0 = even, 1 = odd, 2 = mark, 3 = space.

STOP_BITS
parameter
Number of stop bits, 0 to crazy non-standard amounts.

DATA_BITS
parameter
Number of data bits, 1 to crazy non-standard amounts.

RX_DELAY
parameter
Delay in rx data input.

RX_BAUD_DELAY
parameter
Delay in rx baud enable. This will delay when we sample a bit (default is midpoint when rx delay is 0).

TX_DELAY
Delay in tx data output. Delays the time to output of the data.

parameter

TX_BAUD_DELAY Delay in tx baud enable. This will delay the time the bit output starts.
parameter

Ports

aclk	Clock for all devices in the core
arstn	Negative reset
s_axi_awvalid	Axi Lite aw valid
s_axi_awaddr	Axi Lite aw addr
s_axi_awprot	Axi Lite aw prot
s_axi_awready	Axi Lite aw ready
s_axi_wvalid	Axi Lite w valid
s_axi_wdata	Axi Lite w data
s_axi_wstrb	Axi Lite w strb
s_axi_wready	Axi Lite w ready
s_axi_bvalid	Axi Lite b valid
s_axi_bresp	Axi Lite b resp
s_axi_bready	Axi Lite b ready
s_axi_arvalid	Axi Lite ar valid
s_axi_araddr	Axi Lite ar addr
s_axi_arprot	Axi Lite ar prot
s_axi_arready	Axi Lite ar ready
s_axi_rvalid	Axi Lite r valid
s_axi_rdata	Axi Lite r data
s_axi_rresp	Axi Lite r resp
s_axi_rready	Axi Lite r ready

irq	Interrupt when data is received
tx	transmit for UART (output to RX)
rx	receive for UART (input from TX)
rts	request to send is a loop with CTS
cts	clear to send is a loop with RTS

up_rreq

```
wire up_rreq
```

uP read bus request

up_rack

```
wire up_rack
```

uP read bus acknowledge

up_raddr

```
wire [ADDRESS_WIDTH-3:0] up_raddr
```

uP read bus address

up_rdata

```
wire [31:0] up_rdata
```

uP read bus request

up_wreq

```
wire up_wreq
```

uP write bus request

up_wack

```
wire up_wack
```

uP write bus acknowledge

up_waddr

```
wire [ADDRESS_WIDTH-3:0] up_waddr
```

uP write bus address

up_wdata

```
wire [31:0] up_wdata
```

uP write bus data

INSTANTIANTED MODULES

inst_up_axi

```
up_axi #(
                                                                    .
AXI_ADDRESS_WIDTH(ADDRESS_WIDTH)
) inst_up_axi ( .up_rstn (arstn), .up_clk (aclk), .up_axi_awvalid(s_axi_aw
```

Module instance of up_axi for the AXI Lite bus to the uP bus.

inst_up_uart

```
up_uart #(
                                                                    .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                    .
CLOCK_SPEED(CLOCK_SPEED),
                                                                    .
BAUD_RATE(BAUD_RATE),
                                                                    .
PARITY_ENA(PARITY_ENA),
                                                                    .
PARITY_TYPE(PARITY_TYPE),
                                                                    .
STOP_BITS(STOP_BITS),
                                                                    .
DATA_BITS(DATA_BITS),
                                                                    .
RX_DELAY(RX_DELAY),
                                                                    .
RX_BAUD_DELAY(RX_BAUD_DELAY),
                                                                    .
TX_DELAY(TX_DELAY),
                                                                    .
TX_BAUD_DELAY(TX_BAUD_DELAY)
) inst_up_uart ( .clk(aclk), .rstn(arstn), .up_rreq(up_rreq), .up_rack(up_ra
```

Module instance of up_uart creating a Logic wrapper for uart axis bus cores to interface with uP bus.

wishbone_classic_uart.v

AUTHORS

JAY CONVERTINO

DATES

2024/02/29

INFORMATION

Brief

AXI Lite 1553 is a core for interfacing with 1553 devices over the AXI lite bus.

License MIT

wishbone_classic_uart

```
module wishbone_classic_uart #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
CLOCK_SPEED
```

```
 =
100000000,
parameter
BAUD_RATE
 =
115200,
parameter
PARITY_ENA
 =
0,
parameter
PARITY_TYPE
 =
0,
parameter
STOP_BITS
 =
1,
parameter
DATA_BITS
 =
8,
parameter
RX_DELAY
 =
0,
parameter
RX_BAUD_DELAY
 =
0,
parameter
TX_DELAY
 =
0,
parameter
TX_BAUD_DELAY
 =
0
) ( input clk, input rst, input s_wb_cyc, input s_wb_stb, input s_wb_we, inp
```

AXI Lite based uart device.

Parameters

ADDRESS_WIDTH Width of the address bus in bits.
parameter

BUS_WIDTH Width of the data bus in bytes.
parameter

CLOCK_SPEED This is the aclk frequency in Hz
parameter

BAUD_RATE Serial Baud, this can be any value including non-standard.
parameter

PARITY_ENA Enable Parity for the data in and out.
parameter

PARITY_TYPE Set the parity type, 0 = even, 1 = odd, 2 = mark, 3 = space.
parameter

STOP_BITS Number of stop bits, 0 to crazy non-standard amounts.
parameter

DATA_BITS Number of data bits, 1 to crazy non-standard amounts.
parameter

RX_DELAY parameter	Delay in rx data input.
RX_BAUD_DELAY parameter	Delay in rx baud enable. This will delay when we sample a bit (default is midpoint when rx delay is 0).
TX_DELAY parameter	Delay in tx data output. Delays the time to output of the data.
TX_BAUD_DELAY parameter	Delay in tx baud enable. This will delay the time the bit output starts.

Ports

clk	Clock for all devices in the core
rst	Positive reset
s_wb_cyc	Bus Cycle in process
s_wb_stb	Valid data transfer cycle
s_wb_we	Active High write, low read
s_wb_addr	Bus address
s_wb_data_i	Input data
s_wb_sel	Device Select
s_wb_bte	Burst Type Extension
s_wb_cti	Cycle Type
s_wb_ack	Bus transaction terminated
s_wb_data_o	Output data
s_wb_err	Active high when a bus error is present
irq	Interrupt when data is received
tx	transmit for UART (output to RX)
rx	receive for UART (input from TX)
rts	request to send is a loop with CTS
cts	clear to send is a loop with RTS

up_rreq

```
wire up_rreq
```

uP read bus request

up_rack

```
wire up_rack
```

uP read bus acknowledge

up_raddr

```
wire [ADDRESS_WIDTH-3:0] up_raddr
```

uP read bus address

up_rdata

```
wire [31:0] up_rdata
```

uP read bus request

up_wreq

```
wire up_wreq
```

uP write bus request

up_wack

```
wire up_wack
```

uP write bus acknowledge

up_waddr

```
wire [ADDRESS_WIDTH-3:0] up_waddr
```

uP write bus address

up_wdata

```
wire [31:0] up_wdata
```

uP write bus data

INSTANTIANTED MODULES

inst_up_wishbone_classic

```
up_wishbone_classic #(
                                                                    .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                    .
BUS_WIDTH(BUS_WIDTH)
) inst_up_wishbone_classic ( .clk(clk), .rst(rst), .s_wb_cyc(s_wb_cyc), .s_w
```

Module instance of up_wishbone_classic for the Wishbone Classic bus to the uP bus.

inst_up_uart

```
up_uart #(
                                                                    .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                    .
BUS_WIDTH(BUS_WIDTH),
                                                                    .
CLOCK_SPEED(CLOCK_SPEED),
                                                                    .
BAUD_RATE(BAUD_RATE),
                                                                    .
PARITY_ENA(PARITY_ENA),
                                                                    .
PARITY_TYPE(PARITY_TYPE),
                                                                    .
STOP_BITS(STOP_BITS),
                                                                    .
DATA_BITS(DATA_BITS),
                                                                    .
RX_DELAY(RX_DELAY),
                                                                    .
RX_BAUD_DELAY(RX_BAUD_DELAY),
                                                                    .
TX_DELAY(TX_DELAY),
                                                                    .
TX_BAUD_DELAY(TX_BAUD_DELAY)
) inst_up_uart ( .clk(clk), .rstn(~rst), .up_rreq(up_rreq), .up_rack(up_rack
```

Module instance of up_uart creating a Logic wrapper for uart axis bus cores to interface with uP bus.

up_uart.v

AUTHORS

JAY CONVERTINO

DATES

2024/02/29

INFORMATION

Brief

uP Core for interfacing with axis uart.

License MIT

up_1553

uP based 1553 communications device.

Parameters

ADDRESS_WIDTH	Width of the uP address port.
BUS_WIDTH	Width of the uP bus data port.
CLOCK_SPEED	This is the aclk frequency in Hz
BAUD_RATE	Serial Baud, this can be any value including non-standard.
PARITY_ENA	Enable Parity for the data in and out.

PARITY_TYPE	Set the parity type, 0 = even, 1 = odd, 2 = mark, 3 = space.
STOP_BITS	Number of stop bits, 0 to crazy non-standard amounts.
DATA_BITS	Number of data bits, 1 to crazy non-standard amounts.
RX_DELAY	Delay in rx data input.
RX_BAUD_DELAY	Delay in rx baud enable. This will delay when we sample a bit (default is midpoint when rx delay is 0).
TX_DELAY	Delay in tx data output. Delays the time to output of the data.
TX_BAUD_DELAY	Delay in tx baud enable. This will delay the time the bit output starts.

Ports

clk	Clock for all devices in the core
rstn	Negative reset
up_rreq	uP bus read request
up_rack	uP bus read ack
up_raddr	uP bus read address
up_rdata	uP bus read data
up_wreq	uP bus write request
up_wack	uP bus write ack
up_waddr	uP bus write address
up_wdata	uP bus write data
irq	Interrupt when data is received
tx	transmit for UART (output to RX)
rx	receive for UART (input from TX)
rts	request to send is a loop with CTS
cts	clear to send is a loop with RTS

FIFO_DEPTH

```
localparam FIFO_DEPTH = 16
```

Depth of the fifo, matches UART LITE (xilinx), so I kept this just cause

REGISTER INFORMATION

Core has 4 registers at the offsets that follow.

RX_FIFO_REG	h0
TX_FIFO_REG	h4
STATUS_REG	h8
CONTROL_REG	hC

RX_FIFO_REG

```
localparam RX_FIFO_REG = 4'h0
```

Defines the address offset for RX FIFO

RX FIFO REGISTER	
31:8	7:0
UNUSED	RECEIVED DATA

Valid bits are from DATA_BITS:0, which are data.

TX_FIFO_REG

```
localparam TX_FIFO_REG = 4'h4
```

Defines the address offset to write the TX FIFO.

TX FIFO REGISTER	
31:8	7:0
UNUSED	TRANSMIT DATA

Valid bits are from DATA_BITS:0, which are data.

STATUS_REG

```
localparam STATUS_REG = 4'h8
```

Defines the address offset to read the status bits.

STATUS REGISTER								
31:8	7	6	5	4	3	2	1	0
UNUSED	PE	FE	OE	irq_en	tx_full	tx_empty	rx_full	rx_valid

Status Register Bits

PE 7, Parity error, active high on error

FE 6, Frame error, active high on error

OE 5, Overrun error, active high on error

irq_en 4, 1 when the IRQ is enabled by CONTROL_REG

tx_full 3, When 1 the tx fifo is full.

tx_empty 2, When 1 the tx fifo is empty.

rx_full 1, When 1 the rx fifo is full.

rx_valid 0, When 1 the rx fifo contains valid data.

CONTROL_REG

```
localparam CONTROL_REG = 4'hC
```

Defines the address offset to set the control bits.

CONTROL REGISTER				
31:5	4	3:2	1	0
UNUSED	ENA_INTR_BIT	UNUSED	RST_RX_BIT	RST_TX_BIT

See Also: ENABLE_INTR_BIT, RESET_RX_BIT, RESET_TX_BIT

Control Register Bits

ENABLE_INTR_BIT 4, Control Register offset bit for enabling the interrupt.

RESET_RX_BIT 1, Control Register offset bit for resetting the RX FIFO.

RESET_TX_BIT 0, Control Register offset bit for resetting the TX FIFO.

INSTANTIATED MODULES

inst_axis_uart

```
axis_uart #(
                                                                    .
BAUD_CLOCK_SPEED(CLOCK_SPEED),
                                                                    .
BAUD_RATE(BAUD_RATE),
                                                                    .
PARITY_ENA(PARITY_ENA),
                                                                    .
PARITY_TYPE(PARITY_TYPE),
                                                                    .
STOP_BITS(STOP_BITS),
                                                                    .
DATA_BITS(DATA_BITS),
                                                                    .
RX_DELAY(RX_DELAY),
                                                                    .
RX_BAUD_DELAY(RX_BAUD_DELAY),
                                                                    .
TX_DELAY(TX_DELAY),
                                                                    .
TX_BAUD_DELAY(TX_BAUD_DELAY)
) inst_axis_uart ( .aclk(clk), .arstn(rstn), .parity_err(s_parity_err), .fra
```

UART instance with AXIS interface for TX/RX

inst_rx_fifo

```
fifo #(
                                                                                .
FIFO_DEPTH(FIFO_DEPTH),
                                                                                .
BYTE_WIDTH(BUS_WIDTH),
                                                                                .
COUNT_WIDTH(8),
                                                                                .
FWFT(1),
                                                                                .
RD_SYNC_DEPTH(0),
                                                                                .
WR_SYNC_DEPTH(0),
                                                                                .
DC_SYNC_DEPTH(0),
                                                                                .
COUNT_DELAY(0),
                                                                                .
COUNT_ENA(0),
                                                                                .
DATA_ZERO(0),
                                                                                .
ACK_ENA(0),
                                                                                .
RAM_TYPE("block")
) inst_rx_fifo ( .rd_clk(clk), .rd_rstn(rstn & r_rstn_rx_delay[0]), .rd_en(s
```

Buffer up to 16 items output from the axis_1553_encoder.

inst_tx_fifo

```
fifo #(
                                                                                .
FIFO_DEPTH(FIFO_DEPTH),
                                                                                .
BYTE_WIDTH(BUS_WIDTH),
                                                                                .
COUNT_WIDTH(8),
                                                                                .
FWFT(1),
                                                                                .
RD_SYNC_DEPTH(0),
                                                                                .
WR_SYNC_DEPTH(0),
                                                                                .
DC_SYNC_DEPTH(0),
                                                                                .
COUNT_DELAY(0),
                                                                                .
COUNT_ENA(0),
                                                                                .
DATA_ZERO(0),
                                                                                .
ACK_ENA(0),
                                                                                .
RAM_TYPE("block")
) inst_tx_fifo ( .rd_clk(clk), .rd_rstn(rstn & r_rstn_tx_delay[0]), .rd_en(s
```

Buffer up to 16 items to input to the axis_1553_decoder.