# FAST_AXIS_UART


June 11, 2025

Jay Convertino

# Contents

# 1 Usage

## 1.1 Introduction

Simple FAST UART core for TTL rs232 software mode data communications. No hardware handshake. This contains its own internal baud rate generator that creates an enable to allow data output or sampling. Baud clock and aclk can be the same clock.

## 1.2 Dependencies

The following are the dependencies of the cores.

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

### 1.2.1 fusesoc_info Depenecies

- dep
  - AFRL:clock:mod_clock_ena_gen:1.1.1
  - AFRL:utility:helper:1.0.0
  - AFRL:simple:piso:1.0.0
  - AFRL:simple:sipo:1.0.0

## 1.3 In a Project

This core connects a UART to the AXIS bus. Meaning this is a streaming device only. Connect the RX/TX to the UART in question and connect the AXIS to its intended endpoints.

# 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,

- **fast_axis_uart** Interface with UART and present the data over AXIS interface (see core for documentation).
- **mod_clk_gen_ena** Generates enable pulses at the baud rate based on the input clock.
- **PISO** Take parallel input data and output in a serial fashion.
- **SIPO** Take serial data input and output parallel data.

# 3 Building

The FAST AXIS 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. Linting is performed by verible using the lint target.

## 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 fusesoc_info File List

- src
  - src/fast_axis_uart.v
- tb_cocotb_full
  - 'tb/tb_cocotb_full.py': 'file_type': 'user', 'copyto': '.'
  - 'tb/tb_cocotb_full.v': 'file_type': 'verilogSource'

## 3.3 Targets

### 3.3.1 fusesoc_info Targets

- default

  Info: Default for IP intergration.
- lint

  Info: Lint with Verible
- sim_cocotb_full

  Info: Cocotb unit tests

## 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 cocotb

To use the cocotb tests you must install the following python libraries.

```
$ pip install cocotb
$ pip install cocotbext-axi
```

Each module has a cocotb based simulation. These use the cocotb extensions made by Alex. The two extensions used are cocotbext-axi and cocotbext-uart. These provide outside verification of the implimentation. These tests consist of 3 different fusesoc targets.

- **sim_cocotb_full** Standard simulation of TX/RX passing data to and from cocotbexts.

Then you must use the cocotb sim target. The targets above can be run with various bus and fifo parameters.

```
$ fusesoc run --target AFRL:device_converter:fast_axis_uart:1.0.0
```

# 5 Module Documentation

- **fast_axis_uart** Wrapper for all of the modules to create a singular device to interface with.

# fast_axis_uart.v

## AUTHORS

### JAY CONVERTINO

## DATES

### 2025/06/11

## INFORMATION

### Brief

Fast UART AXIS core that allows for back to back transmissions.

### License MIT



### fast_axis_uart

```
module fast_axis_uart #(
parameter
CLOCK_SPEED
 =
2000000,
parameter
BAUD_RATE
 =
2000000,
parameter
PARITY_TYPE
 =
0,
parameter
```

```
STOP_BITS
 =
1,
parameter
DATA_BITS
 =
8,
parameter
RX_BAUD_DELAY
 =
0,
parameter
TX_BAUD_DELAY
 =
0
) ( input aclk, input arstn, output parity_err, output frame_err, input [ 7
```

AXIS UART, fast simple UART with AXI Streaming interface.

### Parameters

| | |
|---|---|
| **CLOCK_SPEED**<br>parameter | This is the aclk frequency in Hz |
| **BAUD_RATE**<br>parameter | Serial Baud, this can be any value including non-standard. |
| **PARITY_TYPE**<br>parameter | Set the parity type, 0 = none, 1 = odd, 2 = even, 3 = mark, 4 = space. |
| **STOP_BITS**<br>parameter | Number of stop bits, 0 to crazy non-standard amounts. |
| **DATA_BITS**<br>parameter | Number of data bits, 1 to 8. |
| **RX_BAUD_DELAY**<br>parameter | Delay in rx baud enable. This will delay when we sample a bit (default is midpoint when rx delay is 0). |
| **TX_BAUD_DELAY**<br>parameter | Delay in tx baud enable. This will delay the time the bit output starts. |

### Ports

| | |
|---|---|
| **aclk** | Clock for AXIS |
| **arstn** | Negative reset for AXIS |
| **parity_err** | Indicates error with parity check (active high) |
| **frame_err** | Indicates error with frame (active high) |
| **s_axis_tdata** | Input data for UART TX. |
| **s_axis_tvalid** | When set active high the input data is valid |
| **s_axis_tready** | When active high the device is ready for input data. |
| **m_axis_tdata** | Output data from UART RX |
| **m_axis_tvalid** | When active high the output data is valid |
| **m_axis_tready** | When set active high the output device is ready for data. |
| **tx** | transmit for UART (output to RX) |
| **rx** | receive for UART (input from TX) |

## INSTANTIATED MODULES

## uart_baud_gen_tx

```
mod_clock_ena_gen #(
                                                                                .
CLOCK_SPEED(CLOCK_SPEED),
                                                                                .
DELAY(TX_BAUD_DELAY)
) uart_baud_gen_tx ( .clk(aclk), .rstn(arstn), .start0(1'b1), .clr(1'b0), .h
```

Generates TX BAUD rate for UART modules using modulo divide method.

## uart_baud_gen_rx

```
mod_clock_ena_gen #(
                                                                                .
CLOCK_SPEED(CLOCK_SPEED),
                                                                                .
DELAY(RX_BAUD_DELAY)
) uart_baud_gen_rx ( .clk(aclk), .rstn(arstn), .start0(1'b0), .clr(r_rx_clr)
```

Generates RX BAUD rate for UART modules using modulo divide method.

## inst_sipo

```
sipo #(
                                                                                .
BUS_WIDTH(32),
                                                                                .
COUNT_AMOUNT(BITS_PER_TRANS)
) inst_sipo ( .clk(aclk), .rstn(arstn), .ena(uart_ena_rx), .rev(1'b1), .loa
```

Captures RX data for uart receive

## inst_piso

```
piso #(
                                                                                .
BUS_WIDTH(32),
                                                                                .
COUNT_AMOUNT(BITS_PER_TRANS),
                                                                                .
DEFAULT_RESET_VAL(1),
                                                                                .
DEFAULT_SHIFT_VAL(1)
) inst_piso ( .clk(aclk), .rstn(arstn), .ena(uart_ena_tx), .rev(1'b1), .loa
```

Generates TX data for uart transmit

# tb_cocotb.py

## AUTHORS

### JAY CONVERTINO

## DATES

### 2024/12/09

## INFORMATION

### Brief

Cocotb test bench

### License MIT



## FUNCTIONS

### random_bool

```
def random_bool()
```

Return a infinte cycle of random bools

Returns: List

### start_clock

```
def start_clock(
dut
)
```

Start the simulation clock generator.

### Parameters

**dut** Device under test passed from cocotb test function

## reset_dut

```
async def reset_dut(
dut
)
```

Cocotb coroutine for resets, used with await to make sure system is reset.

## single_word

```
@cocotb.test()
async def single_word(
dut
)
```

Coroutine that is identified as a test routine. This routine tests for writing a single word, and then reading a single word.

### Parameters

**dut** Device under test passed from cocotb.

## in_reset

```
@cocotb.test()
async def in_reset(
dut
)
```

Coroutine that is identified as a test routine. This routine tests if device stays in unready state when in reset.

### Parameters

**dut** Device under test passed from cocotb.

## no_clock

```
@cocotb.test()
async def no_clock(
dut
)
```

Coroutine that is identified as a test routine. This routine tests if no ready when clock is lost and device is

left in reset.

### Parameters

**dut** Device under test passed from cocotb.

# tb_cocotb.v

## AUTHORS

### JAY CONVERTINO

## DATES

### 2025/01/21

## INFORMATION

### Brief

Test bench wrapper for cocotb

### License MIT



### tb_cocotb

```
module tb_cocotb #(
parameter
CLOCK_SPEED
 =
2000000,
parameter
BAUD_RATE
 =
2000000,
parameter
PARITY_TYPE
 =
0,
parameter
```

```
STOP_BITS
 =
1,
parameter
DATA_BITS
 =
8,
parameter
RX_BAUD_DELAY
 =
0,
parameter
TX_BAUD_DELAY
 =
0
) ( input aclk, input arstn, output parity_err, output frame_err, input [ 7
```

Test bench for axis uart.

## Parameters

| | |
|---|---|
| **BAUD_CLOCK_SPEED** | This is the aclk frequency in Hz |
| **BAUD_RATE** parameter | Serial Baud, this can be any value including non-standard. |
| **PARITY_ENA** | 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** | 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. |
| **TX_BAUD_DELAY** parameter | Delay in tx baud enable. This will delay the time the bit output starts. |
| **BUS_WIDTH** | AXIS data bus width in bytes. |

## Ports

| | |
|---|---|
| **aclk** | Clock for AXIS |
| **arstn** | Negative reset for AXIS |
| **parity_err** | Indicates error with parity check (active high) |
| **frame_err** | Indicates error with frame (active high) |
| **s_axis_tdata** | Input data for UART TX. |
| **s_axis_tvalid** | When set active high the input data is valid |
| **s_axis_tready** | When active high the device is ready for input data. |
| **m_axis_tdata** | Output data from UART RX |
| **m_axis_tvalid** | When active high the output data is valid |
| **m_axis_tready** | When set active high the output device is ready for data. |
| **uart_clk** | Clock used for BAUD rate generation |
| **uart_rstn** | Negative reset for UART, for anything clocked on uart_clk |
| **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 |

## INSTANTIATED MODULES

### dut

```
fast_axis_uart #(
                                                                    .
CLOCK_SPEED(CLOCK_SPEED),
                                                                    .
BAUD_RATE(BAUD_RATE),
                                                                    .
PARITY_TYPE(PARITY_TYPE),
                                                                    .
STOP_BITS(STOP_BITS),
                                                                    .
DATA_BITS(DATA_BITS),
                                                                    .
RX_BAUD_DELAY(RX_BAUD_DELAY),
                                                                    .
TX_BAUD_DELAY(TX_BAUD_DELAY)
) dut ( .aclk(aclk), .arstn(arstn), .parity_err(parity_err), .frame_err(fra
```

Device under test, fast_axis_uart