UART Controller v1.1

IP User Guide

January 2024

Licensing Notice

UART Controller v1.1 is a fully tested, portable, configurable, and synthesisable soft IP core provided by **Chip**munk **Logic**™. The IP source files are complied with IEEE VHDL/Verilog/System-Verilog standards. All the source codes are open-source licensed and hence may be used, modified, and shared without any restrictions or conflicts of interest with the original developer.

This IP core is provided 'as is,' without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and non-infringement. **Chip**munk **Logic**™ shall not be liable or held accountable for any loss or damage (direct or indirect) resulting from the use of any product, as the designs are not intended to be fail-safe or for use in any application requiring fail-safe performance. Hence, the user shall assume sole risk and liability for the use of any of our products in any of their applications.

Table of Contents

1. UART Controller

UART Controller IP Core provides a simple asynchronous serial interface for data transmission and reception. UART signalling is implemented at the serial interface. The core provides a parallel data interface and control interface to control the data flow. It performs parallel-to-serial conversion of data received at parallel data interface, and serial-to-parallel conversion of data received at serial interface.

2. Features

- ✓ Full duplex communication, 8-bit data.
- ✓ Simple valid-ready handshaking at the data interface of UART transmitter and receiver for ease of integration with FIFOs.
- ✓ Configurable parity: Odd, Even, or no parity.
- ✓ Built-in Baud Generator with configurable baud rate.
- ✓ Built-in CDC synchronizer at the receiver line.

3. Overview

The UART Controller transmits and receives the Least Significant bit (LSb) first. The core's transmitter (TX) and receiver (RX) are functionally independent, but use the same data format and baud rate.

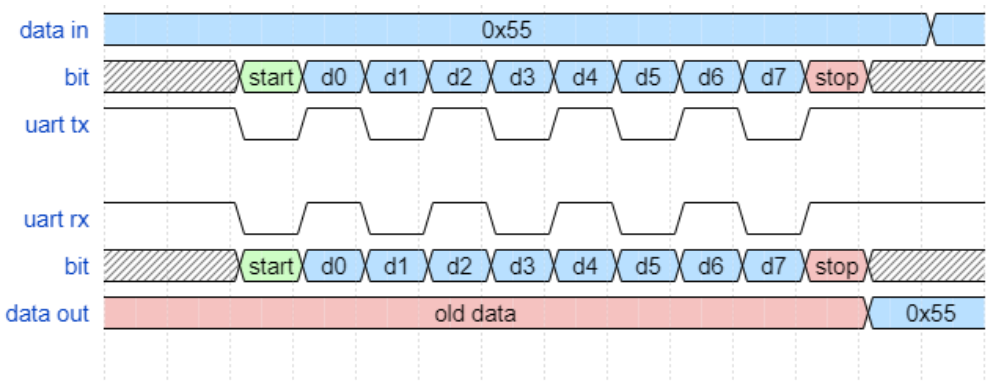

Figure 3.1: UART – Data Format

Figure 3.1 shows the functional block diagram of UART Controller. Various interfaces and sub-blocks are:

- **Control Interface**: Set of signals to control the operation of the IP, like baud rate, packet format etc.
- **Data Interface**: Parallel data and handshaking interface.
- **TX and RX**: Serial data interface for transmission and reception.
- **Status flags**: Status flags for errors in communication.
- **Clock and Reset**: Core clock and reset.
- **UART Transmitter**: Converts parallel data to serial data and sends via TX. Contains one TX buffer to hold the data to be transmitted.
- **UART Receiver**: Converts serial data received via RX to parallel data. Contains two RX buffers. One to shift and store the incoming data and second buffer holds the valid data to be read out. The Receiver samples the data at x8 rate and samples in the middle of the data (see Figure 3.3).
- **Baud Generator**: Generate baud clocks for TX and RX and controls the baud rate. RX uses 8x sample clock that of TX.

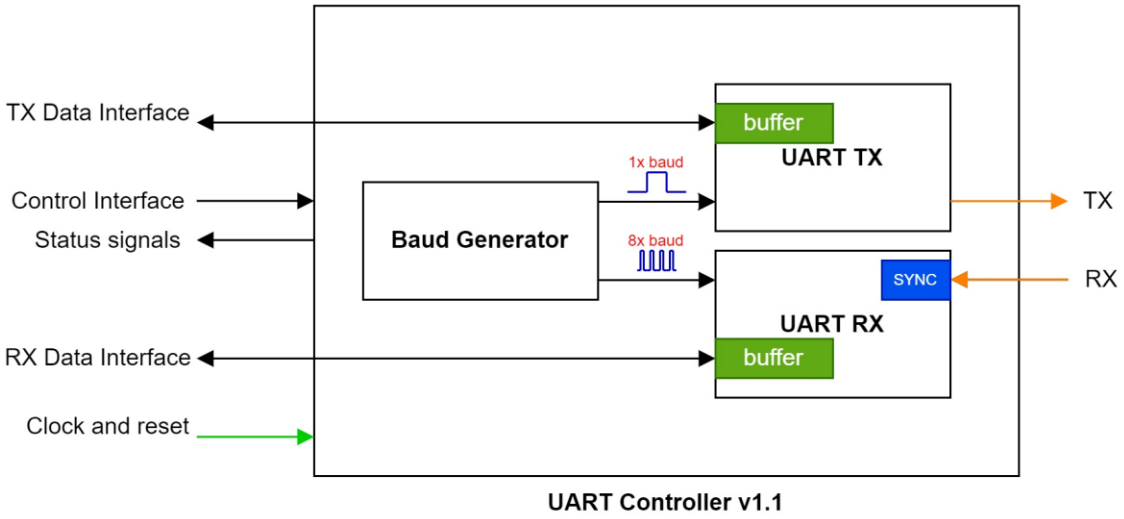

Figure 3.2: UART Controller – Block Diagram

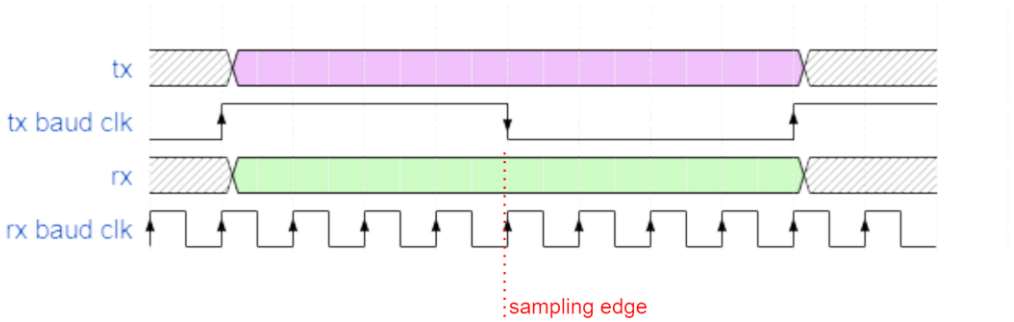

Figure 3.3: UART RX – Sampling at x8

4. Top-level Ports/Interfaces

Table 4.1 lists all top-level I/O ports/interfaces of the IP.

Signal Name	Direction	Width	Description
clk	input	1	Core clock
rstn	input	1	Core reset (synchronous active-low)
Control Interface			
i_baudrate	input	16	Baud rate configuration value, see here for details.
i_parity_mode	input	2	Parity mode configuration. 2'bx0 = No parity bit in serial data 2'b01 = Odd parity bit 2'b11 = Even parity bit
i_frame_mode	input	1	No. of start and stop bits in serial data. 1'b0 = 1 start bit and 1 stop bit 1'b1 = 1 start bit and 2 stop bits
i_tx_en	input	1	To enable the transmitter and its baud clock. Disabling will reset the transmitter and the buffer.
i_rx_en	input	1	To enable the receiver and its baud clock. Disabling will reset the receiver and the buffers.
TX – Serial/Parallel Data Interface			
i_data	input	8	Byte to be transmitted
i_data_valid	input	1	Byte valid
o_ready	output	1	Transmitter ready to accept the byte to be transmitted. 1'b0 = Not ready to accept the byte. Either the transmitter is disabled or previous transmission is in progress. 1'b1 = Ready to accept the byte, and no serial transmission is going on.
o_tx	output	1	Serial data out
RX – Serial/Parallel Data Interface			
i_rx	input	1	Serial data in
o_data	output	8	Byte which is received
o_data_valid	output	1	Byte valid 1'b0 = No byte has been received 1'b1 = Byte has been received and it is valid
i_ready	input	1	To read out the byte received at the receiver.

Status flags			
o_parity_err	output	1	Indicates that parity error has occurred for the last byte received. The received byte is still available to read out from the receiver. See here for details.
o_frame_err	output	1	Indicates that frame error (one or more stop bits not detected) has occurred for the last byte received. The received byte is ignored and not available to read out from the receiver. See here for details.

Table 4.1: Top-level Ports/Interfaces

5. Designing with the IP

This chapter discusses guidelines, clocking and reset, configuration, performance, and other known limitations while designing with the IP.

5.1 Clocking and Reset

The core clock `clk` synchronizes the complete operation of the core. Baud clocks for the UART transmitter and receiver are generated from this clock. Reset `rstn` is fully synchronous to `clk` and active-low.

5.2 Configuring the IP

Clock and Reset Sequencing

1. Bring up the core clock.
2. Assert the reset for at least 8 clock cycles.
3. Release the reset.
4. The core is now ready for configuration.

Configuring and Enabling TX

1. Configure parity mode, frame mode, baud rate.
2. Load the byte to be transmitted.
3. Enable TX. The core transmits the byte through serial TX.
4. Check status flags for errors.
5. Load the next byte when TX data interface is ready again.
6. TX can be disabled any time.

Configuring and Enabling RX

1. Configure parity mode, frame mode, baud rate.
2. Enable RX. The core is now ready to receive the byte through serial RX.
3. Read the byte out when RX data interface drives valid data.
4. Check status flags for errors.
5. RX can be disabled any time.

Baud Rate Configuration

The integer value B should be configured in `i_baudrate` to set the baud rate of serial data transfer at the UART transmitter and receiver, can be calculated as follows:

$$B = \mathrm{INT}((\frac{Core\ clock\ freq}{Baud\ rate\ required})/8 - 1)$$

Where INT(x) = Nearest integer to x.

For e.g., if the core clock is 100 MHz, and baud rate required is 9600, then $\boldsymbol{B}$ is configured as:

$$\frac{100 \times 10^6}{9600}/8 - 1 \approx \mathbf{1301}$$

The maximum supported baud rate for a given core clock is for $\boldsymbol{B}$ = 1, $\frac{Core\ clock\ freq}{16}$.

Therefore, the minimum value which can be configured in $\boldsymbol{B}$ = 1.

The receiver samples serial data at 8x. i.e., internally, 8x baud clock is required by the receiver. Baud Generator generates 1x baud clock for the transmitter and 8x baud clock for the receiver from the configured baud rate.

Baud rate error can be calculated as:

$$Target\ baud\ rate = 9600\ bps$$
$$Actual\ baud\ rate\ = \frac{100000000}{(1301+1)*8} = 9600.61\ bps$$
$$Baud\ rate\ error = \frac{(9600.61 - 9600)}{9600} = 0.006\%$$

Typically, in UART 8-bit data transfer, the maximum tolerable baud rate error is $\pm 5\%$. Since $\boldsymbol{B}$ can be configured only as integer, it introduces a rounding-off error in baud clock rate. It is imperative that this error is kept within the prescribed range for reliable data transfer. Use higher-frequency core clock to achieve more accurate baud clock generation, especially if higher baud rates are targeted.

Following tables list examples of baud rate configuration for core clock = 100 MHz and 10 MHz. The IP supports variety of wide range of baud rates. For custom baud rates, error % must be ensured within the accepted tolerance before configuring.

Core clock = 100 MHz			
Target Baud Rate (bps)	**Actual Baud Rate (bps)**	**% Error**	**B value (16-bit decimal)**
300	300.00	-0.001	41666
600	600.01	+0.002	20832
1200	1199.96	-0.003	10416
2400 (Min)	2400.04	+0.006	5207
4800	4800.08	+0.006	2603
9600	9600.61	+0.006	1301
19200	19201.23	+0.006	650
38400	38402.46	-0.147	325
57600	57603.69	+0.006	216
115200	115207.40	+-0.452	108

Table 5.1: Baud Rate configuration for core clock 100 MHz

Core clock = 10 MHz			
Target Baud Rate (bps)	**Actual Baud Rate (bps)**	**% Error**	**B value (16-bit decimal)**
300 (Min)	299.98	-0.008	4166
600	600.10	+0.016	2082
1200	1199.62	-0.032	1041
2400	2399.23	-0.032	520
4800	4807.69	+0.160	259
9600	9615.38	+0.160	129
19200	19230.77	+0.160	64
38400	37878.79	-1.357	32
57600	56818.18	-1.357	21
115200	113636.36	-1.357	10

Table 5.2: Baud Rate configuration for 10 MHz core clock

5.3 Data Transfer with the IP

The core has implemented a simple valid-ready handshaking at the parallel data interface. The byte to be sent/received has to be properly communicated with the core using handshake signals: *valid* and *ready*. Figure 5.1 shows how a typical handshaking is done with the core to transmit and receive two bytes.

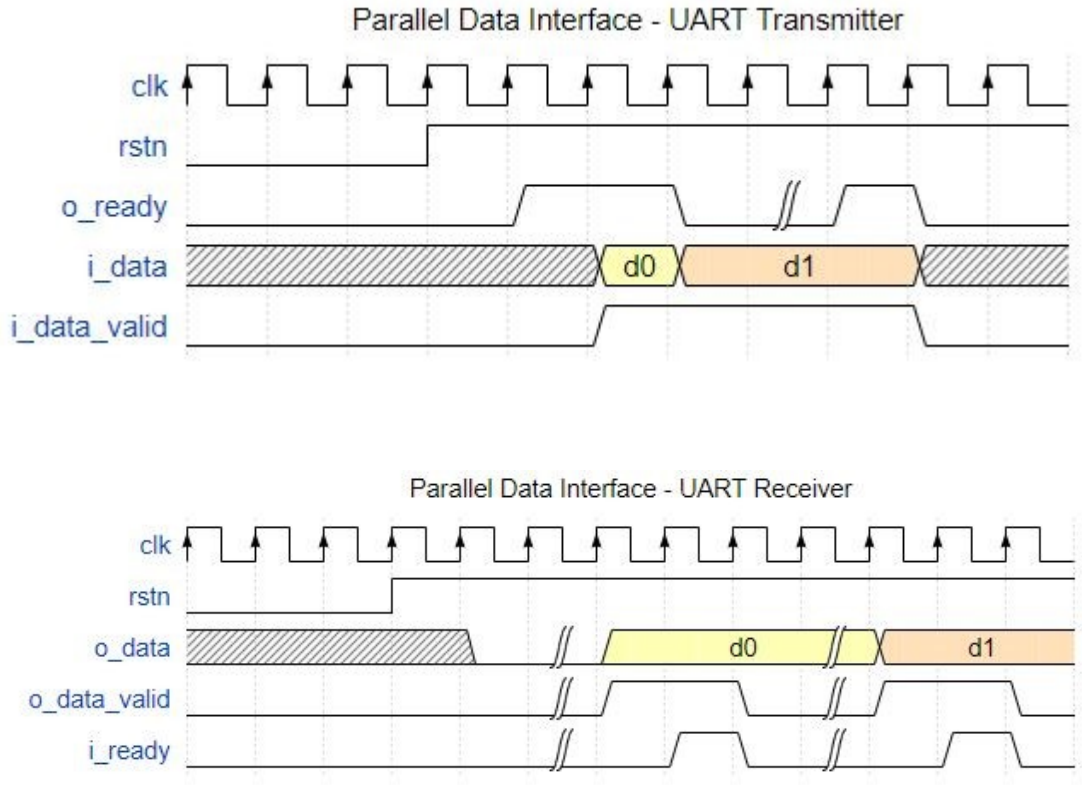

Figure 5.1: Valid-Ready Handshaking with IP

Writing a Byte to Core at TX Data Interface

User may assert `i_data_valid` when a byte has to be transmitted via `o_tx`. The byte is written on `i_data`. User needs not wait for `o_ready` to be asserted. The core asserts `o_ready` when it has finished the transmission of the previous byte, and is ready to accept a new byte. It need not wait for `i_data_valid` to be asserted. However, the byte is written to the core only when both `i_data_valid` and `o_ready` are high.

Reading a Byte from Core at RX Data Interface

The core asserts `o_data_valid` when a byte has been received via `i_rx`. The byte is available on `o_data`. It needs not wait for `i_ready` to be asserted. User may assert `i_ready` to read the byte. It's not mandatory that `i_ready` should be asserted only after `o_data_valid;` it may be asserted in advance. However, the byte is read from the core only when both `o_data_valid` and `i_ready` are high.

6. Integrating the IP

6.1 FIFO Integration and Interrupts

The IP has no FIFOs integrated at the transmitter or receiver. However, the valid-ready handshaking at the parallel data interface eases the integration of FIFOs at user's will. The *valid* and *ready* signals can be directly interfaced to a typical FIFO with minimal/no glue-logic.

The core doesn't provide interrupt control. The feature is left for the flexibility of the user who integrates the core. Interrupts can be derived from the FIFO status at TX and RX. If FIFOs are not implemented, *valid* and *ready* signals can be used to generate a level-sensitive interrupt and interrupt acknowledge. For e.g.: a crude interrupt implementation at TX data interface would look like:

1. Enable TX.
2. The core would assert *ready* to accept new byte.
3. Load the byte to be transmitted and assert *valid*. The core would de-assert *ready* and transmits the byte through serial TX.
4. The core would assert *ready* after the byte transmission is complete. // Interrupt set
5. Acknowledge the core by loading the next byte and asserting *valid*, or disabling TX if no more bytes to send. // Interrupt acknowledge/clear

6.2 Error Handling

The core reports two types of errors on reception of serial data via RX.

- Parity error
- Frame error

Byte parity is embedded in the 9-bit data packet on transmission and reception, if parity is enabled. Parity error is reported after receiving a byte, if any. The byte is still buffered for reading out; however, the byte could be erroneous.

Frame error is reported after receiving a byte if the stop bit (or in any of the two stop bits in case of two stop bits configuration) is not received correctly. The byte is dropped and not buffered for reading out. The core tries to recover from frame error by re-syncing with the next start bit. However, the data integrity is not guaranteed anymore. If the stop bit continues to be erroneously asserted in the RX line as active-low for another half-bit period, it would be sampled as start bit and the receiver initiates the byte reception. Hence, the frame error may be used to detect if the receiver has gone out-of-sync and appropriate system level action should be taken like abort, retry, reset etc.

7. Testing the IP

UART Controller can be tested with the test benches provided with the IP package. The test bench verifies the IP functionality in loopback configuration. There is also a synthesisable test bench to test the IP on-board. On board, the core is tested by connecting TX and RX pins in loopback configuration. The on-board test bench:

1. Configures the IP in user-defined configuration after reset.
2. Enables TX and RX.
3. Drive data 0x00 to 0xFF at TX.
4. Reports errors on receiving, if any.

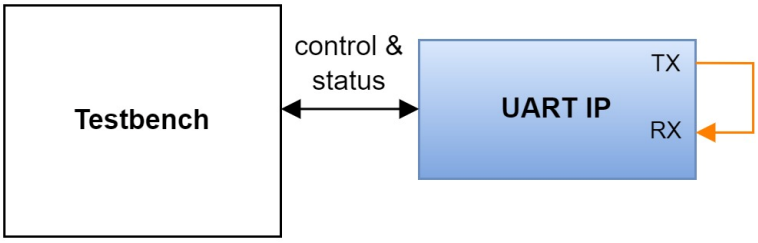

Figure 7.1: UART IP – Testing in Loopback configuration

8. Application Notes

- Once enabled the receiver, The RX line is expected to be idled at 1'b1 at all time when no byte transfer is happening. Else, it may report frame error.
- Disabling the TX and RX is instantaneous and resets the corresponding internal control logic and buffers. If TX has been already sending a data, the transmission is abruptly aborted. If RX has been receiving a data, the data is abruptly dropped. So, care must be taken to not violate the intent of disabling TX or RX.
- The core should be disabled before re-configuring to not break the data integrity.
- The core samples the RX data only in the middle of the receiving bit. It is a make-or-break sampler.

9. Known Limitations/Issues

- The core doesn't support the transmission or reception of break character.

Appendix

a) FPGA Resource Utilization

FPGA Targeted	Artix-7 on Basys-3 Board (XC7A-35T-CPG236-1)
Synthesiser	Vivado 2015.4
Targeted clock frequency	200 MHz
LUTs	129
Registers	98

b) Test Summary

FPGA Targeted	Artix-7 on Basys-3 Board (XC7A-35T-CPG236-1)
Synthesiser	Vivado 2015.4
Core clock	100 MHz
Baud rates tested	300 to 115200 bps
Parity modes tested	All modes
Frame modes tested	All modes
Test mode	Loop back, and with other UART devices
Test result	Successfully passed

Revision History

The following tables shows the revision history of this document.

Date	IP Version	Revision
July-2021	1.1	• Initial version
Jan-2024	1.1	• Added baud rate generation table • Added Error handling section • Updated block diagram • Added Application notes, Testing chapter

UART Controller v1.1

An open-source licensed soft IP core

Developer : **Mitu Raj**
Vendor : **Chip**munk **Logic**™, chip@chipmunklogic.com
Website : **chipmunklogic.com**