UART Controller v1.2

IP User Guide

February 2024

Licensing Notice

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.
- ✓ Fully independent control over TX and RX. Supports operation in half-duplex mode.
- ✓ Configurable parity: Odd, Even, or no parity.
- ✓ Configurable no. of stop bits: 1 or 2.
- ✓ Built-in Baud Generator with 16-bit pre-scaler and configurable baud rate.
- ✓ Break frame transmission/reception.
- ✓ 8x oversampling at RX for balanced speed and error tolerance.
- ✓ Error detection: Frame and Parity errors.
- ✓ Internal loopback support for testing.
- ✓ Simple valid-ready handshaking at the data interface of UART transmitter and receiver for ease of integration with FIFOs.

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.

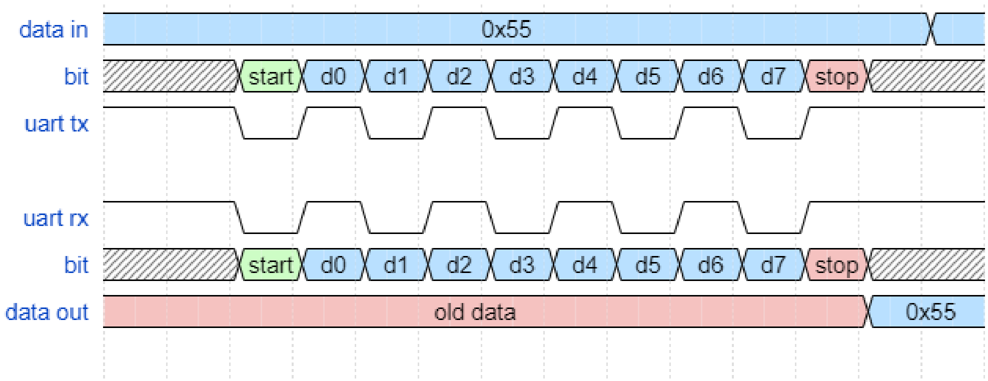

Figure 3.1: UART – 8-bit Data Format

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 8x 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. 16-bit pre-scaling is supported to generate wide range of baud clocks. TX uses baud clock of frequency = desired baud rate, while RX uses 8x oversampling clock that of TX.

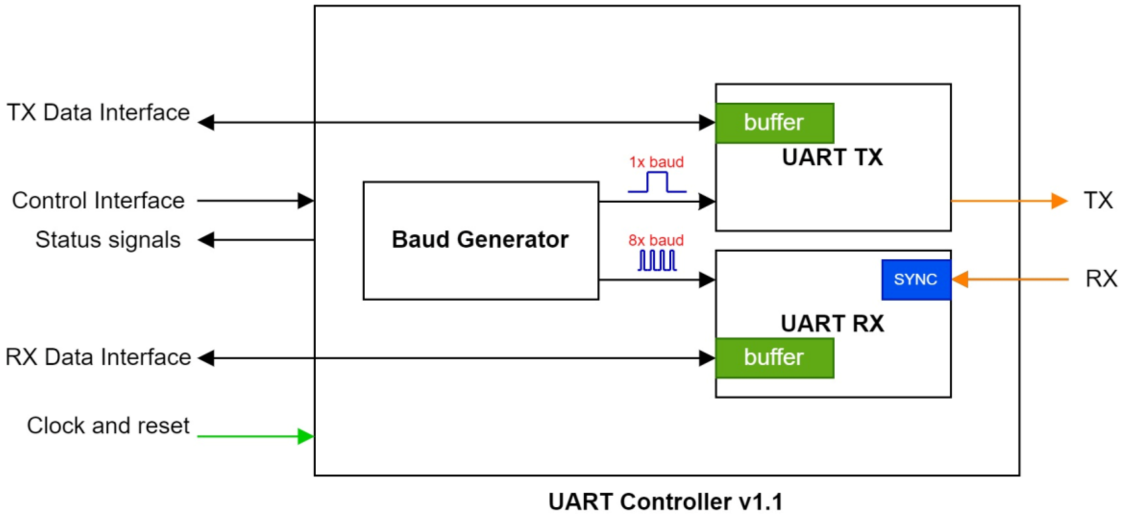

Figure 3.2: UART Controller – Block Diagram

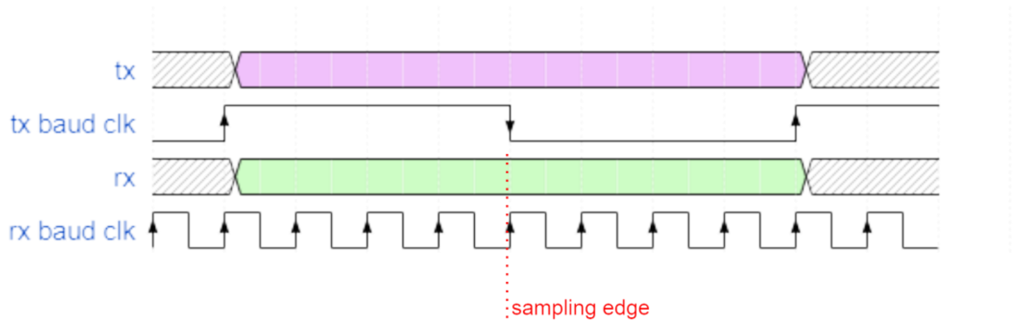

Figure 3.3: UART RX – Sampling at x8

4. Top-level Ports/Interfaces

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

Signal Name	Direction	Width	Description
Clock and Reset			
clk	input	1	Core clock
rstn	input	1	Core reset (Asynchronous active-low)
Serial Interface			
o_tx	output	1	Serial data out
i_rx	input	1	Serial data in
Control Interface			
i_baudrate	input	16	Baud rate configuration value. Valid values: [1, 65535] 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_lpbk_mode_en	input	1	Internal Loopback enable. To enable internal loopback of TX and RX ports. 1'b0 = Internal loopback is disabled. This is the functional mode of the IP. 1'b1 = Internal loopback is enabled; TX is connected to RX internally. This is for testing/diagnostic purpose only. See here for details.
i_tx_break_en	input	1	TX break enable. Enables sending break frame in the next transmission on TX. See here for details.
i_tx_en	input	1	TX enable. To enable the transmitter and the baud clock for serial transmission.
i_rx_en	input	1	RX enable.

			To enable the receiver and the baud clock for serial reception.
i_tx_rst	input	1	TX reset. Resets UART TX and the corresponding baud generator. Active-high reset.
i_rx_rst	input	1	RX reset. Resets UART RX and the corresponding baud generator. Active-high reset.
TX – 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 reset state or previous transmission is in progress. 1'b1 = Ready to accept the byte, and no serial transmission is going on.
RX – Data Interface			
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_tx_state	output	1	State of UART TX. 1'b0 = TX is in disabled state 1'b1 = TX is in enabled state
o_rx_state	output	1	State of UART TX. 1'b0 = RX is in disabled state 1'b1 = RX is in enabled state
o_rx_break	output	1	Break flag. The flag indicates that a break frame has been received. This status bit is valid when the received byte is valid. This is a sticky status bit until next byte is received. See here for details.
o_parity_err	output	1	Parity error.

			Indicates that parity error has occurred for the last byte received. The received byte is still available to read out from the receiver. This error status bit is valid when the received byte is valid. This is a sticky error until next byte is received. See here for details.
o_frame_err	output	1	Frame error. Indicates that frame error (one or more stop bits not detected) has occurred for the last byte received. The received byte is still available to read out from the receiver. This error status bit is valid when the received byte is valid. This is a sticky error until next byte is received. See here for details.

Table 4.1: Top-level Ports/Interfaces

5. Designing with the IP

This chapter discusses guidelines including clocking and reset, configuration, and other considerations 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 asynchronous and active-low. The assertion is asynchronous and the de-assertion should be synchronized to `clk`. The core has built-in reset synchronizer to take care of the clean de-assertion.

5.2 Configuring the IP

Clock and Reset Sequencing

1. Assert the core reset.
2. Bring up the core clock.
3. Assert the core reset for at least 8 clock cycles.
4. Release the reset.
5. 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. Load the next byte when TX data interface is ready again.
5. 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

An integer value $\boldsymbol{B}$ should be configured in `i_baudrate` to set the baud rate of serial data transfer at the UART transmitter and receiver. It can be calculated as follows:

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

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 Start and Stop bit Transmission and Detection

The TX line idles at 1'b1 by default after reset. The transmitter begins the transmission of byte by pulling the TX line down to 1'b0 for a bit period. This is the start bit of the frame. After sending the byte, the TX line is pulled up to 1'b1 for a bit period. This is the stop bit of the frame. The transmitter brings the TX line back to the idle state of 1'b1 after the transmission.

The receiver remains in idle state as long as the RX line idles at 1'b1. Any falling edge is detected by the receiver and is regarded as possible start bit transition. The receiver then samples the bit in the middle to confirm whether it's a valid start bit. If 1'b1 is sampled in the middle, it is a valid start bit, otherwise the receiver aborts the start bit detection and returns to the idle state.

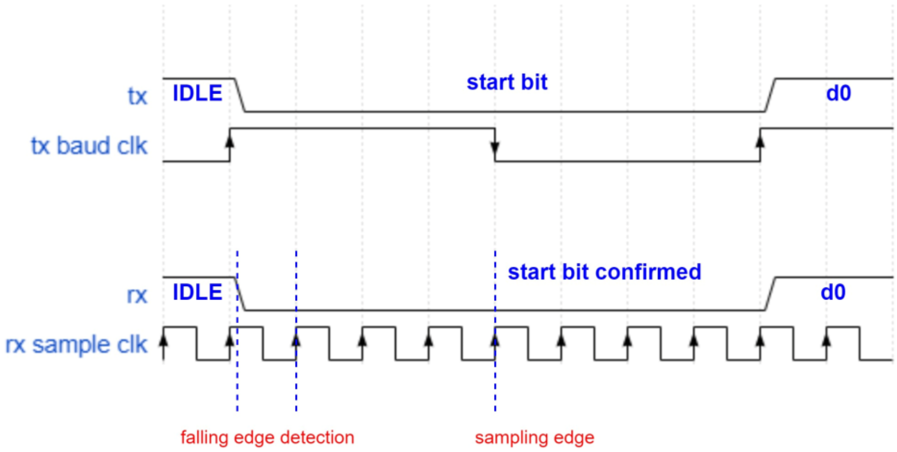

Figure 5.1: Start bit Transmission and Detection

5.4 Frame Format

The core supports different frame formats as shown in the figure.

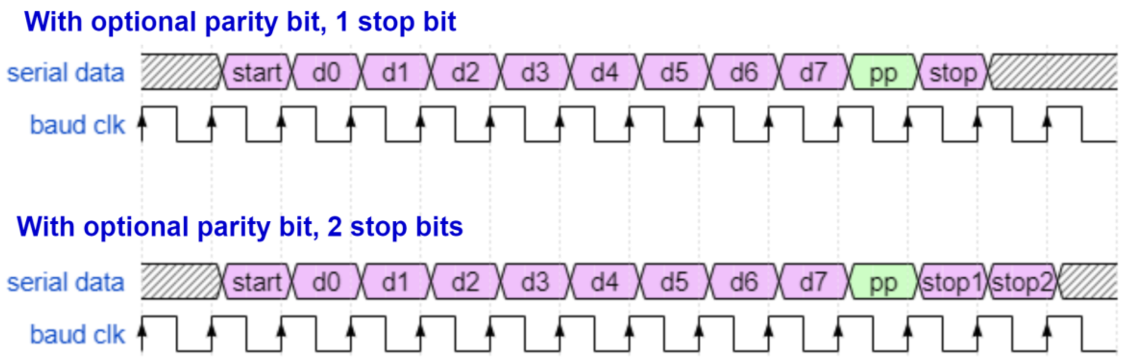

Figure 5.2: Frame formats supported by the core

The frame can be 8-bit or 9-bit (if parity bit is enabled). Parity bit is the optional 9th bit if parity is enabled. The parity is byte parity (Odd/Even).

No. of 1s in the data byte	Odd parity bit	Even parity bit
Odd	1’b0	1’b1
Even	1’b1	1’b0

Table 5.3: Parity bit in a frame

If two stop bits are enabled, two stop bits are added after the byte by the transmitter. And the receiver expects two stop bits in the frame received. The receiver verifies both stop bits for frame error.

5.5 Resetting, Enabling, and Disabling Transmitter

The transmitter should be enabled to enable the TX baud clock and serial transmission. On enabling, the loaded byte is transmitted via TX.

The transmitter can be disabled any time. If the transmitter is idle, it is immediately disabled. If the transmitter is busy transmitting a byte, the byte is allowed to transmit completely before it is actually disabled. The state of transmitter (`o_tx_state`) should be read as 1'b0 to confirm that it has been disabled.

The transmitter can also be put in reset state by driving 1'b1 at `i_tx_rst`. This immediately disables and resets the transmitter, baud clock, and buffers regardless of whether it was transmitting a byte or not.

5.6 Resetting, Enabling, and Disabling Receiver

The receiver should be enabled to enable the RX baud clock and serial reception. On enabling, the receiver starts sampling the RX line and waits for start bit detection.

The receiver can be disabled any time. If the receiver is idle and waiting for start bit detection, it is immediately disabled. If the receiver has already detected start bit and is busy receiving the byte, it waits for the stop bit and completes the byte reception before it is actually disabled. The state of receiver (`o_rx_state`) should be read as 1'b0 to confirm that it has been disabled.

The receiver can also be put in reset state by driving 1'b1 at `i_rx_rst`. This immediately disables and resets the receiver, baud clock, and buffers regardless of whether it was receiving a byte or not. If it was busy receiving a byte, the byte is lost.

5.7 Data Transfer with the IP

The core has 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 5.3 shows how a typical handshaking is done with the core to transmit and receive two bytes.

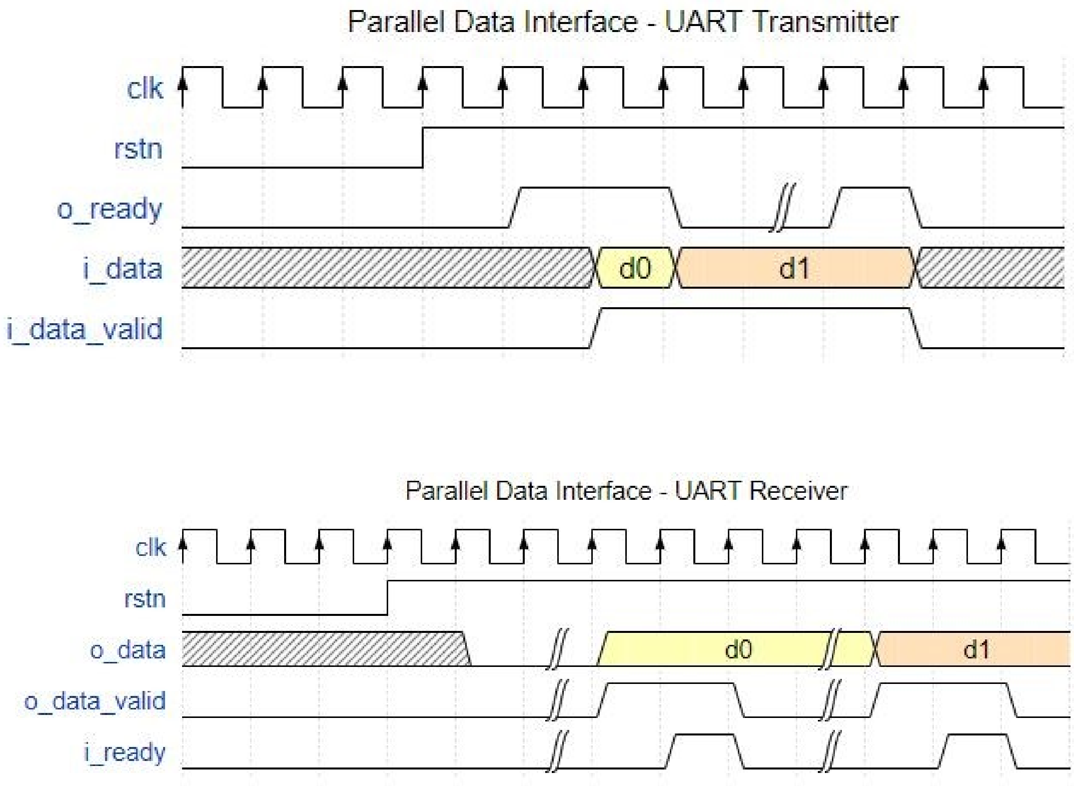

Figure 5.3: Valid-Ready handshaking with the IP

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.

5.8 Sending Breaks

The core supports sending break character. A break frame consists of start bit, followed by 9 or 10 bits of 0s, and a stop bit. The transmitter inserts 1 or 2 stop bits (break-limiter stop bits) at the end of a break frame to guarantee the recognition of the start bit of the next frame.

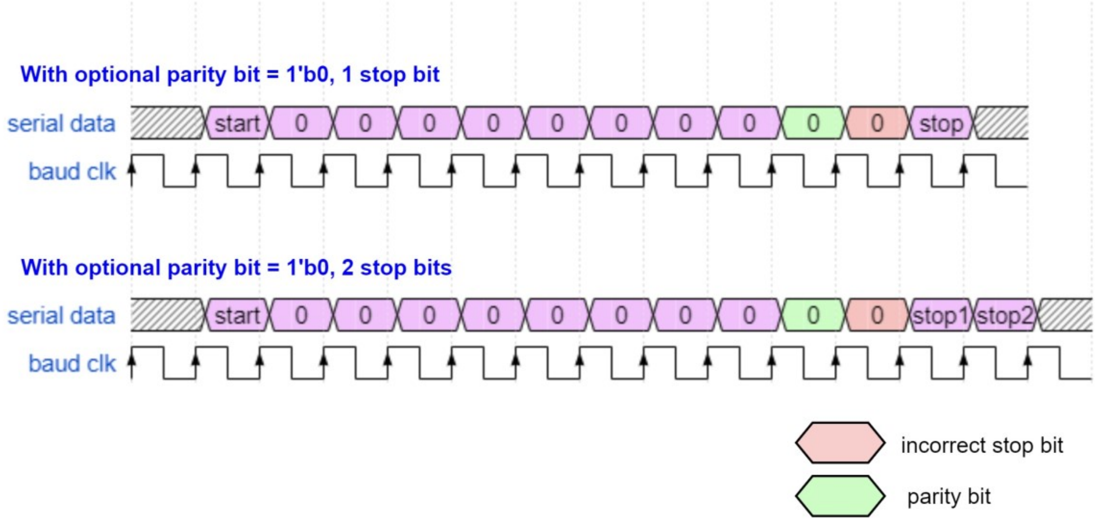

Figure 5.4: Break Frame transmission

Even if parity is enabled, the parity bit is always sent as 1'b0.

Sequence to send a Break Frame

1. Enable TX break by setting `i_tx_break_en` to 1'b1.
2. Load a dummy byte to TX (the data value is ignored).
3. Enable TX.
4. Break frame is sent on TX with 10 or 11 bits of 0s followed by stop bit.
5. To send a data byte after the break, disable TX break and load the byte to TX.

The length of break frame is fixed. It is not possible to send longer break frames. To achieve longer breaks, multiple break frames have to be sent in succession instead.

5.9 Receiving Breaks

The core supports receiving breaks during data reception. The break condition is minimum of 10 or 11 bits of 0s (if parity is enabled) in a frame. Following are the conditions in the receiver to detect a break frame.

1. Start bit followed by at least 9 bits or 10 bits of 0s (if parity enabled) is regarded as break frame.
2. Always sets the frame error because the break frame contains incorrect stop bit.
3. Sets the flag `o_rx_break`.
4. The received data in the buffer will be 0x00.
5. Parity error is also generated if odd parity is configured at RX (because the parity bit = 1'b0 in a break frame, instead of the expected parity bit = 1'b1).

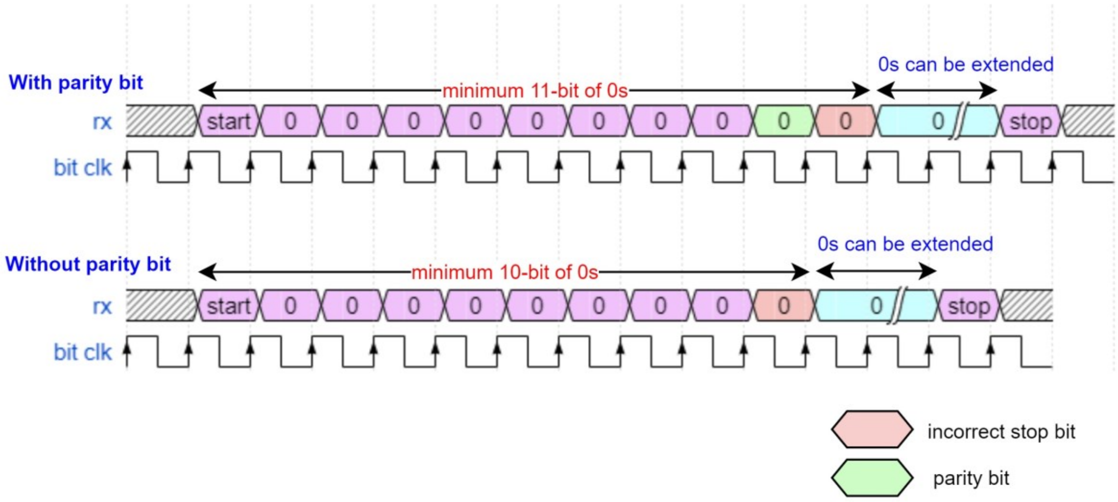

Figure 5.5: Break Frame reception

Breaks longer than 10 or 11 bits of 0s (if parity is enabled) can be received. The receiver idles after receiving a break. Next data frame can be received only when a stop bit (at least one stop bit) is received after the break. The receiver synchronizes to the start bit of the next frame and resumes the reception.

5.10 Internal Loopback support

The core supports internal loopback mode for testing and diagnostic purpose. In loopback mode, TX port is connected to RX port internally.

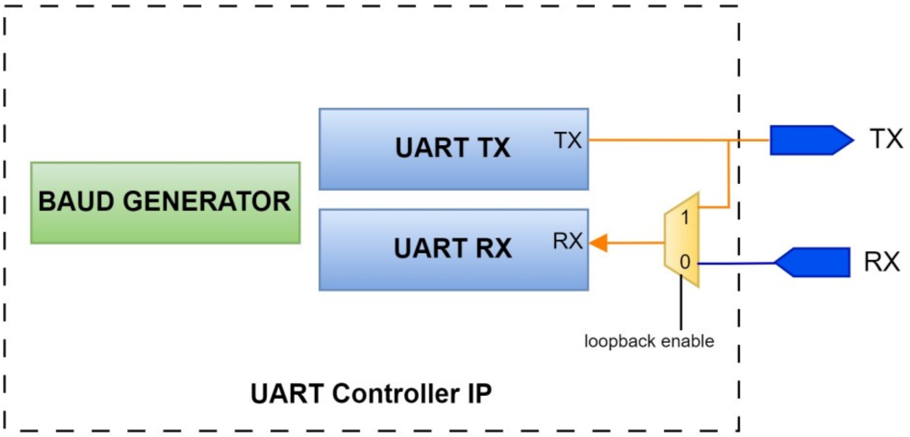

Figure 5.6: Loopback mode in UART Controller

Sequence to enable Loopback testing

1. Disable TX and RX.
2. Configure TX and RX.
3. Enable loopback mode by setting `i_lpbk_mode_en` to 1'b1.
4. Enable TX and RX.
5. Load byte pattern to TX, observe the pattern received at RX.

Loopback mode should be enabled/disabled only when TX and RX are in the disabled state to avoid glitches/broken frames.

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

The error becomes valid when the byte received is valid and remains sticky until next byte is received.

If parity is enabled, parity bit is expected to be embedded in the 9-bit data packet on transmission and reception. Parity is checked and 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 (any of the two stop bits in case of two stop bits configuration) is not received correctly. The byte is still buffered for reading out. The core will try to recover from frame error by re-syncing with the next start bit. However, frame error should be addressed correctly by the system.

- If the frame error was due to the reception of break character (break flag should be read as 1'b1), the core will re-sync to the next start bit (the falling edge after the break-limiter stop bit) and the reception continues without any synchronization errors.
- If the frame error was not due to break (break flag should be read as 1'b0)., but due to other reasons such as noise, de-synchronization, broken communication link, wrong frame format etc, the core will re-sync to the next falling edge assuming it is the start bit. In this case, the communication link integrity cannot be guaranteed anymore. The receiver could have gone out-of-sync and appropriate system level corrective action should be taken to re-establish the communication link synchronization.

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 with frequent break frames at TX.
4. Reports errors on receiving, if any.

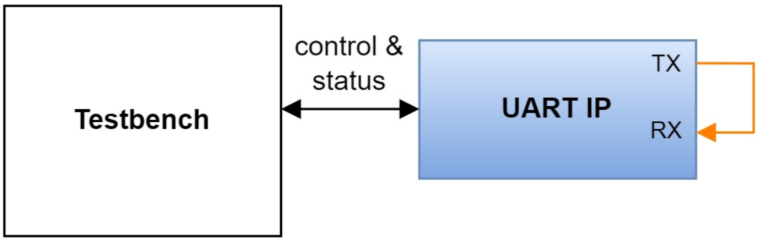

Figure 7.1: UART IP – Testing in Loopback configuration

8. Application Notes

- The receiver can be configured for 1 stop bit and can still receive two or more stop bits from the external transmitter. In this case, the receiver verifies only the first stop bit and gets more time to synchronize to the next frame. But, if the receiver is configured for 2 stop bits, it always expects and verifies both stop bits in the frame received.
- The core can function in half-duplex mode as only a "Transmitter" or "Receiver" by disabling RX or TX and enabling only the other. Alternatively, RX or TX can be kept in reset state. This saves power.
- 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.
- The core supports wide range of core clock and baud rates with the built-in 16-bit pre-scaler in the Baud Generator. The value configured should adhere to the supported range and the baud rate error tolerance as described in Baud Rate Configuration.

9. Known Limitations/Issues

[NOT APPLICABLE]

Appendix

a) FPGA Resource Utilization

FPGA Targeted	Xilinx Zybo Z7-20 (XC7-Z020-CLG400-1), Artix-7 FPGA based board
Synthesiser	Vivado 2019.2
Targeted clock frequency	100 MHz
LUTs	142
Registers	115

b) Test Summary

FPGA Targeted	Xilinx Zybo Z7-20 (XC7-Z020-CLG400-1), Artix-7 FPGA based board
Synthesiser	Vivado 2019.2
Core clock	10-100 MHz
Baud rates tested	300 to 115200 bps
Parity modes tested	All modes
Frame modes tested	All modes
Test mode	Loopback internal/external, 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
Feb-2024	1.2	• Initial version

UART Controller v1.2

An open-source licensed soft IP core

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