UART IP

IP Version: v1.5.0

User Guide

FPGA-IPUG-02105-1.6

December 2025

Disclaimers

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
CPU	Central Processing Unit
DLR	Divisor Latch Register
EBR	Embedded Block Random Access Memory
EIA	Electronic Industries Association
FIFO	First-In, First-Out
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
IER	Interrupt Enable Register
IIR	Interrupt Identification Register
IP	Intellectual Property
LCR	Line Control Register
LSR	Line Status Register
LSE	Lattice Synthesis Engine
LUT	Look-Up Table
PFU	Programmable Function Unit
RBR	Receiver Buffer Register
RTL	Register Transfer Language
RO	Read-Only
RSR	Receiver Shift Register
RSVD	Reserved
RW	Read-Write
Rx	Receiver
THR	Transmitter Holding Register
TSR	Transmitter Shift Register
Tx	Transmitter
UART	Universal Asynchronous Receiver/Transmitter
WO	Write-Only

1. Introduction

Universal Asynchronous Receiver/Transmitter (UART) Transceiver IP core performs serial-to-parallel conversion on data characters received from a peripheral UART device, and parallel-to-serial conversion on data characters received from the host located inside the FPGA through an APB Interface.

1.1. Overview of the IP

The Lattice Semiconductor UART IP core is designed for use in serial communication and supports the RS-232 protocol. The IP shares many characteristics with the NS16450 UART; however, to conserve FPGA resources, the IP is not identical to the NS16450 UART. As a result, it is not source-code compatible, meaning existing driver code for the NS16450 UART does not work on the Lattice UART IP core.

1.2. Quick Facts

Table 1.1. Summary of the UART IP

IP Requirements	Supported Devices	All
	IP Changes[1]	For a list of changes to the IP, refer to the UART IP Release Notes (FPGA-RN-02023).
Resource Utilization	Supported User Interface	Advanced Peripheral Bus (APB)
	Resources	Refer to Appendix A. Resource Utilization.
Design Tool Support	Lattice Implementation	IP core v1.5.0 – Lattice Radiant software 2025.2
	Synthesis	Lattice Synthesis Engine (LSE) Synopsys Synplify Pro® for Lattice
	Simulation	For the list of supported simulators, see the Lattice Radiant Software user guide.

Note:

1. In some instances, the IP may be updated without changes to the user guide. This user guide may reflect an earlier IP version but remains fully compatible with the later IP version. Refer to the IP Release Notes for the latest updates.

1.3. IP Support Summary

Table 1.2. UART IP Support Readiness

Device Family	Radiant Timing Model	Hardware Validated
Crosslink-NX	Final	No
Certus-NX	Final	No
Certus-N2	Preliminary	No
MachXO4	Preliminary	No

1.4. Features

Key features of the UART IP include:

- APB 1.0 interface.
- Similarity to the National Semiconductor NS16450 UART, with different register addresses.
- Optional 16-word-deep FIFO implemented in the UART transmit/receive path when FIFO mode is selected.
- Insertion or extraction of standard asynchronous communication bits (start, stop, and parity) to or from the serial data.
- Holding and shifting registers, eliminating the need for precise synchronization between the host (APB interface) and serial data.
- Common interrupt line for all internal UART data and error events. Interrupt conditions include:
    - Receiver line errors
    - Receiver buffer available
    - Transmit buffer empty
    - Detection of status flag change
- Fully prioritized interrupt system control.
- Fully programmable serial interface characteristics, including:
    - Configurable data widths of 5, 6, 7, or 8 bits
    - Even-, odd-, or no-parity bit generation and detection
    - 1-, 1.5-, or 2-stop bit generation and detection
    - False start bit detection
    - Line break generation and detection
    - Interactive control signaling and status reporting capabilities
- Configurable Baud Rate support for Standard and Custom modes:
    - In standard mode, a programmable divisor latch for baud rates provides fixed pre-defined values.
    - In custom mode, baud rate accepts any value in the range of 2,400–1,000,000.

1.5. Licensing and Ordering Information

The UART IP is provided at no additional cost with the Lattice Radiant software.

1.6. Minimum Device Requirements

There is no limitation in device speed grade for UART IP core. For the minimum required resources to instantiate this IP and maximum clock frequency supported, see the Appendix A. Resource Utilization section.

1.7. Naming Conventions

1.7.1. Nomenclature

The nomenclature used in this document is based on Verilog HDL.

1.7.2. Signal Names

- _*n* are active low (asserted when value is logic 0)
- _*i* are input signals
- _*o* are output signals

1.7.3. Attribute Names

Attribute names in this document are formatted in title case and italicized (*Attribute Name*).

2. Functional Description

2.1. IP Architecture Overview

The UART IP Core performs two main functions:

- Serial-to-parallel conversion on data characters received from an external UART device; and
- Parallel-to-serial conversion on data characters received from the host located in the FPGA.

The host can read the complete status of the UART at any time during the functional operation. Status information reported includes the type and condition of the transfer operations being performed by the UART IP core, as well as any error conditions (parity, overrun, framing, or break interrupt).

The UART IP implements a processor-interrupt system similar to UART 16450. Interrupts can be programmed to meet specific requirements, minimizing the computing required to handle the communication link. However, the UART IP currently does not implement the MODEM control feature of UART 16450.

The registers of UART IP Core are accessed by the host (FPGA internal components) through an AMBA APB interface. The functional block diagram of UART IP Core is shown in Figure 2.1. The dashed lines in the figure are optional components/signals, which means they may not be available in the IP when disabled in the attribute.

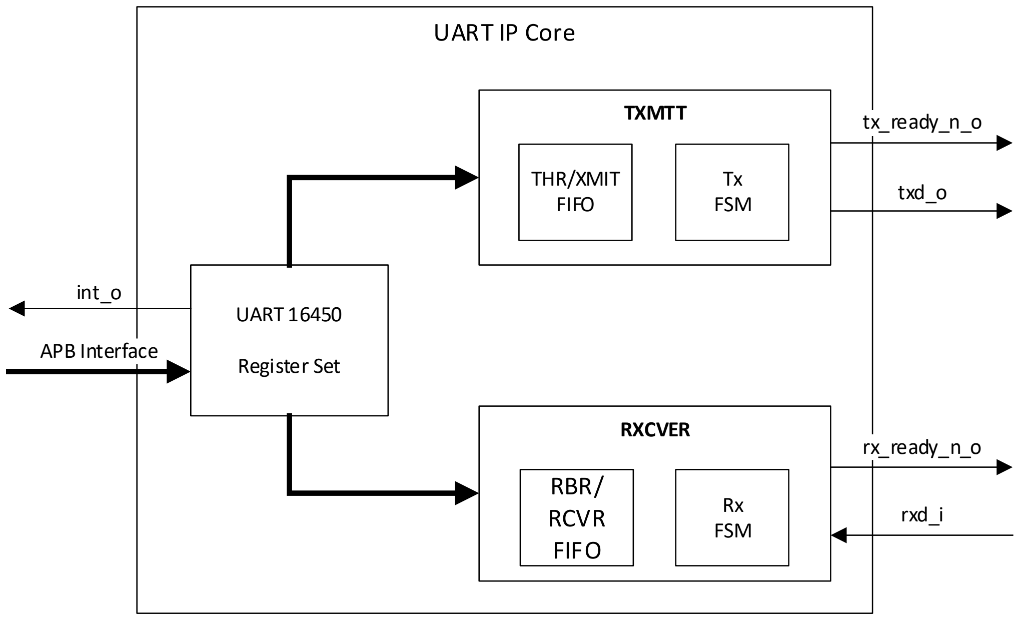


Figure 2.1. Functional Block Diagram

2.2. Clocking

The UART IP core uses a single clock source, clk_i.

2.3. Reset

The UART IP Core has a single hardware reset signal, rst_n_i, which is an asynchronous active-low reset. While the assertion of the reset can be asynchronous, the reset negation is synchronous. When asserted, all output ports and internal registers are forced to their reset values.

2.4. User Interfaces

Table 2.1. User Interfaces and Supported Protocols

User Interface	Supported Protocols	Description
Control	APB	The APB interface has no wait state for both write and read access. For more information on the APB interface and the timing diagrams, refer to the AMBA 3 APB Protocol v1.0 Specification.

2.5. Operation Details

2.5.1. UART Clock Frequency

The UART only has a single clock input, clk_i, which is used for the entire IP. This includes the APB interface for transferring information with the host internal to the FPGA, and the UART's txd_o and rxd_i for sending and receiving serial data with an external UART device.

2.5.2. Receiver

In non-FIFO mode, the Serial Receiver (RXCVER) section contains an 8-bit Receiver Buffer Register (RBR) and a Receiver Shift Register (RSR). In FIFO mode, the RBR is a 16-word-deep FIFO.

Since the serial frame is asynchronous to the receiving clock, a high-to-low transition on the rxd_i pin is treated as the start bit of a frame. However, to avoid receiving incorrect data due to rxd_i signal noise, false-start bit detection is implemented. The UART requires the start bit to be low or at least 50% of the baud rate clock. The UART samples rxd_i for half the bit duration, and if the sample is low, a start bit is detected.

Once a valid start bit is received, the data bits, parity bit, and stop bit are sampled at the programmed baud rate. The start bit is expected to be exactly equal to the bit duration. Thus, each of the following bits is sampled at the center of the bit period itself.

The receive logic monitors the incoming data stream and reports the state of the incoming line in the Line Status Register (LSR). The LSR is used to indicate overrun, parity, and framing errors. It also indicates when a line break has been received.

Whenever a framing error is detected, the UART assumes that the error is due to the start bit of the following frame and tries to resynchronize it. To do this, it samples the start bit twice. If both samples of the rxd_i are low, the UART resynchronizes and accepts the data following the second start bit. The resynchronization does not occur if a framing error is caused by a break character.

Once the entire data bit is received, the data in the Receive Shift Register (RSR) is transferred to the Receive Buffer Register (RBR). The LSR.data_rdy bit is set to 1'b1 to indicate to the CPU that a byte of data has been received. The external rx_ready_n_o output is asserted (that is, logic 0) simultaneously with the LSR.data_rdy bit. You can also configure the UART to generate an interrupt upon successful transfer from the RSR to the RBR by writing 1'b1 to IER.rda_int_en.

2.5.3. Transmitter

The Serial Transmitter (TXMITT) section consists of an 8-bit Transmitter Holding Register (THR) and Transmitter Shift Register (TSR) when the UART is in non-FIFO mode. When the UART is in FIFO mode, the THR is a 16-word-deep FIFO. The UART provides two methods of indicating the status of THR: a tx_ready_n_o output signal or a Transmitter Holding Register empty (thr_empty) flag in the Line Status Register (LSR). If UART is in non-FIFO mode when THR is empty, the tx_ready_n_o signal becomes active low, and the LSR.thr_empty flag is set to a logic 1'b1. THR is written only when LSR.thr_empty is 1'b1 or tx_ready_n_o is 1'b0. After the data is loaded in THR, it is loaded to TSR for transmission. When another data is written to THR the LSR.thr_empty is reset to logic 0 and the tx_ready_n_o pin goes inactive high.

If the UART is in FIFO mode, when the THR FIFO is not full, the tx_ready_n_o signal becomes active low to signal that the transmitter logic can still receive new data for transition. For this mode, the LSR.thr_empty is set to logic 1'b1 only when FIFO is already empty. This is because the host in the FPGA fabric cannot determine the actual amount of data in the FIFO. Thus, it is recommended that the host writes up to 16 data to RBR when LSR.thr_empty is 1'b1. The serial data transmission is automatically enabled after the data is loaded into THR. First, a start bit (logic 0) is transmitted and the data in THR is automatically parallel-loaded to TSR. The data bits are shifted out of TSR, followed by the parity bit, if parity is enabled. Finally, the stop bit (logic 1) is generated to indicate the end of the frame. After a frame is fully transmitted, another frame is transmitted immediately if THR is not empty. This automatic sequencing causes the frames to be transmitted back-to-back, which increases the transmission bandwidth. The txd_o signal is held high when no transmission is in progress.

2.5.4. Interrupt

The common interrupt request signal (int_o) asserts when any interrupt condition occurs, and the interrupt is enabled in the Interrupt Enable Register (IER). The UART prioritizes interrupts into three levels to minimize external software interaction and records these in the interrupt identification register (IIR). Table 2.2 shows the four levels of interrupt types, the condition for asserting the interrupts, and the corresponding clearing control.

Performing a read cycle on IIR freezes all interrupts and indicates the highest priority pending interrupt to the host. The IIR does not acknowledge or record new interrupts until the host services the pending interrupt. Whenever the IIR is read, the current pending interrupt is cleared, meaning the IIR can record the new interrupt again. Any pending lower-priority interrupt becomes visible in the IIR after the previous interrupt is cleared and the IIR is read.

Table 2.2. Interrupt Control Function

IIR[2:0]	Priority	Interrupt Type[1]	Interrupt Assertion Cause	Interrupt Clearing Control
3'b001	—	None	None	—
3'b110	Highest	Receiver Line Status	Overrun Error or Parity Error or Framing Error or Break Interrupt	Reading the Line Status Register
3'b100	Second	Received Data Available	RCVR FIFO reached the trigger level (FCR.trig_lvl)	Reading the Receiver Buffer Register until RCVR FIFO drops below trigger level
3'b010	Third	Transmitter Holding Register Empty	XMIT FIFO is empty	Writing to XMIT FIFO.
3'b000	Fourth	MODEM Status	Not yet supported	—

Note:

1. If the interrupt type is disabled in the IER, the interrupt control function treats the interrupt as not asserted even if the interrupt source is asserted. Hence, the IIR value is affected by the IER.

2.6. Data Format

The character data is written to THR and read from RBR in little-endian format, as shown in Figure 2.2.

7-Bit Data

0	D6	D5	D4	D3	D2	D1	D0

8-Bit Data

D7	D6	D5	D4	D3	D2	D1	D0

Figure 2.2. Tx/Rx Data Format

2.7. Timing Diagrams

During transmission, data is shifted out by the transmitter logic in LSB-first format, as shown in Figure 2.3. This timing diagram shows the transmission of one character without flow control. The UART IP core then sends the start bit, character data, and stop bit on the txd_o line.

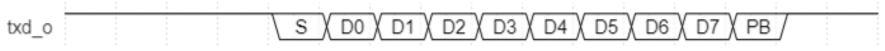


Figure 2.3. Transmit Operation Timing Diagram without Flow Control Signals (1 byte)

When transmitting multiple data bytes, the next byte is transmitted immediately after the stop bit of the first byte. This is shown in Figure 2.4. In this diagram, one stop bit is generated.

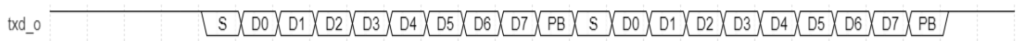


Figure 2.4. Transmit Operation Timing Diagram without Flow Control Signals (2 bytes)

Figure 2.5 shows the receive operation wherein the external UART transmits the data being received. The data format is similar to Figure 2.3 but the transfer occurs on the rxd_i signal and transfer direction is reversed.

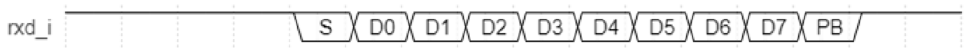


Figure 2.5. Receive Operation Timing Diagram with Flow Control Signals (1 byte)

Figure 2.6 shows the timing diagram for receiving multiple data bytes. Similar to Figure 2.4, the next transaction starts immediately after the previous stop bit.

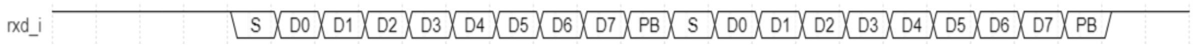


Figure 2.6. Transmit Operation Timing Diagram with Flow Control Signals (2 bytes)

The APB interface is not described in this document. Refer to the AMBA 3 APB Protocol v1.0 Specification for APB interface detail and the timing diagram. The UART's APB interface has no wait state for both write and read access.

3. IP Parameter Description

The configurable attributes of the UART IP are shown in the following tables. You can configure the IP by setting the attributes accordingly in the IP Catalog Module/IP Block Wizard of the Lattice Radiant software.

Wherever applicable, default values are in bold.

3.1. General

Table 3.1. General Attributes

Attribute	Selectable Values	Description
General		
Enable APB	**Checked**	Selects the memory-mapped interface for register access by the host. The interface is currently fixed to the AMBA APB. For informational purposes only.
System Clock Frequency (MHz)	2–200, **50**	Specifies the target frequency of system clock. This value is used for baud rate calculation.
Serial Data Width	5, 6, 7, **8**	Specifies the default data bit width of UART transactions by setting the reset value of char_len_sel field of Line Control Register: • 5 – Reset value is 2’b00 • 6 – Reset value is 2’b01 • 7 – Reset value is 2’b10 • 8 – Reset value is 2’b11
Stop Bits	**1**, 2	Specifies the absence/presence of parity by setting the reset value of parity_en bit of the Line Control Register: • Checked – Reset value is 1’b1 • Unchecked – Reset value is 1’b0
Parity Enable	Checked, **Unchecked**	Specifies the absence/presence of parity by setting the reset value of parity_en bit of the Line Control Register: • Checked – Reset value is 1’b1 • Unchecked – Reset value is 1’b0
ODD Parity	Checked, **Unchecked**	Specifies the default parity type (odd/even) by setting the reset value of even_parity_sel in the Line Control Register: • Checked – Reset value is 1’b0 • Unchecked – Reset value is 1’b1 Active only if *Parity Enable* is Checked.
Enable Stick Parity	Checked, **Unchecked**	Enables the generation and checking of stick parity by specifying the reset value of the stick_parity_en bit of Line Control Register: • Checked – Reset value is 1’b1 • Unchecked – Reset value is 1’b0 Active only if *Parity Enable* is Checked.
Baud Rate		
Baud Rate Type	**Standard**, Custom	Selects between standard baud rate and custom baud rate for the reset value of Divisor Latch Register. The selected baud rate is used to set the reset value of Divisor Latch Register as follows: {DLR_MSB, DLR_LSB} = *System Clock Frequency (MHz)* × 1,000,000 / Selected Baud Rate
Standard Baud Rate	2,400, 4,800, 9,600, 14,400, 19,200, 28,800, 38,400, 56,000, 57,600, **115,200**	Specifies the baud rate of UART transactions from the standard baud rate options. Active only if *Baud Rate Type* == Standard.

Attribute	Selectable Values	Description
Custom Baud Rate	2,400–1,000,000, **115,200**	Specifies the baud rate of UART transactions based on a user-defined value. Active only if *Baud Rate Type* == Custom.
UART Feature Enables		
FIFO Enable	Checked, **Unchecked**	Enables the implementation of FIFO in both the transmit and receive paths. This feature is disabled when the device family is MachXO4.
MODEM Enable	**Unchecked**	Enables the presence of MODEM control signals. This feature is currently not supported. For informational purposes only.
Rx Ready Enable	Checked, **Unchecked**	Enables the presence of the rx_ready_n_o signal in the generated IP.
Tx Ready Enable	Checked, **Unchecked**	Enables the presence of the tx_ready_n_o signal in the generated IP.

4. Signal Description

This section describes the UART IP ports along with their description.

Table 4.1. Ports Description

Port Name	Type	Width	Description
Clock and Reset			
clk_i	Input	1	System clock. Frequency range is from 2 MHz – 200 MHz.
rst_n_i	Input	1	Asynchronous active low reset.
Interrupt Port			
int_o	Output	1	Interrupt signal.
Serial Interface			
txd_o	Output	1	Serial Output – serial data output to the communication link. Reset value: 1'b1
rxd_i	Input	1	Serial Input – serial data input from the communication link.
APB Interface			
apb_psel_i	Input	1	Select signal – indicates that the completer device is selected, and a data transfer is required.
apb_paddr_i	Input	6	Address signal.
apb_pwdata_i	Input	32	Write data signal. Bits [31:8] are not used.
apb_pwrite_i	Input	1	Direction signal: • Write = 1 • Read = 0
apb_penable_i	Input	1	Enable signal – indicates the second and subsequent cycles of an APB transfer.
apb_pready_o	Output	1	Ready signal – indicates transfer completion. The completer device uses this signal to extend an APB transfer.
apb_pslverr_o	Output	1	Error signal – indicates a transfer failure. This signal is tied to 1'b0.
apb_prdata_o	Output	32	Read data signal. Bits [31:8] are not used.
Auxiliary Signals			
rx_ready_n_o	Output	1	Indicates that the UART receiver has received data and is available in the Receive Buffer Register. Reset value: 1'b1
tx_ready_n_o	Output	1	Indicates that the UART transmitter is ready to receive new data for transmission through the txd_o signal. In FIFO mode (when *FIFO Enable* is checked), this signal is asserted low when at least one data item is available in the XMIT FIFO. In non FIFO mode, this signal is asserted low when the Transmitter Holding Register is empty. Reset value: 1'b1

5. Register Description

The register address map, shown in Table 5.1, specifies the available IP Core registers. This is based on register set of UART 16450, but the offset address has been changed to simplify the access to each registers.

The offset of each register increments by four to allow easy interfacing with the processor and system buses. In this case, each register is 32-bit wide, with the lower 8 bits being used and the upper 24 bits remaining unused. The unused bits are treated as reserved – write access is ignored, and read access returns *0*.

Table 5.1. Registers Address Map

Offset	Register Name	Access Type	Description
0x00	RBR	RO	Receive Buffer Register
0x00	THR	WO	Transmitter Holding Register
0x04	IER	RW	Interrupt Enable Register
0x08	IIR	RO	Interrupt Identification Register
0x0C	LCR	RW	Line Control Register
0x10	Reserved	RSVD	Reserved
0x14	LSR	RO	Line Status Register
0x18–0x1C	Reserved	RSVD	Reserved
0x20	DLR_LSB	WO	Divisor Latch Register LSB
0x24	DLR_MSB	WO	Divisor Latch Register MSB
0x28–0x3C	Reserved	RSVD	Reserved

The behavior of registers to write and read access is defined by its access type, which is defined in Table 5.2.

Table 5.2. Access Type Definition

Access Type	Behavior on Read Access	Behavior on Write Access
RO	Returns register value	Ignores write access.
WO	Returns 0	Updates register value.
RW	Returns register value	Updates register value.
RSVD	Returns 0	Ignores write access.

5.1.1. Receive Buffer Register (RBR)

The Receive Buffer Register is the interface to the Receiver Buffer/FIFO (RCVR FIFO). In FIFO mode, reading from this register returns the output data stored in the RCVR FIFO. If read operation is performed when the RCVR FIFO is empty, the last data in the FIFO is returned. In non-FIFO mode, this register can only hold one data. Thus, the receive data should be read before the next data is completely received.

The rx_ready_n_o signal is set to 1'b0 and LSR.data_rdy is set to 1'b1 when this register/FIFO has data.

Table 5.3. Receive Buffer Register

Field	Name	Description	Access	Width	Reset
[7:0]	rx_data	Receive Data – contains the received data from UART.	RO	8	0x00

5.1.2. Transmitter Holding Register (THR)

The Transmitter Holding Register is the interface to the Transmitter Buffer/FIFO (XMIT FIFO). In FIFO mode, writing to this register adds the write data to the XMIT FIFO. If a write operation is performed when XMIT FIFO is full, data is lost.

The host can only determine if the XMIT FIFO is empty or not (through LSR.thr_empty); it cannot determine how many characters are currently in the XMIT FIFO. Thus, it is recommended to initiate the write sequence to this register only when the XMIT FIFO is empty, at which the host can write up to 16 characters of data.

In non-FIFO mode, this register can store only one data item while the previously written data is being sent. For example, you can write two data items in which the first data item immediately goes to the Transmitter Shift Register for transmission, and the second data is stored in this register until the previous one is fully transmitted.

Table 5.4. Transmitter Holding Register

Field	Name	Description	Access	Width	Reset
[7:0]	tx_data	Transmit Data – contains the transmit data to UART.	WO	8	0x00

5.1.3. Interrupt Enable Register (IER)

The Interrupt Enable Register enables three types of UART interrupts. When enabled, each interrupt can individually activate the int_o output signal.

Each interrupt can individually be enabled or disabled by writing to the corresponding bit in the IER. Disabling an interrupt prevents it from being indicated as active in the IIR and from activating the int_o output signal. On the other hand, the UART status, including Line Status Registers, is not affected by the IER setting.

Table 5.5. Interrupt Enable Register

Field	Name	Description	Access	Width	Reset
[7:3]	reserved	—	RSVD	5	—
[2]	rls_int_en	Receiver Line Status Interrupt Enable – this bit controls the enabling of Receiver Line Status Interrupt: • 1'b0 – Interrupt is disabled • 1'b1 – Interrupt is enabled	RW	1	Refer to the *Receiver Line Status Interrupt* attribute.
[1]	thre_int_en	Transmitter Holding Register Empty Interrupt Enable – this bit controls the enabling of Transmitter Holding Register Empty Interrupt: • 1'b0 – Interrupt is disabled • 1'b1 – Interrupt is enabled	RW	1	Refer to the *Transmitter Holding Register Empty Interrupt* attribute.
[0]	rda_int_en	Received Data Available Interrupt Enable – this bit controls the enabling of Received Data Available Interrupt and Timeout Interrupt (timeout_int): • 1'b0 – Interrupt is disabled • 1'b1 – Interrupt is enabled	RW	1	Refer to the *Received Data Available Interrupt* attribute.

5.1.4. Interrupt Identification Register (IIR)

To minimize software overhead during data character transfers, the UART IP core prioritizes interrupts into four levels and records these in the Interrupt Identification Register.

When the host reads the IIR, the IP core only indicates the highest priority pending interrupt. After clearing the source of the highest priority pending interrupt, the IIR updates to reflect the next priority pending interrupt. The summary of how to control (assert and clear) the interrupt is described in the Interrupt section.

Table 5.6. Interrupt Identification Register

Field	Name	Description	Access	Width	Reset
[7:6]	fifos_en	FIFOs Enabled – fixed value based on the *FIFO Enable* attribute. If FIFO is enabled, fifos_en is set to logic 2'b11 to indicate that both write and read FIFOs are enabled: • *FIFO Enable* is unchecked – Fixed to 2'b00 • *FIFO Enable* is checked – Fixed to 2'b11	RO	2	Refer to the description.
[5:3]	reserved	—	RSVD	3	—

Field	Name	Description	Access	Width	Reset
[2:1]	int_prio	Interrupt Priority – this bit indicates the highest-priority pending interrupt. For example, if all interrupts are asserted, int_prio returns 2'b11 when read. After Receiver Line Status Interrupt is cleared, int_prio becomes 2'b10 to indicate the next highest-priority pending interrupt: • 2'b11 – Receiver Line Status Interrupt • 2'b10 – Received Data Available Interrupt • 2'b01 – Transmitter Holding Register Empty Interrupt • 2'b00 – MODEM Status Interrupt (not yet supported)	RO	2	2'b00
[0]	int_pending	Interrupt Pending – this bit indicates that an interrupt is pending. • 1'b0 – An interrupt is pending, as indicated by int_prio and timeout_int. • 1'b1 – No pending interrupt.	RO	1	1'b1

5.1.5. Line Control Register (LCR)

The Line Control Register configures character length, number of stop bits, and parity bit.

Table 5.7. Line Control Register

Field	Name	Description	Access	Width	Reset
[7]	reserved	—	RSVD	1	1'b0
[6]	break_ctrl_en	Break Control Enable – when break control is enabled, it causes the serial output (txd_o) to go and stay at logic 0, which is interpreted as a long stream of 0 bits by the receiving UART (the break condition). The Break Control Bit acts only on txd_o and does not control the transmitter logic: • 1'b0 – Disables break control • 1'b1 – Enables break control	RW	1	1'b0
[5]	stick_parity_en	Stick Parity. When Bit 3, Bit 4, and Bit 5 of this register are logic 1, the parity bit is transmitted and checked as a logic 0. When Bit 3 and Bit 5 are logic 1 and Bit 4 is logic 0, the parity bit is transmitted and checked as a logic 1. Stick parity is usually used to verify the parity check behavior of an external UART: • 1'b0 – Disables stick parity • 1'b1 – Enables stick parity	RW	1	Refer to the *Enable Stick Parity* attribute.
[4]	even_parity_sel	Even Parity Select – this bit is enabled when parity_en is 1'b1: • 1'b0 – Odd parity. An odd number of logic 1s are transmitted and checked in data character bits. • 1'b1 – Even parity. An even number of logic 1s are transmitted and checked in data character bits.	RW	1	Refer to the *Parity Type* attribute.
[3]	parity_en	Parity Enable – enables the transmission and checking of the parity bit: • 1'b0 – Disables parity • 1'b1 – Enables parity	RW	1	Refer to the *Parity Type* attribute.

Field	Name	Description	Access	Width	Reset
[2]	stop_bit_ctrl	Stop Bit Control – specifies the number of stop bits. The receiver only checks the first stop bit, regardless of the setting: • 1'b0 – One stop bit is generated in the transmitted data. • 1'b1 – If char_len_sel==2'b00, one and a half stop bits are generated. Otherwise, two stop bits are generated.	RW	1	Refer to the *Stop Bits* attribute.
[1:0]	char_len_sel	Character length select – selects a character length: • 2'b00 – 5 bits • 2'b01 – 6 bits • 2'b10 – 7 bits • 2'b11 – 8 bits	RW	2	Refer to the *Serial Data Width* attribute.

5.1.6. Line Status Register (LSR)

The Line Status Register provides status information to the host concerning data transfer. This register is not affected by the Interrupt Enable Register. For example, if Receiver Line Status Interrupt is disabled, an interrupt is not generated, but the host can still read the actual status of the transfer from this register.

Table 5.8. Line Status Control Register

Field	Name	Description	Access	Width	Reset
[7]	reserved	—	RSVD	1	—
[6]	xmitr_empty	Transmitter Empty indicator: • 1'b0 – Indicates if XMIT FIFO or Transmitter Shift Register contains data. • 1'b1 – Indicates if XMIT FIFO and Transmitter Shift Register are both empty.	RO	1	1'b1
[5]	thr_empty	Transmitter Holding Register Empty indicator. In FIFO mode, thr_empty is asserted after the last data in FIFO is sent to the Transmitter Shift Register. This bit negates when at least one byte is written to the XMIT FIFO. In non-FIFO mode, thr_empty is asserted when the Transmitter Holding Register is empty. The thr_empty bit generates an interrupt when IER.thre_int_en is set to logic 1: • 1'b0 – Indicates if XMIT FIFO contains at least one data item. • 1'b1 – Indicates if XMIT FIFO is empty.	RO	1	1'b1
[4]	break_cond	Break Condition indicator. A break condition occurs when the received data input is held in the spacing state (logic 0) for longer than a full word transmission time (the total time of start bit, data bits, parity bit, and a stop bits). The break_cond bit is reset whenever the host reads the contents of the Line Status Register. This error is associated with the particular character in the FIFO to which it applies, and is asserted when its associated character reaches the output end of the FIFO. When a break occurs, only one zero character is loaded into the FIFO. The next character transfer is enabled after rxd_i goes to the marking state and receives the next valid start bit. The break_cond bit generates an interrupt when asserted, if Receiver Line Status Interrupt is enabled (IER.rls_int_en == 1): • 1'b0 – No break condition • 1'b1 – Break condition occurs	RO	1	1'b0

Field	Name	Description	Access	Width	Reset
[3]	framing_err	Framing Error indicator. A framing error occurs when a received character does not have a valid stop bit. This bit asserts whenever the stop bit following the last data bit or parity bit is detected as a logic 0 bit (spacing level). The framing_err bit is reset whenever the host reads the contents of the Line Status Register. This error is associated with the particular character in the FIFO to which it applies, and is asserted when its associated character reaches the output end of the FIFO. The framing_err bit generates an interrupt when asserted, if Receiver Line Status Interrupt is enabled (IER.rls_int_en == 1): • 1'b0 – No framing error • 1'b1 – Framing error occurs	RO	1	1'b0
[2]	parity_err	Parity Error indicator. A parity error occurs when a received character does not have a correct even or odd parity, as selected by LCR.even_parity_sel. The parity_err bit is reset whenever the host reads the contents of the Line Status Register. This error is associated with the particular character in the FIFO to which it applies, and is asserted when its associated character is reaches the output end of the FIFO. The parity_err generates an interrupt when asserted, if Receiver Line Status Interrupt is enabled (IER.rls_int_en == 1): • 1'b0 – No parity error • 1'b1 – Parity error occurs	RO	1	1'b0
[1]	overrun_err	Overrun Error indicator. An overrun error occurs when the RCVR FIFO is full and the next character has been fully received. In this case, the new character is lost. The overrun_err bit generates an interrupt when asserted, if Receiver Line Status Interrupt is enabled (IER.rls_int_en == 1): • 1'b0 – No overrun error • 1'b1 – Overrun error occurs	RO	1	1'b0
[0]	data_rdy	Data Ready Indicator – this bit is asserted when a complete incoming character is received and transferred into the Receiver Buffer Register or the FIFO. This bit is reset to a logic 0 by reading all of the data in the Receiver Buffer Register or the FIFO: • 1'b0 – Receive Buffer Register has no data • 1'b1 – Receive Buffer Register has data	RO	1	1'b0

5.1.7. Divisor Latch Register (DLR_MSB, DLR_LSB)

The UART IP core contains a programmable baud generator that is capable of dividing the reference clock input (clk_i) by divisors of 1 to (2^{16}–1), and producing a 16× clock for driving the internal transmitter and receiver logic. Two 8-bit registers (DLR_MSB and DLR_LSB) store the divisor in a 16-bit binary format. These Divisor Latch Registers must be loaded during initialization to ensure proper operation of the baud generator. Upon loading either of the Divisor Latch Registers, a 16-bit Baud Counter is immediately loaded. These registers can be accessed when LCR.dlab = 1.

Table 5.9. Line Control Register

DLR_MSB Field	Name	Description	Access	Width	Reset
[7:0]	divisor_msb	Upper byte of the Divisor Latch Register.	RW	8	Refer to the *Baud Rate* attribute.

Table 5.10. Line Control Register

DLR_LSB Field	Name	Description	Access	Width	Reset
[7:0]	divisor_lsb	Lower byte of the Divisor Latch Register. The divisor ({divisor_msb, divisor_lsb}) is calculated as follows: ({divisor_msb, divisor_lsb}) = (frequency input) / (baud rate). For example, Table 5.11 shows the values of the Divisor Latch Register for the standard baud rates supported by the UART IP, based on a system clock of 55.296 MHz.	RW	8	Refer to the *Baud Rate* attribute.

Table 5.11. Standard Baud Rates Grid with DLR Values for 55.296 MHz System Clock

Supported Baud Rate Grid	Divisor Latch (divisor_msb, divisor_lsb)
2,400	23,040
4,800	11,520
9,600	5,760
14,400	3,840
19,200	2,880
28,800	1,920
38,400	1,440
56,000	987
57,600	960
115200	480

6. Designing with the IP

This section provides information on how to generate the IP Core using the Lattice Radiant software and how to run simulation and synthesis. For more details on the Lattice Radiant software, refer to the Lattice Radiant Software User Guide.

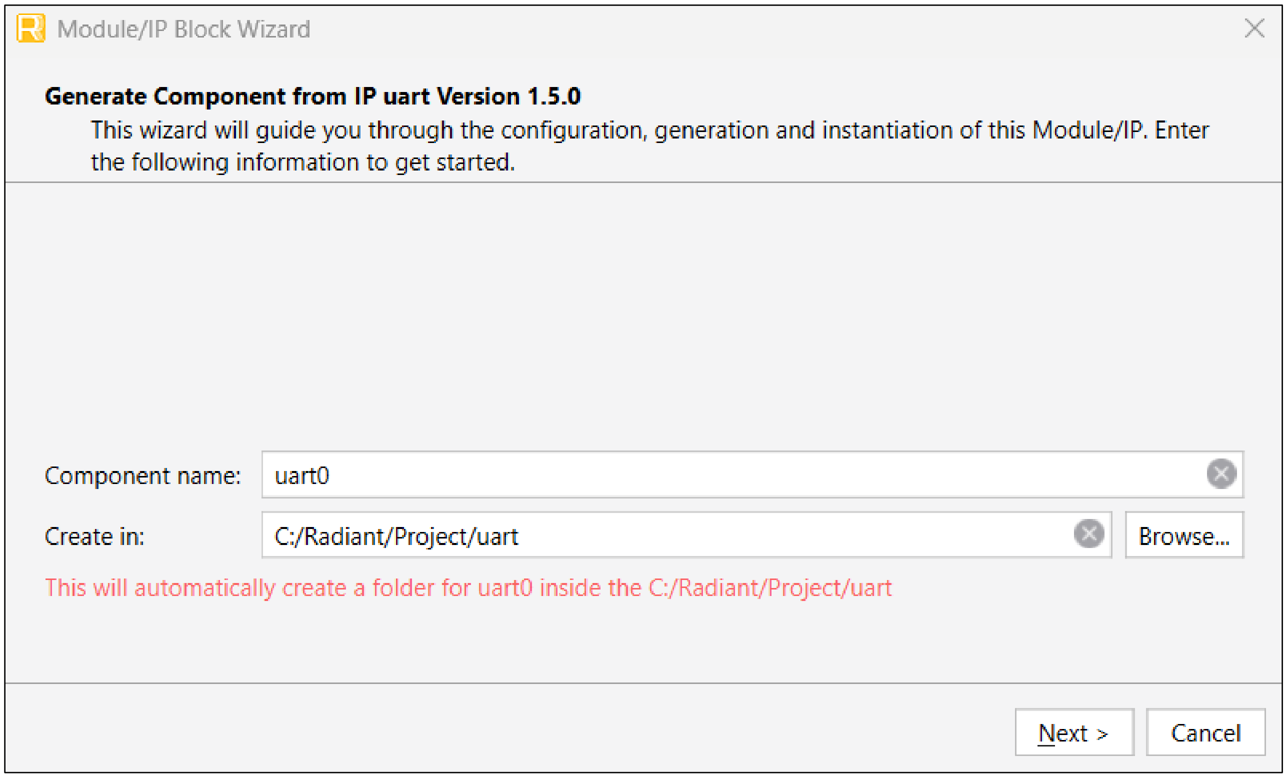
Note: The screenshots provided are for reference only. Details may vary depending on the version of the IP or software being used. If there have been no significant changes to the GUI, a screenshot may reflect an earlier version of the IP.

6.1. Generating and Instantiating the IP

You can use the Lattice Radiant software to generate IP modules and integrate them into the device architecture. The steps below describe how to generate the UART IP in the Lattice Radiant software.

To generate the UART IP:

1. Create a new Lattice Radiant software project or open an existing project.
2. Click the **IP Catalog** button to view the **IP Catalog** pane.
3. On the **IP on Local** tab, double-click **UART** under the **IP, Processors_Controllers_and_Peripherals** category. The **Module/IP Block Wizard** opens.
   Note: If the IP is not available on the **IP on Local** tab, download the IP from the **IP on Server** tab.


Module/IP Block Wizard

Generate Component from IP uart Version 1.5.0

This wizard will guide you through the configuration, generation and instantiation of this Module/IP. Enter the following information to get started.

Component name: uart0

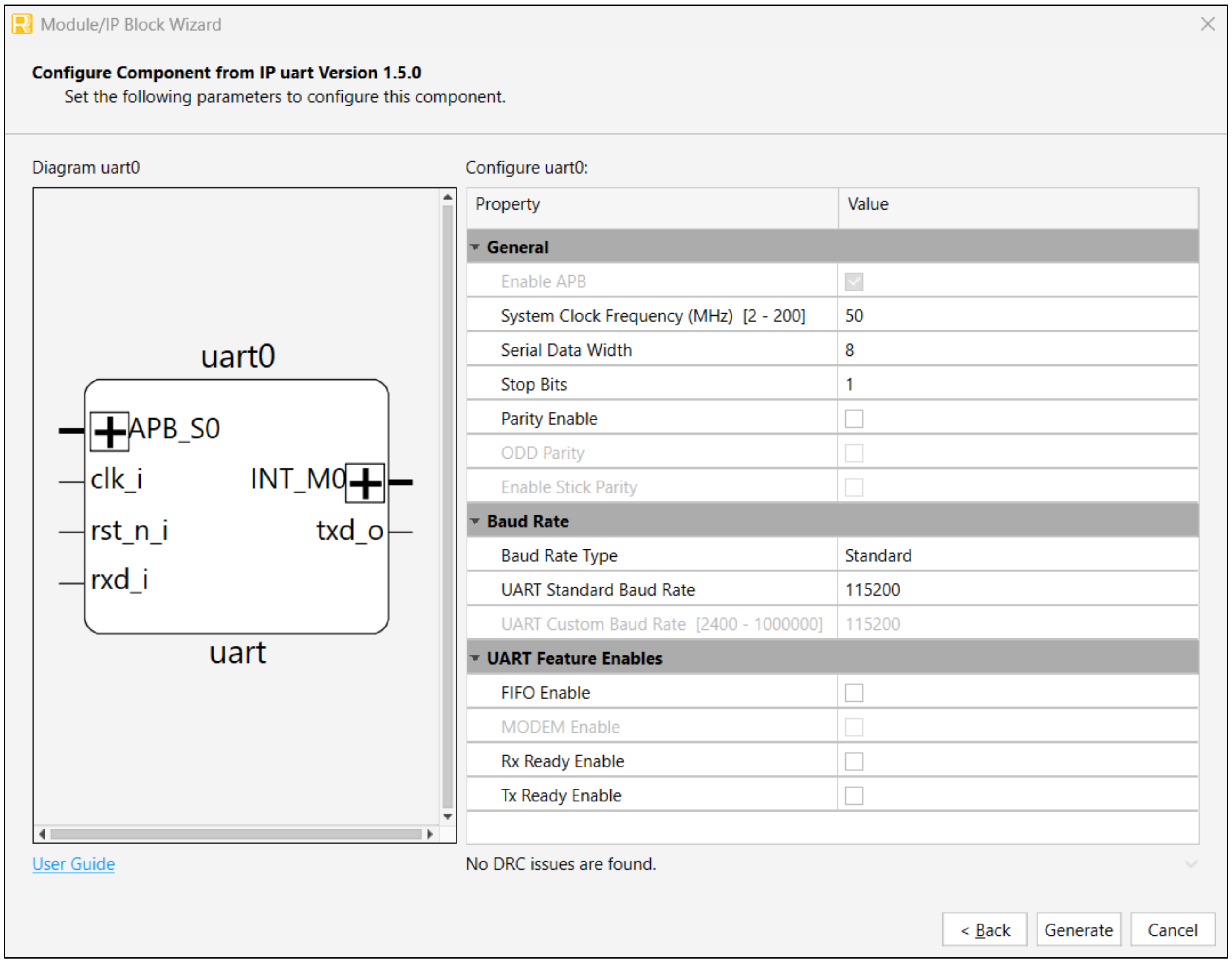
Create in: C:/Radiant/Project/uart

This will automatically create a folder for uart0 inside the C:/Radiant/Project/uart

Figure 6.1. Module/IP Block Wizard

4. Enter values in the **Component name** and **Create in** fields, then click **Next**.

5. Customize the selected UART IP using drop-down lists and check boxes. Figure 6.2 shows an example configuration of the UART IP. For details on the configuration options, refer to the IP Parameter Description section.



Configure Component from IP uart Version 1.5.0

Set the following parameters to configure this component.

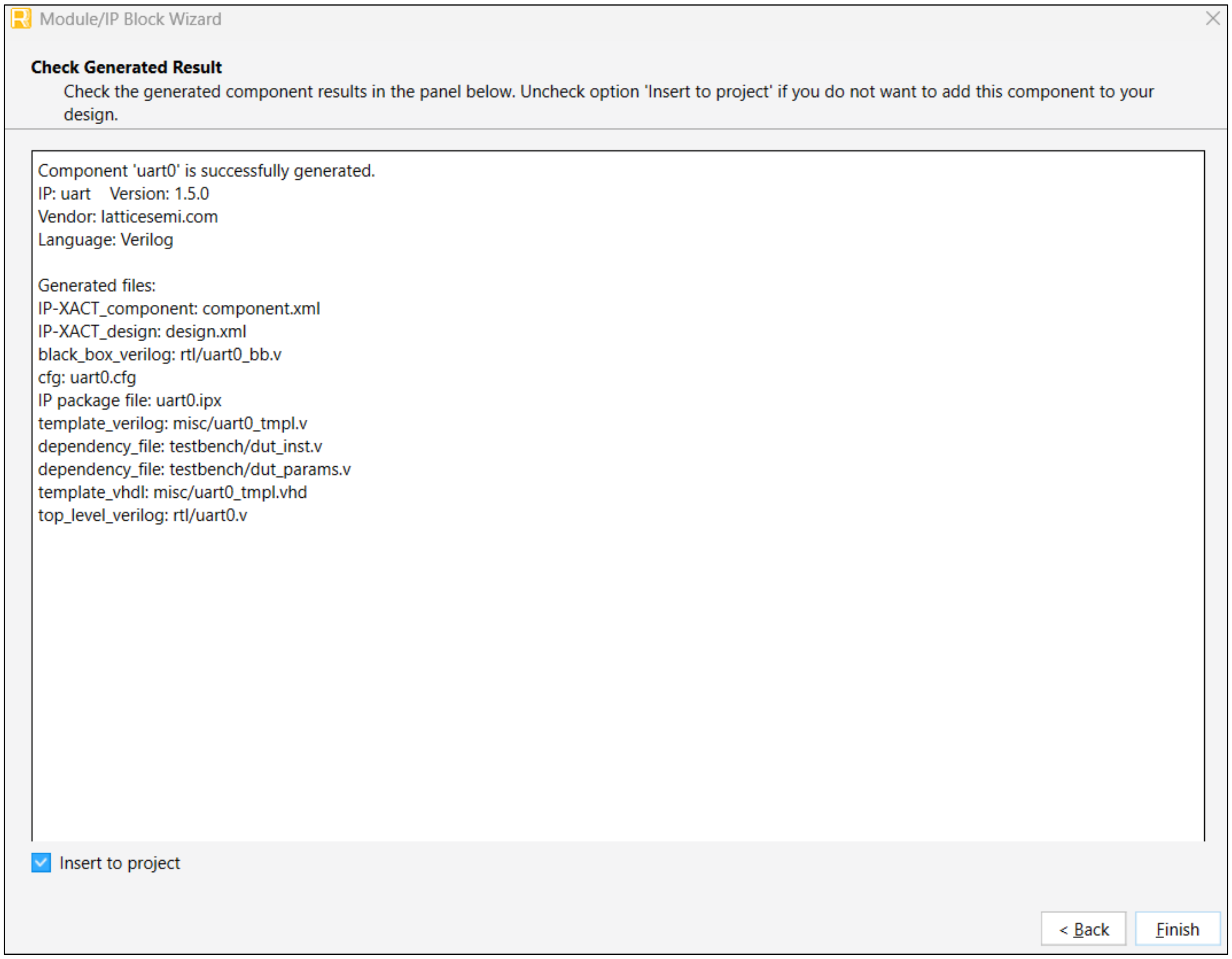
Configure uart0:

Property	Value
General	
Enable APB	✓
System Clock Frequency (MHz) [2 - 200]	50
Serial Data Width	8
Stop Bits	1
Parity Enable	☐
ODD Parity	☐
Enable Stick Parity	☐
Baud Rate	
Baud Rate Type	Standard
UART Standard Baud Rate	115200
UART Custom Baud Rate [2400 - 1000000]	115200
UART Feature Enables	
FIFO Enable	☐
MODEM Enable	☐
Rx Ready Enable	☐
Tx Ready Enable	☐

No DRC issues are found.

Figure 6.2. IP Configuration

6. Click **Generate**. The **Check Generated Result** window opens. This window shows design block messages and results.



Check the generated component results in the panel below. Uncheck option 'Insert to project' if you do not want to add this component to your design.

```
Component 'uart0' is successfully generated.
IP: uart    Version: 1.5.0
Vendor: latticesemi.com
Language: Verilog

Generated files:
IP-XACT_component: component.xml
IP-XACT_design: design.xml
black_box_verilog: rtl/uart0_bb.v
cfg: uart0.cfg
IP package file: uart0.ipx
template_verilog: misc/uart0_tmpl.v
dependency_file: testbench/dut_inst.v
dependency_file: testbench/dut_params.v
template_vhdl: misc/uart0_tmpl.vhd
top_level_verilog: rtl/uart0.v
```

Figure 6.3. Check Generated Result

7. Click **Finish**. All generated files are placed in the directory specified by the **Component name** and **Create in** fields shown in Figure 6.1.

6.1.1. Generated Files and File Structure

The generated UART module package includes the closed-box (*<Component name>_bb.v*) and instance templates (*<Component name>_tmpl.v/vhd*) that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file (*<Component name>.v*) that can be used as an instantiation template for the module is also provided. You may also use this example as the starting template for your top-level design. The generated files are listed in Table 6.1.

Table 6.1. Generated File List

Attribute	Description
<Component name>.ipx	This file contains the information on the files associated to the generated IP.
<Component name>.cfg	This file contains the parameter values used in IP configuration.
component.xml	Contains the ipxact: component information of the IP.
design.xml	Documents the configuration parameters of the IP in IP-XACT 2014 format.
rtl/<Component name>.v	This file provides an example RTL top file that instantiates the module.
rtl/<Component name>_bb.v	This file provides the synthesis closed-box.

Attribute	Description
misc/<Component name>_tmpl.v misc /<Component name>_tmpl.vhd	These files provide instance templates for the module.

6.2. Design Implementation

Completing your design includes additional steps to specify analog properties, pin assignments, and timing and physical constraints. You can add and edit the constraints using the Device Constraint Editor or by manually creating a PDC File.

Post-Synthesis constraint files (.pdc) contain both timing and non-timing constraint.pdc source files for storing logical timing/physical constraints. Constraints that are added using the Device Constraint Editor are saved to the active .pdc file. The active post-synthesis design constraint file is then used as input for post-synthesis processes.

Refer to the relevant sections in the Lattice Radiant Software User Guide for more information on how to create or edit constraints and how to use the Device Constraint Editor.

6.3. Specifying the Strategy

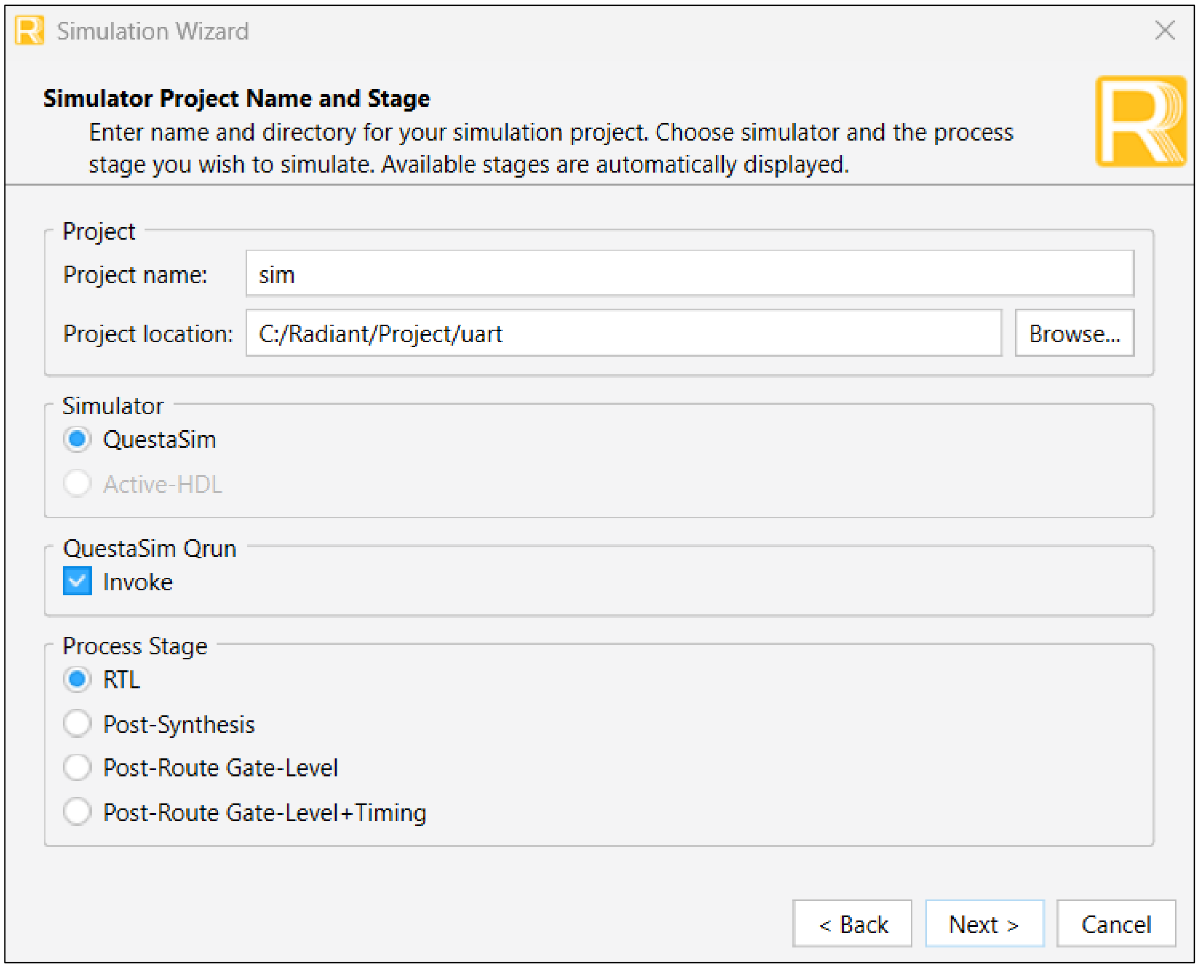
The Radiant software provides two predefined strategies: Area and Timing. The software also enables you to create customized strategies. For details on how to create a new strategy, refer to the Strategies section of the Lattice Radiant Software user guide.

6.4. Running Functional Simulation

You can run functional simulation after the IP is generated.

To run functional simulation:

1. Click the button located on the **Toolbar** to initiate the **Simulation Wizard** shown in Figure 6.4.



Simulation Wizard

Simulator Project Name and Stage

Enter name and directory for your simulation project. Choose simulator and the process stage you wish to simulate. Available stages are automatically displayed.

Project

Project name: sim

Project location: C:/Radiant/Project/uart

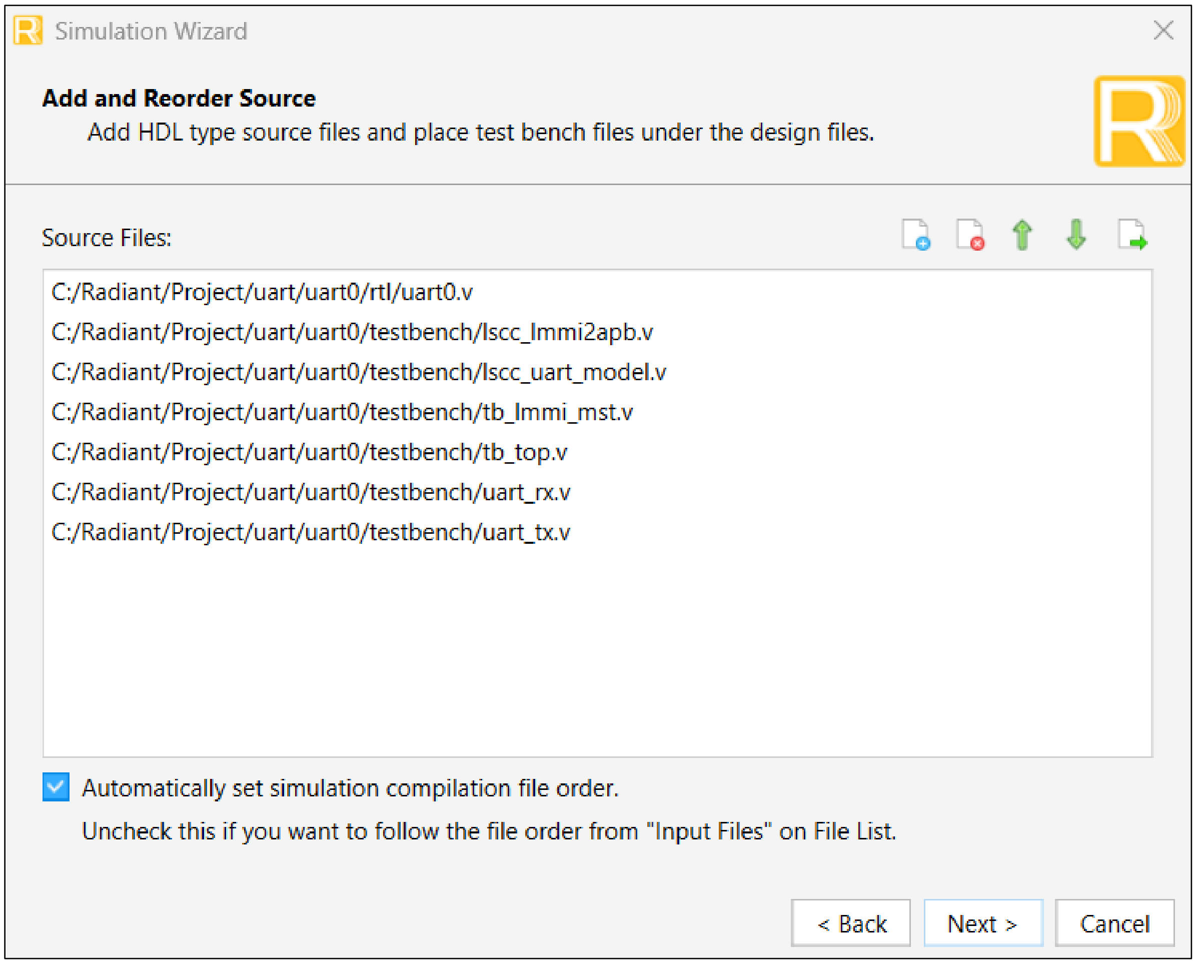
Simulator: [✓ QuestaSim] Active-HDL

QuestaSim Qrun: [✓ Invoke]

Process Stage: [✓ RTL] Post-Synthesis Post-Route Gate-Level Post-Route Gate-Level+Timing

Figure 6.4. Simulation Wizard

2. Click **Next** to open the **Add and Reorder Source** window as shown in Figure 6.5.


Simulation Wizard

Add and Reorder Source
Add HDL type source files and place test bench files under the design files.

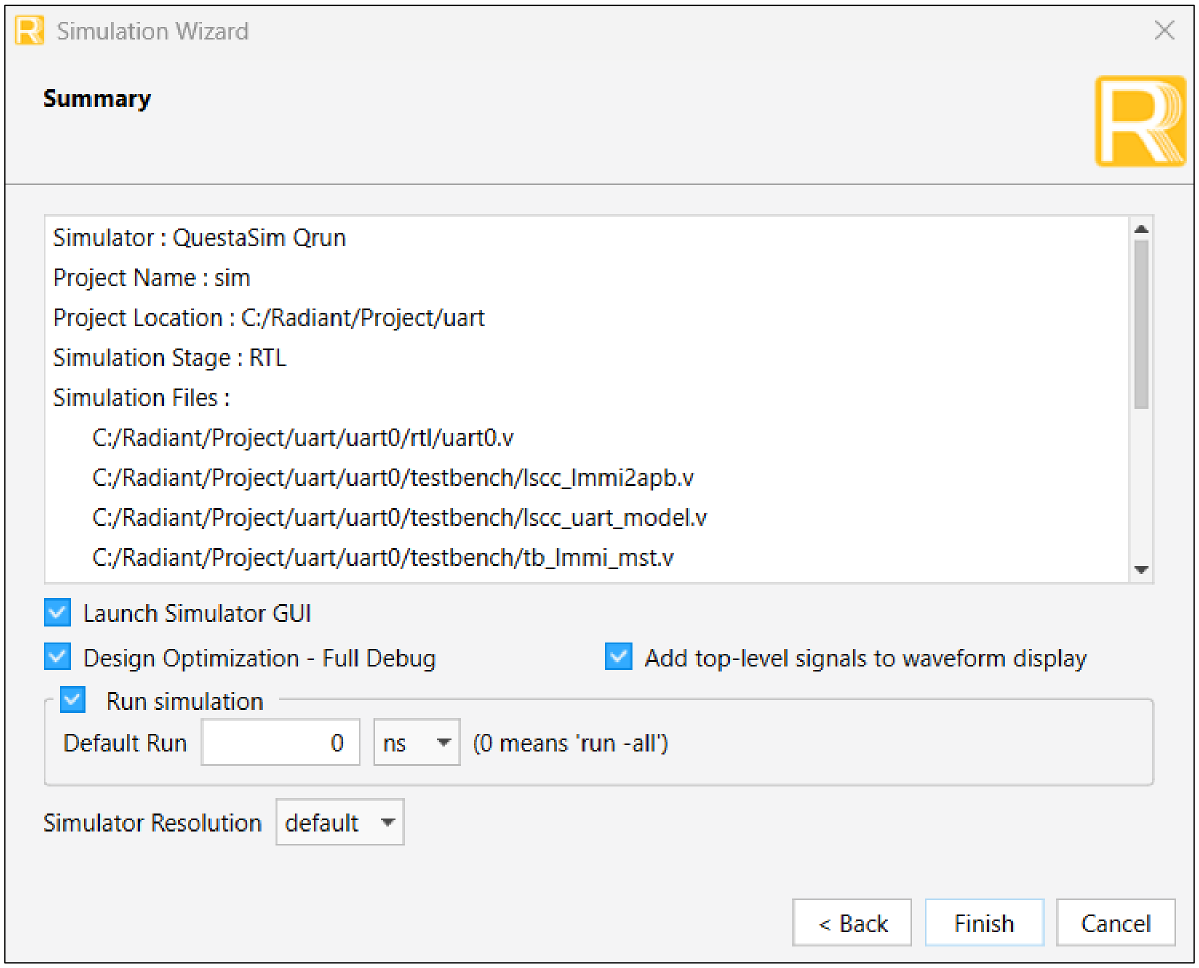
Source Files:

C:/Radiant/Project/uart/uart0/rtl/uart0.v
C:/Radiant/Project/uart/uart0/testbench/lscc_lmmi2apb.v
C:/Radiant/Project/uart/uart0/testbench/lscc_uart_model.v
C:/Radiant/Project/uart/uart0/testbench/tb_lmmi_mst.v
C:/Radiant/Project/uart/uart0/testbench/tb_top.v
C:/Radiant/Project/uart/uart0/testbench/uart_rx.v
C:/Radiant/Project/uart/uart0/testbench/uart_tx.v

[✓] Automatically set simulation compilation file order.
Uncheck this if you want to follow the file order from "Input Files" on File List.

Figure 6.5. Add and Reorder Source

3. Click **Next**. The **Summary** window is shown in Figure 6.6.



Simulator : QuestaSim Qrun
Project Name : sim
Project Location : C:/Radiant/Project/uart
Simulation Stage : RTL
Simulation Files :
C:/Radiant/Project/uart/uart0/rtl/uart0.v
C:/Radiant/Project/uart/uart0/testbench/lscc_lmmi2apb.v
C:/Radiant/Project/uart/uart0/testbench/lscc_uart_model.v
C:/Radiant/Project/uart/uart0/testbench/tb_lmmi_mst.v

[✓] Launch Simulator GUI
[✓] Design Optimization - Full Debug
[✓] Add top-level signals to waveform display
[✓] Run simulation
Default Run 0 ns (0 means 'run -all')
Simulator Resolution default

Figure 6.6. Summary

4. Click **Finish** to run the simulation.

The waveform in Figure 6.7 shows an example simulation result.

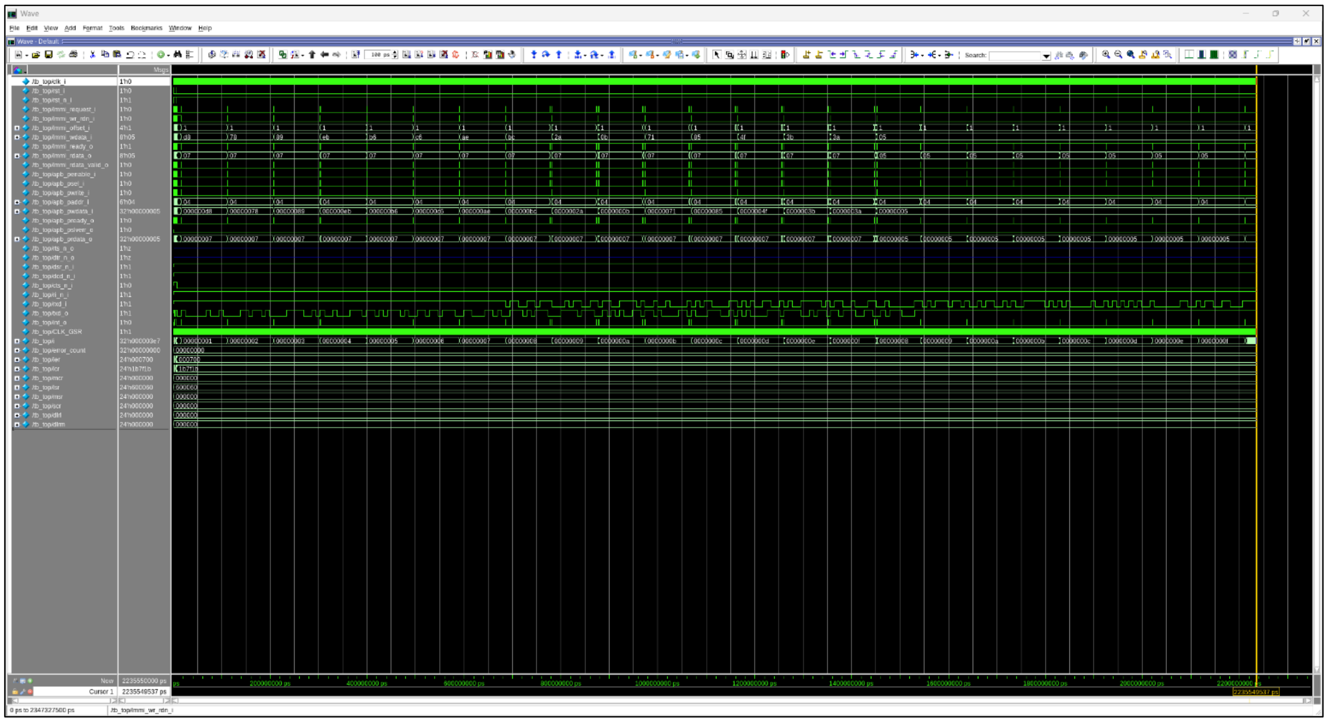
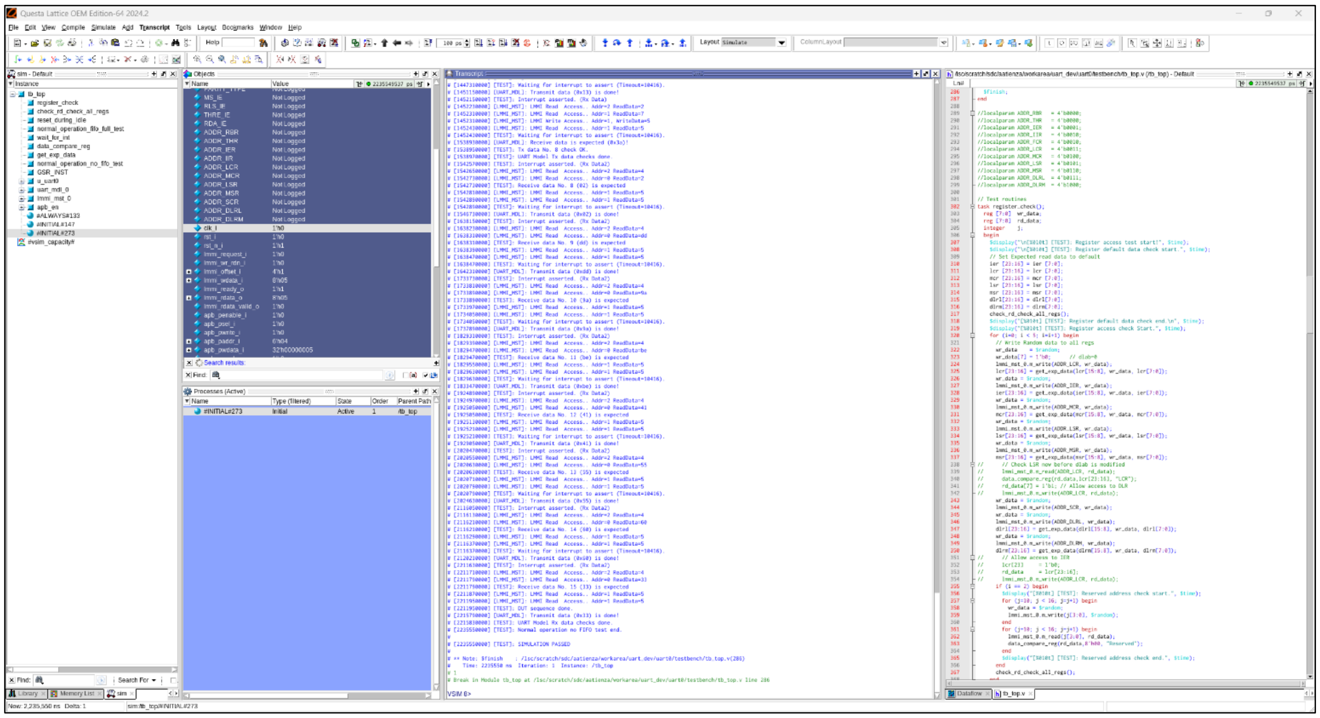


Figure 6.7. Simulation Waveform

6.4.1. Simulation Results

The simulation results log, as shown in Figure 6.8, indicates whether the simulation has passed or failed.



```
# [1447310000] [TEST]: Waiting for interrupt to assert (Timeout=10416).
# [1451150000] [UART_MDL]: Transmit data (0x13) is done!
# [1452150000] [TEST]: Interrupt asserted. (Rx Data)
# [1452230000] [LMMI_MST]: LMMI Read  Access.. Addr=2 ReadData=2
# [1452310000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=7
# [1452310000] [LMMI_MST]: LMMI Write Access.. Addr=1, WriteData=5
# [1452430000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1452430000] [TEST]: Waiting for interrupt to assert (Timeout=10416).
# [1538930000] [UART_MDL]: Receive data is expected (0x3a)!
# [1538950000] [TEST]: Tx data No. 8 check OK.
# [1538970000] [TEST]: UART Model Tx data checks done.
# [1542570000] [TEST]: Interrupt asserted. (Rx Data2)
# [1542650000] [LMMI_MST]: LMMI Read  Access.. Addr=2 ReadData=4
# [1542730000] [LMMI_MST]: LMMI Read  Access.. Addr=0 ReadData=2
# [1542730000] [TEST]: Receive data No. 8 (02) is expected
# [1542810000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1542850000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1542850000] [TEST]: Waiting for interrupt to assert (Timeout=10416).
# [1546730000] [UART_MDL]: Transmit data (0x02) is done!
# [1638150000] [TEST]: Interrupt asserted. (Rx Data2)
# [1638230000] [LMMI_MST]: LMMI Read  Access.. Addr=2 ReadData=4
# [1638310000] [LMMI_MST]: LMMI Read  Access.. Addr=0 ReadData=dd
# [1638310000] [TEST]: Receive data No. 9 (dd) is expected
# [1638350000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1638470000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1638470000] [TEST]: Waiting for interrupt to assert (Timeout=10416).
# [1642310000] [UART_MDL]: Transmit data (0xdd) is done!
# [1733730000] [TEST]: Interrupt asserted. (Rx Data2)
# [1733810000] [LMMI_MST]: LMMI Read  Access.. Addr=2 ReadData=4
# [1733850000] [LMMI_MST]: LMMI Read  Access.. Addr=0 ReadData=9a
# [1733850000] [TEST]: Receive data No. 10 (9a) is expected
# [1733970000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1734050000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1734050000] [TEST]: Waiting for interrupt to assert (Timeout=10416).
# [1737850000] [UART_MDL]: Transmit data (0x9a) is done!
# [1829310000] [TEST]: Interrupt asserted. (Rx Data2)
# [1829350000] [LMMI_MST]: LMMI Read  Access.. Addr=2 ReadData=4
# [1829470000] [LMMI_MST]: LMMI Read  Access.. Addr=0 ReadData=be
# [1829470000] [TEST]: Receive data No. 11 (be) is expected
# [1829550000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1829630000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1829630000] [TEST]: Waiting for interrupt to assert (Timeout=10416).
# [1833470000] [UART_MDL]: Transmit data (0xbe) is done!
# [1924850000] [TEST]: Interrupt asserted. (Rx Data2)
# [1924970000] [LMMI_MST]: LMMI Read  Access.. Addr=2 ReadData=4
# [1925050000] [LMMI_MST]: LMMI Read  Access.. Addr=0 ReadData=41
# [1925050000] [TEST]: Receive data No. 12 (41) is expected
# [1925130000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1925210000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [1925210000] [TEST]: Waiting for interrupt to assert (Timeout=10416).
# [1929050000] [UART_MDL]: Transmit data (0x41) is done!
# [2020470000] [TEST]: Interrupt asserted. (Rx Data2)
# [2020550000] [LMMI_MST]: LMMI Read  Access.. Addr=2 ReadData=4
# [2020630000] [LMMI_MST]: LMMI Read  Access.. Addr=0 ReadData=55
# [2020630000] [TEST]: Receive data No. 13 (55) is expected
# [2020710000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [2020750000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [2020750000] [TEST]: Waiting for interrupt to assert (Timeout=10416).
# [2024630000] [UART_MDL]: Transmit data (0x55) is done!
# [2116050000] [TEST]: Interrupt asserted. (Rx Data2)
# [2116130000] [LMMI_MST]: LMMI Read  Access.. Addr=2 ReadData=4
# [2116210000] [LMMI_MST]: LMMI Read  Access.. Addr=0 ReadData=60
# [2116210000] [TEST]: Receive data No. 14 (60) is expected
# [2116250000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [2116370000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [2116370000] [TEST]: Waiting for interrupt to assert (Timeout=10416).
# [2120210000] [UART_MDL]: Transmit data (0x60) is done!
# [2211630000] [TEST]: Interrupt asserted. (Rx Data2)
# [2211710000] [LMMI_MST]: LMMI Read  Access.. Addr=2 ReadData=4
# [2211750000] [LMMI_MST]: LMMI Read  Access.. Addr=0 ReadData=33
# [2211750000] [TEST]: Receive data No. 15 (33) is expected
# [2211870000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [2211950000] [LMMI_MST]: LMMI Read  Access.. Addr=1 ReadData=5
# [2211950000] [TEST]: DUT sequence done.
# [2215750000] [UART_MDL]: Transmit data (0x33) is done!
# [2215830000] [TEST]: UART Model Rx data checks done.
# [2235550000] [TEST]: Normal operation no FIFO test end.
#
# [2235550000] [TEST]: SIMULATION PASSED
#
# ** Note: $finish    : /lsc/scratch/sdc/aatienza/workarea/uart_dev/uart0/testbench/tb_top.v(286)
#    Time: 2235550 ns  Iteration: 1  Instance: /tb_top
# 1
# Break in Module tb_top at /lsc/scratch/sdc/aatienza/workarea/uart_dev/uart0/testbench/tb_top.v line 286

VSIM 0>
```

```
    $finish;
  end

  //localparam ADDR_RBR   = 4'b0000;
  //localparam ADDR_THR   = 4'b0000;
  //localparam ADDR_IER   = 4'b0001;
  //localparam ADDR_IIR   = 4'b0010;
  //localparam ADDR_FCR   = 4'b0010;
  //localparam ADDR_LCR   = 4'b0011;
  //localparam ADDR_MCR   = 4'b0100;
  //localparam ADDR_LSR   = 4'b0101;
  //localparam ADDR_MSR   = 4'b0110;
  //localparam ADDR_DLRL  = 4'b0111;
  //localparam ADDR_DLRM  = 4'b1000;

  // Test routines
  task register_check();
    reg [7:0]  wr_data;
    reg [7:0]  rd_data;
    integer    j;
    begin
      $display("\n[%010t] [TEST]: Register access test start!", $time);
      $display("\n[%010t] [TEST]: Register default data check start.", $time);
      // Set Expected read data to default
      ier [23:16] = ier [7:0];
      lcr [23:16] = lcr [7:0];
      mcr [23:16] = mcr [7:0];
      lsr [23:16] = lsr [7:0];
      msr [23:16] = msr [7:0];
      dlrl[23:16] = dlrl[7:0];
      dlrm[23:16] = dlrm[7:0];
      check_rd_check_all_regs();
      $display("[%010t] [TEST]: Register default data check end.\n", $time);
      $display("[%010t] [TEST]: Register access check Start.", $time);
      for (i=0; i < 5; i=i+1) begin
        // Write Random data to all regs
        wr_data    = $random;
        wr_data[7] = 1'b0;       // dlab=0
        lmmi_mst_0.m_write(ADDR_LCR, wr_data);
        lcr[23:16] = get_exp_data(lcr[15:8], wr_data, lcr[7:0]);
        wr_data = $random;
        lmmi_mst_0.m_write(ADDR_IER, wr_data);
        ier[23:16] = get_exp_data(ier[15:8], wr_data, ier[7:0]);
        wr_data = $random;
        lmmi_mst_0.m_write(ADDR_MCR, wr_data);
        mcr[23:16] = get_exp_data(mcr[15:8], wr_data, mcr[7:0]);
        wr_data = $random;
        lmmi_mst_0.m_write(ADDR_LSR, wr_data);
        lsr[23:16] = get_exp_data(lsr[15:8], wr_data, lsr[7:0]);
        wr_data = $random;
        lmmi_mst_0.m_write(ADDR_MSR, wr_data);
        msr[23:16] = get_exp_data(msr[15:8], wr_data, msr[7:0]);
  //      // Check LSR now before dlab is modified
  //      lmmi_mst_0.m_read(ADDR_LCR, rd_data);
  //      data_compare_reg(rd_data,lcr[23:16], "LCR");
  //      rd_data[7] = 1'b1; // Allow access to DLR
  //      lmmi_mst_0.m_write(ADDR_LCR, rd_data);
        wr_data = $random;
        lmmi_mst_0.m_write(ADDR_SCR, wr_data);
        wr_data = $random;
        lmmi_mst_0.m_write(ADDR_DLRL, wr_data);
        dlrl[23:16] = get_exp_data(dlrl[15:8], wr_data, dlrl[7:0]);
        wr_data = $random;
        lmmi_mst_0.m_write(ADDR_DLRM, wr_data);
        dlrm[23:16] = get_exp_data(dlrm[15:8], wr_data, dlrm[7:0]);
  //      // Allow access to IER
  //      lcr[23]     = 1'b0;
  //      rd_data     = lcr[23:16];
  //      lmmi_mst_0.m_write(ADDR_LCR, rd_data);
        if (i == 2) begin
          $display("[%010t] [TEST]: Reserved address check start.", $time);
          for (j=10; j < 16; j=j+1) begin
            wr_data = $random;
            lmmi_mst_0.m_write(j[3:0], $random);
          end
          for (j=10; j < 16; j=j+1) begin
            lmmi_mst_0.m_read(j[3:0], rd_data);
            data_compare_reg(rd_data,8'h00, "Reserved");
          end
          $display("[%010t] [TEST]: Reserved address check end.", $time);
        end
        check_rd_check_all_regs();
```

Figure 6.8. Simulation Result Log

7. Design Considerations

7.1.1. Initialization

The following UART register fields should be set properly before performing a UART transaction:

- Line Control Register – even_parity_sel, parity_en, stop_bit_ctrl, char_len_sel
- Divisor Latch Registers – divisor_msb, divisor_lsb

Match these settings to the corresponding settings in the communicating UART for the serial transaction to be successful. Note that the reset values of these register fields are configurable during IP generation. Thus, in some applications, the initialization step is not necessary when attributes are properly set.

7.1.2. Transmit Operation

The steps for transmitting character data through the UART IP core are shown below. This assumes that the IP core is not currently performing a transmit operation, or at least that the XMIT FIFO is empty.

Transmit Operation – Interrupt Mode

1. Write data to THR. In FIFO mode, you can write up to 16 characters.
2. Set IER.thre_int_en = 1’b1 to enable Transmit Holding Register Empty interrupt.
3. Wait for Transmit Holding Register Empty interrupt to assert:
   a. Wait for interrupt assertion and check that IIR[3:0] = 4’b0010.
4. If you need to send more characters, repeat steps 1–3 until all characters are sent.
5. When done using the interrupt, set IER.thre_int_en = 1’b0 to disable it.

Transmit Operation – Polling Mode

1. Write data to THR. It is recommended not to enable FIFO to polling mode to save resources.
2. Read LSR until the thr_empty bit asserts.
3. If you need to send more characters, repeat steps 1 and 2 until all characters are sent.

7.1.3. Receive Operation

The steps for receiving character data through the UART IP core are shown below. This assumes that the IP core is not currently performing a receive operation.

Receive Operation – Interrupt Mode

1. Enable the following interrupts:
   a. Received Data Available Interrupt (IER.rda_int_en = 1’b1) – notifies the host that a data is received.
   b. Receiver Line Status interrupt (IER.rls_int_en = 1’b1) – notifies the host of receive status conditions such as errors or break conditions.
2. Wait for interrupt assertion and check that IIR[2:0] = 3’b100 (Receive Data Available). If the Receiver Line Status Interrupt asserts (IIR[2:0] = 3’b110), read the LSR to determine the cause.
3. If the Receiver Line Status Interrupt does not occur, read the character data from RBR:
   a. If Receive Data Available Interrupt occurs, read data from RBR.
   b. If Character Timeout Interrupt occurs, read LSR. If LSR.data_rdy = 1’b1, read RBR.
4. Repeat steps 2–3 until all expected data is received.

Receive Operation – Polling Mode

1. Read LSR until the data_rdy bit asserts. Also, check that no error status bits are asserted.
2. Read RBR if there are no errors.
3. If you need to receive more characters, repeat steps 1 and 2 until all characters are received.

Appendix A. Resource Utilization

Table A.1 shows a sample resource utilization of the UART IP on MachXO3D devices using the Lattice Synthesis Engine in the Lattice Diamond software.

Table A.1. MachXO3D Devices Resource Utilization

Configuration	Clk Fmax (MHz)[1]	PFU Registers	LUTs	EBRs
Default	107.898	150	232	0
FIFO Enable is Checked, Others = Default	102.323	267	334	0

Note:

1. Fmax is generated when the FPGA design contains only UART IP core with a target frequency of 50 MHz. These values may be reduced when user logic is added to the FPGA design.

Table A.2 shows a sample resource utilization of the UART IP on the LFD2NX-9-7MG121C device using the Lattice Synthesis Engine in the Lattice Radiant software version 2024.2.

Table A.2. LFD2NX-9-7MG121C Device Resource Utilization

Configuration	Clk Fmax (MHz)[1]	PFU Registers	LUTs	EBRs
Default	185.701	150	418	0
FIFO Enable is Checked, Others = Default	157.381	611	810	0

Note:

1. Fmax is generated when the FPGA design contains only UART IP core with a target frequency of 50 MHz. These values may be reduced when user logic is added to the FPGA design.

Table A.3 shows a sample resource utilization of the UART IP on the LFD2NX-17-7MG121C device using the Lattice Synthesis Engine in the Lattice Radiant software version 2024.2.

Table A.3. LFD2NX-17-7MG121C Device Resource Utilization

Configuration	Clk Fmax (MHz)[1]	PFU Registers	LUTs	EBRs
Default	185.701	150	418	0
FIFO Enable is Checked, Others = Default	157.381	611	810	0

Note:

1. Fmax is generated when the FPGA design contains only UART IP core with a target frequency of 50 MHz. These values may be reduced when user logic is added to the FPGA design.

Table A.4 shows a sample resource utilization of the UART IP on the LFD2NX-28-7MG121C device using the Lattice Synthesis Engine in the Lattice Radiant software version 2024.2.

Table A.4. LFD2NX-28-7MG121C Device Resource Utilization

Configuration	Clk Fmax (MHz)[1]	PFU Registers	LUTs	EBRs
Default	169.005	150	418	0
FIFO Enable is Checked, Others = Default	153.163	611	810	0

Note:

1. Fmax is generated when the FPGA design contains only UART IP core with a target frequency of 50 MHz. These values may be reduced when user logic is added to the FPGA design.

Table A.5 shows a sample resource utilization of the UART IP on the LFD2NX-40-7MG121C device using the Lattice Synthesis Engine in the Lattice Radiant software version 2024.2.

Table A.5. LFD2NX-40-7MG121C Device Resource Utilization

Configuration	Clk Fmax (MHz)[1]	PFU Registers	LUTs	EBRs
Default	169.005	150	418	0
FIFO Enable is Checked, Others = Default	153.163	611	810	0

Note:

1. Fmax is generated when the FPGA design contains only UART IP core with a target frequency of 50 MHz. These values may be reduced when user logic is added to the FPGA design.

Table A.6 shows a sample resource utilization of the UART IP on the LN2-CT-20-1CBG484C device using the Lattice Synthesis Engine in the Lattice Radiant software version 2024.2.

Table A.6. LN2-CT-20-1CBG484C Device Resource Utilization

Configuration	Clk Fmax (MHz)[1]	PFU Registers	LUTs	EBRs
Default	250.000	151	366	0
FIFO Enable is Checked, Others = Default	250.000	611	753	0

Note:

1. Fmax is generated when the FPGA design contains only UART IP core with a target frequency of 50 MHz. These values may be reduced when user logic is added to the FPGA design.

Table A.7 shows a sample resource utilization of the UART IP on the LFMXO4-110HC-5BBG256I device using the Lattice Synthesis Engine in the Lattice Radiant software version 2025.2.

Table A.7. LFMXO4-110HC-5BBG256I Device Resource Utilization

Configuration	Clk Fmax (MHz)[1]	PFU Registers	LUTs	EBRs
Default	126.438	157	319	0
FIFO Enable is Checked, Others = Default	101.020	582	729	0

Note:

1. Fmax is generated when the FPGA design contains only UART IP core with a target frequency of 50 MHz. These values may be reduced when user logic is added to the FPGA design.

References

- UART IP Release Notes (FPGA-RN-02023)
- UART IP Core web page
- Certus-N2 web page
- Certus-NX web page
- CrossLink-NX web page
- Mach-NX web page
- MachXO2 web page
- MachXO3 web page
- MachXO3D web page
- MachXO4 web page
- Lattice Diamond Software web page
- Lattice Propel Design Environment web page
- Lattice Radiant Software web page
- Lattice Solutions IP Cores web page
- Lattice Solutions Reference Designs web page
- Lattice Insights web page for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Note: In some instances, the IP may be updated without changes to the user guide. The user guide may reflect an earlier IP version but remains fully compatible with the later IP version. Refer to the IP Release Notes for the latest updates.

Revision 1.6, IP v1.5.0, December 2025

Section	Change Summary
All	• Updated the IP version information on the cover page. • Added a note on the IP version in the *Quick Facts* and *Revision History* sections. • Made editorial fixes.
Abbreviations in This Document	• Replaced *acronyms* with *abbreviations* in this section. • Removed *LMMI*. • Added *AMBA*, *CPU*, *DLR*, *EBR*, *FPGA*, *GUI*, *IER*, *IIR*, *IP*, *LCR*, *LSR*, *LSE*, *LUT*, *PFU*, *RBR*, *RO*, *RSR*, *RSVD*, *RW*, *Rx*, *THR*, *TSR*, *Tx*, and *WO*.
Introduction	• Updated the following sections: • Introduction • Features • Licensing and Ordering Information • Naming Conventions • Added the following sections: • Overview of the IP • Quick Facts • IP Support Summary • Minimum Device Requirements
Functional Descriptions	• Renamed the previous *Overview* section to *IP Architecture Overview* and updated its content. • Added the following sections: • Clocking • Reset • User Interfaces • Updated the following sections: • All Operation Details subsections • Data Format • Timing Diagrams
IP Parameter Description	Moved the content from previous *Attributes Summary* subsection to this new section and updated its content.
Signal Description	Moved the content from previous *Signal Description* subsection to this new section and updated its content.
Register Description	Moved the content from previous *Register Description* subsection to this new section and updated its content.
Designing with the IP	Added this section.
Design Considerations	Moved the content from previous *Programming Flow* subsection to this new section and updated its content.
Resource Utilization	Added resource utilizations for the Lattice Radiant software version 2025.2.
References	Added the *MachXO4*, *UART IP Core*, and *Lattice Solutions Reference Designs* web pages.

Revision 1.5, IP v1.4.0, February 2025

Section	Change Summary
Introduction	Updated the *configurable Baud Rate support* for *Custom mode* in the Features section.
Functional Descriptions	• Updated the *Width* values for *reserved* in Table 2.8. Interrupt Enable Register and Table 2.9. Interrupt Identification Register. • Updated the *Width* value for *char_len_sel* in Table 2.10. Line Control Register.

Revision 1.4, IP v1.4.0, December 2024

Section	Change Summary
All	Added the IP version information on the cover page.
Introduction	Added the *Certus-N2* device to Table 1.1. FPGA Software for IP Configuration, Generation, and Implementation.
Functional Descriptions	Updated the instances of *Slave* to *Target* for *APB Interface* in Table 2.1. UART IP Core Signal Description.
Resource Utilization	Added resource utilizations for the Lattice Radiant software version 2024.2 and made editorial fixes.
References	Added the *Certus-N2* web page, *Lattice Diamond Software* web page, *Lattice Propel Software* web page, *Lattice Radiant Software* web page, *Lattice Solutions IP Cores* web page, and *UART IP Release Notes (FPGA-RN-02023)*.

Revision 1.3, June 2024

Section	Change Summary
All	• Updated the document title from UART IP Core - Lattice Propel Builder to UART IP. • Made editorial fixes.
Introduction	• Updated the *Baud Rate* value range from *1-999999* to *240-1000000* in Features section. • Added the Licensing Information section.
References	Updated this section.
Technical Support Assistance	Updated this section.

Revision 1.2, April 2021

Section	Change Summary
Introduction	Updated Table 1.1 to add support for MachXO2 and MachXO3.
References	Added references to MachXO2 and MachXO3.

Revision 1.1, December 2020

Section	Change Summary
Introduction	Modified second paragraph and added Table 1.1. FPGA Software for IP Configuration, Generation, and Implementation.
References	Updated this section.

Revision 1.0, May 2020

Section	Change Summary
All	Initial release.