HB0095
Handbook
CoreUART v5.7

Microsemi
a Microchip company

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 14.0

Added PolarFire® SoC support.

1.2 Revision 13.0

Added PolarFire® support.

1.3 Revision 12.0

The following is a summary of the changes made in this revision.

- Updated core version from 5.5 to 5.6.
- Added PolarFire device values in Table 1 and Table 2.

1.4 Revision 11.0

The following is a summary of the changes made in this revision.

- Updated core version from 5.4 to 5.5.
- Added RTG4 device values in Table 1 and Table 2.

1.5 Revision 10.0

The following is a summary of the changes made in this revision.

- Formatted the document as per the new HB specifications.
- Added a note in Table 1.
- Updated “Baud Rate” section.
- Updated Table 4.

1.6 Revision 9.0

The following is a summary of the changes made in this revision.

- Added RTG4 information to Supported Families section, FAMILY parameter and to all Device Utilization tables.
- Changed core version from 5.3 to 5.4 in numerous places.

1.7 Revision 8.0

Updated Note 2 in “Framing Error” section and Note 2 in “Framing Error in Legacy Mode” section (SAR 56623).

1.8 Revision 7.0

Added Note 3 (SAR 56293).

1.9 Revision 6.0

Added Note 2 (SAR 55499).

1.10 Revision 5.0

The following is a summary of the changes made in this revision.

- The “Core Version” section was updated to v5.3. IGLOO2 was added to the “Supported Families” section.
- IGLOO2 was added in Table 5.
- Figure 5 was updated (SAR 49707).

1.11 Revision 4.0

The following is a summary of the changes made in this revision.

- The “Core Version” was updated to v5.2. SmartFusion2 was added to the “Supported Families” section (SAR 37391).
- The FIFO depth for SmartFusion2 devices was added to the second note for Table 1 • CoreUART Utilization in FIFO Mode (SAR 37391).
- SmartFusion2 was added to Table 5 • CoreUART Configurable Options (SAR 37391).

1.12 Revision 3.0

The following is a summary of the changes made in this revision.

- The “Core Version” was updated to v5.1.
- BAUD_VAL_FRACTION was added to the programmable inputs for CoreUART in the “Programmable Options” section. The “Baud Value Precision” section is new and additional information and settings were added to the “Baud Rate” section (SAR 37390).
- Figure 10 • Smart Design CoreUART Configuration Window was replaced (SAR 37390).
- BAUD_VAL_FRACTION [2:0] was added to Table 4 • CoreUART Signals and BAUD_VAL_FRCTN_EN was added to Table 5 • CoreUART Configurable Options (SAR 37390).

1.13 Revision 2.0

The following is a summary of the changes made in this revision.

- The core version was updated to v4.2.
- SmartFusion and ProASIC3E were added to the “Supported Families” section.
- SmartFusion was added to Table 1 • CoreUART Utilization in FIFO Mode and Table 2 • CoreUART Utilization in Normal Mode.
- EQ 2 is new.
- The “Tool Flows” chapter was rewritten. The “Testbench Operation” chapter was deleted because its relevant content was incorporated into the “Tool Flows” chapter.
- Signal names were made all upper case. The second note for Table 4 • CoreUART Signals was revised to add, “FRAMING_ERR should be treated similarly, when RX_FIFO = 1."
- SmartFusion was added to Table 5 • CoreUART Configurable Options.

1.14 Revision 1.0

The following is a summary of the changes made in this revision.

- Added framing error (FRAMING_ERR signal) support.
- Updated the “Tool Flows” chapter with SmartDesign flow.
- Updated Table 4 • CoreUART Signals to include FRAMING_ERR signal.
- Updated Table 5 • CoreUART Configurable Options with new configurable options.
- Added Framing Error and Framing Error in Legacy Mode timing diagrams.

1 Preface

1.1 About this Document

This handbook provides details about the CoreUART IP core and how to use it.

1.2 Intended Audience

Designers using Libero® System-on-Chip (SoC) or Libero Integrated Design Environment (IDE).

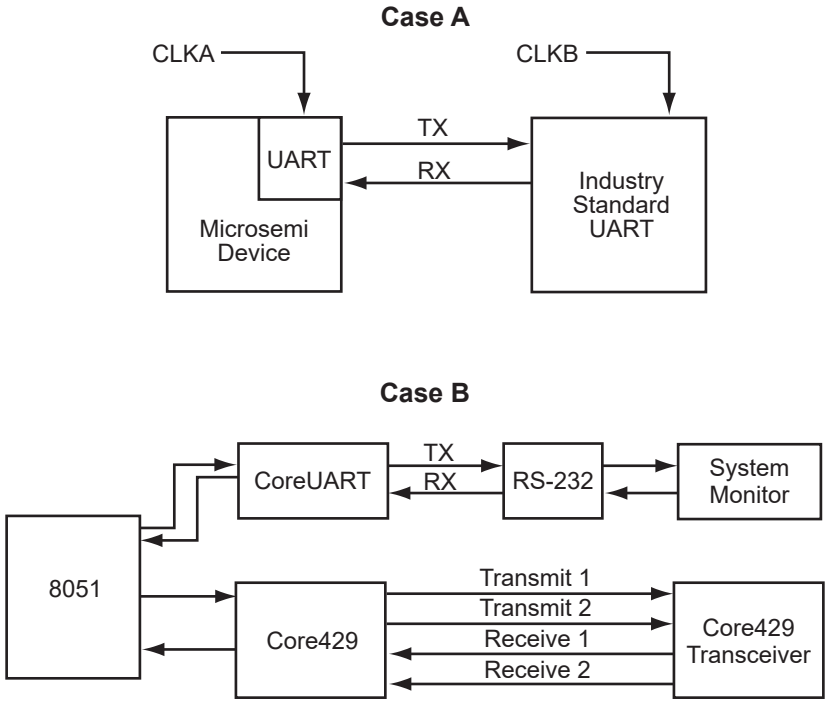
2 Introduction

2.1 Overview

CoreUART is a serial communication controller with a flexible serial data interface that is intended primarily for embedded systems. CoreUART can be used to interface directly to industry standard UARTs. CoreUART is intentionally a subset of full UART capability to make the function cost-effective in a programmable device. Figure 1 illustrates the various usages of CoreUART.

Case A in Figure 1 represents the interface to an industry standard UART, such as an 8251 or a 16550. In Case B, CoreUART is transferring data from the 8051 to the system monitor through the RS-232 interface and vice versa.

Figure 1 • **System Block Diagram Depicting CoreUART Usage**



2.2 Key Features

CoreUART is a highly configurable core and has the following features:

- Asynchronous mode to interface with industry standard UART
- Optional Transmit and Receive FIFOs

2.3 Core Version

This handbook applies to CoreUART v5.7. The release notes provided with the core list known discrepancies between this handbook and the core release associated with the release notes.

2.4 Supported Families

- PolarFire® SoC
- PolarFire®
- RTG4™
- IGLOO®2
- SmartFusion®2
- SmartFusion®
- IGLOO®
- IGLOOe
- IGLOO PLUS
- ProASIC®3
- ProASIC3E
- ProASIC3L
- Fusion
- ProASICPLUS®
- Axcelerator®
- RTAX-S™
- SX-A
- RTSX-S

2.5 Device Utilization and Performance

Utilization statistics for targeted devices are listed in Table 1 and Table 2, page 6.

Table 1 • **CoreUART Utilization in FIFO Mode**

	Cells or Tiles			Memory	Utilization		Performance
Family	**Sequential**	**Combinatorial**	**Total**	**Blocks**	**Device**	**Total**	**MHz**
IGLOO IGLOOe IGLOO PLUS	116	192	308	2	AGL600V5	2%	71
ProASIC3 ProASIC3E ProASIC3L	116	192	308	2	A3P600	5%	128
SmartFusion	116	192	308	2	A2F500M3F	7%	118
SmartFusion2	217	249	466	2	M2S150T	0.35%	250
Fusion	116	192	308	2	AFS600	2%	127
ProASICPLUS	118	281	399	2	APA600	13%	68
Axcelerator	171	215	386	2	AX250	9%	194
RTAX-S	195	199	394	2	RTAX250S	9%	153
SX-A	430	309	739	0	A54SX16A	51%	96
RTSX-S	432	308	740	0	RT54SX32S	26%	62
IGLOO2	217	249	466	2	M2GL150T	0.35%	250
RTG4	238	322	560	2	RT4G150	0.37%	154
PolarFire	237	244	481	2	PA5M500	0.09	258

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

Note:

1. CoreUART supports all standard baud rates, including 110, 300, 1,200, 2,400, 4,800, 9,600, 19,200, 38,400, 57,600, 115,200, 230,400, 460,800, and 921,600 baud.
2. The depth of the FIFO for SX-A and RTSX-S families is 16. The depth of the FIFO for SmartFusion2/IGLOO2 devices is 128. For the other families, the depth of the FIFO is 256.

3. CoreUART does not support baud value of zero.

Table 2 • **CoreUART Utilization in Normal Mode**

Family	Cells or Tiles Sequential	Combinatorial	Total	Memory Blocks	Utilization Device	Total	Performance MHz
IGLOO IGLOOe IGLOO PLUS	84	167	251	0	AGL600	2%	116
ProASIC3 ProASIC3E ProASIC3L	84	167	251	0	A3P600	4%	198
SmartFusion	84	167	251	0	A2F500M3F	6%	200
SmartFusion2	87	113	200	0	M2S150T	0.2%	250
Fusion	84	167	251	0	AFS600	2%	197
ProASICPLUS	85	254	339	0	APA600	11%	89
Axcelerator	86	108	194	0	AX500	5%	185
RTAX-S	86	108	194	0	RTAX250S	5%	138
SX-A	82	93	750	0	A54SX16SA	12%	138
RTSX-S	80	92	172	0	RT54SX32S	6%	87
IGLOO2	87	113	200	0	M2GL150T	0.2%	250
RTG4	90	149	239	0	RT4G150	0.16%	174
PolarFire	90	126	216	2	PA5M500	0.05	371

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

Note: CoreUART supports all standard baud rates, including 110, 300, 1,200, 2,400, 4,800, 9,600, 19,200, 38,400, 57,600, 115,200, 230,400, 460,800, and 921,600 baud.

3 Functional Block Description

Figure 2 shows the block diagram of the CoreUART normal mode functionality. Figure 3, page 8 shows the block diagram of CoreUART with FIFO mode functionality. The baud generator creates a divided down clock enable that correctly paces the transmit and receive state machines.

The function of the receive and transmit state machines is affected by the control inputs BIT8, PARITY_EN, and ODD_N_EVEN. These signals indicate to the state machines how many bits should be transmitted. In addition, the signals suggest the type of parity and whether parity should be generated or checked. The activity of the state machines is paced by the outputs of the baud generator.

To transmit data, it is first loaded into the transmit data buffer in normal mode, and into the transmit FIFO in FIFO mode. Data can be loaded into the buffer until the TXRDY signal is driven inactive. The transmit state machine will immediately begin to transmit data and will continue transmission until the data buffer is empty in normal mode, and until the transmit FIFO is empty in FIFO mode. The state machine first transmits a START bit, followed by the data (LSB first), then the parity (optional), and finally the STOP bit. The data buffer is double-buffered in normal mode, so there is no loading latency.

The receive state machine monitors the activity of the RX signal. Once a START bit is detected, the receive state machine begins to store the data in the receive buffer in normal mode and the receive FIFO in FIFO mode. When the transaction is complete, the RXRDY signal indicates that valid data is available. Parity errors are reported on the PARITY_ERR signal (if enabled), and data overrun conditions are reported on the OVERFLOW signal. Framing errors are reported on the FRAMING_ERR signal. A framing error is defined as a missing stop bit detected by the UART receiver.

Figure 2 • **Block Diagram of CoreUART Normal Functionality**

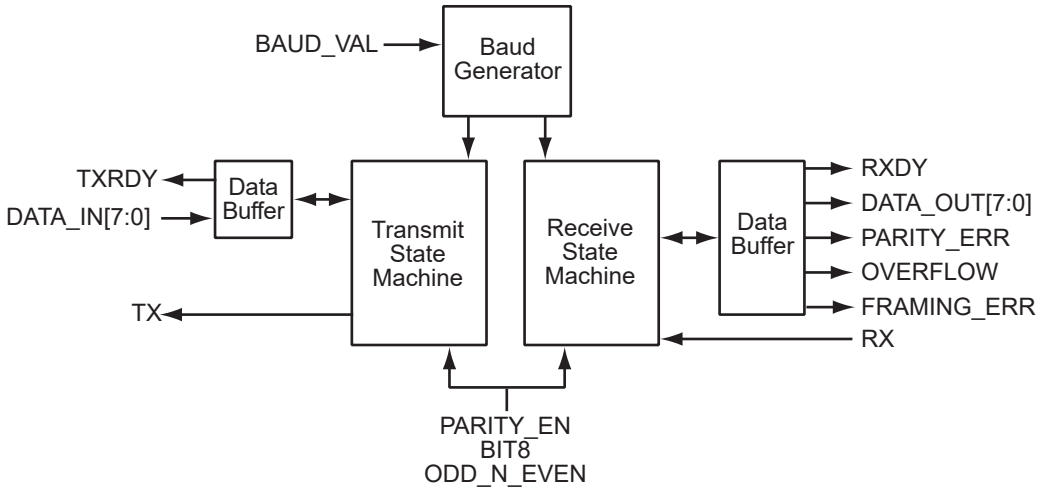
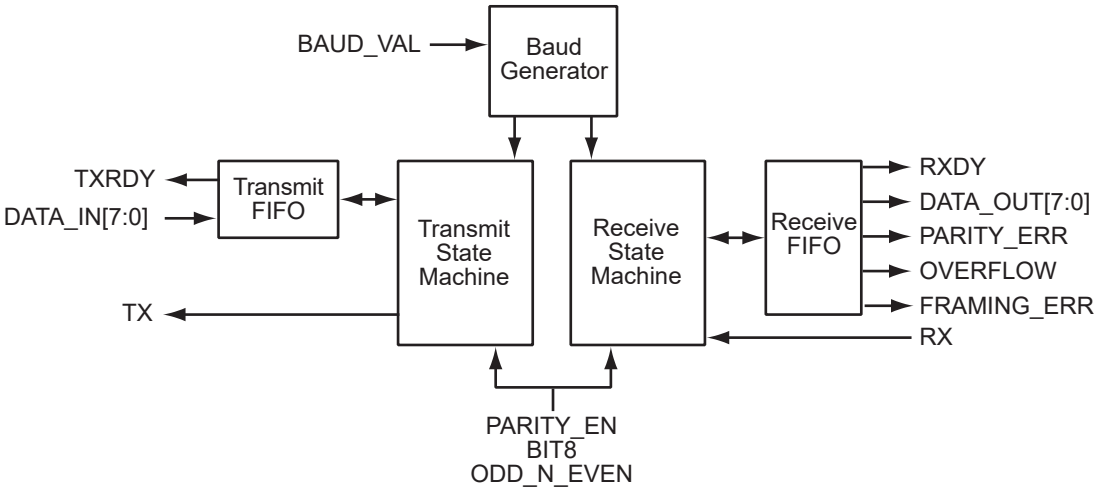


Figure 3 • **Block Diagram of CoreUART with FIFO Functionality**



3.1 Programmable Options

There are five programmable inputs to CoreUART: BAUD_VAL (baud rate), BAUD_VAL_FRACTION (fraction part of baud value), BIT8 (number of data bits), PARITY_EN (parity enable), and ODD_N_EVEN (odd or even parity).

3.1.1 Number of Data Bits

The input BIT8 is used to define the number of valid data bits in the serial bitstream. The most significant bit is a “don't care” for the seven-bit case.

3.1.2 Parity

Parity is enabled/disabled with the input PARITY_EN. When parity is enabled, the ODD_N_EVEN input defines the type of parity.

3.1.3 Baud Value Precision

Input BAUD_VAL_FRACTION [2:0] is used to set a fractional part for the baud value. The baud value can be set with a precision of 0.125. Enable Extra Precision must be selected in the Core Configuration to enable this input.

Note: BAUD_VAL_FRACTION [2:0] is only visible in SmartDesign when Enable Extra Precision is selected in the Core Configuration.

3.1.4 Baud Rate

This baud value is a function of the system clock and the desired baud rate. The value should be set according to EQ 1.

$$\text{baud rate} = \frac{\text{clk}}{(\text{baudval} + 1) \times 16}$$

EQ 1

where

clk = the frequency of the system clock in hertz

baud rate = the desired baud rate

and

$$baudval = \frac{clk}{16 \times baudrate} - 1$$

EQ 2

The term baudval must be rounded to the nearest integer and must be greater than or equal to 1 or less than or equal to 8191. For example, a system with a 33 MHz system clock and a desired baud rate of 9,600 must have a baud_value of 214 decimal or D6 hex. So, to get the desired baud rate, you must assign 16#D6 to BAUD_VAL input. More accurate baud rates can be achieved when baud value precision is enabled and desired precision is selected.

Baud Value Precision

If Enable Extra Precision is selected, the input BAUD_VAL_FRACTION [2:0] will become available. The precision is set as shown in Table 3.

Table 3 • **Baud Value Precision Settings**

BAUD_VAL_FRACTION [2:0]	Precision
000	+0.0
001	+0.125
010	+0.25
011	+0.375
100	+0.5
101	+0.625
110	+0.75
111	+0.875

For example, a system with a 24 MHz system clock and a desired baud rate of 230,400 should have a baud_value of 5.51 decimal. Rounding the baud_value to the nearest integer, which is 6 decimal in this case, causes the percentage error to be higher than the allowed error of approx 4.54%. So, to achieve the desired baud rate, the user should assign 5 decimal to BAUD_VAL input, select **Enable Extra Precision** and assign 100 binary to BAUD_VAL_FRACTION to get better precision.

4 Interface Description

Signal descriptions for CoreUART are given in Table 4.

Table 4 • **CoreUART Signals**

Name1	Type	Description
CLK	Input	Main system clock
RESET_N	Input	Active low asynchronous reset
DATA_IN[7:0]	Input	Transmit write data bus
DATA_OUT[7:0]	Output	Receive read data bus
WEN	Input	Active low write enable. This signal indicates that the data presented on the DATA_IN[7:0] bus should be registered by the transmit buffer/FIFO logic. This signal should only be active for a single clock cycle per transaction and should only be active when the TXRDY signal is active.
OEN	Input	Active low read enable. This signal is used to indicate that the data on DATA_OUT[7:0] has been read and will reset the RXRDY bit and any error conditions (OVERFLOW or PARITY_ERR).
CSN	Input	Active low chip select. The CSN signal qualifies both the WEN and OEN signals. For embedded applications, this signal should be tied to logic 0.
BIT8	Input	Control bit for data bit width for both receive and transmit functions. When BIT8 is logic 1, the data width is 8 bits; otherwise, the data width is 7 bits, data defined by DATA_IN[7] is ignored, and DATA_OUT[7] is “don't care.”
PARITY_EN	Input	Control bit to enable parity for both receive and transmit functions. Parity is enabled when the bit is set to logic 1.
ODD_N_EVEN	Input	Control bit to define odd or even parity for both receive and transmit functions. When the PARITY_EN control bit is set, a 1 on this bit indicates odd parity and a 0 indicates even parity.
BAUD_VAL[12:0]	Input	13-bit control bus used to define the baud rate. **Note:** BAUD_VAL = 0 is not supported.
BAUD_VAL_FRACTION [2:0]	Input	3-bit control bus used to define the fractional part of baud value. Only available when **Enable Extra Precision** is selected in the Core Configuration.
TXRDY	Output	Status bit; when set to logic 0, indicates that the transmit data buffer/FIFO is not available for additional transmit data.
RXRDY	Output	Status bit; when set to logic 1, indicates that data is available in the receive data buffer/FIFO to be read by the system logic. The data buffer/FIFO controller must be notified of the receipt by simultaneous activation of the OEN and CSN signals to prevent erroneous overflow conditions.
PARITY_ERR	Output	Status bit; when set to logic 1, indicates a parity error during a receive transaction. This bit is synchronously cleared by simultaneous activation of the OEN and CSN signals.

Table 4 • **CoreUART Signals *(continued)***

Name1	Type	Description
OVERFLOW	Output	Status bit; when set to logic 1, indicates that a receive overflow has occurred. This bit is synchronously cleared by simultaneous activation of the OEN and CSN signals.
RX	Input	Serial receive data
TX	Output	Serial transmit data
FRAMING_ERR	Output	Status bit; when set to logic 1, indicates that a framing error (missing stop bit) has occurred. This bit is synchronously cleared by simultaneous activation of the OEN and CSN signals.

Note:

1. Active low signals are designated with a trailing uppercase N.
2. When *RX_FIFO* is enabled, *PARITY_ERR* is asserted when a parity error occurs, but deasserted before *CoreUART* receives the next byte. It is the user's responsibility to monitor the *PARITY_ERR* signal (for example, treat it as an interrupt signal), as it is non-persistent when *RX_FIFO = 1*. *FRAMING_ERR* should be treated similarly, when *RX_FIFO = 1*.

4.1 Core Parameters

4.1.1 CoreUART Configurable Options

There are a number of configurable options that apply to CoreUART, as shown in Table 5. If a configuration other than the default is required, the user should use the configuration dialog box in CoreConsole to select appropriate values for the configurable options.

Table 5 • **CoreUART Configurable Options**

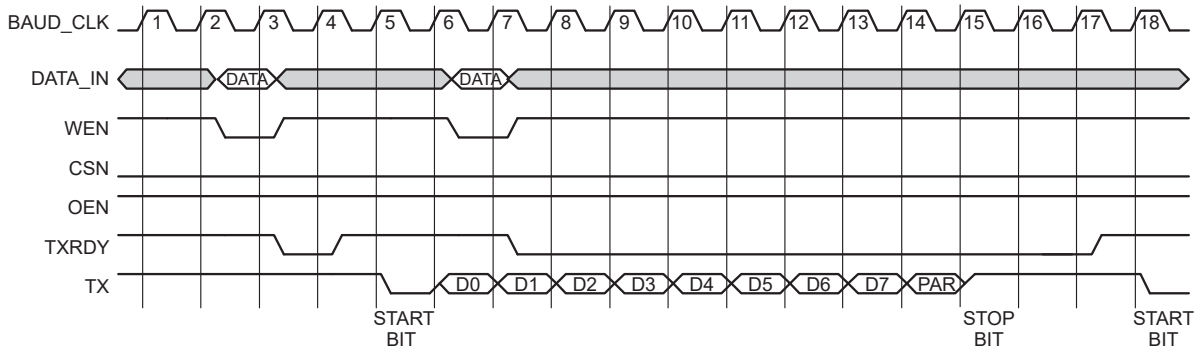
Configurable Options	Default Setting	Description
TX_FIFO	Disabled	Enables or disables transmit FIFO
RX_FIFO	Disabled	Enables or disables receive FIFO
FAMILY	ProASIC3	Selects target family. Must be set to match the supported FPGA family. 8 – SX-A 9 – RTSXS 11 – Axcelerator 12 – RTAX-S 14 – ProASICPLUS 15 – ProASIC3 16 – ProASIC3E 17 – Fusion 18 – SmartFusion 19 – SmartFusion2 20 – IGLOO 21 – IGLOOe 22 – ProASIC3L 23 – IGLOO PLUS 24 – IGLOO2 25 – RTG4 26 – PolarFire 27 – PolarFire SoC
RX_LEGACY_MODE	Disabled	When disabled, the RXRDY signal is synchronized with the FRAMING_ERR output, which occurs after the stop bit. When enabled (Legacy mode), the RXRDY signal is asserted after all data bits have been received, but before the stop bit.
USE_SOFT_FIFO	Disabled	When disabled, the FIFO is implemented using a device-specific hard macro. When enabled, a 16-byte FIFO is implemented in FPGA logic instead. RTAX and RTSX-S devices use this soft FIFO by default.
BAUD_VAL_FRCTN_EN	Disabled	When enabled, input BAUD_VAL_FRACTION[2:0] can be used to set a fractional part for the baud value. The baud value can be set with a precision of 0.125.

5 Timing Diagrams

The UART waveforms can be broken down into a few basic functions: transmit data, receive data, and errors. Figure 4 shows serial transmit signals, and Figure 5, page 14 shows serial receive signals. Figure 6, page 14 and Figure 7, page 15 show the parity and overflow error cycles, respectively. The number of clock cycles required is equal to the clock frequency divided by the baud rate. All waveforms assume that eight bits of data and parity are enabled. All waveforms, except Framing Error, page 15, assume Legacy mode is ENABLED.

5.1 Serial Transmit

Figure 4 • **Serial Transmit**



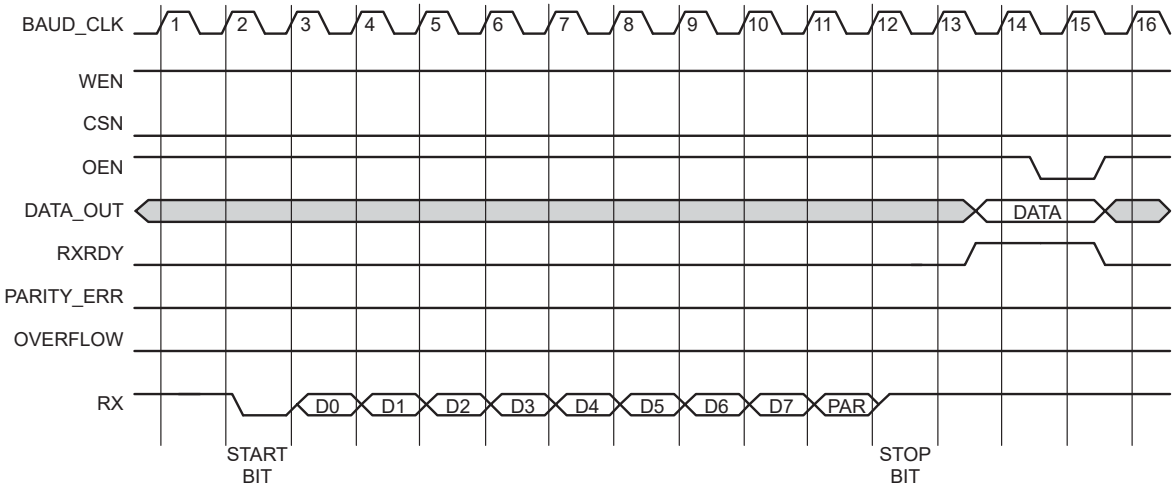
Note:

1. A serial transmit is initiated by writing data into CoreUART. This is accomplished by providing valid data and asserting the WEN and CSN signals. The TXRDY signal will become inactive for one cycle while the data is being transferred from the transmit hold register to the transmit register that begins the serial transfer.
2. It is recommended that after a reset, the data is not written into the transmitter channel register until 11 baud clock cycles. However if the reset is held for 11 baud clock cycles or more, the data can be written into the transmitter channel register straight after the deassertion of the reset.
3. The transmission begins with a START bit, followed by data bits 0 through 6, the optional seventh bit, the optional parity bit, and finally the STOP bit.
4. Because the UART is double-buffered, data can be queued in the transmit hold register (cycle 7). The TXRDY line, when Low, indicates that no more data can be transferred to the UART.

   Once the previous serial transfer is complete, the data in the transmit hold register is passed to the transmit register, and the transfer begins. The TXRDY line is also asserted, indicating that the next data byte can be loaded.

5.2 Serial Receive

Figure 5 • **Serial Receive**

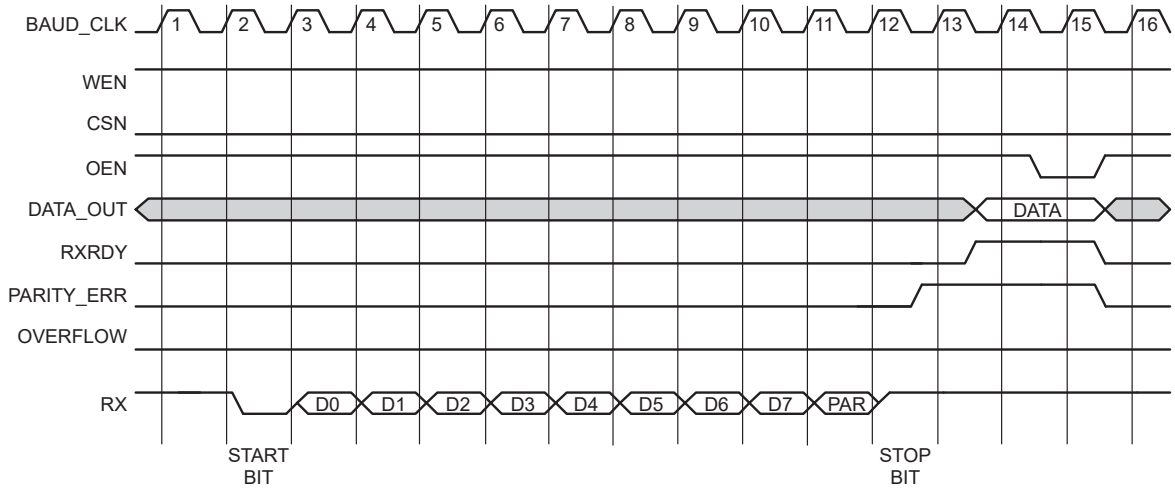


Note:

1. CoreUART continuously monitors the RX line, polling for a START bit. Once the START bit is detected, CoreUART registers the data stream. The optional parity bit is also registered and checked. A START bit is defined as logic 0-bit value on the RX line when the core is idle.
2. The data is then loaded into the receive hold buffer, and the RXRDY signal is asserted. The RXRDY signal will remain asserted until the data is read externally, indicated by the simultaneous assertion of CSN and OEN.

5.3 Parity Error

Figure 6 • **Parity Error**



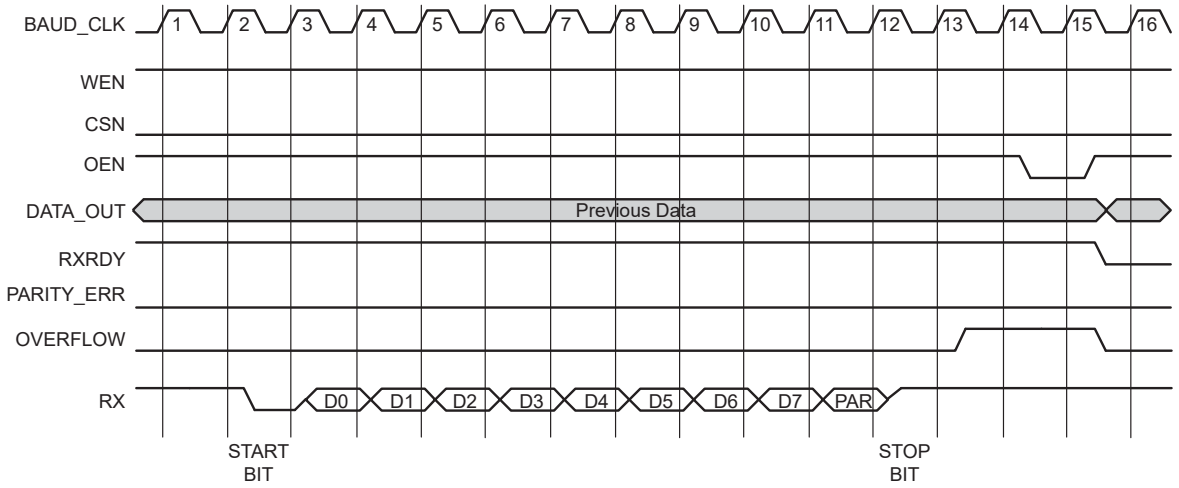
Note:

1. When a parity error occurs, the PARITY_ERR signal is asserted.
2. The error is cleared by the same method used to read the data, simultaneous assertion of CSN and OEN.
3. When RX_FIFO=1, the data comes out in first in first out (FIFO). However, if a parity error occurs, the data which caused the parity error goes to the output (if the read request enabled) until the parity error is de-asserted. This gives a chance to read the data that caused the parity error when it occurs. This data will not be stored in the RX_FIFO because it is invalid. This timing diagram shows the data

that caused the parity error being read. During a parity error, no data will be read from the RX_FIFO so no valid data is lost.

5.4 Overflow Error

Figure 7 • **Overflow Error**

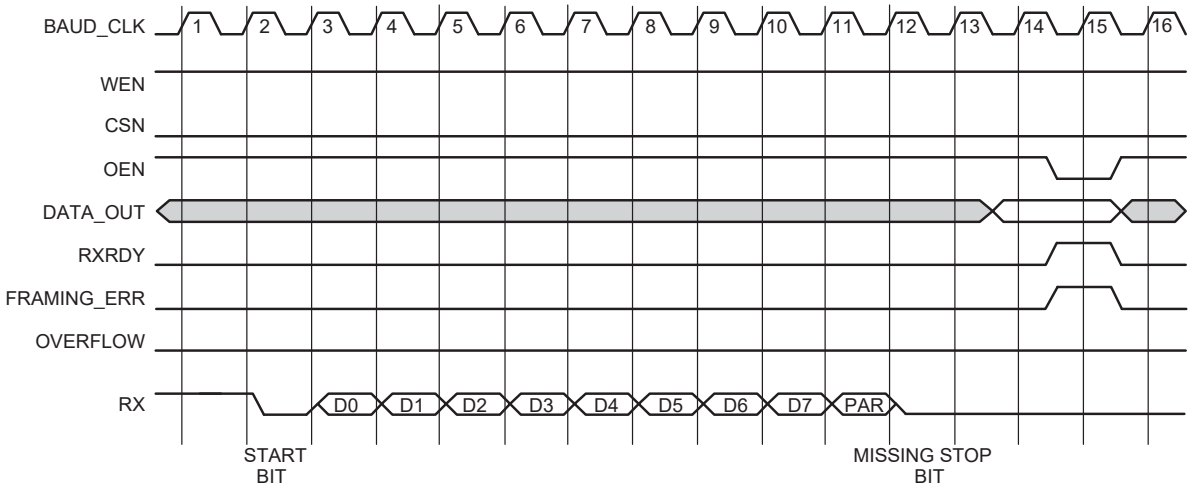


Note:

1. When a data overflow error occurs, the OVERFLOW signal is asserted.
2. The previous data is held and the new data is lost. The error is cleared by the same method used to read the data, simultaneous assertion of CSN and OEN.

5.5 Framing Error

Figure 8 • **Framing Error**

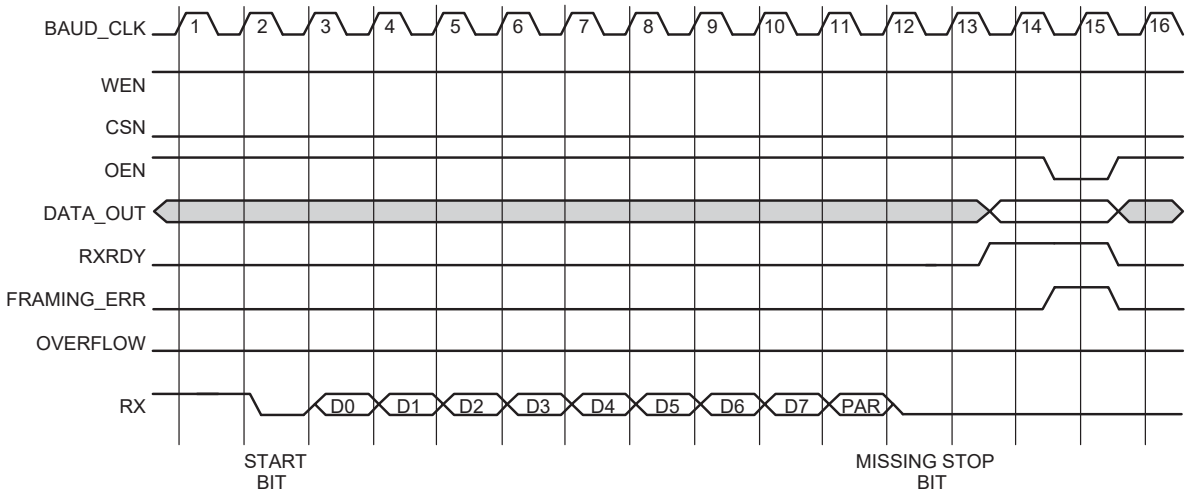


Note:

1. Legacy mode is disabled in this timing diagram.
2. In Normal (non-legacy) mode, RXRDY and FRAMING_ERR are one system clock cycle apart. The FRAMING_ERR signal gets asserted before one system clock cycle of the RXRDY signal assertion. The error is cleared using a read operation, which is simultaneous assertion of OEN and CSN.

5.6 Framing Error in Legacy Mode

Figure 9 • **Framing Error in Legacy Mode**



Note:

1. Legacy mode is enabled in this timing diagram.
2. In Legacy mode, the FRAMING_ERR signal gets asserted after one frame bit period of the RXRDY signal assertion. When the data is available, the RXRDY signal gets asserted and then the check for missing stop bit occurs. The error is cleared using a read operation, which is simultaneous assertion of OEN and CSN.

6 Tool Flows

6.1 License

No license is required for CoreUART.

6.1.1 Obfuscated

Complete RTL code is provided for the core, allowing the core to be instantiated with SmartDesign. Simulation, Synthesis, and Layout can be performed within Libero SoC. The RTL code for the core is obfuscated[1] and some of the testbench source files are not provided; they are precompiled into the compiled simulation library instead.

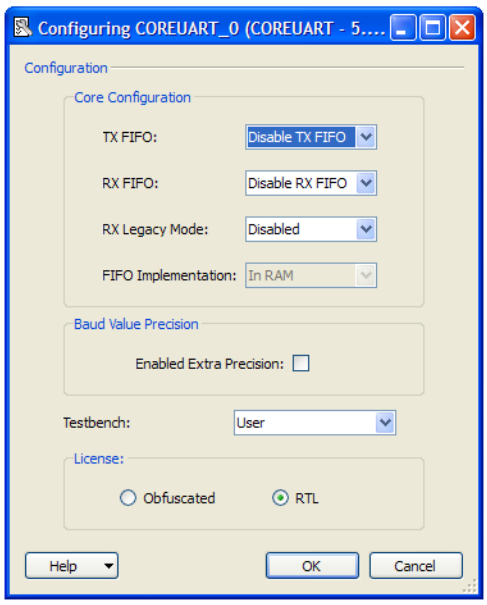
6.1.2 RTL

Complete RTL source code is provided for the core and testbenches.

6.2 SmartDesign

CoreUART is available for download in the SmartDesign IP deployment design environment. The core can be configured using the configuration GUI within SmartDesign, as shown in Figure 10.

Figure 10 • **SmartDesign CoreUART Configuration Window**



To know how to create SmartDesign project using the IP cores, refer to *Libero SoC documents page* and use the latest SmartDesign user guide.

1.Obfuscated means the RTL source files have had formatting and comments removed, and all instance and net names have been replaced with random character sequences.

6.3 Simulation Flows

The user testbench for CoreUART is included in all releases.

To run simulations, select the user testbench flow within SmartDesign and click **Generate Design** under the SmartDesign menu. The user testbench is selected through the Core Testbench Configuration GUI.

When SmartDesign generates the Libero project, it will install the user testbench files.

To run the user testbench, set the design root to the CoreUART instantiation in the Libero Design Hierarchy pane and click the Simulation icon in the Libero Design Flow window. This will invoke ModelSim® and automatically run the simulation.

6.4 Synthesis in Libero

Click the **Synthesis** icon in Libero. The Synthesis window appears, displaying the Synplify® project. Set Synplify to use the Verilog 2001 standard if Verilog is being used. To run Synthesis, select the **Run** icon.

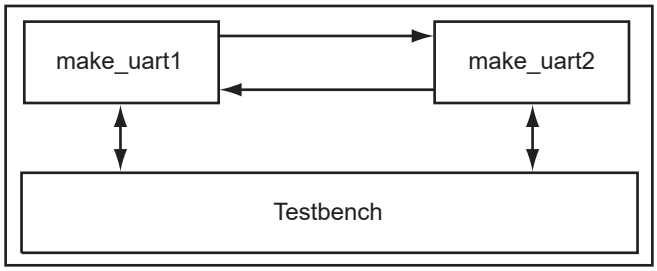
6.5 Place-and-Route in Libero

Click the **Layout** icon in Libero to invoke Designer. CoreUART requires no special place-and-route settings.

7 Testbench Operation and Modification

CoreUART is provided with a user testbench (Figure 11) to demonstrate sample UART operation. The testbenches are available in both Verilog and VHDL and contain two instances of CoreUART connected to each other. The source code is made available with Obfuscated and RTL licenses of the core.

Figure 11 • **Verification Testbench**



The testbench contains the tests listed in Table 6.

Table 6 • **Verification Tests**

No.	Bit	Parity	Parity Setting	Parity Error	Overflow Error	Framing Error	Procedure Call
1	8	Enabled	Even	No	No	No	txrxtest
2	8	Enabled	Odd	No	No	No	txrxtest
3	7	Enabled	Even	No	No	No	txrxtest
4	7	Enabled	Odd	No	No	No	txrxtest
5	8	Disabled	N/A	No	No	No	txrxtest
6	8	Disabled	N/A	No	No	No	txrxtest
7	7	Disabled	N/A	No	No	No	txrxtest
8	7	Disabled	N/A	No	No	No	txrxtest
9	8	Enabled	Even	Yes	No	No	paritytest
10	8	Enabled	Odd	Yes	No	No	paritytest
11	7	Enabled	Even	Yes	No	No	paritytest
12	7	Enabled	Odd	Yes	No	No	paritytest
13	8	Enabled	Odd	No	Yes	No	testoverflow
14	8	Enabled	Odd	No	No	Yes	N/A

The procedure calls txrxtest, paritytest, and testoverflow are defined in the file tbpack.vhd. The top-level testbench, testbench.vhd, utilizes these procedures to perform the corresponding tests listed in Table 6.

Refer to the source directory for source code for the testbench.