UART Core User Guide

UG-CORE-UART-v3.8
February 2023
www.efinixinc.com

Contents

Introduction

The Universal Asynchronous Receiver/Transmitter (UART) core converts serial data to parallel data on characters sent to or received from a peripheral device.

Use the IP Manager to select IP, customize it, and generate files. The UART core has an interactive wizard to help you set parameters. The wizard also has options to create a testbench and/or example design targeting an Efinix® development board.

Features

- Simple, easy-to-use UART with small logic utilization
- Configurable baud rates from 1.2 kbps to 115.2 kbps
- Supports 8, 16, and 32 bit interfaces
- Supports an optional parity bit (even or odd parity for transmit and receive)
- Supports run-time configurable baud rate
- Verilog RTL and simulation testbench
- Includes example designs targeting the Trion® T20 BGA256 Development Board and Titanium Ti60 F225 Development Board

FPGA Support

The UART core supports all Trion® and Titanium FPGAs.

Resource Utilization and Performance

Note: The resources and performance values provided are just guidance and change depending on the device resource utilization, design congestion, and user design.

Titanium Resource Utilization and Performance

FPGA	Logic and Adders	Flip-flops	Memory Blocks	DSP48 Blocks	f_{MAX} (MHz)[1]	Efinity® Version[2]
Ti60 F225 C4	182	208	0	0	380	2022.1

Trion Resource Utilization and Performance

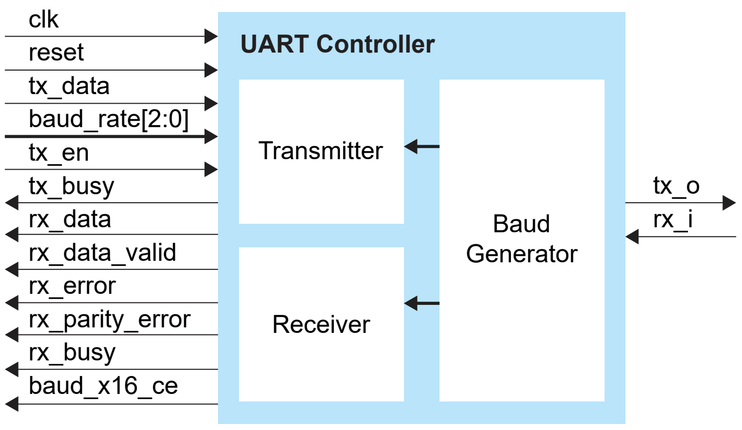
FPGA	Logic Utilization (LUTs)	Registers	Memory Blocks	Multipliers	f_{MAX} (MHz)	Efinity® Version[2]
T20 BGA256 C4	171	198	0	0	123[1]	2022.1

(1) Using default parameter settings.
(2) Using Verilog HDL.

Functional Description

The UART controller consists of a UART transmitter finite state machine (FSM), UART receiver FSM, and a baud rate generator.

Figure 1: UART Controller System Block Diagram



Baud Generator

The baud generator block generates a baud tick at 16 times the baud rate. You configure the baud generator using the `BAUD` parameter.

Transmitter

The transmitter block takes data from `tx_data` and serializes it when `tx_en` is asserted. After power up, the transmitter block sends an ASCII `OK`. You define the number of data bytes for the transmitter using the `BYTE` parameter. If you set the transmitter to more than 1 byte, the transmitter transfers the highest byte first.

Receiver

The receiver block contains an 8-bit receiver buffer register and a shift register.

Figure 2: Receiver State Machine

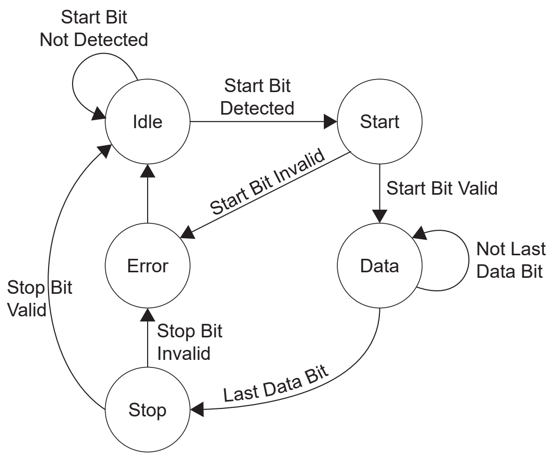


Table 1: Receiver State Machine States

State	Description
Idle	After a reset, or after the stop or error states, the receiver FSM resets to idle. In idle, the receiver waits for `rx_i` to change from high to low. When it detects the start bit, the FSM changes to the start state.
Start	The receiver checks whether the start bit remains low for one bit. If it does, the receiver considers it a valid start bit. Once it detects a valid start bit, the FSM changes to the data state. Otherwise, it changes to the error state.
Data	The receiver waits one for one bit for each data bit to shift into the 8-bit shift register. After the last bit is shifted In, the FSM changes to the stop state.
Stop	The FSM waits for one bit then samples the stop bit. The receiver does not check the stop bit length (1, 1.5, or 2 bits). After it detects a valid stop bit, the FSM changes to the idle state. Otherwise, it changes to the error state.
Error	The FSM sets the `rx_error` bit to indicate the transmission error. The FSM changes to the idle state.

Ports

Table 2: UART Core Ports

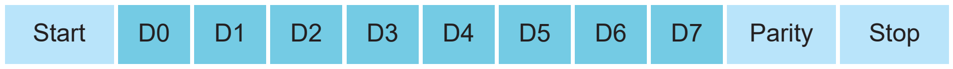
Port	Interface	Direction	Description
clk	System	Input	System clock.
reset	System	Input	Synchronous active high reset.
tx_data	System	Input	Data to transmit from the UART controller to a UART peripheral. Configure the data width using the BYTE parameter.
baud_rate[2:0]	System	Input	Run-time configurable baud rate. `000`: 115200 `001`: 57600 `010`: 38400 `011`: 19200 `100`: 9600 `101`: 4800 `110`: 2400 `111`: 1200 Set the Fixed Baud Rate parameter to **Disable** if you want to configure the baud rate during run-time.
tx_en	System	Input	1: Latches the tx_data and initiates transmission. 0: No operation
tx_busy	System	Output	1: Data transfer in progress 0: Idle
rx_data	System	Output	8-bit receive data from the UART peripheral
rx_data_valid	System	Output	1: rx_data is valid 0: Idle
rx_error	System	Output	A start/stop bit error was detected during the start/stop bit.

Port	Interface	Direction	Description
rx_parity_error	System	Output	1: Parity error 0: No parity error
rx_busy	System	Output	1: Data transfer in progress 0: Idle
baud_x16_ce	System	Output	Enable tick from baud generator with 16 times the baud rate. This sampling signal is useful, monitoring, debugging, and can be used to toggle the receiver operation.
tx_o	Serial	Output	Serial data output to the communication link.
rx_i	Serial	Input	Serial data input from the communication link.

UART Data Format

A transaction begins with a low start bit. The data word follows (LSB to MSB), and a high stop bit ends the transaction.

Figure 3: UART Data Frame



Parity Bits

Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers. Parity provides low-level bit error checking and has two modes: odd and even.

To calculate the parity bit, the 8 bits of the data byte are added, and the evenness of the sum determines whether the bit is set. For example, if the parity is even and is added to 8-bit data `1010 1110`, which has an odd number of 1s (5), the parity bit is set to 1. Conversely, if the parity mode is set to odd, the parity bit would be `0`.

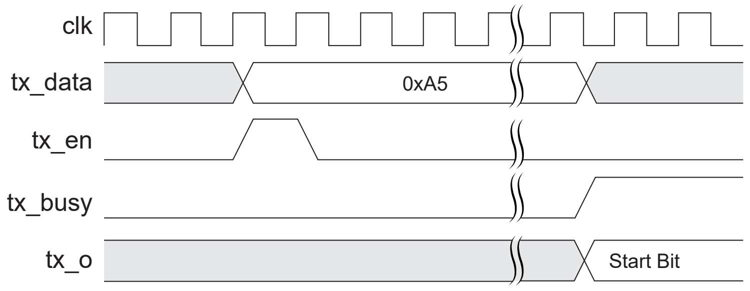
When the parity bit matches the data, the UART's transmission is free of errors.

Transmit Operation

The following figure shows the waveform when writing data into the UART controller. The `tx_data` is sent within one baud tick of when the `tx_en` flag is high.

During data transmission, `tx_busy` goes high. When data transmission completes, `tx_busy` goes low.

Figure 4: UART Transmit Waveform

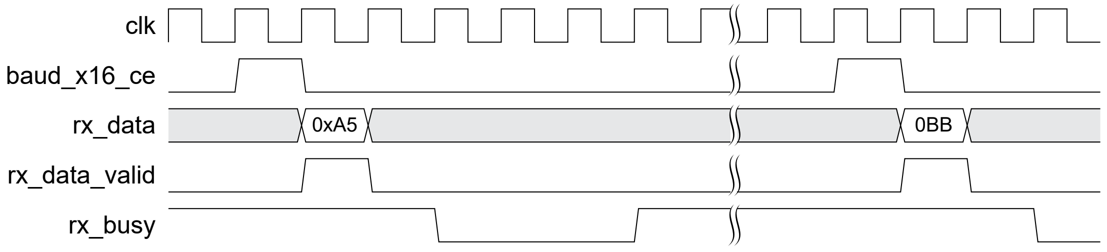


Receive Operation

The following figures show the waveforms when reading data from the UART controller.

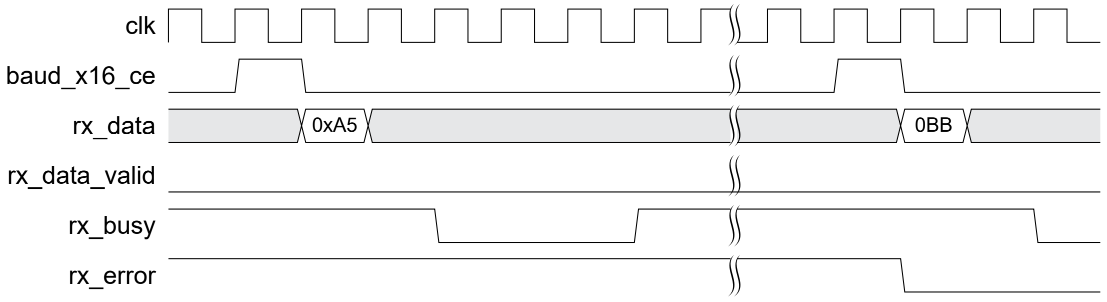
As shown in the following figure, when `baud_x16_ce` goes high, the `rx_data_valid` flag goes high on next clock cycle. When the `rx_data_valid` flag is high, the `rx_data` is valid. The `rx_busy` status is low after receiving the stop bit and it is not affected by the `baud_x16_ce` signal.

Figure 5: Normal Receive Operation Waveform



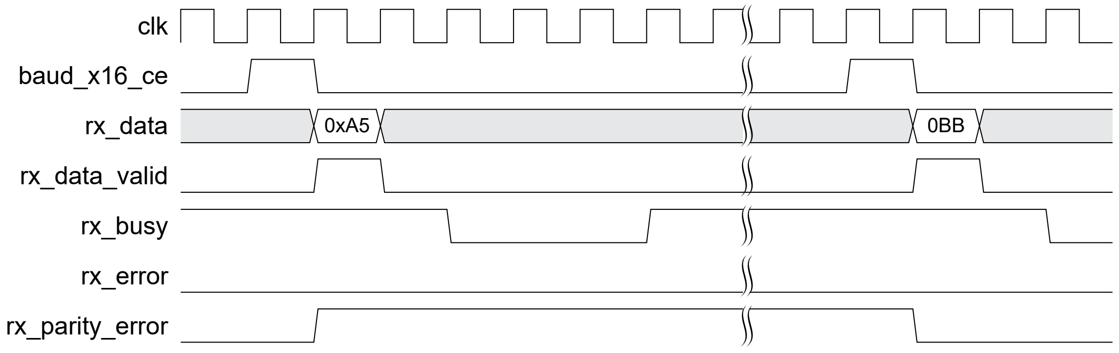
When `rx_error` goes high, there is a transmission frame error. The `rx_data_valid` signal goes low to prevent incorrect data reads. The `rx_error` status goes low on the next `rx_data` byte.

Figure 6: Receive Error Waveform



When there is a parity error, `rx_parity_error` is set to high. The `rx_parity_error` goes low on the next `rx_data` byte.

Figure 7: Parity Error Waveform



IP Manager

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Efinix® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Efinix development board and/or a testbench. This wizard is helpful in situations in which you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core for different projects.

Note: Not all Efinix IP cores include an example design or a testbench.

Generating a Core with the IP Manager

The following steps explain how to customize an IP core with the IP Configuration wizard.

1. Open the IP Catalog.
2. Choose an IP core and click **Next**. The **IP Configuration** wizard opens.
3. Enter the module name in the **Module Name** box.

   Note: You cannot generate the core without a module name.

4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the IP core's user guide or on-line help.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Efinix® development board and/or testbench. For SoCs, you can also optionally generate embedded software example code. These options are turned on by default.
6. (Optional) In the **Summary** tab, review your selections.
7. Click **Generate** to generate the IP core and other selected deliverables.
8. In the **Review configuration generation** dialog box, click **Generate**. The Console in the **Summary** tab shows the generation status.

   Note: You can disable the **Review configuration generation** dialog box by turning off the **Show Confirmation Box** option in the wizard.

9. When generation finishes, the wizard displays the **Generation Success** dialog box. Click **OK** to close the wizard.

The wizard adds the IP to your project and displays it under **IP** in the Project pane.

Generated Files

The IP Manager generates these files and directories:

- **<module name>_define.vh**—Contains the customized parameters.
- **<module name>_tmpl.v**—Verilog HDL instantiation template.
- **<module name>_tmpl.vhd**—VHDL instantiation template.
- **<module name>.v**—IP source code.
- **settings.json**—Configuration file.
- **<kit name>_devkit**—Has generated RTL, example design, and Efinity® project targeting a specific development board.
- **Testbench**—Contains generated RTL and testbench files.

Note: Refer to the IP Manager chapter of the Efinity® Software User Guide for more information about the Efinity® IP Manager.

Customizing the UART

The core has parameters so you can customize its function. You set the parameters in the General tab of the core's IP Configuration window.

Table 3: UART Core Parameters

Parameter	Options	Description
Data Bytes for transmitter	1 - 4	Define the number of data bytes for the transmitter. Example: If BYTE is 2, the tx_data is 16 bits or 2 bytes. Default: 4
System Clock Frequency (Hz)	25000000, 50000000, 100000000	Define the system clock frequency in Hz. Default: 50000000
Baud Rate	1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200	Define the baud rate in Hz. Default: 115200
Fixed Baud Rate	Disable, Enable	Enable: Enable the fixed baud rate. Disable: Use run-time configurable baud rate. Default: Enable
Enable Optional Parity Bit	Disable, Enable	Enable the optional parity bit. Applicable to TX only. Default: Disable
Parity Mode	Even, Odd	Parity mode. Default: Even
Enable Bootup Check	Disable, Enable	Enable the bootup check with an ASCII OK message. Default: Disable

UART Example Design

You can choose to generate the example design when generating the core in the IP Manager Configuration window. Compile the example design project and download the **.hex** or **.bit** file to your board.

Important: Efinix tested the example design generated with the default parameter options only.

The example design targets the Trion® T20 BGA256 Development Board and Titanium Ti60 F225 Development Board. The design implements a UART controller in the FPGA, which allows you to control the LEDs on the board by sending commands from a terminal program on your computer to the board. Additionally, the design allows you to change the baud rate during run-time by using the DIP switches on the development boards.

Note: The **Fixed Baud Rate** parameter is set to **Disable** by default. Set the parameter to **Disable** if you want to configure the baud rate during run-time.

The design has these blocks:

- *Command State*—Decodes and encodes the read/write command, address, and data to and from the UART controller.
- *User Register*—Stores the register mapping.
- *LED Control*—Latches the last 4 bits of written data and displays the lower byte on the LEDs in binary.

Figure 8: UART Core Example Design

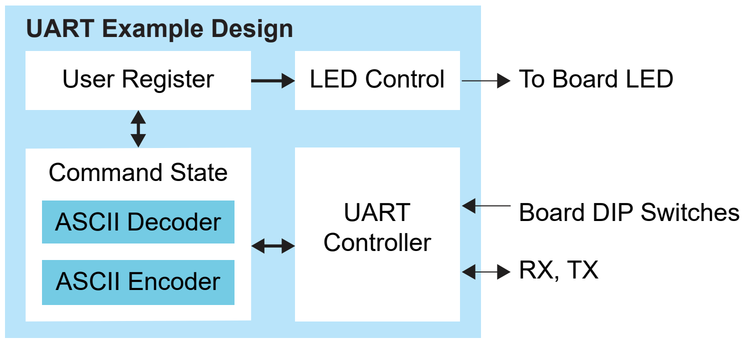


Table 4: Trion® Example Design Implementation

FPGA	LUTs	Registers	Memory Blocks	Multipliers	f_{MAX} (MHz)[3]	Efinity® Version[4]
T20 BGA256 C4	492	565	0	0	92	2022.1

Table 5: Titanium Example Design Implementation

FPGA	Logic and Adders	Flip-flops	Memory Blocks	DSP48 Blocks	f_{MAX} (MHz)[3]	Efinity® Version[4]
Ti60 F225 C4	503	566	0	0	282	2022.1

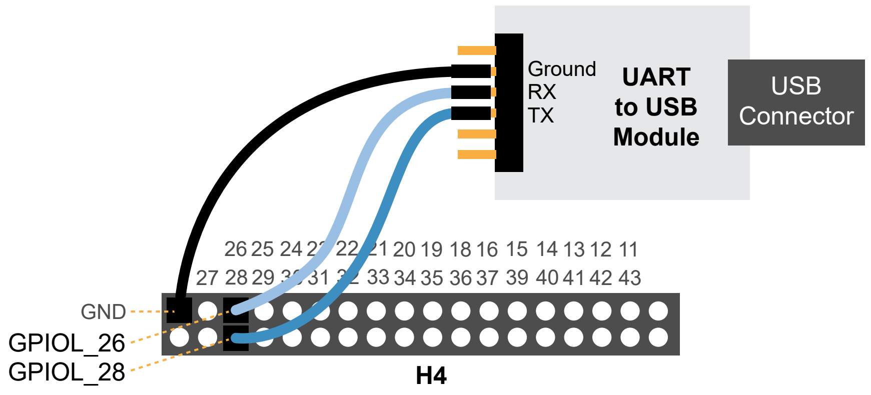
[3] Using default parameter settings.
[4] Using Verilog HDL.

Set Up a USB-to-UART Module

The Trion® T20 BGA256 Development Board does not have a USB-to-UART converter, therefore, you need to use a separate USB-to-UART converter module. A number of modules are available from various vendors; any USB-to-UART module should work.

Note: This section is not applicable to Titanium Ti60 F225 Development Board.

Figure 9: Connect the UART Module to I/O Header H4



1. Connect the UART's RX, TX, and ground pins to the Trion® T20 BGA256 Development Board using:
   - *RX*—GPIOL_28, which is labeled as 28 on header H4
   - *TX*—GPIOL_26, which is labeled as 26 on header H4
   - *Ground*—Ground, connect to one of the GND pins on header H4
2. Plug the UART module into a USB port on your computer. The driver should install automatically if needed.

Finding the COM Port (Windows)

1. Type Device Manager in the Windows search box.
2. Expand **Ports (COM & LPT)** to find out which COM port Windows assigned to the UART module; it is listed as USB Serial Port (COM*n*) where *n* is the assigned port number. Note the COM number.

Finding the COM Port (Linux)

In a terminal, type the command:

```
dmesg | grep ttyUSB
```

The terminal displays a series of messages about the attached devices.

```
usb <number>: <adapter> now attached to ttyUSB<number>
```

There are many USB-to-UART converter modules on the market. Some use an FTDI chip which displays a message similar to:

```
usb 3-3: FTDI USB Serial Device converter now attached to ttyUSB0
```

However, the Trion® T20 BGA256 Development Board also has an FTDI chip and gives the same message. So if you have both the UART module and the board attached at the same time, you may receive three messages similar to:

```
usb 3-3: FTDI USB Serial Device converter now attached to ttyUSB0
usb 3-2: FTDI USB Serial Device converter now attached to ttyUSB1
usb 3-2: FTDI USB Serial Device converter now attached to ttyUSB2
```

In this case the second 2 lines (marked by `usb 3-2`) are the development board and the first line (`usb 3-3`) is the UART module.

Install the Tera Term Software

You need a terminal program to communicate with the T20 development board. This document explains how to use the free open-source Tera Term software for Windows with the example design.

1. Download the Tera Term software from ttssh2.osdn.jp/index.html.en.
2. Run the executable to launch the installer.
3. Follow the wizard to complete installation.

Set Up the Tera Term Software

1. Run the Tera Term software.
2. In the Tera Term: New Connection dialog box, choose **Serial**.
3. Specify these serial port settings:

Option	Setting
Speed	With default parameter settings: 115200 With Fixed Baud Rate disabled: Set according to the board's DIP switch settings.
Data	8 bit
Parity	none
Stop bits	1 bit
Flow control	none

4. Click **OK**.
5. Choose **Setup** > **Terminal**.
6. Under **New-line**, set **Receive** and **Transmit** to **LF**.
7. Turn on **Local echo**.
8. Click **OK**.

Setting the Baud Rate

You can set the baud rate using DIP switches on the development boards. The DIP switch value must match with the speed value set in the Tera Term software. You need to press pushbutton SW5 (`rst_n`) after each time you change the baud rate.

Table 6: Baud Rate and DIP Switch Settings for Trion® T20 BGA256 Development Board

Baud Rate	DIP Switch Settings		
	SW3.1 baud_rate[2]	**SW3.2** baud_rate[1]	**SW3.3** baud_rate[0]
115200	0	0	0
57600	0	0	1
38400	0	1	0
19200	0	1	1
9600	1	0	0
4800	1	0	1
2400	1	1	0
1200	1	1	1

Table 7: Baud Rate and DIP Switch Settings for Titanium Ti60 F225 Development Board

The `baud_rate[2]` is connected to GPIOR_N_13 and fixed to 0 in the design because there are only two DIP switches available on the Titanium Ti60 F225 Development Board. As a result, you can only select 4 baud rates in this example.

Baud Rate	DIP Switch Settings	
	SW2.1 (baud_rate[1])	**SW2.2** (baud_rate[0])
115200	0	0
57600	0	1
38400	1	0
19200	1	1

Terminal Command Format

The UART core processes commands you enter in the terminal. Enter commands in the following formats:

- *Writes*—<*address*> <*operator*> <*data*>
- *Reads*—<*address*> <*operator*>

where:

- <*address*> is the address to read/write
- <*operator*> is ! for write and @ for read
- <*data*> is the ASCII character to send

For example, a write command is `0002!12EF` and a read command is `0002@`.

Control the LEDs

With the example design downloaded into the board and the Tera Term software set up, you are ready to control the LEDs by entering commands in the terminal. The command is of the format:

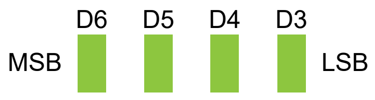
- Writes—<*address*> <*operator*> <*LED data*>
- Reads—<*address*> <*operator*>

where: <*operator*> is ! for write and @ for read

After you issue a write command, the LEDs display the last data written in hex.

Trion® T20 BGA256 Development Board

Figure 10: Displaying the data on the LEDs



Enter commands in the following sequence:

Table 8: Command Sequence

Command Sequence	Action	Address (hex)	Operator	LED Data (hex)	Terminal Returns	LEDs Display
1	Write data 0x5555 to address 0x0000	0x0000	!	0x5555	-	0101
2	Write data 0xaaaa to address 0x0001	0x0001	!	0xaaaa	-	1010
3	Write data 0x6666 to address 0x0002	0x0002	!	0x6666	-	0110
4	Read previously written data from address 0x0000	0x0000	@	-	0x5555	0101
5	Read previously written data from address 0x0001	0x0001	@	-	0xaaaa	1010
6	Read previously written data from address 0x0002	0x0002	@	-	0x6666	0110

The example design supports address range `0x0000 - 0x000F`. You can modify the **user_register.v** file to increase the address range.

The following table correlates the UART ports, FPGA pin assignments, and board labels.

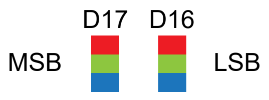
Table 9: Trion® T20 BGA256 Development Board Pin Assignments

UART Design Port	FPGA Pin Assignment	Label
led[0]	GPIOR_104	D3
led[1]	GPIOR_105	D4
led[2]	GPIOR_117	D5
led[3]	GPIOR_118	D6
rst_n	GPIOL_04	SW5
baud_rate[0]	GPIOR_130	SW3.3
baud_rate[1]	GPIOR_129	SW3.2
baud_rate[2]	GPIOR_128	SW3.1
rx_i	GPIOL_28	H4 - 28 [5]
tx_o	GPIOL_26	H4 - 26 [5]

[5] For more information, refer to the Trion® T20 BGA256 Development Board schematics

Titanium Ti60 F225 Development Board

Figure 11: Displaying the data on the LEDs



Enter commands in the following sequence:

Table 10: Command Sequence

Command Sequence	Action	Address (hex)	Operator	LED Data (hex)	Terminal Returns	LED Display
1	Write data 0x4949 to address 0x0000	0x0000	!	0x4949	–	D16 blue D17 red, green, blue
2	Write data 0x1212 to address 0x0001	0x0001	!	0x1212	-	D16 green D17 red, green
3	Write data 0x2424 to address 0x0002	0x0002	!	0x2424	-	D16 red D17 red, green
4	Read previously written data from address 0x0000	0x0000	@	-	0x4949	D16 blue D17 red, green, blue
5	Read previously written data from address 0x0001	0x0001	@	-	0x1212	D16 green D17 red, green
6	Read previously written data from address 0x0002	0x0002	@	-	0x2424	D16 red D17 red, green

The example design supports address range `0x0000` - `0x000F`. You can modify the **user_register.v** file to increase the address range.

The following table correlates the UART ports, FPGA pin assignments, and board labels.

Table 11: Titanium Ti60 F225 Development Board Pin Assignments

UART Design Port	FPGA Pin Assignment	Label
led[0]	GPIOR_P_07	D16
led[1]	GPIOR_P_08	D16
led[2]	GPIOR_P_09	D16
led[3]	GPIOR_N_07	D17
rst_n	GPIOR_P_06	SW5
baud_rate[0]	GPIOL_N_13_CBSEL1	SW2.2
baud_rate[1]	GPIOL_P_13_CBSEL0	SW2.1
baud_rate[2]	GPIOR_N_13	U51
rx_i	GPIOL_01	[6]
tx_o	GPIOL_02	[6]

UART Testbench

You can choose to generate the testbench when generating the core in the IP Manager Configuration window.

Note: You must include all **.v** files generated in the **/testbench** directory in your simulation.

Efinix provides a simulation script for you to run the testbench quickly using the Modelsim software. To run the Modelsim testbench script, run `vsim -do modelsim.do` in a terminal application. You must have Modelsim installed in your computer to use this script.

The testbench performs read and write tests. Each test case indicates pass or fail for the register read/write tests.

Before running the simulation, uncomment line 19 (`//'include "uart_defines.v"`) in the **uart_demo_top.v** file. After running the simulation, the test prints the following message indicating the pass/fail results:

```
# PASSED: Address 00000000 , DATA 5555
# PASSED: Address 00000001 , DATA aaaa
# PASSED: Address 00000002 , DATA 5555
# PASSED: Address 00000003 , DATA aaaa
# PASSED: Address 00000004 , DATA 5555
# PASSED: Address 00000005 , DATA aaaa
# PASSED: Address 00000006 , DATA 5555
# PASSED: Address 00000007 , DATA aaaa
# PASSED: Address 00000008 , DATA 5555
# PASSED: Address 00000009 , DATA aaaa
# PASSED: Address 0000000a , DATA 5555
# PASSED: Address 0000000b , DATA aaaa
# PASSED: Address 0000000c , DATA 5555
# PASSED: Address 0000000d , DATA aaaa
# PASSED: Address 0000000e , DATA 5555
# PASSED: Address 0000000f , DATA 2aaa
```

[6] The Titanium Ti60 F225 Development Board includes on-board UART module.

Revision History

Table 12: Revision History

Date	Version	Description
February 2023	3.8	Added note about the resource and performance values in the resource and utilization table are for guidance only.
January 2023	3.7	Updated baud_x16_ce port description and receiver operation waveforms. (DOC-1105)
October 2022	3.6	Corrected UART module connection setup. (DOC-924) Added support for run-time configurable baud rate and added Fixed baud rate parameter. (DOC-934)
August 2022	3.5	Added support for 9600 baud rate.
March 2022	3.4	Corrected example design block diagram, and LEDs ports, commands and assignments. (DOC-758)
February 2022	3.3	Corrected supported configurable baud rates in Features section.
January 2022	3.2	Updated resource utilization table. (DOC-700)
October 2021	3.1	Added note to state that the f_{MAX} in Resource Utilization and Performance, and Example Design Implementation tables were based on default parameter settings. Updated design example target board to production Titanium Ti60 F225 Development Board and updated Resource Utilization and Performance, and Example Design Implementation tables. (DOC-553)
June 2021	3.0	Added note about including all **.v** generated in testbench folder is required for simulation. Updated resource utilization and performance table. Updated example design output and implementation table. Added support for Titanium FPGAs and example design for Titanium Ti60 F225 Development Board. Updated for Efinity v2021.1.
December 2020	2.0	Updated user guide for Efinix® IP Manager which includes added IP Manager topics, updated parameters, and user guide structure.
March 2020	1.1	Updated the transmit operation waveform; hold tx_data until tx_busy goes high. Corrected minor typos.
October 2019	1.0	Initial release.